Applying Tree Languages in Proof Theory

Stefan Hetzl[⋆]

Institute of Discrete Mathematics and Geometry
Vienna University of Technology
Wiedner Hauptstraße 8-10, A-1040 Vienna, Austria
hetzl@logic.at

Abstract. We introduce a new connection between formal language theory and proof theory. One of the most fundamental proof transformations in a class of formal proofs is shown to correspond exactly to the computation of the language of a certain class of tree grammars. Translations in both directions, from proofs to grammars and from grammars to proofs, are provided. This correspondence allows theoretical as well as practical applications.

1 Introduction

Proof theory developed from Hilbert's programme in the foundations of mathematics at the beginning of the 20th century. The fundamental observation of Hilbert was that even though mathematical proofs speak about infinite objects (such as real numbers, real-valued functions, vector spaces of such functions, etc.) they do so by using only a finite amount of symbols and of space. Therefore, considering proofs as mathematical objects in their own right makes them amenable to analysis by mathematical (finitary, discrete) means. Hilbert's original aim was to justify mathematical reasoning by consistency proofs which, by Gödel's second incompleteness theorem, turned out to be too ambitious. However, other kinds of analyses of proofs are possible.

One type of analysis that has received a lot of attention in recent years (see e.g. [15]) is *proof mining*: the extraction of additional mathematical information from existing proofs. Such additional information can often be thought of as concrete values for existential quantifiers. In the most simple situations it can be straightforward to read off such values, for example in case a proof of a statement $\exists x\, \varphi(x)$ starts with "Let us show $\varphi(a)$." for some concrete value a. In general the situation is more complicated, consider the following famous example:

Theorem. There are $x, y \in \mathbb{R} \setminus \mathbb{Q}$ s.t. $x^y \in \mathbb{Q}$.

Proof. If $\sqrt{2}^{\sqrt{2}} \in \mathbb{Q}$, let $x = y = \sqrt{2}$ and we are done as $\sqrt{2} \in \mathbb{R} \setminus \mathbb{Q}$. Otherwise $\sqrt{2}^{\sqrt{2}} \in \mathbb{R} \setminus \mathbb{Q}$, let $x = \sqrt{2}^{\sqrt{2}}, y = \sqrt{2}$ and observe $x^y = \sqrt{2}^2 = 2 \in \mathbb{Q}$. □

This proof does not give us any information on whether $\sqrt{2}^{\sqrt{2}} \in \mathbb{Q}$ or not.

⋆ This work was supported by a Marie Curie Intra European Fellowship within the 7th European Community Framework Programme and by the projects I-603 N18 and P22028 of the Austrian Science Fund (FWF).

The values of the existential quantifiers $\exists x$ and $\exists y$ are not unique, we have to represent this proof by a *disjunction of instances of the theorem*: if $\varphi(x, y)$ abbreviates "$x \in \mathbb{R} \setminus \mathbb{Q}$ and $y \in \mathbb{R} \setminus \mathbb{Q}$ and $x^y \in \mathbb{Q}$", the above proof only shows that $\varphi(\sqrt{2}, \sqrt{2}) \vee \varphi(\sqrt{2}^{\sqrt{2}}, \sqrt{2})$ is provable (from suitable basic axioms). This situation is described from a logical point of view by Herbrand's theorem [9,2]. In its simplest form it states that $\exists x\, \varphi(x)$ for a quantifier-free formula $\varphi(x)$ is valid iff there are terms $t_1, \ldots, t_n$ s.t. $\bigvee_{i=1}^{n} \varphi(t_i)$ is a tautology. Such a disjunction is therefore also called *Herbrand-disjunction*.

It was easy to read off an Herbrand-disjunction from the above proof. The reason is that it contains the instances of its quantifiers in plain sight. In general however proofs use lemmas and values of quantifiers of the theorem depend on objects whose existence is asserted by these lemmas. The proofs of these lemmas may in turn rely on other lemmas and so on. In total, there can be very complicated dependencies between values of quantifiers that have to be unwound in order to obtain concrete values. A proof transformation that carries out this unwinding is *cut-elimination* (the root of this terminology being that the *cut*-rule formalises the use of a lemma). This transformation is, for a number of reasons, of central importance in proof theory. It essentially works by a stepwise reduction of the complexity of the cuts (lemmas), the interested reader is referred to [19].

In this paper we show that for a certain class of proofs, cut-elimination corresponds *exactly* to the computation of the language of a certain tree grammar: we give translations from proofs to grammars and from grammars to proofs s.t. this correspondence holds. The connection point is the observation that *an Herbrand-disjunction is given by a finite set of terms, hence a finite tree language.* A proof with lemmas then corresponds to a tree grammar whose language is an Herbrand-disjunction. Therefore, one can obtain concrete values for existential quantifiers by computing the language of a grammar. In Section 2 we will develop a suitable notion of tree grammar corresponding to rigid tree automata [13,14], in Section 3 we describe how to translate proofs to grammars and in Section 4 how to translate grammars to proofs. Most proofs in this paper will only be sketched, the reader interested in more details is referred to the technical report [10].

2 Rigid Tree Languages

A feature which is important for many applications of tree languages but not present in regular tree languages is the ability to carry out equality tests between subterms, for instance to recognise patterns of the form $f(x, x)$. This need has led to the development of several classes of tree automata providing this ability: some allow to specify local equality constraints as side conditions of transition rules by giving term positions explicitly, see [3] for a survey, while others consider global constraints specified via states. An important class of the latter kind are *tree automata with global equalities and disequalities* (TAGED) [5,6,7]. For the purposes of this paper it will turn out to be natural to work with *rigid tree automata* that have been introduced in [13], see also [14]. They are a subclass of TAGED (characterised by having minimal equality and disequality relations).

Definition 1. *A* tree automaton *on a signature Σ is a tuple $\langle Q, F, \Delta\rangle$ where Q is a finite set of state symbols, $F \subset Q$ is the set of final states and Δ a set of transition rules of the form: $f(q_1, \ldots, q_n) \to q$ where $f \in \Sigma$ and $q, q_1, \ldots, q_n \in Q$.*

A rigid tree automaton *on Σ is a tuple $\langle Q, R, F, \Delta\rangle$ where $\langle Q, F, \Delta\rangle$ is a tree automaton and $R \subseteq Q$ is the set of rigid states.*

As usual a position is a list of natural numbers, $\mathrm{Pos}(t)$ is the set of positions of the term t, ε is the empty (root) position and concatenation of positions p_1 and p_2 is written as $p_1.p_2$. The subterm of t at a position $p \in \mathrm{Pos}(t)$ is denoted as $t|_p$. We write $\mathrm{Pos}(x, t)$ for the set of positions of the symbol x in t and we write $x \in t$ if $\mathrm{Pos}(x, t) \neq \emptyset$. A run of a tree automaton on a term t is a function $r : \mathrm{Pos}(t) \to Q$ s.t. for all $f \in \Sigma$ and all $p \in \mathrm{Pos}(f, t)$: $f(r(p.1), \ldots, r(p.n)) \to r(p) \in \Delta$. A run of a rigid tree automaton on t is a run of the underlying tree automaton satisfying the additional *rigidity condition*: for all $p_1, p_2 \in \mathrm{Pos}(t)$: if $r(p_1) = r(p_2) \in R$ then $t|_{p_1} = t|_{p_2}$. $\mathcal{T}(\Sigma)$ denotes the set of ground terms over a signature Σ. The language of an automaton A in a state q is denoted as $L(A, q)$ and defined as the set of $t \in \mathcal{T}(\Sigma)$ s.t. there exists a run r on t with $r(\varepsilon) = q$. The language of A is defined as $L(A) = \bigcup_{q \in F} L(A, q)$.

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 recognised by the rigid tree automaton $\langle Q, R, F, \Delta\rangle$ where $Q = \{q, q_r, q_f\}$, $R = \{q_r\}$, $F = \{q_f\}$ and $\Delta = \{0 \to q, 0 \to q_r, s(q) \to q, s(q) \to q_r, f(q_r, q_r) \to q_f\}$.

For the proof-theoretic purposes of this paper it is considerably more natural and technically useful to work with grammars instead of automata.

Definition 3. *A* regular tree grammar *is a tuple $\langle \alpha, N, \Sigma, P\rangle$ composed of an axiom α, a set N of non-terminal symbols with arity* 0 *and $\alpha \in N$, a term signature Σ with $\Sigma \cap N = \emptyset$ and a set P of production rules of the form $\beta \to t$ where $\beta \in N$ and $t \in \mathcal{T}(\Sigma \cup N)$.*

A rigid tree grammar *is a tuple $\langle \alpha, N, R, \Sigma, P\rangle$ where $\langle \alpha, N, \Sigma, P\rangle$ is a regular tree grammar and $R \subseteq N$ is the set of rigid non-terminal symbols.*

The derivation relation $\to_G$ of a regular tree grammar G is defined for $s, t \in \mathcal{T}(\Sigma \cup N)$ as $s \to_G t$ if there is a production rule $\beta \to u$ and a position p s.t. $s|_p = \beta$ and t is obtained from s by replacing β at p by u. A derivation of a term $t \in \mathcal{T}(\Sigma)$ in a regular tree grammar is a list of terms $t_1, \ldots, t_n \in \mathcal{T}(\Sigma \cup N)$ s.t. $t_1 = \alpha, t_n = t$ and $t_i \to_G t_{i+1}$ for $i = 1, \ldots, n-1$. A derivation of t in a rigid tree grammar is a derivation in the underlying regular tree grammar satisfying the additional *rigidity condition*: if $t_i \to_G t_{i+1}$ and $t_j \to_G t_{j+1}$ are applications of productions rules at positions p_i, p_j with the same left-hand side $\beta \in R$, then $t|_{p_i} = t|_{p_j}$. The language of a tree grammar $L(G)$ is the set of $t \in \mathcal{T}(\Sigma)$ that are derivable in G. A production whose left-hand side is β will be called β-production. Let $G = \langle \alpha, N, R, T, P\rangle$ be a rigid tree grammar; a rigid tree grammar $G' = \langle \alpha, N, R, T, P'\rangle$ is called *projection of* G if $P' \subseteq P$ and P' contains at most one β-production for every $\beta \in R$. A first basic but useful observation about rigid tree grammars is the following

Lemma 4. *Let G be a rigid tree grammar and let $t \in L(G)$. Then there is a projection G' of G s.t. $t \in L(G')$.*

Proof. Use the rigidity condition, replace subderivations if needed.

In order to establish the connection with the existing literature we quickly sketch a proof of the equivalence of rigid tree grammars and rigid tree automata. The reader interested in the details is referred to [10].

Definition 5. *A grammar is called* normalised *if every production rule has the form $\gamma \to a$ or $\gamma \to f(\gamma_1, \dots, \gamma_n)$ for $a, f \in \Sigma$ and $\gamma, \gamma_1, \dots, \gamma_n \in N$.*

Lemma 6. *If G is a rigid tree grammar, then there is a normalised rigid tree grammar G^* s.t. $L(G) = L(G^*)$.*

Proof. In a first phase of normalisation, productions $\beta \to f(t_1, \dots, t_n)$ are replaced by $\beta \to f(\beta_1, \dots, \beta_n), \beta_1 \to t_1, \dots, \beta_n \to t_n$. In the second phase, productions of the form $\beta \to \gamma$ are removed which – due to the rigidity condition – necessitates a different treatment depending on which of β, γ are rigid.

Theorem 7. *A set of terms is language of a rigid tree grammar iff it is language of a rigid tree automaton.*

Proof. It is straightforward to translate between normalised rigid tree grammars and rigid tree automata, the result then follows from Lemma 6.

Definition 8. *A rigid tree grammar is called* totally rigid *if all non-terminals are rigid.*

Totally rigid tree grammars will simply be written as $\langle \alpha, R, \Sigma, P \rangle$.

Definition 9. *Let G be a regular or rigid tree grammar with non-terminals N and productions P. Define an order $<^1_G$ on N as $\alpha <^1_G \beta$ if $\alpha \to t \in P$ and $\beta \in t$ and write $<_G$ for the transitive closure of $<^1_G$. G is called* acyclic *if $<_G$ is.*

Example 10. The totally rigid grammar $G = \langle \alpha, R, \Sigma, P \rangle$ with $R = \{\alpha, \beta, \gamma\}$, $\Sigma = \{f/1, g/1, g/1, a/0, b/0\}$, and $P = \{\alpha \to h(\beta) | h(\gamma),\ \beta \to f(\gamma) | a,\ \gamma \to g(\beta) | b\}$ is cyclic because $\beta <_G \gamma$ and $\gamma <_G \beta$ but removing $\beta \to f(\gamma)$ or $\gamma \to g(\beta)$ (or both) from the productions yields an acyclic grammar.

Totally rigid acyclic grammars are central for this paper as they correspond to proofs. Furthermore, they allow the following description of their language in terms of substitutions. As usual, a substitution is a mapping from variables to terms which is different from the identity for only a finite number of variables. A substitution is written as $[x_1 \backslash t_1, \dots, x_n \backslash t_n]$. Application of a substitution σ to a term t is written as $t\sigma$.

Lemma 11. *If G is totally rigid and acyclic, then up to renaming of the non-terminals $G = \langle \alpha_0, \{\alpha_0, \dots, \alpha_n\}, \Sigma, P \rangle$ 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 a renaming of non-terminals s.t. $\alpha_i >_P \alpha_j$ implies $i > j$. The result then follows from re-arranging the derivation and Lemma 4.

Consequently, the language $L(G)$ of a totally rigid and acyclic G is finite.

3 From Proofs to Tree Languages

We now turn to proof theory. In the seminal article [8], which can be considered the founding work of structural proof theory, Gentzen introduced the sequent calculus and proved the cut-elimination theorem. A sequent is a pair of multisets of formulas written $A_1, \dots, A_n \vdash B_1, \dots, B_m$ whose intended meaning is the formula $(\bigwedge_{i=1}^n A_i) \to (\bigvee_{j=1}^m B_j)$.

Definition 12. *A proof in the sequent calculus is a tree that starts with sequents of the form $A \vdash A$ for an atomic formula A and is built up using the following rules:*
The logical rules:

$$\frac{\Gamma \vdash \Delta, A \quad \Pi \vdash \Lambda, B}{\Gamma, \Pi \vdash \Delta, \Lambda, A \wedge B}\ \wedge_{\mathrm{r}} \qquad \frac{A, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta}\ \wedge_{\mathrm{l}_1} \qquad \frac{B, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta}\ \wedge_{\mathrm{l}_2}$$

$$\frac{A, \Gamma \vdash \Delta \quad B, \Pi \vdash \Lambda}{A \vee B, \Gamma, \Pi \vdash \Delta, \Lambda}\ \vee_{\mathrm{l}} \qquad \frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, A \vee B}\ \vee_{\mathrm{r}_1} \qquad \frac{\Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \vee B}\ \vee_{\mathrm{r}_2}$$

$$\frac{\Gamma \vdash \Delta, A}{\neg A, \Gamma \vdash \Delta}\ \neg_{\mathrm{l}} \qquad \frac{A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \neg A}\ \neg_{\mathrm{r}}$$

$$\frac{A[x\backslash t], \Gamma \vdash \Delta}{\forall x\, A, \Gamma \vdash \Delta}\ \forall_{\mathrm{l}} \qquad \frac{\Gamma \vdash \Delta, A[x\backslash \alpha]}{\Gamma \vdash \Delta, \forall x\, A}\ \forall_{\mathrm{r}} \qquad \frac{A[x\backslash \alpha], \Gamma \vdash \Delta}{\exists x\, A, \Gamma \vdash \Delta}\ \exists_{\mathrm{l}} \qquad \frac{\Gamma \vdash \Delta, A[x\backslash t]}{\Gamma \vdash \Delta, \exists x\, A}\ \exists_{\mathrm{r}}$$

where t is a term, α is a variable and the quantifier rules are subject to the following conditions:

1. *t must not contain a bound variable,*
2. *α is called* eigenvariable *and must not occur in $\Gamma \cup \Delta \cup \{A\}$*

The structural rules weakening, contraction and cut:

$$\frac{\Gamma \vdash \Delta}{A, \Gamma \vdash \Delta}\ \mathrm{w}_{\mathrm{l}} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A}\ \mathrm{w}_{\mathrm{r}} \qquad \frac{A, A, \Gamma \vdash \Delta}{A, \Gamma \vdash \Delta}\ \mathrm{c}_{\mathrm{l}} \qquad \frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}\ \mathrm{c}_{\mathrm{r}}$$

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

The formula A in an application of the cut-rule is called *cut-formula.* The sequent at the root of a proof is called *end-sequent* of that proof. This calculus is sound and complete for classical first-order logic in the sense that a formula F is valid iff there is a proof whose end-sequent is $\vdash F$. We consider $A \to B$ to be an abbreviation of $\neg A \vee B$ and also allow free use of corresponding rule abbreviations $\to_{\mathrm{l}}$ and $\to_{\mathrm{r}}$ for

$$\frac{\Gamma \vdash \Delta, A \quad B, \Pi \vdash \Lambda}{A \to B, \Gamma, \Pi \vdash \Delta, \Lambda}\ \to_{\mathrm{l}} \qquad \text{and} \qquad \frac{A, \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \to B}\ \to_{\mathrm{r}} \quad .$$

Furthermore, $\to$ is right-associative.

Example 13. Define the formulas $A_1 = P(a) \vee P(b)$, $A_2 = \forall x\,(P(x) \to Q(f(x)))$ and $A_3 = \forall x \forall y\,(P(x) \to Q(y) \to R(g(x,y)))$ and the proof $\pi =$

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{matrix}(\pi_1)\\ A_1 \vdash P(a), P(b)\end{matrix}}{A_1 \vdash \exists x P(x), P(b)}\,\exists_\mathrm{r}}{A_1 \vdash \exists x P(x), \exists x P(x)}\,\exists_\mathrm{r}}{A_1 \vdash \exists x P(x)}\,\mathrm{c_r} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\begin{matrix}(\pi_2)\\ P(\alpha), A_2 \vdash Q(f(\alpha))\end{matrix}}{P(\alpha), A_2 \vdash \exists x Q(x)}\,\exists_\mathrm{r} \quad \dfrac{\dfrac{\begin{matrix}(\pi_3)\\ P(\alpha), Q(\beta), A_3 \vdash R(g(\alpha,\beta))\end{matrix}}{P(\alpha), Q(\beta), A_3 \vdash \exists x R(x)}\,\exists_\mathrm{r}}{P(\alpha), \exists x Q(x), A_3 \vdash \exists x R(x)}\,\exists_\mathrm{l}}{P(\alpha), P(\alpha), A_2, A_3 \vdash \exists x R(x)}\,\mathrm{cut}}{P(\alpha), A_2, A_3 \vdash \exists x R(x)}\,\mathrm{c_l}}{\exists x P(x), A_2, A_3 \vdash \exists x R(x)}\,\exists_\mathrm{l}}{A_1, A_2, A_3 \vdash \exists x R(x)}\,\mathrm{cut}$$

where the proofs π_1, π_2 and π_3 are left to the reader. This proof contains two cuts, one whose cut-formula is $\exists x\, P(x)$ and another whose cut-formula is $\exists x\, Q(x)$.

A quantifier in a formula A is called *positive* if it is below an even number of negations in the syntax tree of A and negative if it is under an odd number of negations. Positive universal and negative existential quantifiers are called *strong*, the others are called *weak*. It is straightforward to show that a strong quantifier is introduced by $\forall_\mathrm{r}$ or $\exists_\mathrm{l}$ and a weak quantifier by $\forall_\mathrm{l}$ or $\exists_\mathrm{r}$. A proof is called *regular* if different strong quantifier inferences have different eigenvariables. From now on we will assume – as a convention on variable-naming – that all proofs are regular.

In the above definition some formulas are mentioned explicitly like $A \wedge B$, A and B in the case of $\wedge_\mathrm{r}$. The formula $A \wedge B$ below the rule is called *main formula* and the formulas A and B above it are called *auxiliary formulas* of the rule. Analogous definitions apply to all other rules. One then defines an *ancestor* relation on the formula occurrences in a proof as follows: an auxiliary formula occurrence is ancestor of a main formula occurrence and furthermore the occurrence of a formula in the context $\Gamma, \Delta, \Pi, \Lambda$ above an inference is ancestor of the corresponding occurrence below the inference. For illustration of the ancestor relation see Definitions 14, 17 and Examples 15, 18.

From now on and for the rest of this paper, T will denote a universal theory, i.e. a set of formulas of the form $\forall x_1 \cdots \forall x_n B$ with B quantifier-free. It is a standard result of mathematical logic that every theory has a conservative universal extension which is obtained by Skolemisation, see e.g. [22]. Concentrating on universal theories hence does not significantly restrict (though simplifies technically) the results of this paper.

Definition 14. *Let π be a proof of $T \vdash \exists x\, A$ with A quantifier-free and ψ a subproof of π. The* Herbrand-set $\mathrm{H}(\psi, \pi)$ *of ψ w.r.t. π is defined as follows. If ψ is an axiom, then* $\mathrm{H}(\psi, \pi) = \emptyset$. *If ψ is of the form*

$$\dfrac{\begin{matrix}(\psi')\\ \Pi \vdash \Lambda, A[x\backslash t]\end{matrix}}{\Pi \vdash \Lambda, \exists x\, A}\,\exists_\mathrm{r}$$

where the main formula $\exists x\, A$ is ancestor of the formula $\exists x\, A$ in the end-sequent, then $\mathrm{H}(\psi, \pi) = \mathrm{H}(\psi', \pi) \cup \{A[x\backslash t]\}$*. If ψ ends with any other unary inference and ψ' is its immediate subproof then* $\mathrm{H}(\psi, \pi) = \mathrm{H}(\psi', \pi)$*. If ψ ends with a binary inference and ψ_1, ψ_2 are its immediate subproofs, then* $\mathrm{H}(\psi, \pi) = \mathrm{H}(\psi_1, \pi) \cup \mathrm{H}(\psi_2, \pi)$*. We write* $\mathrm{H}(\pi)$ *for* $\mathrm{H}(\pi, \pi)$*.*

Example 15. The proof π of Example 13 has $\mathrm{H}(\pi) = \{R(g(\alpha, \beta))\}$.

Example 16. A (suitable) formalisation of the proof discussed in the introduction has the Herbrand-set $\{\varphi(\sqrt{2}, \sqrt{2}), \varphi(\sqrt{2}^{\sqrt{2}}, \sqrt{2})\}$.

Definition 17. *Let π be a proof and Q be a quantifier occurrence in π. Define a set of terms* $\mathrm{t}(Q)$ *associated with Q as follows: if Q occurs in the main formula of a weakening, then* $\mathrm{t}(Q) = \emptyset$*. If Q is introduced by a quantifier inference from a term t or a variable x, then* $\mathrm{t}(Q) = \{t\}$ *or* $\mathrm{t}(Q) = \{x\}$ *respectively. If Q occurs in the main formula of a contraction and Q_1, Q_2 are the two corresponding quantifiers in the auxiliary formulas of the contraction, then* $\mathrm{t}(Q) = \mathrm{t}(Q_1) \cup \mathrm{t}(Q_2)$*. In all other cases Q has exactly one immediate ancestor Q' and* $\mathrm{t}(Q) = \mathrm{t}(Q')$*.*

Let π be a proof and c be a cut in π. Write $\mathrm{Q}(c)$ *for the set of pairs (Q, Q') of quantifier occurrences where Q is a strong occurrence in one occurrence of the cut-formula of c and Q' the corresponding weak occurrence in the other occurrence of the cut-formula. Define the set of base substitutions of c as* $\mathrm{B}(c) = \bigcup_{(Q,Q') \in \mathrm{Q}(c)} \{[x\backslash t] \mid x \in \mathrm{t}(Q), t \in \mathrm{t}(Q')\}$*. For $c_1, \ldots, c_n$ being the cuts in π define the base substitutions of π as* $\mathrm{B}(\pi) = \bigcup_{i=1}^{n} \mathrm{B}(c_i)$*.*

Example 18. The proof π of Example 13 has $\mathrm{B}(\pi) = \{[\alpha\backslash a], [\alpha\backslash b], [\beta\backslash f(\alpha)]\}$.

Definition 19. *For a proof π define the totally rigid tree grammar* $\mathrm{G}(\pi) = \langle \varphi, N, \Sigma, P\rangle$ *by* $N = \{\varphi\} \cup \mathrm{EV}(\pi)$*,* $\Sigma = \Sigma(\pi) \cup \{\wedge, \vee, \neg\}$*, and* $P = \{\varphi \to F \mid F \in \mathrm{H}(\pi)\} \cup \{\alpha \to t \mid [\alpha\backslash t] \in \mathrm{B}(\pi)\}$*, where* $\mathrm{EV}(\pi)$ *is the set of eigenvariables of the proof π and $\Sigma(\pi)$ is its first-order signature.*

Example 20. The proof π of Example 13 has $\mathrm{G}(\pi) = \langle \varphi, N, \Sigma, P\rangle$ where $N = \{\varphi, \alpha, \beta\}$, $\Sigma = \{a, b, f, g, P, Q, R, \wedge, \vee, \neg\}$ and $P = \{\varphi \to R(g(\alpha, \beta)), \beta \to f(\alpha), \alpha \to a, \alpha \to b\}$ hence $L(\mathrm{G}(\pi)) = \{R(g(a, f(a))), R(g(b, f(b)))\}$. The reader is invited to verify that $A_1, A_2, A_3 \vdash L(\mathrm{G}(\pi))$ is provable for A_1, A_2, A_3 as in Example 13.

Definition 21. *A proof π is called* simple *if every cut-formula in π contains at most one quantifier.*

We will now restrict our attention to simple proofs. The main result of this paper is that cut-elimination in the class of simple proofs corresponds exactly (in a sense made precise below) to the computation of the language of a totally rigid acyclic tree grammar. While this restriction on proofs substantially decreases the scope of the present analysis, simple proofs are still of considerable interest: they do contain quantified cuts and hence allow the formalisation of some mathematical lemmas and their cut-elimination is of exponential complexity.

The size of a grammar G, written as $|G|$, is the number of production rules. The size of a proof π, written as $|\pi|$, is the number of inferences.

Theorem 22. *If π is a simple proof of $T \vdash \exists x\, A$ with A quantifier-free, then there is a totally rigid acyclic grammar G with $|G| \leq |\pi|$ and $L(G) \subseteq L(\mathrm{G}(\pi))$ s.t. $T \vdash L(G)$ is provable.*

Proof. Only a sketch of the proof is described here, the reader interested in the details is referred to the technical report [10]. We have $|\mathrm{G}(\pi)| \leq |\pi|^2$, the quadratic size being due to quantifiers in cut-formulas which are introduced from a linear number of terms (on their weak side) and a linear number of eigenvariables (on their strong side).

Let β, γ be two non-terminals of $\mathrm{G}(\pi)$ and write $\beta \sim \gamma$ if there is a strong quantifier occurrence Q in a cut-formula with $\beta, \gamma \in \mathrm{t}(Q)$. The equivalence relation $\sim$ defines a partition of the non-terminals of $\mathrm{G}(\pi)$ into n classes where n is the number of cuts in π that contain a quantifier. Define the totally rigid grammar G from $\mathrm{G}(\pi)$ by identifying all non-terminals of the same $\sim$-class. Then $|G| \leq |\pi|$ and $L(G) \subseteq L(\mathrm{G}(\pi))$.

Furthermore, let d be a new ("dummy") constant and, for a given proof ψ, write $\mathrm{G}_{\mathrm{nd}}(\psi)$ for $\mathrm{G}(\psi)$ from which all productions of the form $\beta \to d$ for any non-terminal β have been removed. By proof-theoretic transformations – in particular the prenexification of [1], see also [4, Theorem VII.4.7], applied to cut-formulas – we obtain a proof π' all of whose cuts are of the form $\exists x\, B$ for B quantifier-free and which satisfies $L(\mathrm{G}_{\mathrm{nd}}(\pi')) = L(G)$. The role of the dummy constant is to mark artefacts introduced by prenexification into the grammar as such.

The central part of the proof then consists in applying a suitable procedure for cut-elimination to π' and to show that this process is computing $L(\mathrm{G}_{\mathrm{nd}}(\pi'))$ step-by-step from $\mathrm{G}(\pi')$ in the proof: the systematic unfolding of the proof induced by cut-elimination essentially transforms a grammar into its language, the occurrences of the dummy constant are deleted by this process due to their positions in the proof. Finally we obtain a cut-free proof π^* with $\mathrm{H}(\pi^*) = L(\mathrm{G}_{\mathrm{nd}}(\pi'))$. This allows to conclude that $T \vdash L(\mathrm{G}_{\mathrm{nd}}(\pi'))$, i.e. $T \vdash L(G)$, is provable.

Corollary 23. *If π is a simple proof of $T \vdash \exists x\, A$ with A quantifier-free, then $T \vdash L(\mathrm{G}(\pi))$ is provable.*

Proof. Append weakenings to the proof of $T \vdash L(G)$ obtained from Theorem 22.

Example 24. Applying Corollary 23 to the proof π of Example 13 shows that

$$A_1, A_2, A_3 \vdash R(g(a, f(a))), R(g(b, f(b)))$$

is provable. The reader is invited to verify that a standard algorithm for cut-elimination (see e.g. [19]) gives the same result.

Note that Theorem 22 together with Lemma 11 provides an exponential upper bound on the complexity of cut-elimination in simple proofs, more precisely: for every simple proof π of $T \vdash \exists x\, A$ with A quantifier-free there are $t_1, \ldots, t_k$ with $k \leq |\pi|^{|\pi|}$ s.t. $T \vdash A[x\backslash t_1], \ldots, A[x\backslash t_k]$. On the other hand, cut-elimination in general is non-elementary [23,17,18] which shows that simplicity is a necessary assumption for Theorem 22.

4 From Tree Languages to Proofs

Already the results in [11] show that a simple proof induces an acyclic regular(!) tree grammar whose finite language is an Herbrand-disjunction. So what have we gained from strengthening this result by adding total rigidity? On the one hand, we have gained an exponent: in contrast to the bound n^n obtained in the totally rigid case, there are acyclic regular tree grammars G_n with $2n$ productions and $|L(G_n)| = n^{n^n}$ (G_n builds a tree with depth n, branching-degree n and n choices at each leaf).

However there is another – more fundamental – motivation for this result: in this section we show that the compression power of simple proofs corresponds *exactly* to that of totally rigid acyclic grammars. Given such a grammar G we will obtain a simple proof π that induces G and whose cut-elimination essentially computes $L(G)$ from G. As $L(\mathrm{G}(\pi))$ is a set of formulas but $L(G)$ is a set of terms (which do not necessarily represent formulas), we cannot expect to obtain $\mathrm{G}(\pi) = G$. The closest possible connection is to wrap up the term language of G in some new unary predicate symbol R_1. Therefore the proofs constructed in the theorem below have $T \vdash \exists x\, R_1(x)$ as end-sequent. For proofs π and π' we write $\pi \to \pi'$ if π' can be obtained from π by applying the standard reduction rules of cut-elimination as in [19].

Theorem 25. *For every totally rigid acyclic tree grammar $G = \langle \alpha_1, R, \Sigma, P\rangle$ there is a simple proof π with $\mathrm{G}(\pi) = \langle \alpha_0, R \cup \{\alpha_0\}, \Sigma, P \cup \{\alpha_0 \to R_1(\alpha_1)\}\rangle$ and a cut-free proof π' with $\pi \to \pi'$ and $\mathrm{H}(\pi') = L(\mathrm{G}(\pi))$.*

Proof. By Lemma 11 we can assume that $G = \langle \alpha_1, \{\alpha_1, \ldots, \alpha_n\}, \Sigma, P\rangle$ s.t. α_i depends only on α_j with $j > i$. The proof π is defined in the language $\Sigma \cup \{R_i \mid 1 \le i \le n\}$ where the R_i are unary predicate symbols with intended interpretation "being reachable from the non-terminal α_i". For each rule $\alpha_i \to t$ define the formula

$$\varphi_{\alpha_i \to t} \;=\; \forall x_{i+1} \cdots \forall x_n \;(\; R_{i+1}(x_{i+1}) \to \cdots R_n(x_n) \to R_i(t[\alpha_j \backslash x_j]_{j=i+1}^{n}) \;).$$

For each non-terminal α_i with rules $\alpha_i \to t_1, \ldots, \alpha_i \to t_m$ define the formula

$$\varphi_i \;=\; \bigvee_{j=1}^{m} \varphi_{\alpha_i \to t_j}$$

and the proof $\psi_i =$

$$\dfrac{\cdots \qquad \dfrac{\dfrac{R_i(t_j) \vdash R_i(t_j)}{R_i(t_j) \vdash \exists x\, R_i(x)}\,\exists_{\mathrm{r}}}{\varphi_{\alpha_i \to t_j}, R_{i+1}(\alpha_{i+1}), \ldots, R_n(\alpha_n) \vdash \exists x\, R_i(x)}\,\forall_{\mathrm{l}}^{*}, \to_{\mathrm{l}}^{*} \qquad \cdots}{\varphi_i, R_{i+1}(\alpha_{i+1}), \ldots, R_n(\alpha_n) \vdash \exists x\, R_i(x)}\,\mathrm{c}^{*}, \vee_{\mathrm{l}}^{*}\;.$$

Now define proofs $\pi_i : \varphi_1, \ldots, \varphi_i, R_{i+1}(\alpha_{i+1}), \ldots, R_n(\alpha_n) \vdash \exists x\, R_1(x)$ for $i \in \{0, \ldots, n\}$ and $\pi'_i : \varphi_1, \ldots, \varphi_i, \exists x\, R_{i+1}(x), R_{i+2}(\alpha_{i+2}), \ldots, R_n(\alpha_n) \vdash \exists x\, R_1(x)$

for $i \in \{0, \dots, n-1\}$ by

$$\pi'_i \;=\; \dfrac{\begin{array}{c}(\pi_i)\\ \varphi_1, \dots, \varphi_i, R_{i+1}(\alpha_{i+1}), \dots, R_n(\alpha_n) \vdash \exists x\, R_1(x)\end{array}}{\varphi_1, \dots, \varphi_i, \exists x\, R_{i+1}(x), R_{i+2}(\alpha_{i+2}), \dots, R_n(\alpha_n) \vdash \exists x\, R_1(x)}\ \exists_{\mathrm{l}}$$

and

$$\pi_0 \;=\; \dfrac{R_1(\alpha_1) \vdash R_1(\alpha_1)}{R_1(\alpha_1), \dots, R_n(\alpha_n) \vdash \exists x\, R_1(x)}\ \mathrm{w}_{\mathrm{l}}^*, \exists_{\mathrm{r}}$$

and

$$\pi_{i+1} \;=\; \dfrac{\begin{array}{c}(\psi_{i+1})\\ \varphi_{i+1}, R_{i+2}(\alpha_{i+2}), \dots, R_n(\alpha_n) \vdash \exists x\, R_{i+1}(x)\end{array} \qquad (\pi'_i)}{\varphi_1, \dots, \varphi_{i+1}, R_{i+2}(\alpha_{i+2}), \dots, R_n(\alpha_n) \vdash \exists x\, R_1(x)}\ \mathrm{c}_{\mathrm{l}}^*, \mathrm{cut} \quad .$$

Then it is straightforward to verify that $\pi = \pi_n : \varphi_1, \dots, \varphi_n \vdash \exists x\, R_1(x)$ has the desired grammar. In order to obtain π', reduce the cuts in a bottom-up order which for each production rule of an α_{i+1} will create a new copy of π_i hence computing the language $L(\mathrm{G}(\pi))$ by expansion from right to left (in the representation of Lemma 11).

5 Applications

The above results pave the way for several applications of formal language theory in proof theory to be further explored in future work. First of all, for carrying out concrete analyses of simple proofs one can use rigid tree grammars instead of the more cumbersome cut-elimination to compute values for existential quantifiers. Secondly, standard problems of formal language theory such as membership, intersection, etc. assume a proof-theoretic meaning by allowing to answer whether a given value is obtained from a given proof, what values can be obtained from both of two given proofs, etc.

Furthermore, these results show that the length of a proof with cut (which is notoriously difficult to control) is intimately related to measures such as automatic complexity [21] and automaticity [20], more precisely:

Corollary 26. *Let $\exists x\, A$ be a formula and $k \in \mathbb{N}$ s.t. $T \vdash A[x\backslash t_1], \dots, A[x\backslash t_n]$ implies that every totally rigid acyclic grammar G with $L(G) = \{t_1, \dots, t_n\}$ has $|G| \geq k$, then every simple proof π of $T \vdash \exists x\, A$ has $|\pi| \geq k$.*

Proof. Suppose there was a simple proof π_0 of $T \vdash \exists x\, A$ with $|\pi_0| < k$, then by Theorem 22 there would be a totally rigid acyclic tree grammar G_0 with $|G_0| \leq |\pi_0| < k$ s.t. $T \vdash L(G_0)$ would be provable, contradiction.

Via this connection, a lower bound on grammars thus translates to a lower bound on proofs with cut.

Another intriguing perspective is to exploit these results computationally by abbreviating a cut-free proof through the introduction of cuts which are obtained from first computing a small grammar: the cut-free proof of $T \vdash \exists x\, A$

is represented by its Herbrand-disjunction $A[x\backslash t_1], \ldots, A[x\backslash t_n]$ which in turn is represented by the trivial grammar whose axiom is x and whose productions are $x \to t_1, \ldots, x \to t_n$. An analysis of the structure of the t_i can lead to a smaller grammar representing the same language. It then only remains to check whether the grammar can be realised by cut-formulas of a simple proof (which for the case of a single cut is always the case). A first algorithm based on this approach for the case of a single cut can be found in [12].

6 Conclusion

We have shown that cut-elimination in proofs where each cut contains at most one quantifier corresponds exactly to the computation of the language of a totally rigid acyclic tree grammar. This work constitutes a proof-of-concept result for a new connection between proof theory and formal language theory arising from exact characterisations of classes of proofs by classes of grammars. In principle, such a result is conceivable for any proof system that possesses an Herbrand-like theorem, i.e. even full higher-order logic as in [16]. The challenge consists in finding an appropriate type of grammars.

Acknowledgements. The author would like to thank the anonymous referees for helpful comments and suggestions that led to significant improvements of this paper.

References

1. Baaz, M., Leitsch, A.: Cut Normal Forms and Proof Complexity. Annals of Pure and Applied Logic 97, 127–177 (1999)
2. Buss, S.R.: On Herbrand's Theorem. In: Leivant, D. (ed.) LCC 1994. LNCS, vol. 960, pp. 195–209. Springer, Heidelberg (1995)
3. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata: Techniques and Applications (2007), `http://www.grappa.univ-lille3.fr/tata` (release October 12, 2007)
4. Cook, S., Nguyen, P.: Logical Foundations of Proof Complexity. Perspectives in Logic. Cambridge University Press (2010)
5. Filiot, E., Talbot, J.-M., Tison, S.: Satisfiability of a Spatial Logic with Tree Variables. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 130–145. Springer, Heidelberg (2007)
6. Filiot, E., Talbot, J.-M., Tison, S.: Tree Automata with Global Constraints. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 314–326. Springer, Heidelberg (2008)
7. Filiot, E., Talbot, J.M., Tison, S.: Tree Automata With Global Constraints. International Journal of Foundations of Computer Science 21(4), 571–596 (2010)
8. Gentzen, G.: Untersuchungen über das logische Schließen. Mathematische Zeitschrift 39, 176–210, 405–431 (1934-1935)
9. Herbrand, J.: Recherches sur la théorie de la démonstration. Ph.D. thesis, Université de Paris (1930)

10. Hetzl, S.: Proofs as Tree Languages (preprint), http://hal.archives-ouvertes.fr/hal-00613713/
11. Hetzl, S.: On the form of witness terms. Archive for Mathematical Logic 49(5), 529–554 (2010)
12. Hetzl, S., Leitsch, A., Weller, D.: Towards Algorithmic Cut-Introduction (submitted)
13. Jacquemard, F., Klay, F., Vacher, C.: Rigid Tree Automata. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 446–457. Springer, Heidelberg (2009)
14. Jacquemard, F., Klay, F., Vacher, C.: Rigid tree automata and applications. Information and Computation 209, 486–512 (2011)
15. Kohlenbach, U.: Applied Proof Theory: Proof Interpretations and their Use in Mathematics. Springer, Heidelberg (2008)
16. Miller, D.: A Compact Representation of Proofs. Studia Logica 46(4), 347–370 (1987)
17. Orevkov, V.P.: Lower bounds for increasing complexity of derivations after cut elimination. Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta 88, 137–161 (1979)
18. Pudlák, P.: The Lengths of Proofs. In: Buss, S. (ed.) Handbook of Proof Theory, pp. 547–637. Elsevier (1998)
19. Schwichtenberg, H., Troelstra, A.S.: Basic Proof Theory. Cambridge University Press (1996)
20. Shallit, J., Breitbart, Y.: Automaticity I: Properties of a Measure of Descriptional Complexity. Journal of Computer and System Sciences 53, 10–25 (1996)
21. Shallit, J., Wang, M.W.: Automatic complexity of strings. Journal of Automata, Languages and Combinatorics 6(4), 537–554 (2001)
22. Shoenfield, J.R.: Mathematical Logic, 2nd edn. AK Peters (2001)
23. Statman, R.: Lower bounds on Herbrand's theorem. Proceedings of the American Mathematical Society 75, 104–107 (1979)