Transforming and Analyzing Proofs in the CERES-system [*]

Stefan Hetzl Alexander Leitsch Daniel Weller
Bruno Woltzenlogel Paleo
Institute of Computer Languages (E185),
Vienna University of Technology,
Favoritenstraße 9, 1040 Vienna, Austria

Abstract

Cut-elimination is the most prominent form of proof transformation in logic. The elimination of cuts in formal proofs corresponds to the removal of intermediate statements (lemmas) in mathematical proofs. Cut-elimination can be applied to *mine* real mathematical proofs, i.e. for extracting explicit and algorithmic information. The system CERES (cut-elimination by resolution) is based on automated deduction and was successfully applied to the analysis of nontrivial mathematical proofs. In this paper we focus on the input-output environment of CERES, and show how users can interact with the system and extract new mathematical knowledge.

1 Introduction

Cut-elimination introduced by Gentzen [6] is the most prominent form of proof transformation in logic and plays an important role in automating the analysis of mathematical proofs. The removal of cuts corresponds to the elimination of intermediate statements (lemmas) from proofs resulting in a proof which is analytic in the sense, that all statements in the proof are subformulas of the result. Therefore, the proof of a *combinatorial statement* is converted into a purely *combinatorial proof*.

In a formal sense Girard's famous analysis of van der Waerden's theorem [7] consists in the application of cut-elimination to the proof of Fürstenberg and Weiss (which uses topological arguments) with the "perspective" of obtaining van der Waerden's elementary proof. Indeed, an application of a complex proof transformation like cut-elimination by humans requires a goal oriented strategy.

CERES [4, 5] is a cut-elimination method that is based on resolution. The method roughly works as follows: The structure of an **LK**-proof containing cuts is mapped to an unsatisfiable set of clauses $\mathscr{C}$ (the *characteristic clause set*). A resolution refutation of $\mathscr{C}$, which is obtained using a first-order theorem prover, serves as a skeleton for the new proof which contains only atomic cuts (AC normal form). In a final step also these atomic cuts can be eliminated, provided the (atomic) axioms are valid sequents; but this step is of minor mathematical interest and of low complexity. In the system CERES[1] this method of cut-elimination has been implemented. The extension of CERES from **LK** to **LKDe**, a calculus containing definition introductions and equality rules (see [10] and [1]), moved the system closer to real mathematical proofs. The system CERES has been applied successfully to a well known mathematical proof, namely to Fürstenberg's proof of the infinity of primes [2]; it was shown that the elimination of topological arguments from the proof resulted in Euclid's famous proof. Though CERES did not (yet) produce substantial mathematical proofs previously unknown, the analysis of Fürstenberg's proof demonstrates the potential of the method to handle nontrivial mathematics.

The main task of the CERES-method is *proof transformation*, not just proof verification. For the latter one there are powerful higher-order systems like Isabelle[2] (see [12]) and Coq[3] (see [11]); these systems have been used successfully to verify complicated and famous mathematical proofs like this of the four color theorem (see `http://research.microsoft.com/~gonthier/`). The core algorithm of CERES, however, works on **LK**-proofs and performs cut-elimination. For this reason we have developed

*Supported by the Austrian Science Fund (project P19875)
[1] available at `http://www.logic.at/ceres/`
[2] available at `http://www.cl.cam.ac.uk/research/hvg/Isabelle/`
[3] available at `http://coq.inria.fr/`

the higher proof language HLK which is close to the sequent calculus **LK**, making translations to **LK** easy and efficient.

In this paper we present the input-output environment of CERES and illustrate the interaction of users with the system. In the first step proofs are formalized in HLK. The formalized proof is then compiled to **LKDe** and analyzed by the CERES-algorithm, and finally the resulting atomic cut normal form can be viewed by Prooftool. Moreover, from the normal form a Herbrand sequent can be extracted which contains mathematical information in a more condensed form. We illustrate all phases of proof specification and analysis by an example, the tape proof of Christian Urban (see also [1]).

2 System overview

Figure 1 sketches how HLK, CERES and ProofTool can be used by a mathematician to analyze existing mathematical proofs and obtain new ones. According to the labels in the edges of Figure 1, the following

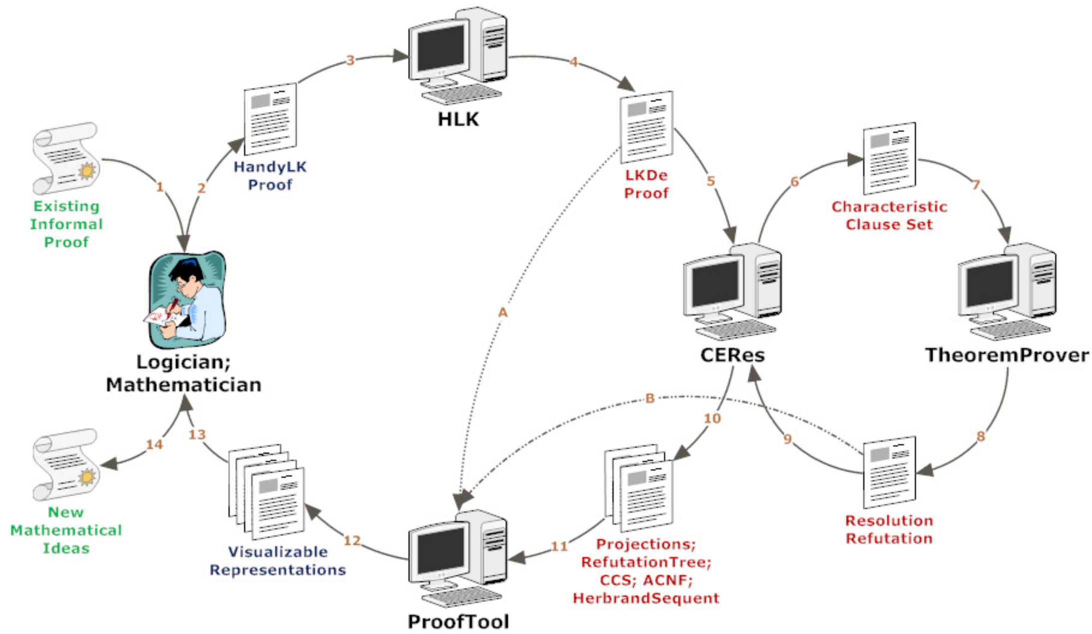


Figure 1: Working with HLK, CERES and ProofTool.

steps are executed within the system and in interaction with the user:

1. The user (intended to be a mathematician with a background in logic) 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*, a higher proof language, intermediary between natural mathematical language and formal calculi.

3. The *HandyLK* proof is input to the compiler HLK.

4. HLK generates a formal proof in sequent calculus **LKDe**.

5. The formal proof is input to CERES, which is responsible for all sorts of proof transformations, including cut-elimination.

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. Otter[4] or Prover9[5].

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 is obtained from the instantiation information of the quantifiers of the end-sequent (resulting in an (equationally) valid sequent) and output as well. The Herbrand sequent typically summarizes the creative content of the ACNF. For details, see [8].

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.

3 Proof analysis with CERES

Our calculus **LKDe** is based on standard **LK** with permutation, contraction and weakening, atomic axioms and multiplicative rules. For example, the following rules deal with the $\wedge$-connective:

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

In addition, to bring the calculus closer to mathematical practice, definition introduction and equality handling rules are used:

Let A be a first-order formula with the free variables $x_1, \ldots, x_k$ (denoted by $A(x_1, \ldots, x_k)$) and P be a *new* k-ary predicate symbol (corresponding to the formula A). Then the *definition introduction* rules are:

$$\frac{A(t_1, \ldots, t_k), \Gamma \vdash \Delta}{P(t_1, \ldots, t_k), \Gamma \vdash \Delta} \mathrm{def}_P : l \quad \frac{\Gamma \vdash \Delta, A(t_1, \ldots, t_k)}{\Gamma \vdash \Delta, P(t_1, \ldots, t_k)} \mathrm{def}_P : r$$

for arbitrary sequences of terms $t_1, \ldots, t_k$. Definition introduction is a simple and very powerful tool in mathematical practice. Note that the introduction of important concepts and notations like groups, integrals etc. can be formally described by introduction of new symbols. There are also definition introduction rules for new function symbols which are of similar type.

The *equality rules* (also called *paramodulation rules*) are:

[4]available at `http://www-unix.mcs.anl.gov/AR/otter/`
[5]available at `http://www.cs.unm.edu/~mccune/prover9/`

$$\frac{\Gamma_1 \vdash \Delta_1, s = t \quad A[s]_\Lambda, \Gamma_2 \vdash \Delta_2}{A[t]_\Lambda, \Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2} =: l1 \quad \frac{\Gamma_1 \vdash \Delta_1, t = s \quad A[s]_\Lambda, \Gamma_2 \vdash \Delta_2}{A[t]_\Lambda, \Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2} =: l2$$

for inference on the left and

$$\frac{\Gamma_1 \vdash \Delta_1, s = t \quad \Gamma_2 \vdash \Delta_2, A[s]_\Lambda}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, A[t]_\Lambda} =: r1 \quad \frac{\Gamma_1 \vdash \Delta_1, t = s \quad \Gamma_2 \vdash \Delta_2, A[s]_\Lambda}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, A[t]_\Lambda} =: r2$$

on the right, where Λ denotes a set of positions of subterms where replacement of s by t has to be performed. We call $s = t$ the *active equation* of the rules.

We will now introduce the CERES system by presenting the analysis of a proof from [13]; it was formalized in **LKDe** and analyzed by CERES in [1]. The end-sequent formalizes the statement: on a tape with infinitely many cells which are all labelled by 0 or by 1 there are two cells labelled by the same number. $f(x) = 0$ expresses that the cell nr. x is labelled by 0. Indexing of cells is done by number terms defined over $0, 1$ and $+$. The proof φ below uses two lemmas: (1) there are infinitely many cells labelled by 0 and (2) there are infinitely many cells labelled by 1. These lemmas are eliminated by CERES and a more direct argument is obtained in the resulting proof φ'.

3.1 Specifying proofs in *HandyLK*

In this section, we introduce the higher proof language *HandyLK* by presenting its most important features and syntax by means of our example proof. *HandyLK* is based on the idea that many aspects of writing **LKDe** proofs can be automatized. It supports a many-sorted first-order language.

Before starting to write proofs, it is first necessary to define the language and the function and predicate definitions one intends to use. In our example, we deal with a language with one sort (natural numbers) and the constants mentioned above. In *HandyLK*, we write:

```
define type nat;
define constant 0, 1 of type nat;
define infix function + of type nat,nat to nat with weight 100;
define function f of type nat to nat;
```

In the definition of infix functions like $+$, we may set a weight that allows terms to be written in the usual mathematical way, where superfluous brackets may be dropped. In addition to the constants, we define some variables that will be used in the proof. Here, the function f is the labelling function assigning labels to tape cells.

```
define variable k, l, n, p, q, x, n_0, n_1 of type nat;
```

We also define the axioms we will use. Note that `:-` represents $\vdash$ in *HandyLK*.

```
define axiom :- k + l = l + k;
define axiom :- k + (l + n) = (k + l) + n;
define axiom k = k + (1 + l) :- ;
define axiom :- k = k;
```

Finally, we introduce some predicate definitions.

```
define predicate A by all x ( f(x) = 0 or f(x) = 1 );
define predicate I by all n ex k f( n + k ) = x;
```

Note that in the definition of I, x is a free variable. The free variables determine the arity of the defined predicate, and can be instantiated by parameters. If P is a defined predicate, F is its defining formula and $\bar{x}$ are the free variables of F, then the predicate definition is interpreted as $\forall\bar{x}(P(\bar{x}) \iff F)$. In our example, the predicate A is the assumption: All cells are labelled either 0 or 1. The predicate $I(x)$ states that there are infinitely many cells labelled x.

We now turn to formalizing our example proof φ. In **LKDe**, φ is

$$\dfrac{\dfrac{\overset{(\tau)}{A \vdash I(0), I(1)} \quad \overset{(\varepsilon(0))}{I(0) \vdash \exists p \exists q (p \neq q \wedge f(p) = f(q))}}{A \vdash \exists p \exists q (p \neq q \wedge f(p) = f(q)), I(1)}\, cut \quad \overset{(\varepsilon(1))}{I(1) \vdash \exists p \exists q (p \neq q \wedge f(p) = f(q))}}{A \vdash \exists p \exists q (p \neq q \wedge f(p) = f(q))}\, cut$$

Here τ is a proof of the fact that either infinitely many cells are labelled 0, or infinitely many cells are labelled 1. The two applications of cut encode the application of two lemmas $\varepsilon(n)$ for $n \in \{0,1\}$: if there are infinitely many cells labelled n, then there are two cells labelled by the same number.

In *HandyLK*, we start the specification of the proof by giving an identifier, here we choose the name *the-proof*:

```
define proof the-proof
```

Next, the end-sequent is fixed:

```
  proves A :- ex p ex q ( not p = q and f(p) = f(q) );
```

Now, we begin by specifying the inferences from the bottom up. In *HandyLK*, only the active formulas of a rule have to be specified, and no structural rules (except cut) have to be written down explicitly. The HLK compiler keeps track of the formulas used in the proof, and automatically inserts the structural rules necessary to be able to apply the rules specified by the user. The first inference in φ is a cut, which is a binary rule. Due to the linear nature of the *HandyLK* language, one of the subproofs above the binary rule has to be referenced. The following proof reference clauses are available:

by proof ⟨*proof*⟩
auto propositional ⟨*sequent*⟩
explicit axiom ⟨*sequent*⟩

The first clause gives the name of the subproof with which to continue, the second clause states that the subproof ends in a propositional tautology whose proof should be computed automatically, and the last clause asserts that the subproof consists solely of an axiom. In our example we reference the proof $\varepsilon(1)$.

```
  with cut I(1)
    right by proof \epsilon(1);
```

This specifies an application of cut with cut formula $I(1)$, where the right subproof will be $\varepsilon(1)$. In **LKDe**, this will be a rule application

$$\dfrac{\Gamma \vdash \Delta, I(1) \quad \overset{(\varepsilon(1))}{I(1), \Pi \vdash \Lambda}}{\Gamma, \Pi \vdash \Delta, \Lambda}\, cut$$

where $\Gamma, \Pi, \Delta, \Lambda$ will be populated by the appropriate formulas automatically by the HLK compiler.

We continue the *HandyLK* specification of this proof part with the left subproof of the first application of cut by encoding the second application: here, we reference both the left subproof τ and the right subproof $\varepsilon(0)$.

```
  with cut I(0)
    left by proof \tau
    right by proof \epsilon(0);
;
```

This completes the *HandyLK* specification of φ — but to be able to actually compile the proof into an **LKDe**-proof, the subproofs τ, $\varepsilon(0)$ and $\varepsilon(1)$ have to be encoded in *HandyLK*. We continue by showing the *HandyLK* rule applications in τ, which are definition introduction and quantifier rules. The proof τ starts by expanding the two occurrences of the defined predicate I:

$$\dfrac{\dfrac{A \vdash \forall n \exists k f(n+k)=0, \forall n \exists k f(n+k)=1}{A \vdash I(0), \forall n \exists k f(n+k)=1}\ \mathrm{def}_I: r}{A \vdash I(0), I(1)}\ \mathrm{def}_I: r$$

In *HandyLK*, it suffices to specify the predicate to be expanded and give the auxiliary formulas of the rules, the correct main formula is then found by HLK through matching:

```
  with undef I
    :- all n ex k f( n + k ) = 0, all n ex k f( n + k ) = 1;
```

Next, the two strong quantifiers are eliminated by $\forall: r$ rules, introducing the eigenvariables n_0 and n_1:

$$\dfrac{\dfrac{A \vdash \exists k f(n_0+k)=0, \exists k f(n_1+k)=1}{A \vdash \exists k f(n_0+k)=0, \forall n \exists k f(n+k)=1}\ \forall: r}{A \vdash \forall n \exists k f(n+k)=0, \forall n \exists k f(n+k)=1}\ \forall: r$$

The corresponding *HandyLK* code is

```
  with all right
    :- ex k f( n_0 + k ) = 0, ex k f( n_1 + k ) = 1;
```

Again, it suffices to specify the auxiliary formulas, the correct main formulas are found by matching. The same holds for the weak quantifier rules. In our example, we instantiate them by n_1 and n_0, respectively:

$$\dfrac{\dfrac{A \vdash f(n_0+n_1)=0, f(n_1+n_0)=1}{A \vdash \exists k f(n_0+k)=0, f(n_1+n_0)=1}\ \exists: r}{A \vdash \exists k f(n_0+k)=0, \exists k f(n_1+k)=1}\ \exists: r$$

```
  with ex right
    :- f( n_0 + n_1 ) = 0, f( n_1 + n_0 ) = 1;
```

Of course, HLK checks whether the quantifier rule applications are legal. Our proof τ is nearly done — it remains to decompose our assumption A and to use the commutativity of $+$. We only show the latter, which is encoded with an application of paramodulation:

$$\dfrac{\vdash n_0+n_1 = n_1+n_0 \quad f(n_1+n_0)=1 \vdash f(n_1+n_0)=1}{f(n_0+n_1)=1 \vdash f(n_1+n_0)=1}\ =: l_2$$

Such a paramodulation application is common when formalizing proofs from mathematics: an equality axiom from the background theory is used as the active equation. For this reason, it receives special treatment in *HandyLK*: the active equation is simply specified in a by clause.

```
  with paramod by n_0 + n_1 = n_1 + n_0
    right f( n_1 + n_0 ) = 1 :- ;
```

This rule application can also be encoded in the general syntax for binary rules, using a proof reference:

```
with paramod
  f( n_1 + n_0 ) = 1 :-
  left explicit axiom :- n_0 + n_1 = n_1 + n_0 ;
```

Recall that the definition of φ in *HandyLK* contained references to proofs $\varepsilon(0)$ and $\varepsilon(1)$, which show that under the assumption that there are infinitely many cells labelled 0 and 1, respectively, it follows that two cells are labelled by the same number. Clearly, these proofs have a similar structure — this fact can be exploited in *HandyLK* by writing a meta proof that can be instantiated with some specific terms:

```
define proof \epsilon
  with meta term i of type nat;

  proves
    I(i) :- ex p ex q ( not p = q and f(p) = f(q) );
```

We omit the details of the specification of the meta proof $\varepsilon(i)$. Additionally, *HandyLK* supports the definition of proofs in a recursive way, which is a convenient way to encode sequences of proofs. This feature was used to encode a sequence of proofs for the formula scheme from [3].

3.2 XML format for proofs

The programs in the CERES system do not work directly with proofs written in *HandyLK* — these proofs are compiled into a flexible XML format using the HLK compiler. XML is a well known data representation language which allows the use of arbitrary and well known utilities for editing, transformation and presentation and standardized programming libraries. We are interested in tree-style proofs, formulas and terms. Data is structured in trees in XML, therefore the encoding and decoding of these objects is very straightforward.

The format is flexible in the sense that it only assumes that proofs are trees or directed acyclic graphs of rule applications that are labelled by sequents. In our practice, it has been used successfully to encode **LKDe** and resolution proofs. The following abbreviated XML code represents the proof φ:

```
<proof symbol="the-proof" calculus="LK">
 <rule symbol="c:r" type="contrr" param="2">
  <sequent>...</sequent>
  <rule symbol="cut" type="cut">
   <sequent>...</sequent>
   <rule symbol="\pi:r" type="permr" param="(1 2)">
    <sequent>...</sequent>
    <rule symbol="cut" type="cut">
     <sequent>...</sequent>
     <prooflink symbol="\tau"/>
     <prooflink symbol="\epsilon(0)"/>
    </rule>
   </rule>
   <prooflink symbol="\epsilon(1)"/>
  </rule>
 </rule>
</proof>
```

The *symbol* attributes are used for the visual presentation of the proof trees: they allow the user to identify proofs and rules. The *param* attributes contain additional rule information, in the example they specify which formulas are to be contracted (by the contraction rule `c:r`) and how the formulas are to be permuted (by the permutation rule `\pi:r`). The `prooflink` tags reference other proofs by their name and allow the representation of proofs as DAGs.

In the above example, the formulas contained in the sequents have been left out. To give an example of how a formula is encoded in the XML format, consider the following XML code corresponding to the formula $R(c, f(c)) \wedge P(c)$:

```
<conjunctiveformula type="and">
 <constantatomformula symbol="R">
  <constant symbol="c"/>
  <function symbol="f">
   <constant symbol="c"/>
  </function>
 </constantatomformula>
 <constantatomformula symbol="P">
  <constant symbol="c"/>
 </constantatomformula>
</conjunctiveformula>
```

Data is organized in our XML format in a proofdatabase containing a number of proofs. The proof-database may also contain

1. a list of defined predicates and their definitions,
2. a list of sequents specifying the axioms of the background theory in which the proofs are written,
3. arbitrary lists of sequents identified by some symbol (e.g. to store a Herbrand sequent).

3.3 The CERES method

To prepare for the following section, we will now introduce the theoretical background for the CERES method of cut-elimination, which is used by the CERES system to compute a proof in atomic cut normal form and, in the end, a Herbrand sequent.

The central idea of CERES consists in extracting a so-called characteristic clause set from a proof, and then using a resolution refutation of this set to obtain a proof with only atomic cuts. We consider the proofs in **LKDe** as directed trees with nodes which are labelled by sequents, where the root is labelled by the end-sequent. According to the inference rules, we distinguish binary and unary nodes. In an inference

$$\frac{\nu_1 : S_1 \quad \nu_2 : S_2}{\nu : S}\ x$$

where ν is labelled by S, ν_1 by S_1 and ν_2 by S_2, we call ν_1, ν_2 *predecessors* of ν. Similarly ν' is predecessor of ν in a unary rule if ν' labels the premiss and ν the consequent. Then the *predecessor relation* is defined as the reflexive and transitive closure of the relation above. Every node is predecessor of the root, and the axioms have only themselves as predecessors. For a formal definition of the concepts we refer to [4] and [5]. A similar relation holds between *formula occurrences* in sequents. Instead of a formal definition we give an example.

Consider the rule:

$$\frac{\forall x.P(x) \vdash P(a) \quad \forall x.P(x) \vdash P(b)}{\forall x.P(x) \vdash P(a) \wedge P(b)}\ \wedge : r$$

The occurrences of $P(a)$ and $P(b)$ in the premiss are *ancestors* of the occurrence of $P(a) \land P(b)$ in the consequent. $P(a)$ and $P(b)$ are called *auxiliary formulas* of the inference, and $P(a) \land P(b)$ the *main formula*. $\forall x.P(x)$ in the premisses are ancestors of $\forall x.P(x)$ in the consequent. Again the *ancestor relation* is defined by reflexive transitive closure.

Let Ω be the set of all occurrences of cut-formulas in sequents of an **LKDe**-proof φ. The cut-formulas are not ancestors of the formulas in the end-sequent, but they might have ancestors in the axioms (if the cuts are not generated by weakening only). The construction of the characteristic clause set is based on the ancestors of the cuts in the axioms. Note that *clauses* are just defined as atomic sequents. We define a set of clauses $\mathscr{C}_\nu$ for every node ν in φ inductively:

- If ν is an occurrence of an axiom sequent $S(\nu)$, and S' is the subsequent of $S(\nu)$ containing only the ancestors of Ω then $\mathscr{C}_\nu = \{S'\}$.
- Let ν' be the predecessor of ν in a unary inference then $\mathscr{C}_\nu = \mathscr{C}_{\nu'}$.
- Let ν_1, ν_2 be the predecessors of ν in a binary inference. We distinguish two cases

  (a) The auxiliary formulas of ν_1, ν_2 are ancestors of Ω. Then

  $$\mathscr{C}_\nu = \mathscr{C}_{\nu_1} \cup \mathscr{C}_{\nu_2}.$$

  (b) The auxiliary formulas of ν_1, ν_2 are not ancestors of Ω. Then

  $$\mathscr{C}_\nu = \mathscr{C}_{\nu_1} \times \mathscr{C}_{\nu_2}.$$

  where $\mathscr{C} \times \mathscr{D} = \{C \circ D \mid C \in \mathscr{C}, D \in \mathscr{D}\}$ and $C \circ D$ is the merge of the clauses C and D.

The *characteristic clause set* $\mathrm{CL}(\varphi)$ of φ is defined as $\mathscr{C}_{\nu_0}$, where ν_0 is the root.

Theorem 1. *Let φ be a proof in* **LKDe**. *Then the clause set* $\mathrm{CL}(\varphi)$ *is equationally unsatisfiable.*

Remark. A clause set $\mathscr{C}$ is equationally unsatisfiable if $\mathscr{C}$ does not have a model where $=$ is interpreted as equality over a domain.

Proof. This proof first appeared in [1]. Let ν be a node in φ and $S'(\nu)$ the subsequent of $S(\nu)$ which consists of the ancestors of Ω (i.e. of a cut). It is shown by induction that $S'(\nu)$ is **LKDe**-derivable from $\mathscr{C}_\nu$. If ν_0 is the root then, clearly, $S'(\nu_0) = \vdash$ and the empty sequent $\vdash$ is **LKDe**-derivable from the axiom set $\mathscr{C}_{\nu_0}$, which is just $\mathrm{CL}(\varphi)$. As all inferences in **LKDe** are sound over equational interpretations (where new symbols introduced by definition introduction have to be interpreted according to the defining equivalence), $\mathrm{CL}(\varphi)$ is equationally unsatisfiable. Note that, without the rules $=: l$ and $=: r$, the set $\mathrm{CL}(\varphi)$ is just unsatisfiable. Clearly the rules $=: l$ and $=: r$ are sound only over equational interpretations. □

The next steps in CERES are

(1) the computation of the proof projections $\varphi[C]$ w.r.t. clauses $C \in \mathrm{CL}(\varphi)$,

(2) the refutation of the set $\mathrm{CL}(\varphi)$, resulting in an RP-tree γ, i.e. in a deduction tree defined by the inferences of resolution and paramodulation, and

(3) "inserting" the projections $\varphi[C]$ into the leaves of γ.

For step (1) we skip in φ all inferences where the auxiliary resp. main formulas are ancestors of a cut. Instead of the end-sequent S we get $S \circ C$ for a $C \in \mathrm{CL}(\varphi)$.

Step (2) consists in ordinary theorem proving by resolution and paramodulation (which is equationally complete). For refuting $\mathrm{CL}(\varphi)$ any first-order resolution prover can be used. By the completeness of the methods we find a refutation tree γ as $\mathrm{CL}(\varphi)$ is unsatisfiable by Theorem 1.

Step (3) makes use of the fact that, after computation of the simultaneous most general unifier of the inferences in γ, the resulting tree γ' is actually a derivation in **LKDe**. Indeed, after computation of the simultaneous unifier, paramodulation becomes $=: l$ and $=: r$ and resolution becomes cut in **LKDe**. Note that the definition rules, like the logical rules, do not appear in γ'. Now for every leaf ν in γ', which is labelled by a clause C' (an instance of a clause $C \in \mathrm{CL}(\varphi)$) we insert the proof projection $\varphi[C']$. The result is a proof with only atomic cuts.

The proof projection is only sound if the proof φ is skolemized, i.e. there are no strong quantifiers (i.e. quantifiers with eigenvariable conditions) in the end-sequent. If φ is not skolemized a priori it can be transformed into a skolemized proof φ' in polynomial (at most quadratic) time; for details see [3].

For illustration, consider the following example:

$\varphi =$

$$\dfrac{\varphi_1 \qquad \varphi_2}{(\forall x)(P(x) \to Q(x)) \vdash (\exists y)(P(a) \to Q(y))}\ cut$$

$\varphi_1 =$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(u)^\star \vdash P(u) \quad Q(u) \vdash Q(u)^\star}{P(u)^\star, P(u) \to Q(u) \vdash Q(u)^\star}\ \to: l}{P(u) \to Q(u) \vdash (P(u) \to Q(u))^\star}\ \to: r}{P(u) \to Q(u) \vdash (\exists y)(P(u) \to Q(y))^\star}\ \exists: r}{(\forall x)(P(x) \to Q(x)) \vdash (\exists y)(P(u) \to Q(y))^\star}\ \forall: l}{(\forall x)(P(x) \to Q(x)) \vdash (\forall x)(\exists y)(P(x) \to Q(y))^\star}\ \forall: r$$

$\varphi_2 =$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(a) \vdash P(a)^\star \quad Q(v)^\star \vdash Q(v)}{P(a), (P(a) \to Q(v))^\star \vdash Q(v)}\ \to: l}{(P(a) \to Q(v))^\star \vdash P(a) \to Q(v)}\ \to: r}{(P(a) \to Q(v))^\star \vdash (\exists y)(P(a) \to Q(y))}\ \exists: r}{(\exists y)(P(a) \to Q(y))^\star \vdash (\exists y)(P(a) \to Q(y))}\ \exists: l}{(\forall x)(\exists y)(P(x) \to Q(y))^\star \vdash (\exists y)(P(a) \to Q(y))}\ \forall: l$$

We have $\mathrm{CL}(\varphi) = \{P(u) \vdash Q(u);\ \vdash P(a);\ Q(v) \vdash\}$. The resolution refutation δ of $\mathrm{CL}(\varphi)$

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

does the job of refuting $\mathrm{CL}(\varphi)$. By applying the most general unifier σ of δ, we obtain a ground refutation $\gamma = \delta\sigma$:

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

This will serve as a skeleton for a proof in ACNF. To complete the construction, we combine the skeleton with the following projections (grounded by σ):

$\varphi(C_1) =$

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

$\varphi(C_2) =$

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

$\varphi(C_3) =$

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

The composition of skeleton and projections yields
$\varphi(\gamma) =$

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\varphi(C_2)\\ B \vdash C, P(a)\end{array} \quad \begin{array}{c}\varphi(C_1)\\ P(a), B \vdash C, Q(a)\end{array}}{B, B \vdash C, C, Q(a)} cut \quad \begin{array}{c}\varphi(C_3)\\ Q(a), B \vdash C\end{array}}{B, B, B \vdash C, C, C} cut}{B \vdash C} \text{contractions}$$

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.

3.4 Cut-elimination, proof visualization and Herbrand sequent extraction with CERES

After compiling the *HandyLK* source to XML with HLK, it is possible to perform proof analysis using the CERES tool. In our example, we are interested in extracting a Herbrand sequent from a cut-free proof of the end-sequent of φ. To this end, the CERES method is used, which extracts an unsatisfiable set of clauses from the proof and refutes it. In theory, any resolution prover can be used to refute this set of clauses; Table 1 lists the format conversions currently supported by our implementation.

Table 1: Format conversions supported by CERES

From	**To**
XML	TPTP
XML	Otter input
XML	Prover9 input
Otter proof object	XML
Prover9 output	XML

In our example, we use Otter to find a resolution refutation. Running CERES yields the following characteristic clause set in Otter input format

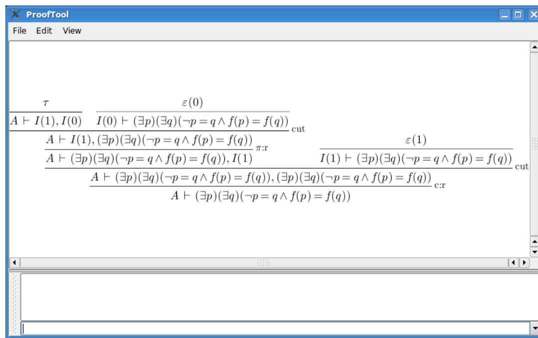
```
-> =("f"(+(x4, x5)), "0"), =("f"(+(x5, x4)), "1"). % (C1)
=("f"(+(x4, x6)), "0"), =("f"(+(+(+(x4, x6), "1"), x7)), "0") ->. % (C2)
=("f"(+(x4, x6)), "1"), =("f"(+(+(+(x4, x6), "1"), x7)), "1") ->. % (C3)
```

which corresponds to the set of clauses

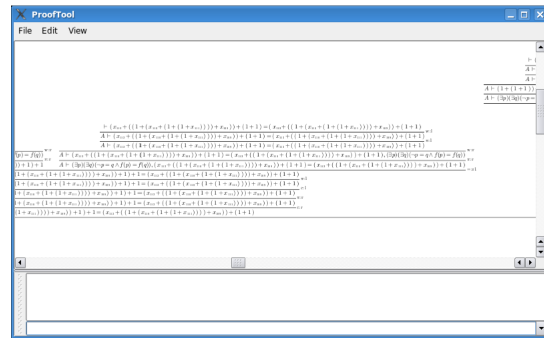
$$\begin{aligned}\mathrm{CL}(\pi) = \{ \; & \vdash f(x_4+x_5)=0, f(x_5+x_4)=1; \\ & f(x_4+x_6)=0, f(((x_4+x_6)+1)+x_7)=0 \vdash; \\ & f(x_4+x_6)=1, f(((x_4+x_6)+1)+x_7)=1 \vdash \quad \}\end{aligned}$$

which is refutable in arithmetic.

The Otter proof object resulting from the successful refutation is then converted into a resolution tree in XML format. According to the CERES method, this refutation is then transformed into an **LKDe** proof φ' of the end-sequent of φ in ACNF. φ' consists of 1262 rules, and is clearly too large to be depicted here (a small part of it can be seen in Figure 2(b)). From φ', a Herbrand sequent is extracted and stored in XML as a list of sequents (containing only the Herbrand sequent). All proofs and sequents computed during the analysis can be visualized using `ProofTool` as seen in Figure 2.



(a) Visualizing the input proof φ.



(b) A bird's eye view on the output proof φ'.

Figure 2: `ProofTool` screenshots

To make the Herbrand sequent extracted in our example easier to read, the terms have been simplified modulo commutativity and associativity of $+$, and the following abbreviations are used:

$$\begin{array}{llll} p_1 = x+z+2, & p_2 = x+z+4, & p_3 = z+1, & p_4 = x+y+z+3, \\ p_5 = w+x+y+z+6, & p_6 = w+x+y+z+8, & p_7 = w+x+y+z+4, & \end{array}$$

With this, the Herbrand sequent of the ACNF of the tape proof is

$$\begin{array}{l} f(p_1)=0 \vee f(p_1)=1, f(p_2)=0 \vee f(p_2)=1, f(p_3)=0 \vee f(p_3)=1, \\ f(p_4)=0 \vee f(p_4)=1, f(p_5)=0 \vee f(p_5)=1, f(p_6)=0 \vee f(p_6)=1, \\ f(p_7)=0 \vee f(p_7)=1 \\ \vdash \\ p_1 \neq p_2 \wedge f(p_1)=f(p_2), p_3 \neq p_1 \wedge f(p_3)=f(p_1), \\ p_3 \neq p_2 \wedge f(p_3)=f(p_2), p_1 \neq p_4 \wedge f(p_1)=f(p_4), \\ p_5 \neq p_6 \wedge f(p_5)=f(p_6), p_7 \neq p_5 \wedge f(p_7)=f(p_5), \\ p_7 \neq p_6 \wedge f(p_7)=f(p_6), p_4 \neq p_7 \wedge f(p_4)=f(p_7). \end{array}$$

Clearly, the Herbrand sequent is much smaller than the ACNF from which it is extracted. As it contains all quantifier instantiation information from the proof, it is much better suited to human interpretation than the proof itself. The Herbrand sequent we extracted from φ' can be interpreted as the following proof:

Theorem 2. *On a tape with infinitely many cells where each cell is labelled* 0 *or* 1*, there are two distinct cells that are labelled the same.*

Proof. It is easy to see that the following inequalities hold:

$$p_1 \neq p_2, \quad p_3 \neq p_1, \quad p_3 \neq p_2, \quad p_1 \neq p_4,$$
$$p_5 \neq p_6, \quad p_7 \neq p_5, \quad p_7 \neq p_6, \quad p_4 \neq p_7.$$

We may therefore delete their occurrences from the right side of the Herbrand sequent and obtain the simplified sequent

$$\begin{array}{l} f(p_1) = 0 \vee f(p_1) = 1, f(p_2) = 0 \vee f(p_2) = 1, f(p_3) = 0 \vee f(p_3) = 1, \\ f(p_4) = 0 \vee f(p_4) = 1, f(p_5) = 0 \vee f(p_5) = 1, f(p_6) = 0 \vee f(p_6) = 1, \\ f(p_7) = 0 \vee f(p_7) = 1 \\ \vdash \\ f(p_1) = f(p_2), f(p_3) = f(p_1), f(p_3) = f(p_2), f(p_1) = f(p_4), \\ f(p_5) = f(p_6), f(p_7) = f(p_5), f(p_7) = f(p_6), f(p_4) = f(p_7). \end{array}$$

Now, we assume that this sequent is false and obtain a contradiction. If the sequent is false, then all formulas on the left side are true and all formulas on the right side are false, so we may assume $f(p_i) = 0 \vee f(p_i) = 1$ for $i \in \{1, \ldots, 7\}$. Let $f(p_1) = a$ with $a \in \{0,1\}$ and let $\bar{a} = 1 - a$. We have assumed that the first formula on the right side is false, so $f(p_2) = \bar{a}$. From the second formula we obtain $f(p_3) = \bar{a}$, and from the third $f(p_3) = \bar{\bar{a}} = a$, which yields the desired contradiction. □

Note that the proof extracted from the Herbrand sequent uses only 3 cells even though the sequent itself is not minimal. Still, the proof yields a minimal Herbrand sequent directly:

$$\begin{array}{l} f(p_1) = 0 \vee f(p_1) = 1, f(p_2) = 0 \vee f(p_2) = 1, f(p_3) = 0 \vee f(p_3) = 1, \\ \vdash \\ p_1 \neq p_2 \wedge f(p_1) = f(p_2), p_3 \neq p_1 \wedge f(p_3) = f(p_1), p_3 \neq p_2 \wedge f(p_3) = f(p_2). \end{array}$$

Comparing the proof obtained from the Herbrand sequent with the original proof, we have gained an important piece of information: in the original proof, it is shown that either infinitely many cells are labelled 0 or 1, while in the cut-free proof, only finitely many cells are used in the proof.

4 Open problems and future work

On the theoretical side we are working on the following two major extensions of the CERES-method: The first is an extension of CERES to second-order logic, a first step in that direction has been achieved by extending it to the fragment of second-order proofs defined by containing only quantifier-free comprehension in [9]. This extension will allow the formalization of a much broader range of mathematical proofs and yield a more convenient proof formalization. The second theoretical extension enables the method to work without skolemization whose main benefit will be the ability to eliminate single cuts or even parts of a cut which is more realistic in mathematical applications than the simultaneous elimination of all cuts.

Concerning the practical aspects, it became clear in our experiments with the system that there are two bottlenecks: 1. the formalization of the input proof and 2. finding a resolution refutation of the characteristic clause set. Accordingly also these two points are of major concern for our future work.

To ease the formalization of the input proof, we plan to enhance the capabilities of `HLK`, in particular those of the `auto propositional`-mode to cover also reasoning in equational background theories

specified by term rewriting systems. We expect this feature to greatly reduce the amount of time spent on proof formalization, as large parts of formalized proofs consist of equational reasoning. In the long term, however, an interesting option would be to use one of the available proof assistants for the formalization of proofs as these are highly developed tools. The theoretical obstacle to the use of existing proof assistants lies in the fact that a translation of the formalized proof is necessary, as proof assistants typically work in set theory or higher-order logic. Moreover, this translation should not be uniform because, depending on the formalized proof and the aims of the proof analysis, one logical translation will be more useful than another. The main practical obstacle lies in transforming proof objects generated by the proof assistant (if they are provided at all) into the sequent calculus format as described in Section 3.2. Here, a lot of work will have to go into importing the standard library of basic number systems, basic operations, data structures, etc.

The search for a refutation of the characteristic clause set turned out to be a hard problem for current theorem provers (as described in [2]). To handle this problem, we need automated theorem provers with high flexibility, allowing an interactive construction of the resolution refutation. For using several provers in order to combine their respective advantages it would be very beneficial to have a standard output format for encoding resolution refutations, a goal which is partially realized by the TPTP output format. Another approach to enhance the power of the provers is to exploit the fact that the characteristic clause sets of the CERES-method form a very specific subclass of theorem proving problems. In particular, it seems promising to store in the clause set certain information about the structure of the original proof as hints on how to find a refutation and to develop resolution refinements for CERES to use this additional information.

Another important area for future improvement is the human-readable representation of the output proofs as well as other results of the analysis (e.g. Herbrand sequents); this aspect is also critical for the method to adapt better to larger proofs. As the terms generated by cut-elimination are frequently very large it would pay out to implement term simplifiers. A general approach to the simplification of Herbrand sequents lies in computing a minimal variant of it by most general unification and a (possibly external) tautology checker. For specific theories, e.g. arithmetic, one can even do better by using term simplification algorithms based on term rewriting systems. Another useful addition is to include a flexible handling of definitions in the extraction and display of Herbrand sequents which would make them an even more powerful tool for understanding the mathematical content of a formal proof. These changes will further contribute to the creation of an interface for proof analysis which is attractive to logicians and mathematicians alike.

References

[1] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. Proof transformation by CERES. In *Lecture Notes in Artificial Intelligence*, volume 4108, pages 82–93. Mathematical Knowledge Management, Springer Berlin, 2006.

[2] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. Ceres: An Analysis of Fürstenberg's Proof of the Infinity of Primes. *Theoretical Computer Science*, 403:160–175, August 2008.

[3] Matthias Baaz and Alexander Leitsch. Cut normal forms and proof complexity. *Annals of Pure and Applied Logic*, 97(1–3):127–177, 1999.

[4] Matthias Baaz and Alexander Leitsch. Cut-elimination and Redundancy-elimination by Resolution. *Journal of Symbolic Computation*, 29(2):149–176, 2000.

[5] Matthias Baaz and Alexander Leitsch. Towards a clausal analysis of cut-elimination. *Journal of Symbolic Computation*, 41:381–410, 2006.

[6] G. Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210,405–431, 1934–1935.

[7] Jean-Yves Girard. *Proof Theory and Logical Complexity*. Elsevier, 1987.

[8] Stefan Hetzl, Alexander Leitsch, Daniel Weller, and Bruno Woltzenlogel Paleo. Herbrand sequent extraction. In *Lecture Notes in Artificial Intelligence*, volume 5144, pages 462–477. Mathematical Knowledge Management, Springer Berlin, 2008.

[9] Stefan Hetzl, Alexander Leitsch, Daniel Weller, and Bruno Woltzenlogel Paleo. A Clausal Approach to Proof Analysis in Second-Order Logic. In *Symposium on Logical Foundations of Computer Science (LFCS 2009)*, Lecture Notes in Computer Science. Springer, 2009. to appear.

[10] Alexander Leitsch and Clemens Richter. Equational theories in ceres. unpublished (available at `http://www.logic.at/ceres/`), 2005.

[11] The Coq development team. *The Coq proof assistant reference manual*. LogiCal Project, 2004. Version 8.0.

[12] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — a proof assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer Berlin, 2002.

[13] C. Urban. *Classical Logic and Computation*. PhD thesis, University of Cambridge Computer Laboratory, 2000.