Proving with Computer Assistance
Lecture 1.2
Untyped λ-calculus

Herman Geuvers

λ-abstraction

Defining a function

$$\begin{aligned} f(x) &:= x^2 + 2 \\ f &: x \mapsto x^2 + 2 \\ g(x, y) &:= x^2 + y + 2 \end{aligned}$$

In λ-calculus we use λ-abstraction:

$$\begin{aligned} f &:= \lambda x.\, x^2 + 2 \\ g &:= \lambda x.\, \lambda y.\, x^2 + y + 2 \end{aligned}$$

- distinguish between term with a variable $x^2 + 2$ and the function $\lambda x.\, x^2 + 2$ that sends x to $x^2 + 2$.
- make explicit which variables are abstracted over.
- clearly distinguish between free and bound (occurrences) of variables.

Application

We have seen the functions f and g:

$$\begin{aligned} f &:= \lambda x.\, x^2 + 2 \\ g &:= \lambda x.\, \lambda y.\, x^2 + y + 2 \end{aligned}$$

Application:

$$f(3) \quad \text{no!} \qquad f\,3 \quad \text{or} \quad f \cdot 3 \quad \text{or} \quad (f\,3)$$

$\Rightarrow$ application is a binary operator which is usually not written.

Giving two arguments:

$$(g\,3)\,4 \qquad \text{or just} \quad g\,3\,4$$

because we omit brackets by associating them to the left.

Untyped λ-calculus

Untyped λ-calculus = Variables + λ-abstraction + application

$$\Lambda ::= \mathsf{Var} \mid (\Lambda\,\Lambda) \mid (\lambda \mathsf{Var}.\Lambda)$$

Notation

$M\,N\,P$	denotes	$(M\,N)\,P$	(so <u>not</u> $M\,(N\,P)$)
$\lambda xyz.M$	denotes	$\lambda x.\lambda y.\lambda z.M$	(or more precisely $\lambda x.(\lambda y.(\lambda z.M))$.)

Examples:
- $\mathbf{I} := \lambda x.x$
- $\mathbf{K} := \lambda x\,y.x$
- $\mathbf{S} := \lambda x\,y\,z.x\,z(y\,z)$
- $\omega := \lambda x.x\,x$
- $\Omega := \omega\,\omega$

Computing with λ-terms

Computation is done via the β-rule

$$(\lambda x.x^2 + 2)\,3 \to_\beta 3^2 + 2$$

DEFINITION β-equality, written as $=_\beta$ is the term reduction generated from the β-rule:

$$\boxed{(\lambda x.M)\,P \to_\beta M[x := P]}$$

where $M[x := P]$ denotes the substitution of P for all occurrences of x in M.

That $\to_\beta$ is a term reduction means that it is closed under the term-forming-operators. More precisely we have

$$\frac{M \to_\beta M'}{M\,P \to_\beta M'\,P} \qquad \frac{P \to_\beta P'}{M\,P \to_\beta M\,P'} \qquad \frac{M \to_\beta M'}{\lambda x.M \to_\beta \lambda x.M'}$$

Examples

Remember $\mathbf{I} := \lambda x.x$, $\mathbf{K} := \lambda x\, y.x$, $\mathbf{S} := \lambda x\, y\, z.x\, z(y\, z)$, $\omega := \lambda x.x\, x$, $\Omega := \omega\, \omega$.

$$\begin{aligned} \mathbf{I}\, P &\to_\beta P \\ \mathbf{K}\, P\, Q &\to_\beta \ldots \to_\beta P \\ \Omega &\to_\beta \Omega \end{aligned}$$

$$\begin{aligned} (\lambda x\, y.y\, x)\, P &\to_\beta \lambda y.y\, P \\ (\lambda x\, y.y\, x)\, y &\stackrel{??}{\to}_\beta \lambda y.y\, y \end{aligned}$$

No!
$\lambda y.M$ binds all occurrences of y in M. We cannot just substitute a term with a free y inside M.

Free and bound variables, alpha-equivalence

- $\lambda y.M$ binds all occurrences of y in M.
- We distinguish bound variables and free variables in a term: BV(M) and FV(M).
  (Better to say: bound and free occurrences of variables.)
- We consider term modulo renaming of bound variables (also called "modulo α-equality"):

$$\lambda x.M \equiv \lambda y.M[x := y]$$

  if y does not occur in M.

A more precise definition of $\to_\beta$:

$$(\lambda x.M)\,P \to_\beta M[x := P]$$

where the substitution $M[x := P]$ is defined by:
(1) rename the bound variables in M that occur free in P, obtaining M';
(2) replace all free occurrences of x in M' by P.

Alpha equivalence

Two terms M, N are α-equal, $M \equiv N$, in case they can be obtained from eachother via renaming bound variables.

EXAMPLES

$$\lambda x.\lambda y.x\,y \overset{??}{\equiv} \lambda y.\lambda x.y\,x$$
$$\lambda x.\lambda y.x\,y \overset{??}{\equiv} \lambda x.\lambda y.y\,x$$
$$\lambda x.\lambda y.x\,y \overset{??}{\equiv} \lambda x.\lambda y.y\,y$$
$$\lambda x.\lambda x.x\,x \overset{??}{\equiv} \lambda x.\lambda y.y\,y$$

Multi-step reduction and β-equality

- $\twoheadrightarrow_\beta$ is the transitive reflexived closure of $\rightarrow_\beta$.
  So $M \twoheadrightarrow_\beta P$ iff M β-reduces to P in 0 or more steps.
- $=_\beta$ is the transitive, reflexive, symmetric closure of $\rightarrow_\beta$.
  So $=_\beta$ is the least congruence obtained from $=_\beta$.

Examples of reductions:

$$
\begin{aligned}
\mathbf{I}\,P &\rightarrow_\beta P \\
\mathbf{K}\,P\,Q &\twoheadrightarrow_\beta P \\
\mathbf{K}\,\mathbf{I}\,P\,Q &\twoheadrightarrow_\beta Q \\
\mathbf{S}\,\mathbf{K}\mathbf{K} &\twoheadrightarrow_\beta \mathbf{I} \\
\Omega &\rightarrow_\beta \Omega
\end{aligned}
$$

Is λ-calculus consistent?

Why does λ-calculus "make sense"?
Could it be the case that $M =_\beta P$ for all M, P? (Then λ-calculus would be inconsistent...)

THEOREM λ-calculus satisfies the Church-Rosser property.

COROLLARY $\mathbf{K} \neq_\beta \mathbf{I}$ and so λ-calculus is consistent.

The computational power of λ-calculus

Untyped λ-calculus is Turing complete
Its power lies in the fact that you can solve recursive equations:
Is there a term M such that

$$M\,x =_\beta x\,M\,x?$$

Is there a term M such that

$$M\,x =_\beta \textbf{if}\ (\textbf{Zero}\,x)\ \textbf{then}\ 1\ \textbf{else}\ \textbf{Mult}\,x\,(M\,(\textbf{Pred}\,x))?$$

Yes, because we have a fixed point combinator:
- $\mathbf{Y} := \lambda f.(\lambda x.f(x\,x))(\lambda x.f(x\,x))$
Property:

$$\boxed{\mathbf{Y}\,f =_\beta f(\mathbf{Y}\,f)}$$

Untyped λ-calculus (ctd.)

Solving recursive equations using the fixed point combinator:

- For M a λ-term, $\mathbf{Y}\,M$ is a fixed point of M, that is

$$M\,(\mathbf{Y}\,M) =_\beta \mathbf{Y}\,M$$

- As a consequence, a question like "Is there a λ-term P such that $P\,x =_\beta x\,P\,x\,P$ (for all x)?" can be answered affirmative:

Representing data in λ-calculus

Booleans

$$\begin{aligned}
\textbf{true} &:= \lambda x\, y.\, x \\
\textbf{false} &:= \lambda x\, y.\, y \\
\textbf{if } M \textbf{ then } P \textbf{ else } Q &:= M\, P\, Q
\end{aligned}$$

Natural Numbers via the so-called Church Numerals

$$\begin{aligned}
c_0 &:= \lambda f\, x.x \\
c_1 &:= \lambda f\, x.f\, x \\
c_2 &:= \lambda f\, x.f(f\, x) \\
\dots & \\
c_n &:= \lambda f\, x.f^n\, x
\end{aligned}$$

where $f^n\, x$ is an n-times application of f on x.

Then, e.g.

$$\begin{aligned}
\textbf{Succ} &:= \lambda n\, f\, x.f(n\, f\, x) \\
\textbf{Zero} &:= \lambda n.n\,(\lambda y.\, \textbf{false}\,)\ \textbf{true}
\end{aligned}$$