FAKULTÄT FÜR !NFORMATIK

M.Sc. Arbeit

Simplification of Herbrand Sequents

ausgeführt am

Institut für Computersprachen
Arbeitsgruppe Theoretische Informatik und Logik
der Technischen Universität Wien

unter der Anleitung von

Univ.Prof. Dr.phil. Alexander Leitsch

durch

Tsvetan Chavdarov Dunchev

Wien, 9.September 2009 .... Tsvetan Dunchev .... Alexander Leitsch

FAKULTÄT FÜR !NFORMATIK

Master Thesis

Simplification of Herbrand Sequents

carried out at the

Institute of Computer Languages
Theory and Logic Group
of the Vienna University of Technology

under the instruction of

Univ.Prof. Dr.phil. Alexander Leitsch

by

Tsvetan Chavdarov Dunchev

Vienna, September 9, 2009 ....

Tsvetan Dunchev Alexander Leitsch

Simplification of Herbrand Sequents

Tsvetan Chavdarov Dunchev

Contents

1 Kurzfassung

Eines der wichtigsten Resultate der mathematischen Logik ist der Satz von Herbrand, welcher besagt dass ein skolemisiertes Sequent $S = A_1, \ldots, A_n \vdash C_1, \ldots, C_m$ gültig ist genau dann wenn es ein Herbrand Sequent S' zu S gibt welches aus Instanzen der A_i and C_j besteht (nach Entfernung der Quantoren) und aussagenlogisch gültig ist. Herbrand Sequente welche aus mathematischen Beweisen extrahiert werden sind ein wichtiges Werkzeug um essentielle mathematische Argumente aus einem formalen LK-Beweis zu gewinnen. Im CERES-System (cut-elimination by resolution) gibt es einen Algorithmus zur Extraktion von Herbrand Sequenten. Das extrahierte Sequent ist allerdings meist sehr redundant und daher schwer interpretierbar. Die Hauptaufgabe dieser Arbeit ist die Entwicklung und Implementation von Algorithmen zur Vereinfachung von Herbrand Sequenten. Vereinfachungen werden dabei auf dem Term-Level und auf dem Formel-Level angewendet. Die Vereinfachungen beruhen auf Termersetzungssystemen welche vom Benutzer des Systems spezifiziert werden. Die entwickelten und getesteten Algorithmen verbessern die Funktionalität von CERES deutlich und ermöglichen damit eine bessere interaktive Beweisanalyse von Beweisen nach der Schnittelimination.

2 Abstract

One of the most important results in mathematical logic is Herbrand's theorem, which says that a skolemized sequent $S = A_1, \ldots, A_n \vdash C_1, \ldots, C_m$ is valid if and only if there exists a Herbrand sequent for S (i.e. a sequent consisting of instances of the A_i and C_j which is propositionally valid). Herbrand sequents which are extracted from an LK-proof are a very useful tool for summarizing and analyzing the essential mathematical information of the proof. Only a few algorithms of Herbrand sequent extraction are known. They differ in some restrictions on the end-sequent (e.g. prenex form) and in the form of admitted proofs (cut-free, or atomic cuts admitted). In the CERES system (cut-elimination by resolution) there exists an algorithm for Herbrand sequent extraction. But the extracted Herbrand sequent is not always the minimal one. It can be minimized in terms of formula occurrences, and minimization of the term complexity of formulas occurrences when a set of rewriting rules (for simplifying equations) is provided. The main topic of this master thesis is to find, investigate and implement an algorithm for the simplification of already extracted Herbrand sequents within the CERES system (Cut-Elimination by Resolution). Simplifications are performed both on the level of formulas and on the level of terms. For term simplification a set of rewriting rules is used which can be extracted from a background theory specified by the user. The simplification of Herbrand sequents is important to the mathematical interpretations of the Herbrand sequents (obtained from proofs after cut-elimination) by humans and increases the quality of interactive proof analysis.

3 Dedication

I dedicate this diploma thesis to my parents.

4 Acknowledgements

I am very thankful to my supervisor Prof. Alexander Leitsch for accepting me in Vienna University of Technology for my second year in EMCL program. I also would like to thank him for the great opportunity to participate in his CERES project as well as for providing me a project work and diploma thesis. His support made my stay in Vienna much easier.

I would like to express my gratitude to the PhD. students of Prof.Leitsch - Bruno Woltzenlogel Paleo, Daniel Weller and Tomer Libal, who helped me a lot with their advices, experience and explanations during my study, my project work and my diploma thesis. The regular project and office hour meetings with them and Prof. Leitsch were very helpful and gave me a motivation to do my project work with a pleasure discovering the area of automated theorem proving which was new for me.

Many ideas in the theoretical part of this diploma thesis I owe to Bruno Woltzenlogel Paleo. Daniel Weller was supervising me during the implementation of this thesis and its integration in CERES.

I also would like to express my thankfulness to Prof. Grabriella Dodero, Prof. Diego Calvanese and Prof. Enrico Franconi from the Free University of Bozen-Bolzano who supported me during my first year in the EMCL program. The work together with Prof. Calvanese, Prof. Davide Martinenghi and my colleague Timofey Asyrkin on the INFOMIX project was very interesting and broadened my knowledge in the are of Databases.

I would like to thank the administration of Province of Bozen-Bolzano, South Tirol, which supported me with a local-ordinary scholarship during my first year in EMCL program.

Finally, I would like to thank all my friends and colleagues for the nice time spent together in Bozen-Bolzano and in Vienna.

5 Introduction

This diploma thesis describes an algorithm for simplification of Herbrand sequents extracted from LK-proofs. Simplifications are performed on the term- and formula level. The simplified sequent should have two properties:

- All terms in the sequent should be in normal form with respect to a given confluent and terminating term-rewriting system.
- The number of the formula occurrences in the simplified sequent should be minimal and the sequent should be still valid.

The basic definitions needed for the first item are presented in the next chapter. Informally, the normal form of the terms and formulas is achieved by introducing a binary ordering relation between terms. This relation is extended also to formulas. The idea behind the term rewriting system should be thought of as an orientation of the background theory equations.

Regarding the second item, reduction of the number of formula occurrences (defined in next chapter) is possible because the extracted Herbrand sequent may contain some irrelevant information for the mathematical mathematical analysis formulas (for example, tautologies) which does not play a role in the validity of the sequent with respect to the background theory. This goal is achieved by using an automated theorem prover (ATP). As an input to the ATP we give the negation of the formula representing the Herbrand sequent. Since the Herbrand sequent is valid, its negation will be unsatisfiable. Transforming the negated Herbrand sequent to a (unsatisfiable) set of clauses allows us to get a refutation of this set. Analyzing the clauses appearing in the refutation allows us to remove the irrelevant formula occurrences from the Herbrand sequent. As ATP we use Otter[2] and its successor Prover9. Here we should mention that the first idea to reduce the number of formula occurrences of the Herbrand sequent was to use a SMT-solver (Sat Modulo Theory). This approach faced some difficulties with the implementation of the interface between CERES and the application of the SMT-solver. Since the background theory was encoded into the term-rewriting system, a better approach which uses the already implemented interface between CERES and Otter was suggested [11].

Finally two experiments are presented which illustrate the simplification of Herbrand sequent. In both cases simplified sequent indeed corresponds to the theoretical analysis.

6 Definitions and notations

6.1 Basic notations

Definition 6.1 (The language)**.** Our language will consist of the following symbols:

1. Constants:
   (a) Individual constants: k_i for $i \in \mathbb{N}$.
   (b) Function constants with i argument-places f^i_j for $i, j \in \mathbb{N}$.
   (c) Predicate constants with i argument-places R^i_j for $i, j \in \mathbb{N}$.
2. Variables:
   (a) Free variables: a_i for $i \in \mathbb{N}$.
   (b) Bound variables x_i for $i \in \mathbb{N}$.
3. Logical symbols: $\neg$, $\vee$ and $\forall$.
4. Auxiliary symbols: '(',')','[',']' and ','.

- Remarks
  1. For convenience we might sometimes omit superscripts and subscripts of functions and predicates, or denote them by a single quote instead of natural numbers.

An expression is any finite sequence of symbols from the language defined above. The next definition is about terms and is given inductively. All inductive definitions will implicitly mean that the objects, which are defined, are only those given by the definition.

Definition 6.2 (Terms and semi-terms)**.** Semi-terms are defined inductively as follows:

1. Every individual constant is a semi-term.
2. Bound and free variable are semi-terms.
3. If f^i is a function constant with i argument-places and t_1,..,t_i are semi-terms, then $f^i(t_1, .., t_i)$ is a semi-term.

Semi-terms which do not contain bound variables are called terms.

Definition 6.3 (Formulas and semi-formulas)**.** If R^i is a predicate constant with i argument-places and $t_1, .., t_i$ are terms, then $R^i(t_1, .., t_i)$ is an atomic formula. Formulas and their outermost logical symbols are defined as follows:

1. Every atomic formula is a formula.
2. If A and B are formulas, then $\neg A$ and $A \vee B$ are formulas with $\neg$ and $\vee$ as their outermost logical symbol.
3. If $A(a)$ is a formula with a free variable 'a' being not necessarily fully indicated in A, then $\forall x A(x)$ is a formula with x a bound variable replacing each occurrence of 'a' in A. The outermost logical symbol is $\forall$.

Semi-formulas differ from formulas in containing semi-terms, which are not bound by a quantifier.

- Remarks
  1. A formula or a term without free variables will be called 'closed'. A closed formula is also called a sentence.
  2. $A(x)$ in the above definition is called the scope of the formula $\forall x A(x)$.
  3. For convenience we might sometimes omit parentheses while having $\neg$ and $\forall$ take precedence over $\vee$.

Replacement on positions play a central role in proof transformations. We first introduce the concept of positions for terms.

Definition 6.4 (Positions)**.** Positions within semi-terms are defined inductively:

- If t is a variable or a constant symbol then 0 is a position in t and $t.0 = t$.
- Let $t = f(t_1, ..., t_n)$ then 0 is a position in t and $t.0 = t$. Let μ : $(0, k_1, ..., k_l)$ be a position in a t_j (for $1 \leq j \leq n$) and $t_j.\mu = s$, then $v : (0, j, k_1, ..., k_l)$ is a position in t and $t.v = s$.

A sub-semi-term s of t is a semi-term $t.v = s$ for some position v in t. Positions will be denoted by [,], i.e. $t[r]_v$ denotes the term t after replacing $t.v$ with r.

- Remarks

  1. Sub-formulas are defined in a similar way to sub-terms. However, they are defined up to replacing previously bound variables.
  2. We will use $P(a)$ to represent a term, formula, sequence of formulas or a whole proof where the variable or term a is fully indicated. $P[a]_\lambda$, where λ can be a single position or a set of positions, will represent the case where a is indicated only at position(s) λ.

Example 6.5 (Sub-semi-formula)**.** *The following are sub-semi-formulas of the formula* $\forall x A(x) \vee B$*:* $\forall x A(x), A(t), A(x)$*, etc.*

Definition 6.6 (Substitutions)**.** A substitution is a mapping σ from the set of free and bound variables to the set of semi-terms such that $\sigma(v) \neq v$ for only a finite number of variables.

Definition 6.7 (Logical complexity of formulas)**.** If F is a formula then the complexity comp(F) is the number of occurrences of logical symbols in F. Later in the thesis we will identify this definition with the definition of grades of formulas.

Definition 6.8 (prenex form)**.** We say that a formula F is in *prenex* form if it is of the form $Q_1x_1 \ldots Q_nx_n(F')$, where F' is a quantifier-free formula and $Q_i \in \{\exists, \forall\}$, for $0 \leq i \leq n$.

Theorem 6.9 (prenex form)**.** For each first-order formula F there exists an equivalent formula F' which is in prenex form.

Definition 6.10 (strong and weak quantifiers)**.** Let $B = (Qx)B'$ be a sub-formula of A. We classify Q as strong or weak according to the following cases:

- If $Q = \forall$ and B is a positive sub-formula of A, then Q is strong in A.
- If $Q = \forall$ and B is a negative sub-formula of A, then Q is weak in A.
- If $Q = \exists$ and B is a positive sub-formula of A, then Q is weak in A.
- If $Q = \exists$ and B is a negative sub-formula of A, then Q is strong in A.

Definition 6.11 (formula skolemization)**.** Let F be a first-order formula. Then the *skolemization* $Sk(F)$ of F, is defined inductively as follows:

1. If F does not have strong quantifiers, then $Sk(F) := F$.

2. If F has strong quantifiers, (Qy) is its first strong quantifier, each quantifier occurs at most once in F and F is rectified, then:

   - If (Qy) is not in the scope of weak quantifiers, then $Sk(F) := Sk(F_{-(Qy)}\{y \leftarrow c_y\})$
   - If (Qy) is in the scope of the weak quantifiers $(Qx_1)(Qx_2)\dots(Qx_n)$ appearing in this order, then
     $Sk(F) := Sk(F_{-(Qy)}\{y \leftarrow f_y(x_1, x_2, \dots, x_n)\})$

   where:

1. c_y is a constant symbol not occurring in F and is called a skolem constant.

2. f_y is a function symbol not occurring in F and is called a skolem function.

3. $F_{-(Qy)}$ means the omission of the quantifier Qy from F.

Skolemization is widely used in the area of automated theorem proving. The advantage is that the operation of skolemization preserves satisfiability, but the resulting formula is not necessary equivalent.

6.2 Sequent calculus for classical logic (LK)

LK is a formal proof system for first-order logic [3]. It was introduced in 1934 by Gerhard Gentzen as a tool for studying natural deduction. It turned out to be a very useful calculus for constructing logical derivations. Here we use an extension of Gentzens Sequent Calculus LK, called $LKDe$, which has in addition definition and equality rules as well as the axioms of equality [10]. $LKDe$ is more convenient then LK because it has higher practical value. In $LKDe$ we need not eliminate non-atomic cut formulas. The basic syntactic element in LK as is the *sequent*.

Sequences of formulas are represented by the greek letters: Γ, Δ, Π and Λ with possible superscripts and subscripts.

Definition 6.12 (Sequents)**.** For arbitrary Γ and Δ, $\Gamma \vdash \Delta$ is called a sequent with $\vdash$ called the sequent symbol. Γ and Δ are called the antecedent and the succedent of the sequent. Each formula in Γ and Δ is called a sequent-formula. A sequent will be denoted by the letter 'S' with or without subscripts, i.e. $A \vdash^S B$.

Definition 6.13 (Semantics of sequents)**.** Semantically a sequent

$$A_1, .., A_n \vdash^S B_1, ..., B_m$$

stands for formula $F(S)$:

$$\bigwedge_{i=1}^{n} A_i \rightarrow \bigvee_{j=1}^{m} B_j.$$

In particular, we define M to be the interpretation of S if it is the interpretation of $F(S)$. If $n = 0$ (i.e. the antecedent is empty), we assign $\top$ to $\bigwedge_{i=1}^{n} A_i$. If $m = 0$ (i.e. the succedent is empty), we assign $\bot$ (falsum) to $\bigvee_{j=1}^{m} B_j$. The empty sequent $\vdash$ is represented by $\top \rightarrow \bot$ which is equivalent to $\bot$. S is true in M if $F(S)$ is true in M and S is valid if $F(S)$ is valid.

Definition 6.14 (Atomic sequents)**.** A sequent $A_1, ..., A_n \vdash B_1, ..., B_m$ is called atomic if for all $1 \leq i \leq n$ and $1 \leq j \leq m$, A_i and B_j are atomic.

Definition 6.15 (Prenex Form)**.** . A formula A is in *prenex* form if and only if it is of the form $(Q_1 x_1)(Q_2 x_2) \dots (Q_n x_n)B$, for some $n \geq 0$, for $Q_1, \dots, Q_n \in \{\forall, \exists\}$, and for B quantifier-free. A sequent is in prenex form if and only if all its formulas are in prenex form. An LK-Proof is in prenex form if and only if all its sequents are in prenex form.

Definition 6.16. (sequent skolemization). Let $S = A_1, \dots, A_n \vdash C_1, \dots, C_m$ be a sequent such that $Sk(A1 \wedge \dots \wedge A_n \rightarrow B_1 \vee \dots \vee B_m) = A'_1 \wedge \dots \wedge A'_n \rightarrow B'_1 \vee \dots \vee B'_m$. Then $Sk(S) = A'_1, \dots, A'_n \vdash C'_1, \dots, C'_m$ is the skolemized sequent of S.

Theorem 6.17 (Validity preservation of skolemization)**.** Let S be a sequent. S is a valid sequent if and only if $Sk(S)$ is a valid sequent.

The basic advantage of the skolemization is that it removes the strong quantifiers of a sequent.

Definition 6.18 (closed sequent)**.** A sequent S is *closed* iff all formulas occurring in S are closed.

Definition 6.19 (instance of a sequent)**.** Let $S = A_1, \dots, A_n \vdash C_1, \dots, C_m$ be a sequent without strong quantifiers. Let $A_i^0(C_i^0)$ is $A_i(C_i)$ after omission of the quantifiers. An *instance* of S is the sequent
$S' = A_1^1, \dots, A_{k_1}^1, \dots, A_1^n, \dots, A_{k_n}^n \vdash C_1^1, \dots, C_{l_1}^1, \dots, C_1^m, \dots, C_{l_m}^m$, where $A_1^i, \dots, A_{k_i}^i$ is a sequence of instances of A_i^0 and $C_1^j, \dots, C_{l_j}^j$ is a sequence of instances of C_j^0, for $i \in \{1, n\}$, $j \in \{1, m\}$.

Definition 6.20 (Herbrand sequent)**.** Let S be a closed sequent containing weak quantifiers only and $\mathcal{A}$ be a theory. Let S' be the sequent S after removal of all its quantifiers. Any sequent valid with respect to the theory $\mathcal{A}$ which is an instance of S' is called *Herbrand sequent* of S.

Example: Let $S = P(0), (\forall x)(P(x) \rightarrow P(s(x))) \vdash P(s(s(0)))$. Then the following sequents are Herbrand Sequents of S:

- $P(0), P(0), P(0) \rightarrow P(s(0)), P(s(0)) \rightarrow P(s(s(0))) \vdash P(s(s(0)))$

- $P(0), P(0) \rightarrow P(s(0)), P(s(0)) \rightarrow P(s(s(0))),$

  $P(s(s(0))) \rightarrow P(s(s(s(0)))) \vdash P(s(s(0)))$

- $P(0), P(0) \rightarrow P(s(0)), P(s(0)) \rightarrow P(s(s(0))) \vdash P(s(s(0)))$

The last sequent is the minimal Herbrand sequent.

Definition 6.21 (Axiom set)**.** A (possibly infinite) set $\mathcal{A}$ of sequents is called an axiom set if it is closed under substitution. I.e. for every $S \in \mathcal{A}$ and a substitution σ we have $\sigma(S) \in \mathcal{A}$. If $\mathcal{A}$ consists only of atomic sequents it is called an atomic axioms set.

Definition 6.22 (Standard axiom set)**.** The standard axiom set is the smallest axiom set containing all sequents of the form $A \vdash A$ for arbitrary atomic formulas A.

Definition 6.23 (formula occurrence)**.** Let $S = A_1, \ldots, A_n \vdash C_1, \ldots, C_m$ be a sequent. Then each element of the set of tuples $\{< l, i > | i \in \{1, n\}\} \bigcup \{< r, j > | j \in \{1, m\}\}$ is called a formula occurrence for S, where l and r stand for antecedent and consequent part of the sequent respectively.

Definition 6.24. A formula occurrence $< l, i >$ $(< r, i >)$ corresponds to a formula F iff F is a subformula of A_i (C_i).

Definition 6.25 (Inference)**.** An inference is an expression of the form:

$$\frac{S_1}{S} \qquad \text{or} \qquad \frac{S_1 \qquad S_2}{S}$$

where S_1, S_2 and S are sequents. S_1 and S_2 are called the upper sequents and S is called the lower sequent of this inference.

Definition 6.26 (Standard LK)**.** The standard (multiplicative) sequent calculus LK contains the standard axiom set and the following rules of inference.

1. Structural rules:

   (a) Weakenings:

$$\frac{\Gamma \vdash \Delta}{D, \Gamma \vdash \Delta}\ (w\colon l) \qquad\qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, D}\ (w\colon r)$$

   (b) Contractions:

$$\frac{D, D, \Gamma \vdash \Delta}{D, \Gamma \vdash \Delta}\ (c\colon l) \qquad\qquad \frac{\Gamma \vdash \Delta, D, D}{\Gamma \vdash \Delta, D}\ (c\colon r)$$

   (c) Exchanges:

$$\frac{\Gamma, C, D, \Pi \vdash \Delta}{\Gamma, D, C, \Pi \vdash \Delta}\ (e\colon l) \qquad\qquad \frac{\Gamma \vdash \Delta, C, D, \Lambda}{\Gamma \vdash \Delta, D, C, \Lambda}\ (e\colon r)$$

   These three rules will be called weak inferences while the others will be called strong inferences.

   (d) Cuts:

$$\frac{\Gamma \vdash \Delta, D \qquad D, \Pi \vdash \Lambda}{\Gamma, \Pi \vdash \Delta, \Lambda}\ (\mathrm{cut}\ :\ \mathrm{D})$$

   D is also called the cut formula of the inference.

2. Logical rules:

   (a) $\neg$-introduction:

$$\frac{\Gamma \vdash \Delta, D}{\neg D, \Gamma \vdash \Delta}\ (\neg\colon l) \qquad\qquad \frac{D, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \neg D}\ (\neg\colon r)$$

   (b) $\vee$-introduction:

$$\frac{C, \Gamma \vdash \Delta \qquad D, \Pi \vdash \Lambda}{(C \vee D), \Gamma, \Pi \vdash \Delta, \Lambda}\ (\vee\colon l)$$

$$\frac{\Gamma \vdash \Delta, C}{\Gamma \vdash \Delta, (C \vee D)}\ (\vee\colon r_1) \qquad\qquad \frac{\Gamma \vdash \Delta, D}{\Gamma \vdash \Delta, (C \vee D)}\ (\vee\colon r_2)$$

   (c) $\wedge$-introduction:

$$\frac{\Gamma \vdash \Delta, C \qquad \Pi \vdash \Lambda, D}{\Gamma, \Pi \vdash \Delta, \Lambda, (C \wedge D)}\ (\wedge\colon r)$$

$$\frac{C \vdash \Gamma, \Delta}{(C \wedge D), \Gamma \vdash \Delta}\ (\wedge\colon l_1) \qquad \frac{C \vdash \Gamma, \Delta}{(D \wedge C), \Gamma \vdash \Delta}\ (\wedge\colon l_2)$$

(d) $\rightarrow$-introduction:

$$\frac{\Gamma \vdash \Delta, A \qquad B, \Pi \vdash \Lambda}{(A \rightarrow B), \Gamma, \Pi \vdash \Delta, \Lambda}\ (\rightarrow\colon l) \qquad \frac{A, \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \rightarrow B}\ (\rightarrow\colon r)$$

The $\neg\colon$, $\wedge\colon$, $\vee\colon$ and $\rightarrow\colon$ -rules are called propositional inferences.

(e) $\forall$-introduction:

$$\frac{F(t), \Gamma \vdash \Delta}{(\forall x F(x)), \Gamma \vdash \Delta}\ (\forall\colon l) \qquad \frac{\Gamma \vdash \Delta, F(a)}{\Gamma \vdash \Delta, (\forall x F(x))}\ (\forall\colon r)$$

(f) $\exists$-introduction:

$$\frac{F(a), \Gamma \vdash \Delta}{(\exists x F(x)), \Gamma \vdash \Delta}\ (\exists\colon l) \qquad \frac{\Gamma \vdash \Delta, F(t)}{\Gamma \vdash \Delta, (\exists x F(x))}\ (\exists\colon r)$$

Where t is an arbitrary term and a does not occur in the lower sequent. The a in $\forall\colon r$ is called the eigenvariable of the inference. The condition that a does not occur in the lower sequent is called the eigenvariable condition of the inference. We will also say that the quantifiers in the lower sequents eliminate the eigenvariable or the term in the upper sequents.

Theorem 6.27 (soundness and completeness of LK)**.** LK is sound and complete with respect to the first-order logic.

6.3 Derivations and proofs

Definition 6.28 (LK-derivations)**.** An LK-derivation is defined as a directed labelled tree where the nodes are labelled by sequents (via the function *seq*) and the edges by inference rules. The label of the root is called the end-sequent. Sequents occurring at the leaves are called initial sequents or axioms. The formal definition is:

- Let v be a node and $seq(v) = S$ for an arbitrary sequent S. Then v is an LK-derivation and v is the root node.

- Let ψ be a derivation tree and v be a leaf in ψ. Let $\xi(S_1, S_2, S)$ be an instance of the binary rule ξ. We extend ψ to ψ' by appending the edges $e_1 : (v, \mu_1)$ and $e_2 : (v, \mu_2)$ to v such that $seq(\mu_1) = S_1$, $seq(\mu_2) = S_2$ and the label of (e_1, e_2) is ξ. ψ' is an LK-derivation with the same root as ψ but with v no longer a leaf. v in ψ is called a ξ-node and μ_1 and μ_2 are leaves.
- The extension by a unary rule is defined analogously.

Definition 6.29 (LK-sub-derivations)**.** Let ψ be an LK-derivation. An LK-sub-derivation of ψ is any sub-tree of ψ.

Definition 6.30 (Formal proof)**.** A proof P in LK is an LK-derivation where the leaves are mapped to initial sequents:

The following terminology and conventions will be used all along this thesis:

- If there exists a proof of S in LK, then S is said to be provable in LK.
- A proof without the cut rule is called cut-free.

Definition 6.31 (Subproofs)**.** Let ψ be a proof. a subproof of ψ is a sub-derivation of ψ which is also a proof.

6.4 $LKDe$

Now we extend LK to $LKDe$ by new rules, namely definition rule and equality rule, and the axiom $\vdash s = s$, for terms s :

(a)Definition rule:

$$\frac{A(t_1, \ldots, t_k), \Gamma \vdash \Delta}{P(t_1, \ldots, t_k), \Gamma \vdash \Delta}\ (def_P : l) \qquad \frac{\Gamma \vdash A(t_1, \ldots, t_k), \Delta}{\Gamma \vdash P(t_1, \ldots, t_k), \Delta}\ (def_P : r)$$

where $t_1, \ldots, t_k$ are terms.
b)Equality rule:

$$\frac{\Gamma \vdash \Delta, s = t \qquad A[s]_\Sigma, \Pi \vdash \Lambda}{A[t]_\Sigma, \Gamma, \Pi \vdash \Delta, \Lambda}\ (=: l_1) \quad \frac{\Gamma \vdash \Delta, t = s \qquad A[s]_\Sigma, \Pi \vdash \Lambda}{A[t]_\Sigma, \Gamma, \Pi \vdash \Delta, \Lambda}\ (=: l_2)$$

for inference on the left and

$$\frac{\Gamma \vdash \Delta, t = s \qquad \Pi \vdash \Lambda, A[s]_\Sigma}{\Gamma, \Pi \vdash \Delta, \Lambda, A[t]_\Sigma} (=: r_1) \quad \frac{\Gamma \vdash \Delta, t = s \qquad \Pi \vdash \Lambda, A[s]_\Sigma}{\Gamma, \Pi \vdash \Delta, \Lambda, A[t]_\Sigma} (=: r_2)$$

for inference on the right, where Σ denotes a set of positions of subterms where replacement of s by t has to be performed. The equality rule is sound with respect to first order logic with equality ($FOL_=$). The definition rule is sound with respect to the axiom $(\forall x)(A(x) \leftrightarrow P(x))$.

6.5 The Mid-sequent theorem

One possible way for obtaining a Herbrand sequent from a cut-free proof is by applying the *Mid-sequent* theorem. The proof of this theorem is constructive algorithm for obtaining the Herbrand sequent from a proof.

Theorem 6.32 (Mid-sequent theorem or Sharpened Hauptsatz)**.** Let φ be a prenex LK-Proof without non-atomic cuts. Then there is an LK-Proof φ' of the same end-sequent such that no quantifier rule occurs above propositional and cut rules[10].

6.6 Term Rewriting Systems

In order to describe the simplification in term level we start with some preliminaries and terminology related with Term Rewriting Systems.

Definition 6.33 (Signature)**.** A *signature* Σ is a set of function symbols, where each $f \in \Sigma$ is associated with non-negative integer n (the arity of f). Elements of Σ with arity zero are called *constants*.

From now on, we assume that a signature Σ is fixed.

We should mention that the term-rewriting is not possible in a variable position. If it was possible, then we could not guarantee termination of the term-rewriting.

Definition 6.34 (substitution)**.** Let V be a set of free variables. Substitution is a function $g : V \to T(\Sigma, V)$, such that $\sigma(x) \neq x$ for finitely many $x \in V$. *Domain* of σ is the set $Dom(\sigma) = \{x \in V \mid \sigma(x) \neq x\}$. *Range* of σ is the set $Range(\sigma) = \bigcup_{x \in Dom(\sigma)} Var(\sigma(x))$.

Definition 6.35 (instance)**.** We say that a term t is an *instance* of a term s iff there exists a substitution σ such that $\sigma(s) = t$.

The next definitions play key-role in the term rewriting theory. We will pay more attention to the identities because the input of our simplification algorithm in term level will expect "oriented" equality. They are used to transform a term into another equivalent term by replacing instances of the left-hand side of the equality with the corresponding instance of the right-hand side and vice versa. Detailed information about Term-rewriting systems can be found in [4].

Definition 6.36 (equation)**.** *Equation* is a pair $(s,t) \in T(\Sigma, V) \times T(\Sigma, V)$ (write $s \approx t$)

Definition 6.37 (Equational theory)**.** The *equational theory* E of a class of structures is the set of universal atomic formulas that hold in all members of the class. For a class of algebras, this is simply the collection of all equations that hold in all members of the class.

Definition 6.38 (Reduction relation)**.** Let E be a set of equalities over a signature Σ. The *reduction relation* $\rightarrow_E \subseteq T(\Sigma, V) \times T(\Sigma, V)$ is defined as follows: $s \rightarrow_E t$ iff there exist $(l,r) \in E$, $p \in Pos(s)$ and σ-substitution, such that $s\mid_p = \sigma(l)$ and $t = s[\sigma(r)]_p$.

Definition 6.39 (closure)**.** Let E be a set of equalities. With $\rightarrow_E^*$ we denote the reflexive transitive closure of $\rightarrow_E$. With $\leftrightarrow_E^*$ we denote the reflexive transitive symmetric closure of $\rightarrow_E$.

The relation $\leftrightarrow_E^*$ is of great interest in term rewriting theory because it is the smallest equivalent relation containing $\rightarrow_E$ and it is closed under substitution. One of the important goals in equational theory is to design decision procedures for $\leftrightarrow_E^*$.

Definition 6.40 (Term rewriting system)**.** Let T be a set of terms over a fixed signature Σ. Then, any binary relation R over T is called *rewriting system* over T. Each element $(l,r) \in R$ is called a rule (usually written as $l \rightarrow r$).

Depending on the structure of the rewrite rules one distinguishes different systems. In our case, we impose some requirements on the term rewriting system such as *confluence* and *termination.* We also want $Var(r) \subseteq Var(l)$, for each rule $l \rightarrow r$.

Definition 6.41 (termination)**.** A term rewriting system T is called *terminating* (or *noetherian*), if there is no infinite sequence of terms $t_0, t_1, \ldots,$ such that $t_i \rightarrow t_{i+1}$, for $i \geq 0$.

Definition 6.42 (confluence)**.** A term rewriting system T is called *confluent* if for all terms l, r_1 and r_2 such that $l \to^* r_1$ and $l \to^* r_2$, then there exists a term s, such that $r_1 \to^* s$ and $r_2 \to^* s$.

Definition 6.43 (Local confluence)**.** A term rewriting system T is called *locally confluent* if for all terms l, r_1 and r_2 such that $l \to r_1$ and $l \to r_2$, then there exists a term s, such that $r_1 \to^* s$ and $r_2 \to^* s$.

Important result in Term Rewriting Systems is the following lemma:

Lemma 6.44 (Newman)**.** If a terminating Term Rewriting System is locally confluent, then it is *confluent*.

We are now ready to introduce an ordering relation over terms and to extend it over formulas.

Definition 6.45 (term ordering)**.** Let T be a set of terms over the signature Σ. The binary relation $>_t \subseteq T \times T$ is defined as follows : $(s, t) \in \geq_t$ iff $s \to^* t$.

The term-rewriting system which we use is obtained after orienting the equations in such a way that the resulting rules set up a confluent and terminating rewriting system. In general, not all of the equations can be oriented. For example, if an equation such as the commutativity axiom, $\forall x \forall y (xy = yx)$ belongs to the background theory, then no orientation preserving termination property is possible. In order to orient such equalities, for example, an ordering relation among terms could be defined. In our algorithm we do not consider such cases. Nevertheless, such unorientable equations are not avoidable because the theorem prover may use them to produce a refutation.

Definition 6.46 ($=_E$)**.** Let E be a set of equalities over the signature Σ which are oriented and the resulting term-rewriting system is confluent and terminating. The binary relation $=_E \subseteq T \times T$ is defined as follows : $(s, t) \in =_t$ iff $s \leftrightarrow^* t$.

Since $=_E$ is and equivalence relation, we can split it to a union of equivalence classes. For each equivalence class we define a binary ordering relation $>_\tau$ among terms in the following way : $s >_\tau t$ if $s \to t$. Since $\to$ is terminating and confluent relation, the relation $>_\tau$ is well founded and hence each equivalence class has a smallest element, called a *normal form*.

Definition 6.47 ($=^f_E$)**.** Let $=^f_E$ is a binary relations such that $P(x_1, \ldots, x_n) =^f_E P(y_1, \ldots, y_n)$ iff $x_i =_E y_i$, for $i = 1, \ldots, n$.

$=_E^f$ is an equivalence relation. Hence, in each equivalence class there exists a smallest element which is a formula with terms in normal form.

Definition 6.48 (sequent ordering)**.** Let $=_E^{seq}$ be a binary relation between quantifier ground sequents and $S = A_1, \ldots, A_n \vdash C_1, \ldots, C_m$, $S' = A'_1, \ldots, A'_n \vdash C'_1, \ldots, C'_m$ are quantifier ground sequents, then we define $S =_E^{seq} S'$ iff $A_i =_E^f A'_i$ and $C_j =_E^f C'_j$, for $i = 1, \ldots, n$, $j = 1, \ldots, m$.

If S is a sequent, then in the equivalence class $[S]_{=_E^{seq}}$ there exists a minimal element S_{min}. We are interested in those Herbrand sequents (valid with respect to the background theory) of S_{min} that consist of minimal number formula occurrences.

7 Algorithms for extracting a Herbrand Sequent from a proof

In CERES [12] we work with $LKDe$-proofs only. The importance of extracting Herbrand sequents from proofs lies on the fact that a Herbrand sequent summarizes the creative content of a proof. Here we present four algorithms for Herbrand sequent extraction. They differ in some restrictions on the end-sequent and the form of admitted proofs. A detailed description and deep analysis of these algorithms can be found in [8]. The algorithms are classified according to two requirements:

- a proof transformation is required
- a prenex form of the end-sequent is required

Proof transformation means that the shape of the proof tree may be modified. For example, some rules can be permuted, added or dropped in a specific way.

7.1 Extraction via Mid-Sequent Reduction

This algorithm requires proof-transformation in such a way that propositional-rules or cut-rules appear below quantifier-rules. This idea was explained in the previous section. A detailed description of the algorithm can be found in [8]. Specific feature is that the algorithm requires prenex form of the end-sequent of the proof. Here we just present a simple example. Consider the following proof:

φ :

$$
\dfrac{\dfrac{\dfrac{P(0)\vdash P(0) \qquad \dfrac{\dfrac{P(s(0))\vdash P(s(0)) \qquad P(s^2(0))\vdash P(s^2(0))}{P(s(0)), P(s(0))\to P(s^2(0))\vdash P(s^2(0))}\to: l}{P(s(0)), \forall x(P(x)\to P(s(x)))\vdash P(s^2(0))}\forall : l}{P(0), P(0)\to P(s(0)), \forall x(P(x)\to P(s(x)))\vdash P(s^2(0))}\to: l}{P(0), \forall x(P(x)\to P(s(x))), \forall x(P(x)\to P(s(x)))\vdash P(s^2(0))}\forall : l}{P(0), \forall x(P(x)\to P(s(x)))\vdash P(s^2(0))}c : l
$$

We do the following proof transformation which permutes the second $\to: l$ rule with the first $\forall : l$ rule.

φ' :

$$\dfrac{\dfrac{\dfrac{P(0) \vdash P(0) \quad \dfrac{P(s(0)) \vdash P(s(0)) \qquad P(s^2(0)) \vdash P(s^2(0))}{P(s(0)), P(s(0)) \to P(s^2(0)) \vdash P(s^2(0))} \to: l}{\dfrac{P(0), P(0) \to P(s(0)), P(s(0)) \to P(s^2(0)) \vdash P(s^2(0))}{P(s(0)), \forall x(P(x) \to P(s(x))) \vdash P(s^2(0))} \forall : l} \to: l}{P(0), \forall x(P(x) \to P(s(x))), \forall x(P(x) \to P(s(x))) \vdash P(s^2(0))} \forall : l}{P(0), \forall x(P(x) \to P(s(x))) \vdash P(s^2(0))} c : l$$

Obviously, in φ' no propositional rule appears below a quantifier rule. The conclusion of the last $\to: l$ inference is the so-called mid-sequent which is the extracted Herbrand sequent, namely:

$$P(0), P(0) \to P(s(0)), P(s(0)) \to P(s^2(0)) \vdash P(s^2(0))$$

7.2 Extraction via Collection of Instances

This algorithm requires a prenex end-sequent. It is much more efficient compared then the Mid-sequent transformation. The Extraction via Collection of instances does not transform the original proof but just analyzes it. The computed Herbrand sequent of both algorithms is the same. The idea is to notice that quantifier-free substitution instances in occurrences of Herbrand sequent are necessarily the auxiliary occurrences of some quantifier-rules in the proof. Therefore we can compute the Herbrand sequent just by analyzing the proof, collecting its appropriate auxiliary occurrences and then constructing a sequent by composing the quantifier-free subsequent with the sequent formed from these collected auxiliary occurrences. In other words, we remove the quantified occurrences of the end-sequent and replace them by substitution instances given by the auxiliary occurrences of the quantifier-rules. The detailed description of the algorithm can be found in [10]. Here we present a simple example. Consider again the proof φ from the previous algorithm:

φ :

$$
\dfrac{P(0) \vdash P(0) \qquad \dfrac{\dfrac{P(s(0)) \vdash P(s(0)) \qquad P(s^2(0)) \vdash P(s^2(0))}{P(s(0)), P(s(0)) \to P(s^2(0)) \vdash P(s^2(0))} \to: l}{P(s(0)), \forall x(P(x) \to P(s(x))) \vdash P(s^2(0))} \forall : l}{\dfrac{\dfrac{P(0), P(0) \to P(s(0)), \forall x(P(x) \to P(s(x))) \vdash P(s^2(0))}{P(0), \forall x(P(x) \to P(s(x))), \forall x(P(x) \to P(s(x))) \vdash P(s^2(0))} \forall : l}{P(0), \forall x(P(x) \to P(s(x))) \vdash P(s^2(0))} c : l} \to: l
$$

We proceed as follows. Consider all quantifier rules. In this example they are two:

$$
\dfrac{P(s(0)), P(s(0)) \to P(s^2(0)) \vdash P(s^2(0))}{P(s(0)), \forall x(P(x) \to P(s(x))) \vdash P(s^2(0))} \forall : l
$$

$$
\dfrac{P(0), P(0) \to P(s(0)), \forall x(P(x) \to P(s(x))) \vdash P(s^2(0))}{P(0), \forall x(P(x) \to P(s(x))), \forall x(P(x) \to P(s(x))) \vdash P(s^2(0))} \forall : l
$$

We build a new sequent $S' = \Gamma' \vdash \Delta'$, where Γ' is a set of all auxiliary antecedent occurrences of the two rules above and Δ' is the set of all auxiliary consequent occurrences of the two rules above(Δ' is empty). Hence, we get:
$S' = P(0) \to P(s(0)), P(s(0)) \to P(s^2(0)) \vdash$

The next step is to get a subsequent S'' of the end-sequent of φ which contains no quantified formula occurrences. Hence, we get:
$S'' = P(0) \vdash P(s^2(0))$

The extracted Herbrand sequent from φ is the composition:
$S' \circ S'' = P(0), P(0) \to P(s(0)), P(s(0)) \to P(s^2(0)) \vdash P(s^2(0))$.

7.3 Extraction via Proof Transformation to Quantifier-free LK_A

This Herbrand sequent extraction algorithm is the most important one for this thesis because it is implemented in the CERES system. Using the interface in CERES, the output of this algorithm serves as input for the simplification algorithm which is described in details in the next section. The algorithm is for non-prenex end-sequent of the proof.

The method of extraction via proof transformation to quantifier-free LK_A-calculus basically consists of transforming the LK-Proof into a quantifier free proof in a modified version of Sequent Calculus called LK_A, which admits sequents containing so-called array-formulas. Then the end-sequent of this LK_A-Proof is transformed back to a sequent without array-formulas. This final sequent is indeed a Herbrand sequent of the end-sequent of the original LK-Proof, and thus the algorithm is sound. Here we give the basic definitions and present an example. A detailed description of the algorithm as well as proofs of its soundness and determinism can be found in [1].

Definition 7.1 (array formula)**.** An array formula is defined by induction:

1. any first order formula is an array formula
2. If $A_1, A_2, \ldots, A_n$ are array formulas, then $\langle A_1, A_2, \ldots, A_n \rangle$ is an array formula
3. If A and B are Array formulas, then $\neg A$, $A \vee B$, $A \wedge B$ and $A \rightarrow B$ are Array formulas.

Definition 7.2 (sequent calculus LK_A)**.** If $A_1, A_2, \ldots, A_n$ and $B_1, B_2, \ldots, B_m$ are Array formulas, then $A_1, A_2, \ldots, A_n \vdash B_1, B_2, \ldots, B_m$ is called an Array sequent. The Sequent Calculus LK_A is the Sequent Calculus LK with addition of the following rules:

$$\frac{A, B, \Delta, \Gamma, \Pi \vdash \Lambda}{\langle A, B \rangle, \Delta, \Gamma, \Pi \vdash \Lambda}\ \langle\rangle : l \qquad\qquad \frac{\Lambda \vdash \Delta, \Gamma, \Pi, A, B}{\Lambda \vdash \Delta, \Gamma, \Pi, \langle A, B \rangle}\ \langle\rangle : r$$

The idea of the algorithm is that it uses two kind of mappings. The first one is Ψ which transforms an LK-proof to a quantifier-free LK_A-proof applying the following steps:

1. From the LK-proof we remove all quantifier rules.
2. Replace $c\colon l$ rules with $\langle\rangle : l$ rules and $c\colon r$ rules with $\langle\rangle : r$ rules.

The second mapping is Φ. It maps an array formula to a sequence of first order formulas in the following way:

1. if A is a First-order logic formula, the $\Phi(A) = A$

2. $\Phi(\langle A, B\rangle) = \Phi(A), \Phi(B)$

3. if $\Phi(A) = A_1, \ldots, A_n$, then $\Phi(\neg A) = \neg A_1, \ldots, \neg A_n$

4. if $\Phi(A) = A_1, \ldots, A_n$ and $\Phi(B) = B_1, \ldots, B_m$, then $\Phi(A \circ B) = A_1 \circ B_1, \ldots, A_1 \circ B_m, \ldots, A_n \circ B_1, \ldots, A_n \circ B_m$, for $\circ \in \{\vee, \wedge, \rightarrow\}$

5. if $\Phi(A) = A_1, \ldots, A_n$, then $\Phi((Qx)A) = (Qx)A_1, \ldots, (Qx)A_n$, for $Q \in \{\forall, \exists\}$

6. if $\Phi(A_1, \ldots, A_n \vdash B_1, \ldots, B_m) = \Phi(A_1), \ldots, \Phi(A_n) \vdash \Phi(B_1), \ldots, \Phi(B_m)$

If φ is an LK-Proof, then the extracted Herbrand sequent is obtained applying the Φ-operation to the end sequent of the proof $\Psi(\varphi)$.

Consider the proof:

φ :

$$\dfrac{\dfrac{\dfrac{P(0) \vdash P(0) \qquad \dfrac{\dfrac{P(s(0)) \vdash P(s(0)) \qquad P(s^2(0)) \vdash P(s^2(0))}{P(s(0)), P(s(0)) \rightarrow P(s^2(0)) \vdash P(s^2(0))}\ \rightarrow: l}{P(s(0)), \forall x(P(x) \rightarrow P(s(x))) \vdash P(s^2(0))}\ \forall : l}{\dfrac{P(0), P(0) \rightarrow P(s(0)), \forall x(P(x) \rightarrow P(s(x))) \vdash P(s^2(0))}{P(0), \forall x(P(x) \rightarrow P(s(x))), \forall x(P(x) \rightarrow P(s(x))) \vdash P(s^2(0))}\ \forall : l}\ \rightarrow: l}{P(0), \forall x(P(x) \rightarrow P(s(x))) \vdash P(s^2(0))}\ c : l}{P(0) \wedge \forall x(P(x) \rightarrow P(s(x))) \vdash P(s^2(0))}\ \wedge : l$$

We apply the transformation Ψ to φ and get a proof without quantifier rules and contractions replaced by $\langle\rangle : l$ rules:

φ' :

$$\dfrac{\dfrac{\dfrac{P(0) \vdash P(0) \qquad \dfrac{P(s(0)) \vdash P(s(0)) \qquad P(s^2(0)) \vdash P(s^2(0))}{P(s(0)), P(s(0)) \rightarrow P(s^2(0)) \vdash P(s^2(0))}\ \rightarrow: l}{P(0), P(0) \rightarrow P(s(0)), P(s(0)) \rightarrow P(s^2(0)) \vdash P(s^2(0))}\ \rightarrow: l}{P(0), \langle P(0) \rightarrow P(s(0)), P(s(0)) \rightarrow P(s^2(0))\rangle \vdash P(s^2(0))}\ <>: l}{P(0) \wedge \langle P(0) \rightarrow P(s(0)), P(s(0)) \rightarrow P(s^2(0))\rangle \vdash P(s^2(0))}\ \wedge : l$$

Now, we apply the Φ transformation to the end-sequent of φ'. The result a Herbrand sequent:

$$\Phi(P(0) \wedge \langle P(0) \to P(s(0)), P(s(0)) \to P(s^2(0))\rangle \vdash P(s^2(0))) =$$

$$= \Phi(P(0)) \wedge \Phi(\langle P(0) \to P(s(0)), P(s(0)) \to P(s^2(0))\rangle) \vdash \Phi(P(s^2(0))) =$$

$$= P(0) \wedge P(0) \to P(s(0)), P(0) \wedge P(s(0)) \to P(s^2(0)) \vdash P(s^2(0))$$

7.4 Extraction via Collection of Sub-Formula Instances

This algorithm generalizes the algorithm of Extraction via Collection of Instances for the case of non-prenex end-sequents. It does not need to transform the proof in order to extract the Herbrand sequent. We notice that the latter collects all the instances of all quantified occurrences of the end-sequent and then it constructs a Herbrand sequent by removing all those quantified occurrences of the end-sequent and inserting all the collected instances. However, in the non-prenex case, one difficulty arises: we need to substitute collected instances not for occurrences in the end-sequent, but for specific sub-formulas in the end-sequent. Here we just mention this algorithm. Detailed description the reader can find in [1] as well as example.

8 Simplification of Herbrand Sequents

Simplification of Herbrand Sequents is needed in order to improve the readability of the sequent as well as to delete the information which is useless for interpreting the mathematical meaning encoded in it. The simplification goes through two different steps. The first one is simplification on term level. The second one is simplification on logical (formula) level.

- In the first step we try to rewrite each term in each atom formula to a term which is in normal form according to a given confluent and terminating system of rewriting rules. As a result we obtain the minimal sequent S_{min}, where S be the sequent to be simplified.

- The simplification in formula level takes S_{min} and tries to remove all formula occurrences which are irrelevant for the validity with respect to the background theory. The following steps are executed:

  1) transforms S_{min} to a formula and negate it. Let the resulting formula be F

  2) remove all implications from F. The resulting formula F' contains $\wedge, \vee$ and $\neg$ as logical symbols only.

  3) F' is transformed to a formula F'' in Negation Normal Form (NNF).

  4) to each atomic formulas in F'' is assigned a unique label (a natural number) and this label is encoded into the name of the corresponding atomic formula.

  5) F'' is transformed to a formula F''' in Conjunctive Normal Form (i.e. F''' is in Clause Form).

  6) F''' is given to the theorem prover (Otter[1] or Prover9) which returns a resolution refutation ρ.

  7) ρ is analyzed and all atomic formulas in F'' which are also in ρ are marked with a special marker (in our case $\star$).

  8) decode the names of all atomic formulas in F'', i.e. remove the label from the predicate name of each atomic formula. Keep the marker of all marked atomic formulas. Call the sequent $F'^{\star}$

  9) for each formula occurrence in $F'^{\star}$ check whether all atomic formulas are NOT marked. If this is the case, remove the whole formula occurrence. Call the sequent $F^{\star}$. This is the simplified Herbrand sequent.

For Otter[1] and Prover9 there is already written an interface in CERES [2] which allows manipulation and visualization of the obtained result with the Prooftool[7].

The next two subsection describe the term and formula simplification in details.

8.1 Simplification on the term level

The simplification of the Herbrand sequent in term level is an algorithm which rewrites the sequent to a minimal one with respect to the ordering $\geq_{seq}$. That means that we firs have to rewrite all terms to a normal form. The rewriting of the terms is done according to a term-rewriting system. In order to guarantee the existence of unique normal form for each term, we assume that the term-rewriting system is confluent and terminating.

The term-rewriting algorithm is described as follows:

INPUT: a Herbrand Sequent S
OUTPUT: a minimal Herbrand Sequent with respect to $\geq_{seq}$
VARIABLES:
occ : a formula occurrence in the sequent
pos : a term position in a term
P : atomic formula
$l \rightarrow r$: a rule
σ : substitution
t : term
TRS : set of rules

Algorithm 8.1: *RewritingTermsInHerbrandSequent*(S)

for each $occ \in S$
  do { **for each** $P \in occ$
    do { **repeat**
      for each $pos \in P$
      do { **for each** $(l \to r) \in TRS$
        do { $t \leftarrow P.pos$
          if $unifiable(t, l)$
          then { $\sigma \leftarrow mgu(t, l)$
            $P.pos \leftarrow \sigma(r)$ }}}
    until *noFurtherReductionIsPossble* }}

return (S)

The implementation of the algorithm in C++ can be seen in the file opHerbrandSequentSimiplication which contains a all operation classes. It is a part of the *prooflib* library of the CERES system.

Theorem 8.1 (correctness)**.** The algorithm for term-rewriting a Herbrand Sequent returns the minimal sequent with respect to the ordering $\geq_{seq}$.

Proof:

Since we are using a terminating and confluent term-rewriting system, then each term has a normal form. This normal form is the minimal unique term with respect to the ordering $\geq_{\tau}$.

Once all terms in all atomic formulas in a formula occurrence are in normal form, then according to the definition of $=^f_E$, the whole formula corresponding to this formula occurrence is the minimal one with respect to $>_f = \{(P(t_1, \ldots, t_n), P(s_1, \ldots, s_n))|\ s_i$ is the normal form of t_i, for $i = 1, \ldots, n, s, t \in T, P \in PS\}$.

Since each formula corresponding to each formula occurrence is the minimal one, then S_{min} is the smallest sequent according to. the ordering $>_{seq} = \{(S, S')|\ S'$ is the result of substitution a formula with the minimal one$\}$. $\Box$

8.2 Simplification on the formula level

Simplifying a sequent in logical (formula) level consists in removing all formula occurrences in a Herbrand sequent which are irrelevant for the validity of the sequent as well as marking these atom formulas that are not essential for the validity of the sequent but which can not be deleted. A formula occurrence can be deleted only in the case that all atom formulas in it are marked by the algorithm. Formally, let $S = A_1, \ldots, A_n \vdash C_1, \ldots, C_m$ be a Herbrand sequent. Our goal is to find a sequent $S^{occ}_{min} = A_{i_1}, \ldots, A_{i_k} \vdash C_{j_1}, \ldots, C_{j_r}$, $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$ and $\{j_1, \ldots, j_r\} \subseteq \{1, \ldots, m\}$, such that S^{occ}_{min} is minimal with respect to the number of formula occurrences and is still a valid sequent. Hence, according to the definition, it is also a Herbrand sequent. The general idea is that we negate the extracted Herbrand sequent S and transform it into a formula $\varphi = A_1 \wedge \cdots \wedge A_n \wedge (\neg C_1) \wedge \cdots \wedge (\neg C_m)$. Then, we transform φ into equivalent formula φ' in Negation Normal Form (NNF). For this purpose we apply the De-Morgan rules to each disjunct in φ pushing the negation as much as possible according to the following rules (we apply them to each conjunct in φ till no further reduction is possible):

$$1)(F_1 \rightarrow F_2) \Rightarrow (\neg F_1 \vee F_2)$$

$$2)\neg(F_1 \wedge F_2) \Rightarrow (\neg F_1 \vee \neg F_2))$$

$$3)\neg(F_1 \vee F_2) \Rightarrow (\neg F_1 \wedge \neg F_2))$$

$$4)\neg\neg F \Rightarrow F$$

The formulas F, F_1 and F_2 are quantifier-free and only conjunction, disjunction and negation signs can appear. The cases for quantifier formula are omitted because the extracted Herbrand sequent contains grounded quantifier-free formulas only.

The last transformation applies the distributivity laws. The obtained formula in clause form, i.e. $\varphi_{CNF} = H_1 \wedge \cdots \wedge H_p$, such that each H_i, $i \in \{1, \ldots, p\}$ is a clause, i.e. $H_i = B_{k_1} \vee \cdots \vee B_{k_q}$, where each B_j is a literal (atom formula or its negation). In fact this is exactly the conjunctive-normal form of φ.

Then, we transform φ_{CNF} to a clause set. Then we give this clause set and the axioms of the background theory to the theorem prover. The result of the theorem prover is a refutation tree. The atom formulas in the Herbrand sequent which occur in formulas which are used in a particular refutation are marked. All non-marked formula can be removed from the Herbrand sequent, in the following two cases:

1)if the corresponding formula occurrence consists of only this atom formula

2)if all atom formulas in a corresponding formula occurrence are not marked. Then the whole formula occurrence can be dropped.

Otherwise we just keep the marked formula. The algorithm in pseudocode for the simplification in formula level looks as follows:

INPUT: a Herbrand Sequent S with terms in normal form

OUTPUT: minimal Herbrand Sequent

VARIABLES:

$S = A_1, \dots, A_n \vdash C_1, \dots, C_m$: a sequent with quantifier-free grounded formulas only

S' : the empty sequent

$\varphi, \varphi' F_1, F_2, F_1', F_2'$: quantifier-free ground formulas

$clauseSet$, $axiomClauseSet$: sets of clauses

$resProof$: a resolution proof

Algorithm 8.2: *SequentToClause*(S)

$\varphi \leftarrow A_1 \wedge \cdots \wedge A_n \wedge (\neg C_1) \wedge \cdots \wedge (\neg C_m)$
for each $(i \in \{1, n\}, j \in \{1, m\})$
$\quad$ **do** $\begin{cases} A_i \leftarrow \textsc{RemoveImplication}(A_i) \\ C_j \leftarrow \textsc{RemoveImplication}(C_j) \end{cases}$

for each $(i \in \{1, n\}, j \in \{1, m\})$
$\quad$ **do** $\begin{cases} A_i \leftarrow \textsc{TransformToNNF}(A_i) \\ C_j \leftarrow \textsc{TransformToNNF}(C_j) \end{cases}$

$\varphi' \leftarrow \textsc{LabelingAtomFormulas}(\varphi)$

$\varphi' \leftarrow \textsc{RenameAtomFormulasToLabels}(\varphi')$

$clauseSet \leftarrow \textsc{makeEqualities}(\varphi, \varphi')$

$\varphi_{CNF} \leftarrow \textsc{TransformToCNF}(\varphi')$

$clauseSet \leftarrow \textsc{union}(clauseSet, \textsc{TransformToClauseSet}(\varphi_{CNF}))$

$resProof \leftarrow \textsc{theoremProver}(clauseSet)$

$axiomClauseSet \leftarrow \textsc{getAxiomClauses}(resProof)$

$\varphi' \leftarrow \textsc{markAtomFormulas}(axiomClauseSet, \varphi')$

$\varphi'' \leftarrow \textsc{un-renameAtomFormulas}(\varphi')$

$S' \leftarrow \textsc{TransformFormulaToSequent}(\varphi'')$

for each $(occ \in S')$
$\quad$ **do** $\begin{cases} \textbf{if } (\textsc{allAtomFormulasInOccurrenceAreMarked}(occ)) \\ \quad \textbf{then } \textsc{delete}(occ, S') \end{cases}$

return (S')

Now we give an explanation about the function which are called in the pseudo-code above. Function *RemoveImplication* applies the De Morgan's rules to transform a formula to a formula containing only conjunction, disjunction and negation.

Function *TransformToNNF* transforms a formula into a negation-normal form applying the rules described above.

Function *LabelingAtomFormulas* sets **unique** labels to all atom formulas. The labels are set linearly with respect to the formula seen as a linear text, not as a binary tree.

The function *RenameAtomFormulasToLabels* renames all atom formulas in a way such that the corresponding label is coded into the new predicate name. The reason for this renaming is because we want to keep an eye on each atom formula in the returned from the theorem prover refutation tree. Since it is quite possible the same atom formula to occur in formulas in different formula occurrences, it is impossible to understand where a formula in the refutation tree comes from if there are many such formulas in the Herbrand sequent. We illustrate this with the following simple example. Assume that this is a Herbrand sequent :

$(A \rightarrow B), A \vdash \neg A, B$, where A and B are atomic formulas. We can see that the formulas A and B occur in different occurrences. We can only notice that there is a redundancy of the formula A in the first and in the second formula occurrences in the antecedent part of the sequent. How can we decide which A could be removed and the sequent to be still a Herbrand sequent? According to the Herbrand sequent definition, only whole formula occurrences can be removed. So, we have to see whether the second formula occurrences A in the antecedent part can be removed. Indeed, it can be removed. To see this, we transform the sequent to a formula $(\neg A \vee B) \wedge A \wedge A \wedge \neg B$. The theorem prover produces a refutation from the clauses $(\neg A \vee B), A, \neg B$. But now the system does not know where the second A comes from. Exactly for this reason we label the atom formulas and encode the labels into their names. Then of course, we add the clauses which say that if a formula has two labels, then the renamed formulas are equivalent.

The function *makeEqualities* returns a set of clauses representing an equalities between all renamed formulas which have the same un-renamed origin formula.

The function *TransformToCNF* transforms a formula to a conjunctive-normal form. This is needed in order to obtain it as a set of clauses. Each conjunct is a clause. The atom formulas with negative polarity in each conjunct go to antecedent part of the sequent and those with positive polarity go in the consequent part of the sequent.

The function *union* performs a union of the two sets of clauses.

The function *theoremProver* calls the theorem prover and returns the refutation tree.

The function *getAxiomClauses* takes the axioms from the resolution refutation tree. In fact we take only axioms because all other nodes of the resolution refutation are subsequents of the axioms.

The function *markAtomFormulas* marks with a marker those atom formulas in the renamed Herbrand sequent which occur in the set of axioms obtained from the resolution refutation.

The function *un-renameAtomFormulas* renames the renamed atom formulas with their original names. It keeps the marker of all marked formulas.

The function *TransformFormulaToSequent* transforms the negation of the formula to a sequent. In this way we obtain the original Herbrand sequent but with marked atom formulas. This allows us to remove those formula occurrences of the Herbrand sequent whose atom formulas are all marked.

allAtomFormulasInOccurrenceAreMarked is a predicate which checks whether all atom formulas in a formula occurrence are marked.

The function *delete* deletes the whole formula occurrences from the sequent.

Consider the following

Example 8.2. *Let* $S = P(0), P(0), P(0) \to P(1) \vdash P(1), P(1) \wedge P(2)$

Construct the negation of S *and transform it to a formula:*

$$F = P(0) \wedge P(0) \wedge (P(0) \to P(1)) \wedge (\neg P(1)) \wedge \neg(P(1) \wedge P(2))$$

Labeling and renaming of the atom formulas:

$$F' = P_1(0) \wedge P_2(0) \wedge (P_3(0) \to P_4(1)) \wedge (\neg P_5(1)) \wedge \neg(P_6(1) \wedge P_7(2))$$

Transform F' *to a NNF and then to CNF:*

$$F'' = P_1(0) \wedge P_2(0) \wedge (\neg P_3(0) \vee P_4(1)) \wedge (\neg P_5(1)) \wedge (\neg P_6(1) \vee \neg P_7(2))$$

Transform F'' *to a clause set:*

$$\mathcal{C} = \{P_1(0) \vdash; P_2(0) \vdash; P_3(0) \vdash P_4(1); P_5(1) \vdash; P_6(1), P_7(2)) \vdash\}$$

Create the set of equivalences (";" is the separator instead of ","):

$$\mathcal{C}' = \{P_1(0) \vdash P_2(0);\ P_2(0) \vdash P_1(0);\ P_2(0) \vdash P_3(0);\ P_3(0) \vdash P_2(0);$$

$$P_4(1) \vdash P_5(1);\ P_5(1) \vdash P_4(1);\ P_5(1) \vdash P_6(1);\ P_6(1) \vdash P_5(1)\}$$

The set $\mathcal{C} \cup \mathcal{C}'$ is given to theorem prover and the following refutation is returned:
$\{P_2(0),\ \neg P_2(0) \vee P_3(0),\ P_3(0),\ \neg P_3(0) \vee P_4(1),$

$P_4(1),\ \neg P_4(1) \vee P_5(1),\ P_5(1), \neg P_5(1),\ \Box\}$

Hence, we mark the following atom formulas in F':

$$F = P_1(0) \wedge P_2^{\bigstar}(0) \wedge (P_3^{\bigstar}(0) \rightarrow P_4^{\bigstar}(1)) \wedge (\neg P_5^{\bigstar}(1)) \wedge \neg(P_6(1) \wedge P_7(2))$$

The next step is to unlabel the formulas, negate it and transform it back to a sequent :

$$S = P(0), P^{\bigstar}(0), P^{\bigstar}(0) \rightarrow P^{\bigstar}(1) \vdash P^{\bigstar}(1), P(1) \wedge P(2)$$

The last step is to delete the formula occurrences which can be deleted :

$$P^{\bigstar}(0), P^{\bigstar}(0) \rightarrow P^{\bigstar}(1) \vdash P^{\bigstar}(1)$$

9 Description of the implementation

The implementation of the simplification algorithm is integrated in the CERES system. It goes through the following stages:

- Term-rewriting simplification part
- Formula simplification part
    1. Labeling linearly all atomic formulas in the sequent
    2. Transforming the negation of the sequent to a set of clauses (including NNF and CNF transformation of the formula occurrences)
    3. Calling the SAT-solver (in this case Otter or Prover9) giving the transformed sequent as an input
    4. Analyzing the output of the SAT-solver and marking the formula occurrences that are used in the resolution refutation
    5. Removing those formula occurrences whose all atomic formulas are marked.

All steps and substeps in the implementation follow the main object oriented architecture of the CERES system, namely the *visitor design pattern* technique[9]. The main advantage of this approach is the ability to add new operations to existing object structures such as ProofTrees, DataNodes, Sequents, Clauses, Functions, Formulas etc., without modifying those structures. Thus, using the visitor design pattern helps conformance with the open/closed principle of the object oriented programming. In essence, the object called "visitor" allows one to add new virtual functions to a set of classes without modifying the classes themselves. Instead, one creates a visitor class that implements all of the appropriate specializations of the virtual function. The visitor takes the instance reference as input, and implements the goal through the programming technique called *double dispatch.*[9]

The term-rewriting implementation part starts with the definition of an object representing a term-rewriting system, namely a set of rules and interface (functions, methods) for manipulating with those rules. The rules are defined as pairs of two terms. The basic operations over terms such as substitution and unification have been provided by the CERES system. The most important classes are **OpDFSTermRewriting** which rewrites a term according to a given rule, **OpOpSimplifyingGroundedFormulaInTermLevel** which rewrites all terms in a formula to terms in normal form according to the term-rewriting system, and **OpSimplifyingSequentInTermLevel** which is applied to the whole Herbrand sequent and includes the operations above.

The formula simplification part is done by several operations over formulas, sequents and set of clauses. The first of them, **transformSequentToFormula** operation, is an operation whichtransforms the negation of the sequent to a formula. The renaming of the atom formulas is done by the operation **OpRenameFormaulasToLabelsInSequent** which calls the formula operation **OpRenameToLabels** in it. It uses the operation which labels the atom formulas **OpLabelAtomsInFormula**.Then, we apply the negation normal form operation **OpTransformFormulaToNNF** which transforms the formula to an equivalent formula in NNF (push the negation as much as possible into the formula). Then, we apply conjunctive normal form operation OpTransformFormulaToCNF to the formula. The result is a formula which is a conjunction of disjunction of literals. The opeartion **OpDisjFormulaToClause** transforms the CNF formula to a set of clauses which are given to the theorem prover together with the set of the clauses obtained by the function **MakeEqualities**. The theorem prover returns a resolution proof which is analyzed by the operation **OpGetInitialsFromRProof**. With the operation **OpMarkResFormaulasInSequent** we mark those atom formulas in the Herbrand sequent which occur in the leafs of the resolution refutation and delete the those formula occurrences that are irrelevant.

10 Experiments

10.1 Simplification of the Herbrand Sequent of the lattice proof

Here we test the simplification algorithm giving as an input the extracted Herbrand Sequent. Detailed description and analysis of the Herbrand Sequent extracted from the Lattice proof can be found in [5] and we are not going into these details. We start directly with the extracted Herbrand sequent , $S = A_1, A_2, A_3, A_4, A_5 \vdash C_1$, where

$A_1 : s_1 \cup (s_1 \cup (s_1 \cap s_2)) = s_1 \cup (s_1 \cap s_2) \rightarrow s_1 \cap (s_1 \cup (s_1 \cap s_2)) = s_1$
$A_2 : s_1 \cap s_1 = s_1 \rightarrow s_1 \cup s_1 = s_1$
$A_3 : (s_1 \cap s_2) \cap s_1 = s_1 \cap s_2 \rightarrow (s_1 \cap s_2) \cup s_1 = s_1,$
$A_4 : (s_1 \cup (s_1 \cap s_2)) \cup s_1 = s_1 \rightarrow (s_1 \cup (s_1 \cap s_2)) \cap s_2 = s_1 \cup (s_1 \cap s_2)$
$A_5 : s_1 \cup (s_1 \cup s_2) = s_2 \cup s_4 \rightarrow s_1 \cap (s_1 \cup s_2) = s_1$
$C_1 : (s_1 \cap s_2) \cup s_1 = s_1 \wedge (s_1 \cup s_2) \cap s_1 = s_1$

Before we proceed we introduce the following

Definition 10.1 ($L1$-lattice)**.** A $L1$-lattice is a set L together with the operations meet ($\cap$) and join ($\cup$) such that both $(L, \cup)$ and $(L, \cap)$ are semilattices and the following property holds:
$(\forall x)(\forall y)x \cap y = x \leftrightarrow x \cup y = y$

Definition 10.2 ($L2$-lattice)**.** A $L1$-lattice is a set L together with the operations meet ($\cap$) and join ($\cup$) such that both $(L, \cup)$ and $(L, \cap)$ are semilattices and the following absorption laws hold:
$(\forall x)(\forall y)(x \cap y) \cup x = x$ and $(\forall x)(\forall y)(x \cup y) \cap x = x$

Our goal is to see whether the simplified Herbrand sequent corresponds to the theoretical observation that formula occurrences A_1, A_2 and A_4 are not essential for the conclusion C_1 [5]. According to the theory all $L1$-lattices are $L2$-lattices. Indeed, A_3 and A_5 together imply the formula $(\forall)(\forall)x \cap y = x \leftrightarrow x \cup y = y$ which is the property for $L1$-lattices. From another hand, C_1 represents the formula $(\forall x)(\forall y)(x \cap y) \cup x = x \wedge (\forall x)(\forall y)(x \cup y) \cap x = x$ which is the property for $L2$-lattices.

In this case the term-rewriting system consists of the rules $x \cap x \rightarrow x$ and $x \cup x \rightarrow x$. Then A_2 immediately can be removed:
$A_1 : s_1 \cup (s_1 \cup (s_1 \cap s_2)) = s_1 \cup (s_1 \cap s_2) \rightarrow s_1 \cap (s_1 \cup (s_1 \cap s_2)) = s_1$

$A_3 : (s_1 \cap s_2) \cap s_1 = s_1 \cap s_2 \to (s_1 \cap s_2) \cup s_1 = s_1,$
$A_4 : (s_1 \cup (s_1 \cap s_2)) \cup s_1 = s_1 \to (s_1 \cup (s_1 \cap s_2)) \cap s_2 = s_1 \cup (s_1 \cap s_2)$
$A_5 : s_1 \cup (s_1 \cup s_2) = s_2 \cup s_4 \to s_1 \cap (s_1 \cup s_2) = s_1$
$C_1 : (s_1 \cap s_2) \cup s_1 = s_1 \wedge (s_1 \cup s_2) \cap s_1 = s_1$

We give the background theory for the semi-lattices:

- $x \cap y = y \cap x$
- $(x \cap y) \cap z = x \cap (y \cap z)$
- $x \cap x = x$
- $x \cup y = y \cup x$
- $(x \cup y) \cup z = x \cup (y \cup z)$
- $x \cup x = x$

The next step is to create a set of clauses from these axioms. We write the equality formulas like an atom formulas with predicate symbol $=$:

axiom-Clause-Set:

$\{\vdash = (x \cap y, y \cap x);$
$\vdash = ((x \cap y) \cap z, x \cap (y \cap z));$
$\vdash = (x \cap x, x);$
$\vdash = (x \cup y, y \cup x);$
$\vdash = ((x \cup y) \cup z, x \cup (y \cup z));$
$\vdash = (x \cup x, x)\}$

Now, we should mark the atom formulas in the extracted Herbrand sequent S. Again, the reason is because we would like to distinguish all formulas in S, even those which are syntactically equivalent. This would eliminate the confusion which of those syntactically identical formulas should be deleted if the formula does not appear in the refutation tree produced by the theorem prover. The marking is linearly labeling of the atomic formulas of the sequent. For this purpose we call the operation **class OpMarkAtomFormulasInHerbrandSequent** which calls for each formula occurrence the operation **class OpLabelAtomsInFormula**. The result is the following Herbrand Sequent $S_{lab} = A'_1, A'_3, A'_4, A'_5 \vdash C'_1$, where :

$A'_1 : s_1 \cup (s_1 \cup (s_1 \cap s_2)) =^1 s_1 \cup (s_1 \cap s_2) \to s_1 \cap (s_1 \cup (s_1 \cap s_2)) =^2 s_1$

$A'_3 : (s_1 \cap s_2) \cap s_1 =^3 s_1 \cap s_2 \rightarrow (s_1 \cap s_2) \cup s_1 =^4 s_1,$
$A'_4 : (s_1 \cup (s_1 \cap s_2)) \cup s_1 =^5 s_1 \rightarrow (s_1 \cup (s_1 \cap s_2)) \cap s_2 =^6 s_1 \cup (s_1 \cap s_2)$
$A'_5 : s_1 \cup (s_1 \cup s_2) =^7 s_1 \cup s_2 \rightarrow s_1 \cap (s_1 \cup s_2) =^8 s_1$
$C'_1 : (s_1 \cap s_2) \cup s_1 =^9 s_1 \wedge (s_1 \cup s_2) \cap s_1 =^{10} s_1$

Since these labels can not be given as an argument to the theorem prover, we should encode them into the atom-formula's name. This is done by the operation **class OpRenameFormaulasToLabelsInSequent**. The result is the sequent $S^{ren}_{lab} = A''_1, A''_3, A''_4, A''_5 \vdash C''_1$, where:

$A''_1 : P_1(s_1 \cup (s_1 \cup (s_1 \cap s_2)), s_1 \cup (s_1 \cap s_2)) \rightarrow P_2(s_1 \cap (s_1 \cup (s_1 \cap s_2)), s_1)$
$A''_3 : P_3((s_1 \cap s_2) \cap s_1, s_1 \cap s_2) \rightarrow P_4((s_1 \cap s_2) \cup s_1, s_1)$
$A''_4 : P_5((s_1 \cup (s_1 \cap s_2)) \cup s_1, s_1) \rightarrow P_6((s_1 \cup (s_1 \cap s_2)) \cap s_2, s_1 \cup (s_1 \cap s_2))$
$A''_5 : P_7(s_1 \cup (s_1 \cup s_2), s_1 \cup s_2) \rightarrow P_8(s_1 \cap (s_1 \cup s_2), s_1)$
$C''_1 : P_9((s_1 \cap s_2) \cup s_1, s_1) \wedge P_{10}((s_1 \cup s_2) \cap s_1, s_1)$

The next steps of the algorithm transform the sequent S^{ren}_{lab} to a sequent with formula occurrences in Negated Normal Form and Conjunctive Normal Form. The operations that we use for this purpose are **class OpTransformFormulaToNNF** and **class OpTransformFormulaToCNF**. This allows us to transform the negated sequent to set of clauses :
$\{P_1(s_1 \cup (s_1 \cup (s_1 \cap s_2)), s_1 \cup (s_1 \cap s_2)) \vdash P_2(s_1 \cap (s_1 \cup (s_1 \cap s_2)), s_1)$;
$P_3((s_1 \cap s_2) \cap s_1, s_1 \cap s_2) \vdash P_4((s_1 \cap s_2) \cup s_1, s_1)$;
$P_5((s_1 \cup (s_1 \cap s_2)) \cup s_1, s_1) \vdash P_6((s_1 \cup (s_1 \cap s_2)) \cap s_2, s_1 \cup (s_1 \cap s_2))$;
$P_7(s_1 \cup (s_1 \cup s_2), s_1 \cup s_2) \vdash P_8(s_1 \cap (s_1 \cup s_2), s_1)$;
$P_9((s_1 \cap s_2) \cup s_1, s_1), P_{10}((s_1 \cup s_2) \cap s_1, s_1) \vdash\}$

calling the function **transformNegatedSequentToClauseSet**.

We notice that the formulas $P_4((s_1 \cap s_2) \cup s_1, s_1)$ and $P_9((s_1 \cap s_2) \cup s_1, s_1)$ are renamed version of the formula $(s_1 \cap s_2) \cup s_1 = s_1$ of the original Herbrand sequent. In general one atom formula can have a lot of renamed versions. In order not to loose this important information we should to the clause set above the following clauses :

$\{P_4((s_1 \cap s_2) \cup s_1, s_1) \vdash (s_1 \cap s_2) \cup s_1 = s_1$;
$(s_1 \cap s_2) \cup s_1 = s_1 \vdash P_4((s_1 \cap s_2) \cup s_1, s_1)$;
$(s_1 \cap s_2) \cup s_1 = s_1 \vdash P_9((s_1 \cap s_2) \cup s_1, s_1)$;
$P_9((s_1 \cap s_2) \cup s_1, s_1) \vdash (s_1 \cap s_2) \cup s_1 = s_1\}$

Analogously, we do this for all formulas which are renamed versions of some atomic formula from the original Herbrand sequent. Since here we have ten atomic formulas, we need twenty clauses in order to encode the equivalences. We should that this way of making equivalences is not complete because we loose some properties of the equalities. Nevertheless, in the worst case the obtained sequent till be a simplification of the original Herbrand sequent but not necessary the smallest one.

This set of clauses is stored in the **eqList** which is a member of the **OpRenameFormaulasToLabelsInSequent** operation (we create this list during the renaming in order to safe computational time).

The last essential information, namely the background theory(axioms) axiom-Clause-Set which we defined above, is also added to the set of clauses. Now, we can give this set of clauses to the theorem prover using the **Write()** member function of the **class ExportOtter** . The prover returns a resolution proof. From this resolution proof we are interested only of the axioms on the leafs of the proof-tree. Using the operation **OpGetInitialsFromR-Proof** we collect these axioms (all of them are atom formulas) and compare which of them corresponds to the renamed atom formulas from the renamed Herbrand sequent. We mark those of the atom formulas of the renamed Herbrnd sequent which occur in the leafs of the resolution proof. This is done by the **OpMarkResFormaulasInSequent** operation. The last step is to unrename the formulas in the renamed Herbrand sequent keeping the marker of those formulas which are marked. The result is the sequent: $S_{min} = A_1''', A_2''', A_3''', A_4''', A_5''' \vdash C_1'''$, where

$$\begin{array}{l}
A_1''' : s_1 \cup (s_1 \cup (s_1 \cap s_2)) = s_1 \cup (s_1 \cap s_2) \to s_1 \cap (s_1 \cup (s_1 \cap s_2)) = s_1 \\
A_3''' : (s_1 \cap s_2) \cap s_1 =^\star s_1 \cap s_2 \to (s_1 \cap s_2) \cup s_1 =^\star s_1, \\
A_4''' : (s_1 \cup (s_1 \cap s_2)) \cup s_1 = s_1 \to (s_1 \cup (s_1 \cap s_2)) \cap s_2 = s_1 \cup (s_1 \cap s_2) \\
A_5''' : s_1 \cup (s_1 \cup s_2) =^\star s_1 \cup s_2 \to s_1 \cap (s_1 \cup s_2) =^\star s_1 \\
C_1''' : (s_1 \cap s_2) \cup s_1 =^\star s_1 \wedge (s_1 \cup s_2) \cap s_1 =^\star s_1
\end{array}$$

One can notice that all atomic formulas in the formula occurrences A_1''' and A_4''' are not marked (with the marker *). Then, according to the definition of Herbrand sequent we can remove this formula occurrences from the extracted Herbrand sequent S. Hence, we obtain as a simplified Herbrand sequent :

$$\begin{array}{l}
(s_1 \cap s_2) \cap s_1 =^\star s_1 \cap s_2 \to (s_1 \cap s_2) \cup s_1 =^\star s_1 \\
s_1 \cup (s_1 \cup s_2) =^\star s_1 \cup s_2 \to s_1 \cap (s_1 \cup s_2) =^\star s_1 \\
\vdash
\end{array}$$

$(s_1 \cap s_2) \cup s_1 =^\bigstar s_1 \wedge (s_1 \cup s_2) \cap s_1 =^\bigstar s_1$

Indeed, the result shows that L1-lattices are L2-lattices [5].

10.2 Simplification of an arithmetic Herbrand Sequent

The next example shows the simplification of a Herbrand sequent in formula level as well as in term level. The signature Σ consists of one constant symbol 0 interpreted as a $0 \in \mathbb{N}$, a unary function symbol s interpreted as a succesor function over $\mathbb{N}$, and two binary function symbols $+$ and $*$ interpreted as a sum and multiplication operation over $\mathbb{N}$ respectively. The sequent is :

$S : A_1, A_2, A_3, A_4 \vdash C_1, C_2$, where

$A_1 : P((s(0) + s(0)) + s(0))$
$A_2 : P(s(0) + s(s(s(0) + s(0))))$
$A_3 : P(s(0) + s(s(s(0) + s(0)))) \rightarrow P(s(s(0)) * s(s(0) + s(0)))$
$A_4 : P(s(s(0) + s(0)) * s(s(0) + 0)) \rightarrow P(s(s(s(0)) * s(s(s(0)))))$
$C_1 : P(s(s(0)) * s(S(0)))$
$C_2 : P(s(s(s(s(0))) * s(s(0))))$

In this case the background theory consists of the following Axiom schema:

- $x + 0 = x$
- $x * 0 = 0$
- $x + s(y) = s(x + y)$
- $x * s(y) = x * y + x$

The sequent can be taught of as $P(3), P(4), P(4) \rightarrow P(5), P(5) \rightarrow P(7) \vdash P(4), P(7)$. The idea is to show that $P(3)$ from the antecedent part and $P(4)$ from the consequent part are irrelevant for the validity of the sequent.

This background theory can be turned into a Term-rewriting system. Furthermore, this term-rewriting system is confluent and terminating. Once we have all terms in normal form, we do not need the background theory anymore. We orient the equation from left to right and add them in the data structure which is a list of **TermRewritingRules**. Then we call the operation **class OpSimplifyingSequentInTermLevel** which apply the rewrite rules till no further reduction is possible. The result is a sequent $S_{trw} : A'_1, A'_2, A'_3, A'_4 \vdash C'_1, C'_2$, where

$A'_1 : P(s^3(0))$
$A'_2 : P(s^5(0))$
$A'_3 : P(s^5(0)) \rightarrow P(s^6(0))$
$A'_4 : P(s^6(0)) \rightarrow P(s^7(0))$
$C'_1 : P(s^4(0))$
$C'_2 : P(s^7(0))$

Now we transform the formulas in Conjunctive-normal forma and label the atom formulas. After that, we rename the sequent in such a way that we encode the labels of the atom formulas into the names of the same atom formulas calling the operation **OpRenameFormaulasToLabelsInSequent**. The result is the sequent:

$P_1(s^3(0))$,
$P_2(s^5(0))$,
$\neg P_3(s^5(0)) \vee P_4(s^6(0))$,
$\neg P_5(s^6(0)) \vee P_6(s^7(0))$
$\vdash$
$P_7(s^4(0))$,
$P_8(s^7(0))$

Now we negate the sequent and transform in to a set of clauses calling the **transformNegatedSequentToClauseSet** operation. The result is the following set of clauses:

$\{\vdash P_1(s^3(0)), \vdash P_2(s^5(0)), P_3(s^5(0)) \vdash P_4(s^6(0)), P_5(s^6(0)) \vdash P_6(s^7(0)),$
$P_7(s^4(0)) \vdash, P_8(s^7(0)) \vdash\}$

Since the formula pairs $P_2(s^5(0))$ and $P_3(s^5(0))$, $P_4(s^6(0))$ and $P_5(s^6(0))$ and $P_6(s^7(0))$ and $P_8(s^7(0))$ are renamed version of the formulas $P(s^5(0))$, $P(s^6(0))$ and $P(s^7(0))$ respectively, we construct the union of the set of clauses above with the following set of clauses representing the equivalence of the formulas in each pair:

$$\{P_2(s^5(0)) \vdash P_3(s^5(0)), P_3(s^5(0)) \vdash P_2(s^5(0)), P_4(s^6(0)) \vdash P_5(s^6(0)), P_5(s^6(0)) \vdash P_4(s^6(0)), P_6(s^7(0)) \vdash P_8(s^7(0)), P_8(s^7(0)) \vdash P_6(s^7(0))\}$$

Now we are ready to give the new set of clauses as an input to the theorem prover. The outcome is a refutation tree from which we collect all the axioms. The marked sequent is:

$$P(s^3(0)), P^\star(s^5(0)), P^\star(s^5(0)) \to P^\star(s^6(0)), P^\star(s^6(0)) \to P^\star(s^7(0))$$
$$\vdash$$
$$P(s^4(0)), P^\star(s^7(0))$$

The unmarked formulas $P(s^3(0))$ and $P(s^4(0))$ correspond to formula occurrence that can be dropped. Hence, the simplified Herbrand sequent is:

$$P^\star(s^5(0)), P^\star(s^5(0)) \to P^\star(s^6(0)), P^\star(s^6(0)) \to P^\star(s^7(0)) \vdash P^\star(s^7(0))$$

11 Conclusion and future work

The main contribution of this thesis is the development and the implementation of an algorithm for simplification of a Herbrand sequent. The algorithm is integrated into the CERES system. It simplifies the extracted Herbrand sequent in both term- and formula level.

As an input the algorithm expects a sequent only. Hence, it does not take into account the LK-proof from which more information for further simplification could be concluded. But analyzing the LK-proof could even help finding the minimal Herbrand sequent directly from the proof without further need to call the algorithm described in this thesis after extracting the Herbrand sequent from the proof. However, this is a possible approach for future investigation of different Herbrand sequent simplification methods.

References

[1] M.Baaz; A.Leitsch, *On skolemization and proof complexity.* Fundamenta Informaticae, (20):353379, 1994.

[2] William McCune; *Automated theorem prover for first-order and equational logic.* Argonne National Laboratory 9700 South Cass Avenue Argonne, IL 60439, http://www.cs.unm.edu/ mccune/otter/

[3] Gerhard Gentzen, *Untersuchungen ueber das logische Schliessen.* Mathematische Zeitschrift 39: 405431

[4] F. Baader, T. Nipkow; *Term Rewriting and All That.* Cambridge University Press, 1998

[5] S. Hetzl, A. Leitsch, D. Weller, B. Woltzenlogel Paleo; *Herbrand Sequent Extraction.* in: S. Autexier et al. (Eds.), Intelligent Computer Mathematics, Vol. 5144 of Lecture Notes in Computer Science, Springer, 2008, pp. 462-477

[6] Gaisi Takeuti, *Proof Theory* . North Holland, 2nd Edition, 1987.

[7] Theory and Logic Group; *A program to review sequent calculi proofs* . Vienna University of Technology, http://www.logic.at/prooftool/

[8] B. Woltzenlogel Paleo, *Herbrand Sequent Extraction* . VDM Verlag Dr. Mueller e.K. (February 7, 2008), ISBN-10: 3836461528

[9] Robert C. Martin, *The Principles, Patterns, and Practices of Agile Software Development* . Prentice Hall

[10] S. Hetzl, A. Leitsch, D. Weller, B. Woltzenlogel Paleo;, *Proof Analysis with HLK, CERES and ProofTool.* Vienna University of Technology

[11] Personal communication from Bruno Woltzenlogel Paleo, 3^{th} of June, 2009

[12] M. Baaz, A. Leitsch; *Cut-elimination and Redundancy-elimination by Resolution.* Journal of Symbolic Computation, 149-176 (2000)