

# HOAS

Randy Pollack

Version of November 2, 2011

# Outline

- 1 HOAS by example
- 2 What is a Formal System?
- 3 A Simply Typed Framework
- 4 What Does it Mean?
- 5 Canonical LF
  - Judgement Forms and Rules
  - Hereditary Substitution
  - Judgements are Canonical
- 6 References

# Outline

- 1 HOAS by example
- 2 What is a Formal System?
- 3 A Simply Typed Framework
- 4 What Does it Mean?
- 5 Canonical LF
  - Judgement Forms and Rules
  - Hereditary Substitution
  - Judgements are Canonical
- 6 References

# HOAS by Example: Syntax of FOL

**Signature** to encode syntax of FOL:

```
i      : type.                % base type of Individuals
zero   : i.                   % constants in signature ...
suc    : i -> i.
plus   : i -> i -> i.

o      : type.                % base type of Propositions
imp    : o -> o -> o.         % constants in signature ...
and    : o -> o -> o.
or     : o -> o -> o.
forall : (i -> o) -> o.
eq     : i -> i -> o.
```

**Simply typed framework handles binding** (consider forall).

# HOAS by Example: Some Natural Deduction Rules

Signature to encode judgements using dependent types.

$$\begin{array}{c}
 [A] \\
 B \\
 \hline
 A \implies B
 \end{array}
 \quad
 \frac{(A \implies B) \quad A}{B}
 \quad
 \frac{A}{A \vee B}
 \quad
 \frac{A \vee B \quad [A] \quad [B] \quad C \quad C}{C}$$

`prf` : `o`  $\rightarrow$  `type`.

`impi` : `{A, B : o}` `(prf A`  $\rightarrow$  `prf B)`  $\rightarrow$  `prf (A imp B)`.

`impe` : `prf (A imp B)`  $\rightarrow$  `prf A`  $\rightarrow$  `prf B`.

`oril` : `prf A`  $\rightarrow$  `prf (A or B)`.

`ore` : `prf (A or B)`  $\rightarrow$   
`(prf A`  $\rightarrow$  `prf C)`  $\rightarrow$  `(prf B`  $\rightarrow$  `prf C)`  $\rightarrow$   
`prf C`.

**Dependent types** to represent judgements.

**Framework handles hypothetical and schematic judgements.**

# HOAS by Example: Some more rules

$$\frac{A(x)}{\forall x.A} \text{ (x fresh)} \quad \frac{\forall x.A}{A(t)} \quad \frac{(A \implies B) \quad \left[ \begin{array}{c} [A] \\ B \\ C \end{array} \right]}{C} \text{ (SH)}$$

```
foralli : {A : i -> o}
          ({x:i} prf (A x)) -> prf (forall A).
foralll : prf (forall A) -> {t:i} prf (A t).
-- A Schroeder-Heister version of imp elim
SH_impe : prf (A imp B) ->
          ((prf A -> prf B) -> prf C) ->
          -----
          prf C
```

**Framework handles freshness and substitution.**

The SH rule is clear in the framework (it is third order).

# A look at actual Twelf: Pure lambda terms

```

tm : type.      %name tm M x.      % type of lambda terms
app : tm -> tm -> tm.
lam : (tm -> tm) -> tm.

%abbrev @ = app.      % infix application
%infix left 10 @.

% relation of beta reduction on lambda terms
step : tm -> tm -> type.
s-beta : step ((lam F) @ M) (F M).  % contraction
s-1 : step (M1 @ M2) (M1' @ M2)    % congruence ...
      <- step M1 M1'.
s-2 : step (M1 @ M2) (M1 @ M2')
      <- step M2 M2'.

```

In s-beta,  $(F M)$  is **application at framework level**.

# A look at actual Twelf: Simple type assignment

```

tp : type.           %name tp A B.      % simple types
o  : tp.            % base type
arr : tp -> tp -> tp.

%abbrev => = arr.    % infix arrow
%infix right 10 =>.

tpng : tm -> tp -> type.
tapp : tpng (M1 @ M2) B
      <- tpng M1 (A => B)
      <- tpng M2 A.
tlam  : tpng (lam F) (A => B)
      <- ({x} tpng x A -> tpng (F x) B).

```

No typing contexts; instead **natural deduction style discharge**.

# Frameworks for relations over syntax

- Syntax of expressions ( $\lambda$ -terms, FOL terms and formulas) can be represented in HOAS style using simple types.
- Used dependent types (but predicative, logically weak) to represent relations over syntax (FOL provability, typing of  $\lambda$ -terms).
- There are at least two other approaches to representing relations over syntax:
  - The *two layer approach* of a logic over a simply-typed framework (Abella, Hybrid, ...)
  - Simply typed higher order logic (Isabelle, [Felty 1991]).
- Can we reason by induction over structure of expressions or derivations?
  - That is another long story.

# Outline

- 1 HOAS by example
- 2 What is a Formal System?**
- 3 A Simply Typed Framework
- 4 What Does it Mean?
- 5 Canonical LF
  - Judgement Forms and Rules
  - Hereditary Substitution
  - Judgements are Canonical
- 6 References

# Rules of Inference, Formal Systems

Let  $\mathcal{J}$  be a set (call these judgements).

A rule of inference is a partial function from  $\mathcal{J}$ -lists into  $\mathcal{J}$ .

If  $\mathcal{R}$  is a rule and  $A_1, \dots, A_n \in \mathcal{J}$ , we say

$$\frac{A_1, \dots, A_n}{\mathcal{R}[A_1, \dots, A_n]}$$

is an instance of  $\mathcal{R}$ , with premises  $A_1, \dots, A_n$  and conclusion  $\mathcal{R}[A_1, \dots, A_n]$ .

An axiom is a rule all of whose instances have an empty list of premises.

A formal system,  $\mathcal{F}$ , is a set of rules.

# Derivations, Theorems

Let  $\mathcal{F}$  be a formal system with language of judgements  $\mathcal{J}$ .

Derivation is a relation between  $\mathcal{J}$ -lists and  $\mathcal{J}$ , defined inductively

$$\frac{}{js \vdash r(js)} \quad r \in \mathcal{F} \qquad \frac{js \vdash j \quad jsl@(j::jsr) \vdash c}{jsl@(js@jsr) \vdash c}$$

$j \in \mathcal{J}$  is a theorem iff  $\vdash j$ . Also say that a theorem is provable, and a derivation of form  $\vdash j$  is a proof.

- **No schematic derivations** in this model.
  - By talking about instances we avoid talk of “variables” and “substitutions”.
- We will need rules for weakening and permutation of premises.

# Remarks on Formal Systems

- **Relational rules** Some presentations generalize the notion “rule” to decidable relations between  $\mathcal{J}$ -lists and  $\mathcal{J}$ .
  - Closer to informal notion; e.g. side conditions may refer to the conclusion as well as to the premises.
  - For relational rules, the definition of derivation is modified

$$\frac{}{js \vdash c} \quad r \in \mathcal{F}, r(js, c) \quad \dots$$

- **Number of premises** Some presentations restrict each rule to a fixed number of premises.
  - In reasonably strong meta systems lists of premises can still be represented.

# Derivable Rules

Let  $\mathcal{F}$  be a formal system over judgements  $\mathcal{J}$ .

A rule,  $\mathcal{R}$ , is derivable in  $\mathcal{F}$  if, for every instance of  $\mathcal{R}$ , there is a derivation in  $\mathcal{F}$  of its conclusion from its premises.

$$\frac{\begin{array}{c} \text{premises of } \mathcal{R} \\ \vdots \end{array}}{\text{conclusion of } \mathcal{R}}$$

- Informally, a rule is derivable in  $\mathcal{F}$  if it can be proved by gluing rules of in  $\mathcal{F}$  together.
- If  $\mathcal{R}$  is derivable in  $\mathcal{F}$ , then  $\mathcal{R}$  is derivable in any extension of  $\mathcal{F}$  by new axioms and rules.
- **Adding derivable rules to a formal system does not change its set of theorems.**

# A Derivable Rule: forgetting an assumption

$$\frac{G \vdash b}{G \vdash a \rightarrow b}$$

A derivation of its conclusion from its premise:

$$\frac{\frac{\frac{\frac{}{G, b, a \vdash b} \text{AXIOM}}{G, b \vdash a \rightarrow b} \text{INTRO}}{G \vdash b \rightarrow (a \rightarrow b)} \text{INTRO} \quad G \vdash b}{G \vdash a \rightarrow b} \text{ELIM}}$$

# Admissible Rules

A rule,  $\mathcal{R}$ , is admissible in  $\mathcal{F}$  if, for every instance of  $\mathcal{R}$ , whenever its premises are provable (from no assumptions), then so is its conclusion.

- Every derivable rule is admissible.
- $\mathcal{R}$  is admissible in  $\mathcal{F}$  **iff** adding  $\mathcal{R}$  to  $\mathcal{F}$  does not change  $\mathcal{F}$ 's set of theorems.
- Admissibility in  $\mathcal{F}$  is not necessarily preserved by extending  $\mathcal{F}$  with new axioms or rules.
- For axioms, admissible and derivable coincide.

# An Admissible Rule: weakening

$$\frac{K \vdash a}{G \vdash a} \quad (K \subseteq G)$$

Prove weakening is admissible by induction on the derivation of  $K \vdash a$ :

$$\frac{\frac{\frac{\overline{K, b, a \vdash b} \text{ AXIOM}}{K, b \vdash a \rightarrow b} \text{ INTRO}}{K \vdash b \rightarrow (a \rightarrow b)} \text{ INTRO}}{\frac{\frac{\overline{G, b, a \vdash b} \text{ AXIOM}}{G, b \vdash a \rightarrow b} \text{ INTRO}}{G \vdash b \rightarrow (a \rightarrow b)} \text{ INTRO}} \Longrightarrow$$

- We must eliminate the particular premise to construct a derivation of the particular conclusion.
- Weakening cannot be proved by gluing rules together.

# Structural Completeness

- By definition every derivable rule is admissible.
- In some formal systems (*structurally complete*) every admissible rule is derivable.
- Classical propositional logic (CPC) (Hilbert style rules, including a constant for False), is structurally complete.
  - Let  $\frac{A}{B}$  be a rule that is not derivable.
  - Since CPC is complete for truth tables, there is an assignment,  $\sigma$ , of truth values to propositional constants such that  $\sigma(A)$  is True and  $\sigma(B)$  is False.
  - Since CPC is sound for truth tables,  $\sigma(A)$  is a theorem and  $\sigma(B)$  is not a theorem.
  - Thus  $\frac{A}{B}$  is not admissible. □

# Logical strength

- The Edinburgh Logical Framework (ELF, used in Twelf) types no more functions than STLC.
  - Being dependently typed, it gives more informative types.
  - We can erase the dependency in an ELF derivation to get simple typing [Luo, Paulin-Mohring].
- Normalization of ELF (hence consistency) can be proven in Peano Arithmetic.
- The representation of an object system describes its syntax, not its logical strength.
  - In most representations, exactly the **derivable rules** of a judgement are typed in the framework.
  - To prove the admissible rules (requires induction over structure) is a whole different story.
- E.g. we can represent ZFC, and check derivations of ZFC judgements in ELF without committing to consistency of ZFC.

# Expressiveness

- ELF can represent many formal systems.
  - “Natural deduction” style systems are most naturally represented.
  - Many other styles can be “coded”: explicit contexts, explicit mention of variables, ...
- But some formal systems cannot be represented in ELF
  - A standard example is any linear system, since ELF has a multiplicative context.
    - A new *Linear Logical Framework LLF* must be defined.
  - Unary substitution comes for free, but what about simultaneous substitution ...
  - ... or binding a set of variables at once?

# Outline

- 1 HOAS by example
- 2 What is a Formal System?
- 3 A Simply Typed Framework**
- 4 What Does it Mean?
- 5 Canonical LF
  - Judgement Forms and Rules
  - Hereditary Substitution
  - Judgements are Canonical
- 6 References

# A simply typed framework: Signatures and Contexts

- Atomic types of the framework: A set of atomic types,  $A, B, \dots$   
**These are the types being represented.**
  - To represent pure  $\lambda$  terms, only base type is  $\text{tm}$ .
  - To represent FOL, base types are  $i$  and  $o$ .
- Simple types over the set of atomic types,  $S, T, \dots$
- A **signature**  $\Sigma$  of constants for the representation.
  - To represent pure  $\lambda$  terms, the signature is:
 
$$\text{app}:\text{tm}\rightarrow(\text{tm}\rightarrow\text{tm}), \text{ lam}:(\text{tm}\rightarrow\text{tm})\rightarrow\text{tm}.$$
  - To represent FOL,
 
$$\text{zero}:i, \text{ suc}:i\rightarrow i, \dots, \text{ forall}:(i\rightarrow o)\rightarrow o \dots$$
  - $\text{app}, \text{ lam}, \text{ zero}, \text{ forall}, \dots$  are a class of **framework constant** names.
- A context,  $\Gamma$ , as usual, binding atomic types to *framework variable* names.

# A simply typed framework: Typing rules

$$\frac{\Sigma \text{ valid} \quad \Gamma \text{ valid} \quad C:T \in \Sigma}{\Sigma; \Gamma \vdash C : T}$$

$$\frac{\Sigma \text{ valid} \quad \Gamma \text{ valid} \quad X:A \in \Gamma}{\Sigma; \Gamma \vdash X : A}$$

$$\frac{\Sigma; \Gamma \vdash m : A \rightarrow B \quad \Sigma; \Gamma \vdash n : A}{\Sigma; \Gamma \vdash m \cdot n : B}$$

$$\frac{\Sigma; (\Gamma, Y:A) \vdash m : B}{\Sigma; \Gamma \vdash [Y]m : A \rightarrow B}$$

- Contexts only contain atomic types.
  - We only need to abstract over objects being represented.

# Representation of object systems

- We represent pure  $\lambda$  terms by
  - specifying  $t_m$  as the only atomic type,
  - giving the signature

$$\Sigma \triangleq \text{app} : t_m \rightarrow t_m \rightarrow t_m, \text{lam} : (t_m \rightarrow t_m) \rightarrow t_m$$

- There is a **representation function** of informal lambda terms into framework:

$$\begin{aligned} \ulcorner x \urcorner &\triangleq x \\ \ulcorner m n \urcorner &\triangleq \text{app} \cdot \ulcorner m \urcorner \cdot \ulcorner n \urcorner \\ \ulcorner \lambda x. m \urcorner &\triangleq \text{lam} \cdot ([x] \ulcorner m \urcorner) \end{aligned}$$

$--$  and  $[ - ]$  are the framework level constructors.

- $\ulcorner - \urcorner$  should be an **adequate** representation:
  - A substitution preserving isomorphism between informal  $\lambda$  terms and **canonical forms** in the framework with signature  $\Sigma$ .

# Outline

- 1 HOAS by example
- 2 What is a Formal System?
- 3 A Simply Typed Framework
- 4 What Does it Mean?**
- 5 Canonical LF
  - Judgement Forms and Rules
  - Hereditary Substitution
  - Judgements are Canonical
- 6 References

# Adequate representation

- Let  $\Sigma$  be the signature of  $\lambda$  terms, and  $m$  be an (object level)  $\lambda$  term,
- Let  $\Gamma = x_1:t_m, \dots, x_n:t_m$ 
  - where  $x_1, \dots, x_n$  includes all the free variables of  $m$ .
- $\ulcorner \_ \urcorner$  **is well typed:**  $\Sigma; \Gamma \vdash \ulcorner m \urcorner : t_m$
- $\ulcorner \_ \urcorner$  **respects substitution:**  $\ulcorner [m/X]n \urcorner = [\ulcorner m \urcorner / X] \ulcorner n \urcorner$ .
  - **Substitution comes for free** in this style of framework.
- $\ulcorner \_ \urcorner$  **is injective.**
- $\ulcorner \_ \urcorner$  **is surjective?** Here we must be careful.
  - Surjection on **canonical forms** of the framework.

# Canonical Forms: $\alpha\beta$ -equivalence

- We want a compositional isomorphism between object language expressions and **canonical forms** of the LF.
  - We have implicitly been speaking up-to  $\alpha$ -equivalence.
- In order for the framework to handle substitution, we identify meta-terms up to  $\beta$ .

- The framework definition of  $\beta$ -contraction:

```
step : tm -> tm -> type.
```

```
s-beta : step ((lam F) @ M) (F M). % contraction
```

- `step ((lam.( $[x]^{\Gamma} m^{\neg}$ ) @  $^{\Gamma} n^{\neg}$ ) (( $[x]^{\Gamma} m^{\neg}$ ). $^{\Gamma} n^{\neg}$ ))`
- In object system  $(\lambda x.m) n \rightarrow^{\beta} [n/x]m$ .
- So the framework must identify  $(([x]^{\Gamma} m^{\neg}).^{\Gamma} n^{\neg})$  with  $[^{\Gamma} n^{\neg}/x]^{\Gamma} m^{\neg}$ , i.e. up to  $\beta$ -equivalence.
- **Canonical forms are  $\beta$ -normal.**

## Canonical Forms: $\eta$ -equivalence

- $\beta$  is not enough, we need  $\eta$ -equivalence in the notion of *canonical*.

- The two  $\beta$ -normal, well-typed framework terms

$$\text{suc} \quad [x:i] (\text{suc } x)$$

represent the same object expression (a function symbol of the language of FOL).

- $\eta$ -equivalence identifies these two terms

$$\lambda x.(M x) \stackrel{\eta}{\sim} M \quad (x \notin M)$$

- Problem:**  $\eta$  **reduction** is not Church–Rosser on raw terms of ELF:

$$[x : A]C \stackrel{\beta}{\longleftarrow} [x : A](((y : B)C)x) \stackrel{\eta}{\longrightarrow} [y : B]C \quad x \notin B, C$$

- Equality becomes mutually recursive with the typing relation.
- Thus, the meta theory of  $\beta\eta$ -equivalence for the ELF language (with type tags) is complicated.

## Canonical Forms: $\eta$ -equivalence

- Further, many extensions (singleton types, dependent pairs, linear types, ...) interact badly with  $\eta$ .
- Consider a unit type  $* \in 1$ .  $\eta$  equivalence is

$$m \stackrel{\eta}{\sim} * \quad (\text{if } m \in 1)$$

With  $\eta$ -reduction (left to right), confluence is lost, because for any variable  $f : S \rightarrow 1$  we have two normal forms:

$$\lambda x.* \stackrel{\eta}{\leftarrow} \lambda x.(f x) \stackrel{\eta}{\rightarrow} f$$

But  $\eta$ -expansion preserves confluence.

- $\eta$ -expansion (left to right) must be controlled by typing

$$\begin{array}{ll} f \stackrel{\eta}{\leftarrow} \lambda x.f x & \text{if } f : S \rightarrow T \text{ is a variable} \\ m \stackrel{\eta}{\leftarrow} * & \text{if } m : 1 \end{array}$$

With this we have  $\beta\eta$  strong normalization and confluence.

# Outline

- 1 HOAS by example
- 2 What is a Formal System?
- 3 A Simply Typed Framework
- 4 What Does it Mean?
- 5 Canonical LF**
  - Judgement Forms and Rules
  - Hereditary Substitution
  - Judgements are Canonical
- 6 References

# Canonical LF

- Accepts exactly the **canonical** judgements accepted by ELF.
- Developed by Watkins, Pfenning, and co-workers [WCPW02, HL06].
- This work precedes both Adams' TF and Plotkin's DMBEL.
- Only  $\beta$ -normal terms are syntactically formed.
- Only canonical terms ( $\eta$ -expanded,  $\beta$ -normal) are well-typed.
  - Equality is syntactic identity of expressions.
  - A notion of hereditary substitution is needed.
- Bi-directional typechecking: syntax directed and decidable.

# Canonical LF: Syntax

identifiers       $a$  (type constants),  $c$  (term constants),  
 $x$  (term variables)

kinds             $K ::= \text{type} \mid \Pi x:A.K$

types             $A, B ::= P \mid \Pi x:A.B$

atomic types     $P ::= a \mid P M$

terms             $M ::= R \mid \lambda x.M$

atomic terms     $R ::= c \mid x \mid R M$

contexts         $\Gamma, \Delta ::= \bullet \mid \Gamma, x:A$

signatures       $\Sigma ::= \bullet \mid \Sigma, a:K \mid \Sigma, c:A$

# Canonical LF: Judgement Forms

Validity of signatures and contexts:

$$\vdash \Sigma \text{ sig}$$

$$\vdash_{\Sigma} \Gamma \text{ cxt}$$

Checking correctness of kinds and types:

$$\Gamma \vdash_{\Sigma} K \text{ kind}$$

$$\Gamma \vdash_{\Sigma} A \text{ type}$$

Inferring kinds of atomic types, and the types of atomic terms:

$$\Gamma \vdash_{\Sigma} P \Rightarrow K$$

$$\Gamma \vdash_{\Sigma} R \Rightarrow A$$

Checking the types of terms:

$$\Gamma \vdash_{\Sigma} M \Leftarrow A$$

- *Bi-directional* typechecking, completely syntax directed.
- Cannot infer the type of  $\lambda x.M$ : no type annotation on  $x$ .

# Canonical LF: Rules

## Signatures

$$\frac{}{\vdash \bullet \text{ sig}}$$

$$\frac{\vdash \Sigma \text{ sig} \quad \vdash_{\Sigma} K \text{ kind}}{\vdash \Sigma, a:K \text{ sig}}$$

$$\frac{\vdash \Sigma \text{ sig} \quad \vdash_{\Sigma} A \text{ type}}{\vdash \Sigma, c:A \text{ sig}}$$

## Contexts

$$\frac{\vdash \Sigma \text{ sig}}{\vdash_{\Sigma} \bullet \text{ cxt}}$$

$$\frac{\vdash_{\Sigma} \Gamma \text{ cxt} \quad \Gamma \vdash_{\Sigma} A \text{ type}}{\vdash_{\Sigma} \Gamma, x:A \text{ cxt}}$$

# Canonical LF: Rules

Kinds

$$\frac{\vdash_{\Sigma} \Gamma \text{ cxt}}{\Gamma \vdash_{\Sigma} \text{ type kind}}$$

$$\frac{\Gamma, x:A \vdash_{\Sigma} K \text{ kind}}{\Gamma \vdash_{\Sigma} \Pi x:A. K \text{ kind}}$$

Types

$$\frac{\Gamma, x:A \vdash_{\Sigma} B \text{ type}}{\Gamma \vdash_{\Sigma} \Pi x:A. B \text{ type}}$$

$$\frac{\Gamma \vdash_{\Sigma} P \Rightarrow \text{ type}}{\Gamma \vdash_{\Sigma} P \text{ type}}$$

Atomic Types

$$\frac{\vdash_{\Sigma} \Gamma \text{ cxt}}{\Gamma \vdash_{\Sigma} a \Rightarrow K} \quad (a:K \in \Sigma)$$

$$\frac{\Gamma \vdash_{\Sigma} P \Rightarrow \Pi x:A. K \quad \Gamma \vdash_{\Sigma} M \Leftarrow A}{\Gamma \vdash_{\Sigma} P M \Rightarrow [M^{(A)^{-}} / x] K}$$

# Canonical LF: Rules

## Terms

$$\frac{\Gamma, x:A \vdash_{\Sigma} M \Leftarrow B}{\Gamma \vdash_{\Sigma} \lambda x.M \Leftarrow \Pi x:A.B}$$

$$\frac{\Gamma \vdash_{\Sigma} R \Rightarrow P}{\Gamma \vdash_{\Sigma} R \Leftarrow P} (*)$$

## Atomic Terms

$$\frac{\vdash_{\Sigma} \Gamma \text{ cxt}}{\Gamma \vdash_{\Sigma} x \Rightarrow A} (x:A \in \Gamma)$$

$$\frac{\vdash_{\Sigma} \Gamma \text{ cxt}}{\Gamma \vdash_{\Sigma} c \Rightarrow A} (c:A \in \Sigma)$$

$$\frac{\Gamma \vdash_{\Sigma} R \Rightarrow \Pi x:A.B \quad \Gamma \vdash_{\Sigma} M \Leftarrow A}{\Gamma \vdash_{\Sigma} R M \Rightarrow [M^{(A)^{-}} / x] B}$$

(\*) This restriction to atomic types,  $P$ , makes judgements canonical.

# Hereditary Substitution: Pure $\lambda$ -terms

Let  $L, M, N$  be lambda terms.

Let  $A, B, C$  be simple types

$$\begin{aligned}
 [M^A/x]x &= M^A \\
 [M^A/x]y &= y && \text{if } x \neq y \\
 [M^A/x](\lambda y.N) &= \lambda y.[M^A/x]N && y \text{ fresh} \\
 [M^A/x](L N) &= ([\hat{L}^B/y]M')^C && \text{if } \hat{L} = (\lambda y.M')^{B \rightarrow C} \\
 &= \hat{L} \hat{N} && \text{otherwise}
 \end{aligned}$$

where  $\hat{L} = [M^A/x]L$  and  $\hat{N} = [M^A/x]N$

- If  $M$  and  $N$  have no  $\beta$ -redexes, then  $[M/x]N$ 
  - is  $\beta$ -equal to the usual substitution  $[M/x]N$
  - contains no  $\beta$ -redexes
- **However  $[M/x]N$  may not terminate.**
- We use types to control how deep the substitution goes.

# Hereditary Substitution: **Simply typed terms** [Abe06]

Let  $L, M, N$  be lambda terms.

Let  $A, B, C$  be simple types

$$\begin{aligned}
 [M^A/x]x &= M^A \\
 [M^A/x]y &= y && \text{if } x \neq y \\
 [M^A/x](\lambda y.N) &= \lambda y.[M^A/x]N && y \text{ fresh} \\
 [M^A/x](L N) &= ([\hat{N}^B/y]M')^C && \text{if } \hat{L} = (\lambda y.M')^{B \rightarrow C} \\
 &= \hat{L} \hat{N} && \text{otherwise}
 \end{aligned}$$

where  $\hat{L} = [M^A/x]L$  and  $\hat{N} = [M^A/x]N$

- If  $M$  and  $N$  have no  $\beta$ -redexes, then  $[M/x]N$ 
  - is  $\beta$ -equal to the usual substitution  $[M/x]N$
  - contains no  $\beta$ -redexes **if the type annotation is correct**
- **Always terminates**: newly created substitutions operate on subterms, or have syntactically smaller types.

# Hereditary Substitution: Canonical LF

- LF types exactly the same lambda-terms as simple types ...
- ... just throw away the dependency.

$$\begin{aligned} (a)^- &= a \\ (P M)^- &= (P)^- \\ (\Pi x:A.B)^- &= (A)^- \rightarrow (B)^- \end{aligned}$$

Only the shape of the simple type matters: can take  $(a)^- = \bullet$ .

- One more problem: canonical terms cannot contain redexes.
  - Instead of returning a term with a redex (when the given type is incorrect), the canonical hereditary substitution algorithm must return failure.
- Finally,  $[M^{(A)^-} / x]_-$  is a congruence, generated by the replacement of term variables in terms, as explained above.

# Judgements are Canonical by Stratified Syntax

We want to know that no two derivable judgements differ only by  $\beta\eta$ .

- Let  $\Sigma = a:\text{type}$ .
  - $(a \rightarrow a)$  is a non-atomic type, syntactic class  $A$ , not class  $P$ .

Direction  $\Rightarrow$ :

- $f:(a \rightarrow a) \vdash_{\Sigma} f \Rightarrow (a \rightarrow a)$  is derivable,
- $f:(a \rightarrow a) \vdash_{\Sigma} \lambda x.f x \Rightarrow (a \rightarrow a)$  is not derivable:
  - $\lambda x.f x$  is of class  $M$ , not of class  $R$ ,
  - there are no rules of shape  $\Gamma \vdash_{\Sigma} M \Rightarrow A$ .

# Judgements are Canonical by Stratified Syntax

Direction  $\Leftarrow$  :

- $f:(a \rightarrow a) \vdash_{\Sigma} f \Leftarrow (a \rightarrow a)$  is not derivable.
  - $f$  is not of shape  $\lambda x.M$  and  $a \rightarrow a$  is not of shape P, so no rule applies.
- $f:(a \rightarrow a) \vdash_{\Sigma} \lambda x.f x \Leftarrow (a \rightarrow a)$  is derivable:  
 Let  $\Gamma = f:(a \rightarrow a), x:a$ .

$$\frac{
 \frac{f:(a \rightarrow a) \vdash_{\Sigma} a \Rightarrow \text{type}}{f:(a \rightarrow a) \vdash_{\Sigma} a \Leftarrow \text{type}}
 \quad
 \frac{
 \Gamma \vdash_{\Sigma} f \Rightarrow (a \rightarrow a) \quad
 \frac{\Gamma \vdash_{\Sigma} x \Rightarrow a}{\Gamma \vdash_{\Sigma} x \Leftarrow a} (*) \quad
 [x/_]a = a
 }{\Gamma \vdash_{\Sigma} f x \Rightarrow a} (*)
 }{\Gamma \vdash_{\Sigma} \lambda x.f x \Leftarrow (a \rightarrow a)}$$

(\*) At atomic type.

# Outline

- 1 HOAS by example
- 2 What is a Formal System?
- 3 A Simply Typed Framework
- 4 What Does it Mean?
- 5 Canonical LF
  - Judgement Forms and Rules
  - Hereditary Substitution
  - Judgements are Canonical
- 6 **References**

# References

- [Abe06] Andreas Abel. **Implementing a normalizer using sized heterogeneous types.**  
In *Workshop on Mathematically Structured Functional Programming, MSFP 2006*.
- [Ada08] Robin Adams. **Lambda-free logical frameworks.**  
arXiv:0804.1879v2 [cs.LO].
- [Gog05] Healdene Goguen. **A syntactic approach to eta equality in type theory.**  
In *POPL 2005*.
- [HL06] Robert Harper and Daniel R. Licata. **Mechanizing Metatheory in a Logical Framework.**  
Submitted for publication. Available from <http://www.cs.cmu.edu/~drl/>, Oct. 2006.
- [HP05] Robert Harper and Frank Pfenning. **On equivalence and canonical forms in the LF type theory.**  
*ACM Trans. on Computational Logic*, 6(1), 2005.
- [Luo03] Zhaohui Luo. **PAL<sup>+</sup> : a lambda-free logical framework.**  
*Journal of Functional Programming*, 13(2):317–338, 2003.
- [Sal] Anne Salvesen. **The Church-Rosser theorem for LF with beta/eta reduction.**  
Talk given at First Workshop on Logical Frameworks, Antibes, 1990.
- [WCPW02] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker.  
**A concurrent logical framework I: Judgments and properties.**  
Technical Report CMU-CS-02-101, Department of Computer Science, Carnegie Mellon University, 2002. Revised May 2003.
- [CNV11] James Cheney, Michael Norrish and René Vestergaard.  
**Formalizing adequacy: a case study for higher-order syntax.**  
*Journal of Automated Reasoning*, 2011.