A positive perspective on term representation

Dale Miller and Jui-Hsuan Wu

Inria Saclay & LIX, Institut Polytechnique de Paris

CSL 2023, Warsaw, Poland

15 February 2023

Roles for proof theory: 1 - 4 of 6

1. Proof theory was started as a way to address the crisis of consistency in mathematics: Hilbert, Frege, Russell, Gentzen, ..., Voevodsky. Hilbert-Frege proof systems are eminently trustable: you only need to trust a small set of axioms and inferences, and perform simple computations.
2. Ordinal analysis of systems of arithmetic: Gentzen, Kreisel, Rathjen, Schütte, Pohlers, etc.
3. Constructive reasoning, program extraction, proof mining: Kohlenbach, Oliva, Hayashi, Schwichtenberg, etc.
4. Reverse mathematics, H. Friedman, S. Simpson, etc.

Ref: Rathjen and Sieg, **Proof Theory**, The Stanford Encyclopedia of Philosophy

Roles for proof theory: 5 of 6

5. Proof theoretic semantics.

   Use inference rules and proofs to provide meaning instead of using references to truth, i.e., in contrast to using set theory, type theory, category theory, and denotational semantics.

   - Gentzen, Prawitz, Schreoder-Heister, etc. used this approach to define logical connectives and their properties.
   - Miller, Nadathur, Scedrov, Pfenning, Pym, Harland, Andreoli, Pareschi, Hodas, etc, used this approach to define the meaning of logic programming languages. Also the SOS of Plotkin, etc.

Ref: Schroeder-Heister, **Proof Theoretic Semantics**, The Stanford Encyclopedia of Philosophy

Ref: Miller, **A Survey of the Proof-Theoretic Foundations of Logic Programming**, *Theory and Practice of Logic Prog*, 2022

Roles for proof theory: 6 of 6

6. Principled approach to syntax.

   - λ-tree syntax, mobility of binders (a.k.a. HOAS)
   - Focused proofs determine term structures. Cut-free focused proofs yield normal terms. Cut-elimination determines substitution.
     - Variant of the λ-calculi: Herbelin, Dyckhoff, Lengrand, Espírito Santo, Scherer, etc.
     - Their work on the λ-calculus relies on *negative polarity*, possibly with disjunction and existentials (positive connectives).

Our project continues this line of work by putting *positive polarity* at the center.

Term structures

Terms (or expressions) are used in various settings.

- **Mathematics:** equations, formulas, proofs
- **Programming:** AST, types, intermediate representations
- **Proof assistants:** formulas, types, proofs

Term structures

Terms (or expressions) are used in various settings.

- **Mathematics:** equations, formulas, proofs
- **Programming:** AST, types, intermediate representations
- **Proof assistants:** formulas, types, proofs

Terms come in *different formats*:

```
(1 + 2) + (1 + (1 + 2))
let x = 1 + 2 in let y = (1 + (1 + 2)) in x + y
let x = 1 + 2 in let y = 1 + x in x + y
```

Terms can be given *graphical* representations: labeled trees, directed acyclic graphs (DAGs)

Term structures

Terms (or expressions) are used in various settings.

- **Mathematics:** equations, formulas, proofs
- **Programming:** AST, types, intermediate representations
- **Proof assistants:** formulas, types, proofs

Terms come in *different formats*:

```
(1 + 2) + (1 + (1 + 2))
let x = 1 + 2 in let y = (1 + (1 + 2)) in x + y
let x = 1 + 2 in let y = 1 + x in x + y
```

Terms can be given *graphical* representations: labeled trees, directed acyclic graphs (DAGs)

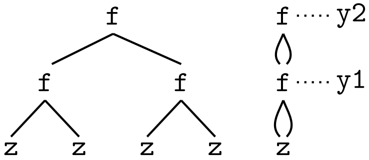
There are numerous *operations* on terms: equality, substitution, evaluation, unification, transformations

Things can get *tricky*: bindings? meta-variables? nested quantification? Skolemization?

Two examples of term structures

```
(f (f z z) (f z z))

name y1 = (f  z  z) in
name y2 = (f y1 y1) in y2.
```

These terms can be displayed as a labeled tree and a DAG.



Proof theory for term representations

NB: We are concerned primarily with *proofs-as-terms* and not *proofs-as-programs*!

NB: We are going against the mantra in dependently typed λ-terms: "*proofs are just terms.*" Instead, we are considering "*terms are proofs.*"

Proof theory for term representations

NB: We are concerned primarily with *proofs-as-terms* and not *proofs-as-programs*!

NB: We are going against the mantra in dependently typed λ-terms: "*proofs are just terms.*" Instead, we are considering "*terms are proofs.*"

Which proof system to use? Gentzen [1935] provided two choices.

- *Natural deduction (NJ)*: too rigid; does not address sharing.
- *Sequent calculus (LJ)*: too low-level, noisy, and chaotic.

Proof theory for term representations

NB: We are concerned primarily with *proofs-as-terms* and not *proofs-as-programs*!

NB: We are going against the mantra in dependently typed λ-terms: "*proofs are just terms.*" Instead, we are considering "*terms are proofs.*"

Which proof system to use? Gentzen [1935] provided two choices.

- *Natural deduction (NJ)*: too rigid; does not address sharing.
- *Sequent calculus (LJ)*: too low-level, noisy, and chaotic.

Focused proof systems greatly improve the sequent calculus.

- Uniform proofs [M, Nadathur, Pfenning, & Scedrov 1991]
- Focused linear logic [Andreoli 1992]
- Focused intuitionistic logic *LJF* [Liang & M, 2009]

Focused proof systems construct *synthetic inference rules*. Different *polarizations* yield different normal forms of proofs.

The elements of focusing

We read sequent calculus rules from conclusion to premises.

rule application	invertible	vs	non-invertible
oracle	no information	vs	essential information
non-determinism	don't care	vs	don't know
phase	negative $\Uparrow$	vs	positive $\Downarrow$

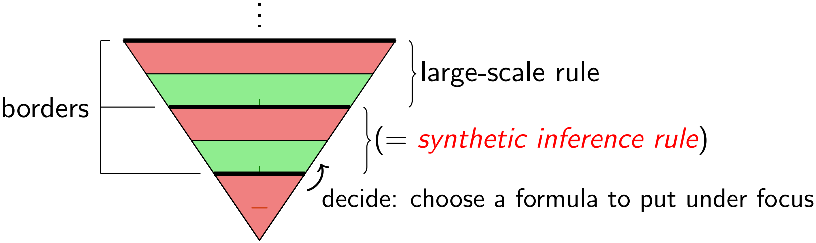
Focused proofs alternative between two phases.

First developed with linear logic where the positive/negative status for logical connectives is *unambiguous*.

Later: applied to LJ and LK: LJT, LJQ, LKT, LKQ, etc. These were generalized by *LJF* and *LKF* [Liang & M, 2009].

Focusing allows for defining *synthetic inference rules* which use one positive phase below negative phases.

Two-phase structure and large-scale rules



$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{\vdash A^\perp, A}\,ax \qquad \dfrac{}{\vdash B, B^\perp}\,ax
}{\vdash A^\perp, B^\perp, A \otimes B}\,\otimes
}{\vdash A^\perp, B^\perp \oplus (C^\perp \otimes D^\perp), A \otimes B}\,\oplus_1
\qquad
\dfrac{
\dfrac{}{\vdash A^\perp, A}\,ax \qquad
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{\vdash C, C^\perp}\,ax \qquad \dfrac{}{\vdash D^\perp, D}\,ax
}{\vdash C, D, C^\perp \otimes D^\perp}\,\otimes
}{\vdash C, D, B^\perp \oplus (C^\perp \otimes D^\perp)}\,\oplus_2
}{\vdash C \parr D, B^\perp \oplus (C^\perp \otimes D^\perp)}\,\parr
}{\vdash A^\perp, B^\perp \oplus (C^\perp \otimes D^\perp), A \otimes (C \parr D)}\,\otimes
}{
\dfrac{\vdash A^\perp, B^\perp \oplus (C^\perp \otimes D^\perp), (A \otimes B) \,\&\, (A \otimes (C \parr D))}{\vdash A^\perp \parr (B^\perp \oplus (C^\perp \otimes D^\perp)), (A \otimes B) \,\&\, (A \otimes (C \parr D))}\,\parr
}\,\&
$$

The *LJF* system with only implication

Formulas are built using atomic formulas and implication.

In *LJF*, formulas are *polarized*.

- Implications are negative.
- Atomic formulas are either positive or negative.
  (forward-chaining / backchaining)

A polarized formula (resp. theory) is a formula together with an *atomic bias assignment* $\delta : \mathcal{A} \to \{+, -\}$.

Different polarizations do not affect provability, but they yield *different normal forms of proofs*.

Theorem: If a formula is provable in *LJF* for some polarization, then it is provable for all polarizations.

Sequents in a focused proof

$$\Gamma \Uparrow \Theta \vdash \Delta \Uparrow \Delta' \qquad \Gamma \Downarrow \Theta \vdash \Delta \Downarrow \Delta'$$

All four *zones* Γ, Θ, Δ, and Δ' are multisets of formulas.
The multiset union $\Delta \cup \Delta'$ is always a singleton.

Γ and Δ' are called the left and right *storage zones*.
Θ and Δ are called the left and right *staging areas*.

$\Gamma \Uparrow \cdot \vdash \cdot \Uparrow \Delta$ are called *border sequents*: these sequents form the conclusion and premises of synthetic inference rules.

Sequents in a focused proof

$$\Gamma \Uparrow \Theta \vdash \Delta \Uparrow \Delta' \qquad \Gamma \Downarrow \Theta \vdash \Delta \Downarrow \Delta'$$

All four *zones* Γ, Θ, Δ, and Δ' are multisets of formulas.
The multiset union $\Delta \cup \Delta'$ is always a singleton.

Γ and Δ' are called the left and right *storage zones*.
Θ and Δ are called the left and right *staging areas*.

$\Gamma \Uparrow \cdot \vdash \cdot \Uparrow \Delta$ are called *border sequents*: these sequents form the conclusion and premises of synthetic inference rules.

Notation conventions

- drop $\cdot \Downarrow$ and $\cdot \Uparrow$ when they appear on the right,
- drop $\Downarrow \cdot$ and $\Uparrow \cdot$ when they appear on the left.
- Thus, $\Gamma \Uparrow \cdot \vdash \cdot \Uparrow E$ can be written as $\Gamma \vdash E$. Border sequents in *LJF* resemble sequents in *LJ*.

The *LJF* system with only implication

Decide, Release, and Store Rules

$$\frac{N, \Gamma \Downarrow N \vdash A}{N, \Gamma \vdash A}\ D_l \qquad \frac{\Gamma \vdash P \Downarrow}{\Gamma \vdash P}\ D_r \qquad \frac{\Gamma \Uparrow P \vdash A}{\Gamma \Downarrow P \vdash A}\ R_l \qquad \frac{\Gamma \vdash N \Uparrow}{\Gamma \vdash N \Downarrow}\ R_r$$

$$\frac{\Gamma, C \Uparrow \Theta \vdash \Delta' \Uparrow \Delta}{\Gamma \Uparrow \Theta, C \vdash \Delta' \Uparrow \Delta}\ S_l \qquad \frac{\Gamma \Uparrow \Theta \vdash A}{\Gamma \Uparrow \Theta \vdash A \Uparrow}\ S_r$$

Initial Rules

$$\frac{\delta(A) = +}{A, \Gamma \vdash A \Downarrow}\ I_r \qquad \frac{\delta(A) = -}{\Gamma \Downarrow A \vdash A}\ I_l$$

Introduction Rules for Implication

$$\frac{\Gamma \vdash B \Downarrow \quad \Gamma \Downarrow B' \vdash A}{\Gamma \Downarrow B \supset B' \vdash A} \supset L \qquad \frac{\Gamma \Uparrow \Theta, B \vdash B' \Uparrow}{\Gamma \Uparrow \Theta \vdash B \supset B' \Uparrow} \supset R$$

P is positive, N is negative, C is negative or atomic.

The *LJF* system with only implication

Decide, Release, and Store Rules

$$\frac{N, \Gamma, N \vdash A}{N, \Gamma \vdash A}\ D_l \qquad \frac{\Gamma \vdash P}{\Gamma \vdash P}\ D_r \qquad \frac{\Gamma, P \vdash A}{\Gamma, P \vdash A}\ R_l \qquad \frac{\Gamma \vdash N}{\Gamma \vdash N}\ R_r$$

$$\frac{\Gamma, C, \Theta \vdash \Delta', \Delta}{\Gamma, \Theta, C \vdash \Delta', \Delta}\ S_l \qquad \frac{\Gamma, \Theta \vdash A}{\Gamma, \Theta \vdash A}\ S_r$$

Initial Rules

$$\frac{}{A, \Gamma \vdash A}\ I_r \qquad \frac{}{A, \Gamma \vdash A}\ I_l$$

Introduction Rules for Implication

$$\frac{\Gamma \vdash B \quad \Gamma, B' \vdash A}{\Gamma, B \supset B' \vdash A} \supset L \qquad \frac{\Gamma, \Theta, B \vdash B'}{\Gamma, \Theta \vdash B \supset B'} \supset R$$

P is positive, N is negative, C is negative or atomic.

The *LJF* system with only implication

Decide, Release, and Store Rules

$$\frac{N, \Gamma, N \vdash A}{N, \Gamma \vdash A}$$

Initial Rules

$$\frac{}{A, \Gamma \vdash A}$$

Introduction Rules for Implication

$$\frac{\Gamma \vdash B \quad \Gamma, B' \vdash A}{\Gamma, B \supset B' \vdash A} \supset L \qquad \frac{\Gamma, \Theta, B \vdash B'}{\Gamma, \Theta \vdash B \supset B'} \supset R$$

P is positive, N is negative, C is negative or atomic.

Synthetic inference rules

Synthetic inference rule = large-scale rule = $\Downarrow$-phase + $\Uparrow$-phase

A *left synthetic inference rule* for B is an inference rule of the form

$$\frac{\Gamma_1 \vdash A_1 \quad \ldots \quad \Gamma_n \vdash A_n}{\Gamma \vdash A} \; B$$

justified by a derivation (in LJF) of the form

$$\begin{array}{c} \Gamma_1 \Uparrow \cdot \vdash \cdot \Uparrow A_1 \qquad \ldots \qquad \Gamma_n \Uparrow \cdot \vdash \cdot \Uparrow A_n \\ \vdots \; \Uparrow \text{phase} \\ \vdots \; \Downarrow \text{phase} \\ \dfrac{\Gamma \Downarrow B \vdash A}{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow A} \; D_l, \text{ where } B \in \Gamma \end{array}$$

In our settings, there is a unique synthetic rule for every formula B.

Two definitions

The *order of a formula* is defined as follows:

- $ord(B) = 0$ if B is atomic and
- $ord(B \supset C) = \max(ord(B) + 1, ord(C))$.

For example, $ord(a \supset (b \supset c)) = 1$ and $ord((a \supset b) \supset c) = 2$.

We name two specific *atomic bias assignments*:

- $\delta^-(A) = -$ for all atomic A.
- $\delta^+(A) = +$ for all atomic A.

Axioms as rules

Let $\mathcal{T}$ be a finite set of formulas of order 1 or 2. Let δ be an atomic bias assignment. $LJ\lfloor\delta, \mathcal{T}\rfloor$ extends LJ with the left synthetic inference rules for $\mathcal{T}$: for every left synthetic inference rule

$$\frac{B, \Gamma_1 \vdash A_1 \quad \dots \quad B, \Gamma_n \vdash A_n}{B, \Gamma \vdash A}\ B$$

with $B \in \mathcal{T}$, the following inference rule is added to $LJ\lfloor\delta, \mathcal{T}\rfloor$.

$$\frac{\Gamma_1 \vdash A_1 \quad \dots \quad \Gamma_n \vdash A_n}{\Gamma \vdash A}\ B$$

Axioms as rules

Let $\mathcal{T}$ be a finite set of formulas of order 1 or 2. Let δ be an atomic bias assignment. $LJ\lfloor\delta,\mathcal{T}\rfloor$ extends LJ with the left synthetic inference rules for $\mathcal{T}$: for every left synthetic inference rule

$$\dfrac{B,\Gamma_1 \vdash A_1 \quad \dots \quad B,\Gamma_n \vdash A_n}{B,\Gamma \vdash A}\ B$$

with $B \in \mathcal{T}$, the following inference rule is added to $LJ\lfloor\delta,\mathcal{T}\rfloor$.

$$\dfrac{\Gamma_1 \vdash A_1 \quad \dots \quad \Gamma_n \vdash A_n}{\Gamma \vdash A}\ B$$

Theorem

$\mathcal{T},\Gamma \vdash A$ is provable in $LJ \Leftrightarrow \Gamma \vdash A$ is provable in $LJ\lfloor\delta,\mathcal{T}\rfloor$.

For related work, see Negri & von Plato, *Cut elimination in the presence of axioms*, BSL 1998.

An example

Let $\mathcal{T}$ be the collection of formulas

$$D_1 = a_0 \supset a_1, \; D_2 = a_0 \supset a_1 \supset a_2, \; \cdots, \; D_n = a_0 \supset \cdots \supset a_n, \; \cdots$$

where a_i are atomic.

An example

Let $\mathcal{T}$ be the collection of formulas

$$D_1 = a_0 \supset a_1,\ D_2 = a_0 \supset a_1 \supset a_2,\ \cdots,\ D_n = a_0 \supset \cdots \supset a_n,\ \cdots$$

where a_i are atomic.

Backchaining: The inference rules in $LJ\lfloor \delta^-, \mathcal{T} \rfloor$ include

$$\frac{\Gamma \vdash a_0 \qquad \cdots \qquad \Gamma \vdash a_{n-1}}{\Gamma \vdash a_n}$$

An example

Let $\mathcal{T}$ be the collection of formulas

$$D_1 = a_0 \supset a_1,\ D_2 = a_0 \supset a_1 \supset a_2,\ \cdots,\ D_n = a_0 \supset \cdots \supset a_n,\ \cdots$$

where a_i are atomic.

Backchaining: The inference rules in $LJ\lfloor \delta^-, \mathcal{T} \rfloor$ include

$$\frac{\Gamma \vdash a_0 \qquad \cdots \qquad \Gamma \vdash a_{n-1}}{\Gamma \vdash a_n}$$

Forwardchaining: The inference rules in $LJ\lfloor \delta^+, \mathcal{T} \rfloor$ include

$$\frac{\Gamma, a_0, \cdots, a_{n-1}, a_n \vdash A}{\Gamma, a_0, \cdots, a_{n-1} \vdash A}$$

Backchaining and Forward-chaining

What are the proofs of $a_0 \vdash a_n$?

Backchaining and Forward-chaining

What are the proofs of $a_0 \vdash a_n$?

When a_i are all given the negative bias, we have:

$$\frac{\Gamma \vdash a_0}{\Gamma \vdash a_1} \quad \frac{\Gamma \vdash a_0 \quad \Gamma \vdash a_1}{\Gamma \vdash a_2} \quad \cdots \quad \frac{\Gamma \vdash a_0 \quad \cdots \quad \Gamma \vdash a_{n-1}}{\Gamma \vdash a_n} \quad \cdots$$

The *unique* proof of $a_0 \vdash a_n$ has **exponential** size.

Backchaining and Forward-chaining

What are the proofs of $a_0 \vdash a_n$?

When a_i are all given the negative bias, we have:

$$\frac{\Gamma \vdash a_0}{\Gamma \vdash a_1} \quad \frac{\Gamma \vdash a_0 \quad \Gamma \vdash a_1}{\Gamma \vdash a_2} \quad \cdots \quad \frac{\Gamma \vdash a_0 \quad \cdots \quad \Gamma \vdash a_{n-1}}{\Gamma \vdash a_n} \quad \cdots$$

The *unique* proof of $a_0 \vdash a_n$ has **exponential** size.

When a_i are all given the positive bias, we have:

$$\frac{\Gamma, a_0, a_1 \vdash A}{\Gamma, a_0 \vdash A} \quad \frac{\Gamma, a_0, a_1, a_2 \vdash A}{\Gamma, a_0, a_1 \vdash A} \quad \cdots \quad \frac{\Gamma, a_0, \ldots, a_{n-1}, a_n \vdash A}{\Gamma, a_0, \ldots, a_{n-1} \vdash A}$$

The *smallest* proof of $a_0 \vdash a_n$ has **linear** size.

Annotating rules and proofs

Now we annotate the inference rules in the previous example.

$$\frac{\Gamma \vdash a_0}{\Gamma \vdash a_1} \qquad \frac{\Gamma \vdash a_0 \qquad \Gamma \vdash a_1}{\Gamma \vdash a_2} \qquad \cdots$$

$$\frac{\Gamma \vdash a_0 \qquad \cdots \qquad \Gamma \vdash a_{n-1}}{\Gamma \vdash a_n}$$

Consider the proofs of $a_0 \vdash a_4$.

Annotating rules and proofs

Now we annotate the inference rules in the previous example.

$$\frac{\Gamma \vdash t_0 : a_0}{\Gamma \vdash E_1 t_0 : a_1} \qquad \frac{\Gamma \vdash t_0 : a_0 \quad \Gamma \vdash t_1 : a_1}{\Gamma \vdash E_2 t_0 t_1 : a_2} \qquad \cdots$$

$$\frac{\Gamma \vdash t_0 : a_0 \quad \cdots \quad \Gamma \vdash t_{n-1} : a_{n-1}}{\Gamma \vdash E_n t_0 \cdots t_{n-1} : a_n}$$

Consider the proofs of $a_0 \vdash a_4$.

Annotating rules and proofs

Now we annotate the inference rules in the previous example.

$$\frac{\Gamma \vdash t_0 : a_0}{\Gamma \vdash E_1 t_0 : a_1} \qquad \frac{\Gamma \vdash t_0 : a_0 \quad \Gamma \vdash t_1 : a_1}{\Gamma \vdash E_2 t_0 t_1 : a_2} \qquad \cdots$$

$$\frac{\Gamma \vdash t_0 : a_0 \quad \cdots \quad \Gamma \vdash t_{n-1} : a_{n-1}}{\Gamma \vdash E_n t_0 \cdots t_{n-1} : a_n}$$

Consider the proofs of $d_0 : a_0 \vdash t : a_4$.

Annotating rules and proofs

Now we annotate the inference rules in the previous example.

$$\frac{\Gamma \vdash t_0 : a_0}{\Gamma \vdash E_1 t_0 : a_1} \qquad \frac{\Gamma \vdash t_0 : a_0 \qquad \Gamma \vdash t_1 : a_1}{\Gamma \vdash E_2 t_0 t_1 : a_2} \qquad \cdots$$

$$\frac{\Gamma \vdash t_0 : a_0 \qquad \cdots \qquad \Gamma \vdash t_{n-1} : a_{n-1}}{\Gamma \vdash E_n t_0 \cdots t_{n-1} : a_n}$$

Consider the proofs of $d_0 : a_0 \vdash t : a_4$.

The term t is

$$\begin{array}{l} (E_4\ (E_3\ (E_2\ (E_1\ d_0)\ (E_1\ d_0)) \\ \qquad\qquad (E_2\ (E_1\ d_0)\ (E_1\ d_0))) \\ \qquad (E_3\ (E_2\ (E_1\ d_0)\ (E_1\ d_0)) \\ \qquad\qquad (E_2\ (E_1\ d_0)\ (E_1\ d_0)))) \end{array}$$

Annotating rules and proofs

Now we annotate the inference rules in the previous example.

$$\frac{\Gamma, a_0, a_1 \vdash A}{\Gamma, a_0 \vdash A} \qquad \frac{\Gamma, a_0, a_1, a_2 \vdash A}{\Gamma, a_0, a_1 \vdash A} \qquad \cdots$$

$$\frac{\Gamma, a_0, \cdots, a_{n-1}, a_n \vdash A}{\Gamma, a_0, \cdots, a_{n-1} \vdash A}$$

Consider the proofs of $a_0 \vdash a_4$.

Annotating rules and proofs

Now we annotate the inference rules in the previous example.

$$\frac{\Gamma, x_0 : a_0, x_1 : a_1 \vdash t : A}{\Gamma, x_0 : a_0 \vdash F_1 x_0(\lambda x_1.t) : A} \qquad \frac{\Gamma, x_0 : a_0, x_1 : a_1, x_2 : a_2 \vdash t : A}{\Gamma, x_0 : a_0, x_1 : a_1 \vdash F_2 x_0 x_1(\lambda x_2.t) : A}$$

$$\frac{\Gamma, x_0 : a_0, \cdots, x_{n-1} : a_{n-1}, x_n : a_n \vdash t : A}{\Gamma, x_0 : a_0, \cdots, x_{n-1} : a_{n-1} \vdash F_n x_0 \cdots x_{n-1}(\lambda x_n.t) : A}$$

Consider the proofs of $d_0 : a_0 \vdash t : a_4$.

Annotating rules and proofs

Now we annotate the inference rules in the previous example.

$$\frac{\Gamma, x_0 : a_0, x_1 : a_1 \vdash t : A}{\Gamma, x_0 : a_0 \vdash F_1 x_0 (\lambda x_1 . t) : A} \qquad \frac{\Gamma, x_0 : a_0, x_1 : a_1, x_2 : a_2 \vdash t : A}{\Gamma, x_0 : a_0, x_1 : a_1 \vdash F_2 x_0 x_1 (\lambda x_2 . t) : A}$$

$$\frac{\Gamma, x_0 : a_0, \cdots, x_{n-1} : a_{n-1}, x_n : a_n \vdash t : A}{\Gamma, x_0 : a_0, \cdots, x_{n-1} : a_{n-1} \vdash F_n x_0 \cdots x_{n-1} (\lambda x_n . t) : A}$$

Consider the proofs of $d_0 : a_0 \vdash t : a_4$.

The term t annotating the shortest proof is

$$\begin{array}{ll}
(F_1\ d_0 & (\lambda x_1. \\
(F_2\ d_0\ x_1 & (\lambda x_2. \\
(F_3\ d_0\ x_1\ x_2 & (\lambda x_3. \\
(F_4\ d_0\ x_1\ x_2\ x_3 & (\lambda x_4.\ x_4))))))))
\end{array}$$

Encodings of untyped λ-terms: the theory

We use a primitive type (atomic formula) D for untyped λ-terms.

We fix the theory $\mathcal{T} = \{\Phi : D \supset (D \supset D), \Psi : (D \supset D) \supset D\}$ and consider proofs of sequents of the form

$$\mathcal{T}, x_1 : D, \cdots, x_k : D \vdash t : D$$

Encodings of untyped λ-terms: the theory

We use a primitive type (atomic formula) D for untyped λ-terms.

We fix the theory $\mathcal{T} = \{\Phi : D \supset (D \supset D), \Psi : (D \supset D) \supset D\}$ and consider proofs of sequents of the form

$$\mathcal{T}, x_1 : D, \cdots, x_k : D \vdash t : D$$

This theory is *inconsistent* in the sense that every formula over D and $\supset$ is provable from $\mathcal{T}$.

As a result, the usual way we speak of cut-elimination is now *trivialized!*

Encodings of untyped λ-terms: the synthetic rules

When D is given the negative bias, we have the following synthetic inference rules:

$$\frac{\Gamma \vdash D \qquad \Gamma \vdash D}{\Gamma \vdash D}\ \Phi$$

$$\frac{\Gamma, D \vdash D}{\Gamma \vdash D}\ \Psi$$

and the initial rule.

Encodings of untyped λ-terms: the synthetic rules

When D is given the negative bias, we have the following synthetic inference rules:

$$\frac{\Gamma \vdash t : D \qquad \Gamma \vdash u : D}{\Gamma \vdash D}\ \Phi$$

$$\frac{\Gamma, D \vdash D}{\Gamma \vdash D}\ \Psi$$

and the initial rule.

Encodings of untyped λ-terms: the synthetic rules

When D is given the negative bias, we have the following synthetic inference rules:

$$\frac{\Gamma \vdash t : D \qquad \Gamma \vdash u : D}{\Gamma \vdash \Phi\ t\ u : D}\ \Phi$$

$$\frac{\Gamma, D \vdash D}{\Gamma \vdash D}\ \Psi$$

and the initial rule.

Encodings of untyped λ-terms: the synthetic rules

When D is given the negative bias, we have the following synthetic inference rules:

$$\frac{\Gamma \vdash t : D \qquad \Gamma \vdash u : D}{\Gamma \vdash \Phi\ t\ u : D}\ \Phi$$

$$\frac{\Gamma, x : D \vdash t : D}{\Gamma \vdash D}\ \Psi$$

and the initial rule.

Encodings of untyped λ-terms: the synthetic rules

When D is given the negative bias, we have the following synthetic inference rules:

$$\frac{\Gamma \vdash t : D \qquad \Gamma \vdash u : D}{\Gamma \vdash \Phi\ t\ u : D}\ \Phi$$

$$\frac{\Gamma, x : D \vdash t : D}{\Gamma \vdash \Psi\ (\lambda x.t) : D}\ \Psi$$

and the initial rule.

Encodings of untyped λ-terms: the synthetic rules

When D is given the positive bias, we have the following synthetic inference rules:

$$\frac{\Gamma, D, D, D \vdash D}{\Gamma, D, D \vdash D}\ \Phi$$

$$\frac{\Gamma, D \vdash D \qquad \Gamma, D \vdash D}{\Gamma \vdash D}\ \Psi$$

and the initial rule.

Encodings of untyped λ-terms: the synthetic rules

When D is given the positive bias, we have the following synthetic inference rules:

$$\frac{\Gamma, x : D, y : D, z : D \vdash t : D}{\Gamma, D, D \vdash D}\ \Phi$$

$$\frac{\Gamma, D \vdash D \qquad \Gamma, D \vdash D}{\Gamma \vdash D}\ \Psi$$

and the initial rule.

Encodings of untyped λ-terms: the synthetic rules

When D is given the positive bias, we have the following synthetic inference rules:

$$\frac{\Gamma, x : D, y : D, z : D \vdash t : D}{\Gamma, x : D, y : D \vdash \Phi\ x\ y\ (\lambda z.t) : D}\ \Phi$$

$$\frac{\Gamma, D \vdash D \qquad \Gamma, D \vdash D}{\Gamma \vdash D}\ \Psi$$

and the initial rule.

Encodings of untyped λ-terms: the synthetic rules

When D is given the positive bias, we have the following synthetic inference rules:

$$\frac{\Gamma, x : D, y : D, z : D \vdash t : D}{\Gamma, x : D, y : D \vdash \Phi\ x\ y\ (\lambda z.t) : D}\ \Phi$$

$$\frac{\Gamma, x : D \vdash t : D \qquad \Gamma, y : D \vdash u : D}{\Gamma \vdash D}\ \Psi$$

and the initial rule.

Encodings of untyped λ-terms: the synthetic rules

When D is given the positive bias, we have the following synthetic inference rules:

$$\frac{\Gamma, x : D, y : D, z : D \vdash t : D}{\Gamma, x : D, y : D \vdash \Phi\ x\ y\ (\lambda z.t) : D}\ \Phi$$

$$\frac{\Gamma, x : D \vdash t : D \qquad \Gamma, y : D \vdash u : D}{\Gamma \vdash \Psi\ (\lambda x.t)\ (\lambda y.u) : D}\ \Psi$$

and the initial rule.

Two formats for untyped λ-terms

Two different polarity assignments give **two different term structures**: (The infix backslash is the syntax of λProlog and Abella for λ-abstraction.)

D is negative: yields top-down, tree-like structure

x	`nvar x`	x
$\Phi\ t\ u$	`napp t u`	$t\ u$
$\Psi\ (\lambda x.t)$	`nabs (x\t)`	$\lambda x.t$

D is positive: yields bottom-up, DAG structure

x	`pvar x`	x
$\Phi\ x\ y\ (\lambda z.t)$	`papp x y (z\t)`	$\mathit{name}\ z\ =\ \mathit{app}\ x\ y\ \mathit{in}\ t$
$\Psi\,(\lambda x.t)\,(\lambda y.s)$	`pabs (x\t) (y\s)`	$\mathit{name}\ y\ =\ \mathit{abs}(\lambda x.t)\ \mathit{in}\ s$

Some examples for the positive-bias syntax

`name y = app x x in name z = app y y in z`

- Arguments of `app` are all *names*

Some examples for the positive-bias syntax

```
name y = app x x in name z = app y y in z
```

- Arguments of app are all *names*

```
name y1 = app x x in name y2 = app x x in
name z = app y1 y2 in z
```

- *Redundant* naming

Some examples for the positive-bias syntax

```
name y = app x x in name z = app y y in z
```

- Arguments of app are all *names*

```
name y1 = app x x in name y2 = app x x in
name z = app y1 y2 in z
```

- *Redundant* naming

```
name y1 = app x x in name y2 = app y y in
name z = app y1 y1 in z
```

- *Vacuous* naming

Some examples for the positive-bias syntax

```
name y = app x x in name z = app y y in z
```

- Arguments of app are all *names*

```
name y1 = app x x in name y2 = app x x in
name z = app y1 y2 in z
```

- *Redundant* naming

```
name y1 = app x x in name y2 = app y y in
name z = app y1 y1 in z
```

- *Vacuous* naming

```
name y1 = app x x in name y2 = app y y in
name z = app y1 y2 in z
```

- *Parallel* naming

Some examples for the positive-bias syntax

```
name y = app x x in name z = app y y in z
```

- Arguments of app are all *names*

```
name y1 = app x x in name y2 = app x x in
name z = app y1 y2 in z
```

- *Redundant* naming

```
name y1 = app x x in name y2 = app y y in
name z = app y1 y1 in z
```

- *Vacuous* naming

```
name y1 = app x x and y2 = app y y in
name z = app y1 y2 in z
```

- *Parallel* naming (by introducing *multi-focusing*)

Cut-elimination for $LJ\lfloor\delta, \mathcal{T}\rfloor$

The following theorem[1] states that cut is admissible for the extensions of *LJ* with polarized theories based on synthetic inference rules.

Theorem (Cut admissibility for $LJ\lfloor\delta, \mathcal{T}\rfloor$)

Let $\mathcal{T}$ be a finite polarized theory of order 2 or less. Then the cut rule is admissible for the proof system $LJ\lfloor\delta, \mathcal{T}\rfloor$.

[1] S. Marin, D. Miller, E. Pimentel, and M. Volpe. **From axioms to synthetic inference rules via focusing.** *Annals of Pure and Applied Logic 173(5).*

Cut-elimination for $LJ\lfloor\delta, \mathcal{T}\rfloor$

The following theorem[1] states that cut is admissible for the extensions of LJ with polarized theories based on synthetic inference rules.

Theorem (Cut admissibility for $LJ\lfloor\delta, \mathcal{T}\rfloor$)

Let $\mathcal{T}$ be a finite polarized theory of order 2 or less. Then the cut rule is admissible for the proof system $LJ\lfloor\delta, \mathcal{T}\rfloor$.

The proof is based on a cut elimination procedure for LJF, and it yields the notion of *substitution* for terms.

[1] S. Marin, D. Miller, E. Pimentel, and M. Volpe. **From axioms to synthetic inference rules via focusing.** *Annals of Pure and Applied Logic 173(5).*

Cut-elimination for $LJ\lfloor\delta, \mathcal{T}\rfloor$

The following theorem[1] states that cut is admissible for the extensions of *LJ* with polarized theories based on synthetic inference rules.

Theorem (Cut admissibility for $LJ\lfloor\delta, \mathcal{T}\rfloor$)

Let $\mathcal{T}$ be a finite polarized theory of order 2 or less. Then the cut rule is admissible for the proof system $LJ\lfloor\delta, \mathcal{T}\rfloor$.

The proof is based on a cut elimination procedure for *LJF*, and it yields the notion of *substitution* for terms.

When we restrict to *atomic* cut formulas, the cut elimination procedure can be presented in a big-step style.

- Cuts are permuted with *synthetic rules* instead of *LJF* rules.

---

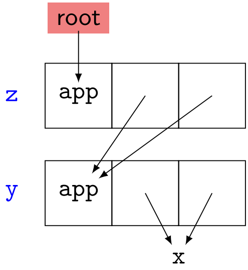
[1] S. Marin, D. Miller, E. Pimentel, and M. Volpe. **From axioms to synthetic inference rules via focusing.** *Annals of Pure and Applied Logic 173(5).*

Untyped λ-terms (substitution)

The cut-elimination procedure of *LJF* gives us the following definitions of substitutions.

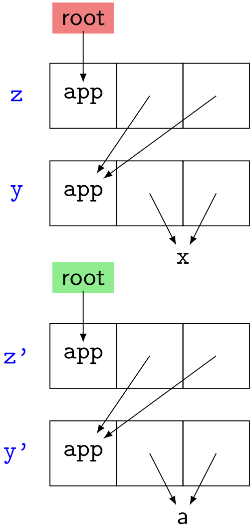
```
type nsubst   tm -> (val -> tm) -> tm -> o.
type psubst   tm -> (val -> tm) -> tm -> o.

nsubst T (x\ napp (R x) (S x)) (napp R' S') :-
               nsubst T R R', nsubst T S S'.
nsubst T (x\ nabs y\ R x y) (nabs y\ R' y) :-
          pi y\ nsubst T (x\ R x y) (R' y).
nsubst T (x\ nvar Y) (nvar Y).
nsubst T (x\ nvar x) T.

psubst (papp U V K) R (papp U V H) :-
                pi x\ psubst (K x) R (H x).
psubst (pabs S K) R (pabs S H) :-
                pi x\ psubst (K x) R (H x).
psubst (pvar U) R (R U).
```

An example




```
name y = app x x in
name z = app y y in
z
```

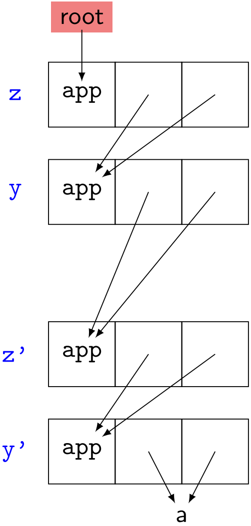
An example




```
name y = app x x in
name z = app y y in
z
```

```
name y' = app a a in
name z' = app y' y' in
z'
```

An example




```
name y = app x x in
name z = app y y in
z
```

```
name y' = app a a in
name z' = app y' y' in
name y = app z' z' in
name z = app y y in z
```

```
name y' = app a a in
name z' = app y' y' in
z'
```

Equality on terms

We have two different formats for untyped λ-terms.

When should two such expressions be considered the same?

Equality on terms

We have two different formats for untyped λ-terms.

When should two such expressions be considered the same?

"White box" approach: Look at the actual syntax of proofs.

- Transform proofs between systems: see Pimentel, Nigam, & Neto, *Multi-focused proofs with different polarity assignments*, LSFA 2015.
- Expensive since sharing is usually unwound.

Equality on terms

We have two different formats for untyped λ-terms.

When should two such expressions be considered the same?

“White box” approach: Look at the actual syntax of proofs.

- Transform proofs between systems: see Pimentel, Nigam, & Neto, *Multi-focused proofs with different polarity assignments*, LSFA 2015.
- Expensive since sharing is usually unwound.

“Black box” approach: Use concurrency theory notions of *traces* and *bisimulation*.

Traces in untyped λ-terms: Using the negative bias syntax

```
kind  tm                    type.
type  napp                  tm -> tm -> tm.
type  nabs                  (tm -> tm) -> tm.

kind  trace                 type.
type  left, right           trace -> trace.
type  bnd                   (trace -> trace) -> trace.

type  tm                    tm -> o.
type  trace                 tm -> trace -> o.


tm (napp M N) :- tm M, tm N.
tm (nabs R)   :- pi x\ tm x => tm (R x).

trace (napp M N) (left  P) :- trace M P.
trace (napp M N) (right P) :- trace N P.
trace (nabs R)   (bnd S)   :- pi x\ pi p\ trace x p => trace (R x) (S p).
```

The following theorem has a simple proof in Abella.

```
Theorem trace_eq :
   forall X Y, {tm X} ->
               (forall T, {trace X T} -> {trace Y T}) -> X = Y.
```

Traces in untyped λ-terms: Using the positive bias syntax

```
ptrace (papp U V K) P :-
   pi x\ (pi P\ ptrace (pvar x) (left  P) :- ptrace (pvar U) P) =>
         (pi P\ ptrace (pvar x) (right P) :- ptrace (pvar V) P) =>
   ptrace (K x) P.

ptrace (pabs R K) P :-
   pi x\  (pi Q\ ptrace (pvar x) (bnd Q) :-
                 pi p\ pi u\ ptrace (pvar u) p => ptrace (R u) (Q p))
           => ptrace (K x) P.
```

Traces in untyped λ-terms: Using the positive bias syntax

```
ptrace (papp U V K) P :-
   pi x\ (pi P\ ptrace (pvar x) (left  P) :- ptrace (pvar U) P) =>
         (pi P\ ptrace (pvar x) (right P) :- ptrace (pvar V) P) =>
   ptrace (K x) P.

ptrace (pabs R K) P :-
   pi x\  (pi Q\ ptrace (pvar x) (bnd Q) :-
                 pi p\ pi u\ ptrace (pvar u) p => ptrace (R u) (Q p))
           => ptrace (K x) P.
```

```
% Order 3
ptrace :- (ptrace :- ptrace) =>
          (ptrace :- ptrace) => ptrace.
% Order 4
ptrace :- (ptrace :- ptrace => ptrace) =>
   ptrace.
```

Traces in untyped λ-terms: Using the positive bias syntax

```
ptrace (papp U V K) P :-
   pi x\ (pi P\ ptrace (pvar x) (left  P) :- ptrace (pvar U) P) =>
         (pi P\ ptrace (pvar x) (right P) :- ptrace (pvar V) P) =>
   ptrace (K x) P.

ptrace (pabs R K) P :-
   pi x\  (pi Q\ ptrace (pvar x) (bnd Q) :-
                pi p\ pi u\ ptrace (pvar u) p => ptrace (R u) (Q p))
          => ptrace (K x) P.
```

```
% Order 3
ptrace :- (ptrace :- ptrace) =>
          (ptrace :- ptrace) => ptrace.
% Order 4
ptrace :- (ptrace :- ptrace => ptrace) =>
    ptrace.
```

However, trace-based equality tests are necessarily exponential in cost since all sharing is unfolded.

Graphical representations

The positive-bias syntax is better displayed graphically.

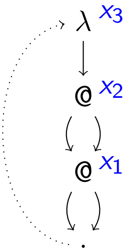
- `name` introduces *new nodes* and gives them a *label*.

Graphical representations

The positive-bias syntax is better displayed graphically.

- `name` introduces *new nodes* and gives them a *label*.

An example: the term $(\lambda x.(xx)(xx))$ as a graph:



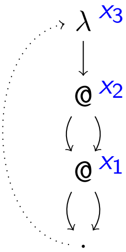

```
name x3 =
 abs (x\ name x1 = app x x in
          name x2 = app x1 x1 in x2) in x3
```

Graphical representations

The positive-bias syntax is better displayed graphically.

- `name` introduces *new nodes* and gives them a *label*.

An example: the term $(\lambda x.(xx)(xx))$ as a graph:

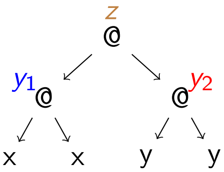



```
name x3 =
 abs (x\ name x1 = app x x in
          name x2 = app x1 x1 in x2) in x3
```

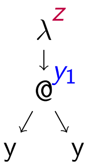
Bisimulation on such graphs can be checked in linear time: see A. Condoluci, B. Accattoli, & C. Sacerdoti Coen, *Sharing equality is linear*, *PPDP 2019*. The Abella specification is in our paper.

Graphical representations and parallel naming

Parallel naming can be captured by graphical representations:




```
name y1 = app x x  in  name y2 = app y y
in name z = app y1 y2 in z
name y2 = app y y  in  name y1 = app x x
in name z = app y1 y2 in z
```




```
name z = abs (x\ name y1 = app y y  in y1)
in z
name y1 = app y y  in  name z = abs  (x\ y1)
in z
```

Related and future work

- Change the way speak of cut elimination: it should be able partial proofs and not complete proofs.

Related and future work

- Change the way speak of cut elimination: it should be able partial proofs and not complete proofs.
- Can we rephrase the *concurrency-inspired* methods for checking term equality in *proof-theoretic* terms?

Related and future work

- Change the way speak of cut elimination: it should be able partial proofs and not complete proofs.
- Can we rephrase the *concurrency-inspired* methods for checking term equality in *proof-theoretic* terms?
- Conjecture: *Maximal* multifocused proofs are isomorphic to some graphical structure in the case of untyped λ-terms.

Related and future work

- Change the way speak of cut elimination: it should be able partial proofs and not complete proofs.
- Can we rephrase the *concurrency-inspired* methods for checking term equality in *proof-theoretic* terms?
- Conjecture: *Maximal* multifocused proofs are isomorphic to some graphical structure in the case of untyped λ-terms.
- Connection with the literature in programming language theory (administrative-normal form, etc.).

Related and future work

- Change the way speak of cut elimination: it should be able partial proofs and not complete proofs.
- Can we rephrase the *concurrency-inspired* methods for checking term equality in *proof-theoretic* terms?
- Conjecture: *Maximal* multifocused proofs are isomorphic to some graphical structure in the case of untyped λ-terms.
- Connection with the literature in programming language theory (administrative-normal form, etc.).
- Explore connections with other approaches to term structures: terms-as-graphs by Grabmayer and bigraphs by Milner.

Related and future work

- Change the way speak of cut elimination: it should be able partial proofs and not complete proofs.
- Can we rephrase the *concurrency-inspired* methods for checking term equality in *proof-theoretic* terms?
- Conjecture: *Maximal* multifocused proofs are isomorphic to some graphical structure in the case of untyped λ-terms.
- Connection with the literature in programming language theory (administrative-normal form, etc.).
- Explore connections with other approaches to term structures: terms-as-graphs by Grabmayer and bigraphs by Milner.
- Relate term structures to evaluation strategies: call-by-value is based on sharing, while call-by-name is not.