Inductive types

I. What we have seen so far

$\lambda\Pi$-calculus and $\lambda\Pi$-calculus modulo

Permit to express all theories:
reduction rules become reduction rules
axioms (if any) become variables

In particular: Arithmetic and Simple type theory

Arithmetic

Arithmetic with Peano's symbol N

Arithmetic without Peano's symbol N (and an induction axiom)

II. Arithmetic without Peano's symbol

Transform most of the axioms into reduction rules

$$y \,\epsilon\, f_{x_1,...,x_n,y,A}(x_1,...,x_n) \longrightarrow A$$

$$x = y \longrightarrow \forall c\ (x \,\epsilon\, c \Rightarrow y \,\epsilon\, c)$$

$$Pred(0) \longrightarrow 0$$

$$Pred(S(x)) \longrightarrow x$$

$$0 + y \longrightarrow y$$

$$S(x) + y \longrightarrow S(x + y)$$

$$0 \times y \longrightarrow 0$$

$$S(x) \times y \longrightarrow x \times y + y$$

$$Null(0) \longrightarrow \top$$

$$Null(S(x)) \longrightarrow \bot$$

Induction

Remains an axiom

$$\forall c \ (0 \ \epsilon \ c \Rightarrow \forall x \ (x \ \epsilon \ c \Rightarrow S(x) \ \epsilon \ c) \Rightarrow \forall y \ y \ \epsilon \ c)$$

or a deduction rule

$$\frac{\Gamma \vdash 0 \ \epsilon \ c \quad \Gamma \vdash \forall x \ (x \ \epsilon \ c \Rightarrow S(x) \ \epsilon \ c)}{\Gamma \vdash t \ \epsilon \ c}$$

Expressing proofs as terms

A symbol for this axiom or this rule

If c class, π proof of $0 \epsilon c$, π' proof of $\forall x\ (x \epsilon c \Rightarrow S(x) \epsilon c)$, and t a term
$Rec(c, \pi, \pi', t)$ proof of the proposition $t \epsilon c$

Last rule property in jeopardy

Class $c = f_{y,(y=0 \vee \exists z \ (y=S(z)))}$
σ proof of $0 = 0$
$\pi = i(\sigma)$ proof of $0 \ \epsilon \ c$
σ' proof of $S(x) = S(x)$
$\pi' = \lambda x \ \lambda \alpha \ (j(\langle x, \sigma' \rangle))$ proof of $\forall x \ (x \ \epsilon \ c \Rightarrow S(x) \ \epsilon \ c)$

$Rec(c, \pi, \pi', S(S(0)))$ proof of $S(S(0)) \ \epsilon \ c$
that is $S(S(0)) = 0 \vee \exists z \ (S(S(0)) = S(z))$
Irreducible proof and does not end with an introduction

Recover: closed cut-free proofs end with an introduction

Extend the notion of cut (mimic proofs by induction in the formulation of arithmetic with Peano's predicate symbol)

Two rules

$$Rec(c, \pi, \pi', 0) \longrightarrow \pi$$

$$Rec(c, \pi, \pi', S(x)) \longrightarrow (\pi'~x~Rec(c, \pi, \pi', x))$$

This way:
$Rec(c, \pi, \pi', S(S(0)))$ reduces to $j(\langle S(0), (S(0)/x)\sigma' \rangle)$

III. Gödel System T

Simply typed λ-calculus + *Rec*

A single base type *nat*, constants $0 : nat$, $S : nat \to nat$, Rec^A of arity $\langle A, nat \to A \to A, nat, A\rangle$

$$((\lambda x : A\ t)\ u) \longrightarrow (u/x)t$$

$$Rec^A(a, g, 0) \longrightarrow a$$

$$Rec^A(a, g, (S\ n)) \longrightarrow (g\ n\ Rec^A(a, g, n))$$

Examples

Multiplication by two

$$d = \lambda a : nat\ Rec^{nat}(0, \lambda x : nat\ \lambda y : nat\ (S\ (S\ y)), a)$$

Addition, multiplication, power

$$+ = \lambda a : nat\ \lambda b : nat\ Rec^{nat}(a, \lambda x : nat\ \lambda y : nat\ (S\ y), b)$$

$$\times = \lambda a : nat\ \lambda b : nat\ Rec^{nat}(0, \lambda x : nat\ \lambda y : nat\ (+\ y\ a), b)$$

$$\uparrow = \lambda a : nat\ \lambda b : nat\ Rec^{nat}((S\ 0), \lambda x : nat\ \lambda y : nat\ (\times\ y\ a), b)$$

Predecessor

$$pred = \lambda a : nat\ Rec^{nat}(0, \lambda x : nat\ \lambda y : nat\ x, a)$$

Characteristic functions

$$\chi_{\{0\}} = \lambda a : nat\ Rec^{nat}((S\ 0), \lambda x : nat\ \lambda y : nat\ 0, a)$$

$$\chi_{\mathbb{N}\setminus\{0\}} = \lambda a : nat\ Rec^{nat}(0, \lambda x : nat\ \lambda y : nat\ (S\ 0), a)$$

$$\chi_{2\mathbb{N}} = \lambda a : nat\ Rec^{nat}((S\ 0), \lambda x : nat\ \lambda y : nat\ (\chi_{\{0\}}\ y), a)$$

Primitive recursive functions

Definition by induction

$$f(x_1, ..., x_{p-1}, 0) = a(x_1, ..., x_{p-1})$$

$$f(x_1, ..., x_{p-1}, S(n)) = g(x_1, ..., x_{p-1}, n, f(x_1, ..., x_{p-1}, n))$$

In the System T

$f = \lambda x_1 ... \lambda x_{p-1} \lambda x_p$

$$Rec^{nat}((a\ x_1\ ...\ x_{p-1}), \lambda n \lambda m\ (g\ x_1\ ...\ x_{p-1}\ n\ m), x_p)$$

All primitive recursive functions

Non primitive recursive functions

Ackermann's function A defined by

$$A(0, x) = 2^x$$

$$A(S(n), 0) = 1$$

$$A(S(n), S(x)) = A(n, A(S(n), x))$$

$$\lambda n \; Rec^{nat \to nat}(P, \lambda p \lambda f \lambda m \; (Rec^{nat} \; (S \; 0) \; (\lambda q \lambda s \; (f \; s)) \; m), n)$$

where $P = x \mapsto 2^x$

IV. The termination of Gödel System T

Simulate the recursor with Parigot numbers

The theory $\mathcal{T}$:
A language with a unary predicate symbol ε, a constant *nat* and a binary function symbol $\dot{\Rightarrow}$

$$\varepsilon(nat) \longrightarrow \forall p\ (\varepsilon(p) \Rightarrow (\varepsilon(nat) \Rightarrow \varepsilon(p) \Rightarrow \varepsilon(p)) \Rightarrow \varepsilon(p))$$

$$\varepsilon(x \dot{\Rightarrow} y) \longrightarrow \varepsilon(x) \Rightarrow \varepsilon(y)$$

$\mathcal{T}$ super-consistent, hence its proofs strongly terminate

Termination

Strong termination for proofs of $\mathcal{T}$ implies strong termination for terms of System T

Types of the System T are terms of the theory $\mathcal{T}$

Terms of type A in the System T translate to proofs of $\varepsilon(A)$

- $|x| = x$, $|u\ v| = |u|\ |v|$, $|\lambda x : A\ u| = \lambda x\ |u|$
- $|0| = \lambda p \lambda x \lambda f\ x$
- $|S| = \lambda n \lambda p \lambda x \lambda f\ (f\ n\ (n\ p\ x\ f))$
- $|Rec^A(t, u, n)| = (|n|\ A\ |t|\ |u|)$

If $t \longrightarrow^1 u$ in the System T then $|t| \longrightarrow^+ |u|$ in the theory $\mathcal{T}$

$$|Rec^A(x\ f\ 0)| \longrightarrow^* (|0|\ A\ x\ f) = (\lambda p \lambda x \lambda f\ x)\ A\ x\ f \longrightarrow^+ x$$

$$|Rec^A(x, f, (S\ n))|$$
$$\longrightarrow^* (|(S\ n)|\ A\ x\ f) = (\lambda p \lambda x \lambda f\ (f\ |n|\ (|n|\ p\ x\ f)))\ A\ x\ f)$$
$$\longrightarrow^+ (f\ |n|\ (|n|\ A\ x\ f)) = (f\ |n|\ |Rec^A(x, f, n)|) =$$
$$|(f\ n\ Rec^A(x, f, n))|$$

V. Martin-Löf Type theory

In Deduction modulo, reduction on terms and propositions

In the $\lambda\Pi$-calculus and in the $\lambda 1$-calculus reduction on terms, propositions and proofs

Arithmetic without Peano's symbol, induction as a deduction rule, reduction rules of System T: Martin-Löf Type theory

Equality

Instead of

$$refl : \forall x : nat \ (x = x)$$

a deduction rule: the reflexivity rule

$$\frac{}{\Gamma \vdash t = t}$$

To interpret this rule, a symbol *refl* such that for all t of type *nat*, $refl(t)$ is a proof of $t = t$

Second axiom of equality

$$\forall c \forall x \forall y\ (x = y \Rightarrow x \ \epsilon\ c \Rightarrow y \ \epsilon\ c)$$

a sort for classes and a comprehension scheme?
Instead: $nat \rightarrow Type$, write $(P\ t)$ the proposition formerly written $t \ \epsilon\ P$
Yet, no way to express

$$\forall P : nat \rightarrow Type\ \forall x : nat\ \forall y : nat\ (x = y \Rightarrow (P\ x) \Rightarrow (P\ y))$$

For each term P of type $nat \rightarrow Type$, an axiom

$$\forall x : nat\ \forall y : nat\ (x = y \Rightarrow (P\ x) \Rightarrow (P\ y))$$

or a deduction rule

$$\frac{\Gamma \vdash x = y \quad \Gamma \vdash (P\ x)}{\Gamma \vdash (P\ y)}$$

To interpret this rule, we introduce a symbol L such that if $\pi : (t = u)$ and $\pi' : (P\ t)$, the $L(P, t, u, \pi, \pi') : (P\ u)$

Recursion

Rec such that

if $P : nat \rightarrow Type$

$\pi : (P\ 0)$

$\pi' : (\forall n : nat\ ((P\ n) \Rightarrow (P\ (S\ n))))$

$t : nat$

then $Rec(P, \pi, \pi', n) : (P\ t)$

Reduction

$$L(P, a, a, \mathit{refl}(a), \pi) \longrightarrow \pi$$

refl: intro, *L*: elim

$$\mathit{Rec}(P, a, g, 0) \longrightarrow a$$

$$\mathit{Rec}(P, a, g, (S\ n)) \longrightarrow (g\ n\ \mathit{Rec}(P, a, g, n))$$

$0, S$: intro, *Rec*: elim

Predecessor, the addition and the multiplication can be defined
No need to take them as primitive symbols

But cannot define by induction the predicate symbol
$Null : nat \to Type$ (kind) keep the axiom

$$P_4 : \forall x : nat\ (0 = (S\ x) \Rightarrow \bot)$$

Termination

All terms in Martin-Löf type theory strongly terminate

Final rule

If t irreducible closed term, then

- the term t does not have the type $\bot$
- if the term t has type $\Sigma x : A\ B$ then it has the form $\langle v, w \rangle$
- if the term t has type $A + B$ then it has the form $i(v)$ or $j(w)$
- if the term t has type $v = w$ then it has the form $refl(v)$, hence the terms v and w are identical
- if the term t has type nat then it has the form 0 or $S(v)$

By induction on the structure of t

The term t has the form $(u\ c_1\ ...\ c_n)$ where u is not an application. We consider the following cases:

- u is the constant $0 : nat$,
- u is the constant $S : nat \to nat$, in which case $n = 1$,
- u has the form $Rec(P, a, g, t)$, in which case, by induction hypothesis $t = 0$ or $t = S(v)$, contradicting the fact that the term is irreducible,
- ...

Witness

If a proposition of the form $\exists x : A \; B$ has a closed proof, then there exists a term t such that the proposition $(t/x)B$ is provable

VI. Inductive types

Besides natural numbers, other datatypes: lists, trees, etc.
For instance

$$nil : list$$

$$cons : nat \to list \to list$$

and a rule

$$\frac{a : (P\ nil) \quad g : \forall a : nat\ \forall l : list\ ((P\ l) \to (P\ (cons\ a\ l)))) \quad l : list}{Rec(P, a, g, l) : (P\ l)}$$

Build functions by induction on the structure of lists
Prove properties of list by induction

After the break

Polymorphism