Introduction to logic and typed λ-calculus

Frédéric Blanqui

INRIA

Tsinghua University, March 2008

Outline

First-order term algebra

we assume given a set $\mathcal{X}$ of variables x, y, ...
and a signature Σ, *i.e.* a set $\mathcal{F}$ of function symbols f, g, ...
equipped with an arity function $ar : \mathcal{F} \to \mathbb{N}$

a term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ is either:

- a variable x
- or a function symbol f of arity n ($ar(f) = n$) applied to n terms $t_1, \ldots, t_n$ written $ft_1 \ldots t_n$

Example of first-order term algebra

arithmetic expressions can be represented by taking for $\mathcal{F}$:

- 0 of arity 0 for zero
- s of arity 1 for successor
- $+$ of arity 2 for addition
- $\times$ of arity 2 for multiplication
- ...

examples of terms: 0, $s(s0)$, $s(s0) \times s(s0)$

Higher-order terms

by higher-order terms, we mean terms with binding constructions like in $\Sigma_{i=1}^{n} x_i$, $\iint_\Omega f(x,y)dxdy$, $\forall x P(x)$, …[Fiore-Plotkin-Turi 1999]

the binding arity of a function symbol f is a sequence of natural numbers $[k_1; \ldots; k_n]$ ($n \in \mathbb{N}$ is the arity of f), each k_i denoting the number of variables bounds in the i-th argument of f

- Σ whose arguments are 1, n and x_i has binding arity $[0;0;1]$ since i is bound in x_i
- $\iint$ whose arguments are Ω and $f(x,y)$ has binding arity $[0;2]$ since x and y are bound in $f(x,y)$
- $\forall$ whose argument is $P(x)$ has binding arity $[1]$ since x is bound in $P(x)$

Higher-order term algebra

we assume given a set $\mathcal{X}$ of variables x, y, ...
a set $\mathcal{F}$ of function symbols f, g, ... of fixed binding arity

a term is either:

- a variable x
- or a function symbol f of binding arity $[k_1; \ldots; k_n]$ applied to n terms $t_1, \ldots, t_n$ written $f(x_1^1 \ldots x_1^{k_1}.t_1) \ldots (x_n^1 \ldots x_n^{k_n}.t_n)$

examples: $\Sigma_{i=1}^{n} x_i$ is represented by $\Sigma 1 n(i.x_i)$, and $\iint_\Omega f(x,y)dxdy$ is represented by $\iint \Omega(xy.f(x,y))$

α-conversion

in higher-order term algebra, bound variables are not significative and can be renamed without changing the meaning:

$\Sigma_{i=1}^{n} x_i = \Sigma_{j=1}^{n} x_j$ and $\iint_\Omega f(x,y)dxdy = \iint_\Omega f(u,v)dudv$ and $\forall x P(x) = \forall y P(y)$

renaming of bound variables is called <u>α-conversion</u> and written $=_\alpha$

this is an equivalence relation

in fact, higher-order terms are usually defined as the set of α-equivalence classes

Free and bound variables

the set $FV(t)$ of variables free in t is defined as follows:

- $FV(x) = \{x\}$
- $FV(f(x_1^1 \ldots x_1^{k_1}.t_1) \ldots (x_n^1 \ldots x_n^{k_n}.t_n)) = (FV(t_1) \setminus \{x_1^1, \ldots, x_1^{k_1}\}) \cup \ldots \cup (FV(t_n) \setminus \{x_n^1, \ldots, x_n^{k_n}\})$

a term is closed if it has no free variable ($FV(t) = \emptyset$)

Higher-order substitution

with higher-order terms, substitution must take care of bound variables

example with a symbol λ of binding arity [1]:

- $(\lambda y.v)_x^u = \lambda y.v$ if $y = x$
- $(\lambda y.v)_x^u = \lambda y.v_x^u$ if $y \neq x$ and $y \notin FV(u)$
- $(\lambda y.v)_x^u = \lambda z.v_{y\,x}^{z\,u}$ if $y \neq x$ and $y \in FV(u)$ and $z \notin FV(v)$ requires an α-conversion to avoid variable capture

example: $(\lambda y.x)_x^y = \lambda z.y$

Example of higher-order algebra: the untyped λ-calculus

the pure untyped λ-calculus is the higher-order term algebra with the following symbols:

- λ of binding arity [1] for abstraction
- @ of binding arity [0; 0] for application

$@(t, u)$ is often simply written tu

the evaluation of a function application is called β-reduction:

$$(\lambda x.t)u \rightarrow_\beta t_x^u$$

λ-calculus has been invented by Alonzo Church in 1928

Computational power of the untyped λ-calculus

it is possible to express <u>any computable</u> function in λ-calculus
i.e. λ-calculus is Turing-complete

$0 = \lambda xy.y$ (iterate 0 time x on y)
$1 = \lambda xy.xy$ (iterate 1 time x on y)
$2 = \lambda xy.x(xy)$ (iterate 2 times x on y)
...
$+ = \lambda pqxy.px(qxy)$
$\times = \lambda pqxy.p(qx)y$
...

example: $2 + 2 \rightarrow_\beta^* 4$

Relations on terms

given two relations $\rightarrow_R$ and $\rightarrow_S$, let $\rightarrow_R\rightarrow_S$ be their composition:

$$t \rightarrow_R\rightarrow_S v \text{ if there is } u \text{ such that } t \rightarrow_R u \text{ and } u \rightarrow_S v$$

given a relation $\rightarrow_R$, we denote by:

- $\leftarrow_R$ its inverse: $t \leftarrow_R u$ if $u \rightarrow_R t$
- $\rightarrow_R^k$ its k iteration:
  - $\rightarrow_R^0$ is its reflexive closure ($t \rightarrow_R^0 u$ if $t = u$)
  - $\rightarrow_R^{k+1}$ is the composition of $\rightarrow_R$ and $\rightarrow_R^k$
- $\rightarrow_R^+$ its transitive closure ($\rightarrow_R^+ = \bigcup\{\rightarrow_R^k \mid k > 0\}$)
- $\rightarrow_R^*$ its reflexive and transitive closure ($\rightarrow_R^* = \bigcup\{\rightarrow_R^k \mid k \geq 0\}$)
- $=_R$ its reflexive, symmetric and transitive closure

Confluence and Church-Rosser properties

a relation R is:

- confluent if $(\forall tuv) t \rightarrow_R^* u, v \Rightarrow (\exists w) u, v \rightarrow_R^* w$
- Church-Rosser if $(\forall tu) t =_R u \Rightarrow (\exists w) t, u \rightarrow_R^* w$

these properties are equivalent

example: β-reduction is confluent [Church-Rosser 1936]

Non-terminating terms and fixpoints

a relation R is terminating (or well-founded, noetherian, strongly normalizing) if there is no infinite sequence of R steps $t_0 R t_1 R \ldots$

λ-calculus has non-terminating terms:
$(\lambda x.xx)(\lambda x.xx) \rightarrow_\beta (\lambda x.xx)(\lambda x.xx) \rightarrow_\beta \ldots$

λ-calculus has fixpoints:
with $Y = \lambda f.(\lambda x.fxx)(\lambda x.fxx)$, we have $Yf \rightarrow_\beta f(Yf) \rightarrow_\beta \ldots$

How to get rid of bound variables ?

bound variables creates technical difficulties (see the notion of substitution, ...)

some of these difficulties can be avoided by using a first-order representation of λ-terms

there are two approaches:

- Schönfinkel's combinators (1920)
- de Bruijn indices (1972)

Schönfinkel's combinatory logic (1920)

- purely applicative terms are the terms made of variables and applications only
- SK-terms are the closed terms made of the constants S and K and applications only

combinatorial completeness: for all purely applicative term t whose free variables are $x_1, \ldots, x_n$, there exist an SK-term A such that $Ax_1 \ldots x_n = t$ in the following first-order equational theory:

- $Sxyz = xz(yz)$
- $Kxy = x$

(A represents the λ-term $\lambda x_1 \ldots x_n t$)

de Bruijn indices (1972)

a bound variable x is replaced by the number of λ's one has to go through before reaching the one that binds x

- $\lambda x \lambda y y$ is represented by $\lambda\lambda 0$
- $\lambda x \lambda y x$ is represented by $\lambda\lambda 1$

then α-conversion boils down to syntactic equality

Explicit substitutions (1973 - today)

the atomic meta-level higher-order substitution can be defined in more atomic terms by extending the term algebra with substitutions and using suitable rules

$$
\begin{array}{rcll}
x[x/v] & = & v & \\
y[x/v] & = & y & \text{if } x \neq y \\
(tu)[x/v] & = & t[x/v]u[x/v] & \\
(\lambda yt)[x/v] & = & \lambda yt[x/v] & \\
t[x/v][y/w] & = & t[y/w][x/v[y/w]] &
\end{array}
$$

finding rules deciding this equational theory and having good properties motivated many researches (see [Kesner 2007])

Typed term algebra

pure untyped algebra can be restricted by considering a typing discipline

we assume given a set $\mathbb{T}$ of types

in arities, a natural number k is replaced by a type sequence of length k together with an output type, written $[T_1; \ldots; T_k]T$

example: let $\mathbb{T} = \{B, N\}$ for booleans and natural numbers

- $+$ has arity $[N; N]N$
- if has arity $[B; N; N]N$
- $\forall_N$ has arity $[[N]B]B$ (quantification over N)

Typed terms

well typed terms are defined as follows:

let Γ be a function mapping variables to types

$$(\text{var}) \quad \frac{x : T \in \Gamma}{\Gamma \vdash x : T}$$

$$(\text{fun}) \quad \frac{\begin{array}{c} ar(f) = [T_1; \ldots; T_n]B \\ T_i = [T_i^1; \ldots; T_i^{k_i}]U_i \\ \Gamma, x_i^1 : T_i^1, \ldots, x_i^{k_i} : T_i^{k_i} \vdash t_i : U_i \end{array}}{\Gamma \vdash f(x_1^1 \ldots x_1^{k_1}.t_1) \ldots (x_n^1 \ldots x_n^{k_n}.t_n) : B}$$

Simple types

the set of simple types $\mathbb{T}^{\rightarrow}(\mathcal{B})$ over a set of type constants $\mathcal{B}$ is the first-order term algebra with the following symbols:

- a symbol of arity 0 for every type constant
- the arrow type constructor $\rightarrow$ of arity 2

examples: B, $B \rightarrow B$, $(B \rightarrow B) \rightarrow B$, ...

Example of typed HO algebra: the simply-typed λ-calculus

the simply-typed λ-calculus is the simply-typed higher-order term algebra with the following symbols:

- λ_T^U of binding arity $[[T]U](T \rightarrow U)$
- $@_T^U$ of binding arity $[T \rightarrow U; T]U$

introduced by Church in 1937

it is confluent and terminating [Turing 1942]

Computational power of $\lambda^{\rightarrow}$

representing natural numbers as terms of type $(\iota \rightarrow \iota) \rightarrow (\iota \rightarrow \iota)$:
$0 = \lambda xy.y$ (iterate x 0 times on y)
$1 = \lambda xy.xy$ (iterate x 1 times on y)
$2 = \lambda xy.x(xy)$ (iterate x 2 times on y)
...
$\lambda^{\rightarrow}$ can express any element of the smallest set of functions from $\mathbb{N}^k$ to $\mathbb{N}$ closed for composition and containing polynomials and the characteristic functions of $\{0\}$ and $\mathbb{N} \setminus \{0\}$ [Schwichtenberg 1976]

examples: $+ = \lambda pqxy.px(qxy)$, $\times = \lambda pqxy.p(qx)y$, ...

Outline

Logical language

allows to denote objects and express facts about them

- objects are represented by terms of some term algebra
- facts about objects (propositions) are represented by terms of the following typed higher-order term algebra:
  - user-defined predicate symbols P, Q, ... of arity $[\iota; \dots; \iota]o$
  - $\bot : o$ (proposition always false)
  - $\neg : [o]o$ (negation)
  - $c : [o; o]o$ where $c \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$ (logical connectors)
  - $\kappa_\iota : [[\iota]o]o$ where $\kappa \in \{\exists, \forall\}$ (first-order quantifiers)

  where ι is the type of objects and o is the type of propositions

Natural deduction: the rules of logic

let Γ denote a set of propositions (the assumptions)

$$(\text{hyp}) \quad \frac{A \in \Gamma}{\Gamma \vdash A}$$

$$(\bot E) \quad \frac{\Gamma \vdash \bot}{\Gamma \vdash A}$$

$$(\neg I) \quad \frac{\Gamma \cup \{A\} \vdash \bot}{\Gamma \vdash \neg A} \qquad\qquad (\neg E) \quad \frac{\Gamma \vdash A \quad \Gamma \vdash \neg A}{\Gamma \vdash \bot}$$

I stands for introduction, E for elimination

Natural deduction: the rules of logic

$$(\wedge I) \quad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

$$(\wedge E1) \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \qquad (\wedge E2) \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$$

Natural deduction: the rules of logic

$$(\vee I1) \quad \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \qquad (\vee I2) \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$$

$$(\vee E) \quad \frac{\Gamma \vdash A \vee B \quad \Gamma \cup \{A\} \vdash C \quad \Gamma \cup \{B\} \vdash C}{\Gamma \vdash C}$$

Natural deduction: the rules of logic

$$(\forall I) \quad \frac{\Gamma \vdash A \quad x \notin FV(\Gamma)}{\Gamma \vdash \forall x A}$$

$$(\forall E) \quad \frac{\Gamma \vdash \forall x A}{\Gamma \vdash A_x^t}$$

$$(\exists I) \quad \frac{\Gamma \vdash A_x^t}{\Gamma \vdash \exists x A}$$

$$(\exists E) \quad \frac{\Gamma \vdash \exists x A \quad \Gamma \cup \{A\} \vdash B \quad x \notin FV(\Gamma, B)}{\Gamma \vdash B}$$

Natural deduction: the rules of logic

$$(\Rightarrow I) \quad \frac{\Gamma \cup \{A\} \vdash B}{\Gamma \vdash A \Rightarrow B}$$

$$(\Rightarrow E) \quad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

for classical logic, add also the rule:

$$(\text{excluded-middle}) \quad \frac{}{\Gamma \vdash A \vee \neg A}$$

Provability

a proposition A is true if there is a derivation of $\emptyset \vdash A$

example: $A \Rightarrow (B \Rightarrow A)$ is provable:

$$\cfrac{\cfrac{\cfrac{A \in \{A, B\}}{\{A, B\} \vdash A}\,(hyp)}{\{A\} \vdash B \Rightarrow A}\,(\Rightarrow I)}{\emptyset \vdash A \Rightarrow (B \Rightarrow A)}\,(\Rightarrow I)$$

this presentation of logic (natural deduction) is independently due to Stanislaw Jaskowski and Gerhard Gentzen in 1934

1+1=2 ?

representing objects by first-order terms
with 0 of arity 0, s of arity 1 and $+$ of arity 2

taking the predicate symbol $=$ of arity $[\iota;\iota;o]$

let Γ be the following set of axioms:

- $=$ is an equivalence relation:
  - $\forall x.x = x$
  - $\forall x.\forall y.x = y \Rightarrow y = x$
  - $\forall x.\forall y.\forall z.x = y \wedge y = z \Rightarrow x = z$
- definition of $+$:
  - $\forall x.x + 0 = x$
  - $\forall x.\forall y.x + (sy) = s(x + y)$

$1+1=2$!

proof of $\Gamma \vdash s0 + s0 = s(s0)$ on board yet, we have seen that $1 + 1 \to_\beta^* 2$ when representing natural numbers as simply-typed λ-terms of type $(\iota \to \iota) \to (\iota \to \iota)$

representing objects by simply-typed λ-terms provide more computational power, hence simpler proofs if we add the following deduction rule:

$$(\mathit{conv}) \quad \frac{\Gamma \vdash A \quad A =_\beta B}{\Gamma \vdash B}$$

deduction steps are made modulo β-equivalence with deduction modulo, the proof of $\Gamma \vdash 1+1=2$ becomes:

$$\dfrac{\dfrac{\dfrac{\forall x.x=x \in \Gamma}{\Gamma \vdash \forall x.x=x}\,(hyp)}{\Gamma \vdash s(s0)=s(s0)}\,(\forall E)}{\Gamma \vdash s0+s0=s(s0)}\,(conv)$$

Deduction modulo

more generally, natural deduction can be extended into natural deduction modulo a decidable congruence on propositions $\equiv$:

$$(conv) \quad \frac{\Gamma \vdash A \quad A \equiv B}{\Gamma \vdash B}$$

[Dowek-Hardin-Kirchner 1998]

Simple type theory and higher-order logic

objects are the simply-typed λ-terms whose type is in $\mathbb{T}^{\rightarrow}(\{\iota\})$

i-th-order logic allows quantifications $\kappa_\sigma : [[\sigma]o]o$ ($\kappa \in \{\exists, \forall\}$) over any type $\sigma \in \mathbb{T}^{\rightarrow}(\{\iota, o\})$ such that $order(\sigma) < i$
higher-order logic allows quantifications over any type

- $order(\iota) = 0$
- $order(o) = 1$
- $order(T_1 \rightarrow \ldots \rightarrow T_n \rightarrow B) = 1 + max\{order(T_i) \mid 1 \leq i \leq n\}$

examples: ι is of order 0 (first-order logic), $\iota \rightarrow \iota \rightarrow \iota$ and $\iota \rightarrow \iota \rightarrow o$ are of order 1 (second-order logic), $(\iota \rightarrow \iota) \rightarrow \iota$ and $(\iota \rightarrow o) \rightarrow o$ are of order 2 (third-order logic), etc.

Quantification over predicates (impredicativity)

in simple type theory/higher-order logic, we can quantify over predicates like in:

$$\forall_o P.P \Rightarrow P$$

such a formula is said <u>impredicative</u> since P can be replaced by the formula itself, yielding $(\forall_o P.P \Rightarrow P) \Rightarrow (\forall_o P.P \Rightarrow P)$

Quantification over predicates (impredicativity)

quantification over predicates allows to define other predicates:

$A \wedge B$	$\forall C.(A \Rightarrow B \Rightarrow C) \Rightarrow C$
$A \vee B$	$\forall C.(A \Rightarrow C) \Rightarrow (B \Rightarrow C) \Rightarrow C$
$\exists x.P$	$\forall C.(\forall x.Px \Rightarrow C) \Rightarrow C$
$\perp$	$\forall C.C$
$x = y$	$\forall P.Px \Rightarrow Py$
$x \in \mathbb{N}$	$\forall P.P0 \Rightarrow (\forall y.Py \Rightarrow P(y+1)) \Rightarrow Px$

Cut elimination

$$\dfrac{\pi_1 = \dfrac{\dots}{\Gamma, A \vdash B} \quad \dfrac{}{\Gamma \vdash A \Rightarrow B}(\Rightarrow I) \qquad \pi_2 = \dfrac{\dots}{\Gamma \vdash A}}{\Gamma \vdash B}(\Rightarrow E)$$

can be simplified into π_1' where π_1' is obtained from π_1 by:

- removing A from the hypotheses of π_1
- replacing every subproof $\dfrac{A \in \Gamma \cup \Delta}{\Gamma \cup \Delta \vdash A}$ of π_1 by π_2^Δ
  where π_2^Δ is π_2 with Γ replaced by $\Gamma \cup \Delta$ (weakening)

Cut elimination

other simplifications exist for the other combinations of an introduction rule followed by an elimination rule

in intuitionistic logic (*i.e.* without the excluded-middle rule), cut elimination terminates [Gentzen 1934]

Properties of cut-free proofs

in intuitionistic logic, a cut-free assumption-free proof necessarily ends with an introduction rule

- if $A \vee B$ has a cut-free assumption-free proof then either A or B has a (cut-free) proof
- if $\exists x A$ has a cut-free assumption-free proof then it contains t such that A_x^t has a (cut-free) proof

Program extraction

program specification: $S = \forall x \exists y P(x, y)$

given an assumption-free proof of S, cut-elimination provides a way, given x, to compute y such that $P(x, y)$ holds

an intuitionnistic proof of a program specification provides <u>bug-free</u> program !

Outline

Curry-Howard isomorphism

a proof can be represented by a λ-term/program

first described by Curry in 1958 and extended by Howard in 1969

logic	λ-calculus/programming
formula	program type
connector/quantifier	type constructor
proof	term/program
logical rule	term constructor
assumption	variable
cut elimination	program evaluation

Example: $\Rightarrow$

logic	λ-calculus
$\Rightarrow$	arrow type $\rightarrow$
$\Rightarrow I$	abstraction $\lambda : [[A]B](A \Rightarrow B)$
$\Rightarrow E$	application $@ : [A \Rightarrow B; A]B$
$\Rightarrow$-cut	$(\lambda xt)u \rightarrow t_x^u$

Example: $\Rightarrow$

$$(\mathrm{hyp})\quad \frac{x : A \in \Gamma}{\Gamma \vdash x : A}$$

$$(\Rightarrow I)\quad \frac{\Gamma \cup \{x : A\} \vdash t : B}{\Gamma \vdash \lambda x t : A \Rightarrow B}$$

$$(\Rightarrow E)\quad \frac{\Gamma \vdash t : A \Rightarrow B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

Cut elimination

$$\dfrac{\pi_1 = \quad \dfrac{\ldots}{\Gamma, x : A \vdash t : B}}{\Gamma \vdash \lambda xt : A \Rightarrow B}\,(\Rightarrow I) \qquad \pi_2 = \quad \dfrac{\ldots}{\Gamma \vdash u : A}$$

$$\dfrac{\Gamma \vdash \lambda xt : A \Rightarrow B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda xt)u : B}\,(\Rightarrow E)$$

π_1' corresponds to t_x^u

Other logical connector: $\wedge$

logic	λ-calculus
$\wedge$	cartesian product $\times$
$\wedge I$	pairing $\langle \_, \_ \rangle : [A; B](A \times B)$
$\wedge E1$	1st projection $\pi_1 : [A \times B]A$
$\wedge E2$	2nd projection $\pi_2 : [A \times B]B$
$\wedge$-cut	$\pi_i \langle x_1, x_2 \rangle \rightarrow x_i$

$$\frac{\Gamma \vdash t_1 : T_1 \quad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \langle t_1, t_2 \rangle : T_1 \times T_2} \qquad \frac{\Gamma \vdash p : T_1 \times T_2}{\pi_i p : T_i}$$

Other logical connector: $\vee$

logic	λ-calculus
$\vee$	disjoint sum $+$
$\vee I1$	1st injection $\iota_1 : [A](A+B)$
$\vee I2$	2nd injection $\iota_2 : [B](A+B)$
$\vee E$	pattern-matching match : $[A+B; [A]C; [B]C]C$
$\vee$-cut	match $\iota_i t$ with $\{\iota_1 x \mapsto u_1, \iota_2 x \mapsto u_2\} \;\rightarrow\; {u_i}_x^t$

$$\frac{\Gamma \vdash t : T_i}{\Gamma \vdash \iota_i t : T_1 + T_2} \qquad \frac{\Gamma \vdash t : T_1 + T_2 \quad \Gamma, x : T_1 \vdash u_1 : U \quad \Gamma, x : T_2 \vdash u_2 : U}{\Gamma \vdash \text{match } t \text{ with } \{\iota_1 x \mapsto u_i, \iota_2 x \mapsto u_2\} : U}$$

Propositional logic

up to now, we have:

$$\begin{array}{rrcl} \text{types/propositions} & T & = & X \mid T \to T \mid T \times T \mid T + T \\ \text{terms/proofs} & t & = & x \mid \lambda x : T.t \mid tt \mid \langle t, t\rangle \mid \pi_i t \mid \iota_i^T t \\ & & & \mid \text{match } t \text{ with } \{\iota_1^T x \mapsto t, \iota_2^T x \mapsto t\} \\ \text{contexts/assumptions} & \Gamma & = & \emptyset \mid \Gamma, x : T \end{array}$$

type annotations are necessary for type unicity:

$$\frac{\Gamma, x : T \vdash u : U}{\Gamma \vdash \lambda x : T.u : T \to U} \quad \frac{\Gamma \vdash t : T_1}{\Gamma \vdash \iota_1^{T_2} t : T_1 + T_2} \quad \frac{\Gamma \vdash t : T_2}{\Gamma \vdash \iota_2^{T_1} t : T_1 + T_2}$$

$$\frac{\Gamma \vdash t : T_1 + T_2 \quad \Gamma, x : T_1 \vdash u_1 : U \quad \Gamma, x : T_2 \vdash u_2 : U}{\Gamma \vdash \text{match } t \text{ with } \{\iota_1^{T_2} x \mapsto u_i, \iota_2^{T_1} x \mapsto u_2\} : U}$$

Outline

System F [Girard 1971]

Curry-Howard isomorphism for quantification on propositions:

$$\begin{array}{rrcl} \text{types} & T & = & X \mid T \to T \mid \forall XT \\ \text{terms} & t & = & x \mid \lambda x:T.t \mid tt \mid \Lambda XT \mid tT \\ \text{contexts} & \Gamma & = & \emptyset \mid \Gamma, x:T \end{array}$$

$$\frac{\Gamma \vdash t : T \quad X \notin \Gamma}{\Gamma \vdash \Lambda Xt : \forall XT} \qquad \frac{\Gamma \vdash v : \forall XT}{\Gamma \vdash vU : T_X^U}$$

Computational power of system F

with natural numbers of type $N = \forall X (X \rightarrow X) \rightarrow (X \rightarrow X)$:
$0 = \Lambda X \lambda xy.y$ (iterate 0 times x on y)
$1 = \Lambda X \lambda xy.xy$ (iterate 1 times x on y)
$2 = \Lambda X \lambda xy.x(xy)$ (iterate 2 times x on y)
...
system F can express any function whose existence is provable in second-order arithmetic

examples: $s = \lambda p \Lambda X \lambda xy.x(pXxy)$, $+ = \lambda pq \Lambda X \lambda xy.pNsq$,
$\times = \lambda pq \Lambda X \lambda xy.pN(+q)0$, $power = \lambda pq \Lambda X \lambda xy.qN(\times p)1$, ...

Data types in system F

$$
\begin{aligned}
&A \times B = \forall X.(A \to B \to X) \to X\\
&\langle x, y\rangle = \Lambda X \lambda f.fxy\\
&\pi_1 x = xA(\lambda xy.x)\\
&\pi_2 x = xB(\lambda xy.y)
\end{aligned}
$$

$$
\begin{aligned}
&A + B = \forall X.(A \to X) \to (B \to X) \to X\\
&\iota_1 x = \Lambda X \lambda u_1 u_2.u_1 x\\
&\iota_2 x = \Lambda X \lambda u_1 u_2.u_2 x\\
&\mathsf{case}_C\ t\ \mathsf{with}\ \{\iota_1 x \mapsto u_1 \mid \iota_2 x \mapsto u_2\} = tC(\lambda x u_1)(\lambda x u_2)
\end{aligned}
$$

Inductive data types

many other data types can be built using $\times$ and $+$:

$$T = X \mid 1 \mid T \times T \mid T + T \mid \mu X.T$$

- natural numbers $\mathbb{N} = \mu X.1 + X$
  $0 = \iota_1$
  $s = \iota_2$
- binary trees $\mathbb{T} = \mu X.1 + X \times X$
  $leaf = \iota_1$
  $node = \iota_2$

Inductive data types in system F

- $[\![X]\!] = X$
- $[\![1]\!] = \forall X.X \to X$
- $[\![A \times B]\!] = \forall X.([\![A]\!] \to [\![B]\!] \to X) \to X$ (X fresh)
- $[\![A + B]\!] = \forall X.([\![A]\!] \to X) \to ([\![B]\!] \to X) \to X$ (X fresh)
- $[\![\mu X.T]\!] = \forall X.[\![T]\!]$

example: $[\![\mu X.1 + X]\!] = \forall X.(1 \to X) \to (X \to X) \to X$

Outline

Proofs as objects

so far we have seen that:

- objects are λ-terms of type $\sigma \in \mathbb{T}^{\rightarrow}(\{\iota\})$, the set of object types: a λ-term t is an object if $\Gamma \vdash t : \sigma$ where Γ maps every free object variable of t to some object type
- proofs are λ-terms of type a Curry-Howard type: a λ-term t is a proof if $\Gamma \vdash t : T$ where T is a Curry-Howard type and Γ maps every free predicate variable of t to a Curry-Howard type

- taking $\iota : o$, object types can be seen as predicates and objects as proofs (*e.g.* $\mathbb{N}$ is a predicate and 0, $s0$, ... are proofs of $\mathbb{N}$)
- to extend the Curry-Howard isomorphism to quantifications on objects, proofs can be seen as objects:
  - the type corresponding to $\forall x : T.U$ is often written $\Pi x : T.U$
    $T \rightarrow U$ is the particular case of $\Pi x : T.U$ when $x \notin FV(U)$

$$\frac{\Gamma, x : T \vdash u : U}{\Gamma \vdash \lambda x : T.u : \Pi x : T.U} \quad \frac{\Gamma \vdash v : \Pi x : T.U \quad \Gamma \vdash t : T}{\Gamma \vdash vt : U_x^t}$$

  - the type corresponding to $\exists x : T.U$ is often written $\Sigma x : T.U$
    $T \times U$ is the particular case of $\Sigma x : T.U$ when $x \notin FV(U)$

$$\frac{\Gamma \vdash t : T \quad \Gamma \vdash u : U_x^t}{\Gamma \vdash \langle t, u \rangle : \Sigma x : T.U} \quad \frac{\Gamma \vdash v : \Sigma x : T.U}{\Gamma \vdash \pi_1 v : T} \quad \frac{\Gamma \vdash v : \Sigma x : T.U}{\Gamma \vdash \pi_2 v : T_x^{\pi_1 v}}$$

Pure Type Systems

all previous systems are instances of the following general framework [Barendregt 1992]:

- let $\mathcal{S}$ be a set of sorts (*e.g.* o)
- the algebra of types and terms is:

$$\boxed{t = s \in \mathcal{S} \mid x \in \mathcal{X} \mid \lambda x : t.t \mid tt \mid \Pi x : t.t}$$

- valid contexts:

$$\vdash \emptyset$$

$$\frac{\vdash \Gamma \quad \Gamma \vdash T : s \in \mathcal{S}}{\vdash \Gamma, x : T}$$

Pure Type Systems

- let $\mathcal{A} \subseteq \mathcal{S}^2$ be a set of typing axioms for sorts:

$$\frac{\vdash \Gamma \quad (s, s') \in \mathcal{A}}{\Gamma \vdash s : s'}$$

- let $\mathcal{R} \subseteq \mathcal{S}^2$ be a set of product formation rules:

$$\frac{\Gamma \vdash T : s \quad \Gamma, x : T \vdash U : s' \quad (s, s') \in \mathcal{R}}{\Gamma \vdash \Pi x : T.U : s'}$$

- conversion rule:

$$\frac{\Gamma \vdash t : T \quad T =_\beta T'}{\Gamma \vdash t : T'}$$

Pure Type Systems

- valid terms:

$$\frac{\vdash \Gamma \quad x : T \in \Gamma}{\Gamma \vdash x : T}$$

$$\frac{\Gamma, x : T \vdash u : U \quad \Gamma \vdash \Pi x : T.U : s \in \mathcal{S}}{\Gamma \vdash \lambda x : T.u : \Pi x : T.U}$$

$$\frac{\Gamma \vdash v : \Pi x : T.U \quad \Gamma \vdash t : T}{\Gamma \vdash vt : U_x^t}$$

Barendregt's λ-cube

for instance, take $\mathcal{S} = \{o, \Box\}$ and $\mathcal{A} = \{(o, \Box)\}$:

$\mathcal{R}$	allowed constructions	example of valid context
(o, o)	simple types	$\vdash \iota : o, f : \iota \rightarrow \iota$
$(o, \Box)$	dependent types	$\vdash \iota : o, P : \iota \rightarrow o$
$(\Box, o)$	polymorphic types	$\vdash \iota : o, f : o \rightarrow \iota$
$(\Box, \Box)$	type constructors	$\vdash \iota : o, P : o \rightarrow o$

$\mathcal{R} = \mathcal{S}^2$ is the Calculus of Constructions [Coquand-Huet 1988]

this is the basis of the Coq system

What about proofs by induction ?

induction principle on the set $\mathbb{N}$ of natural numbers:

$$rec : \forall P : \mathbb{N} \Rightarrow o.P0 \Rightarrow (\forall n : \mathbb{N}.Pn \Rightarrow P(sn)) \Rightarrow \forall n : \mathbb{N}.Pn$$

cut elimination rules:

$$\begin{array}{rcl} recPuv0 & \rightarrow_\iota & u \\ recPuv(sn) & \rightarrow_\iota & vn(recPuvn) \end{array}$$

non-dependent case:

$$rec' : \forall X : o.X \Rightarrow (\mathbb{N} \Rightarrow X \Rightarrow X) \Rightarrow \mathbb{N} \Rightarrow X$$

Defining functions by induction

definition of addition by induction on its 2nd argument:

$$+ = \lambda p : \mathbb{N}.\lambda q : \mathbb{N}.rec'\mathbb{N}p(\lambda n : \mathbb{N}.\lambda r : \mathbb{N}.sr)q$$

cut elimination rules:

$$\begin{array}{rcl} +p0 & \rightarrow^*_{\beta\iota} & p \\ +p(sn) & \rightarrow^*_{\beta\iota} & s(+pn) \end{array}$$

Defining functions by induction

a more readable presentation using a fixpoint:

$$
\begin{array}{rl}
+ = & \lambda p:\mathbb{N}.\lambda q:\mathbb{N}. \\
 & \text{match } q \text{ with} \\
 & \{0 \mapsto p, \\
 & sn \mapsto s(+pn)\}
\end{array}
$$

Polymorphic and dependent inductive types

polymorphic lists of fixed length (polymorphic arrays):

$$
\begin{array}{rcl}
list & : & o \Rightarrow \mathbb{N} \Rightarrow o \\
nil & : & \forall A : o.listA0 \\
cons & : & \forall A : o.\forall n : \mathbb{N}.A \Rightarrow listAn \Rightarrow listA(sn) \\
app & : & \forall A : o.\forall n : \mathbb{N}.listAn \Rightarrow \forall p : \mathbb{N}.listAp \Rightarrow listA(n+p)
\end{array}
$$

$$
\begin{array}{rl}
app = & \lambda A : o.\lambda n : \mathbb{N}.\lambda N : listAn.\lambda p : \mathbb{N}.\lambda P : listAp. \\
& \text{match } N \textit{ with} \\
& \{nilA \mapsto P, \\
& consAxqQ \mapsto consAx(q+p)(appAqQpP)\}
\end{array}
$$

Inductive predicates

ordering on natural numbers:

$$\begin{array}{rcl} \leq & : & \mathbb{N} \Rightarrow \mathbb{N} \Rightarrow o \\ \leq_0 & : & \forall x : \mathbb{N}. 0 \leq x \\ \leq_s & : & \forall x : \mathbb{N}. \forall y : \mathbb{N}. x \leq y \Rightarrow sx \leq sy \end{array}$$

sorted lists:

$$\begin{array}{rcl} sorted & : & \forall A : o. \forall n : \mathbb{N}. listAn \Rightarrow o \\ sorted_0 & : & \forall A : o. sortedA0(nilA) \\ sorted_1 & : & \forall A : o. \forall x : A. sortedA1(consAx0(nilA)) \\ sorted_2 & : & \forall A : o. \forall x : A. \forall y : A. \forall n : \mathbb{N}. \forall N : listAn. \\ & & sortedA(n+1)(consAynN) \Rightarrow x \leq y \\ & & \Rightarrow sortedA(n+2)(consAx(n+1)(consAynN)) \end{array}$$

On board

- transitive closure of a relation and Tarski's fixpoint theorem
- correctness and completeness of a type-checking algorithm for the simply-typed λ-calculus
- correctness and completeness of a type-inference algorithm for pure untyped λ-terms in the simply-typed λ-calculus
- strong normalization proof of $\rightarrow_\beta$ in the simply-typed λ-calculus based on Tait and Girard's notion of computability

Bibliography

- *History of Lambda-Calculus and Combinatory logic*, F. Cardone and J. R. Hindley, to appear in Vol. 5 of the Handbook of the History of Logic, Elsevier, `www-maths.swan.ac.uk/staff/jrh/papers/JRHHislamWeb.pdf`
- *The Lambda Calculus: Its Syntax and Semantics* (2nd ed.), H. Barendregt, North-Holland, 1984
- *Lambda Calculi with types*, H. Barendregt, in the Handbook of Logic in Computer Science, Oxford University Press, 1992
- *Rewrite Systems*, N. Dershowitz and J.-P. Jouannaud, in the Handbook of Theoretical Computer Science, North Holland, 1990
- *Term Rewriting Systems*, Cambridge Tracts in Theoretical Computer Science, Vol. 55, Cambridge University Press, 2003