

Proof Analysis with HLK, CERES and ProofTool: Current Status and Future Directions

Stefan Hetzl¹, Alexander Leitsch¹, Daniel Weller¹, Bruno Woltzenlogel Paleo¹

¹Vienna University of Technology

{hetzl, leitsch, weller, bruno}@logic.at

Abstract

CERES, HLK and ProofTool form together a system for the computer-aided analysis of mathematical proofs. This analysis is based on a proof transformation known as cut-elimination, which corresponds to the elimination of lemmas in the corresponding informal proofs. Consequently, the resulting formal proof in atomic-cut normal form corresponds to a direct, i.e. without lemmas, informal mathematical proof of the given theorem.

In this paper, we firstly describe the current status of the whole system from the point of view of its usage. Subsequently, we discuss each component in more detail, briefly explaining the formal calculi (**LK** and **LKDe**) used, the intermediary language *HandyLK*, the *CERES* method of cut-elimination by resolution and the extraction of Herbrand sequents. Three successful cases of application of the system to mathematical proofs are then summarized. And finally we discuss extensions of the system that are currently under development or that are planned for the short-term future.

1 Introduction

Proof synthesis and analysis form the very core of mathematical activity. While the application of automated reasoning techniques to proof synthesis is the focus of large research fields such as *automated theorem proving* and *interactive theorem proving*, the corresponding use of automated reasoning techniques in proof analysis has received considerably less attention. Nevertheless, the importance of proof analysis to mathematics should not be underestimated. Indeed, many mathematical concepts such as the notion of group or the notion of probability were introduced by analyzing existing mathematical arguments [Pol54a, Pol54b].

CERES, HLK and ProofTool are three computer programs that form a system for the computer-aided analysis of mathematical proofs. HLK is responsible for the formalization of mathematical proofs; CERES is responsible for transformations of formal proofs and for the extraction of relevant information from them; and ProofTool is responsible for the visualization of formal proofs.

The most useful proof transformation implemented by CERES for the purpose of proof analysis is cut-elimination. The elimination of cuts from the formalized proofs corresponds to the elimination of lemmas from the original informal proofs. By interpreting the resulting (cut-free or in atomic-cut normal form) proof, a mathematician can obtain a new direct informal proof of the theorem under consideration. The elimination performed by CERES follows the *CERES* method [BL00], which relies on the *resolution calculus* and is essentially different from reductive methods such as the first algorithm for cut-elimination introduced by Gentzen [Gen69]. On the other hand, for the extraction and storage of relevant information from proofs, CERES currently uses the ideas of *characteristic clause set* and *Herbrand sequent*.

In this paper we start with an overview of the system, followed by a discussion of some details of each of the three component programs. Three successful cases of application of the system to real mathematical proofs are then summarized. And finally, we present our plans for the further development and extension of the system.

2 Overview of the Architecture and Workflow

Figure 1 sketches how HLK, CERES and ProofTool can be used by a mathematician to analyze existing mathematical proofs and obtain new ones.

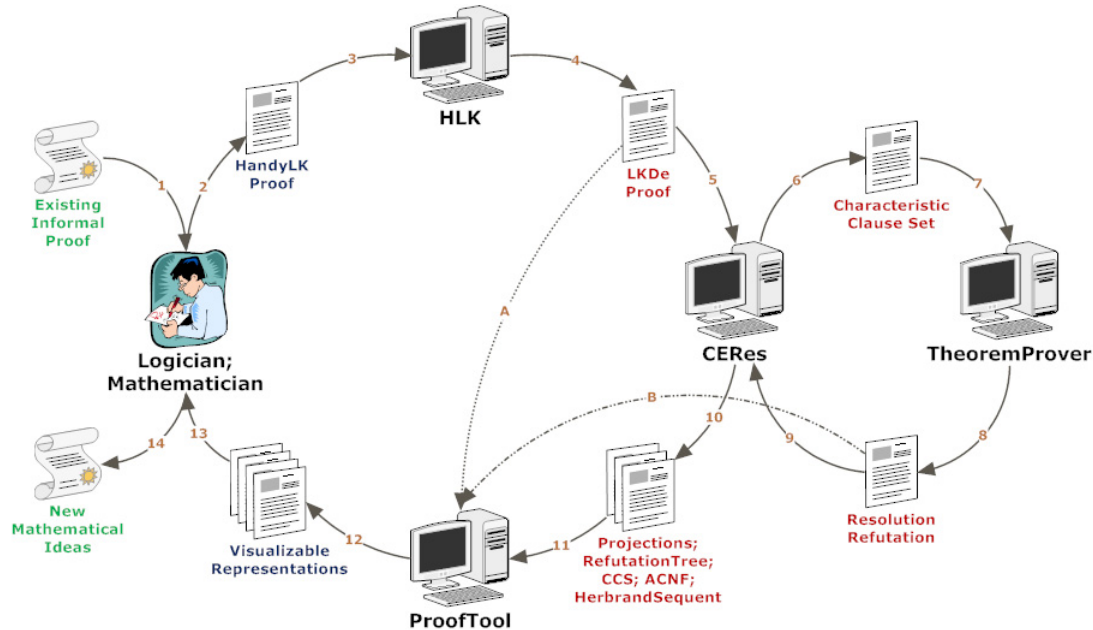


Figure 1: Working with HLK, CERES and ProofTool.

According to the labels in the edges of Figure 1, the following steps are executed within the system and in interaction with the user:

1. The user (usually a mathematician) selects an interesting informal mathematical proof to be transformed and analyzed. Informal mathematical proofs are proofs in natural language, as they usually occur in mathematics.
2. The user writes the selected proof in *HandyLK*, an intermediary language between natural mathematical language and formal calculi.
3. The proof written in *HandyLK* is input to HLK, the compiler of *HandyLK*.
4. HLK generates a formal proof in sequent calculus **LKDe**.
5. The formal proof is input to CERES, which is responsible for cut-elimination and various other proof-transformations that serve as pre- and post-processing, such as proof skolemization, upward shifting of equality rules and Herbrand sequent extraction.
6. CERES extracts from the formal proof a *characteristic clause set*, which contains clauses formed from ancestors of cut-formulas in the formal proof.

7. The characteristic clause set is then input to a *resolution theorem prover*, e.g. Otter¹ or Prover9².
8. The resolution theorem prover outputs a refutation of the characteristic clause set.
9. CERES receives the refutation, which will be used as a skeleton for the transformed proof in atomic-cut normal form (ACNF).
10. CERES outputs the grounded refutation in a tree format and the characteristic clause set. Moreover it extracts projections from the formal proof and plugs them into the refutation in order to generate the ACNF. The projections and the ACNF are also output. A Herbrand sequent that summarizes the creative content of the ACNF is extracted and output as well.
11. All outputs and inputs of CERES can be opened with ProofTool.
12. ProofTool, a graphical user interface, renders all proofs, refutations, projections, sequents and clause sets so that they can be visualized by the user.
13. The information displayed via ProofTool is analyzed by the user.
14. Based on his analysis, the user can formulate new mathematical ideas, e.g. a new informal direct proof corresponding to the ACNF.

2.1 Implementation

HLK³, CERES⁴ and ProofTool⁵ are free and open-source programs written in ANSI-C++ (the C++ Programming Language following the International Standard 14882:1998 approved as an American National Standard), and they are therefore likely to run on all operating systems. However, at this time only Linux and Mac OS X are supported. Nevertheless, to make it easier for the system to be used also by users of unsupported operating systems or even by users of Linux and Mac OS X who are not so familiar with software development, a virtual machine containing the whole system pre-installed is also available for download.

CERES and ProofTool use XML files satisfying the *proofdatabase* DTD⁶ for the storage of formal logical structures (e.g. the original proof, the ACNF, the projections, the Herbrand sequent).

3 HLK for Proof Formalization

In order to analyze mathematical proofs in an automated way, we must firstly write down the proof according to the rules of a formal logical calculus, so that the formalized proofs are then well-defined data-structures, suitable to be manipulated automatically by computers. On the one hand, these data structures should be as simple as possible, to allow theoretical logical investigations as well as the easy implementation of algorithms. On the other hand, they should be as rich and close

¹Otter Website: <http://www-unix.mcs.anl.gov/AR/otter/>

²Prover9 Website: <http://www.cs.unm.edu/mccune/prover9/>

³HLK Website: <http://www.logic.at/hlk>

⁴CERES Website: <http://www.logic.at/ceres>

⁵ProofTool Website: <http://www.logic.at/prooftool>

⁶ProofDatabase DTD: <http://www.logic.at/ceres/xml/4.0/proofdatabase.dtd>

to the natural language of informal mathematical proofs as possible. Therefore, the chosen formal calculus should optimize a trade-off between simplicity and richness of the data structures.

With this in mind, the sequent calculus **LK** was the initial choice. It is a very well-known and well-studied logical calculus, for which proof-theoretical results abound. The specific variant that we use is detailed in Subsection 3.1. To bring it closer to the natural language of informal mathematics, we extended it with definition and equality rules. The resulting calculus is called **LKDe** and its details are discussed in Subsections 3.2 and 3.3.

However, **LKDe** is still uncomfortable to be used directly to write proofs down, because its data structures for proofs are nested and contain many repetitions of formulas in the context of ancestor sequents. To solve this problem *HandyLK* was created to serve as an intermediary language, as explained in detail in Subsection 3.4.

3.1 The Sequent Calculus LK

A *sequent* is a pair of sequences of formulas. The sequence in the left side of the sequent is called its *antecedent*; and the sequence in the right is its *succedent*. A sequent can be interpreted as the statement that at least one of the formulas in the succedent can be proved from the formulas in the antecedent. A sequent calculus *proof* or *derivation* is a tree where the leaf-nodes are *axiom sequents* and the edges are inferences according to the *inference rules* of the sequent calculus. Axiom sequents are arbitrary atomic sequents specified by a *background theory* (e.g. the reflexivity axiom scheme $\vdash t = t$ for all terms t , expressing the reflexivity of equality) or tautological atomic sequents (i.e. $A \vdash A$).

There are many sequent calculi. Our variant uses the following rules:

- **Propositional-rules:**

$$\frac{\Gamma \vdash \Delta, A \quad \Pi \vdash \Lambda, B}{\Gamma, \Pi \vdash \Delta, \Lambda, A \wedge B} \wedge : r \quad \frac{A, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta} \wedge : l1 \quad \frac{A, \Gamma \vdash \Delta}{B \wedge A, \Gamma \vdash \Delta} \wedge : l2$$

$$\frac{A, \Gamma \vdash \Delta \quad B, \Pi \vdash \Lambda}{A \vee B, \Gamma, \Pi \vdash \Delta, \Lambda} \vee : l \quad \frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, A \vee B} \vee : r1 \quad \frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, B \vee A} \vee : r2$$

$$\frac{\Gamma \vdash \Delta, A \quad B, \Pi \vdash \Lambda}{A \rightarrow B, \Gamma, \Pi \vdash \Delta, \Lambda} \rightarrow : l \quad \frac{A, \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \rightarrow B} \rightarrow : r$$

$$\frac{A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \neg A} \neg : r \quad \frac{\Gamma \vdash \Delta, A}{\neg A, \Gamma \vdash \Delta} \neg : l$$

In the rules $\wedge : l1, \wedge : l2, \vee : r1, \vee : r2$, we say that the formula B is a *weak subformula* of A .

- **Quantifier-rules:**

$$\frac{\Gamma \vdash \Delta, A\{x \leftarrow \alpha\}}{\Gamma \vdash \Delta, (\forall x)A} \forall : r \quad \frac{A\{x \leftarrow t\}, \Gamma \vdash \Delta}{(\forall x)A, \Gamma \vdash \Delta} \forall : l$$

$$\frac{\Gamma \vdash \Delta, A\{x \leftarrow t\}}{\Gamma \vdash \Delta, (\exists x)A} \exists : r \quad \frac{A\{x \leftarrow \alpha\}, \Gamma \vdash \Delta}{(\exists x)A, \Gamma \vdash \Delta} \exists : l$$

For the $\forall : r$ and the $\exists : l$ rules, the *eigenvariable condition* must hold: α can occur neither in Γ nor in Δ nor in A . For the $\forall : l$ and the $\exists : r$ rules the term t must not contain a variable that is bound in A .

- **Weakening-rules:**

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A_1, \dots, A_n} w : r \quad \frac{\Gamma \vdash \Delta}{A_1, \dots, A_n, \Gamma \vdash \Delta} w : l$$

where $n > 0$

- **Contraction-rules:**

$$\frac{\Gamma \vdash A_1^{(m_1)}, \dots, A_n^{(m_n)}}{\Gamma \vdash A_1, \dots, A_n} c(m_1, \dots, m_n) : r \quad \frac{A_1^{(m_1)}, \dots, A_n^{(m_n)} \vdash \Delta}{A_1, \dots, A_n \vdash \Delta} c(m_1, \dots, m_n) : l$$

where $m_i > 0$ for $1 \leq i \leq n$ and $A^{(k)}$ denotes k copies of A .

- **Permutation-rules:**

$$\frac{A_1, \dots, A_n \vdash \Delta}{B_1, \dots, B_n \vdash \Delta} \pi(\sigma) : l \quad \frac{\Gamma \vdash A_1, \dots, A_n}{\Gamma \vdash B_1, \dots, B_n} \pi(\sigma) : r$$

where σ is a permutation which is interpreted as a function specifying bottom-up index mapping, i.e. $B_i = A_{\sigma(i)}$.

- **Cut-rule:**

$$\frac{\Gamma \vdash \Delta, A \quad A, \Pi \vdash \Lambda}{\Gamma, \Pi \vdash \Delta, \Lambda} \text{cut}$$

3.2 The Sequent Calculus LKe

Equality is widely used in real mathematical proofs. Carrying the axioms of equality along the proof within the antecedent of the sequents would render large, redundant and unreadable proofs. Therefore, **LK** was extended in [BHL⁺06] to **LKe** with the following rules:

- **Equality-rules:**

$$\frac{\Gamma \vdash \Delta, s = t \quad \Pi \vdash \Lambda, A[s]_{\Xi}}{\Gamma, \Pi \vdash \Delta, \Lambda, A[t]_{\Xi}} = (\Xi) : r1 \quad \frac{\Gamma \vdash \Delta, s = t \quad A[s]_{\Xi}, \Pi \vdash \Lambda}{A[t]_{\Xi}, \Gamma, \Pi \vdash \Delta, \Lambda} = (\Xi) : l1$$

$$\frac{\Gamma \vdash \Delta, t = s \quad \Pi \vdash \Lambda, A[s]_{\Xi}}{\Gamma, \Pi \vdash \Delta, \Lambda, A[t]_{\Xi}} = (\Xi) : r2 \quad \frac{\Gamma \vdash \Delta, t = s \quad A[s]_{\Xi}, \Pi \vdash \Lambda}{A[t]_{\Xi}, \Gamma, \Pi \vdash \Delta, \Lambda} = (\Xi) : l2$$

where Ξ is a set of positions in A and s and t do not contain variables that are bound in A .

3.3 The Sequent Calculus LKDe

Definition introduction is a simple and very powerful tool in mathematical practice, allowing the easy introduction of important concepts and notations (e.g. groups, lattices, ...) by the introduction of new symbols. Therefore, **LKe** was extended in [BHL⁺06] to **LKDe** with the following rules:

- **Definition-rules:** They correspond directly to the *extension principle* in predicate logic and introduce new predicate and function symbols as abbreviations for formulas and terms. Let A be a first-order formula with the free variables x_1, \dots, x_k , denoted by $A(x_1, \dots, x_k)$, and P be a new k -ary predicate symbol (corresponding to the formula A). Then the rules are:

$$\frac{A(t_1, \dots, t_k), \Gamma \vdash \Delta}{P(t_1, \dots, t_k), \Gamma \vdash \Delta} d : l \quad \frac{\Gamma \vdash \Delta, A(t_1, \dots, t_k)}{\Gamma \vdash \Delta, P(t_1, \dots, t_k)} d : r$$

for arbitrary sequences of terms t_1, \dots, t_k .

3.4 HandyLK

In this subsection, we will focus on some particular features of *HandyLK*'s syntax⁷. These features make it more comfortable to write down proofs in *HandyLK* and then automatically translate them with HLK to **LKDe** instead of writing them directly and manually in **LKDe**.

When typing proofs in a naive notation for **LKDe** (e.g. a LISP-like notation for tree structures), two problems are encountered. Firstly, one would have to repeat many formulas that are not changed by the application of inference rules, i.e. those formulas that occur in the contexts Γ , Δ , Π and Λ and are simply carried over from the premises to the conclusions. Secondly, large branching proofs quickly become unreadable due to the nesting of subproofs within subproofs. The partial *HandyLK* code below shows how the first problem is avoided in the system's intermediary language for proof formalization.

```

define proof \varphi_1
  proves
    all x ( not P(x) or Q(x) ) :- all x ex y ( not P(x) or Q(y) );

  with all right
    :- ex y ( not P(\alpha) or Q(y) );
  with all left
    not P(\alpha) or Q(\alpha) :- ;
  with ex right
    :- not P(\alpha) or Q(\alpha);

  continued auto propositional
    not P(\alpha) or Q(\alpha) :- not P(\alpha) or Q(\alpha);
;

```

The code above starts by declaring a proof and the end-sequent that it derives. Then it lists the rules that have to be applied successively to the end-sequent in a bottom up way. For each inference, only the auxiliary formulas have to be specified. Context formulas are handled automatically by HLK.

Another useful feature shown above is the *HandyLK* command “**continued auto propositional**”. It specifies that the current sequent can be derived from tautological axioms by application of propositional rules. In such cases, HLK is able to produce the proof automatically.

For the example above, HLK generates the following **LKDe** proof:

$$\begin{array}{c}
\varphi_1 := \\
\frac{\frac{\frac{P(\alpha) \vdash P(\alpha)}{\vdash P(\alpha), \neg P(\alpha)} \neg : r}{\neg P(\alpha) \vdash \neg P(\alpha)} \neg : l}{\neg P(\alpha) \vdash \neg P(\alpha) \vee Q(\alpha)} \vee : r1 \quad \frac{Q(\alpha) \vdash Q(\alpha)}{Q(\alpha) \vdash \neg P(\alpha) \vee Q(\alpha)} \vee : r2}{\frac{\neg P(\alpha) \vee Q(\alpha) \vdash \neg P(\alpha) \vee Q(\alpha), \neg P(\alpha) \vee Q(\alpha)}{\neg P(\alpha) \vee Q(\alpha) \vdash \neg P(\alpha) \vee Q(\alpha)} c : r}{\frac{\neg P(\alpha) \vee Q(\alpha) \vdash (\exists y)(\neg P(\alpha) \vee Q(y))}{(\forall x)(\neg P(x) \vee Q(x)) \vdash (\exists y)(\neg P(\alpha) \vee Q(y))} \exists : r}{\frac{(\forall x)(\neg P(x) \vee Q(x)) \vdash (\exists y)(\neg P(\alpha) \vee Q(y))}{(\forall x)(\neg P(x) \vee Q(x)) \vdash (\forall x)(\exists y)(\neg P(x) \vee Q(y))} \forall : l} \forall : l
\end{array}$$

⁷HandyLK full syntax specification: <http://www.logic.at/hlk/handy-syntax.pdf>

The other code below shows how *HandyLK* avoids the problem of nesting and branching. A premise of any inference can have a link to another proof instead of having the auxiliary subsequent of that premise (in the example below, the left branch has a link to the subproof φ_1 and the right branch, a link to φ_2). Hence, large subproofs can be specified somewhere else in the file, instead of appearing nested inside the proof.

```
define proof \varphi
  proves
    P(a), all x ( not P(x) or Q(x) ) :- ex y Q(y);

  with cut all x ex y ( not P(x) or Q(y) )
    left by proof \varphi_1
    right by proof \varphi_2;
;
```

Finally it should be noted that *HandyLK* allows parameterizing meta-terms and meta-formulas in proofs. Hence it is thus capable of expressing recursive definitions of proof schemata, enabling the user to define infinite proof sequences⁸. In the code below, for example, a recursive proof is defined, parameterized by the meta-term n . The proof at level n can refer to the proof at level $n - 1$, $pr[n - 1]$, in the same way that the recursive function fr at level n , $fr[n]$, refers to $fr[n - 1]$.

```
define function f of type nat to nat;

define recursive function fr( x ) to nat
  at level 0 by x
  at level n by f( fr[n - 1](x) )
;

define recursive proof pr
  at level 0 proves
    P(a) :- P( fr[0](a) );

  ...

  at level n proves
    P(a), all x ( P(x) impl P( f(x) ) ) :- P( fr[n](a) );

  ...
;
```

HLK can output the **LKDe** proof either as a \LaTeX file or as an XML file satisfying to the proofdatabase DTD.

4 CERES

After the proof has been formalized in **LKDe** with HLK, it can be transformed with the CERES system. Among the transformations that can be performed by CERES the most interesting one is

⁸The Exponential Proofs (<http://www.logic.at/ceres/examples/index.html>) are a good example of *HandyLK* being used to define infinite proof sequences with parameterized meta-terms and meta-formulas

cut-elimination, which produces a proof in atomic-cut normal form. The ACNF is mathematically interesting, because cut-elimination in formal proofs corresponds to the elimination of lemmas in informal proofs. Hence the ACNF corresponds to an informal mathematical proof that is analytic in the sense that it does not use auxiliary notions that are not already explicit in the theorem itself.

To perform cut-elimination, CERES uses a method that employs the resolution calculus to eliminate cuts in a global way, instead of the more traditional local reductive methods [Gen69]. The CERES method has been described in various degrees of detail and with different emphasis in [BL00, BHL⁺05, BHL⁺06, BHL⁺] and hence it is just sketched in Subsection 4.1.

Although the ACNF is mathematically interesting, it is too large and full of redundant information. Its direct analysis and interpretation by humans is therefore too difficult. To overcome this, an algorithm to extract the essential information of proofs has been recently implemented within CERES. It performs *Herbrand sequent extraction* and is sketched in Subsection 4.2.

4.1 The CERES Method

The CERES method transforms any **LKDe**-proof with cuts into an *atomic-cut normal form* (ACNF) containing no non-atomic cuts. The remaining atomic cuts are, generally, non-eliminable, because **LKDe** admits non-tautological axiom sequents.

The transformation to ACNF via Cut-Elimination by Resolution is done according to the following steps:

1. Construct the (always unsatisfiable [BL00]) *characteristic clause set* of the original proof by collecting, joining and merging sets of clauses defined by the ancestors of cut-formulas in the axiom sequents of the proof.
2. Obtain from the characteristic clause set a grounded resolution refutation, which can be seen as an **LKe**-proof by exploiting the fact that the resolution rule is essentially a cut-rule restricted to atomic cut-formulas only and by mapping paramodulations to equality-inferences.
3. For each clause of the characteristic clause set, construct a *projection* of the original proof with respect to the clause.
4. Construct the ACNF by plugging the projections into the corresponding clauses in the leaves of the grounded resolution refutation tree (seen as an **LKe**-proof) and by adjusting the resulting proof with structural inferences (weakening, contractions and permutations) if necessary. Since the projections do not contain cuts and the refutation contains atomic cuts only, the resulting **LKDe** proof will indeed be in atomic-cut normal form.

Theoretical comparisons of the CERES with reductive methods can be found in [BL00, BL06]. For illustration, consider the following example:

$\varphi =$

$$\frac{\varphi_1 \quad \varphi_2}{(\forall x)(P(x) \rightarrow Q(x)) \vdash (\exists y)(P(a) \rightarrow Q(y))} \textit{cut}$$

$\varphi_1 =$

$$\frac{\frac{\frac{\frac{P(u)^* \vdash P(u) \quad Q(u) \vdash Q(u)^*}{P(u)^*, P(u) \rightarrow Q(u) \vdash Q(u)^*} \rightarrow: l}{P(u) \rightarrow Q(u) \vdash (P(u) \rightarrow Q(u))^*} \rightarrow: r}{P(u) \rightarrow Q(u) \vdash (\exists y)(P(u) \rightarrow Q(y))^*} \exists: r}{\frac{(\forall x)(P(x) \rightarrow Q(x)) \vdash (\exists y)(P(u) \rightarrow Q(y))^*}{(\forall x)(P(x) \rightarrow Q(x)) \vdash (\forall x)(\exists y)(P(x) \rightarrow Q(y))^*} \forall: l} \forall: r$$

$\varphi_2 =$

$$\frac{\frac{\frac{\frac{P(a) \vdash P(a)^* \quad Q(v)^* \vdash Q(v)}{P(a), (P(a) \rightarrow Q(v))^* \vdash Q(v)} \rightarrow: l}{(P(a) \rightarrow Q(v))^* \vdash P(a) \rightarrow Q(v)} \rightarrow: r}{(P(a) \rightarrow Q(v))^* \vdash (\exists y)(P(a) \rightarrow Q(y))} \exists: r}{\frac{(\exists y)(P(a) \rightarrow Q(y))^* \vdash (\exists y)(P(a) \rightarrow Q(y))}{(\forall x)(\exists y)(P(x) \rightarrow Q(y))^* \vdash (\exists y)(P(a) \rightarrow Q(y))} \exists: l} \forall: l$$

The crucial information extracted by *CERES* is the characteristic clause set, which is based on the ancestors of the cut-formulas (marked here by \star). For this proof, the set is $CL(\varphi) = \{P(u) \vdash Q(u); \vdash P(a); Q(v) \vdash\}$. This set is always unsatisfiable, here the resolution refutation δ of $CL(\varphi)$

$$\frac{\frac{\frac{\vdash P(a) \quad P(u) \vdash Q(u)}{\vdash Q(a)} R}{\vdash} R \quad Q(v) \vdash R}{\vdash} R$$

does the job. By applying the most general unifier σ of δ , we obtain a ground refutation $\gamma = \delta\sigma$:

$$\frac{\frac{\frac{\vdash P(a) \quad P(a) \vdash Q(a)}{\vdash Q(a)} R}{\vdash} R \quad Q(a) \vdash R}{\vdash} R$$

This will serve as a skeleton for a proof in ACNF. To complete the construction, we have to compute the projections to the clauses that are used in the refutation, this essentially works by leaving out inferences on ancestors of cut-formulas and applying σ :

$\varphi(C_1) =$

$$\frac{\frac{\frac{P(a) \vdash P(a) \quad Q(a) \vdash Q(a)}{P(a), P(a) \rightarrow Q(a) \vdash Q(a)} \rightarrow: l}{P(a), (\forall x)(P(x) \rightarrow Q(x)) \vdash Q(a)} \forall: l}{P(a), (\forall x)(P(x) \rightarrow Q(x)) \vdash (\exists y)(P(a) \rightarrow Q(y)), Q(a)} w: r$$

$\varphi(C_2) =$

$$\frac{\frac{\frac{P(a) \vdash P(a)}{P(a) \vdash P(a), Q(v)} w: r}{\vdash P(a) \rightarrow Q(v), P(a)} \rightarrow: r}{\vdash (\exists y)(P(a) \rightarrow Q(y)), P(a)} \exists: r}{(\forall x)(P(x) \rightarrow Q(x)) \vdash (\exists y)(P(a) \rightarrow Q(y)), P(a)} w: l$$

$$\varphi(C_3) = \frac{\frac{\frac{Q(a) \vdash Q(a)}{P(a), Q(a) \vdash Q(a)} w:l}{Q(a) \vdash P(a) \rightarrow Q(a)} \rightarrow:r}{Q(a) \vdash (\exists y)(P(a) \rightarrow Q(y))} \exists:r}{Q(a), (\forall x)(P(x) \rightarrow Q(x)) \vdash (\exists y)(P(a) \rightarrow Q(y))} w:l$$

Finally, we combine the projections and the ground refutation:

$$\varphi(\gamma) = \frac{\frac{\varphi(C_2)}{B \vdash C, P(a)} \quad \frac{\varphi(C_1)}{P(a), B \vdash C, Q(a)} \quad \frac{\varphi(C_3)}{Q(a), B \vdash C}}{B, B \vdash C, C, Q(a)} \text{ cut}}{\frac{B, B, B \vdash C, C, C}{B \vdash C} \text{ contractions}}$$

where $B = (\forall x)(P(x) \rightarrow Q(x))$, $C = (\exists y)(P(a) \rightarrow Q(y))$. Clearly, $\varphi(\gamma)$ is a proof of the end-sequent of φ in ACNF.

4.2 Herbrand Sequent Extraction

Our motivation to devise and implement Herbrand sequent extraction algorithms was the need to analyze and understand the result of proof transformations performed automatically by the CERES system, especially the ACNF.

Definition 4.1 (Herbrand Sequents of a Sequent). Let s be a sequent without free-variables and containing weak quantifiers only (i.e. universal quantifiers occur only with negative polarity and existential quantifiers occur only with positive polarity). We denote by s_0 the sequent s after removal of all its quantifiers. Any propositionally valid sequent in which the antecedent (respectively, succedent) formulas are instances (i.e. their free variables are possibly instantiated by other terms) of the antecedent (respectively, succedent) formulas of s_0 is called a *Herbrand sequent* of s .

Let s be an arbitrary sequent and s' a skolemization of s . Any Herbrand sequent of s' is a Herbrand sequent of s .

Theorem 4.1 (Herbrand's Theorem). A sequent s is valid if and only if there exists a Herbrand sequent of s .

Proof. Originally in [Her30], stated for Herbrand disjunctions. Also in [Bus95] with more modern proof calculi. \square

Herbrand's theorem guarantees that we can always obtain a Herbrand sequent from a correct proof, a possibility that was firstly exploited by Gentzen in his Mid-Sequent Theorem, which provides an algorithm for the extraction of Herbrand sequents based on the downward shifting of quantifier rules. However, Gentzen's algorithm has one strong limitation: it is applicable only to proofs with end-sequents in prenex form. Although we could transform the end-sequents and the proofs to prenex form, this would compromise the readability of the formulas and require additional computational effort. Prenexification is therefore not desirable in our context, and hence, to overcome this and other limitations in Gentzen's algorithm, we developed and implemented another algorithm.

To extract a Herbrand sequent of the end-sequent s of an **LKDe**-proof φ without cuts containing quantifiers, the implemented algorithm executes two transformations:

1. Ψ ([WP08]): produces a quantifier-rule-free **LKe_A**-proof where quantified formulas are replaced by array-formulas containing their instances.
2. Φ (definition 4.3): transforms the end-sequent of the resulting **LKe_A**-proof into an ordinary sequent containing no array-formulas. The result is a Herbrand sequent of the end-sequent of the original proof.

Definition 4.2 (Sequent Calculus **LKe_A**). The sequent calculus **LKe_A** is the sequent calculus **LKe** extended with the following array-formation-rules:

$$\frac{\Delta, A_1, \Gamma_1, \dots, A_n, \Gamma_n, \Pi \vdash \Lambda}{\Delta, \langle A_1, \dots, A_n \rangle, \Gamma_1, \dots, \Gamma_n, \Pi \vdash \Lambda} \langle \rangle : l \quad \frac{\Lambda \vdash \Delta, A_1, \Gamma_1, \dots, A_n, \Gamma_n, \Pi}{\Lambda \vdash \Delta, \langle A_1, \dots, A_n \rangle, \Gamma_1, \dots, \Gamma_n, \Pi} \langle \rangle : r$$

The intuitive idea behind the computation of $\Psi(\varphi)$ for an **LKDe**-proof φ consists of omitting quantifier-inferences and definition-inferences and replacing unsound contraction-inferences by array-formation-inferences (omissions and replacements are done in a top-down way and formulas in the downward path of a changed formula are changed accordingly). More precisely, if a quantified formula is the main formula of a contraction, then this contraction will not be sound anymore after the omission of the quantifier-inferences, because its auxiliary formulas will now contain different instances of the quantified formula instead of being exact copies of the quantified formula. Hence, the unsound contraction inferences are replaced by array-formation inferences, which will collect all the instances of that quantified formula into an array formula. This idea has been formally defined in [WP08, HLWWP08]. However, the implemented algorithm follows an equivalent recursive definition, which is more natural to implement but quite technical, preventing a clean exposition of the fundamental idea of the transformation to **LKe_A**.

Definition 4.3 (Φ : Expansion of Array Formulas). The mapping Φ transforms array formulas and sequents into first-order logic formulas and sequents. In other words, Φ eliminates $\langle \dots \rangle$ and can be defined inductively by:

1. If A is a first-order logic formula, then $\Phi(A) \doteq A$
2. $\Phi(\langle A_1, \dots, A_n \rangle) \doteq \Phi(A_1), \dots, \Phi(A_n)$
3. If $\Phi(A) = A_1, \dots, A_n$, then $\Phi(\neg A) \doteq \neg A_1, \dots, \neg A_n$
4. If $\Phi(A) = A_1, \dots, A_n$ and $\Phi(B) = B_1, \dots, B_m$, then $\Phi(A \circ B) \doteq A_1 \circ B_1, \dots, A_1 \circ B_m, \dots, A_n \circ B_1, \dots, A_n \circ B_m$, for $\circ \in \{\wedge, \vee, \rightarrow\}$
5. $\Phi(A_1, \dots, A_n \vdash B_1, \dots, B_m) \doteq \Phi(A_1), \dots, \Phi(A_n) \vdash \Phi(B_1), \dots, \Phi(B_m)$

Example 4.1. Let φ be the following **LKDe**-proof:

$$\begin{array}{c} [\varphi'] \\ \frac{P(0), P(0) \rightarrow P(s(0)), P(s(0)) \rightarrow P(s^2(0)) \vdash P(s^2(0))}{P(0), P(0) \rightarrow P(s(0)), (\forall x)(P(x) \rightarrow P(s(x))) \vdash P(s^2(0))} \forall : l \\ \frac{P(0), P(0) \rightarrow P(s(0)), (\forall x)(P(x) \rightarrow P(s(x))) \vdash P(s^2(0))}{P(0), \forall x(P(x) \rightarrow P(s(x))), (\forall x)(P(x) \rightarrow P(s(x))) \vdash P(s^2(0))} \forall : l \\ \frac{P(0), \forall x(P(x) \rightarrow P(s(x))), (\forall x)(P(x) \rightarrow P(s(x))) \vdash P(s^2(0))}{P(0), \forall x(P(x) \rightarrow P(s(x))) \vdash P(s^2(0))} c : l \\ \frac{P(0), \forall x(P(x) \rightarrow P(s(x))) \vdash P(s^2(0))}{P(0) \wedge (\forall x)(P(x) \rightarrow P(s(x))) \vdash P(s^2(0))} \wedge : l \end{array}$$

The **LKDe**-pseudoproof φ_c with unsound contractions, resulting from the omission of quantifier-inferences and definition-inferences is:

$$\frac{\frac{P(0), P(0) \rightarrow P(s(0)), P(s(0)) \rightarrow P(s^2(0)) \vdash P(s^2(0))}{P(0), \nabla \vdash P(s^2(0))} \text{c}^* : l}{P(0) \wedge \nabla \vdash P(s^2(0))} \wedge : l \quad [\varphi']$$

where ∇ stands for the undeterminable main formula of the unsound contraction-inference, since it contains different auxiliary formulas.

The **LKe_A**-proof $\Psi(\varphi)$, after replacement of unsound contractions by array-formations, is:

$$\frac{\frac{P(0), P(0) \rightarrow P(s(0)), P(s(0)) \rightarrow P(s^2(0)) \vdash P(s^2(0))}{P(0), \langle P(0) \rightarrow P(s(0)), P(s(0)) \rightarrow P(s^2(0)) \rangle \vdash P(s^2(0))} \langle \rangle : l}{P(0) \wedge \langle P(0) \rightarrow P(s(0)), P(s(0)) \rightarrow P(s^2(0)) \rangle \vdash P(s^2(0))} \wedge : l \quad [\varphi']$$

Let s be the end-sequent of $\Psi(\varphi)$. Then $\Phi(s)$, which is a Herbrand sequent extracted from φ , is:

$$\left(\begin{array}{l} P(0) \wedge (P(0) \rightarrow P(s(0))), \\ P(0) \wedge (P(s(0)) \rightarrow P(s^2(0))) \end{array} \right) \vdash P(s^2(0))$$

4.2.1 Further Improvements of Herbrand Sequent Extraction

Preliminary tests of the implemented algorithm described above showed that the extracted Herbrand sequent still contained information that was irrelevant for the interpretation of the ACNF to obtain a new informal mathematical proof corresponding to the ACNF. Such irrelevant information occurred in the form of subformulas of the Herbrand sequent that were introduced by weakening-inferences or as the weak subformula of the main formula of some other inference. This problem was solved by omitting the weak subformulas in the construction of $\Psi(\varphi)$. This improvement implies that the extracted sequent is not strictly a Herbrand sequent anymore, because its formulas are not instances of the formulas of the original end-sequent. However, the extracted sequent is still valid and contains the desired relevant information about the used variable instantiations in φ .

5 ProofTool

ProofTool is the graphical user interface that displays the original formal proof, the skolemized proof, the characteristic clause set, the projections, the ACNF and the Herbrand sequent to the user, so that he can analyze them and obtain a better understanding of the original proof or a new direct informal proof for the same theorem based on the Herbrand sequent and on the ACNF. It allows the user to zoom in and out of the proofs and to navigate to subproofs defined by proof links. Although it is mostly intended for proof visualization, some simple forms of proof editing, e.g. formula substitution and splitting of proofs by the creation of new proof links, are also possible.

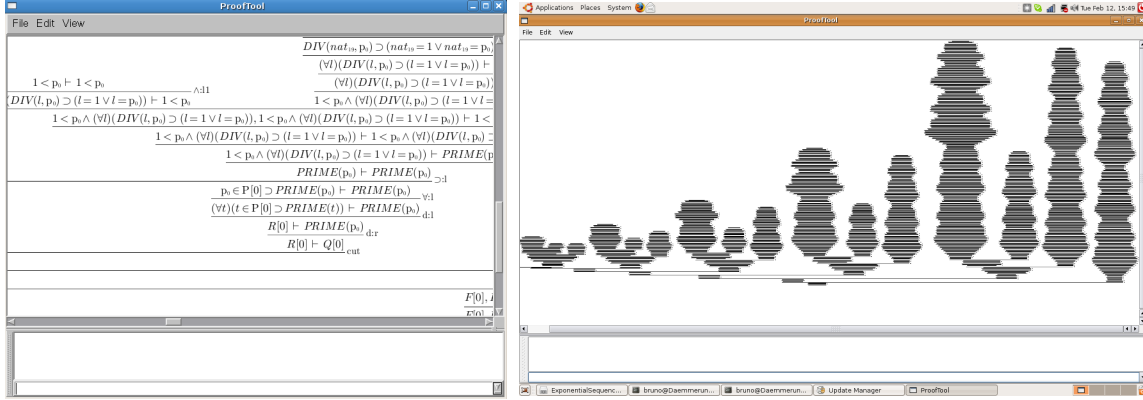


Figure 2: ProofTool Screenshots

6 Successful Experimental Cases

In this section, we summarize three successful cases in which the system was used for the analysis of mathematical proofs. All the examples below constitute interesting proofs for cut-elimination because they contain mathematical concepts in the cut-formulas which do not occur in the theorems shown. More information about each case can be found in the respective referred papers containing the detailed analysis.

6.1 The Tape Proof

The first proof that was analyzed using CERES (in [BHL⁺05] within **LK** and in [BHL⁺06] within **LKDe**) was originally defined in [Urb00]. This proof deals with the following situation: We are given an infinite tape where each cell contains either ‘0’ or ‘1’. We prove that on this tape there are two cells with the same value. The original proof goes on to show that on the tape, there are infinitely many cells containing ‘0’ or infinitely many cells containing ‘1’. From this, it follows that there are two cells having the same value. Clearly, these lemmas are much stronger than what is actually needed to prove the theorem, and exactly these lemmas were eliminated by the use of CERES.

By using different resolution refinements (positive and negative hyperresolution), two different resolution refutations (resulting in different ACNFs) were produced. After analysis of the argument contained in the refutations, it turns out that the refutations differ not only formally, but also in their mathematical interpretation. This showed that for any particular proof ψ , CERES may produce several ACNFs of the end-sequent of ψ and each of them may contain a mathematically different argument. This example exhibited the diversity of analytic proofs that may be obtained by using CERES.

Furthermore, in [Het07], two different tape proofs with different cut-formulas were analyzed comparatively. It has been observed how the characteristic clause sets of each proof determine and restrict the kinds of mathematical arguments and direct informal mathematical proofs that can be obtained from the ACNFs of each proof. This shows the potential of CERES not only for the analysis of single proofs, but also for the comparative analysis of different proofs of the same theorem.

6.2 The Prime Proof

In [BHL⁺], the system was used to analyze Fürstenberg’s proof of the infinity of primes. This proof uses topological concepts in its lemmas and proves that there are infinitely many primes. A central part in the formalization of this proof was the possibility of expressing proof schemas in HLK— here, it was used to formalize the inductive parts of the proof (e.g. that the union of closed sets is closed). Each proof $\varphi(k)$ according to the proof schema φ expresses that there are more than k primes. The collection of all such proofs implies that there are infinitely many primes.

The refutation and analysis of the *characteristic clause set* extracted by CERES led to a direct informal proof of the infinity of primes, which corresponded to the well-known Euclid’s argument for the infinity of primes. Thus CERES could essentially transform Fürstenberg’s proof into Euclid’s proof.

6.3 The Lattice Proof

In [HLWWP08], the usefulness of a Herbrand sequent for understanding a formal proof was demonstrated on a simple proof from lattice theory showing that $L1$ -lattices (semi-lattices satisfying “inverse” laws) are $L2$ -lattices (semi-lattices satisfying the absorption laws) by firstly showing that $L1$ -lattices are $L3$ -lattices (partially-ordered sets with greatest lower bounds and least upper bounds) and then showing that $L3$ -lattices are $L2$ -lattices.

The formalization of the proof with HLK resulted in a proof with 260 inferences where the concept of $L3$ -lattice appears, as expected, as a cut-formula in a cut-inference. The elimination of cuts with CERES resulted in a proof in atomic-cut normal form (ACNF) containing 214 inferences and no essential cuts. The informal proof corresponding to the ACNF would be a direct mathematical argument proving that $L1$ -lattices are $L2$ -lattices. However, due to its size, the interpretation of the ACNF to extract this informal proof is hardly possible. On the other hand, the Herbrand sequent extracted from the ACNF contains only 6 formulas. Analyzing the Herbrand sequent alone, it was possible to obtain the desired informal direct proof.

7 Future Directions

The development of our systems focuses on the analysis of mathematical proofs. However, it could also be extended to other applications of the transformation and analysis of formal proofs, e.g. for the computation of interpolants in symbolic model checking [Mcm03, HJMM04].

In the following subsections, we will discuss a few improvements to the system formed by CERES, HLK and ProofTool that are currently under development or that shall be developed in the future.

7.1 The Proof Profile

The proof profile [Het07, Het08] is a refinement of the characteristic clause set which, in addition to the logical information about the cut-formulas, incorporates also combinatorial information about the structure of the proof. This leads to an optimization of the CERES method in the sense that atomic-cut normal forms generated by using the profile will always be at most the size of those generated by using the characteristic clause set, and furthermore, the profile is stronger in detecting certain kinds of redundancies which can yield even a non-elementary speed-up with respect to an

atomic-cut normal form generated by using the characteristic clause set. Moreover, not only is the proof profile a practical optimization but it also gives important theoretical benefits: they have been used in [HL07] to obtain new results about the relation between simple proof transformations and cut-elimination. Therefore, CERES will be extended to make it possible to use the proof profile instead of the characteristic clause set.

7.2 Extension to Second-order Logic

Second-order logic extends first-order logic by allowing quantification over set variables. The reasons for considering an extension of the *CERES* method to second-order logic are twofold: first, the second-order induction axiom

$$(\forall X)(0 \in X \wedge (\forall x)(x \in X \rightarrow x' \in X) \rightarrow (\forall x)x \in X)$$

enables proofs by induction to be handled on the object level, whereas induction in first-order logic can only be handled by (non-tautological) rules or by proof schemas. This then allows the induction component of proofs to be handled in the same way as any other formula occurring in a proof. Secondly, a comprehension axiom scheme

$$(\exists X)(\forall x)(x \in X \leftrightarrow \varphi(x))$$

can be formulated in second-order logic, which allows one to assert the existence of sets defined by formulas $\varphi(x)$. Certainly, the definition of sets by formulas is a very natural mathematical operation: consider as an example the sentence

$$(\exists X)(\forall x)(x \in X \leftrightarrow (\exists z)2 * z = x)$$

which asserts the existence of the set of even numbers.

Note that the second-order axioms of induction and comprehension are preferable to the corresponding schemes of first-order arithmetic and set theory in this context: the goal of proof analysis by cut-elimination is to remove certain mathematical concepts from the proof. If, however, these concepts appear in instances of axiom schemes in the end-sequent of the proof with cuts, they will also appear in the end-sequent of the cut-free (or ACNF) proof. In the second-order formulation, preserving the end-sequent does not imply preserving the instances of the schemes.

As second-order logic distinguishes between set variables and individual variables, it does not suffer from Russel's paradox. Still, second-order arithmetic (even with restrictions on $\varphi(x)$ in the comprehension scheme) is powerful enough to prove a large part of ordinary mathematics (see [Sim99]).

Extending *CERES* to second-order logic is non-trivial: In first-order logic, proof skolemization is used as an essential technical tool to remove strong quantifier rules going into the end-sequent from the proof. In second-order logic, the corresponding transformation can only be applied in very special cases (we must restrict the way in which weak second-order quantifiers occur). Consider for example:

$$\begin{array}{c}
\frac{P(\beta, a) \vdash P(\beta, a)}{(\forall x)P(x, a) \vdash P(\beta, a)} \forall : l \quad \frac{P(\alpha, b) \vdash P(\alpha, b)}{(\forall z)P(z, b) \vdash P(\alpha, b)} \forall : l \\
\frac{(\forall x)P(x, a) \vdash (\forall z)P(z, a)}{(\forall x)P(x, a) \vdash (\forall z)P(z, a)} \forall : r \quad \frac{(\forall z)P(z, b) \vdash (\forall z)P(z, b)}{(\forall z)P(z, b) \vdash (\forall z)P(z, b)} \forall : r \\
\frac{(\forall x)P(x, a), (\forall z)P(z, a) \rightarrow (\forall z)P(z, b) \vdash (\forall x)P(x, b)}{(\forall x)P(x, a), (\forall X)(X(a) \rightarrow X(b)) \vdash (\forall x)P(x, b)} \rightarrow : l \\
\frac{(\forall x)P(x, a), (\forall X)(X(a) \rightarrow X(b)) \vdash (\forall x)P(x, b)}{(\forall X)(X(a) \rightarrow X(b)) \vdash (\forall x)P(x, a) \rightarrow (\forall x)P(x, b)} \rightarrow : r
\end{array}$$

Skolemization of the proof would yield

$$\begin{array}{c}
\frac{P(s_2, a) \vdash P(s_2, a)}{(\forall x)P(x, a) \vdash P(s_2, a)} \forall : l \quad \frac{P(s_1, b) \vdash P(s_1, b)}{(\forall z)P(z, b) \vdash P(s_1, b)} \forall : l \\
\frac{(\forall x)P(x, a), P(s_2, a) \rightarrow (\forall z)P(z, b) \vdash P(s_1, b)}{(\forall x)P(x, a), (\forall X)(X(a) \rightarrow X(b)) \vdash P(s_1, b)} \rightarrow : l \\
\frac{(\forall x)P(x, a), (\forall X)(X(a) \rightarrow X(b)) \vdash P(s_1, b)}{(\forall X)(X(a) \rightarrow X(b)) \vdash (\forall x)P(x, a) \rightarrow P(s_1, b)} \rightarrow : r
\end{array}$$

where the $\forall^2 : l$ rule application is clearly not sound. For this reason, we will investigate an extension of the *CERES* method that deals with proofs that have not been skolemized.

Also, resolution in second-order logic loses some nice properties of its first-order counterpart: first-order logic is semi-recursive and therefore we can always find a resolution refutation of the characteristic clause set. This is not the case for second-order logic. Still, there exist implementations of higher-order resolution provers (e.g. Leo, see [BK98]), which we plan to adapt for use with *CERES* in second-order logic.

7.3 Handling Non-skolemized Proofs and Elimination of Single Cuts

As mentioned in the previous section, skolemization of proofs in second-order logic is not always possible. Also in first-order logic, it would be advantageous to be able to apply the *CERES* method to non-skolemized proofs: consider, for example, a proof φ containing a proof ψ as a subproof. Applying *CERES* to the skolemization of ψ yields an ACNF ψ' . It is then not possible in general to put ψ' in place of ψ in φ : the end-sequent of ψ may contain formula occurrences that are ancestors of cut-formulas in φ , and skolemization of these formulas will prevent these cuts from being applied (as the cut-formula occurrences have different polarities). This leads to the fact that *CERES* currently only supports elimination of all cuts of a proof at once.

On the other hand, if *CERES* is able to handle non-skolemized proofs this immediately gives rise to a method to eliminate a single uppermost cut ρ in φ by isolating the subproof ending with ρ and applying *CERES* to it.

Our experiments with applying *CERES* to Fürstenberg's proof on the infinity of primes showed that current automated theorem provers are too weak to handle the characteristic clause sets of larger proofs. With a method for eliminating single cuts, the problem of performing cut-elimination on a proof φ can then be reduced to a sequence of cut-eliminations on proofs ψ_i that contain only one non-atomic cut. The characteristic clause sets of the ψ_i will be less complex than that of φ , which will enable the theorem provers to find a refutation more easily.

The main problem that occurs when considering non-skolemized proofs with *CERES* are eigenvariable violations of the strong quantifier rules that appear in the projections. There are different ideas on how this problem may be resolved, a promising approach is the following: we restrict the form of the strong quantifier rules going into the end-sequent by using, instead of

eigenvariables, skolem terms as *eigenterms* of the rules. In the resulting ACNF, we will then have violations of the eigenterm conditions of these rules by ancestors of cut-formulas. By a proof transformation (essentially reductive cut-elimination together with certain rule permutations) we can produce a valid ACNF. Intuitively, this works because reductive cut-elimination always shifts cuts upwards in a proof, so eventually the cut-formulas will be cut out at the top of the proof, and can not cause eigenterm violations below.

7.4 Extension to Superdeduction Calculi

The first logical calculi followed Hilbert's style of having few inference rules and many axiom schemas. The next generations of calculi such as natural deduction and sequent calculi substituted some axiom schemas by new inference rules. Formal proofs in these calculi were therefore closer to natural informal mathematical proofs, simply because the new inference rules corresponded more closely to some kinds of inferences that are usually done in informal mathematical argumentation. However, the new inference rules substituted only axioms that contained purely *logical* information about the behavior of connectives and quantifiers. Informal mathematical argumentation, on the other hand, contains many informal inferences based on axioms containing *mathematical* information about a concrete domain of mathematical practice. The trend in the evolution of formal calculi for the actual formalization of mathematical proofs is to incorporate such mathematical axioms into rules of inference, in the same way that natural deduction and sequent calculi incorporated purely logical axioms into their rules of inference.

The use of arbitrary initial sequents, equation-rules and definition-rules in **LKDe** can be seen as a small step within this trend. Another related but bigger step, though, was done with the proposal of *superdeduction rules*[BHK07], which roughly correspond to a rigorous combination of our definition-rules with other **LK** inferences in such a way that all auxiliary formulas of the superdeduction inference are atomic. More precisely, superdeduction inferences can be easily simulated in **LKDe** by substituting them by many **LK** inferences followed by a definition-inference. Nevertheless, proofs using superdeduction inferences are shorter, more readable and closer to informal mathematical proofs than pure **LKDe**-proofs.

To support superdeduction inferences, the *CERES* method and CERES, HLK and ProofTool will have to be extended to support rules of arbitrary arity, because they currently work with unary and binary rules only and superdeduction rules can have an arbitrary number of premises.

7.5 Better Herbrand Sequents

Our algorithm for Herbrand sequent extraction currently lacks support for definition-rules, because definition-inferences must be omitted in the transformation to **LKe_A**. We plan to modify the algorithm, so that it reinserts defined formulas in the extracted Herbrand sequent, in order to further improve its readability. Better readability could also be achieved by post-processing and compressing long terms appearing in the formulas of Herbrand sequent. Furthermore, the usage of the Herbrand sequent as a guide for the construction of new informal direct mathematical proofs could be made easier by enriching the Herbrand sequent with the axiom links that were used in the proof, similarly to what is done in proof nets and atom flows.

7.6 Resolution Refinements for Cut-Elimination

The *CERES* method relies on a search for a refutation of the characteristic clause set. Although the characteristic clause set is known to be always unsatisfiable, the search space with unrestricted resolution or even with typical resolution refinements (e.g. hyper-resolution, unit-resolution, ...) can be so large that theorem provers like Otter and Prover9 do not find a refutation in a reasonable time. This occurs specially if the proof and its characteristic clause set are large or if the proof contains equality rules, in which case the refutation needs paramodulation.

We plan to develop resolution refinements that will exploit the structure of the proofs in order to restrict the search space and possibly even eliminate the need for search altogether.

7.7 Interactive Resolution Theorem Prover

In order to implement the planned resolution refinements, we have implemented a simple but flexible resolution theorem prover. It was designed with a focus on easy plugability of new refinements and on the possibility of interaction with the user. However, it supports only unrestricted resolution and paramodulation so far. It still needs to be tested with large clause sets and other refinements have to be implemented.

7.8 Improvements for HLK

The **continued auto propositional** feature of HLK is currently restricted to sequents that can be derived from *tautological* axiom sequents. However, since **LKDe** allows arbitrary atomic axioms specified by a background theory, it would be interesting if **continued auto propositional** were extended in order to generate proofs automatically also for sequents that can be propositionally derived from any arbitrary axioms, tautological or not.

7.9 Improvements for ProofTool

ProofTool supports global zooming and scrolling, which allow the user to navigate to different parts of the proof and analyze the structure of the proof in different degrees of detail. However, for very large proofs, navigating only via global zooming and scrolling has some disadvantages:

- If the user globally zooms in to see details of a part of the proof, he loses his view of whole proof. Moreover, if he needs to analyze details of various distant regions of the proof, he has to zoom in to one region, then zoom out and scroll to another far region, then zoom in again, and so on. This is sometimes very impractical. A solution to this problem could be the use of a magnifying glass, which would allow the user to locally zoom in, without losing his view of the whole proof, and to easily change the local zoom to other regions of the proof just by moving the magnifying glass.
- Sometimes the user needs to navigate to some specific regions of the proof, but its precise locations are not known a priori. Examples are:
 - Assume that the user is looking at the root inference of the subproof at the left branch of a binary inference. Then he might want to go to “the root inference of the subproof at the right branch of this binary inference”.

- If the user is looking at a certain formula occurrence, he might want to go to “the inferences that contain ancestors of this formula as their main occurrences”

In such situations, the user has to manually scroll and search for the desired regions. Ideally, this problem could be solved by some simple yet expressive enough query language for positions in proofs. In the short-term, keyboard shortcuts for typical queries could be implemented.

While the previous problems are mainly due to proof size, other problems arise from the existence of bureaucracy and trivial structural information in **LKDe**-proofs. Many features shall be added to **ProofTool** to overcome this. Unary structural inferences could be hidden; colors could be used to highlight different interesting objects (corresponding formulas, ancestor paths, inferences, subproofs) in the proof; and context formulas could be hidden.

Finally, **ProofTool**'s editing capabilities could be extended to allow simple proof transformations to be done directly via the graphical user interface.

7.10 Alternatives for *HandyLK* and HLK

HandyLK is not the only language for the formalization of mathematical proofs. Other languages and systems, such as **COQ**⁹, **Isabelle**¹⁰, **Mizar**¹¹ and **ForTheL**¹², also aim at helping mathematicians in this task. We would like to study the possibility of integrating these languages and systems to **CERES**, as alternatives to *HandyLK* and HLK. We foresee two potential compatibility obstacles. On the theoretical side, these languages and systems might not use the same logics and calculi that **CERES** uses. On the implementation side, these alternatives systems must easily output a formal proof object, and this proof object must be easily convertible to the internal data structures that **CERES** uses.

8 Conclusion

As summarized in Section 6, the system formed by HLK, **CERES** and **ProofTool** has been applied in the transformation and analysis of real mathematical proofs. It constitutes therefore a successful example of automated reasoning being employed in mathematics. In this paper, an overview of the system was given, with a special emphasis on recently added features (e.g. Herbrand sequent extraction) and on extensions that are currently being developed. We showed the current status of our efforts in employing automated proof theoretical methods in mathematical analysis, and we pointed the directions that we intend to follow in order to bring these methods even closer to mathematical practice.

9 Acknowledgements

The authors would like to thank the anonymous referees for their interesting comments and suggestions.

⁹COQ Website: <http://coq.inria.fr/>

¹⁰Isabelle Website: <http://www.cl.cam.ac.uk/research/hvg/Isabelle/>

¹¹Mizar Website: <http://mizar.org/>

¹²ForTheL Website: <http://nevidal.org.ua/>

References

- [BHK07] Paul Brauner, Clement Houtmann, and Claude Krichner. Principles of Superdeduction. In *Twenty-Second Annual IEEE Symposium on Logic in Computer Science (LiCS)*, 2007.
- [BHL⁺] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. Ceres: An Analysis of Fürstenberg’s Proof of the Infinity of Primes. to appear in *Theoretical Computer Science*.
- [BHL⁺05] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. Cut-Elimination: Experiments with CERES. In Franz Baader and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR) 2004*, volume 3452 of *Lecture Notes in Computer Science*, pages 481–495. Springer, 2005.
- [BHL⁺06] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. Proof Transformation by CERES. In Jonathan M. Borwein and William M. Farmer, editors, *Mathematical Knowledge Management (MKM) 2006*, volume 4108 of *Lecture Notes in Artificial Intelligence*, pages 82–93. Springer, 2006.
- [BK98] Christoph Benzmüller and Michael Kohlhase. Leo — A Higher-Order Theorem Prover. In *Proc. CADE-15*, volume 1421, pages 139–143, Heidelberg, Germany, 1998. Springer-Verlag.
- [BL00] Matthias Baaz and Alexander Leitsch. Cut-elimination and Redundancy-elimination by Resolution. *Journal of Symbolic Computation*, 29(2):149–176, 2000.
- [BL06] Matthias Baaz and Alexander Leitsch. Towards a clausal analysis of cut-elimination. *Journal of Symbolic Computation*, 41(3–4):381–410, 2006.
- [Bus95] S. R. Buss. On Herbrand’s theorem. *Lecture Notes in Computer Science*, 960:195, 1995.
- [Gen69] G. Gentzen. Untersuchungen über das logische Schließen. In M.E.Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland Publishing Company, Amsterdam - London, 1969.
- [Her30] J. Herbrand. *Recherches sur la Theorie de la Demonstration*. PhD thesis, University of Paris, 1930.
- [Het07] Stefan Hetzl. *Characteristic Clause Sets and Proof Transformations*. PhD thesis, Vienna University of Technology, 2007.
- [Het08] Stefan Hetzl. *Proof Profiles. Characteristic Clause Sets and Proof Transformations*. VDM, 2008.
- [HJMM04] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. Mcmillan. Abstractions from proofs. In *POPL ’04: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, volume 39, pages 232–244, New York, NY, USA, January 2004. ACM Press.
- [HL07] Stefan Hetzl and Alexander Leitsch. Proof Transformations and Structural Invariance. In Stefano Aguzzoli, Agata Ciabattoni, Brunella Gerla, Corrado Manara, and Vincenzo Marra, editors, *Algebraic and Proof-theoretic Aspects of Non-classical Logics*, volume 4460 of *Lecture Notes in Artificial Intelligence*, pages 201–230. Springer, 2007.
- [HLWWP08] Stefan Hetzl, Alexander Leitsch, Daniel Weller, and Bruno Woltzenlogel Paleo. Herbrand Sequent Extraction. to appear in the proceedings of *Mathematical Knowledge Management (MKM) 2008*, 2008.
- [McM03] K. L. McMillan. *Interpolation and SAT-Based Model Checking*. Lecture Notes in Computer Science. Springer, 2003.
- [Pol54a] G. Polya. *Mathematics and plausible reasoning, Volume I: Induction and Analogy in Mathematics*. Princeton University Press, 1954.
- [Pol54b] G. Polya. *Mathematics and plausible reasoning, Volume II: Patterns of Plausible Inference*. Princeton University Press, 1954.

- [Sim99] Stephen G. Simpson. *Subsystems of Second Order Arithmetic*. Springer-Verlag, Heidelberg, Germany, 1999.
- [Urb00] Christian Urban. *Classical Logic and Computation*. PhD thesis, University of Cambridge, October 2000.
- [WP08] Bruno Woltzenlogel Paleo. *Herbrand Sequent Extraction*. VDM-Verlag, Saarbruecken, Germany, 2008.