0. What we have seen so far
Proofs, constructive proof, theory
Today

An example of theory: arithmetic

A new topic: simply typed lambda-calculus
A theory: arithmetic
Examples of propositions

\[ \forall x \exists y \ (x = 2 \times y \lor x = 2 \times y + 1) \]

\[ \exists y \ (4 = 2 \times y) \]

\[ \exists x \exists y \ (7 = (x + 2) \times (y + 2)) \]

\[ \forall x \exists y \ (y \geq x \land \text{prime}(y)) \]

\[ \geq, \ prime? \]
The language of arithmetic

0, S, Pred, +, ×
Null, =
The axioms of infinity

\[ Pred(0) = 0 \]
\[ \forall x \ (Pred(S(x)) = x) \]

\[ Null(0) \]
\[ \forall x \ \neg Null(S(x)) \]

\[ Pred(0) \rightarrow 0 \]
\[ Pred(S(x)) \rightarrow x \]
\[ Null(0) \rightarrow \top \]
\[ Null(S(x)) \rightarrow \bot \]
The axioms of addition and multiplication

\[ \forall y \ (0 + y = y) \]
\[ \forall x \ \forall y \ (S(x) + y = S(x + y)) \]
\[ \forall y \ (0 \times y = 0) \]
\[ \forall x \ \forall y \ (S(x) \times y = (x \times y) + y) \]

0 + y \rightarrow y
S(x) + y \rightarrow S(x + y)
0 \times y \rightarrow 0
S(x) \times y \rightarrow (x \times y) + y
Induction

No other numbers than those constructed with 0 and $S$
Every class containing 0 and closed by $S$ contains everything
Besides $\iota$, a sort $\kappa$ for classes, a predicate symbol $\epsilon$

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

Comprehension axiom scheme: existence of some classes

$$\forall x_1 \ldots \forall x_n \exists c \forall y \ (y \in c \Leftrightarrow A)$$

if $A$ does not contain $\epsilon$ (predicative arithmetic)
Comprehension as a rewrite rule

\[ \forall x_1 \ldots \forall x_n \exists c \forall y \ (y \in c \iff A) \]

Introduce a notation for this class: \( f_{x_1, \ldots, x_n, y, A(x_1, \ldots, x_n)} \)

\[ \forall x_1 \ldots \forall x_n \forall y \ (y \in f_{x_1, \ldots, x_n, y, A(x_1, \ldots, x_n)} \iff A) \]

\[ y \in f_{x_1, \ldots, x_n, y, A(x_1, \ldots, x_n)} \rightarrow A \]
How to use these axioms to prove $\forall y \ (y + 0 = y)$?

High school proof:
$0 + 0 = 0$
$\forall x \ (x + 0 = x \Rightarrow S(x) + 0 = S(x))$
hence $\forall y \ (y + 0 = y)$

Using the axioms

$\forall y \ (0 + y = y)$

$\forall x \ \forall y \ (S(x) + y = S(x + y))$
How do we know

\[0 + 0 = 0 \implies \forall x \ (x + 0 = x \implies S(x) + 0 = S(x))\]
\[\implies \forall y \ (y + 0 = y)\]
How do we know

\[ 0 + 0 = 0 \Rightarrow \forall x \ (x + 0 = x \Rightarrow S(x) + 0 = S(x)) \]
\[ \Rightarrow \forall y \ (y + 0 = y) \ ? \]

\[ \forall c \ (0 \in c \Rightarrow \forall x \ (x \in c \Rightarrow S(x) \in c) \Rightarrow \forall y \ y \in c) \]
How do we know

\[ 0 + 0 = 0 \Rightarrow \forall x \ (x + 0 = x \Rightarrow S(x) + 0 = S(x)) \]

\[ \Rightarrow \forall y \ (y + 0 = y) ? \]

\[ \forall c \ (0 \in c \Rightarrow \forall x \ (x \in c \Rightarrow S(x) \in c) \Rightarrow \forall y \ y \in c) \]

\[ \exists c \forall y \ (y \in c \Leftrightarrow y + 0 = y) \]
Induction as a rewrite rule

Induction axiom: all objects of sort $\iota$ are natural numbers
Alternative: not all objects are natural numbers, a predicate symbol $N$ for the natural numbers

\[ \forall c \ (0 \in c \Rightarrow \forall x \ (x \in c \Rightarrow S(x) \in c) \Rightarrow \forall y \ (N(y) \Rightarrow y \in c)) \]

More axioms

\[ N(0) \]
\[ \forall x \ (N(x) \Rightarrow N(S(x))) \]

Relativization of quantifiers

\[ \forall x \ (N(x) \Rightarrow \exists y \ (N(y) \land (x = 2 \times y \lor x = 2 \times y + 1))) \]
∀y (N(y) ⇒ ∀c (0 ∈ c ⇒ ∀x (x ∈ c ⇒ S(x) ∈ c) ⇒ y ∈ c))

Converse provable (with N(0) and ∀x (N(x) ⇒ N(S(x))))
Alternative:

∀y (N(y) ⇔ ∀c (0 ∈ c ⇒ ∀x (x ∈ c ⇒ S(x) ∈ c) ⇒ y ∈ c))

(N(0) and ∀x (N(x) ⇒ N(S(x)))) dropped)

N(y) → ∀c (0 ∈ c ⇒ ∀x (x ∈ c ⇒ S(x) ∈ c) ⇒ y ∈ c)
Equality

Classes also used to express the properties of equality

$$\forall x \forall y \ (x = y \iff \forall c \ (x \in c \Rightarrow y \in c))$$

$$x = y \longrightarrow \forall c \ (x \in c \Rightarrow y \in c)$$

Exercise: prove reflexivity, symmetry, transitivity, and substitutivity
To sum up

\[
\begin{align*}
Pred(0) & \rightarrow 0 \\
Pred(S(x)) & \rightarrow x \\
Null(0) & \rightarrow \top \\
Null(S(x)) & \rightarrow \bot \\
0 + y & \rightarrow y \\
S(x) + y & \rightarrow S(x + y) \\
0 \times y & \rightarrow 0 \\
S(x) \times y & \rightarrow (x \times y) + y \\
y \in f_{x_1, \ldots, x_n, y, A}(x_1, \ldots, x_n) & \rightarrow A \\
N(y) & \rightarrow \forall c \ (0 \in c \Rightarrow \forall x \ (x \in c \Rightarrow S(x) \in c) \Rightarrow y \in c) \\
x = y & \rightarrow \forall c \ (x \in c \Rightarrow y \in c)
\end{align*}
\]
Exercise

Prove

\[ \forall x \forall y \ (S(x) = S(y) \Rightarrow x = y) \]
\[ \forall x \ \neg (0 = S(x)) \]
Another exercise

Prove

\[ \forall x \exists y \ (x = 2 \times y \lor x = 2 \times y + 1) \]
Message to take home

A lot of proofs can be formalized in arithmetic

Arithmetic can be defined with rewrite rules only
Simply typed lambda-calculus
I. Functional programming languages
Functions

Do not focus on what computers do, but on what they compute

A function mapping input to output

A programming language: a language of functions
Functions

$18^{th}$ century: $2 \times x + 4$
$20^{th}$ century: $x \mapsto 2 \times x + 4, \lambda x \ 2 \times x + 4$

Two constructions:
abstraction (building a function): $\lambda x \ t$
application (using a function): $app(t, u) = (t \ u)$

Example: $((\lambda x \ (2 \times x + 4)) \ 5)$

Exercise: arity of these symbols?
The $\beta$-reduction rule

$$((\lambda x : A \ t) \ u) \rightarrow (u/x)t$$
Termination?
Termination?

\[ \delta = \lambda x \ (x \ x) \]

\[(\delta \ \delta) \rightarrow (\delta \ \delta) \rightarrow (\delta \ \delta) \rightarrow (\delta \ \delta) \rightarrow \ldots \]
Simple types

Many sorted language with an infinite number of sorts
An inductive definition

- $P_1, \ldots, P_n$ are simple types (e.g. $nat$, $string$, $bool$)
- if $A$ and $B$ are simple types, then $A \rightarrow B$ is a simple type
Terms of simply typed λ-calculus

For each simple type, an infinite set of variables of this type

- if \( x \) is a variable of type \( A \), then \( x \) is a term of type \( A \)
- if \( t \) is a term of type \( A \rightarrow B \) and \( u \) a term of type \( A \), then \( (t \ u) \) is a term of type \( B \)
- if \( x \) is a variable of type \( A \) and \( t \) a term of type \( B \), then \( \lambda x : A \ t \) is a term of type \( A \rightarrow B \)
Context: a type to some variables: \( x : \text{nat}, g : \text{nat} \rightarrow \text{nat} \rightarrow \text{nat} \)

In the context \( \Gamma = (x : \text{nat}, g : \text{nat} \rightarrow \text{nat} \rightarrow \text{nat}) \) the term
\( t = \lambda y : \text{nat} \ (g \ y \ x) \) has type \( A = (\text{nat} \rightarrow \text{nat}) \)

\( \Gamma \vdash t : A \)
Inductive definition

\[ \Gamma 
\vdash x : A \quad x : A \text{ in } \Gamma \]

\[ \Gamma, x : A \vdash t : B \]
\[ \Gamma \vdash \lambda x : A \ t : A \rightarrow B \]

\[ \Gamma \vdash t : A \rightarrow B \quad \Gamma \vdash u : A \]
\[ \Gamma \vdash (t \ u) : B \]
II. The termination of the Simply typed $\lambda$-calculus
A first attempt

Induction over term structure:
• $t = x$ irreducible, hence terminating
• if $t$ terminates, then $\lambda x : A t$ terminates
• if $t$ and $\nu$ terminate, $(t \nu)$ terminates

Reductions can be in $t$, in $\nu$, but also at the root
E.g. (untyped $\lambda$-calculus)
$\delta = \lambda x (x x)$ terminating, but not $(\delta \delta)$
The problem and a solution

If \( t \) is or reduces to \( \lambda x \ u \), then \( (t \ v) \) reduces to \( (v/x)u \)

When a proof by induction fails: try to prove a stronger property: if \( t : A \) then \( t \in R_A \)

- If \( A \) atomic, \( t \) in \( R_A \) if \( t \) strongly terminating
- If \( A = B \rightarrow C \), \( t \) in \( R_A \) if strongly terminating and whenever \( t \) reduces to \( \lambda x : B \ u \), then for every \( v \) in \( R_B \), \( (v/x)u \) in \( R_C \)

Definition by induction over type structure
Four easy lemmas

1. If $A$ is a type and $x$ a variable, then $x \in R_A$
   $x$ strongly terminates and never reduces to abstraction
2. If $t$ in $R_A$ and $t$ reduces to $t'$, then $t'$ in $R_A$
   If $t$ reduces to $t'$ and $t$ strongly terminates, then $t'$ strongly terminates
   If $A = B \rightarrow C$ and $t'$ reduces to $\lambda x : B \ u$, then so does $t$, hence
   for every $v$ in $R_B$, $(v/x)u$ in $R_C$
3. If all one-step reducts of $t = (u_1 u_2)$ in $R_A$, then $t$ in $R_A$
First $t$ strongly terminates. A reduction sequence $t = t_1, t_2, ...$
If one element finite. Otherwise, $t \rightarrow^1 t_2$, $t_2$ in $R_A$, strongly
terminates, reduction sequence finite
If $A = B \rightarrow C$, and $t$ reduces to $\lambda x : B \ v$, let
$t = t_1, t_2, ..., t_n = \lambda x : B \ v$ reduction sequence from $t$ to $\lambda x : B \ v$.
$t_1$ application $t_n$ abstraction, thus $n \geq 2$, $t \rightarrow^1 t_2 \rightarrow^* t_n$,
t_2 \in R_A, thus $(w/x)v$ in $R_C$
4. If \( t_1 \in R_{A \rightarrow B} \) and \( t_2 \in R_A \), then \( (t_1 \, t_2) \in R_B \)

- \( t_1 \) in \( R_{A \rightarrow B} \), \( t_2 \) in \( R_A \), hence strongly terminate
- \( n_1 \) maximum length of a reduction sequence issued from \( t_1 \)
- \( n_2 \) maximum length of a reduction sequence issued from \( t_2 \)

By induction on \( n_1 + n_2 \), \( (t_1 \, t_2) \in R_B \)

Using 3., we only need to prove that every of its one step reducts is in \( R_B \). If reduction in \( t_1 \) or in \( t_2 \) 2. and induction hypothesis

Otherwise, \( t_1 = \lambda x : A \, u \) and reduct is \( (t_2/x)u \). In \( R_B \) by definition of \( R_{A \rightarrow B} \)
The theorem

t term of type $A$ and $\sigma$ substitution mapping each variable of type $B$ to element of $R_B$, then $\sigma t$ in $R_A$

By induction on the structure of $t$
- If $t$ variable, then $\sigma t$ in $R_A$ by definition of $\sigma$
- If $t = (u_1 \ u_2)$ then $u_1 : B \rightarrow A$ and $u_2 : B$

By induction hypothesis, $\sigma u_1$ in $R_{B \rightarrow A}$ and $\sigma u_2$ in $R_B$

By 4. $\sigma t = (\sigma u_1 \ \sigma u_2)$ in $R_A$
• If \( t = \lambda x : B \ u \) where \( u \) of type \( C \)
\( \sigma t = \lambda x : B \ \sigma u \), a reduction sequence issued from this term can only reduce \( \sigma u \), by induction hypothesis, \( \sigma u \) in \( R_C \), thus reduction sequence is finite.
Every reduct of \( \sigma t \) of the form \( \lambda x : B \ v \) where \( v \) reduct of \( \sigma u \)
\( w \) any term of \( R_B \), want \( (w/x)v \) in \( R_C \)
\( (w/x)v \) obtained by reducing \( ((w/x) \circ \sigma)u \)
By induction hypothesis, \( ((w/x) \circ \sigma)u \) in \( R_C \). Hence, by 2. \( (w/x)v \) in \( R_C \)
Finally

If $t$ of type $A$, then $t$ strongly terminates

$\sigma$ mapping each variable of type $B$ to itself (in $R_B$ by 1.)

$t = \sigma t$ in $R_A$. Hence it strongly terminates.
IV. A real programming language?
The computational expressivity of the simply typed \( \lambda \)-calculus

Add 0: \textit{nat} and \textit{S} : \textit{nat} \rightarrow \textit{nat}

The natural numbers: 0, (\textit{S} 0), (\textit{S} (\textit{S} 0))... 

The constant functions can be expressed in simply typed \( \lambda \)-calculus

\[
\lambda x : \textit{nat} \ (\textit{S} \ (\ldots(\textit{S} \ 0)\ldots))
\]

those that add a constant to their arguments also

\[
\lambda x : \textit{nat} \ (\textit{S} \ (\ldots(\textit{S} \ x)\ldots))
\]

But that is all
A lemma

An irreducible $\lambda$-term has the form

$$\lambda x_1 : A_1 \ldots \lambda x_n : A_n \ (x \ u_1 \ldots u_p)$$

where $x$ is either a variable or a constant

$\lambda x_1 : A_1 \ldots \lambda x_n : A_n \ t'$ where $t'$ not a $\lambda$ ($n$ possibly 0)

$t' = (t'' \ u_1 \ldots u_p)$ where $t''$ not an application ($p$ possibly 0)

$t''$ not an abstraction, if $p > 0$ because the term is irreducible and if $p = 0$ because $t'$ is not an abstraction
Message to take home

Simply typed lambda-calculus terminates

But is not very expressive
The witness property for constructive proofs

Termination of proof reduction