Functional Programming in Coq

Yuxin Deng

East China Normal University

`https://faculty.ecnu.edu.cn/_s43/dyx/main.psp`

September 1, 2022

Reading materials

1. Lecture Notes on Functional Programming (in Chinese). `https://faculty.ecnu.edu.cn/_upload/article/files/0b/e3/ed18bad24dd9b9d44a7251fc3d1c/faa8ee9a-0273-465c-80bb-10e05427f8a4.pdf`
2. The Coq proof assistant. `http://coq.inria.fr`
3. Benjamin C. Pierce et al. Software Foundations. `https://softwarefoundations.cis.upenn.edu`
4. Yves Bertot, Pierre Casteran. Coq'Art: The Calculus of Inductive Constructions. Springer-Verlag, 2004.

FP Designers

Alonzo Church:
lambda calculus
1930's

Guy Steele & Gerry Sussman:
Scheme
late 1970's

Xavier Leroy:
Ocaml
1990's

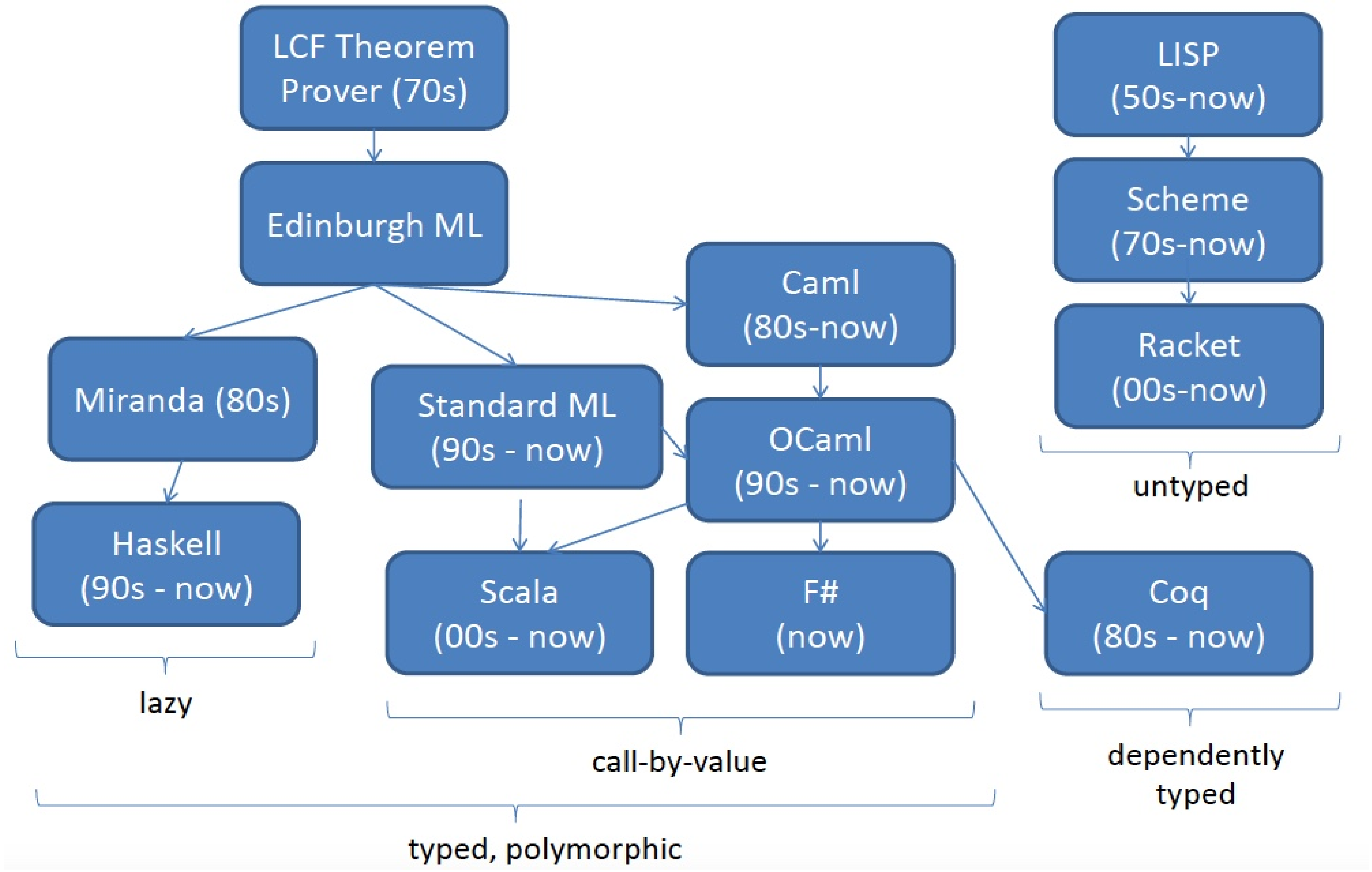
John McCarthy:
LISP
1958

Robin Milner, Mads Tofte, & Robert Harper
Standard ML
1980's

Don Syme:
F#
2000's

from D. Walker's notes

FP Geneology



Coq Designers

- Started from an implementation of the Calculus of Constructions by Thierry Coquand and Gerard Huet in 1984.

- Extended to the Calculus of Inductive Constructions by Christine Paulin in 1991.

- Contributed by 50 people in 30 years.

- Received the 2013 ACM Software System Award

The lambda calculus

Computability

A question in the 1930's: what does it mean for a function $f : \mathbb{N} \to \mathbb{N}$ to be computable?

Informally, there should be a pencil-and-paper method allowing a trained person to calculate $f(n)$, for any given n.

- Turing defined a Turing machines and postulated that a function is computable if and only if it can be computed by such a machine.

- Gödel defined the class of general recursive functions and postulated that a function is computable if and only if it is general recursive.

- Church defined the lambda calculus and postulated that a function is computable if and only if it can be written as a lambda term.

Church, Kleene, Rosser, and Turing proved that all three computational models were equivalent to each other.

The untyped lambda calculus

Def. Assume an infinite set $\mathcal{V}$ of variables, denoted by $x, y, z....$ The set of lambda terms are defined by the Backus-Naur Form:

$$M, N ::= x \mid (MN) \mid (\lambda x.M)$$

Alternatively, the set of lambda terms is the smallest set Λ satisfying:

- whenever $x \in \mathcal{V}$ then $x \in \Lambda$ (variables)
- whenever $M, N \in \Lambda$ then $(MN) \in \Lambda$ (applications)
- whenever $x \in \mathcal{V}$ and $M \in \Lambda$ then $(\lambda x.M) \in \Lambda$ (lambda abstractions)

E.g. $(\lambda x.x)$ $((\lambda x.(xx))(\lambda y.(yy)))$ $(\lambda f.(\lambda x.(f(fx))))$

Convention

- Omit outermost parentheses. E.g., write MN instead of (MN).
- Applications associate to the left, i.e. MNP means $(MN)P$.
- The body of a lambda abstraction (the part after the dot) extends as far to the right as possible. E.g, $\lambda x.MN$ means $\lambda x.(MN)$, and not $(\lambda x.M)N$.
- Multiple lambda abstractions can be contracted; E.g., write $\lambda xyz.M$ for $\lambda x.\lambda y.\lambda z.M$.

Free and bound variables

An occurrence of a variable x inside $\lambda x.N$ is said to be bound. The corresponding λx is called a binder, and the subterm N is the scope of the binder. A variable occurrence that is not bound is free.

E.g. in $M \equiv (\lambda x.xy)(\lambda y.yz)$, x is bound, z is free, variable y has both a free and a bound occurrence.

The set of free variables of term M is $FV(M)$:

$$\begin{array}{rcl} FV(x) & = & \{x\} \\ FV(MN) & = & FV(M) \cup FV(N) \\ FV(\lambda x.M) & = & FV(M) \backslash \{x\} \end{array}$$

Renaming

Write $M\{y/x\}$ for the renaming of x as y in M.

$$\begin{array}{rcll} x\{y/x\} & \equiv & y & \\ z\{y/x\} & \equiv & z, & \text{if } x \neq z \\ (MN)\{y/x\} & \equiv & (M\{y/x\})(N\{y/x\}) & \\ (\lambda x.M)\{y/x\} & \equiv & \lambda y.(M\{y/x\}) & \\ (\lambda z.M)\{y/x\} & \equiv & \lambda z.(M\{y/x\}), & \text{if } x \neq z \end{array}$$

α-equivalence

$$\frac{}{M = M} \qquad \frac{M = M' \quad N = N'}{MN = M'N'}$$

$$\frac{M = N}{N = M} \qquad \frac{M = M'}{\lambda x.M = \lambda x.M'}$$

$$\frac{M = N \quad N = P}{M = P} \qquad \frac{y \notin V(M)}{\lambda x.M = \lambda y.M\{y/x\}}$$

$V(M)$ represents the set of variables in V.

Substitution

The capture-avoiding substitution of N for free occurrences of x in M, in symbols $M[N/x]$ is defined below:

$$
\begin{array}{lll}
x[N/x] & \equiv N & \\
y[N/x] & \equiv y, & \text{if } x \neq y \\
(MP)[N/x] & \equiv (M[N/x])(P[N/x]) & \\
(\lambda x.M)[N/x] & \equiv \lambda x.M & \\
(\lambda y.M)[N/x] & \equiv \lambda y.(M[N/x]), & \text{if } x \neq y \text{ and } y \notin FV(N) \\
(\lambda y.M)[N/x] & \equiv \lambda y'.(M\{y'/y\}[N/x]), & \text{if } x \neq y,\ y \in FV(N), \text{ and } y' \text{ fresh.}
\end{array}
$$

β-reduction

Convention: we identify lambda terms up to α-equivalence.

A term of the form $(\lambda x.M)N$ is β-redex. It reduces to $M[N/x]$ (the reduct).

A lambda term without β-redex is in β-normal form.

$$\begin{array}{rcl} (\lambda x.y)(\underline{(\lambda z.zz)(\lambda w.w)}) & \longrightarrow_\beta & (\lambda x.y)(\underline{(\lambda w.w)(\lambda w.w)}) \\ & \longrightarrow_\beta & \underline{(\lambda x.y)(\lambda w.w)} \\ & \longrightarrow_\beta & y \end{array}$$

$$\underline{(\lambda x.y)((\lambda z.zz)(\lambda w.w))} \longrightarrow_\beta y$$

Observation

- reducing a redex can create new redexes,
- reducing a redex can delete some other redexes,
- the number of steps that it takes to reach a normal form can vary, depending on the order in which the redexes are reduced.

Evaluation

Write $\twoheadrightarrow_\beta$ for $\longrightarrow_\beta^*$, the reflexive transitive closure of $\longrightarrow_\beta$. If $M \twoheadrightarrow_\beta M'$ and M' is in normal form, then we say M evaluates to M'.

Not every term has a normal form.

$$\begin{array}{rcl} (\lambda x.xx)(\lambda y.yyy) & \longrightarrow_\beta & (\lambda y.yyy)(\lambda y.yyy) \\ & \longrightarrow_\beta & (\lambda y.yyy)(\lambda y.yyy)(\lambda y.yyy) \\ & \longrightarrow_\beta & \ldots \end{array}$$

Formal definition of β-reduction

The single-step β-reduction is the smallest relation $\longrightarrow_\beta$ satisfying:

$$\frac{}{(\lambda x.M)N \longrightarrow_\beta M[N/x]}$$

$$\frac{M \longrightarrow_\beta M'}{MN \longrightarrow_\beta M'N}$$

$$\frac{N \longrightarrow_\beta N'}{MN \longrightarrow_\beta MN'}$$

$$\frac{M \longrightarrow_\beta M'}{\lambda x.M \longrightarrow_\beta \lambda x.M'}$$

Write $M =_\beta M'$ if M can be transformed into M' by zero or more reductions steps and/or inverse reduction steps. Formally, $=_\beta$ is the reflexive symmetric transitive closure of $\longrightarrow_\beta$.

Programming in the untyped lambda calculus

Booleans: let $\mathbf{T} = \lambda xy.x$ and $\mathbf{F} = \lambda xy.y$.

Let $\mathbf{and} = \lambda ab.ab\mathbf{F}$. Then

$$\begin{aligned}
\mathbf{and\ TT} &\twoheadrightarrow_\beta \mathbf{T} \\
\mathbf{and\ TF} &\twoheadrightarrow_\beta \mathbf{F} \\
\mathbf{and\ FT} &\twoheadrightarrow_\beta \mathbf{F} \\
\mathbf{and\ FF} &\twoheadrightarrow_\beta \mathbf{F}
\end{aligned}$$

The above encoding is not unique. The "and" function can also be encoded as $\lambda ab.bab$.

Other boolean functions

$$\begin{aligned}
\textbf{not} &= \lambda a.a\mathbf{F}\mathbf{T} \\
\textbf{or} &= \lambda ab.a\mathbf{T}b \\
\textbf{xor} &= \lambda ab.a(b\mathbf{F}\mathbf{T})b \\
\textbf{if-then-else} &= \lambda x.x
\end{aligned}$$

$$\textbf{if-then-else}\ \mathbf{T}MN \twoheadrightarrow_\beta M$$
$$\textbf{if-then-else}\ \mathbf{F}MN \twoheadrightarrow_\beta N$$

Natural numbers

Write $f^n x$ for the term $f(f(\ldots(fx)\ldots))$, where f occurs n times. The nth Church numeral $\bar{n} = \lambda fx.f^n x$.

$$\begin{array}{rcl} \bar{0} & = & \lambda fx.x \\ \bar{1} & = & \lambda fx.fx \\ \bar{2} & = & \lambda fx.f(fx) \\ & \ldots & \end{array}$$

The successor function

Let **succ** $= \lambda nfx.f(nfx)$.

$$\begin{array}{rcl} \textbf{succ}\ \bar{n} & = & (\lambda nfx.f(nfx))(\lambda fx.f^n x) \\ & \longrightarrow_\beta & \lambda fx.f((\lambda fx.f^n x)fx) \\ & \twoheadrightarrow_\beta & \lambda fx.f(f^n x) \\ & = & \lambda fx.f^{n+1}x \\ & = & \overline{n+1} \end{array}$$

Addition and mulplication

Let $\mathbf{add} = \lambda nmfx.nf(mfx)$ and $\mathbf{mult} = \lambda nmf.n(mf)$

Exercises: show that

$$\begin{aligned} \mathbf{add}\ \bar{n}\bar{m} &\twoheadrightarrow_\beta \overline{n+m} \\ \mathbf{mult}\ \bar{n}\bar{m} &\twoheadrightarrow_\beta \overline{n \cdot m} \end{aligned}$$

Exercise: Let $\mathbf{iszero} = \lambda nxy.n(\lambda z.y)x$ and verify $\mathbf{iszero}(0) = \mathbf{T}$ and $\mathbf{iszero}(n+1) = \mathbf{F}$.

Fixed points and recursive functions

Thm. In the untyped lambda calculus, every term F has a fixed point.

Proof. Let $\Theta = AA$ where $A = \lambda xy.y(xxy)$.

$$\begin{array}{rcl} \Theta F & = & AAF \\ & = & (\lambda xy.y(xxy))AF \\ & \twoheadrightarrow_\beta & F(AAF) \\ & = & F(\Theta F) \end{array}$$

Thus ΘF is a fixed point of F.

The term Θ is called Turing's fixed point combinator.

The factorial function

$$
\begin{aligned}
\textbf{fact}\ n &= \textbf{if-then-else}\ (\textbf{iszero}\ n)(\bar{1})(\textbf{mult}\ n(\textbf{fact}\ (\textbf{pred}\ n)))\\
\textbf{fact} &= \lambda n.\textbf{if-then-else}\ (\textbf{iszero}\ n)(\bar{1})(\textbf{mult}\ n(\textbf{fact}\ (\textbf{pred}\ n)))\\
\textbf{fact} &= (\lambda f.\lambda n.\textbf{if-then-else}\ (\textbf{iszero}\ n)(\bar{1})(\textbf{mult}\ n(f(\textbf{pred}\ n)))\textbf{fact}\\
\textbf{fact} &= \Theta(\lambda f.\lambda n.\textbf{if-then-else}\ (\textbf{iszero}\ n)(\bar{1})(\textbf{mult}\ n(f(\textbf{pred}\ n)))
\end{aligned}
$$

Other data types: pairs

Define $\langle M, N\rangle = \lambda z.zMN$. Let $\pi_1 = \lambda p.p(\lambda xy.x)$ and $\pi_2 = \lambda p.p(\lambda xy.y)$. Observe that

$$\begin{aligned} \pi_1\langle M, N\rangle &\twoheadrightarrow_\beta M \\ \pi_2\langle M, N\rangle &\twoheadrightarrow_\beta N \end{aligned}$$

Tuples

Define $\langle M_1, ..., M_n \rangle = \lambda z.zM_1...M_n$ and the ith projection $\pi_1^n = \lambda p.p(\lambda x_1...x_n.x_i)$. Then

$$\pi_i^n \langle M_1, ..., M_n \rangle \twoheadrightarrow_\beta M_i$$

for all $1 \leq i \leq n$.

Lists

Define $\mathbf{nil} = \lambda xy.y$ and $H :: T = \lambda xy.xHT$. Then the function of adding a list of numbers can be:

$$\mathbf{addlist}\ l = l(\lambda ht.\mathbf{add}\ h(\mathbf{addlist}\ t))(\bar{0})$$

Trees

A binary tree can be either a leaf, labeled by a natural number, or a node with two subtrees. Write $\mathbf{leaf}(n)$ for a leaf labeled n, and $\mathbf{node}(L, R)$ for a node with left subtree L and right subtree R.

$$\begin{aligned}\mathbf{leaf}(n) &= \lambda xy.xn \\ \mathbf{node}(L,R) &= \lambda xy.yLR\end{aligned}$$

A program that adds all the numbers at the leaves of a tree:

$$\mathbf{addtree}\ t = t(\lambda n.n)(\lambda lr.\mathbf{add}\ (\mathbf{addtree}\ l)(\mathbf{addtree}\ r))$$

η-reduction

$$\lambda x.Mx \longrightarrow_\eta M, \text{ where } x \notin FV(M).$$

Define the single-step $\beta\eta$-reduction $\longrightarrow_{\beta\eta} = \longrightarrow_\beta \cup \longrightarrow_\eta$ and the multi-step $\beta\eta$-reduction $\twoheadrightarrow_{\beta\eta}$.

Church-Rosser Theorem

Thm. (Church and Rosser, 1936). Let $\twoheadrightarrow$ denote either $\twoheadrightarrow_\beta$ or $\twoheadrightarrow_{\beta\eta}$. Suppose M, N and P are lambda terms such that $M \twoheadrightarrow N$ and $M \twoheadrightarrow P$. Then there exists a lambda term Z such that $N \twoheadrightarrow Z$ and $P \twoheadrightarrow Z$.

This is the Church-Rosser property or confluence.

See Section 4.4 of the λ-calculus lecture notes for the detailed proof.

Some consequences of confluence

Cor. If $M =_\beta N$ then there exists some Z with $M, N \twoheadrightarrow_\beta Z$. Similarly for $\beta\eta$.

Cor. If N is a β-normal form and $M =_\beta N$, then $M \twoheadrightarrow_\beta N$, and similarly for $\beta\eta$.

Cor. If M and N are β-normal forms such that $M =_\beta N$, then $M =_\alpha N$, and similarly for $\beta\eta$.

Cor. If $M =_\beta N$, then neither or both have a β-normal form, and similarly for $\beta\eta$.

Simply-typed lambda calculus

Simple types: assume a set of basic types, ranged over by ι. The set of simple types is given by

$$A, B ::= \iota \mid A \longrightarrow B \mid A \times B \mid 1$$

- $A \longrightarrow B$ is the type of functions from A to B.
- $A \times B$ is the type of pairs $\langle x, y \rangle$
- 1 is a one-element type, considered as “void” or “unit” type in many languages: the result type of a function with no real result.

Convention: $\times$ binds stronger than $\longrightarrow$ and $\longrightarrow$ associates to the right. E.g. $A \times B \longrightarrow C$ is $(A \times B) \longrightarrow C$, and $A \longrightarrow B \longrightarrow C$ is $A \longrightarrow (B \longrightarrow C)$.

Raw typed lambda terms

$$M, N ::= x \mid MN \mid \lambda x^A.M \mid \langle M, N \rangle \mid \pi_1 M \mid \pi_2 M \mid *$$

Typing judgment

Write $M : A$ to mean "M is of type A". A typing judgment is an expression of the form

$$x_1 : A_1, x_2 : A_2, ..., x_n : A_n \vdash M : A$$

The meaning is: under the assumption that x_i is of type A_i, for $i = 1...n$, the term M is a well-typed term of type A. The free variables of M must be contained in $x_1, ..., x_n$

The sequence of assumptions $x_1 : A_1, x_2 : A_2, ..., x_n : A_n$ is a typing context, written as Γ. The notations Γ, Γ' and $\Gamma, x : A$ denote the concatenation of typing contexts, assuming the sets of variables are disjoint.

Typing rules

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

$$\frac{\Gamma \vdash M : A \longrightarrow B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1 M : A}$$

$$\frac{\Gamma, x : A \vdash M : B}{\lambda x^A . M : A \longrightarrow B} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2 M : B}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \times B} \qquad \frac{}{\Gamma \vdash * : 1}$$

Typing derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : A \to A, y : A \vdash x : A \to A} \qquad \dfrac{\dfrac{}{x : A \to A, y : A \vdash x : A \to A} \quad \dfrac{}{x : A \to A, y : A \vdash y : A}}{x : A \to A, y : A \vdash xy : A}}{x : A \to A, y : A \vdash x(xy) : A}}{x : A \to A \vdash \lambda y^{A}.x(xy) : A \to A}}{\vdash \lambda x^{A \to A}.\lambda y^{A}.x(xy) : (A \to A) \to A \to A}$$

Reductions in the simply-typed lambda calculus

β- and η-reductions:

$$\begin{array}{rcl} (\lambda x^A.M)N & \longrightarrow_\beta & M[N/x] \\ \pi_1\langle M, N\rangle & \longrightarrow_\beta & M \\ \pi_2\langle M, N\rangle & \longrightarrow_\beta & N \\ \\ \lambda x^A.Mx & \longrightarrow_\eta & M \\ \langle \pi_1 M, \pi_2 M\rangle & \longrightarrow_\eta & M \\ M & \longrightarrow_\eta & *, \qquad \text{if } M : 1 \end{array}$$

Subject reduction

Thm. If $\Gamma \vdash M : A$ and $M \longrightarrow_{\beta\eta} M'$, then $\Gamma \vdash M' : A$.

Proof: By induction on the derivation of $M \longrightarrow_{\beta\eta} M'$, and by case distinction on the last rule used in the derivation of $\Gamma \vdash M : A$. $\square$

Church-Rosser

The Church-Rosser theorem does not hold for $\beta\eta$-reduction in the simply-typed $\lambda^{\to,\times,1}$-calculus.

E.g. if x has type $A \times 1$, then

$$\langle \pi_1 x, \pi_2 x \rangle \longrightarrow_\eta x$$

$$\langle \pi_1 x, \pi_2 x \rangle \longrightarrow_\eta \langle \pi_1 x, * \rangle$$

Both x and $\langle \pi_1 x, * \rangle$ are normal forms.

If we omit all the η-reductions and consider only β-reductions, then the Church-Rosser property does hold.

Sum types

Simple types:

$$A, B ::= ... \mid A + B \mid 0$$

Sum type is also known as “union” or “variant” type. The type 0 is the empty type, corresponding to the empty set in set theory.

Raw terms:

$$\begin{array}{lll} M, N, P & ::= & ... \mid in_1 M \mid in_2 M \\ & & \mid case\ M\ of\ x^A \Rightarrow N \mid y^B \Rightarrow P \\ & & \mid\ \Box_A M \end{array}$$

Typing rules for sums

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash in_1 M : A + B}$$

$$\frac{\Gamma \vdash M : B}{\Gamma \vdash in_2 M : A + B}$$

$$\frac{\Gamma \vdash M : A + B \quad \Gamma, x : A \vdash N : C \quad \Gamma, y : B \vdash P : C}{\Gamma \vdash (case\ M\ of\ x^A \Rightarrow N \,|y^B \Rightarrow P) : C}$$

$$\frac{\Gamma \vdash M : 0}{\Gamma \vdash \Box_A M : A}$$

The booleans can be defined as $1 + 1$ with $\mathbf{T} = in_1 *,\ \mathbf{F} = in_2 *$, and **if-then-else** $MNP = case\ M\ of\ x^1 \Rightarrow N \,|\, y^1 \Rightarrow P$, where x and y don't occur in N and P. The term $\Box_A M$ is a simple type cast.

Weak and strong normalization

Def. A term M is weakly normalizing if there exists a finite sequence of reductions $M \to M_1 \to \ldots \to M_n$ such that M_n is a normal form. It is strongly normalizing if there does not exist an infinite sequence of reductions starting from M, i.e., if every sequence of reductions starting from M is finite.

- $\Omega = (\lambda x.xx)(\lambda x.xx)$ is neither weakly nor strongly normalizing.
- $(\lambda x.y)\Omega$ is weakly normalizing, but not strongly normalizing.
- $(\lambda x.y)((\lambda x.x)(\lambda x.x))$ is strongly normalizing.
- Every normal form is strongly normalizing.

Strong normalization

Thm. In the simply-typed lambda calculus, all terms are strongly normalizing.

A proof is given in the following book: J.-Y.Girard, Y.Lafont, and P.Taylor. Proofs and Types. Cambridge University Press, 1989.