Algorithmic structuring and compression of proofs

Stefan Hetzl
hetzl@logic.at

Vienna University of Technology

Introduction

Computer-generated proofs are typically analytic: they consist only of formulas which are present in the theorem that is shown.
In contrast, mathematical proofs written by humans almost never are: they are highly structured due to the use of lemmas.
This project aims at developing algorithms and software which structure and abbreviate analytic proofs by computing useful lemmas.
We employ proof-theoretic methods, in particular a recently discovered connection between proof theory and formal language theory.

Aims

Computer-generated proofs are often hard to understand, among other reasons, because they are analytic.
However, even inscrutable analytic proofs *do* carry mathematical knowledge, after all they show that a theorem is true. But they carry this knowledge only in an implicit form which renders it inaccessible to a human reader.

```
390 x + (0 * x) = x.
[para(207(a,1),11(a,1,2)),
rewrite([389(8),221(5),5(3),
18(3),10(2),11(2)]),flip(a)].
```

Fig. 1. Line in ATP output

We want to make this mathematical knowledge explicit by **structuring** and **compression** of proofs. To that aim we generate lemmas which allow a new – more readable – proof of the same theorem.

$$\frac{A \vdash \exists y\,(x \cdot x) \cdot y = y \cdot (x \cdot x) \qquad \ldots \vdash F}{A \vdash F}$$

Fig. 2. Lemma in the sequent calculus

The intended application of our algorithms is as post-processing of automated deduction systems in order to obtain more meaningful proof output.

Methods – Overview

For the generation of new lemmas we strongly rely on theoretical notions and results: the sequent calculus, cut-elimination, Herbrand's theorem, representation of proofs by Herbrand-disjunctions, etc. Of particular importance is a recently discovered **connection between proof theory and formal language theory** [1].

Proof Theory and Formal Language Theory

An analytic (cut-free) proof π can be represented by its **Herbrand-disjunction** $\mathrm{H}(\pi)$ consisting of the information which instances are picked for which quantifiers.
This form of representation has been extended to proofs with cuts (lemmas) of the form $\exists x\, A$ or $\forall x\, A$ where A is quantifier-free [1, 3] using **totally rigid acyclic tree grammars**. Rigid tree languages have been introduced in [4], originally with applications in verification in mind.

$$\begin{array}{ccc} \text{proof } \pi \text{ with cuts} & \xrightarrow{\text{cut-elimination}} & \text{cut-free proof } \pi^* \\ \downarrow & & \downarrow \\ \text{grammar } \mathrm{G}(\pi) & \xrightarrow{\text{defines}} & \mathrm{L}(\mathrm{G}(\pi)) = \mathrm{H}(\pi^*) \end{array}$$

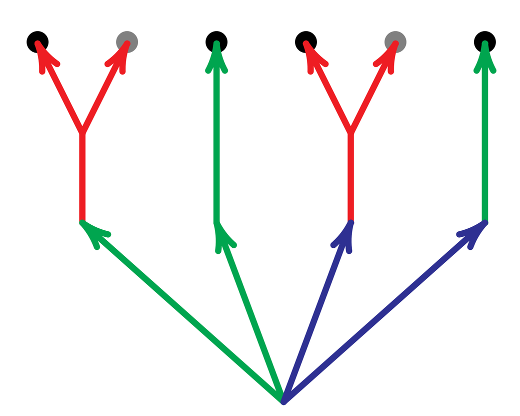
Fig. 3. Proofs and Grammars

On the right level of abstraction, cut-elimination is the process of computing the language of the grammar.

Algorithmic Cut-Introduction

The close relationship between proofs and grammars allows to **reverse cut-elimination** as follows:

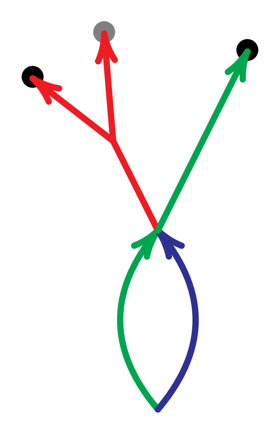
1. Given a cut-free proof, carry out a **structural analysis** of the term set H of its Herbrand-disjunction.



$g(r(c,d))$
$g(g(c))$
$b(r(c,d))$
$b(g(c))$

Fig. 4. A set of terms H

By identification of appropriate regularities, write the terms of the Herbrand-disjunction as a grammar.



$\tau \to g(\alpha), b(\alpha)$
$\alpha \to r(c,d), g(c)$

Fig. 5. A tree grammar G with $\mathrm{L}(G) = H$

2. **Generate a proof** with cut that realises this grammar – this is always possible, the cut-formulas are induced by the structure of the grammar. Apply simplifications as post-processing.

A proof-of-concept algorithm based on this approach is described in [2].

Implementation

Based on:

GAPT
Generic Architecture for Proof Theory
http://code.google.com/p/gapt/

- Supports first-order logic, higher-order logic
- Standard data structures and algorithms from proof theory
- Resolution Prover
- Cut-Elimination
- Graphical User Interface
- Command-Line Interface
- ...
- Implemented in Scala
- GNU GPL

Outlook

- Develop full implementation
- Large-scale tests (TPTP, SMT-LIB)
- Extension of theoretical basis
- Cover larger classes of proofs with cut
- Work modulo theories
- From lemma generation to invariant generation for inductive theorem proving

References

[1] S. Hetzl. Applying Tree Languages in Proof Theory. In A.-H. Dediu and C. Martín-Vide, editors, *Language and Automata Theory and Applications (LATA) 2012*, volume 7183 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2012.

[2] S. Hetzl, A. Leitsch, and D. Weller. Towards Algorithmic Cut-Introduction. In *Logic for Programming, Artificial Intelligence and Reasoning (LPAR-18)*, volume 7180 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 2012.

[3] S. Hetzl and L. Straßburger. Herbrand-Confluence for Cut-Elimination in Classical First-Order Logic. In *Computer Science Logic (CSL) 2012*. to appear.

[4] F. Jacquemard, F. Klay, and C. Vacher. Rigid tree automata and applications. *Information and Computation*, 209:486–512, 2011.