MPRI 2-7-1

week 4 - Oct. 9th

Functions in HOL

Normal terms in simply typed calculus

Normal λ-terms: x , $\lambda x.n$, $\lambda x_1.\lambda x_2.n$ …

$(x\ n_1\ n_2\ \ldots\ n_m)$

in the end : $\lambda x_1.\lambda x_2.\ \ldots\ \lambda x_k.\ (x\ n_1\ n_2\ \ldots\ n_m)$

Consider the following signature :

$0 : \iota$, $S : \iota \to \iota$, $+ \times : \iota \to \iota \to \iota$ what terms $f : \iota \to \iota$ can we construct ?

S $(+\ n)$ $(\times\ n)$

$\lambda x^{\iota}.\ n$

$\lambda x^{\iota}.\ 0$

$(S\ n)$

$(+\ n_1\ n_2)$ only polynomials (with constant exponents)

$(\times\ n_1\ n_2)$

x^{ι}

One version of HOL

base types : ι and o
HOL rules for ⇒ and ∀

constants: 0, S, + ×
Axioms: ∀ x. 0+x = x, ∀ x y . S(x) + y = S(x + y),
∀ x. 0 × x = 0, ∀ x y . S(x) × y = x × y + y,
∀ x. 0 ≠ S(x), injectivity of S
induction

Can be extended with more base types and induction principles
Can be extended with the excluded middle

Implemented and used in real systems : HOL, HOL-light, Isabelle-HOL…

Some properties of HOL

Very simple model

Model of simply typed λ-calculus, $|\iota| \equiv \mathbb{N}$, $|o| \equiv \{0,1\}$

$|\Rightarrow| \equiv$ boolean implication

$|\forall_T|(A) \equiv \min_{\alpha \in |T|} |A|(\alpha)$

$|0| \equiv 0$, $|S| \equiv x \mapsto x+1$, …

The formalism enjoys cut-elimination property

Intuitionistic proofs are constructive

Some inductive definitions in HOL

The smallest set such that :

- *even* (0)
- ∀ x. *even* (x) ⇒ *even* (S(S(x)))

Any set closed by the two properties contains *even* :

$$(\mathit{even}\ \mathrm{n}) \equiv \forall X : \iota \to o\ .$$
$$(X\ 0) \Rightarrow$$
$$(\forall y.\ (X\ y) \Rightarrow (X\ (S\ (S\ y)))) \Rightarrow$$
$$(X\ \mathrm{n})$$

A Proof by induction

$\forall X : \iota \rightarrow o .$

$(X\ 0) \Rightarrow$

$(\forall y.\ (X\ y) \Rightarrow (X\ (S\ (S\ y)))) \Rightarrow$

$(X\ n)$

$$\begin{cases} (\mathit{even}\ x) \Rightarrow \exists\ y\ .\ x = y + y \\ P \equiv \lambda\ x\ .\ \exists\ y\ .\ x = y + y \end{cases}$$

$(\exists\ y\ .\ 0 = y + y) \Rightarrow$

$(\forall\ x.\ \ \exists\ y\ .\ x = y + y \ \Rightarrow\ \exists\ y\ .\ (S\ (S\ x)) = y + y) \Rightarrow$

$\exists\ y\ .\ x = y + y$

two induction cases to prove

A more advanced inductive predicate

What is a *strongly normalizing* term ?

No infinite path : $t \rhd t_1 \rhd t_2 \rhd t_3 \rhd \ldots$

Define it inductively ?

$t \in SN$ iff $\forall\, t',\ t \rhd t' \Rightarrow t' \in SN$

The smallest set s.t. $(\forall\, t',\ t \rhd t' \Rightarrow t' \in SN) \Rightarrow t \in SN$

Only one clause !
base case : t is normal (then it is SN)

$(SN\ u) \equiv$

$\forall\, X : \Lambda \to o\ .$

$(\forall\, t : \Lambda\ .\ (\forall\, t' : \Lambda\ .\ (\beta\ t\ t') \Rightarrow X\ t') \Rightarrow X\ t)$

$\Rightarrow (X\ u)$

Using this definition

$$\forall\, X : \Lambda \to o \;.\quad (\forall\, t : \Lambda \;.\; (\forall\, t' : \Lambda \;.\; (\beta\; t\; t') \Rightarrow X\; t') \;\Rightarrow t) \;\Rightarrow (X\; u)$$

Can we prove $(\beta\; u\; u)$ is false ?

$$(\forall\, t : \Lambda \;.\; (\forall\, t' : \Lambda \;.\; (\beta\; t\; t') \Rightarrow \neg(\beta\; t'\; t')) \Rightarrow \neg(\beta\; t\; t)) \;\Rightarrow \neg(\beta\; u\; u)$$

$$\forall\, t : \Lambda \;.\; (\forall\, t' : \Lambda \;.\; (\beta\; t\; t') \Rightarrow \neg(\beta\; t'\; t')) \Rightarrow \neg(\beta\; t\; t)$$

given t, $\forall\, t' : \Lambda \;.\; (\beta\; t\; t') \Rightarrow \neg(\beta\; t'\; t')$ show $\neg(\beta\; t\; t)$

$(\beta\; t\; t) \Rightarrow \neg(\beta\; t\; t)$ indeed entails $\neg(\beta\; t\; t)$

Specifying a recursive function

We want : $(\exp\ x\ 0) = (S\ 0)$

$(\exp\ x\ (S\ y)) = (\exp\ x\ y) \times x$

$\exp\ x\ 0\ r \quad \Rightarrow \quad r = (S\ 0)$

$\exp\ x\ (S\ y)\ r \quad \Rightarrow \quad r = \times\ x\ r' \wedge \exp\ x\ y\ r'$

$\exp\ a\ b\ c \equiv$

$\forall\ R : \iota \to \iota \to \iota \to o\ .$

$(\forall\ x\ .\ R\ x\ 0\ 1) \to$

$(\forall\ x\ y\ r.\ R\ x\ y\ r \to R\ x\ (S\ y)\ x \times r) \to$

$(R\ a\ b\ c)$

Specifying a recursive function

Ack(0, n) = (S n)
Ack(S m, 0) = Ack(m, (S 0))
Ack(S m, S n) = Ack(m, Ack(S m, n))

λ a:ι.λ b:ι.λ r:ι.
∀ X : ι→ ι→ ι→o .
(∀ n. (X 0 n (S n)) ⇒
(∀ m. ∀ r. (X m (S 0) r) ⇒ (X (S m) 0 r)) ⇒
(∀ m. ∀ n. ∀ r. ∀ r'. (X (S m) n r') ⇒ (X m r' r) ⇒ (X (S m)(S n) r)) =>
(X a b r)

Proving the existence of a recursive function

$$
\begin{aligned}
\mathrm{Ack} \equiv\ & \lambda\, a{:}\iota.\lambda\, b{:}\iota.\lambda\, r{:}\iota. \\
& \forall\, X : \iota \to \iota \to \iota \to o\ . \\
& (\forall\, n.\ (X\ 0\ n\ (S\ n)) \Rightarrow \\
& (\ \forall\, m.\ \forall\, r.\ (X\ m\ (S\ 0)\ r) \Rightarrow (X\ (S\ m)\ 0\ r)) \Rightarrow \\
& (\ \forall\, m.\ \forall\, n.\ \forall\, r.\ \forall\, r'.\ (X\ (S\ m)\ n\ r') \Rightarrow (X\ m\ r'\ r) \Rightarrow (X\ (S\ m)(S\ n)\ r)) => \\
& (X\ a\ b\ r)
\end{aligned}
$$

$\forall\, a\ .\ \forall\, b\ .\ \exists\, r\ .\ (\mathrm{Ack}\ a\ b\ r)$ by induction

induction over a : $\forall\, b\ .\ \exists\, r\ .\ (\mathrm{Ack}\ a\ b\ r)$

$\forall\, b\ .\ \exists\, r\ .\ (\mathrm{Ack}\ 0\ b\ r)$

$\forall\, b\ .\ \exists\, r\ .\ (\mathrm{Ack}\ a\ b\ r) \Rightarrow \forall\, b\ .\ \exists\, r\ .\ (\mathrm{Ack}\ (S\ a)\ b\ r)$

Proving the existence of a recursive function

$$
\begin{aligned}
\mathrm{Ack} \equiv\ & \lambda\, a{:}\iota.\lambda\, b{:}\iota.\lambda\, r{:}\iota. \\
& \forall\, X : \iota \to \iota \to \iota \to o\ . \\
& (\forall\, n.\ (X\ 0\ n\ (S\ n)) \Rightarrow \\
& (\ \forall\, m.\ \forall\, r.\ (X\ m\ (S\ 0)\ r) \Rightarrow (X\ (S\ m)\ 0\ r)) \Rightarrow \\
& (\ \forall\, m.\ \forall\, n.\ \forall\, r.\ \forall\, r'.\ (X\ (S\ m)\ n\ r') \Rightarrow (X\ m\ r'\ r) \Rightarrow (X\ (S\ m)(S\ n)\ r)) => \\
& (X\ a\ b\ r)
\end{aligned}
$$

induction over a : $\forall\, b\ .\ \exists\, r\ .\ (\mathrm{Ack}\ a\ b\ r)$

$\forall\, b\ .\ \exists\, r\ .\ (\mathrm{Ack}\ 0\ b\ r)$

$\forall\, b\ .\ \exists\, r\ .\ (\mathrm{Ack}\ a\ b\ r) \Rightarrow \forall\, b\ .\ \exists\, r\ .\ (\mathrm{Ack}\ (S\ a)\ b\ r)$

induction over b : $\exists\, r\ .\ (\mathrm{Ack}\ (S\ a)\ b\ r)$

$\exists\, r\ .\ (\mathrm{Ack}\ (S\ a)\ 0\ r)$

$\exists\, r\ .\ (\mathrm{Ack}\ (S\ a)\ (S\ b)\ r)$

Naming functions: Hilbert operator

Extending the language

$\varepsilon(P)$ "The" object verifying P ("choice operator")

$$\frac{\vdash (P\ t)}{\vdash (P\ \varepsilon(P))}$$

"If one guy can do it, it's ε"

$\exists\ x\ .\ P\ x \quad \Leftrightarrow \quad P\ \varepsilon(P)$

(can be used instead of $\exists$)

Using the Hilbert operator

exp_f a b = ε(λ x . (exp a b x))

exp_f = λ a . λ b . ε(λ x . (exp a b x))

We can show exp_f a 0 = 1, exp_f a (S b) = a × exp_f a b

The proof of these equations can be mechanized

What do we miss ? Computations !

Architecture of HOL implementations (in a nutshell)

- One does not construct the proof derivation (as a tree data structure)
- ML was invented as the meta-language of HOL implementations !
- Safety architecture :
  - An abstracted datatype for judgements $\Gamma \vdash A$
  - Only a few simple tactics allow to construct these judgements
  - These tactics correspond to logical rules
  - These tactics are the Trusted Computing Base
  - More complex tactics are assembled on top of those tactics (using ML)

How *un*constructive is the ε operator ?

Remarks:

1. $\forall\, x\, .\, \forall\, y\, .\, x{=}y \vee x{\neq}\, y$ is provable in HA
2. $(A \vee \neg A) \wedge (B \vee \neg B) \Rightarrow (A \wedge B) \vee \neg(A \wedge B)$
3. $(A \vee \neg A) \wedge (B \vee \neg B) \Rightarrow (A \vee B) \vee \neg(A \vee B)$
4. $(A \vee \neg A) \wedge (B \vee \neg B) \Rightarrow (A \Rightarrow B) \vee \neg(A \Rightarrow B)$

Why is classical arithmetic undecidable ?

$\forall\, x.\, A(x) \vee \neg A(x)$ does not entail $(\forall\, x.\, A(x)) \vee \neg(\forall\, x.\, A(x))$

does not entail $(\exists\, x.\, A(x)) \vee \neg(\exists\, x.\, A(x))$

with ε, Heyting arithmetic becomes classical !

We prove that for any proposition A, $\vdash A \vee \neg A$ holds (is provable)

by induction over the size of A

1. $\forall\, x\, .\, \forall\, y\, .\, x{=}y \vee x{\neq}\, y$ is provable in HA
2. $(A \vee \neg A) \wedge (B \vee \neg B) \Rightarrow (A \wedge B) \vee \neg(A \wedge B)$
3. $(A \vee \neg A) \wedge (B \vee \neg B) \Rightarrow (A \vee B) \vee \neg(A \vee B)$
4. $(A \vee \neg A) \wedge (B \vee \neg B) \Rightarrow (A \Rightarrow B) \vee \neg(A \Rightarrow B)$

Suppose we know :

$\forall\, x.\, A(x) \vee \neg A(x)$ $(\exists\, x.\, A(x)) \vee \neg(\exists\, x.\, A(x))$

$(\exists\, x.\, A(x)) \vee \neg(\exists\, x.\, A(x))$

We prove that for any proposition A, $\vdash A \vee \neg A$ holds (is provable)

by induction over the size of A

Suppose we know : number of connectives

$\vdash \forall x.\ A(x) \vee \neg A(x)$ let us prove $\vdash (\exists x.\ A(x)) \vee \neg(\exists x.\ A(x))$

by I.H : $\vdash A(\varepsilon(A)) \vee \neg A(\varepsilon(A))$

if $A(\varepsilon(A))$, then $\exists x.\ A(x)$ (trivial)

if $\neg A(\varepsilon(A))$: $\exists x.\ A(x)$ entails $A(\varepsilon(A))$, thus $\bot$.

so $\neg(\exists x.\ A(x))$

$(\exists x.\ A(x)) \vee \neg(\exists x.\ A(x))$

We prove that for any proposition A, $\vdash A \vee \neg A$ holds (is provable)

by induction over the size of A

Suppose we know :

$\vdash \forall x.\ A(x) \vee \neg A(x)$ let us prove $\vdash (\forall x.\ A(x)) \vee \neg(\forall x.\ A(x))$

---

by I.H : $\vdash A(\varepsilon(\neg A)) \vee \neg A(\varepsilon(\neg A))$

if $\neg A(\varepsilon(\neg A))$, then $\neg(\forall x.\ A(x))$ (trivial)

if $A(\varepsilon(\neg A))$: $\exists x.\ \neg A(x)$ entails $\neg A(\varepsilon(\neg A))$, thus $\bot$.

so $\neg(\exists x.\ \neg A(x))$

now, given x, we can show $\neg A(x) \Rightarrow \exists y.\ \neg A(y) \Rightarrow \bot$

so $\forall x.\ \neg\neg A(x)$ but $A(x) \vee \neg A(x)$

so $\forall x.\ A(x)$

Summing up

In other words :
Heyting arithmetic with Hilbert operator = Peano + Hilbert operator

Computing with epsilon is not easy

However, HOL (without epsilon and EM) is constructive

I was asked : what is the difference between HOL and system F ?

HOL : formalism — quantification over propositions

System F : type system — quantification over types

Link : when we view proofs as λ-terms (starting next week)

Normalization of system F (actually F_ω) will allow to show cut elimination in HOL

Defining more function
System T

Adding Product Types

$T ::= At \mid T \to T \mid T \times T$

$t := x \mid t\ t \mid \lambda x.t \mid (t,t) \mid \pi_1(t) \mid \pi_2(t)$

$\pi_1(t,u) \rhd t$

$\pi_2(t,u) \rhd u$

Strong normalization proof follows. Essentially:

$$|T_1 \times T_2| \equiv \{t \in SN \mid t \rhd^* (u_1, u_2) \Rightarrow u_1 \in |T_1| \wedge u_2 \in |T_2|\}$$

possible addition, surjective pairing: $(\pi_1(t), \pi_2(t)) \rhd t$

Adding Sum Types

$T ::= At \mid T \to T \mid T + T$

$t := x \mid t\ t \mid \lambda x.t \mid i(t) \mid j(t) \mid \delta(t, x.t, x.t)$

$\delta(i(t), x.u, y.v) \triangleright u[x \setminus t]$

$\delta(j(t), x.u, y.v) \triangleright v[y \setminus t]$

Again, strong normalization proof follows. Essentially:

$$|T_1 + T_2| \equiv \{t \in SN \mid t \triangleright^* i(u) \Rightarrow u \in |T_1| \wedge t \triangleright^* j(u) \Rightarrow u \in |T_2| \}$$

Going further: System T

Adding primitive natural numbers and a recursor

In Coq:

```
Inductive nat : Type :=
  |  O : nat
  |  S : nat -> nat.
```

Additional atomic type: N

Additional constants: 0 : N, S : N → N

Additional operators: R_T : N → T → (N → T → T) → T

$$R\ 0\ t_0\ t\_S \quad \triangleright \quad t_0$$

$$R\ (S\ n)\ t_0\ t_S \quad \triangleright \quad t_S\ n\ (R\ n\ t_0\ t_S)$$

An expressive system

Defining addition, multiplication
but also exponentiation, predecessor, Ackermann…

Strong Normalization proof