Algorithmic Introduction of Quantified Cuts[☆]

Stefan Hetzl[a], Alexander Leitsch[b], Giselle Reis[b], Daniel Weller[a]

[a]*Institute of Discrete Mathematics and Geometry, Vienna University of Technology, Vienna, Austria*
[b]*Institute of Computer Languages, Vienna University of Technology, Vienna, Austria*

Abstract

We describe a method for inverting Gentzen's cut-elimination in classical first-order logic. Our algorithm is based on first computing a compressed representation of the terms present in the cut-free proof and then cut-formulas that realize such a compression. Finally, a proof using these cut-formulas is constructed. This method allows an exponential compression of proof length. It can be applied to the output of automated theorem provers, which typically produce analytic proofs. An implementation is available on the web and described in this paper.

1. Introduction

Cut-elimination introduced by Gentzen [16] 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), resulting in a proof which is analytic in the sense that all statements in the proof are subformulas of the result. Thus a proof of a combinatorial statement is converted into a purely combinatorial proof. Cut-elimination is therefore an essential tool for the analysis of proofs, especially to make implicit parameters explicit.

In this paper we present a method for inverting Gentzen's cut-elimination by computing a proof with cut from a given cut-free proof as input. As cut-elimination is the backbone of proof theory, there is considerable proof-theoretic interest and challenge in understanding this transformation sufficiently well to be able to invert it. But our interest in cut-introduction is not only of a purely theoretical nature. Proofs with cuts have properties that are essential for applications: one the one hand, cuts are indispensable for formalizing proofs in a human-readable way. One the other hand cuts have a very strong compression power in terms of proof length.

Computer-generated proofs are typically analytic, i.e. they only contain logical material that also appears in the theorem shown. This is due to the fact that analytic proof systems have a considerably smaller search space which makes proof-search practically feasible. In the case of the sequent calculus, proof-search procedures typically work on the cut-free fragment. But also resolution is essentially analytic as resolution proofs satisfy the subformula property of first-order logic. An important property of non-analytic proofs is their considerably smaller length. The exact difference depends on the logic (or theory) under consideration, but it is typically enormous. In (classical and intuitionistic) first-order logic there are proofs with cut of length n whose theorems have only cut-free proofs of length 2_n (where $2_0 = 1$ and $2_{n+1} = 2^{2_n}$) (see [36] and [31]). The length of a proof plays an important role in many situations such as human readability, space requirements and time requirements for proof checking. For most of these situations general-purpose data compression methods cannot be used as the compressed representation is not a proof anymore. It is therefore of high practical interest to develop proof-search methods which produce non-analytic and hence

[☆]This work was supported by the projects P22028-N13, I603-N18 and P25160-N25 of the Austrian Science Fund (FWF), by the ERC Advanced Grant ProofCert and the WWTF Vienna Research Group 12-04.

Email addresses: stefan.hetzl@tuwien.ac.at (Stefan Hetzl), leitsch@logic.at (Alexander Leitsch), giselle@logic.at (Giselle Reis), weller@logic.at (Daniel Weller)

potentially much shorter proofs. In the method presented in this paper we start with a cut-free proof and abbreviate it by computing useful cuts based on a structural analysis of the cut-free proof.

There is another, more theoretical, motivation for introducing cuts which derives from the foundations of mathematics: most of the central mathematical notions have developed from the observation that many proofs share common structures and steps of reasoning. Encapsulating those leads to a new abstract notion, like that of a group or a vector space. Such a notion then builds the base for a whole new theory whose importance stems from the pervasiveness of its basic notions in mathematics. From a logical point of view this corresponds to the introduction of cuts into an existing proof database. While we cannot claim to contribute much to the understanding of such complex historical processes by the current technical state of the art, this second motivation is still worthwhile to keep in mind, if only to remind us that we are dealing with a difficult problem here.

Gentzen's method of cut-elimination is based on reductions of cut-derivations (subproofs ending in a cut), transforming them into simpler ones; basically the cut is replaced by one or more cuts with lower logical complexity. A naive reversal of this procedure is infeasible as it would lead to a search tree which is exponentially branching on some nodes and infinitely branching on others. Therefore we base our procedure on a deeper proof-theoretic analysis: in the construction of a Herbrand sequent S' corresponding to a cut-free proof φ' (see e.g. [3]) obtained by cut-elimination on a proof φ of a sequent S with cuts, only the *substitutions* generated by cut-elimination on quantified cuts are relevant. In fact, it is shown in [19] that, for proofs with Σ_1 and Π_1-cuts only, S' can be obtained just by computing the substitutions defined by cut-elimination without applying Gentzen's procedure as a whole. Via the cuts in the proof φ one can define a tree grammar generating a language consisting exactly of the terms (to be instantiated for quantified variables in S) for obtaining the Herbrand sequent S' [19]. Hence, generating a tree grammar G from a set of Herbrand terms T (generating T) corresponds to an inversion of the quantifier part of Gentzen's procedure. The computation of such an inversion forms the basis of the method of *cut-introduction* presented in this paper. Such an inversion of the quantifier part of cut-elimination determines which instances of the cut-formulas are used but it does not determine the cut-formulas. In fact, a priori it is not clear that every such grammar can be realized by actual cut-formulas. However, we could show that, for any such tree grammar representing the quantifier part of potential cut-formulas, actual cut-formulas can be constructed. Finally, a proof containing these cut-formulas can be constructed.

Work on cut-introduction can be found at a number of different places in the literature. Closest to our work are other approaches which aim to abbreviate or structure *a given input proof*: [41] is an algorithm for the introduction of atomic cuts that is capable of exponential proof compression. The method [14] for propositional logic is shown to never increase the size of proofs more than polynomially. Another approach to the compression of first-order proofs by introduction of definitions for abbreviating terms is [40].

Viewed from a broader perspective, this paper should be considered part of a large body of work on the generation of non-analytic formulas that has been carried out by numerous researchers in various communities. Methods for lemma generation are of crucial importance in inductive theorem proving which frequently requires generalization [7], see e.g. [24] for a method in the context of rippling [8] which is based on failed proof attempts. In automated theory formation [9, 10], an eager approach to lemma generation is adopted. This work has, for example, led to automated classification results of isomorphism classes [34] and isotopy classes [35] in finite algebra. See also [27] for an approach to inductive theory formation. In pure proof theory, an important related topic is Kreisel's conjecture (see footnote 3 on page 400 of [38]) on the generalization of proofs. Based on methods developed in this tradition, [4] describes an approach to cut-introduction by filling a proof skeleton, i.e. an abstract proof structure, obtained by an inversion of Gentzen's procedure with formulas in order to obtain a proof with cuts. The use of cuts for structuring and abbreviating proofs is also of relevance in logic programming: [30] shows how to use focusing in order to avoid proving atomic subgoals twice, resulting in a proof with atomic cuts.

This paper is organized as follows:
In Section 3 we define Herbrand sequents and *extended* Herbrand sequents which represent proofs with cut. The concept of rigid acyclic regular tree grammars is applied to establish a relation between an extended Herbrand sequent S^* and a (corresponding) Herbrand sequent S': the language defined by this grammar

is just the set of terms T to be instantiated for quantifiers in the original sequent S to obtain S'. Given such a grammar G generating T there exists a so-called *schematic extended Herbrand sequent* $\hat{S}$ in which the (unknown) cut-formulas are represented by monadic second-order variables. It is proved that $\hat{S}$ always has a solution, the *canonical solution*. From this solution, which gives an extended Herbrand sequent and the cut-formulas for a proof, the actual proof with these cuts is constructed.

To make the underlying methods more transparent, Section 3 deals only with end-sequents of the form $\forall x\, F$. In Section 4 the method is generalized to sequents of the form

$$\forall \bar{x}_1 F_1, \ldots, \forall \bar{x}_n F_n \to \exists \bar{y}_1 G_1, \ldots, \exists \bar{y}_m G_m$$

where the $\bar{x}_i, \bar{y}_j$ are vectors of variables and $\forall z_1 \cdots z_k$ stands for $\forall z_1 \cdots \forall z_k$. This form of sequents is more useful for practical applications and covers all of first-order logic as an arbitrary sequent can be transformed into one of this form by Skolemization and prenexification. We prove that all results obtained in Section 3 carry over to this more general case.

In Section 5 an algorithm is presented computing a minimal rigid acyclic regular tree grammar generating the Herbrand term set T.

Given a cut-free proof φ and a corresponding Herbrand term set T, the canonical solution corresponding to a non-trivial minimal grammar generating T yields a proof ψ with lower quantifier-complexity (which is the number of quantifier inferences in a proof) than φ, but the length of ψ (the total number of inferences) may be greater than that of φ. In Section 6 a method is presented to overcome this problem. By using a resolution-based method the cut-formulas are simplified under preservation of the quantifier complexity, resulting in proofs with lower number of inferences.

In Section 7 a nondeterministic algorithm CI is defined which is based on the techniques developed in Sections 3 to 6. We show that CI is, in a suitable sense, an inversion of Gentzen's cut-elimination method. A sequence of cut-free proofs is defined and it is proven that the application of CI to this sequence results in an exponential compression of proof length. Finally the existing implementation of CI and some experiments are described in Section 8.

This paper improves the publication [20] in several crucial directions: (1) the method for introducing a single $\forall$-cut is generalized to a method introducing an arbitrary number of $\forall$-cuts, which requires – among others – a length-preserving transformation of extended Herbrand-sequents to proofs with cuts based on Craig interpolation, (2) we show that our method is, in a suitable sense, an inversion of Gentzen's cut-elimination method, (3) the end-sequent may contain blocks of quantifiers instead of just single ones, (4) the decomposition of terms is represented as a problem of grammars and a practical algorithm for computing grammars is developed, (5) it is shown that the proof compression obtained by the new method is exponential while it was only quadratic for the one of [20], and (6) the algorithm has been implemented in the gapt-system[1].

The method CI developed in this paper is a systematic, proof-theoretic method to compress the lengths of first-order proofs by the introduction of cuts. Still, the generated cut-formulas are all universal. A desirable extension of this method to introduce cuts with alternating quantifiers (which is necessary to obtain super-exponential compressions) is left to future work. Such an extension is highly non-trivial as it first requires the development of an adequate notion of tree grammar for extending the underlying proof-theoretic results to cuts with quantifier alternations.

2. A Motivating Example

Consider the sequents

$$S_n \;=\; Pa, \forall x\,(Px \supset Pfx) \to Pf^{2^n}a.$$

The straightforward cut-free proof of S_n in the sequent calculus uses the successor-axiom 2^n times. In fact, it is easy to show that every cut-free proof has to contain all of these 2^n instances. On the other hand, if we

[1] `http://www.logic.at/gapt/`

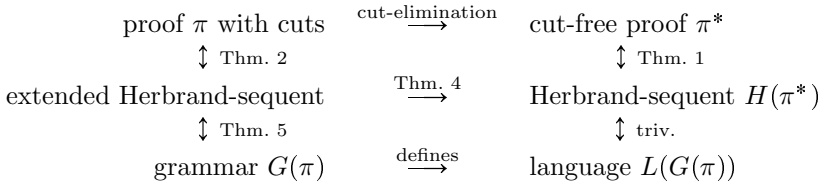

Figure 1: Proof-theoretic setting of this paper

allow the use of cuts, we can give a considerably shorter proof by first showing

$$\forall x\,(Px \supset Pf^2x)$$

from the axiom and then using this formula twice to show

$$\forall x\,(Px \supset Pf^4x)$$

and so on. In general we cut with the constant-length proofs of

$$\forall x\,(Px \supset Pf^{2^i}x) \to \forall x\,(Px \supset P^{2^{i+1}}x)$$

and hence obtain a proof of S_n that uses only $O(n)$ inference steps instead of the $\Omega(2^n)$ of the cut-free proof. Note how the structures of these two proofs are reminiscent of the binary and the unary representation of numbers. This proof sequence is an exponential version of the sequences of Statman [36] and Orevkov [31] and has also been considered by Boolos [6].

In this paper we want to leverage this compression power of lemmas by automatically transforming cut-free proofs into proofs using compressing lemmas.

3. Proof-Theoretic Infrastructure

Gentzen's proof of cut-elimination [16] can be understood as the application of a set of local proof rewriting rules with a terminating strategy. A first naive approach to cut-introduction would be to consider the inversion of these local rewriting steps as a search algorithm. While this procedure would in theory allow to reverse every cut-elimination sequence, it becomes clear quickly that it is not feasible in practice: not only would we have to guess an enormous amount of trivialities (e.g. rule permutations) but the inversion of rewriting rules which erase a part of the proof lead to the necessity of correctly guessing an entire subproof. Therefore we need more abstract proof representations.

The proof representations we will be using and their relationships are depicted in Figure 1. The purpose of this section is to explain these representations and their relationships. As a first orientation let us just mention that the rows of Figure 1 contain notions on increasingly abstract levels: the level of proofs in the first row, that of formulas in the second row and that of terms in the third row. The transformations in each column are complexity-preserving (in a sense that will be made precise soon). The transformation of an object in the left column to an object in the right column increases its complexity considerably (exponentially in this paper).

For this whole section, fix a quantifier-free formula F with one free variable x s.t. $\forall x\,F$ is unsatisfiable. We will, for the sake of simplicity, explain our algorithm first in the setting of proofs of the end-sequent $\forall x\,F \to$. We will show how to abbreviate a given cut-free proof of this sequent by the introduction of cuts, which are of the form $\forall x\,A$ for A quantifier-free, such cuts will be called Π_1-cuts in the sequel. The algorithm will then be generalised to less restrictive end-sequents in Section 4.

3.1. Proofs and Herbrand's Theorem

A *sequent* is an ordered pair of sets of formulas, written as $\Gamma \to \Delta$. While the concrete variant of the sequent calculus is of little importance to the algorithms presented in this paper let us, for the sake of precision, fix it to be **G3c** + Cut_{cs}[2] from [39].

Definition 1 (Herbrand-sequent)**.** A tautological sequent of the form $H : F[x\backslash t_1], \ldots, F[x\backslash t_n] \to$ is called a *Herbrand-sequent* of $\forall x\, F \to$. We define $|H| = n$ and call it the *complexity of H*.

We thus measure the number of instances of $\forall x\, F$ used for showing its unsatisfiability. This complexity-measure is of fundamental importance as the undecidability of first-order logic hinges on it: a bound gives a decision procedure as most general unification can be used for bounding the term size in the number of instances; it then only remains to enumerate all possible instances having at most this bounding size. On the level of proofs we keep track of the number of used instances by counting the number of $\forall_{\text{l}}$- and $\exists_{\text{r}}$-inferences, for the other quantifier rules ($\forall_{\text{r}}$ and $\exists_{\text{l}}$) one application per formula suffices.

Definition 2. We define the *quantifier-complexity of a proof* π, written as $|\pi|_{\text{q}}$ as the number of $\forall_{\text{l}}$- and $\exists_{\text{r}}$-inferences in π.

Herbrand-sequents then correspond to cut-free proofs in the following sense.

Theorem 1. $\forall x\, F \to$ has a cut-free proof π with $|\pi|_{\text{q}} = l$ iff it has a Herbrand-sequent H with $|H| = l$.

Proof Sketch. Given π we obtain H by reading off the instances of $\forall x\, F$ from the proof π and collecting them in a sequent (if π contains some duplicate instances we add dummy instances to H for obtaining $|H| = |\pi|_{\text{q}}$).

Given H we first compute any propositional proof of H and obtain a cut-free proof π of $\forall x\, F \to$ by introducing the universal quantifier for each of those instances and applying a sufficient number of contractions. □

The above theorem shows that we can think of a Herbrand-sequent as a concise representation of a cut-free proof. A first important step towards our cut-introduction algorithm will be the generalisation of this relation to proofs with an arbitrary number of Π_1-cuts (in a way similar to [18]).

Definition 3. Let $u_1, \ldots, u_m$ be terms, let $A_1, \ldots, A_n$ be quantifier-free formulas, let $\alpha_1, \ldots, \alpha_n$ be variables, let $V(t)$ denote the set of variables occurring in the term t, and let $s_{i,j}$ for $1 \leqslant i \leqslant n, 1 \leqslant j \leqslant k_j$ be terms s.t.

1. $\text{V}(A_i) \subseteq \{\alpha_i, \ldots, \alpha_n\}$ for all i, and
2. $\text{V}(s_{i,j}) \subseteq \{\alpha_{i+1}, \ldots, \alpha_n\}$ for all i, j.

Then the sequent

$$H \;=\; F[x\backslash u_1], \ldots, F[x\backslash u_m], A_1 \supset \bigwedge_{j=1}^{k_1} A_1[\alpha_1\backslash s_{1,j}], \ldots, A_n \supset \bigwedge_{j=1}^{k_n} A_n[\alpha_n\backslash s_{n,j}] \to$$

is called an *extended Herbrand-sequent* of $\forall x\, F \to$ if H is a tautology.

What is this cryptic definition supposed to mean? An extended Herbrand-sequent of the above form will represent a proof with n Π_1-cuts whose cut formulas are $\forall\alpha_1\, A_1, \ldots, \forall\alpha_n\, A_n$ (or sometimes minor variants thereof), the α_i are the eigenvariables of the universal quantifiers in these cut-formulas, the $s_{i,j}$ the terms of the instances of the cut-formulas on the right-hand side of the cut and the u_i the terms of the instances of our end-formula $\forall x\, F$. The complexity of an extended Herbrand-sequent H of the above form is defined as $|H| = m + \sum_{j=1}^{n} k_j$. One can view an extended Herbrand-sequent together with a propositional proof of it as a particular form of proof in the ε-calculus [23] with the cuts corresponding to the critical formulas. We obtain the following correspondence to the sequent calculus:

[2] **G3c** + Cut_{cs} has no structural rules and all its rules are invertible.

Theorem 2. $\forall x\, F \to$ has a proof π with Π_1-cuts and $|\pi|_{\mathrm{q}} = l$ iff it has an extended Herbrand-sequent H with $|H| = l$.

While this theorem looks plausible it is not as straightforward to prove as one may expect. Its proof relies on Craig's interpolation theorem [12] which we briefly repeat here for the reader's convenience in the version of [38] and restricted to propositional logic. We split a sequent into two parts by writing it as a *partition* $\Gamma_1 \ ; \ \Gamma_2 \to \Delta_1 \ ; \ \Delta_2$. The purpose of doing so is merely to mark Γ_1, Δ_1 as belonging to one and Γ_2, Δ_2 as belonging to the other part of the partition. The logical meaning of $\Gamma_1 \ ; \ \Gamma_2 \to \Delta_1 \ ; \ \Delta_2$ is just $\Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2$.

Theorem 3. If a quantifier-free sequent $\Gamma_1 \ ; \ \Gamma_2 \to \Delta_1 \ ; \ \Delta_2$ is a tautology, then there is a quantifier-free formula I s.t.

1. Both $\Gamma_1 \to \Delta_1, I$ and $I, \Gamma_2 \to \Delta_2$ are tautologies, and
2. All atoms that appear in I appear in both $\Gamma_1 \to \Delta_1$ and $\Gamma_2 \to \Delta_2$.

Proof. See [38]. □

Proof of Theorem 2. For the left-to-right direction we proceed analogously to the cut-free case: by passing through the proof π and reading off the instances of quantified formulas (of both the end-formula and the cuts) we obtain an extended Herbrand-sequent H with $|H| \leqslant |\pi|_{\mathrm{q}}$ (which can be padded with dummy instances if necessary in order to obtain $|H| = |\pi|_{\mathrm{q}}$).

For the right-to-left direction let

$$H \ = \ F[x\backslash u_1], \ldots, F[x\backslash u_m], A_1 \supset \bigwedge_{j=1}^{k_1} A_1[\alpha_1\backslash s_{1,j}], \ldots, A_n \supset \bigwedge_{j=1}^{k_n} A_n[\alpha_n\backslash s_{n,j}] \to$$

be an extended Herbrand-sequent and let us begin by introducing some abbreviations. For a set of terms T and a formula F, write $F[x\backslash T]$ for the set of formulas $\{F[x\backslash t] \mid t \in T\}$. Abbreviate the "cut-implication" $A_i \supset \bigwedge_{j=1}^{k_i} A_i[\alpha_i\backslash s_{i,j}]$ as CI_i and let $U = \{u_1, \ldots, u_m\}$. Then H can be written more succinctly as $F[x\backslash U], \mathrm{CI}_1, \ldots, \mathrm{CI}_n \to$.

Let $U_i = \{u \in U \mid \mathrm{V}(u) \subseteq \{\alpha_{i+1}, \ldots, \alpha_n\}\}$ for $i = 0, \ldots, n$. First we will show that it suffices to find quantifier-free formulas $A'_1, \ldots, A'_n$ s.t. the sequent

$$H' \ = \ F[x\backslash U], A'_1 \supset \bigwedge_{j=1}^{k_1} A'_1[\alpha_1\backslash s_{1,j}], \ldots, A'_n \supset \bigwedge_{j=1}^{k_n} A'_n[\alpha_n\backslash s_{n,j}] \to$$

has a proof of the following *linear form*:

$$\dfrac{\dfrac{\vdots \qquad\qquad \vdots}{F[x\backslash U] \to A'_1, \ldots, A'_n \quad \bigwedge_{j=1}^{k_1} A'_1[\alpha_1\backslash s_{1,j}], F[x\backslash U_1] \to A'_2, \ldots, A'_n}\supset_l}{\begin{array}{c} F[x\backslash U], \mathrm{CI}'_1 \to A'_2, \ldots, A'_n \\ \vdots \qquad\qquad\qquad\qquad \vdots \\ F[x\backslash U], \mathrm{CI}'_1, \ldots, \mathrm{CI}'_{n-1} \to A'_n \qquad\qquad \bigwedge_{j=1}^{k_n} A'_n[\alpha_n\backslash s_{n,j}], F[x\backslash U_n] \to \end{array}}\supset_l$$

$$F[x\backslash U], \mathrm{CI}'_1, \ldots, \mathrm{CI}'_n \to$$

where CI'_i abbreviates $A'_i \supset \bigwedge_{j=1}^{k_i} A'_i[\alpha_i\backslash s_{i,j}]$. This suffices because in the above proof we can introduce cuts and quantifiers by replacing a segment of the form

$$\dfrac{F[x\backslash U], \mathrm{CI}'_1, \ldots, \mathrm{CI}'_{i-1} \to A'_i, \ldots, A'_n \quad \bigwedge_{j=1}^{k_i} A'_i[\alpha_i\backslash s_{i,j}], F[x\backslash U_i] \to A'_{i+1}, \ldots, A'_n}{F[x\backslash U], \mathrm{CI}'_1, \ldots, \mathrm{CI}'_i \to A'_{i+1}, \ldots, A'_n}\supset_l$$

by

$$\dfrac{\dfrac{\dfrac{F[x\backslash U_{i-1}], \forall x\, F \to A'_i, \ldots, A'_n}{F[x\backslash U_i], \forall x\, F \to A'_i, \ldots, A'_n}\ \forall_{\mathrm{l}}^*}{F[x\backslash U_i], \forall x\, F \to \forall x\, A'_i[\alpha_i\backslash x], A'_{i+1}, \ldots, A'_n}\ \forall_{\mathrm{r}} \quad \dfrac{A'_i[\alpha_i\backslash s_{i,j}]_{j=1}^{k_i}, F[x\backslash U_i] \to A'_{i+1}, \ldots, A'_n}{\forall x\, A'_i[\alpha_i\backslash x], F[x\backslash U_i] \to A'_{i+1}, \ldots, A'_n}\ \forall_{\mathrm{l}}^*}{F[x\backslash U_i], \forall x\, F \to A'_{i+1}, \ldots, A'_n}\ \mathrm{cut}$$

and finishing the proof at its root by

$$\dfrac{F[x\backslash U_n], \forall x\, F \to}{\forall x\, F \to}\ \forall_{\mathrm{l}}^*.$$

This transformation results in a proof whose number of $\forall_{\mathrm{l}}$-inferences is the complexity of the extended Herbrand-sequent as every term of H is introduced exactly once.

Let us now turn to the construction of the A'_i. Write

$$\begin{aligned} L_i \quad &\text{for} \quad \mathrm{CI}_1, \ldots, \mathrm{CI}_{i-1}, F[x\backslash U] \to A'_i, \ldots, A'_n, \text{ and} \\ R_i \quad &\text{for} \quad \bigwedge_{j=1}^{k_i} A'_i[\alpha_i\backslash s_{i,j}], F[x\backslash U_i] \to A'_{i+1}, \ldots, A'_n. \end{aligned}$$

Note that L_i and R_i depend only on those A'_j with $j \geqslant i$ and note furthermore that L_{n+1} is the extended Herbrand-sequent H which is a tautology by assumption. Fix $i \in \{1, \ldots, n\}$. Assuming $\vDash L_{i+1}$ we will now construct A'_i and show $\vDash L_i$ and $\vDash R_i$.

From $\vDash L_{i+1}$ we obtain

$$\vDash \mathrm{CI}_1, \ldots, \mathrm{CI}_{i-1}, F[x\backslash U] \to A_i, A'_{i+1}, \ldots, A'_n \text{ and} \tag{1}$$

$$\vDash \underbrace{\mathrm{CI}_1, \ldots, \mathrm{CI}_{i-1}, F[x\backslash (U\backslash U_i)]}_{\Gamma}, \underbrace{\bigwedge_{j=1}^{k_i} A_i[\alpha_i\backslash s_{i,j}], F[x\backslash U_i]}_{\Pi} \to \underbrace{A'_{i+1}, \ldots, A'_n}_{\Lambda} \tag{2}$$

from an application of $\supset_{\mathrm{l}}$ to CI_i. Applying the propositional interpolation theorem to the partition $\Gamma\ ;\ \Pi \to\ ;\ \Lambda$ of (2) yields I s.t. $\vDash \Gamma \to I$ and $\vDash \Pi, I \to \Lambda$. Furthermore I contains only such atoms which appear in $\Pi \to \Lambda$, hence $\mathrm{V}(I) \subseteq \{\alpha_{i+1}, \ldots, \alpha_n\}$. Define A'_i as $A_i \wedge I$. Observe that $\vDash R_i$ follows from $\vDash \Pi, I \to \Lambda$ and $\vDash L_i$ follows from (1) and $\vDash \Gamma \to I$. Hence L_i, R_i for $i = 1, \ldots, n$ are tautologies. But $L_1, R_1, \ldots, R_n$ are exactly the leaves of the linear proof from above which finishes the proof of the theorem. □

This result does not only generalize Proposition 2 of [20] to the case of an arbitrary number of cuts but also improves it considerably, even for the case of a single cut: the use of interpolants is new in this paper and allows to obtain $|\pi|_{\mathrm{q}} \leqslant |H|$ for an extended Herbrand sequent H. In general it is not possible to read back an extended Herbrand-sequent to a proof of linear form without changing the cut formulas as the following example shows. The reason for insisting on this linear form is that it does not contain any duplicate instances which permits to show the property $|\pi|_{\mathrm{q}} = |H|$. The duplication behavior of connectives in this transformation is reminiscent of the complexity results in [2].

Remark 1. The complexity of the proof π obtained from the extended Herbrand-sequent H can also be bound beyond its pure quantifier complexity $|\pi|_{\mathrm{q}}$. Let $\mathrm{d}(\psi)$ denote the depth of a proof ψ, i.e. the maximal number of inferences on a branch and let $\|H\|$ denote the logical complexity of H. Then the right-to-left direction of Theorem 2 can be strengthened as follows: there is a constant c s.t. for every extended Herbrand sequent H of $\forall x\, F \to$ with n cuts and $|H| = l$ there is a proof π with n Π_1-cuts, $|\pi|_{\mathrm{q}} = l$ and $\mathrm{d}(\pi) \leqslant c^n \|H\|$. This bound can be obtained from carrying out the proofs of Theorem 3 and Theorem 2 using $\vdash^d$ (derivability in depth d) instead of $\vDash$ (validity). It is created by the n-fold iteration of transformations of a proof of depth d to a proof of depth $c \cdot d$.

Example 1. Let $F = P(x) \wedge (P(c) \supset Q(x)) \wedge (Q(x) \supset P(d)) \wedge \neg P(d)$ and $A_1 = P(\alpha_1)$. Furthermore let $m = 1, u_1 = \alpha_1$ and $n = 1, k_1 = 1, s_{1,1} = c$. Then

$$E = F[x\backslash u_1], \ldots, F[x\backslash u_m], A_1 \supset \bigwedge_{j=1}^{k_1} A_1[\alpha_1\backslash s_{1,j}], \ldots, A_n \supset \bigwedge_{j=1}^{k_n} A_n[\alpha_n\backslash s_{n,j}] \rightarrow$$
$$= P(\alpha_1) \wedge (P(c) \supset Q(\alpha_1)) \wedge (Q(\alpha_1) \supset P(d)) \wedge \neg P(d), P(\alpha_1) \supset P(c) \rightarrow$$

is a tautology and hence an extended Herbrand-sequent of $\forall x\, F \rightarrow$.

Let us now try to construct a linear **LK**-proof that corresponds to E. Such a proof contains a cut on $\forall x\, P(x)$ as its last inference. The formula $\forall x\, F$ must be instantiated on the left above this cut to obtain $F[x\backslash\alpha_1]$ as α_1 is the eigenvariable of the cut formula. This leaves the right side of the cut as $P(c) \rightarrow$ which is not valid, a second instance of $\forall x\, F$ would be needed. The solution used in the proof of Theorem 2 is based on computing a propositional interpolant of $F[x\backslash\alpha_1]; P(c) \rightarrow;$. This can be done e.g. by first computing a proof of the sequent $F[x\backslash\alpha_1], P(c) \rightarrow$, e.g. the following $\psi =$

$$\dfrac{\dfrac{\dfrac{P(c) \rightarrow P(c) \quad \dfrac{Q(\alpha_1) \rightarrow Q(\alpha_1) \quad \dfrac{P(\alpha_1), P(d) \rightarrow P(d)}{P(\alpha_1), P(d), \neg P(d) \rightarrow}\,\neg_l}{P(\alpha_1), Q(\alpha_1), Q(\alpha_1) \supset P(d), \neg P(d) \rightarrow}\,\supset_l}{P(\alpha_1), P(c) \supset Q(\alpha_1), Q(\alpha_1) \supset P(d), \neg P(d), P(c) \rightarrow}\,\supset_l}{P(\alpha_1) \wedge (P(c) \supset Q(\alpha_1)) \wedge (Q(\alpha_1) \supset P(d)) \wedge \neg P(d), P(c) \rightarrow}\,\wedge_l^*$$

The propositional interpolant induced by the partition $F[x\backslash\alpha_1]; P(c) \rightarrow;$ of ψ according to the algorithm of [38] is computed as

$$\dfrac{\dfrac{\dfrac{\neg P(c) \quad \dfrac{\dfrac{\bot}{\bot}\,\neg_l \quad \bot}{\bot \vee \bot}\,\supset_l}{\neg P(c) \vee \bot \vee \bot}\,\supset_l}{\neg P(c) \vee \bot \vee \bot}\,\wedge_l^*$$

which simplifies to $\neg P(c)$. Hence the new cut formula is $\forall x\, (P(x) \wedge \neg P(c))$ which renders the right side of the cut provable as $P(c) \wedge \neg P(c) \rightarrow$.

3.2. Proofs and Grammars

Now that we have established the connection between proofs and (extended) Herbrand-sequents we can move on to the term level of Figure 1. A first trivial observation is that, assuming the knowledge of F, a Herbrand-sequent H for $\forall x\, F \rightarrow$ does not carry more information than just the set of terms T s.t. $H = F[x\backslash T] \rightarrow$.

A set of terms, in the terminology of formal language theory, is a tree language. The central theoretical result on which this paper is based is an analogous relation between extended Herbrand-sequents (or: proofs with Π_1-cuts) and *a certain class of tree grammars*. This result has first been proved in [19], see also [21] for a generalization.

Tree languages are a natural generalization of formal (string) languages, see e.g. [15, 11]. Many important notions, such as regular and context-free languages carry over from the setting of strings to that of trees. The class of *rigid* tree languages has been introduced in [25] with applications in verification in mind, see [26]. Rigid tree languages augment regular tree languages by the ability to carry out certain equality tests, a property that is very useful for applications.

In the context of proof theory it is more natural to work with grammars than with automata because of the generative nature of cut-elimination. The class of grammars we will use in this paper is a subclass of rigid grammars: the *totally rigid acyclic tree grammars*. We write $\mathcal{T}_\Sigma(V)$ for the set of terms in the first-order signature Σ over the set of variables V and $\mathcal{T}_\Sigma$ for $\mathcal{T}_\Sigma(\varnothing)$. For a symbol $f \in \Sigma$ we write (f/k) for denoting the arity k of f.

Definition 4. A *regular tree grammar* is a tuple $G = \langle N, \Sigma, \tau, P\rangle$, where N is a finite set of non-terminal symbols, Σ is a first-order signature, $\tau \in N$ is the *start symbol* and P is a finite set of production rules of the form $\beta \to t$ with $\beta \in N$ and $t \in \mathcal{T}_\Sigma(N)$.

The *one-step derivation relation* $\to^1_G$ of a regular tree grammar G consists of all pairs $u[\beta] \to^1_G u[t]$ where $\beta \to t \in P$. A derivation in G is a finite sequence of terms $t_0 = \tau, t_1, \ldots, t_n$ s.t. $t_i \to^1_G t_{i+1}$. The language of G is defined as $L(G) = \{t \in \mathcal{T}_\Sigma \mid t \text{ has a } G\text{-derivation}\}$.

Definition 5. A *rigid tree grammar* is a tuple $G = \langle N, N_R, \Sigma, \tau, P\rangle$, where $\langle N, \Sigma, \tau, P\rangle$, is a regular tree grammar and $N_R \subseteq N$ is the set of *rigid non-terminals*. We speak of a *totally rigid tree grammar* if $N_R = N$. In this case we will just write $\langle N_R, \Sigma, \tau, P\rangle$.

A derivation $t_0 = \tau, t_1, \ldots, t_n = t$ of a term $t \in \mathcal{T}_\Sigma$ in a rigid tree grammar is a derivation in the underlying regular tree grammar that satisfies the additional *rigidity condition*: If there are $i, j < n$, a non-terminal $\beta \in N_R$, and positions p and q such that $t_i|_p = \beta$ and $t_j|_q = \beta$, then $t|_p = t|_q$. The language $L(G)$ of the rigid tree grammar G is the set of all terms $t \in \mathcal{T}_\Sigma$ which can be derived under the rigidity condition. Totally rigid tree grammars are formalisms for specifying sets of substitutions and thus are particularly useful for describing instances generated by cut-elimination.

Example 2. Let $\Sigma = \{0/0, s/1\}$. A simple pumping argument shows that the language $L = \{f(t,t) \mid t \in \mathcal{T}_\Sigma\}$ is not regular. On the other hand, L is generated by the rigid tree grammar $\langle\{\tau, \alpha, \beta\}, \{\alpha\}, \{0/0, s/1, f/2\}, \tau, P\rangle$ where $P = \{\tau \to f(\alpha, \alpha), \alpha \to 0 \mid s(\beta), \beta \to 0 \mid s(\beta)\}$.

Definition 6. The *grammar of an extended Herbrand-sequent*

$$H \;\equiv\; F[x\backslash u_1], \ldots, F[x\backslash u_m], A_1 \supset \bigwedge_{j=1}^{k_1} A_1[\alpha_1\backslash s_{1,j}], \ldots, A_n \supset \bigwedge_{j=1}^{k_n} A_n[\alpha_n\backslash s_{n,j}] \to$$

is defined as the totally rigid $G(H) = \langle N_R, \Sigma, \tau, P\rangle$ where $N_R = \{\tau, \alpha_1, \ldots, \alpha_n\}$, Σ is the signature of H and $P = \{\tau \to u_i \mid 1 \leqslant i \leqslant m\} \cup \{\alpha_i \to s_{i,j} \mid 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant k_i\}$.

A derivation of the form $\beta \to^1_G t_1 \to^1_G \cdots \to^1_G t_n$ is called cyclic if $\beta \in \mathrm{V}(t_n)$. A grammar is called *acyclic* if it does not have any cyclic derivations. Note that condition 2 of Definition 3 ensures that the grammar of an extended Herbrand-sequent is acyclic. Furthermore, by definition, the grammar of an extended Herbrand-sequent is totally rigid. The language of such a grammar can be written in the following normal form.

Lemma 1. If G is totally rigid and acyclic, then up to renaming of the non-terminals $G = \langle\{\alpha_0, \ldots, \alpha_n\}, \Sigma, \alpha_0, P\rangle$ with $L(G) = \{\alpha_0[\alpha_0\backslash t_0] \cdots [\alpha_n\backslash t_n] \mid \alpha_i \to t_i \in P\}$.

Proof. Acyclicity permits to rename the non-terminals in such a way that $\alpha_i \to^1_G t_1 \to^1_G \cdots \to^1_G t_n$ and $\alpha_j \in \mathrm{V}(t_n)$ implies $j > i$. The notation based on substitutions is then possible because, due to total rigidity, each $t \in L(G)$ can be derived using at most one production for each non-terminal. See [21] for a detailed proof. □

In particular, the language of a totally rigid acyclic grammar is finite. This lemma also suggests a compact notation for totally rigid acyclic grammars: we write

$$U \circ_{\alpha_1} S_1 \cdots \circ_{\alpha_n} S_n$$

for the grammar $\langle\{\tau, \alpha_1, \ldots, \alpha_n\}, \Sigma, \tau, P\rangle$ where $P = \{\tau \to u \mid u \in U\} \cup \{\alpha_i \to s_i \mid 1 \leqslant i \leqslant n, s_i \in S_i\}$, τ is some fresh start symbol and Σ is the signature of the terms appearing in P. Using this notation, we can observe that $L(U) = U$ for a set of terms U and $L(G \circ_\alpha S) = \{u[\alpha\backslash s] \mid u \in L(G), s \in S\}$ for a totally rigid acyclic tree grammar G and a set of terms S. If the non-terminals are clear from the context, this notation is further abbreviated as

$$U \circ S_1 \cdots \circ S_n.$$

One can then obtain a cut-elimination theorem based on grammars:

Theorem 4. If H is an extended Herbrand-sequent of $\forall x\, F \to$, then $\{F[x\backslash t] \mid t \in L(G(H))\} \to$ is a Herbrand-sequent of $\forall x\, F \to$.

Proof. This can be shown by following the development of the grammar during a cut-elimination process, see [19, 17] and also [21] for a more general result. □

Throughout this whole paper all the grammars we are dealing with will be totally rigid and acyclic. Therefore we will henceforth use *grammar* as synonym for *totally rigid acyclic tree grammar*.

3.3. Cut-Introduction

We have already observed above that it is not feasible to invert Gentzen's cut-elimination steps literally. The key to our method is that moving from the level of proofs to the level of grammars provides us with a transformation that is much easier to invert. The computation of the language of a grammar can simply be inverted as: *given a finite tree language L, find a grammar G s.t. $L(G) = L$*. We will describe an algorithm for solving this problem in detail in Section 5.

The only piece then still missing in Figure 1 is to obtain an extended Herbrand-sequent from G. Note that, for a given G, the term-part of the extended Herbrand-sequent is already determined using Lemma 1. What we do not know yet are the cut-formulas. Hence we define:

Definition 7. Let $u_1, \ldots, u_m$ be terms, let $X_1, \ldots, X_n$ be monadic second-order variables, let $\alpha_1, \ldots, \alpha_n$ be variables, and let $s_{i,j}$ for $1 \leqslant i \leqslant n, 1 \leqslant j \leqslant k_j$ be terms s.t. $\mathrm{V}(s_{i,j}) \subseteq \{\alpha_{i+1}, \ldots, \alpha_n\}$ for all i, j. Then the sequent

$$H \;=\; F[x\backslash u_1], \ldots, F[x\backslash u_m], X_1(\alpha_1) \supset \bigwedge_{j=1}^{k_1} X_1(s_{1,j}), \ldots, X_n(\alpha_n) \supset \bigwedge_{j=1}^{k_n} X_n(s_{n,j}) \to$$

is called a *schematic extended Herbrand-sequent* of $\forall x\, F \to$ if $\bigwedge_{t \in L(G(H))} F[x\backslash t] \to$ is a tautology (where $G(H)$ is defined analogously to Definition 6).

A *solution* of a schematic extended Herbrand-sequent H is a substitution $\sigma = [X_i\backslash \lambda\alpha_i.A_i]_{i=1}^n$ s.t. $\mathrm{V}(A_i) \subseteq \{\alpha_i, \ldots, \alpha_n\}$ and $H\sigma$ is a tautology.

The reason for calling such a substitution σ a solution is the close relationship of this problem to unification problems modulo the theory of Boolean algebras, in particular to Boolean unification with constants [29, 1]. By comparison with Definition 3 note that if σ is a solution for H, then $H\sigma$ is an extended Herbrand-sequent. There are a number of interesting and practically relevant results about the solutions of such sequents, see Section 6. The central property which is of interest right now is that such a sequent always has a solution.

Definition 8. Let H be a schematic extended Herbrand-sequent. Define

$$C_1 \;=\; \bigwedge_{i=1}^{m} F[x\backslash u_i] \text{ and } C_{i+1} \;=\; \bigwedge_{j=1}^{k_i} C_i[\alpha_i\backslash s_{i,j}] \text{ for } i = 1, \ldots, n.$$

Then

$$\sigma := [X_i\backslash \lambda\alpha_i.C_i]_{i=1}^n$$

is called *canonical substitution* of H.

We will now show that the canonical substitution is, in fact, a solution.

Lemma 2. Let H and C_i be as in Definition 8. Then $C_{n+1} \to$ is a tautology.

Proof. By definition, $C_{n+1} \to$ is $\bigwedge_{i=1}^{m} \bigwedge_{j_1=1}^{k_1} \cdots \bigwedge_{j_n=1}^{k_n} F[x\backslash u_i][\alpha_1\backslash s_{1,j_1}] \cdots [s_{n,\alpha_n}\backslash j_n] \to$ which by Lemma 1 is $\bigwedge_{t \in L(G(H))} F[x\backslash t] \to$ which is a tautology as H is a schematic extended Herbrand-sequent. □

Lemma 3. Let H be a schematic extended Herbrand-sequent and σ be its canonical substitution. Then σ is a solution of H.

Proof. First note that the variable condition is fulfilled as $\mathrm{V}(C_i) \subseteq \{\alpha_i, \ldots, \alpha_n\}$. Then observe that

$$H\sigma \;=\; F[x\backslash u_1], \ldots, F[x\backslash u_m], C_1 \supset \bigwedge_{j=1}^{k_1} C_1[\alpha_1\backslash s_{1,j}], \ldots, C_n \supset \bigwedge_{j=1}^{k_n} C_n[\alpha_n\backslash s_{n,j}] \rightarrow$$

is logically equivalent to

$$C_1, C_1 \supset C_2, \ldots, C_n \supset C_{n+1} \rightarrow .$$

The unsatisfiability of C_{n+1} follows from Lemma 2, hence $H\sigma$ is a tautology. □

In light of the above result we will henceforth call σ the *canonical solution*. Note that the canonical solution permits a sequent calculus proof of a linear form in the sense of the proof of Theorem 2, and hence — for this solution — interpolation is not necessary in the construction of the proof with cuts.

Theorem 5. $\forall x\, F \rightarrow$ has an extended Herbrand-sequent H with $|H| = l$ iff there is a totally rigid acyclic tree grammar G with $|G| = l$ s.t. $\bigwedge_{t \in L(G)} F[x\backslash t] \rightarrow$ is a tautology.

Proof. The left-to-right direction of this statement follows from Theorem 4 together with the observation that $|G(H)| = |H|$. For the right-to-left direction assume that G is given, let H be the schematic extended Herbrand-sequent of G. Then the result follows from Lemma 3. □

Now we have proved all results mentioned in Figure 1 and can finally describe our approach to cut-introduction. It consists in following this diagram in a clockwise fashion from the cut-free proof to the proof with cut. More specifically, given as input a cut-free proof π our algorithm will proceed as follows:

1. Extract the set of terms T of $H(\pi)$ (as in Theorem 1).
2. Find a suitable grammar G s.t. $L(G) = T$.
3. Compute an extended Herbrand-sequent H from G (as in Theorem 5).
4. Construct a proof ψ with cut from H (as in Theorem 2).

Example 3. Consider the sequent $\forall x\, F \rightarrow$ where

$$F = Pa \wedge (Px \supset Pfx) \wedge \neg Pf^9 a.$$

Let π be a straightforward cut-free proof of $\forall x\, F \rightarrow$, then $|\pi|_{\mathrm{q}} = 9$. Following the above outline of an algorithm we carry out the following steps. Extract the set of terms

$$T = \{a, fa, f^2a, f^3a, f^4a, f^5a, f^6a, f^7a, f^8a\}$$

from $H(\pi)$ following Theorem 1. Compute a grammar G with $L(G) = T$, for example

$$G = \{\alpha, f\alpha, f^2\alpha\} \circ_\alpha \{a, f^3a, f^6a\}.$$

As in the proof of Theorem 5, this grammar induces the schematic extended Herbrand-sequent

$$H \;=\; F[x\backslash\alpha], F[x\backslash f\alpha], F[x\backslash f^2\alpha], X(\alpha) \supset (X(a) \wedge X(f^3a) \wedge X(f^6a)) \rightarrow$$

whose canonical solution is

$$\sigma = [X\backslash\lambda\alpha.\, (F[x\backslash\alpha] \wedge F[x\backslash f\alpha] \wedge F[x\backslash f^2\alpha])]$$

Hence $H\sigma$ is an extended Herbrand-sequent with $|H\sigma| = |H| = 6$ which in turn induces a proof ψ as in Theorem 2 which has $|\psi|_{\mathrm{q}} = 6$ and contains a single Π_1-cut whose cut-formula is

$$\forall x\, (F \wedge F[x\backslash fx] \wedge F[x\backslash f^2x]).$$

Observe that we have decreased the quantifier complexity from $|\pi|_{\mathrm{q}} = 9$ to $|\psi|_{\mathrm{q}} = 6$.

While this is a satisfactory situation from the abstract point of view of the quantifier complexity, this procedure is clearly not yet fit for practical applications with the aim of proof compression. The rest of this paper is devoted to making it so: in Section 4 we generalize the results of this section to a sufficiently large class of end-sequents. In Section 5 we present an efficient algorithm for the computation of a grammar and in Section 6 we describe how to obtain solutions for a schematic extended Herbrand-sequent which are shorter than the canonical solution.

4. More General End-Sequents

The class of end-sequents considered in the previous section, while leading to a comparatively simple statement of the central results, is clearly too restricted for concrete applications. We will therefore extend our proof-theoretic infrastructure to proofs of end-sequents of the form

$$\forall x_1 \cdots \forall x_{l_1} F_1, \ldots, \forall x_1 \cdots \forall x_{l_p} F_p \to \exists x_1 \cdots \exists x_{l_{p+1}} F_{p+1}, \ldots, \exists x_1 \cdots \exists x_{l_q} F_q$$

with $l_i \geqslant 0$ and F_i quantifier-free. We say that a sequent in this format is a Σ_1-sequent. Note that every first-order sequent can be transformed to this form by skolemization and prenexing. Permitting l_i to be zero allows for quantifier-free formulas such as in the example of Section 2. While the formalism now gets notationally more complicated, the results and proofs remain essentially the same. We write $\bar{x}$ for a vector $(x_1, \ldots, x_n)$ of variables, $\bar{t}$ for a vector $(t_1, \ldots, t_n)$ of terms and $[\bar{x}\backslash\bar{t}]$ for the substitution $[x_1\backslash t_1, \ldots, x_n\backslash t_n]$. For this whole section, we fix a sequent $\Gamma \to \Delta$ of the above form.

Definition 9. A tautological sequent of the form

$$\{F_i[\bar{x}\backslash\overline{t_{i,j}}] \mid 1 \leqslant i \leqslant p, 1 \leqslant j \leqslant n_i\} \to \{F_i[\bar{x}\backslash\overline{t_{i,j}}] \mid p < i \leqslant q, 1 \leqslant j \leqslant n_i\}$$

is called *Herbrand-sequent* of $\Gamma \to \Delta$.

The size of a Herbrand-sequent is defined as $|H| = \sum_{i=1}^{q} n_i$. Note that we only count formulas obtained by instantiation. Now as we are dealing with blocks of quantifiers it is appropriate to also change the size measure on proofs to consider blocks instead of single quantifiers. To that aim we change the quantifier rules in our sequent calculus to allow the introduction of a block of quantifiers (which is a natural alternative for a number of problems related to proof size, see e.g. [4]):

$$\frac{\forall x_1 \cdots \forall x_n\, A, A[\bar{x}\backslash\bar{t}], \Gamma \to \Delta}{\forall x_1 \cdots \forall x_n\, A, \Gamma \to \Delta}\ \forall^*_{\mathrm{l}} \qquad \frac{\Gamma \to \Delta, A[\bar{x}\backslash\bar{t}], \exists x_1 \cdots \exists x_n\, A}{\Gamma \to \Delta, \exists x_1 \cdots \exists x_n\, A}\ \exists^*_{\mathrm{r}}$$

We write $\|\pi\|_{\mathrm{q}}$ for the number of $\forall^*_{\mathrm{l}}$- and $\exists^*_{\mathrm{r}}$-inferences in the proof π.

Theorem 6. $\Gamma \to \Delta$ has a cut-free proof π with $\|\pi\|_{\mathrm{q}} = l$ iff it has a Herbrand-sequent H with $|H| = l$.

Proof. As for Theorem 1. □

Definition 10. Let $\overline{u_{i,1}}, \ldots, \overline{u_{i,m_i}}$ be vectors of terms with l_i elements each. Let $A_1, \ldots, A_n$ be quantifier-free formulas, let $\alpha_1, \ldots, \alpha_n$ be variables, and let $s_{i,j}$ for $1 \leqslant i \leqslant n, 1 \leqslant j \leqslant k_j$ be terms s.t.

1. $\mathrm{V}(A_i) \subseteq \{\alpha_i, \ldots, \alpha_n\}$ for all i, and
2. $\mathrm{V}(s_{i,j}) \subseteq \{\alpha_{i+1}, \ldots, \alpha_n\}$ for all i, j.

Then the sequent

$$\begin{array}{rcl} H & = & \{F_i[\bar{x}\backslash\overline{u_{i,j}}] \mid 1 \leqslant i \leqslant p, 1 \leqslant j \leqslant m_i\}, A_1 \supset \bigwedge_{j=1}^{k_1} A_1[\alpha_1\backslash s_{1,j}], \ldots, A_n \supset \bigwedge_{j=1}^{k_n} A_n[\alpha_n\backslash s_{n,j}] \\ & & \to \{F_i[\bar{x}\backslash\overline{u_{i,j}}] \mid p < i \leqslant q, 1 \leqslant j \leqslant m_i\} \end{array}$$

is called an *extended Herbrand-sequent* of $\Gamma \to \Delta$ if H is a tautology.

The notion of schematic extended Herbrand-sequent is defined analogously to Definition 7 by replacing the formulas A_i in the above definition by monadic predicate variables X_i. The size of a (schematic) extended Herbrand sequent H of the above form is $|H| = \sum_{i=1}^{q} m_i + \sum_{j=1}^{n} k_j$.

Theorem 7. $\Gamma \to \Delta$ has a proof with Π_1-cuts and $\|\pi\|_{\mathrm{q}} = l$ iff it has an extended Herbrand-sequent H with $|H| = l$.

Proof. Analogous to the proof of Theorem 2, replacing $F[x\backslash U_i]$ by the collection of all instances $F_i[\bar{x}\backslash\overline{u_{i,j}}]$ s.t. all terms in $u_{i,j}$ contain only variables from $\{\alpha_{i+1}, \ldots, \alpha_n\}$. □

The above theorem encapsulates an algorithm for the construction of a proof with Π_1-cuts from an extended Herbrand-sequent. We will henceforth use the abbreviation PCA for this proof-construction algorithm.

In order to represent term vectors, it is helpful to enrich our signature by new function symbols $f_1, \ldots, f_q$ where f_i has arity l_i. The function symbol f_i will serve the purpose of grouping a term-tuple which corresponds to an instantiation of the formula $\forall x_1 \cdots \forall x_{l_i} F_i$ if $i \leqslant p$ (or $\exists x_1 \cdots \exists x_{l_i} F_i$ if $i > p$).

Definition 11. The grammar of an extended Herbrand-sequent

$$H \;=\; \begin{array}{l} \{F_i[\bar{x}\backslash\overline{u_{i,j}}] \mid 1 \leqslant i \leqslant p, 1 \leqslant j \leqslant m_i\}, A_1 \supset \bigwedge_{j=1}^{k_1} A_1[\alpha_1\backslash s_{1,j}], \ldots, A_n \supset \bigwedge_{j=1}^{k_n} A_n[\alpha_n\backslash s_{n,j}] \\ \to \{F_i[\bar{x}\backslash\overline{u_{i,j}}] \mid p < i \leqslant q, 1 \leqslant j \leqslant m_i\} \end{array}$$

is defined as $G(H) = \langle N_R, \Sigma, \tau, P\rangle$ where $N_R = \{\tau, \alpha_1, \ldots, \alpha_n\}$, Σ is the signature of H plus $\{f_1, \ldots, f_q\}$ and $P = \{\tau \to f_i(\overline{u_{i,j}}) \mid 1 \leqslant i \leqslant q, 1 \leqslant j \leqslant m_i\} \cup \{\alpha_i \to s_{i,j} \mid 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant k_i\}$.

Note that this definition also applies to Herbrand-sequents (as in Definition 9): then $n = 0$ and we obtain a trivial grammar $\langle N_R, \Sigma, \tau, P\rangle$ with $N_R = \{\tau\}$. Using this grammar, we define the *Herbrand terms* of a Herbrand-sequent as the set $\{t \mid (\tau \to t) \in P\}$. The *Herbrand terms of a cut-free proof* π are then the Herbrand terms of the Herbrand-sequent extracted from π via Theorem 6.

Example 4. Consider the sequent

$$P(0,0), \forall x \forall y\,(P(x,y) \supset P(s(x),y)), \forall x \forall y\,(P(x,y) \supset P(x,s(y)) \to P(s^4(0), s^4(0)).$$

Abbreviating $P(x,x) \supset P(s^2(x), s^2(x))$ as $F(x)$ we see that $\forall x\, F(x)$ is a useful cut formula that allows to decrease the number of $\forall^*_1$-inferences. A corresponding extended Herbrand-sequent is

$$\begin{array}{rl} E = & P(0,0), P(\alpha,\alpha) \supset P(s(\alpha),\alpha), P(s(\alpha),\alpha) \supset P(s(\alpha),s(\alpha)), P(s(\alpha),s(\alpha)) \supset P(s^2(\alpha),s(\alpha)), \\ & P(s^2(\alpha),\alpha) \supset P(s^2(\alpha),s^2(\alpha)), F(\alpha) \supset (F(0) \wedge F(s^2(0))) \to P(s^4(0),s^4(0)) \end{array}$$

The corresponding grammar is $G(E) = \langle\{\tau,\alpha\}, \{0/0, s/1\}, \tau, P\rangle$, with $|E| = |G(E)| = 6$ and

$$P = \{\tau \to f_1(\alpha,\alpha) | f_2(s(\alpha),\alpha) | f_1(s(\alpha),s(\alpha)) | f_2(s^2(\alpha),\alpha), \alpha \to 0 | s^2(0)\}.$$

In extension of our compact notation for grammars we write

$$(U_1, \ldots, U_q) \circ_{\alpha_1} S_1 \cdots \circ_{\alpha_n} S_n$$

for the grammar $G(H)$ of Definition 11 where $U_i = \{\overline{u_{i,j}} \mid 1 \leqslant j \leqslant m_i\}$ and $S_i = \{s_{i,j} \mid 1 \leqslant j \leqslant k_i\}$. As before, we leave out the α_i if they are obvious from the context. In this notation the function symbols f_i are implicitly specified by the position of U_i in the vector $(U_1, \ldots, U_q)$. Consequently, each $t \in L((U_1, \ldots, U_q) \circ_{\alpha_1} S_1 \cdots \circ_{\alpha_n} S_n)$ has one of the f_i as top-level symbol and these are the only occurrences of f_i. For notational convenience and if $l_i = 1$ for all i we sometimes write $L((U_1, \ldots, U_q) \circ_{\alpha_1} S_1 \cdots \circ_{\alpha_n} S_n)$ as a vector of sets of terms in the form $(T_1, \ldots, T_q)$ where $T_i = \{t \in L((U_1, \ldots, U_q) \circ_{\alpha_1} S_1 \cdots \circ_{\alpha_n} S_n) \mid t = f_i(\bar{s}) \text{ for some } \bar{s}\}$.

Theorem 8. If H is an extended Herbrand-sequent of $\Gamma \to \Delta$, then

$$\{F_i[\bar{x}\backslash\bar{t}] \mid 1 \leqslant i \leqslant p, f_i(\bar{t}) \in L(G(H))\} \to \{F_i[\bar{x}\backslash\bar{t}] \mid p < i \leqslant q, f_i(\bar{t}) \in L(G(H))\}$$

is a Herbrand-sequent of $\Gamma \to \Delta$.

Proof. As for Theorem 4. □

Definition 12. Let H be a schematic extended Herbrand sequent. Define

$$C_1 = \bigwedge_{i=1}^{p} \bigwedge_{j=1}^{m_i} F_i[\bar{x}\backslash\overline{u_{i,j}}] \wedge \bigwedge_{i=p+1}^{q} \bigwedge_{j=1}^{m_i} \neg F_i[\bar{x}\backslash\overline{u_{i,j}}] \text{ and } C_{i+1} \;= \bigwedge_{j=1}^{k_i} C_i[\alpha_i\backslash s_{i,j}] \text{ for } i = 1, \ldots, n.$$

Then

$$\sigma := [X_i\backslash\lambda\alpha_i.C_i]_{i=1}^{n}$$

is called *canonical substitution* of H.

Lemma 4. Let H be a schematic extended Herbrand-sequent and σ be its canonical substitution. Then σ is a solution of H.

Proof. As for Lemma 3. □

As in Section 3, the canonical substitution is hence called *canonical solution.*

Theorem 9. $\Gamma \to \Delta$ has an extended Herbrand-sequent H with $|H| = l$ iff there is a totally rigid acyclic tree grammar G with $|G| = l$ s.t.

$$\{F_i[\bar{x}\backslash\bar{t}] \mid 1 \leqslant i \leqslant p, f_i(\bar{t}) \in L(G(H))\} \to \{F_i[\bar{x}\backslash\bar{t}] \mid p < i \leqslant q, f_i(\bar{t}) \in L(G(H))\}$$

is a tautology.

Proof. Analogous to the proof of Theorem 5, using the canonical solution to obtain an extended Herbrand-sequent from a grammar. □

5. Efficient Computation of a Grammar

Let π be a cut-free proof and T the set of (tuples of) terms used in rules $\forall_l^*$ and $\exists_r^*$ in π. From Theorem 6 we conclude that $|T| = \|\pi\|_q$ and that T is easily obtained from the Herbrand sequent.

In this section we address the problem of obtaining a grammar G such that $L(G) = T$. As it was shown before in Theorems 2 and 5, the size of G will determine the quantifier complexity of the proof with cuts. Since we are interested in reducing this complexity, the main goal is to find a *minimal grammar* G, such that $|G| < |T|$. Whenever this is possible, we are able to construct a proof with cuts ψ such that $\|\psi\|_q < \|\pi\|_q$. Note that this might not be possible for every set of terms T.

Given our representation for $G = U \circ S_1 \circ ... \circ S_n$, the size $|G|$ is the same as $|U| + |S_1| + ... + |S_n|$. Since there is a bound on the size of G, namely, $|T|$, the most naive algorithm for finding grammars would be *guessing* which terms occur in S_i or U and *checking* whether this grammar generates T. We refer to this algorithm as GG for theoretical purposes. However, in this section we describe a more efficient approach for computing grammars that can be used in practice.

Assume π is a proof of an end-sequent of the form:

$$\forall x_1 \cdots \forall x_{l_1} F_1, \ldots, \forall x_1 \cdots \forall x_{l_p} F_p \to \exists x_1 \cdots \exists x_{l_{p+1}} F_{p+1}, \ldots, \exists x_1 \cdots \exists x_{l_q} F_q$$

Its Herbrand sequent, as described in Section 4, is:

$$\{F_i[\bar{x}\backslash\overline{t_{i,j}}] \mid 1 \leqslant i \leqslant p, 1 \leqslant j \leqslant n_i\} \to \{F_i[\bar{x}\backslash\overline{t_{i,j}}] \mid p < i \leqslant q, 1 \leqslant j \leqslant n_i\}$$

The terms used to instantiate the formulas F_i are easily obtained from this sequent: they are the vectors $t_{i,j}$. Let

$$T = \{f_i(t_{i,j}) \mid 1 \leqslant i \leqslant q\}.$$

with f_i being fresh function symbols as in Definition 11. These function symbols will facilitate the computation of a grammar, avoiding the need to deal with tuples of terms instead of terms. Then the algorithm described in this section will compute a grammar, as in Definition 11, for the set of terms T.

We will start by describing how to compute a grammar of the form $U \circ S$, which contains the terms used in a proof with one cut. Later, in Section 5.4, we show how to iterate this procedure in order to get the grammar $U \circ S_1 \circ ... \circ S_n$ which will contain the terms used in a proof with n cuts.

The algorithm will rely on an operation called Δ-vector, explained in Section 5.1, and a data structure called Δ-table, explained in Section 5.2. Intuitively, the operation computes "partial" grammars that are stored in this table, which is later processed to obtain grammars that generate the whole set T.

From now on we consider T as a *sequence* of terms instead of a set. This will guide the search and help prune the search space.

5.1. Δ-vector

The Δ-vector of a sequence of terms T describes the differences between the terms in T. It is defined as:

$$\Delta(t_1, ..., t_n) = \begin{cases} (f(u_1, ..., u_m), (s_1, ..., s_n)) & \text{if all } t_i = f(t_1^i, ..., t_m^i) \text{ and} \\ & \Delta(t_j^1, ..., t_j^n) = (u_j, (s_1, ..., s_n)) \ \ \forall\ j \in \{1, ..., m\} \\ (\alpha, (t_1, ..., t_n)) & \text{otherwise} \end{cases}$$

where α is an eigenvariable.

For example, if $T = (fa, fb)$, its Δ-vector is $(f\alpha, (a, b))$. But in order to make this definition clearer, we will analyse a more involved example. Let $T = \{f(gc, c), f(g^2c, gc), f(g^3c, g^2c)\}$. Then:

$$\Delta(f(gc, c), f(g^2c, gc), f(g^3c, g^2c)) = (f(u_1, u_2), (s_1, s_2, s_3)) \tag{3}$$

if

$$\Delta(gc, g^2c, g^3c) = (u_1, (s_1, s_2, s_3)) \tag{4}$$

$$\Delta(c, gc, g^2c) = (u_2, (s_1, s_2, s_3)) \tag{5}$$

Note that the second element of the pair, the vector (s_1, s_2, s_3) must be the same for the Δ-vector of the arguments.

In order to solve Equation 4, we apply the same definition:

$$\Delta(gc, g^2c, g^3c) = (gu_1', (s_1', s_2', s_3')) \tag{6}$$

if

$$\Delta(c, gc, g^2c) = (u_1', (s_1', s_2', s_3')) \tag{7}$$

Since the terms in (c, gc, g^2c) do not have a common head symbol, it's Δ-vector is: $(\alpha, (c, gc, g^2c))$. This solves Equations 5 and 7, and $u_1' = u_2 = \alpha$ and $(s_1', s_2', s_3') = (s_1, s_2, s_3) = (c, gc, g^2c)$. So now Equation 4 (and 6) is:

$$\Delta(gc, g^2c, g^3c) = (g\alpha, (c, gc, g^2c))$$

And Equation 3 is solved:

$$\Delta(f(gc, c), f(g^2c, gc), f(g^3c, g^2c)) = (f(g\alpha, \alpha), (c, gc, g^2c))$$

It is worth to note that the Δ-vector is already a grammar $U \circ_\alpha S$, but with the particularity of having only one term in the set U (represented by $f(u_1, ..., u_m)$). Thus, $P = \{\tau \to f(u_1, ..., u_m)\} \cup \{\alpha \to s_i | s_i \in S\}$.

Definition 13. A grammar $U \circ_\alpha S$ is called *simple* if the set U contains only one term and it is called *trivial* if it is simple and $U = \{\alpha\}$, i.e., $\tau \to \alpha$ is the only derivation from the start symbol.

Observe that the Δ-vector computes only *simple* grammars. In the following sections we show how to combine these grammars to obtain more complex ones (Section 5.2) and how to find valid grammars that will generate all the terms from T (Section 5.3).

5.2. Δ-table

The Δ-table is a data-structure that stores the *non-trivial* simple grammars $U \circ_\alpha S$ computed by applying the Δ-vector exhaustively to sub-sequences of T.

The motivation behind the exhaustive procedure of computing the Δ-vector of all possible sub-sequences is to obtain the best possible compression for the final grammar, i.e., $U \circ_\alpha S$ such that $|U| + |S|$ is the least possible (and, of course, less than $|T|$). Let us illustrate the situation.

Given a sequence of terms $T = (t_1, ..., t_n)$, the Δ-vector of this sequence is:

$$\Delta(t_1, ..., t_n) = (u_\alpha, (s_1, ..., s_n))$$

in which u_α is the biggest common term of all t_i parametrized with some variable α such that, replacing this variable with each s_i would yield the original set T. As we said before, this is a grammar G with $P = \{\tau \to u_\alpha\} \cup \{\alpha \to s_i \mid 1 \leqslant i \leqslant n\}$, such that $L(G) = T$, but it's not a good one. Observe that U would have only one term (u_α) and S has the same number of terms as the input, i.e., n. So $|G| = n + 1$, which is bigger than $|T|$. Since we are interested in finding grammars that compress the size of the term sequence, this is not a good choice.

By computing the Δ-vector of sub-sequences of T and combining them, we can obtain such a compression. To make this clearer, consider the term sequence:

$$T = (f(c, gc), f(c, g^2c), f(c, g^3c), f(gc, c), f(g^2c, gc), f(g^3c, g^2c))$$

The Δ-vector of this sequence is the following trivial grammar:

$$\alpha \circ_\alpha (f(c, gc), f(c, g^2c), f(c, g^3c), f(gc, c), f(g^2c, gc), f(g^3c, g^2c))$$

But if we take well-chosen subsets of this set, we obtain the pairs:

$$\Delta(f(c, gc), f(c, g^2c), f(c, g^3c)) = (f(c, g\alpha), (c, gc, g^2c))$$
$$\Delta(f(gc, c), f(g^2c, gc), f(g^3c, g^2c)) = (f(g\alpha, \alpha), (c, gc, g^2c))$$

Now let $U = (f(c, g\alpha), f(g\alpha, \alpha))$ and $S = (c, gc, g^2c)$. Note that $L(U \circ_\alpha S) = T$, and $|U| + |S| = 2 + 3 = 5$. In particular, the combination of the first term of U with the terms from S generates the first 3 elements of T, and the combination of the second term of U with the terms from S generates the last 3 elements from T.

The computation of these grammars is done incrementally, starting from sub-sequences of size 1 until n, where n is the size of the term sequence T. The results are stored in a map[3], called Δ-table. The values stored in this map are lists of pairs (u, T), where u is a term and T is a set of terms. They are indexed by another set of terms S s.t. the property $u \circ_\alpha S$ is fulfilled.

For example, let $T' \subset T$ and $\Delta(T') = (u, (s_1, ..., s_k))$. This information is stored in T's Δ-table with $S = (s_1, ..., s_k)$ as the key and (a list of) (u, T') as the value. Since there might be other sub-sequences T'' of T such that $\Delta(T'') = (u', S)$, it is necessary to store a list of pairs.

Algorithm 1 creates, fills and returns the Δ-table for a sequence of terms T. It is important to note that this is different from the naive algorithm, which would just compute and store the Δ-vectors of all sub-sequences of T. Algorithm 1 allows a significant pruning of the search space which is based on the following theorem:

Theorem 10. Let T be a set of terms. If $\Delta(T) = (\alpha, T)$ (trivial grammar), then $\Delta(T') = (\alpha, T')$ for every $T' \supset T$.

Proof. Follows from the definition of Δ-vector. □

Instead of computing the Δ-vector for all (exponentially many) sub-sequences of T, the algorithm computes the Δ-table for all sub-sequences which possess a non-trivial Δ-vector. This is achieved by starting from the empty sub-sequence and increasing the size of a sub-sequence by one element in each iteration. The algorithm does not try to extend sub-sequences having a trivial Δ-vector, a procedure whose completeness is guaranteed by Theorem 10 and which avoids large areas of the search space.

Another optimization of Algorithm 1 is regarding the choice of $t \in T \backslash T'$. Let T be the sequence $[t_1, t_2, t_3, ..., t_n]$. When augmenting the size of a sub-sequence by one element, for example, we might generate the set $\{t_1, t_2, t_3\}$ more than once. First by augmenting $[t_1, t_2]$ with t_3 and then by augmenting $[t_1, t_3]$ by t_2. In order to avoid this, we impose an order to these elements (hence treating T as a sequence in contrast to a set) and augment a sequence T' only with those elements of T that occur after all elements of T'.

[3]A map is a data structure that stores values indexed by keys.

Algorithm 1 Fill Δ-table for a sequence of terms T

```
function Δ-TABLE(T: sequence of terms)
    table ← new HashMap
    table[[]] ← [(null, [])]
    for i = 1 → T.length do
        for (u, T') ∈ table with T'.length = i − 1 do
            for t ∈ T\T' do
                (u', S) ← Δ-vector(T' + t)
                if u' ≠ α then
                    table[S] ← table[S] + (u', (T' + t))
                end if
            end for
        end for
    end for
    return table
end function
```

5.3. Finding valid grammars

After having filled the Δ-table, it is only a matter of combining the simple grammars in order to find a suitable one, i.e., a grammar G such that $L(G) = T$. The idea is the following: let $S \to [(u_1, T_1), ..., (u_r, T_r)]$ be one entry of T's Δ-table. We know that $T_i \subset T$ and that $\{u_i\} \circ S$ is a grammar G_i such that $L(G_i) = T_i$ for each $i \in \{1...r\}$. Take $\{T_{i_1}, ..., T_{i_s}\} \subset \{T_1, ..., T_r\}$ such that $T_{i_1} \cup ... \cup T_{i_s} = T$. Then, since combining each u_{i_j} with S yields T_{i_j}, and the union of these terms is T, the grammar $(u_{i_1}, ..., u_{i_s}) \circ_\alpha S$ will generate all terms from T. Note that there might be several combinations of T_i such that its union covers the set T, so it is often the case that different grammars are found, but only the minimal ones are considered as possible solutions.

Note that due to its construction, the Δ-table does not contain any trivial grammars. However, for obtaining all compressing grammars is can be necessary for some *parts* of the final grammar to be trivial. This is the case, for example, for the set $T = \{a, fa, f^2a, f^3a\}$. The Δ-table built for this set of terms is the following:

$$\begin{aligned} \{a, fa\} &\Rightarrow [(f\alpha, \{fa, f^2a\}), (f^2a, \{f^2a, f^3a\})] \\ \{a, f^2a\} &\Rightarrow [(f\alpha, \{fa, f^3a\})] \\ \{a, fa, f^2a\} &\Rightarrow [(f\alpha, \{fa, f^2a, f^3a\})] \end{aligned}$$

Observe that it is enough to add the trivial grammars $\{\alpha\} \circ T_i$ for all keys T_i of the Δ-table. Thus the new Δ-table of the example would be:

$$\begin{aligned} \{a, fa\} &\Rightarrow [(f\alpha, \{fa, f^2a\}), (f^2a, \{f^2a, f^3a\}), (\alpha, \{a, fa\})] \\ \{a, f^2a\} &\Rightarrow [(f\alpha, \{fa, f^3a\}), (\alpha, \{a, f^2a\})] \\ \{a, fa, f^2a\} &\Rightarrow [(f\alpha, \{fa, f^2a, f^3a\}), (\alpha, \{a, fa, f^2a\})] \end{aligned}$$

And from this, the grammar $\{\alpha, f\alpha\} \circ \{a, f^2a\}$ can be obtained.

5.4. Generalization to multiple cuts

In the previous sections it was explained how to compute a grammar $U \circ_\alpha S$ for a set of terms T. If the term set was extracted from a proof of a skolemized end-sequent, then this allows the introduction of one Π_1-cut ($\forall x.C$) in a proof of the same end-sequent. In this new proof, the end-sequent formulas will be instantiated with the terms from the set U of the grammar (these terms are prefixed with the function symbol f_i, which indicates of which formula these terms should be instances). The terms from the set S will be used to instantiate the cut formula when it occurs on the left, and α will be used as the eigenvariable of the cut formula on the right.

In order to obtain a grammar for constructing a proof with multiple cuts, all that is needed is to iterate the procedure described in the previous sections. Remember that, a totally rigid acyclic grammar for a proof with n cuts can be represented by[4]:

$$U \circ_{\alpha_n} S_n \ldots \circ_{\alpha_1} S_1$$

Given a sequence of terms T, on the first step the algorithm will compute a grammar $U_1 \circ_{\alpha_1} S_1$ such that it generates (and compresses) T. Then, the algorithm is run again, now with input U_1 (and with α_1 considered as a constant), in an attempt to compress even more the term set. Suppose that a grammar $U_2 \circ_{\alpha_2} S_2$ was found for U_1, such that it still compresses this term set. Now we have the grammar $U_2 \circ_{\alpha_2} S_2 \circ_{\alpha_1} S_1$ that generates the original term set T. This procedure can continue until we reach a term set U_n that cannot be compressed anymore via a grammar. At this moment we stop computing grammars, and we can compose them all to generate T. This procedure is illustrated in Figure 2.

$$\begin{aligned} T &= L(U_1 \circ_{\alpha_1} S_1) \\ U_1 &= L(U_2 \circ_{\alpha_2} S_2) \\ &\vdots \\ U_{n-1} &= L(U_n \circ_{\alpha_n} S_n) \end{aligned}$$

Figure 2: For more than one cut, iterate the grammars.

The algorithm to compute grammars described in the previous sections will be henceforth referred to as GC.

5.5. Example

In this section we show the computation of the grammars for an actual cut-free proof. For readability reasons we do not show the computation of every Δ-vector nor the full Δ-table, but only those relevant to find a minimal grammar.

The following sequent:

$$P(0,0), \forall x \forall y(P(x,y) \supset P(x,sy)), \forall x \forall y(P(x,y) \supset P(sx,y)) \rightarrow P(s^4 0, s^4 0)$$

has a cut-free proof whose Herbrand-sequent is:

[4]Note that this representation has the indices reversed from the usual representation used so far. This is only because, in practice, the number n in which the iteration stops is not known in advance.

$$
\begin{array}{lll}
P(0,0), & & \\
(P(0,0) \supset P(s0,0)), & & \\
(P(s0,0) \supset P(s0,s0)), & & \\
(P(s0,s0) \supset P(s^2 0,s0)), & & \\
(P(s^2 0,s0) \supset P(s^2 0,s^2 0)), & \rightarrow & P(s^4 0,s^4 0) \\
(P(s^2 0,s^2 0) \supset P(s^3 0,s^2 0)), & & \\
(P(s^3 0,s^2 0) \supset P(s^3 0,s^3 0)), & & \\
(P(s^3 0,s^3 0) \supset P(s^4 0,s^3 0)), & & \\
(P(s^4 0,s^3 0) \supset P(s^4 0,s^4 0)) & &
\end{array}
$$

Let $F_1 = \forall x \forall y (P(x,y) \supset P(x,sy))$ and $F_2 = \forall x \forall y (P(x,y) \supset P(sx,y))$. Then, the tuple term sets T_1 and T_2 of respectively F_1 and F_2 are extracted:

$$
\begin{array}{lll}
T_1 & = & ((s0,0),(s^2 0,s0),(s^3 0,s^2 0),(s^4 0,s^3 0)) \\
T_2 & = & ((0,0),(s0,s0),(s^2 0,s^2 0),(s^3 0,s^3 0))
\end{array}
$$

As in Definition 11 we will use two fresh function symbols f_1 and f_2, both of arity 2, to build one set of terms from the tuples of terms used to instantiate the formulas F_1 and F_2 respectively:

$$
\begin{array}{lll}
T = \{ & f_1(s0,0), f_1(s^2 0,s0), f_1(s^3 0,s^2 0), f_1(s^4 0,s^3 0), \\
& f_2(0,0), f_2(s0,s0), f_2(s^2 0,s^2 0), f_2(s^3 0,s^3 0)\}
\end{array}
$$

This term set has size 8, and our goal is to find a grammar $U \circ_\alpha S$ such that $|U| + |S| < 8$. The first step of the algorithm is to fill the Δ-table by computing all non-trivial Δ-vectors of subsets of T. In particular, these two Δ-vectors are computed from the first four and last four elements of T:

$$
\begin{array}{rcl}
\Delta(f_1(s0,0), f_1(s^2 0,s0), f_1(s^3 0,s^2 0), f_1(s^4 0,s^3 0)) & = & (f_1(s\alpha,\alpha),(s^3 0,s^2 0,s0,0)) \\
\Delta(f_2(0,0), f_2(s0,s0), f_2(s^2 0,s^2 0), f_2(s^3 0,s^3 0)) & = & (f_2(\alpha,\alpha),(s^3 0,s^2 0,s0,0))
\end{array}
$$

These will be stored in the Δ-table, which thus will have the following entry:

$$
\begin{array}{lll}
\{s^3 0,s^2 0,s0,0\} \Rightarrow & [(f_1(s\alpha,\alpha),\{f_1(s0,0), f_1(s^2 0,s0), f_1(s^3 0,s^2 0), f_1(s^4 0,s^3 0)\}), \\
& (f_2(\alpha,\alpha),\{f_2(0,0), f_2(s0,s0), f_2(s^2 0,s^2 0), f_2(s^3 0,s^3 0)\})]
\end{array}
$$

Given these entries, the algorithm finds the following grammar that generates T:

$$
\{f_1(s\alpha,\alpha), f_2(\alpha,\alpha)\} \quad \circ \quad \{s^3 0,s^2 0,s0,0\}
$$

In fact, for this example, the algorithm finds 31 grammars, of which 3 have the minimal size 6. From Theorems 2 and 5, we know that grammars of size l generate proofs π with Π_1-cuts such that $|\pi|_q = l$. Therefore, initially, all minimal grammars are equally good. In Section 8 we mention a method to decide which grammar to use in such a situation. For an example of the application of this algorithm that involves multiple cuts, please see Section 7.2.1.

6. Improving the canonical solution

After completing the first phase of cut-introduction, namely the computation of a grammar, the next step is to find a solution to the schematic extended Herbrand sequent induced by the grammar. Such a solution is guaranteed to exist by Lemma 3, and its construction is described in Definition 8. But is this solution optimal? If we approach this question from the point of view of the $|\cdot|_q$ measure, Theorem 2 shows that all solutions can be considered equivalent. From the point of view of symbolic complexity or logical complexity, things may be different: there are cases where the canonical solution is large, but small solutions exist. The following example exhibits such a case. In this example, a smaller solution not only exists, but is also more natural than (and hence in many applications preferable to) the canonical solution. The cut-formulas obtained in this example by forgetful resolution are those a human would come up with when given the end-sequent and the number of cuts to prove it with.

Example 5. Consider the sequents

$$S_n \quad \equiv \quad Pa, \forall x\,(Px \supset Pfx) \to Pf^{n^2}a.$$

Note that this is the example from Section 2 where 2^n is replaced by n^2. S_n has a (minimal) Herbrand-sequent

$$H_n \quad \equiv \quad Pa, Pa \supset Pfa, \ldots, Pf^{n^2-1}a \supset Pf^{n^2}a \to Pf^{n^2}a.$$

The terms of this Herbrand-sequent are generated by the grammar

$$\{\alpha, f\alpha, \ldots, f^{n-1}\alpha\} \circ \{a, f^n a, \ldots, f^{(n-1)n}a\}$$

which gives rise to the schematic extended Herbrand-sequent

$$X(\alpha) \supset \bigwedge_{i=0}^{n-1} X(f^{in}a), Pa, P\alpha \supset Pf\alpha, \ldots, Pf^{n-1}\alpha \supset Pf^n\alpha \to Pf^{n^2}a$$

and the canonical solution $\sigma = [X\backslash\lambda\alpha.C]$ with

$$C \quad \equiv \quad Pa \wedge \bigwedge_{i=0}^{n-1}(Pf^i\alpha \supset Pf^{i+1}\alpha) \wedge \neg Pf^{n^2}a.$$

But there also exists a solution θ of constant logical complexity and linear (instead of quadratic) symbol complexity by taking $\theta = [X\backslash\lambda\alpha.A]$ with

$$A \quad \equiv \quad P\alpha \supset Pf^n\alpha.$$

Since the solution for the schematic extended Herbrand sequent is interpreted as the lemmata that give rise to the proof with cuts, and these lemmata will in applications be read and interpreted by humans, it is important to consider the problem of improving the logical and symbolic complexity of the canonical solution. Furthermore, a decrease in the logical complexity of a lemma often yields a decrease in the length of the proof that is constructed from it.

In the following sections, we will describe a method which computes small solutions for schematic Herbrand sequents induced by grammars. The method will be abstract; it will depend on an algorithm $\mathcal{C}$ enumerating consequences of a formula. We describe two concrete consequence generators: one will be complete (but expensive), the other will be incomplete but less expensive.

We start by investigating the case of a single Π_1-cut in the subsequent Section 6.1 (some of these results have essentially been presented already in [20]). We then describe the two consequence generators. Finally, we present an approach to the simplification of the canonical solution for an arbitrary number of Π_1-cuts in Section 6.4.

For simplicity of presentation we will consider a fixed sequent

$$S \quad \equiv \quad \forall x\, F(x) \to$$

although the results can be extended to more general end-sequents as in Section 4. The problem of improving the canonical solution is a propositional one, hence in the sequel, α is to be interpreted as a constant symbol, $\models$ denotes the propositional consequence relation, and all formulas are quantifier-free unless otherwise noted.

6.1. Improving the solution of a single Π_1*-cut*

We start the study of the problem of the simplification of the canonical solution by looking at the case of *1-grammars* $U \circ V$, which give rise to proofs with a single Π_1-cut. In the setting of 1-grammars, a solution is of the form $[X\backslash\lambda\alpha.A]$. Throughout this section, we consider a fixed 1-grammar $U \circ V$, along with a schematic Herbrand sequent

$$H \quad \equiv \quad F[x\backslash u_1], \ldots, F[x\backslash u_m], X(\alpha) \supset \bigwedge_{j=1}^{k} X(s_j) \to$$

and its canonical solution

$$\sigma = [X\backslash\lambda\alpha.C] = [X\backslash\lambda\alpha.\bigwedge_{i=1}^{m} F[x\backslash u_i]].$$

We will use the abbreviation

$$\Gamma = F[x\backslash u_1], \ldots, F[x\backslash u_m].$$

If $[X\backslash\lambda\alpha.A]$ is a solution for H, we will say simply that A is a solution.

The first basic observation is that solvability is a semantic property. The following is an immediate consequence of Definition 7.

Lemma 5. Let A be a solution, B a formula and $\models A \Leftrightarrow B$. Then B is a solution.

Hence we may restrict our attention to solutions which are in *conjunctive normal form* (CNF). Formulas in CNF can be represented as sets of *clauses*, which in turn are sets of *literals*, i.e. possibly negated atoms. It is this representation that we will use throughout this section, along with the following properties: for sets of clauses A, B, $A \subseteq B$ implies $B \models A$, and for clauses C, D, $C \subseteq D$ implies $C \models D$.

Note that the converse of the Lemma above does not hold: given a solution A there may be solutions B such that $\not\models A \Leftrightarrow B$. We now turn to the problem of finding such solutions. In Example 5, we observe that that $C \models A$ (but $A \not\models C$). We can generalize this observation to show that the canonical solution is most general.

Lemma 6. Let C be the canonical solution and A an arbitrary solution. Then $C \models A$.

Proof. Since $\vartheta = [X\backslash\lambda\alpha.A]$ is a solution for H, $H\vartheta = F[x\backslash u_1], \ldots, F[x\backslash u_m], A \supset \bigwedge_{j=1}^{k} A[\alpha\backslash s_j] \rightarrow$ is valid. By definition, $C = \bigwedge_{i=1}^{m} F[x\backslash u_i]$, and therefore $C, A \supset \bigwedge_{j=1}^{k} A[\alpha\backslash s_j] \rightarrow$ is valid, hence $C \rightarrow A$ is valid. □

This result states that any search for simple solutions can be restricted to consequences of the canonical solution. Theoretically, we could simply enumerate "all" such consequences (there are, up to logical equivalence, only finitely many), but of course this is computationally infeasible. Towards a more efficient (but still complete!) iterative solution, we give a criterion that allows us to disregard some of those consequences.

Lemma 7. If $A \models B$ then

(1) If $A[\alpha\backslash s_1], \ldots, A[\alpha\backslash s_k], \Gamma \rightarrow$ is not valid, then B is not a solution.
(2) If A is a solution then $\Gamma \rightarrow B$ is valid.
(3) If A is a solution, then $B[\alpha\backslash s_1], \ldots, B[\alpha\backslash s_k], \Gamma \rightarrow$ is valid iff $[X\backslash\lambda\alpha.B]$ is a solution of H.

Proof. For (1), we will show the contrapositive. By assumption, $B[\alpha\backslash s_1], \ldots, B[\alpha\backslash s_k], \Gamma \rightarrow$ is valid. Since $A \models B$, we find that $A[\alpha\backslash s_1], \ldots, A[\alpha\backslash s_k], \Gamma \rightarrow$ is valid. For (2) it suffices to observe that since A is a solution $\Gamma \rightarrow A$ is valid, and to conclude by $A \models B$. (3) is then immediate by definition. □

Lemma 8 (Sandwich Lemma)**.** Let A, B be solutions and $A \models D \models B$. Then D is a solution.

Proof. By Lemma 7 (2), $\Gamma \rightarrow D$ is valid. By Lemma 7 (1), $D[\alpha\backslash s_1], \ldots, D[\alpha\backslash s_k], \Gamma \rightarrow$ is valid. □

Another observation to be made in Example 5 is that A only contains clauses that contain α. This observation can be generalized as well: we may freely delete α-free clauses from solutions.

Lemma 9. Let A be a solution in CNF and A' be obtained from A by removing all clauses that do not contain α. Then A' is a solution.

Proof. In this proof, we will denote $C[\alpha\backslash t]$ by $C(t)$ for clauses C and terms t. Note that since A is a solution, the sequent $s : \Gamma \rightarrow A$ is valid. Furthermore, note that the validity of

$$h : \Gamma, A' \supset \bigwedge_{j=1}^{k} A'(s_j) \rightarrow$$

follows from the validity of $h_1 : A'(s_1), \ldots, A'(s_k), A \to$ and $h_2 : A \to A'$ since $\Gamma \to A'$ can be derived from s and h_2, and $\Gamma, A'(s_1), \ldots, A'(s_k) \to$ can be derived from s and h_1.

For h_2, validity is clear since $A' \subseteq A$. Now let $A = \bigwedge_{i=1}^{l-1} C_i \wedge \bigwedge_{i=l}^{m} C_i$ such that C_i contains α if and only if $l \leqslant i \leqslant m$. Then $A' = \bigwedge_{i=l}^{m} C_i$. By Lemma 2 and the application of the invertible $\wedge_1$ rule, we know that

$$\bigwedge_{i=1}^{l-1} C_i(s_1), \bigwedge_{i=l}^{m} C_i(s_1), \ldots, \bigwedge_{i=1}^{l-1} C_i(s_k), \bigwedge_{i=l}^{m} C_i(s_k) \to$$

is valid. By assumption, $C_i(t) = C_i$ for all terms t and $1 \leqslant i < l$, so by contraction we derive

$$\bigwedge_{i=1}^{l-1} C_i, \bigwedge_{i=l}^{m} C_i(s_1), \ldots, \bigwedge_{i=l}^{m} C_i(s_k) \to$$

which implies h_1 since $A'(s_j) = \bigwedge_{i=l}^{m} C_i(s_j)$ and $A \to C_i$ is valid for all i, j. □

We have now established all the results required for our method. Before we describe it, we need some more notions. A solution A in CNF is called a *minimal solution* if it has minimal symbol complexity among all solutions in CNF.

Definition 14. Let $\mathcal{C}$ be an algorithm such that $\mathcal{C}(A)$ is a finite set of propositional consequences of a formula A (a *consequence generator*). We say that $\mathcal{C}$ *generates A for B* if either

1. $A \in \mathcal{C}(B)$ or
2. there exists $B' \in \mathcal{C}(B)$ such that $\mathcal{C}$ generates A for B'.

$\mathcal{C}$ is *complete w.r.t. A* if, for all minimal solutions B, $A \models B$ implies that $\mathcal{C}$ generates B for A. $\mathcal{C}$ is *well-founded* if there exists a well-founded order $>$ such that $\mathcal{C}(A) > \mathcal{C}(A')$ for $A' \in \mathcal{C}(A)$.

Let $\mathcal{C}$ be a consequence generator and A a solution. Algorithm 2 then describes the solution-finding algorithm $\mathrm{SF}_{\mathcal{C}}$.

Algorithm 2 $\mathrm{SF}_{\mathcal{C}}$

```
function SF_C(A: solution in CNF)
    A ← A without α-free clauses
    S ← {A}
    for B ∈ C(A) do
        if B[α\s_1], ..., B[α\s_k], Γ → is valid then        ▷ B is a solution
            S ← S ∪ SF_C(B)
        end if
    end for
    return min(S)
end function
```

Theorem 11. Let $\mathcal{C}$ be a consequence generator, let C be the canonical solution in CNF, and let F be a minimal solution in CNF. If $\mathcal{C}$ is complete w.r.t. C, then $F \in \mathrm{SF}_{\mathcal{C}}(C)$. If $\mathcal{C}$ is well-founded, then $\mathrm{SF}_{\mathcal{C}}$ terminates on every input.

Proof. Termination is trivial. For completeness, note that α-free clauses can be removed from C by Lemma 9. Since C is a solution, for $B \in \mathcal{C}(C)$ it suffices to check by Lemma 7 (3) whether $B[\alpha\backslash s_1], \ldots, B[\alpha\backslash s_k], \Gamma \to$ is valid to determine whether B is a solution. If it is not valid, then we know by Lemma 7 (1) that no iteration of $\mathcal{C}$ on B will yield a solution. Since $\mathcal{C}$ is complete w.r.t. C, all minimal solutions will be generated. □

6.2. Simplification by deductive closure

In this and the following section, we describe two concrete consequence generators. Both will be based on the propositional resolution rule which we now recall. Let $A = \{C_i \mid 1 \leqslant i \leqslant n\}$ be a formula in CNF with clauses $C_i = \{L_{i,j} \mid 1 \leqslant j \leqslant n_i\}$, where the $L_{i,j}$ are literals. By $\overline{L}$ we denote the dual of a literal L. For two clauses C_i, C_j, if there exists exactly one pair (k, l) such that $L_{i,k} = \overline{L_{j,l}}$, we define their *propositional resolvent* as the clause

$$\mathrm{res}(C_i, C_j) = (C_i \backslash L_{i,k}) \cup (C_j \backslash L_{j,l})$$

and leave $\mathrm{res}(C_i, C_j)$ undefined otherwise. We define the *deductive closure* $\mathcal{D}(A)$ as the least superset of A such that for all $C_1, C_2 \in \mathcal{D}(A)$ there exists a $C \in \mathcal{D}(A)$ such that $C \subseteq \mathrm{res}(C_1, C_2)$. It is well-known that $\mathcal{D}(A)$ is finite and can be computed from A by repeated application of $\mathrm{res}(\cdot, \cdot)$. Finally, we define the *subset consequence generator*

$$\mathcal{S}(A) = \{B \subset A \mid |B| = |A| - 1\}.$$

where A, B are sets of clauses. Towards showing completeness of $\mathcal{S}$, we recall (a consequence of) a result from [28]:

Theorem 12. Let A be a formula in CNF and C be a non-tautological clause such that $A \models C$. Then there exists $C' \in \mathcal{D}(A)$ such that $C' \subseteq C$.

Theorem 13. Let C be the canonical solution. Then $\mathcal{S}$ is well-founded and complete w.r.t. $\mathcal{D}(C)$. Hence $F \in \mathrm{SF}_{\mathcal{S}}(\mathcal{D}(C))$ for all minimal solutions F, and $\mathrm{SF}_{\mathcal{S}}$ always terminates.

Proof. $\mathcal{S}$ is trivially well-founded. Now let $\mathcal{D}(C) \models B$ with $B = \{B_i \mid 1 \leqslant i \leqslant n\}$ a minimal solution. Since $\mathcal{D}(C)$ is logically equivalent to C, $C \models B$ and by Theorem 12, there exists $B' = \{B'_1, \ldots, B'_n\} \subseteq \mathcal{D}(C)$ such that $B'_i \subseteq B_i$ and hence $C \models B' \models B$. By Lemma 8, B' is a solution, and $B' = B$ by minimality. We conclude by observing that $\mathcal{S}$ generates all $A \subseteq \mathcal{D}(C)$ for $\mathcal{D}(C)$, hence B in particular. □

6.3. Simplification by forgetful resolution

This section proposes another particular consequence generator that will yield a more practical but incomplete algorithm based on the *forgetful resolution operator* which resolves two clauses and then "forgets" them. Letting $A = \{C_i \mid 1 \leqslant i \leqslant n\}$ we define

$$\mathcal{F}(A) = \{\{\mathrm{res}(C_i, C_j)\} \cup (A \backslash \{C_i, C_j\}) \mid 1 \leqslant i < j \leqslant n, \mathrm{res}(C_i, C_j) \text{ defined}\}.$$

Resolution is sound, hence $\mathcal{F}$ is a consequence generator yielding the algorithm $\mathrm{SF}_{\mathcal{F}}$. Furthermore, note that if $A' \in \mathcal{F}(A)$ then $|A| > |A'|$, hence $\mathcal{F}$ is well-founded and $\mathrm{SF}_{\mathcal{F}}$ terminates on all inputs by Theorem 11.

We conclude our investigation of the improvement of the canonical solution in the case of 1-grammars by applying $\mathrm{SF}_{\mathcal{F}}$ to a concrete example.

Example 6. Consider the sequent S_n from Example 5 for $n = 2$. The canonical solution of S_2, written in conjunctive normal form, is

$$C \quad \equiv \quad Pa \wedge (\neg P\alpha \vee Pf\alpha) \wedge (\neg Pf\alpha \vee Pf^2\alpha) \wedge \neg Pf^4a.$$

Application of Lemma 9 yields

$$C' \quad \equiv \quad (\neg P\alpha \vee Pf\alpha) \wedge (\neg Pf\alpha \vee Pf^2\alpha).$$

We have $\mathcal{F}(C') = \{\neg P\alpha \vee Pf^2\alpha\}$. By (2) of Lemma 7, it suffices to check whether

$$Pa, \neg Pa \vee Pf^2a, \neg Pf^2a \vee Pf^4a \to Pf^4a$$

is valid, which is the case. Since $\mathcal{F}(\neg P\alpha \vee Pf^2\alpha) = \varnothing$, search terminates and $\mathrm{SF}_{\mathcal{F}}$ has found the smaller solution $\neg P\alpha \vee Pf^2\alpha$.

6.4. Improving the solution of multiple Π_1-cuts

This section is concerned with finding small solutions in the setting of grammars $U \circ S_1 \circ \cdots \circ S_n$, i.e. in the setting of introduction of n cuts. As in the previous section, the problem of finding a minimal solution is trivially decidable — our aim here is to find an algorithm that traverses the search space in a manner that takes into account the simplifying results we have established so far.

The algorithm we present will be incomplete independently of whether $\mathcal{C}$ is complete, but it will avoid the problem of having to deal at once with all the components of the canonical solution. More precisely, our algorithm will be based on an iteration of the algorithm $\mathrm{SF}_{\mathcal{C}}$ for the 1-cut introduction problem presented in Section 6.1, and hence gives rise to two concrete algorithms by plugging in the consequence generators of Sections 6.2 and 6.3.

Before we start to describe the algorithm, we will make a short detour to define Herbrand-sequents for (some) non-prenex sequents. This is done since, even though we have fixed a particularly simple sequent S, such more general sequents will naturally appear in the description of the algorithm. For the remainder of this section, for notational simplicity we will write $F_i(t)$ for $F_i[\alpha_i \backslash t]$. So consider sequents of the form

$$M \quad \equiv \quad F_1(\alpha_1) \supset \forall x_1 F_1(x_1), \ldots, F_n(\alpha_n) \supset \forall x_n F_n(x_n), \forall x F(x) \to$$

such that $x_i \notin \mathrm{V}(F_j(\alpha_j))$ for all $i, j \leqslant n$. M is logically equivalent to the sequent

$$M' \quad \equiv \quad \forall x_1 (F_1(\alpha_1) \supset F_1(x_1)), \ldots, \forall x_n (F_n(\alpha_n) \supset F_n(x_n)), \forall x F(x) \to$$

which has Herbrand sequents of the form

$$H' \quad \equiv \quad (F_1(\alpha_1) \supset F_1(s_1))_{s_1 \in S_1}, \ldots, (F_n(\alpha_n) \supset F_n(s_n))_{s_n \in S_n}, (F(s))_{s \in S} \to .$$

H' is logically equivalent to the sequent

$$H \quad \equiv \quad F_1(\alpha_1) \supset \bigwedge_{s_1 \in S_1} F_1(s_1), \ldots, F_n(\alpha_n) \supset \bigwedge_{s_n \in S_n} F_n(s_n), (F(s))_{s \in S} \to .$$

Hence we may identify M and M' and H and H' to be able to talk about sequents M and their Herbrand sequents H.

We are now ready to describe our algorithm. We fix a Herbrand-sequent $H = \bigwedge_{t \in T} F[x \backslash t] \to$ of our previously fixed sequent S such that $T = L(G)$ for the grammar $G = U \circ_{\alpha_1} S_1 \cdots \circ_{\alpha_n} S_n$.

Definition 15. We define a *k'th intermediary solution* to be a valid sequent of the form

$$F_n(\alpha_n) \supset \bigwedge_{s_n \in S_n} F_n(s_n), \ldots, F_\ell(\alpha_\ell) \supset \bigwedge_{s_\ell \in S_\ell} F_\ell(s_\ell), F(t)_{t \in T_\ell} \to$$

where $\ell = n - k + 1$ and $T_\ell = L(U \circ_{\alpha_1} S_1 \circ_{\alpha_2} \cdots \circ_{\alpha_{\ell-1}} S_{\ell-1})$.

By the discussion above, a k'th intermediary solution is a Herbrand-sequent of

$$F_n(\alpha_n) \supset \forall x_n F_n(x_n), \ldots, F_\ell(\alpha_\ell) \supset \forall x_\ell F_\ell(x_\ell), \forall x F(x) \to$$

Clearly, the 0'th intermediary solution is just the Herbrand-sequent $(F(t))_{t \in T} \to$, and an n'th intermediary solution is a valid sequent of the form

$$F_n(\alpha_n) \supset \bigwedge_{s_n \in S_n} F_n(s_n), \ldots, F_1(\alpha_1) \supset \bigwedge_{s_1 \in S_1} F_1(s_1), (F(u))_{u \in U} \to .$$

which is actually an extended Herbrand-sequent of S. In other words, the substitution $[X_i \backslash \lambda \alpha_i . F_i(\alpha_i)]_{i=1}^n$ solves the schematic extended Herbrand sequent induced by S and G. We show now how to obtain such an n'th intermediary solution by iteration of the results on the introduction of a single cut. As observed above, we have:

Lemma 10. There exists a 0'th intermediary solution.

Further, the following holds:

Lemma 11. If there exists a k'th intermediary solution I_k, then there exists a $(k+1)$'th intermediary solution I_{k+1}.

Proof. We use the notation from Definition 15 to denote I_k. It suffices to show that

$$D = (S_n, \dots, S_\ell, T_{\ell-1}) \circ_{\alpha_{\ell-1}} S_{\ell-1}$$

generates the terms of I_k, i.e. $L(D) = (S_n, \dots, S_\ell, T_\ell)$. Assume so, and consider the schematic extended Herbrand-sequent

$$F_n(\alpha_n) \supset \bigwedge_{s_n \in S_n} F_n(s_n), \dots, F_\ell(\alpha_\ell) \supset \bigwedge_{s_\ell \in S_\ell} F_\ell(s_\ell), X(\alpha_{\ell-1}) \supset \bigwedge_{s_{\ell-1} \in S_{\ell-1}} X(s_{\ell-1}), (F(t))_{t \in T_{\ell-1}} \to .$$

This is a schematic extended Herbrand sequent since I_k is valid and D generates its terms. Then by Lemma 4 there exists a solution $\sigma = \{\lambda \alpha_{\ell-1}.A\}$. Putting $F_{\ell-1} = A$ we have that

$$F_n(\alpha_n) \supset \bigwedge_{s_n \in S_n} F_n(s_n), \dots, F_{\ell-1}(\alpha_{\ell-1}) \supset \bigwedge_{s_{\ell-1} \in S_{\ell-1}} F_{\ell-1}(s_{\ell-1}), (F(t))_{t \in T_{\ell-1}} \to$$

is valid, which means that a $(k+1)$'th intermediary solution exists. We verify that indeed $L(D) = (S_n, \dots, S_\ell, T_\ell)$:

- For $n \geqslant i \geqslant \ell$, $L(S_i \circ_{\alpha_{\ell-1}} S_{\ell-1}) = S_i$ since $\alpha_{\ell-1} = \alpha_{n-k} \notin V(S_i)$ by the definition of grammar.
- $L(T_{\ell-1} \circ_{\alpha_{\ell-1}} S_{\ell-1}) = L((U \circ_{\alpha_1} S_1 \circ_{\alpha_2} \cdots \circ_{\alpha_{n-k-1}} S_{n-k-1}) \circ_{\alpha_{n-k}} S_{n-k}) = T_\ell$.

□

Hence there exists an n'th intermediary solution which yields a solution for the n-cut introduction problem. Before describing the algorithm $\mathrm{SF}^+_{\mathcal{C}}$, we note that a Herbrand sequent s and a grammar G for its termset induce a schematic Herbrand sequent in a canonical way (where the number of variables X_i depends on the grammar), we denote this schematic Herbrand sequent by $\mathrm{SHS}(s, G)$. In particular, if G is of the form $(U_1, \dots, U_n) \circ_\alpha V$, then $\mathrm{SHS}(s, G)$ contains exactly one schematic variable X_1. The above results (note that

Algorithm 3 $\mathrm{SF}^+_{\mathcal{C}}$

```
function SF^+_C(I_k: k'th intermediary solution in CNF)
    if k = n then
        return I_k
    end if
    ℓ ← n − k + 1
    C ← canonical solution of SHS(I_k, (S_n, ..., S_ℓ, T_{ℓ−1}) ∘_{α_{ℓ−1}} S_{ℓ−1})
    F_{ℓ−1} ← SF_C(C)
    I_{k+1} ← k + 1'th intermediary solution based on I_k and F_{ℓ−1}
    return SF^+_C(I_{k+1})
end function
```

by the proof of Lemma 11, *any* k'th intermediary solution can be used in the iteration) and observations, together with Theorem 11, entail the following.

Theorem 14. Let $\mathcal{C}$ be a consequence generator. If $F \in \mathrm{SF}^+_{\mathcal{C}}(H)$ then F is an extended Herbrand-sequent based on G (in particular, F gives rise to a solution to $\mathrm{SHS}(H, G)$). If $\mathcal{C}$ is well-founded, then $\mathrm{SF}^+_{\mathcal{C}}(H)$ terminates.

Using forgetful resolution, we obtain the concrete algorithm $\mathrm{SF}^+_{\mathcal{F}}$, which we illustrate by an example.

Example 7. Consider the example from Section 2 for $n = 3$. That is, we consider the sequent

$$Pa, \forall x\,(Px \supset Pfx) \to Pf^8a$$

which has a Herbrand sequent H with terms $T = \{a, fa, f^2a, \ldots, f^7a\}$. T is generated by the grammar $U \circ_{\alpha_1} S_1 \circ_{\alpha_2} S_2 = \{\alpha_1, f\alpha_1\} \circ \{\alpha_2, f^2\alpha_2\} \circ \{a, f^4a\}$.

From H, being the 0'th intermediary solution, we compute the first intermediary solution. To this end we consider the grammar $L(U \circ S_1) \circ S_2 = \{\alpha_2, f\alpha_2, f^2\alpha_2, f^3\alpha_2\} \circ \{a, f^4a\}$. This grammar leads to the schematic extended Herbrand sequent

$$X\alpha_2 \supset (Xa \wedge Xf^4a), Pa, P\alpha_2 \supset Pf\alpha_2, \ldots, Pf^3\alpha_2 \supset Pf^4\alpha_2 \to Pf^8a.$$

The canonical solution is $Pa \wedge (P\alpha_2 \supset Pf\alpha_2) \wedge \ldots \wedge (Pf^3\alpha_2 \supset Pf^4\alpha_2) \wedge \neg Pf^8a$ which, when subjected to the simplification procedure for the 1-cut introduction problem, becomes the solution

$$\neg P\alpha_2 \vee Pf^4\alpha_2.$$

Hence our first intermediary solution is

$$I_1 = (\neg P\alpha_2 \vee Pf^4\alpha_2) \supset \bigwedge_{s \in S_2} (\neg Ps \vee Pf^4s), Pa, P\alpha_2 \supset Pf\alpha_2, \ldots, Pf^3\alpha_2 \supset Pf^4\alpha_2 \to Pf^8a$$

which has the terms $(S_2, L(U \circ S_1))$ and is a Herbrand-sequent of

$$(\neg P\alpha_2 \vee Pf^4\alpha_2) \supset \forall x(\neg Px \vee Pf^4x), Pa, \forall x(Px \supset Pfx) \to Pf^8a.$$

We iterate the procedure to obtain the second intermediary solution. We consider the grammar $(S_2, U) \circ_{\alpha_1} S_1 = (\{a, f^4a\}, \{\alpha_1, f\alpha_1\}) \circ \{\alpha_2, f^2\alpha_2\}$ of the terms of I_1. We obtain the schematic extended Herbrand sequent

$$\begin{array}{l} X\alpha_1 \supset (X\alpha_2 \wedge Xf^2\alpha_2), Pa, (\neg P\alpha_2 \vee Pf^4\alpha_2) \supset \bigwedge_{s \in S_2} (\neg Ps \vee Pfs), \\ P\alpha_1 \supset Pf\alpha_1, Pf\alpha_1 \supset Pf^2\alpha_1 \to Pf^8a \end{array}$$

which has the canonical solution

$$Pa \wedge ((\neg P\alpha_2 \vee Pf^4\alpha_2) \supset \bigwedge_{s \in S_2} (\neg Ps \vee Pfs)) \wedge (P\alpha_1 \supset Pf\alpha_1) \wedge (Pf\alpha_1 \supset Pf^2\alpha_1) \wedge \neg Pf^8a$$

which simplifies to

$$\neg P\alpha_1 \vee Pf^2\alpha_1.$$

We finally obtain as the second intermediary solution the valid sequent

$$\begin{array}{rl} I_2 = & (\neg P\alpha_1 \vee Pf^2\alpha_1) \supset \bigwedge_{s_1 \in S_1} (\neg Ps_1 \vee Pf^2s_1), \\ & (\neg P\alpha_2 \vee Pf^4\alpha_2) \supset \bigwedge_{s_2 \in S_2} (\neg Ps_2 \vee Pf^4s_2), \\ & Pa, (Pu \supset Pfu)_{u \in U} \to Pf^8a \end{array}$$

which is an extended Herbrand-sequent of s and induces the cut-formulas $\forall x(\neg Px \vee Pf^2x)$ and $\forall x(\neg Px \vee Pf^4x)$. This can also be seen from the sequent for which I_2 is a Herbrand sequent:

$$\begin{array}{l} (\neg P\alpha_1 \vee Pf^2\alpha_1) \supset \forall x_1(\neg Px_1 \vee Pf^2x_1), (\neg P\alpha_2 \vee Pf^4\alpha_2) \supset \forall x_2(\neg Px_2 \vee Pf^4x_2), \\ Pa, \forall x(Px \supset Pfx) \to Pf^8a. \end{array}$$

7. The Method CI

In Section 3 we have shown that, for any rigid acyclic tree grammar G generating the set of Herbrand terms T of a cut-free proof φ of a sequent S, there exists a solution of the corresponding schematic extended Herbrand sequent H, the canonical solution. This canonical solution yields cut formulas and an extended Herbrand sequent H^* of S. By Theorem 2 we can construct a proof φ^* with cuts from H^*. In Section 6 we have presented several techniques to reduce the length of φ^* (i.e. the logical complexity) under preservation of the quantifier complexity. In combining all these transformations in a systematic way we obtain a nondeterministic algorithm $\mathrm{CI}(A, \mathcal{C})$ where A is an algorithm computing a minimal grammar; we may choose the algorithm GC in Section 5 or the straightforward nondeterministic guessing algorithm GG. $\mathcal{C}$ is one of the consequence generators defined Section 6; e.g., we may choose $\mathcal{C} = \mathcal{S}$ or $\mathcal{C} = \mathcal{F}$.

We now define $\mathrm{CI}(A, \mathcal{C})$:

Input: a cut-free proof φ of a Σ_1-sequent S.

(1) Compute the set of Herbrand terms T of φ.

(2) Compute a minimal rigid acyclic tree grammar G with $L(G) = T$ by A.

(3) Construct the canonical solution corresponding to G.

(4) Improve the canonical solution by $\mathrm{SF}^+_{\mathcal{C}}$.

(5) Construct the proof with the computed cut-formulas.

Note that, also without step (4), we obtain a full cut-introduction procedure.

7.1. Inverting Cut-Elimination

In this section we show that the method CI is complete by proving that – in a suitable sense – it constitutes an inversion of Gentzen's procedure for cut-elimination. To that aim we will strongly rely on the results of [21, 22] which show that the language of a grammar is an invariant of cut-elimination. To be more precise we quickly repeat the most central notions from [21, 22] here, for full details the interested reader is referred to these papers.

Definition 16. We denote with $\rightsquigarrow$ the cut-reduction relation defined by allowing the application of the standard reduction rules without any strategy-restriction. The standard reductions include rules such as e.g.

$$\dfrac{\dfrac{\overset{(\pi_1)}{\Gamma \to \Delta, A[x\backslash\alpha]}}{\Gamma \to \Delta, \forall x\, A}\,\forall_{\mathrm{r}} \quad \dfrac{\overset{(\pi_2)}{A[x\backslash t], \Pi \to \Lambda}}{\forall x\, A, \Pi \to \Lambda}\,\forall_{\mathrm{l}}}{\Gamma, \Pi \to \Delta, \Lambda}\,\mathrm{cut} \quad \mapsto \quad \dfrac{\overset{(\pi_1[\alpha\backslash t])}{\Gamma \to \Delta, A[x\backslash t]} \quad \overset{(\pi_2)}{A[x\backslash t], \Pi \to \Lambda}}{\Gamma, \Pi \to \Delta, \Lambda}\,\mathrm{cut} \quad ,$$

see [21, Figure 1] for the complete list. With $\overset{ne}{\rightsquigarrow}$ we denote the *non-erasing part* of $\rightsquigarrow$, i.e. we disallow application of the reduction rule

$$\dfrac{\dfrac{\overset{(\pi_1)}{\Gamma \to \Delta}}{\Gamma \to \Delta, A}\,\mathrm{w_r} \quad \overset{(\pi_2)}{A, \Pi \to \Lambda}}{\Gamma, \Pi \to \Delta, \Lambda}\,\mathrm{cut} \quad \mapsto \quad \dfrac{\overset{(\pi_1)}{\Gamma \to \Delta}}{\Gamma, \Pi \to \Delta, \Lambda}\,\mathrm{w}^*$$

and its symmetric variant that removes a $\mathrm{w_l}$-inference.

Definition 17. A proof is called *simple* if every cut is of one of the following forms:

$$\dfrac{\dfrac{\Gamma \to \Delta, A[x\backslash\alpha]}{\Gamma \to \Delta, \forall x\, A}\,\forall_{\mathrm{r}} \quad \forall x\, A, \Pi \to \Lambda}{\Gamma, \Pi \to \Delta, \Lambda}\,\mathrm{cut} \qquad \text{or} \qquad \dfrac{\Gamma \to \Delta, \exists x\, A \quad \dfrac{A[x\backslash\alpha], \Pi \to \Lambda}{\exists x\, A, \Pi \to \Lambda}\,\exists_{\mathrm{l}}}{\Gamma, \Pi \to \Delta, \Lambda}\,\mathrm{cut} \qquad \text{or}$$

$$\dfrac{\Gamma \to \Delta, A \quad A, \Pi \to \Lambda}{\Gamma, \Pi \to \Delta, \Lambda}\,\mathrm{cut}$$

where A is quantifier-free.

Each proof with $\Pi_1 \cup \Sigma_1$-cuts can be pruned to obtain a simple proof by permuting $\forall_{\mathrm{r}}$- and $\exists_{\mathrm{l}}$-inferences down and identifying their eigenvariables when needed. All of the reductions of $\rightsquigarrow$ preserve simplicity with the exception of the following situation:

$$\dfrac{\Gamma \to \Delta, A \quad \dfrac{\dfrac{A, \Pi \to \Lambda, B[x\backslash\alpha]}{A, \Pi \to \Lambda, \forall x\, B}\,\forall_{\mathrm{r}} \quad \forall x\, B, \Sigma \to \Theta}{A, \Pi, \Sigma \to \Lambda, \Theta}\,\mathrm{cut}}{\Gamma, \Pi, \Sigma \to \Delta, \Lambda, \Theta}\,\mathrm{cut}$$

$\rightsquigarrow$

$$\dfrac{\dfrac{\Gamma \to \Delta, A \quad \dfrac{A, \Pi \to \Lambda, B[x\backslash\alpha]}{A, \Pi \to \Lambda, \forall x\, B}\,\forall_{\mathrm{r}}}{\Gamma, \Pi \to \Delta, \Lambda, \forall x\, B}\,\mathrm{cut} \quad \forall x\, B, \Sigma \to \Theta}{\Gamma, \Pi, \Sigma \to \Delta, \Lambda, \Theta}\,\mathrm{cut}$$

where the order of the two cuts is exchanged which motivates the following definition.

Definition 18. A *reduction sequence of simple proofs* as one where the above reduction is directly followed by permuting down the $\forall_{\mathrm{r}}$-inference in order to arrive at:

$$\dfrac{\dfrac{\Gamma \to \Delta, A \quad A, \Pi \to \Lambda, B[x\backslash\alpha]}{\Gamma, \Pi \to \Delta, \Lambda, B[x\backslash\alpha]}\,\mathrm{cut}}{\dfrac{\Gamma, \Pi \to \Delta, \Lambda, \forall x\, B}{\Gamma, \Pi, \Sigma \to \Delta, \Lambda, \Theta}\,\mathrm{cut}}\,\forall_{\mathrm{r}} \quad \forall x\, B, \Sigma \to \Theta$$

and symmetrically for the case of $\exists_{\mathrm{l}}$.

We can now state the part of the main result of [22] which is relevant for this paper:

Theorem 15. If $\pi \overset{ne}{\rightsquigarrow} \pi^*$ is a cut-reduction sequence of simple proofs, then $L(G(\pi)) = L(G(\pi^*))$.

Proof. This is the second part of Theorem 7.2 in [22]. □

Definition 19. We write $\pi_1 \approx \pi_2$ if π_1 and π_2 have the same end-sequent and $G(\pi_1) = G(\pi_2)$.

Lemma 12. $\approx$ is an equivalence relation.

Proof. This follows directly from the definition of $\approx$. □

Lemma 13. For simple π_1, π_2 we have $\pi_1 \approx \pi_2$ iff the schematic extended Herbrand-sequents of π_1 and π_2 are identical.

Proof. The left-to-right direction follows from the fact that a schematic extended Herbrand-sequent is uniquely defined by a grammar and the end-sequent. The right-to-left direction follows from being able to read off the grammar from a schematic extended Herbrand-sequent. □

We write $[\pi]$ for the $\approx$-equivalence class of π, i.e. $[\pi] = \{\psi \text{ simple proof} \mid \psi \approx \pi\}$. For a set P of simple proofs we write ${}^{P}\!/_{\approx}$ for $\{[\pi] \mid \pi \in P\}$. This notation will, in particular, be used for the set of results of a non-deterministic algorithm such as $\mathrm{CI}(A, \mathcal{C})$. Consequently, ${}^{\mathrm{CI}(A,\mathcal{C})(\pi^*)}\!/_{\approx}$ is the set of $\approx$-equivalence classes reachable by applying $\mathrm{CI}(A, \mathcal{C})$ to π^*. We can now state the inversion theorem:

Theorem 16. If $\pi \overset{ne}{\rightsquigarrow} \pi^*$ is a cut-reduction sequence of simple proofs and π^* is cut-free, then $[\pi] \in {}^{\mathrm{CI}(\mathrm{GG},\mathcal{C})(\pi^*)}\!/_{\approx}$ for any well-founded consequence generator $\mathcal{C}$.

Proof. By definition of the Herbrand-sequent we have

$$H(\pi^*) = \{F_i[\bar{x}\backslash\bar{t}] \mid 1 \leqslant i \leqslant p, f_i(\bar{t}) \in L(G(\pi^*))\} \to \{F_i[\bar{x}\backslash\bar{t}] \mid p < i \leqslant q, f_i(\bar{t}) \in L(G(\pi^*))\}.$$

By Theorem 15 we have $L(G(\pi^*)) = L(G(\pi))$ and hence

$$H(\pi^*) = \{F_i[\bar{x}\backslash\bar{t}] \mid 1 \leqslant i \leqslant p, f_i(\bar{t}) \in L(G(\pi))\} \to \{F_i[\bar{x}\backslash\bar{t}] \mid p < i \leqslant q, f_i(\bar{t}) \in L(G(\pi))\}.$$

Therefore $G(\pi)$ can be obtained from the grammar-guessing algorithm GG applied to $H(\pi^*)$. Let ψ be the proof obtained from the canonical solution of $G(\pi)$, then $\psi \in \mathrm{CI}(\mathrm{GG}, \mathcal{C})(\pi^*)$ (which is well defined by Theorem 14) and as $G(\psi) = G(\pi)$ we have $\psi \approx \pi$ and therefore $[\pi] \in {}^{\mathrm{CI}(\mathrm{GG},\mathcal{C})(\pi^*)}\!/_{\approx}$. □

Note that the consequence-generator $\mathcal{C}$ is not of relevance to the above theorem as it only concerns the propositional part of the cut-formulas which is factored out by $\approx$.

7.2. Proof Compression

We will prove in this section that application of CI to a sequence of cut-free proofs ϱ_n of sequents S_n can result in an exponential compression of ϱ_n. But we should take care with respect to which *complexity measure* the proofs are compressed. In fact, the steps (1)-(3) and (5) yield proofs χ_n of S_n with cuts such that $|\varrho_n|_q$ is exponential in $|\chi_n|_q$, resulting in an exponential compression of *quantifier complexity*. But these steps alone do not yield an exponential compression of *proof length* (taking into account *all* logical inferences). The compression of proof length is then achieved by using step (4) of the algorithm; step (5) then yields a sequence of proofs φ_n of the sequents S_n with cut s.t. $|\varrho_n|$ is exponential in $|\varphi_n|$.

As input we take a sequence of shortest cut-free proofs ϱ_n of the sequents

$$s_n \colon Pa, \forall x(Px \supset Pfx) \to Pf^{2^{n+1}}a.$$

from Section 2

7.2.1. Exponential compression of quantifier-complexity

We apply step (1) to ϱ_n and obtain a sequence of the corresponding minimal Herbrand sequents

$$s'_n \colon Pa, (Ps \supset Pfs)_{s \in T_n} \to Pf^{2^{n+1}}a,$$

where

$$T_n = \{a, fa, \ldots, f^{2^{n+1}-1}a\}.$$

Note that the quantifier complexity of any cut-free proof of S_n is $\geqslant 2^{n+1}$ and thus exponential in n.

We continue with step (2) using GC: Let n be a fixed (but arbitrary) number > 0. The sets T_n can be generated by the grammars

$$G_n \colon \{\alpha_1, f\alpha_1\} \circ_{\alpha_1} \{\alpha_2, f^2\alpha_2\} \circ_{\alpha_2} \cdots \circ \{\alpha_n, f^{2^{n-1}}\alpha_n\} \circ_{\alpha_n} \{a, f^{2^n}a\}.$$

where the α_i are variables and a is a constant symbol.

For describing the steps of the grammar computation algorithm GC we define

$$\begin{aligned} T_k &= \{\alpha_{k+1}, \ldots, f^{2^{k+1}-1}\alpha_{k+1}\} \text{ for } k = 0, \ldots, n \text{ and } \alpha_{n+1} = a, \\ S_k &= \{\alpha_{k+1}, f^{2^k}\alpha_{k+1}\} \text{ for } k = 0, \ldots, n \text{ and } \alpha_{n+1} = a. \end{aligned}$$

Note that $S_0 = T_0 = \{\alpha_1, f\alpha_1\}$.

We have assumed that $n > 0$. So GC starts with T_n and computes the grammar $T_{n-1} \circ S_n$. Assume now that the grammar

$$T_{n-k} \circ S_{n-k+1} \circ \cdots \circ_{\alpha_n} S_n$$

is already computed. If $k = n$ we are done; otherwise we have

$$T_{n-k} = \{\alpha_{n-k+1}, \ldots, f^{2^{n-k+1}-1}\alpha_{n-k+1}\}$$

which is decomposed via GC into $T_{n-k-1} \circ S_{n-k}$. Putting things together we obtain the grammar

$$T_{n-k-1} \circ S_{n-k} \circ S_{n-k+1} \circ \cdots \circ S_n.$$

Thus, eventually, we obtain the grammar

$$S_0 \circ \cdots \circ S_n.$$

for T_n which is just G_n.

We now move to step (3) and compute the canonical solution of the schematic extended Herbrand sequent H_n corresponding to G_n where

$$H_n = Pa, (Ps \supset Pfs)_{s \in S_0}, X_1\alpha_1 \supset \bigwedge_{s \in S_1} X_1 s, \ldots, X_n\alpha_n \supset \bigwedge_{s \in S_n} X_n s \to Pf^{2^{n+1}}a.$$

The canonical solution of H_n is

$$\theta_n \colon [X_1 \backslash \lambda\alpha_1.C_1 \mid, \ldots, X_n \backslash \lambda\alpha_n.C_n]$$

where

$$\begin{aligned} F &= Pa \wedge (Px \supset Pfx) \wedge \neg Pf^{2^{n+1}}a, \\ C_1 &= \bigwedge_{s \in S_0} F\{x \backslash s\}, \\ C_{i+1} &= \bigwedge_{s \in S_i} C_i\{\alpha_i \backslash s\} \text{ for } 1 \leqslant i < n. \end{aligned}$$

Now let $H_n^* \colon H_n\theta_n$ be the corresponding extended Herbrand sequent. Then $|H_n^*| = 2(n+1)$. By Theorem 2 we can construct proofs χ_n of S_n with $|\chi_n|_q = |H_n^*| = 2(n+1)$. As $|\varrho_n|_q \geqslant 2^{n+1}$ we obtain

$$|\varrho_n|_q \geqslant 2^{|\chi_n|_q/2},$$

and so $|\varrho_n|_q$ is exponential in $|\chi_n|_q$.

7.2.2. Exponential compression of proof length

Let

$$F_k = (\neg P\alpha_k \vee Pf^{2^k}\alpha_k) \supset \bigwedge_{s \in S_k} (\neg Ps \vee Pf^{2^k}s)$$

for $k = 1, \ldots, n$.

We will prove that using phase (4) of $\mathrm{CI}(\mathrm{GC}, \mathcal{F})$ (we apply forgetful resolution) we can obtain the extended Herbrand sequent

$$s_n^* \colon F_n, \ldots, F_1, Pa, (Ps \supset Pfs)_{s \in S_0} \to Pf^{2^{n+1}}a$$

from s_n and the grammar $S_0 \circ \cdots \circ S_n$ for T_n.

For the proof we need two auxiliary lemmas.

Lemma 14. Let $k \geqslant 1$. Then the formula $\neg P\alpha_{n-k+1} \vee Pf^{2^{n-k+1}}\alpha_{n-k+1}$ is derivable by forgetful resolution from $\bigwedge_{s\in T_{n-k}}(Ps \supset Pfs)$.

Proof. Let $G_k = \bigwedge_{s\in T_{n-k}}(Ps \supset Pfs)$. By definition of T_{n-k} G_k is the formula

$$(P\alpha_{n-k+1} \supset Pf\alpha_{n-k+1}) \wedge \cdots \wedge (Pf^{2^{n-k+1}-1}\alpha_{n-k+1} \supset Pf^{2^{n-k+1}}\alpha_{n-k+1}).$$

Its conjunctive normal form G'_k is

$$(\neg P\alpha_{n-k+1} \vee Pf\alpha_{n-k+1}) \wedge \cdots \wedge (\neg Pf^{2^{n-k+1}-1}\alpha_{n-k+1} \vee Pf^{2^{n-k+1}}\alpha_{n-k+1}).$$

By setting $x_i = Pf^i\alpha_{n-k+1}$ we obtain

$$G'_k = \bigwedge_{i=0}^{2^{n-k+1}-1}(\neg x_i \vee x_{i+1}).$$

The first step of forgetful resolution (resolving the first two clauses) gives us the CNF:

$$(\neg x_0 \vee x_2) \wedge \bigwedge_{i=2}^{2^{n-k+1}-1}(\neg x_i \vee x_{i+1}).$$

By repeating resolving the first two clauses we eventually obtain $\neg x_0 \vee x_{2^{n-k+1}}$ which is just $\neg P\alpha_{n-k+1} \vee Pf^{2^{n-k+1}}\alpha_{n-k+1}$. □

Lemma 15. Let $k < n$ and

$$s\colon X\alpha_{n-k} \supset \bigwedge_{s\in S_{n-k}} Xs, F_n, \ldots, F_{n-k+1}, Pa, (Ps \supset Pfs)_{s\in T_{n-k-1}} \to Pf^{2^{n+1}}a$$

be a schematic sequent. Then the substitution

$$[X\backslash\lambda x.(\neg Px \vee Pf^{2^{n-k}}x)]$$

is a solution of s.

Proof. We prove that $s'\colon s[X\backslash\lambda x.(\neg Px \vee Pf^{2^{n-k}})]$ is a valid sequent. s' is of the form

$$F_n, \ldots, F_{n-k+1}, F_{n-k}, Pa, (Ps \supset Pfs)_{s\in T_{n-k-1}} \to Pf^{2^{n+1}}a$$

where

$$F_{n-k} = (\neg P\alpha_{n-k} \vee Pf^{2^{n-k}}\alpha_{n-k}) \supset \bigwedge_{s\in S_{n-k}}(\neg Ps \vee Pf^{2^{n-k}}s).$$

By Lemma 14 we have $(Ps \supset Pfs)_{s\in T_{n-k-1}} \vdash \neg P\alpha_{n-k} \vee Pf^{2^{n-k}}\alpha_{n-k}$ by forgetful resolution. By modus ponens with F_{n-k} we obtain

$$(\neg P\alpha_{n-k+1} \vee Pf^{2^{n-k}}\alpha_{n-k+1}) \wedge (\neg Pf^{2^{n-k}}\alpha_{n-k+1} \vee Pf^{2^{n-k+1}}\alpha_{n-k+1})$$

from which, by resolution, we obtain

$$C_{n-k+1}\colon \neg P\alpha_{n-k+1} \vee Pf^{2^{n-k+1}}\alpha_{n-k+1}.$$

If $k = 0$ we obtain C_{n+1} and the sequent

$$(*)\ Pa, Pa \supset Pf^{2^{n+1}}a, \ldots, \to Pf^{2^{n+1}}a$$

which is valid.

If $k > 0$ then by modus ponens on C_{n-k+1} and F_{n-k+1} and resolving the result we obtain C_{n-k+2}, and so forth. Eventually we obtain C_{n+1} and $(*)$. □

Proposition 1. Let $k \leqslant n$. Then the solution-finding algorithm $\mathrm{SF}^+_{\mathcal{F}}$ constructs the intermediary solution

$$s^+_k \colon F_n, \ldots, F_{n-k+1}, Pa, (Ps \supset Pfs)_{s \in T_{n-k}} \to Pf^{2^{n+1}}a.$$

Proof. For $k = 0$ we just have the (valid) input sequent s_n and the Herbrand instances from T_n, trivially constructed by $\mathrm{SF}^+_{\mathcal{F}}$.

Assume that $0 < k < n$ and $\mathrm{SF}^+_{\mathcal{F}}$ has constructed the intermediary solution

$$s^+_k \colon F_n, \ldots, F_{n-k+1}, Pa, (Ps \supset Pfs)_{s \in T_{n-k}} \to Pf^{2^{n+1}}a$$

with Herbrand terms T_{n-k}. By definition of the S_i, T_i we have that $T_{n-k-1} \circ S_{n-k}$ is a grammar for T_{n-k}. Via this grammar we obtain the schematic extended Herbrand sequent

$$s \colon X\alpha_{n-k} \supset \bigwedge_{s \in S_{n-k}} Xs, F_n, \ldots, F_{n-k+1}, Pa, (Ps \supset Pfs)_{s \in T_{n-k-1}} \to Pf^{2^{n+1}}a.$$

As $V(T_{n-k-1}) = \{\alpha_{n-k}\}$ and α_{n-k} does not occur in $F_n, \ldots, F_{n-k+1}$, the improved canonical solution, according to Lemma 9, constructed by $\mathrm{SF}^+_{\mathcal{F}}$ is

$$[X \backslash \lambda x. \bigwedge_{s \in T_{n-k-1}} (Ps \supset Pfs)[\alpha_{n-k} \backslash x]].$$

By Lemma 14 forgetful resolution constructs the formula

$$\neg Px \vee Pf^{2^{n-k}}x \text{ from } \bigwedge_{s \in T_{n-k-1}} (Ps \supset Pfs)[\alpha_{n-k} \backslash x].$$

By using Lemma 15 we conclude that

$$[X \backslash \lambda x.(\neg Px \vee Pf^{2^{n-k}}x)]$$

is a solution of s yielding the sequent

$$F_n, \ldots, F_{n-k+1}, F_{n-k}, Pa, (Ps \supset Pfs)_{s \in T_{n-k-1}} \to Pf^{2^{n+1}}a.$$

□

Corollary 1. Let s^*_n be the schematic extended Herbrand sequent

$$X_n\alpha_n \supset \bigwedge_{s \in S_n} X_n s, \ldots, X_1\alpha_1 \supset \bigwedge_{s \in S_1} X_1 s, Pa, (Ps \supset Pfs)_{s \in S_0} \to Pf^{2^{n+1}}a$$

corresponding to s_n and the grammar G_n. Then $\mathrm{SF}^+_{\mathcal{F}}$ constructs a solution ϑ for s^*_n where

$$\vartheta = [X_1 \backslash \lambda x.(\neg Px \vee Pf^2 x), \ldots, X_n \backslash \lambda x.(\neg Px \vee Pf^{2^n}x)].$$

Proof. Obvious by Proposition 1 and by definition of the F_i. □

New we construct the proofs. We define a sequence of proofs φ_n (via ψ_n) in the following way:
ψ_0 is a cut-free proof of the sequent $\forall x(Px \supset Pfx) \to \forall x(Px \supset Pf^2x)$ and
$\psi_{k+1} =$

$$\dfrac{\begin{array}{cc}(\psi_k) & (\chi_{k+1}) \\ \forall x(Px \supset Pfx) \to \forall x(\neg Px \vee Pf^{2^{k+1}}x) & \forall x(\neg Px \vee Pf^{2^{k+1}}x) \to \forall x(\neg Px \vee Pf^{2^{k+2}}x)\end{array}}{\forall x(Px \supset Pfx) \to \forall x(\neg Px \vee Pf^{2^{k+2}}x)}\ \text{cut}$$

where the eigenvariables of ψ_k are $\alpha_1, \ldots, \alpha_{k+1}$ and χ_{k+1} is a cut-free proof of constant length with eigenvariable α_{k+2} and $\forall_1$-instances $\{\alpha_{k+2}, f^{2^{k+1}}\alpha_{k+2}\}$.

We can now define φ_n for $n > 0$:

$\varphi_n =$

$$\dfrac{\begin{array}{cc}(\psi_{n-1}) & (\sigma_n)\\ \forall x(Px \supset Pfx) \to \forall x(\neg Px \vee Pf^{2^n}x) & Pa, \forall x(\neg Px \vee Pf^{2^n}x) \to Pf^{2^{n+1}}a\end{array}}{Pa, \forall x(Px \supset Pfx) \to Pf^{2^{n+1}}a}\ \text{cut}$$

where σ_n is a constant length proof with Herbrand terms $\{a, f^{2^n}a\}$.

Proposition 2. Given a proof of s_n for $n > 0$ with minimal Herbrand sequent s'_n (and instances T_n) and the grammar $S_0 \circ \ldots \circ S_n$ for T_n, the proof construction algorithm PCA constructs the proof φ_n.

Proof. By Corollary 1 CI constructs the extended Herbrand sequent

$$F_n, \ldots, F_1, Pa, (Ps \supset Pfs)_{s\in S_0} \to Pf^{2^{n+1}}a$$

where

$$\begin{array}{lcl} F_1 & = & (\neg P\alpha_1 \vee Pf^2\alpha_1) \supset \bigwedge\limits_{s\in S_1}(\neg Ps \vee Pf^2 s)\\ \ldots & & \ldots\\ \ldots & & \ldots\\ F_n & = & (\neg P\alpha_n \vee Pf^{2^n}\alpha_k) \supset \bigwedge\limits_{s\in S_n}(\neg Ps \vee Pf^{2^n}s)\end{array}$$

From the F_i we read off the cut-formulas $\forall x(\neg Px \vee Pf^{2^i}x)$ with eigenvariable substitution $[x\backslash\alpha_i]$ for the left side of the cut and substitutions $[x\backslash\alpha_{i+1}], [x\backslash f^{2^i}\alpha_{i+1}]$ for the right side. Note that these are exactly the quantifier substitutions in the proofs ψ_i. For F_n the corresponding substitutions are $[x\backslash\alpha_n]$ and $[x\backslash a], [x\backslash f^{2^n}a]$, respectively, corresponding to the last cut in φ_n.

Now taking F_1 we construct the proof ψ_1 via $\forall x(Px \supset Pfx)$ (which occurs in the end-sequent) and the cut-formula $\neg Px \vee Pf^2x$ and via the substitutions $[x\backslash\alpha_1]$ for the eigenvariable of the cut, $\{[x\backslash s] \mid s \in S_0\}$ for the instantiations of $\forall x(Px \supset Pfx)$, $\{[x\backslash s] \mid s \in S_1\}$ for the right side of the cut. This gives us all quantifier-inferences of ψ_1; it remains to order the inferences appropriately to obtain ψ_1. Note that Pa on the left hand side of the end-sequent is not needed for constructing the proof.

Having constructed ψ_k $(k < n)$ with the cut-formulas $\neg Px \vee Pf^2x, \ldots, \neg Px \vee Pf^{2^k}x$ we have processed the formulas $F_1, \ldots, F_k$. Now

$$F_{k+1} = (\neg P\alpha_{k+1} \vee Pf^{2^{k+1}}\alpha_{k+1}) \supset \bigwedge_{s\in S_{k+1}}(\neg Ps \vee Pf^{2^{k+1}}s)$$

We know by construction that the last eigenvariable substitution in ψ_k is $[x\backslash\alpha_{k+1}]$ and the end-formula on the right hand side is $\forall x(\neg Px \vee Pf^{2^{k+1}}x)$, which by F_{k+1} is the next cut-formula. The instantiations for the right hand side of the cut are $\{[x\backslash s] \mid s \in S_{k+1}\}$. If $k+1 < n$ we have F_{k+2}, from which $\forall x(\neg Px \vee Pf^{2^{k+2}}x)$ is the cut-formula, which is also the end-formula of ψ_{k+1}. With this information we have all necessary quantifier substitutions and formulas to construct ψ_{k+1}. If $k+1 = n$ we have

$$F_n = (\neg P\alpha_n \vee Pf^{2^n}\alpha_n) \supset \bigwedge_{s\in\{a, f^{2^n}a\}}(\neg Ps \vee Pf^{2^n}s)$$

and thus $\{a, f^{2^n}a\}$ as substitutions of the right side of the cut. We also know the end sequent s_n. This information eventually yields the proof φ_n. □

Theorem 17. Let $(\rho_n)_{(n\in\mathbb{N})}$ be a sequence of shortest cut-free proofs of $(s_n)_{(n\in\mathbb{N})}$. Then $|\rho_n| > 2^n$ and there are constants a,b s.t. the cut-introduction algorithm, applied to ρ_n, constructs a sequence of proofs φ_n of s_n s.t. $|\varphi_n| \leqslant a * n + b$ for all $n \geqslant 2$.

Proof. We have $|\rho| > 2^n$ for any cut-free proof ρ of s_n and, in particular, for a shortest proof ρ_n. The Herbrand instances of shortest proofs of s_n must be T_n, as T_n is a minimal set of Herbrand terms for s_n, and therefore the Herbrand instances for ρ_n are just T_n. So the GC constructs the grammar $S_0 \circ \cdots \circ S_n$ of T_n. Then by Proposition 2 the algorithm constructs the sequence of proofs φ_n of s_n for $n \geqslant 2$. The length of φ_n is linear in n. In fact $|\varphi_n| = |\psi_{n-1}| + |\sigma_n| + 1$, where $|\sigma_n| = c$ for a constant c.

Moreover, $|\psi_0| = d$ for some constant d and $|\psi_{k+1}| = |\psi_k| + |\chi_{k+1}| + 1$, where $|\chi_{k+1}| = e$ for some constant e. So $|\psi_k| = d + k * (e + 1)$ and

$$|\varphi_n| = c + 1 + d + (n - 1) * (e + 1).$$

□

8. Implementation and Experiments

The algorithm described in this paper was implemented in the gapt-system[5] for introducing a single Π_1-cut into a sequent calculus proof. Gapt is a framework for implementing proof transformations written in the programming language Scala. It was initially developed for eliminating cuts of proofs by using resolution (CERES), but it has proven to be general enough so that other transformations could be implemented in it. In this section, we explain how to run the cut-introduction algorithm to compress a proof.

In order to install gapt, you need to have a Java Runtime Environment (JRE[6]) installed. Then, go to `http://www.logic.at/gapt` and the `gapt-cli-1.4.zip` file should be available in the "downloads" section. After uncompressing this file, you should see a directory where you can find the running script `cli.sh` and a README file. To run the program, just execute this script.

Gapt opens in a Scala interactive shell (`scala>`) from where you can run all the commands provided by the system. To see a list of them, type `help`. The commands are separated in categories, and we are interested in the ones listed under "Cut-Introduction" and "Proof Examples". In this list you can see the types of the functions and a brief description of what they do. Observe that there exist functions for each of the steps described in this paper for the introduction of cuts, and there is also a "cutIntro" command that does all steps automatically. Under "Proof Examples" there is a set of functions that generate (minimal) cut-free proofs of some parametrized end-sequents. For example, `LinearExampleProof(n)` will generate a cut-free proof of the end-sequent $P0, \forall x(Px \supset Psx) \to Ps^n0$ for some n.

We will use as input one of these proofs generated by the system, namely, `LinearExampleProof(9)`. But the user can also, for example, write his own proofs in `hlk`[7] and input these files to the system. Currently, there is also an effort to implement a parser for proofs obtained from the TPTP and SMT-LIB problem libraries so that large scale experiments can be carried out. Meanwhile, we use the motivating example (Section 2) for a brief demonstration.

First of all, we instantiate the desired proof and store this in a variable:

```
scala> val p = LinearExampleProof(9)
```

You will see that a big string representing the proof is printed. Gapt also contains a viewer for proofs and other elements [13]. You can open it to view a proof p at any moment with the command `prooftool(p)`. It is possible to see some information about a proof on the command line by calling:

[5] `http://www.logic.at/gapt/`
[6] `http://www.oracle.com/technetwork/java/javase/downloads/jre7-downloads-1880261.html`
[7] `http://www.logic.at/hlk/`

```
scala> printProofStats(p)
------------- Statistics ---------------
Cuts: 0
Number of quantifier rules: 9
Number of rules: 28
Quantifier complexity: 9
----------------------------------------
```

Now we need to extract the terms used to instantiate the $\forall$ quantifiers of the end-sequent:

```
scala> val ts = extractTerms(p)
```

The system indicates how many terms were extracted, which is nine for this case, as expected. The next step consists in computing grammars that generate this term set (Section 5):

```
scala> val grms = computeGrammars(ts)
```

The number of grammars found is shown, 693 in this case. They are ordered by size, and one can see the first ones by calling:

```
scala> seeNFirstGrammars(grms, 5)
```

This will print on the screen the first 5 grammars, and we can choose which one to use for compressing the proof, in this case we take the second one:

```
scala> val g = grms(1)
```

Given the end-sequent of the proof and a grammar, the extended Herbrand sequent can be computed:

```
scala> val ehs = generateExtendedHerbrandSequent(p.root, g)
```

As was shown in Lemma 3, the cut-introduction problem has a canonical solution:

```
scala> val cs = computeCanonicalSolution(p.root, g)
```

The canonical solution is then printed on the screen. The extended Herbrand sequent generated previously has the canonical solution as default, but this solution can be improved, as demonstrated in Section 6.

```
scala> minimizeSolution(ehs)
```

The user can see then that the canonical solution: $\forall x.((P(x) \supset P(s(x))) \wedge ((P(s(s(x))) \supset P(s(s(s(x))))) \wedge (P(s(x)) \supset P(s(s(x))))))$ is transformed into a simpler one: $\forall x.(P(s(s(s(x)))) \vee \neg P(x))$. Finally, the proof with cut is constructed:

```
scala> val fp = buildProofWithCut(ehs)
```

In order to compare this with the initial proof, one can again count the number of rules:

```
scala> printProofStats(fp)
------------- Statistics ---------------
Cuts: 1
Number of quantifier rules: 7
Number of rules: 25
Quantifier complexity: 6
----------------------------------------
```

We showed how to run the cut-introduction algorithm step by step. There is, though, a command comprising all these steps:

```
scala> val fp2 = cutIntro(p)
```

Name	End-sequent	Cut-formula	CR
`SquareDiagonal`	$P(0,0), \forall x,y.P(x,y) \supset P(s(x),y),$ $\forall x,y.P(x,y) \supset P(x,s(y))$ $\rightarrow P(s^8(0),s^8(0))$	$\forall x.P(x,x) \supset P(s^2(x),s^2(x))$	0.53
`LinearEq`	$\forall x.f(x) = x \rightarrow f^8(a) = a$	$\forall x.x = a \supset f^2(x) = a$	0.54
`SumOfOnes`	$\forall x.x + 0 = x, \forall x,y.x + s(y) = s(x + y)$ $\rightarrow 1 + 1 + 1 + 1 + 1 + 1 = s^6(0)$	$\forall x.x + s(0) = s(x)$	0.56
`SumOfOnesF`	$\forall x.x + 0 = x, \forall x,y.x + s(y) = s(x + y)$ $f(0) = 0, \forall x.f(s(x)) = f(x) + s(0)$ $\rightarrow f(s^4(0)) = s^4(0)$	$\forall x.f(s(x)) = s(f(x))$	0.69
`SumOfOnesF2`	$\forall x.x + 0 = x, \forall x,y.x + s(y) = s(x + y)$ $f(0) = 0, \forall x.f(s(x)) = f(x) + s(0)$ $\rightarrow f(s^4(0)) = s^4(0)$	$\forall x.f(x) = x \supset f(s(x)) = s(x)$	0.45

Table 1: Initial experimental results

Regarding the choice of the grammar, this command will compute the proofs with all minimal grammars, and will output the smallest proof (with respect to the number of rules). Of course, there might be two grammars that generate equally small proofs. In this case, any grammar/proof can be chosen as a solution.

Besides `LinearExampleProof`, other sequences of cut-free proofs — similar in spirit, but technically different from `LinearExampleProof` — are encoded in the gapt-system. To emphasize the potential of our cut-introduction method, we present in Table 1 the results (i.e. the generated cut-formulas and the compression ratio obtained by dividing the number of inferences of the generated proof by the number of inferences of the input proof) of applying the implementation to some instances of these sequences. All of the displayed proofs involving = use a usual axiomatization of equality based on reflexivity, symmetry, transitivity, and congruence. Of course, these results do not constitute a systematic empirical investigation, and evaluation of the method using larger data sets is a necessity. Such experiments are left for future work and are discussed in the following section.

9. Conclusion and Future Work

We have described a method for the inversion of Gentzen's cut-elimination method by the introduction of quantified cuts into an existing proof. Our method is based on separating the problem into two phases: first the minimization of a tree grammar and secondly: finding a solution of a unification problem. This separation is based on proof-theoretic results which makes the method computationally feasible as demonstrated by its implementation.

The work presented in this paper is only a first step and opens up several important directions for future work: a straightforward extension of this algorithm is to use blocks of quantifiers in the cut-formulas. This ability is useful for obtaining additional abbreviations. It would require modifications in the computation of the Δ-vector and, given the length of the paper, we have decided to leave this feature to future work. An equally obvious – but less straightforward – extension of this method is to cover cut-formulas with quantifier alternations. As prerequisite for this work, the extension of the connection between cut-elimination and tree languages established in [19] to the corresponding class of formulas is necessary.

On the empirical side an important aspect of future work will be to assess the abilities for compression of proofs produced by theorem provers: we intend to carry out large scale experiments with proofs produced from the TPTP-library [37] and the SMT-LIB [5]. Additional features that we consider important for such applications are to include the ability to work modulo simple theories and to systematically compute grammars whose language is a superset of the given set of terms.

On the theoretical side, we could only scratch the surface of many questions in this paper: What is the complexity of grammar minimization? What are good exact algorithms? Can we find incompressible tree languages? And more generally: study the complexity of cut-free proofs along the lines of measures such as automatic complexity [33] and automaticity [32]. Also the unification problem poses a number of

interesting theoretical challenges: Is the general unification problem of monadic predicate variables modulo propositional logic decidable? If yes, what is its complexity, what are good algorithms? What is the structure of the solution space of unification problems induced by cut-introduction? How can we navigate this structure systematically to find solutions of minimal size? How can we prover lower bounds on the size of such solutions?

References

[1] Franz Baader. On the complexity of Boolean unification. *Information Processing Letters*, 67:215–220, 1998.

[2] Matthias Baaz, Stefan Hetzl, and Daniel Weller. On the complexity of proof deskolemization. *Journal of Symbolic Logic*, 77(2):669–686, 2012.

[3] Matthias Baaz and Alexander Leitsch. On skolemization and proof complexity. *Fundmamenta Informaticae*, 20(4):353–379, 1994.

[4] Matthias Baaz and Richard Zach. Algorithmic Structuring of Cut-free Proofs. In *Computer Science Logic (CSL) 1992*, volume 702 of *Lecture Notes in Computer Science*, pages 29–42. Springer, 1993.

[5] Clark Barrett, Aaron Stump, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2010.

[6] George Boolos. Don't Eliminate Cut. *Journal of Philosophical Logic*, 13:373–378, 1984.

[7] Alan Bundy. The Automation of Proof by Mathematical Induction. In Andrei Voronkov and John Alan Robinson, editors, *Handbook of Automated Reasoning*, volume 1, pages 845–911. Elsevier, 2001.

[8] Alan Bundy, David Basin, Dieter Hutter, and Andrew Ireland. *Rippling: Meta-Level Guidance for Mathematical Reasoning*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2005.

[9] Simon Colton. *Automated Theory Formation in Pure Mathematics*. PhD thesis, University of Edinburgh, 2001.

[10] Simon Colton. *Automated Theory Formation in Pure Mathematics*. Springer, 2002.

[11] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata: Techniques and Applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

[12] William Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *Journal of Symbolic Logic*, 22(3):269–285, 1957.

[13] Cvetan Dunchev, Alexander Leitsch, Tomer Libal, Martin Riener, Mikheil Rukhaia, Daniel Weller, and Bruno Woltzenlogel-Paleo. ProofTool: GUI for the GAPT Framework. 10th International Workshop On User Interfaces for Theorem Provers, 2012.

[14] Marcelo Finger and Dov Gabbay. Equal Rights for the Cut: Computable Non-analytic Cuts in Cut-based Proofs. *Logic Journal of the IGPL*, 15(5–6):553–575, 2007.

[15] Ferenc Gécseg and Magnus Steinby. Tree Languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages: Volume 3: Beyond Words*, pages 1–68. Springer, 1997.

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

[17] Stefan Hetzl. Proofs as Tree Languages. submitted, preprint available at `http://hal.archives-ouvertes.fr/hal-00613713/`.

[18] Stefan Hetzl. Describing proofs by short tautologies. *Annals of Pure and Applied Logic*, 159(1–2):129–145, 2009.

[19] Stefan Hetzl. Applying Tree Languages in Proof Theory. In Adrian-Horia Dediu and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications (LATA) 2012*, volume 7183 of *Lecture Notes in Computer Science*. Springer, 2012.

[20] Stefan Hetzl, Alexander Leitsch, and Daniel 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.

[21] Stefan Hetzl and Lutz Straßburger. Herbrand-Confluence for Cut-Elimination in Classical First-Order Logic. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL) 2012*, volume 16 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 320–334. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012.

[22] Stefan Hetzl and Lutz Straßburger. Herbrand-Confluence. *Logical Methods in Computer Science*, 9(4), 2013.

[23] David Hilbert and Paul Bernays. *Grundlagen der Mathematik II*. Springer, 1939.

[24] Andrew Ireland and Alan Bundy. Productive Use of Failure in Inductive Proof. *Journal of Automated Reasoning*, 16(1–2):79–111, 1996.

[25] Florent Jacquemard, Francis Klay, and Camille Vacher. Rigid tree automata. In Adrian Horia Dediu, Armand-Mihai Ionescu, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications (LATA) 2009*, volume 5457 of *Lecture Notes in Computer Science*, pages 446–457. Springer, 2009.

[26] Florent Jacquemard, Francis Klay, and Camille Vacher. Rigid tree automata and applications. *Information and Computation*, 209:486–512, 2011.

[27] Moa Johansson, Lucas Dixon, and Alan Bundy. Conjecture synthesis for inductive theories. *Journal of Automated Reasoning*, 47(3):251–289, 2011.

[28] Richard Char-Tung Lee. *A completeness theorem and computer program for finding theorems derivable from given axioms*. PhD thesis, University of California, Berkeley, CA, 1967.

[29] Ursula Martin and Tobias Nipkow. Boolean Unification – The Story So Far. *Journal of Symbolic Computation*, 7(3-4):275–293, 1989.

[30] Dale Miller and Vivek Nigam. Incorporating tables into proofs. In *16th Conference on Computer Science and Logic (CSL07)*, volume 4646 of *Lecture Notes in Computer Science*, pages 466–480. Springer, 2007.

[31] V.P. Orevkov. Lower bounds for increasing complexity of derivations after cut elimination. *Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta*, 88:137–161, 1979.

[32] Jeffrey Shallit and Yuri Breitbart. Automaticity I: Properties of a Measure of Descriptional Complexity. *Journal of Computer and System Sciences*, 53:10–25, 1996.

[33] Jeffrey Shallit and Ming-Wei Wang. Automatic complexity of strings. *Journal of Automata, Languages and Combinatorics*, 6(4):537–554, 2001.

[34] Volker Sorge, Simon Colton, Roy McCasland, and Andreas Meier. Classification results in quasigroup and loop theory via a combination of automated reasoning tools. *Commentationes Mathematicae Universitatis Carolinae*, 49(2):319–339, 2008.

[35] Volker Sorge, Andreas Meier, Roy McCasland, and Simon Colton. Automatic Construction and Verification of Isotopy Invariants. *Journal of Automated Reasoning*, 40(2-3):221–243, 2008.

[36] Richard Statman. Lower bounds on Herbrand's theorem. *Proceedings of the American Mathematical Society*, 75:104–107, 1979.

[37] Geoff Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[38] Gaisi Takeuti. *Proof Theory*. North-Holland, 2nd edition, 1987.

[39] Anne S. Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, second edition, 2000.

[40] Jiři Vyskočil, David Stanovský, and Josef Urban. Automated Proof Compression by Invention of New Definitions. In E. M. Clark and A. Voronkov, editors, *Logic for Programming, Artifical Intelligence and Reasoning (LPAR-16)*, volume 6355 of *Lecture Notes in Computer Science*, pages 447–462. Springer, 2010.

[41] Bruno Woltzenlogel Paleo. Atomic Cut Introduction by Resolution: Proof Structuring and Compression. In E. M. Clark and A. Voronkov, editors, *Logic for Programming, Artifical Intelligence and Reasoning (LPAR-16)*, volume 6355 of *Lecture Notes in Computer Science*, pages 463–480. Springer, 2010.