# Applying Tree Languages in Proof Theory

### Stefan Hetzl Institute of Discrete Mathematics and Geometry Vienna University of Technology

#### LATA 2012 Language and Automata Theory and Applications

A Coruña, Spain

March 6, 2012

1/14

### Proof theory

- $\blacktriangleright\,$  Hilbert's Programme, Foundations of Mathematics,  $\sim\,$  1920s
- Study mathematical proof as formal objects (i.e. strings)

## Motivation

- Proof theory
  - $\blacktriangleright\,$  Hilbert's Programme, Foundations of Mathematics,  $\sim\,$  1920s
  - Study mathematical proof as formal objects (i.e. strings)
- Proof mining
  - Extract concrete information from abstract proofs
  - Example: proof of  $\exists x (x = f(x))$ . Find such x, a "witness".

- Proof theory
  - $\blacktriangleright\,$  Hilbert's Programme, Foundations of Mathematics,  $\sim\,$  1920s
  - Study mathematical proof as formal objects (i.e. strings)
- Proof mining
  - Extract concrete information from abstract proofs
  - Example: proof of  $\exists x (x = f(x))$ . Find such x, a "witness".
  - ▶ **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  $x^y = \sqrt{2}^{\sqrt{2} \cdot \sqrt{2}} = \sqrt{2}^2 = 2 \in \mathbb{Q}$ .  $\Box$
  - In general: a finite set of witnesses

(日) (個) (E) (E) (E)

- Proof theory
  - $\blacktriangleright\,$  Hilbert's Programme, Foundations of Mathematics,  $\sim\,$  1920s
  - Study mathematical proof as formal objects (i.e. strings)
- Proof mining
  - Extract concrete information from abstract proofs
  - Example: proof of  $\exists x (x = f(x))$ . Find such x, a "witness".
  - ▶ **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  $x^y = \sqrt{2}^{\sqrt{2} \cdot \sqrt{2}} = \sqrt{2}^2 = 2 \in \mathbb{Q}$ .  $\Box$
  - In general: a finite set of witnesses, i.e. a finite tree language!

- Proof mining: cut-elimination
- Rigid tree languages
- From proofs to grammars
- From grammars to proofs

Cut: formalisation of the use of a lemma

$$\frac{T \vdash A \quad T, A \vdash B}{T \vdash B} \text{ cut}$$

- Cut-elimination: stepwise transformation of proof
- Cut-free proof: possible to read of witnesses
- Witnesses for  $T \vdash \exists x A$ :  $t_1, \ldots, t_n$  s.t.  $T \vdash \bigvee_{i=1}^n A[x \setminus t_i]$ .

Cut: formalisation of the use of a lemma

$$\frac{T \vdash A \quad T, A \vdash B}{T \vdash B} \text{ cut}$$

- Cut-elimination: stepwise transformation of proof
- Cut-free proof: possible to read of witnesses
- Witnesses for  $T \vdash \exists x A$ :  $t_1, \ldots, t_n$  s.t.  $T \vdash \bigvee_{i=1}^n A[x \setminus t_i]$ .
- ► Basic idea of this talk: proof  $\pi$  with cuts  $\longrightarrow^{\text{cut-elimination}}$  cut-free proof  $\pi^*$   $\downarrow$   $\downarrow$   $\downarrow$ grammar  $G(\pi)$   $\longrightarrow^{\text{defines}}$   $\stackrel{\text{utnesses of } \pi^*}{= L(G(\pi))}$

<ロ> (四) (四) (三) (三) (三) (三)

### $\checkmark$ Proof mining: cut-elimination

- Rigid tree languages
- From proofs to grammars
- From grammars to proofs

### Tree Automata with Equality Contraints

- ▶ Local equality contraints, e.g.  $f(q_1, q_1) \stackrel{1=2}{\rightarrow} q_2$
- Global equality contraints via states, e.g. TAGED [Filiot, Talbot, Tison '07]
- Rigid tree automata [Jacquemard, Clay, Vacher '09]
  - subclass of TAGED
  - ▶ Rigid tree automaton  $\langle Q, R, F, \Delta \rangle$  where  $R \subseteq Q$
  - Rigidity condition on run  $r : Pos(t) \rightarrow Q$ :  $\forall p_1, p_2 \in Pos(t)$  with  $r(p_1) = r(p_2) \in R$ :  $t|_{p_1} = t|_{p_2}$ .
  - L(A) = all terms which have runs satisfying rigidity condition

# **Rigid Tree Grammars**

- ► Rigid tree grammar  $\langle \alpha, N, R, \Sigma, P \rangle$  where  $R \subseteq N$  rigid
- ► Rigidity condition on derivation: if two productions with β ∈ R as left hand side are applied at positions p<sub>1</sub>, p<sub>2</sub>, then t|<sub>p1</sub> = t|<sub>p2</sub>.
- Example: α → f(β, β), β → g(γ), γ → a | g(γ) with R = {β} has L = {f(g<sup>n</sup>(a), g<sup>n</sup>(a)) | n ≥ 1}.

# **Rigid Tree Grammars**

- ► Rigid tree grammar  $\langle \alpha, N, R, \Sigma, P \rangle$  where  $R \subseteq N$  rigid
- ► Rigidity condition on derivation: if two productions with β ∈ R as left hand side are applied at positions p<sub>1</sub>, p<sub>2</sub>, then t|<sub>p1</sub> = t|<sub>p2</sub>.
- Example: α → f(β, β), β → g(γ), γ → a | g(γ) with R = {β} has L = {f(g<sup>n</sup>(a), g<sup>n</sup>(a)) | n ≥ 1}.
- ► **Theorem.** *L* language of a rigid tree grammar iff *L* language of rigid tree automaton.

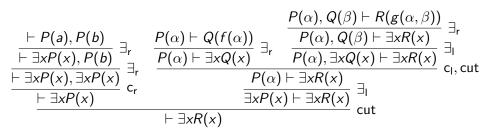
# **Rigid Tree Grammars**

- ► Rigid tree grammar  $\langle \alpha, N, R, \Sigma, P \rangle$  where  $R \subseteq N$  rigid
- ► Rigidity condition on derivation: if two productions with β ∈ R as left hand side are applied at positions p<sub>1</sub>, p<sub>2</sub>, then t|<sub>p1</sub> = t|<sub>p2</sub>.
- Example: α → f(β, β), β → g(γ), γ → a | g(γ) with R = {β} has L = {f(g<sup>n</sup>(a), g<sup>n</sup>(a)) | n ≥ 1}.
- Theorem. L language of a rigid tree grammar iff L language of rigid tree automaton.
- **Definition.** A grammer is called *totally rigid* if N = R.
- ▶ Definition. A grammar is called acyclic if there is no derivation β → t with β ∈ V(t)
- this paper: totally rigid acyclic tree grammars (!)

- $\checkmark$  Proof mining: cut-elimination  $\checkmark$  Rigid tree languages
- From proofs to grammars
- From grammars to proofs

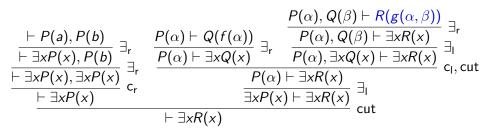
- Definition. Given a proof π, define a totally rigid acyclic tree grammar G(π). (next slide: example)
- Definition. A proof is called *simple* if every cut-formula contains at most one quantifier.
- **Theorem**. Let  $\pi$  be a simple proof of  $T \vdash \exists x A$  with A quantifier-free. Then  $L(G(\pi)) = \{A[x \setminus t_1], \ldots, A[x \setminus t_n]\}$  and  $T \vdash \bigvee_{i=1}^n A[x \setminus t_i]$  is provable.

(in other words:  $L(G(\pi))$  contains the witnesses for  $\exists x A$ )



10/14

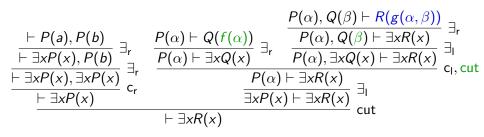
$$\mathsf{G}(\pi) = \langle \varphi, R, \Sigma, P \rangle$$
 where  $R = \{\varphi, \alpha, \beta\}$  and  $P = \{$ 



(日)

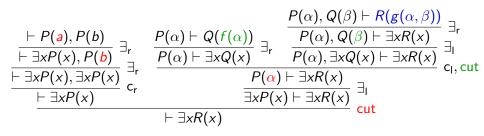
10/14

 $\begin{aligned} \mathsf{G}(\pi) &= \langle \varphi, R, \Sigma, P \rangle \text{ where } R = \{\varphi, \alpha, \beta\} \text{ and } \\ P &= \{\varphi \rightarrow R(\mathbf{g}(\alpha, \beta)) \end{aligned}$ 



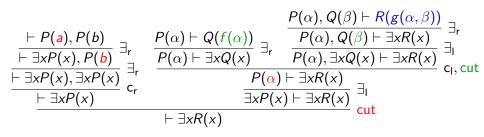
$$\begin{aligned} \mathsf{G}(\pi) &= \langle \varphi, R, \Sigma, P \rangle \text{ where } R = \{\varphi, \alpha, \beta\} \text{ and } \\ P &= \{\varphi \to R(g(\alpha, \beta)), \ \beta \to f(\alpha) \end{aligned}$$

> < □ > < 直 > < 直 > < 直 > 三 の Q (P) 10/14



$$\begin{aligned} \mathsf{G}(\pi) &= \langle \varphi, R, \Sigma, P \rangle \text{ where } R = \{\varphi, \alpha, \beta\} \text{ and } \\ P &= \{\varphi \to R(g(\alpha, \beta)), \ \beta \to f(\alpha), \alpha \to a, \alpha \to b\} \end{aligned}$$

・ 4 部 ・ 4 語 ト 4 語 ト 語 の Q ペ 10/14



$$G(\pi) = \langle \varphi, R, \Sigma, P \rangle \text{ where } R = \{\varphi, \alpha, \beta\} \text{ and } P = \{\varphi \to R(g(\alpha, \beta)), \beta \to f(\alpha), \alpha \to a, \alpha \to b\}$$

Hence  $L(G(\pi)) = \{R(g(a, f(a))), R(g(b, f(b)))\}$ 

√ Proof mining: cut-elimination
√ Rigid tree languages
√ From proofs to grammars
▶ From grammars to proofs

Given grammar find proof!

Caveat: L(G) is a set of terms,  $L(G(\pi))$  is a set of formulas  $\Rightarrow$  Wrap up L(G) using a new predicate symbol

- Theorem. For every totally rigid acyclic tree grammar G = ⟨β, R, Σ, P⟩ there is a simple proof π with G(π) = ⟨α, R ∪ {α}, Σ, P ∪ {α → Q(β)}⟩ s.t. cut-elimination of π computes L(G(π)).
- ⇒ Compression power of totally rigid acyclic tree grammars corresponds *exactly* to that of simple proofs.
- $\Rightarrow$  Characterisation of class of proofs by class of grammars.

Proofs and tree languages are intimately related

Applications / Future Work:

- Proof mining using tree grammars
- Cut-introduction (LPAR paper)
- Lower bounds on proofs
- Operations on languages get proof-theoretic meaning

- Go beyond simple proofs
- Does there exist a finite set T of terms s.t. every totally rigid acyclic tree grammar G with L(G) = T has |G| = |T|. (Uncompressible term-set ⇒ lower bounds on proof length)
- What is the complexity of the problem: Given finite set T of terms, find minimal G with L(G) = T? (Cut-introduction)
- Further cut-introduction algorithms