λ-computability

Basic data and operations

Fixpoints

Decidability

The λ-calculus is a model of computable functions

Decidability, traditional presentation

λ-computability

Basic data and operations

Fixpoints

Decidability

The λ-calculus is a model of computable functions

Decidability, traditional presentation

Functions

Identity function

$$\mathsf{I} \quad \equiv \quad \lambda x.x$$

Function composition

$$g \circ f \quad \equiv \quad \lambda x.g(f(x))$$

Example

$$\begin{array}{rcl} \mathsf{I} \circ \mathsf{I} & \equiv & \lambda x.\mathsf{I}(\mathsf{I}x) \\ & \equiv & \lambda x.(\lambda y.y)\ ((\lambda z.z)\ x) \\ & \to_\beta & \lambda x.(\lambda z.z)\ x \\ & \to_\beta & \lambda x.x \end{array}$$

Booleans and conditionals

Boolean values

$$\begin{array}{rcl} \mathsf{T} & \equiv & \lambda x\, y.x \\ \mathsf{F} & \equiv & \lambda x\, y.y \end{array}$$

Conditional expression

$$\text{if } c \text{ then } a \text{ else } b \quad \equiv \quad c\ a\ b$$

Example

$$\begin{array}{rcl} \text{if } \mathsf{T} \text{ then } a \text{ else } b & \equiv & \mathsf{T}\ a\ b \\ & \equiv & (\lambda x\, y.x)\ a\ b \\ & \to_\beta & (\lambda y.a)\ b \\ & \to_\beta & a \end{array}$$

Exercise: boolean operators

The following λ-term encodes a boolean operator.
Which one?

$$\lambda a\, b. a b \mathsf{F}$$

Write terms for the other common operators.

Pairs and projections

Pair

$$\langle a, b\rangle \quad \equiv \quad \lambda s.s\ a\ b$$

Projections

$$\begin{array}{lcll}\pi_1 & \equiv & \lambda p.p\ (\lambda ab.a) & \qquad (\equiv \lambda p.p\ \mathsf{T}) \\ \pi_2 & \equiv & \lambda p.p\ (\lambda a\, b.b) & \qquad (\equiv \lambda p.p\ \mathsf{F})\end{array}$$

Example

$$\begin{array}{rcl}\pi_2\ \langle A, B\rangle & \equiv & (\lambda p.p\ (\lambda ab.b))\ \langle A, B\rangle \\ & \to_\beta & \langle A, B\rangle\ \lambda ab.b \\ & \equiv & (\lambda s.s\ A\ B)\ \lambda ab.b) \\ & \to_\beta & (\lambda ab.b)\ A\ B \\ & \to_\beta & (\lambda b.b)\ B \\ & \to_\beta & B\end{array}$$

Generalisation

- set with 3 elements : $\{\lambda a\, b\, c.a,\ \ \lambda a\, b\, c.b,\ \ \lambda a\, b\, c.c\}$
- triple of 3 elements (a, b, c) represented by the λ-term : $\lambda x.x\, a\, b\, c$.

Algebraic data types

For instance, here is a definition of binary trees in caml (with integers at the leaves)

```
type tree =
  | L of int
  | N of tree * tree
```

We can encode such a tree following these shapes:

$$\begin{array}{rcll} \mathsf{L}(\mathsf{k}) & \mapsto & \lambda a\, b.a\,[k] & \qquad ([k] \text{ encodes k assumed non-negative}) \\ \mathsf{N}(t_1, t_2) & \mapsto & \lambda ab.b\ t_1\ t_2 & \end{array}$$

Pattern-matching

Then pattern matching, as was the conditional, is just an application of the encoded term to the terms representing the various branches.

```
match t with
  | L(k)      -> f
  | N(x, y)  -> g
```

will be encoded as

$$t\ (\lambda k.f)\ (\lambda x\, y.g)$$

(where the term f may contain occurrences of the variable k, and the term g may contain occurrences of the variables x and y)

Integers

For each $n \in \mathbb{N}$ we define a λ-term $[n]$

$$\begin{array}{rcl} [0] & \equiv & \mathsf{I} \\ [n+1] & \equiv & \langle \mathsf{F}, [n] \rangle \end{array}$$

Some basic operations

$$\begin{array}{rcll} \mathsf{S} & \equiv & \lambda x.\langle \mathsf{F}, x \rangle & \text{successor} \\ \mathsf{P} & \equiv & \lambda x.x\mathsf{F} & \text{predecessor} \\ \mathsf{isZ} & \equiv & \lambda x.x\mathsf{T} & \text{zero?} \end{array}$$

Exercise: integers

Summary of the definitions

$$\begin{array}{rcl} [0] & \equiv & \mathsf{I} \\ [n+1] & \equiv & \langle \mathsf{F}, [n] \rangle \end{array} \qquad \begin{array}{rcl} \mathsf{S} & \equiv & \lambda x.\langle \mathsf{F}, x \rangle \\ \mathsf{P} & \equiv & \lambda x.x\mathsf{F} \\ \mathsf{isZ} & \equiv & \lambda x.x\mathsf{T} \end{array} \qquad \begin{array}{rcl} \langle a, b \rangle & \equiv & \lambda c.cab \\ \mathsf{T} & \equiv & \lambda ab.a \\ \mathsf{F} & \equiv & \lambda ab.b \end{array}$$

Check the following equalities

$$\begin{array}{rcl} \mathsf{S}\ [n] & =_\beta & [n+1] \\ \mathsf{P}\ [n+1] & =_\beta & [n] \\ \mathsf{P}\ [0] & =_\beta & \mathsf{F} \\ \mathsf{isZ}\ [0] & =_\beta & \mathsf{T} \\ \mathsf{isZ}\ [n+1] & =_\beta & \mathsf{F} \end{array}$$

Can you define a term add such that

$$\text{add}\ [n]\ [m] \quad = \quad [n+m]$$

Addition

We would like to write a recursive function

$$\mathrm{add}\ n\ m \quad = \quad \mathrm{if\ isZ}\ n\ \mathrm{then}\ m\ \mathrm{else\ add}\ (\mathrm{P}\ n)\ (\mathrm{S}\ m)$$

Problem: finding a λ-term add this way consists in solving an equation

λ-computability

Basic data and operations

Fixpoints

Decidability

The λ-calculus is a model of computable functions

Decidability, traditional presentation

Fixpoints for numeric functions

A fixpoint of a function f is an x such that

$$f(x) = x$$

Finding such a fixpoint f means solving the equation $x = f(x)$

Numeric functions may have various numbers of fixpoints

$x \mapsto x$	∞
$x \mapsto x + 1$	none
$x \mapsto x^2$	two (0 and 1)
$f : [0; 1] \to [0; 1]$	at least one if continuous

Fixpoints for λ-calculus

In the λ-calculus, t is a fixpoint of f if

$$f\ t\ =_\beta\ t$$

Fixpoint theorem

Any λ-term f has a fixpoint

The fixpoint theorem guarantees that, in the λ-calculus,
the equation $t =_\beta f\ t$ has always a solution

Curry's fixpoint combinator

A term that builds fixpoints

$$Y \quad \equiv \quad \lambda f.(\lambda x.f(x\,x))(\lambda x.f(x\,x))$$

First remark that

$$\begin{array}{rcl} Y\,f & \equiv & (\lambda f.(\lambda x.f(x\,x))(\lambda x.f(x\,x)))\,f \\ & \rightarrow_\beta & (\lambda x.f(x\,x))(\lambda x.f(x\,x)) \end{array}$$

The term $(\lambda x.f(x\,x))(\lambda x.f(x\,x))$, written Fix_f, is a fixpoint of f :

$$\begin{array}{rcl} \mathrm{Fix}_f & \equiv & (\lambda x.f(x\,x))(\lambda x.f(x\,x)) \\ & \rightarrow_\beta & f\ ((\lambda x.f(x\,x))(\lambda x.f(x\,x))) \\ & \equiv & f\ \mathrm{Fix}_f \end{array}$$

For any λ-term f, the term $Y\ f$ builds a fixpoint of f.

Turing's fixpoint combinator

Another term that builds fixpoints, even more directly.

$$\begin{aligned} \Theta &\equiv A\,A \\ A &\equiv \lambda x\,y.y\,(x\,x\,y) \end{aligned}$$

Checking that $f(\Theta f) =_\beta \Theta f$

$$\begin{aligned} \Theta\,f &\equiv (\lambda x\,y.y\,(x\,x\,y))\,A\,f \\ &\to_\beta (\lambda y.y\,(A\,A\,y))\,f \\ &\equiv (\lambda y.y\,(\Theta\,y))\,f \\ &\to_\beta f\,(\Theta\,f) \end{aligned}$$

For any λ-term f, the term $\Theta\,f$ *is* a fixpoint of f.
(We have $\Theta\,f \to_\beta^* f(\Theta\,f)$ but only $Y\,f =_\beta f\,(Y\,f)$)

Mutual recursion

Double fixpoint theorem

$$\forall f, g \quad \exists a, b \qquad a =_\beta f\ a\ b \quad \wedge \quad b =_\beta g\ a\ b$$

Proof: define

$$\begin{array}{rcl} d & \equiv & \Theta\ (\lambda x.\langle f\ (\pi_1 x)\ (\pi_2 x), g\ (\pi_1 x)\ (\pi_2 x)\rangle) \\ a & \equiv & \pi_1\ d \\ b & \equiv & \pi_2\ d \end{array}$$

Then

$$\begin{array}{rcl} d & \to^* & \langle f\ (\pi_1 d)\ (\pi_2 d), g\ (\pi_1 d)\ (\pi_2 d)\rangle \\ a \equiv \pi_1 d & \to^* & f\ (\pi_1 d)\ (\pi_2 d)\ \equiv\ f\ a\ b \\ b \equiv \pi_2 d & \to^* & g\ (\pi_1 d)\ (\pi_2 d)\ \equiv\ g\ a\ b \end{array}$$

This can be extended to a n-ary fixpoint, for any n.

Back to the addition

We want

$$\begin{aligned}
\mathrm{add}\; n\; m &= \mathrm{if}\;\; \mathrm{isZ}\; n \;\; \mathrm{then}\;\; m \;\; \mathrm{else}\;\; \mathrm{add}\,(\mathrm{P}\; n)\,(\mathrm{S}\; m) \\
\mathrm{add} &= \lambda n\, m.\mathrm{if}\;\; \mathrm{isZ}\; n \;\; \mathrm{then}\;\; m \;\; \mathrm{else}\;\; \mathrm{add}\,(\mathrm{P}\; n)\,(\mathrm{S}\; m) \\
\mathrm{add} &= (\lambda f\; n\, m.\mathrm{if}\;\; \mathrm{isZ}\; n \;\; \mathrm{then}\;\; m \;\; \mathrm{else}\;\; \mathrm{f}\,(\mathrm{P}\; n)\,(\mathrm{S}\; m))\;\; \mathrm{add}
\end{aligned}$$

We define add as a fixpoint with

$$\mathrm{add} \;\equiv\; \Theta\,(\lambda f\; n\, m.\mathrm{if}\;\; \mathrm{isZ}\; n \;\; \mathrm{then}\;\; m \;\; \mathrm{else}\;\; \mathrm{f}\,(\mathrm{P}\; n)\,(\mathrm{S}\; m))$$

Exercise: Fibonacci sequence

Define a λ-term representing the Fibonacci function, defined by

$$\begin{array}{rcl} f(0) & = & 0 \\ f(1) & = & 1 \\ f(n+2) & = & f(n+1) + f(n) \end{array}$$

Exercise: paradoxical fixpoint?

We said that:

- $f : x \mapsto x + 1$ is function with zero fixpoint
- $F = \lambda x.\mathrm{S}\ x$ is a λ-term, and therefore it has a fixpoint

How can these two facts both be true?

Exercise: Church's integers (iterators)

Alternative representation for $[n]$

$$[n] \equiv \lambda f\ x. f^n\ x$$

Idea: $[n]$ *takes as argument a function* f *and an initial value* x *and returns a function that iterates n times* f *from* x

Show that $\lambda n\, f\ x. f f\ (n\, f\ x)$ represents the successor function

Find terms representing addition, multiplication, and predecessor

λ-computability

Basic data and operations

Fixpoints

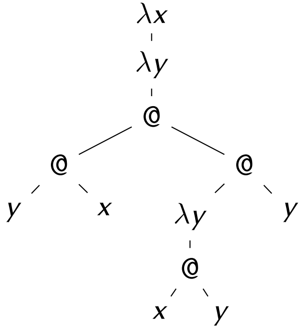
Decidability

The λ-calculus is a model of computable functions

Decidability, traditional presentation

de Bruijn notation: use numbers instead of variable names

$\lambda x.\lambda y.(y\ x\ ((\lambda y.x\ y)\ y))$



Replace each variable occurrence with the number of λ between the occurrence and its binder

$$\lambda.\lambda.0\ 1\ ((\lambda.2\,0)\ 0)$$

What we gain ? the need for variable renamings disappears.
The syntax of terms is easier to represent as a λ-encoded data structure.

Translations between named and nameless variables

For any named closed term t, write $[\![t]\!]$ its nameless version. Generalization to term with free variables: let ℓ be a list of variable names that contains all the free variables of t, define $[\![t]\!]_\ell$ the translation where each free variable x of t is associated to the index at which x appears in t.

$$\begin{array}{rcl} [\![x]\!]_\ell & = & \mathrm{index\_of}(x, \ell) \\ [\![t\ u]\!]_\ell & = & [\![t]\!]_\ell\ [\![u]\!]_\ell \\ [\![\lambda x.t]\!]_\ell & = & \lambda.[\![t]\!]_{x:\ell} \end{array}$$

with index_of a function that returns the index at which the name x appears in the list ℓ, starting from 0.

Translations between named and nameless variables

Reverse: for any nameless closed term t, write $(\!|t|\!)$ its named version. Generalization to term with free variables: let ℓ be a list of variable names that is long enough to account for every indices in t, define $(\!|t|\!)_\ell$ the translation where each free index of t is associated to the element at corresponding index of ℓ.

$$\begin{array}{rcll} (\!|k|\!)_\ell & = & \mathsf{nth}(k,\ell) & \\ (\!|t\ u|\!)_\ell & = & (\!|t|\!)_\ell\ (\!|u|\!)_\ell & \\ (\!|\lambda.t|\!)_\ell & = & \lambda x.(\!|t|\!)_{x:\ell} & \text{for } x \text{ a fresh variable name} \end{array}$$

with nth a function that returns the element at index k in the list ℓ.

Encoding the abstract syntax of nameless λ-terms.

Nameless terms can be represented with the following three constructors.

```
type term =
  | Var of int
  | App of term * term
  | Abs of term
```

Representation of such a data structure using λ-terms:

$$\begin{array}{rcl} [k] & = & \lambda abc.a\,[k] \\ [t\ u] & = & \lambda abc.b\,[t]\,[u] \\ [\lambda.t] & = & \lambda abc.c\,[t] \end{array}$$

Note: $[k]$ on the left of the first equation is the encoding of a λ-term made of the de Bruijn index k, defined by the equation, whereas $[k]$ on the right of the same equation is the encoding of the naturel number k, as proposed at the beginning of the chapter.

Encoding the abstract syntax of named λ-terms.

One obtains an encoding of usual, named λ-terms by composing the translation to nameless representation with the previous translation. Here is a set of combined equations:

$$\begin{array}{rcl} [x]_\ell & = & \lambda abc.a\,[\text{index\_of}(x,\ell)] \\ [t\ u]_\ell & = & \lambda abc.b\,[t]_\ell\ [u]_\ell \\ [\lambda x.t]_\ell & = & \lambda abc.c\ [t]_{x:\ell} \end{array}$$

(again, $[\text{index\_of}(x,\ell)]$ is the encoding of a natural number as defined at the beginning of the chapter)

Self-interpreter

Using the previous term representation, one can define an interpreter of the λ-calculus, in the λ-calculus.
Such a function can be called a *self-interpreter*, and also corresponds to the concept of *universal machine* that you will hear of again in the computability course.
This interpreter is a term e such that for any term t and any list ℓ we have

$$e\ [t]_\ell\ [\ell] =_\beta t$$

(this assumes that the list ℓ can also be encoded as a λ-term $[\ell]$, which is left as an exercise)

Self-interpreter

For such an interpreter, we want the following equations:

$$\begin{array}{rclcl}
e\,[x]_\ell\;[\ell] & = & e\,(\lambda abc.a\,[k])\;[\ell] & = & \mathrm{nth}(k,[\ell]) \\
e\,[t\;u]_\ell\;[\ell] & = & e\,(\lambda abc.b\,b\,[t]_\ell\;[u]_\ell)\;[\ell] & = & (e\,[t]_\ell\;\ell)\;(e\,[t]_\ell\;[\ell]) \\
e\,[\lambda x.t]_\ell\;[\ell] & = & e\,(\lambda abc.c\,[t]_{x:\ell})\;[\ell] & = & \lambda x.(e\,[t]_{x:\ell}\;[x:\ell])
\end{array}$$

Thus we propose the following term:

$$\begin{array}{rcl}
e & = & Y\,(\lambda e.\lambda t.\lambda \ell.\;t\,(\lambda k.\mathrm{nth}(k,\ell)) \\
 & & \qquad\qquad\qquad (\lambda tu.(e\;t\;\ell)\;(e\;u\;\ell)) \\
 & & \qquad\qquad\qquad (\lambda t.\lambda x.e\;t\;(x:\ell)))
\end{array}$$

Correctness of the self-interpreter

Assuming that lists of names ℓ can be encoded as λ-terms as well as the two functions index_of and nth, we prove that for any term t and any list ℓ containing (at least) the free variables of t :

$$e\ [t]_\ell\ [\ell] =_\beta t$$

Write $e = Y\ e'$. We have in one step

$$e \quad = \quad Y\ e' \quad \rightarrow \quad (\lambda x.e'(x\,x))(\lambda x.e'(x\,x)) \quad \equiv \quad e''$$

where the obtained term e'' is the fixpoint of e' produced by Y.

Correctness of the self-interpreter

Since all encodings share a common structure, first remark that

$$\begin{array}{rcl} e\ [t]_\ell\ [\ell] & \equiv & Y\ e'\ [t]_\ell\ [\ell] \\ & \to & (\lambda x.e'(xx))(\lambda x.e'(x\,x))\ [t]_\ell\ [\ell] \\ & \equiv & e''\ [t]_\ell\ [\ell] \\ & \to & e'\ e''\ [t]_\ell\ [\ell] \\ & \to^3 & [t]_\ell\ e_1\ e_2\ e_3 \end{array}$$

where

$$\begin{array}{rcl} e_1 & = & \lambda k.\mathrm{nth}(k, [\ell]) \\ e_2 & = & \lambda t\,u.(e''\ t\ [\ell])\ (e''\ u\ [\ell]) \\ e_3 & = & \lambda t.\lambda x.e''\ t\ (x : [\ell]) \end{array}$$

Correctness of the self-interpreter

Now prove the result by induction on t:

- Case of a variable x (assumed in ℓ):

$$\begin{array}{rcl} e\,[x]_\ell\,[\ell] & \rightarrow & [x]_\ell\ e_1\ e_2\ e_3 \\ & = & (\lambda abc.a\,[\text{index\_of}(x,\ell)])\ e_1\ e_2\ e_3 \\ & \rightarrow^3 & e_1\,[\text{index\_of}(x,\ell)] \\ & = & (\lambda k.\text{nth}(k,[\ell]))\,[\text{index\_of}(x,\ell)] \\ & = & \text{nth}([\text{index\_of}(x,\ell)],[\ell]) \end{array}$$

  The specifications of nth and index_of indeed require that nth([index_of(x,ℓ)], [ℓ]) is equal to x (when x is in ℓ).

- Case of an application $t\ u$:

$$\begin{array}{rcll} e\,[t\ u]_\ell\,[\ell] & \rightarrow & [t\ u]_\ell\ e_1\ e_2\ e_3 & \\ & = & (\lambda abc.b\,[t]_\ell\,[u]_\ell)\ e_1\ e_2\ e_3 & \\ & \rightarrow^3 & e_2\,[t]_\ell\,[u]_\ell & \\ & = & (\lambda t\,u.(e''\ t\,[\ell])\,(e''\ u\,[\ell]))\,[t]_\ell\,[u]_\ell & \\ & \rightarrow^2 & (e''\,[t]_\ell\,[\ell])\,(e''\,[u]_\ell\,[\ell]) & \\ & =_\beta & t\ u & \text{by induction hy} \end{array}$$

Correctness of the self-interpreter

- Case of an abstraction $\lambda x.t$:

$$\begin{array}{rcll}
e\,[\lambda x.t]_\ell\,[\ell] & \to & [\lambda x.t]_\ell\ e_1\ e_2\ e_3 & \\
 & = & (\lambda a\,b\,c.c\,[t]_{x:\ell})\ e_1\ e_2\ e_3 & \\
 & \to^3 & e_3\,[t]_{x:\ell} & \\
 & = & (\lambda t.\lambda x.e''\ t\ (x : [\ell]))\ [t]_{x:\ell} & \\
 & \to & \lambda x.e''\,[t]_{x:\ell}\,[x : \ell] & \\
 & =_\beta & \lambda x.t & \text{by induction hyp.}
\end{array}$$

Second fixpoint theorem

$$\forall f \; \exists t \quad f\ [t] =_\beta t$$

Proof of the second fixpoint theorem

First remark that one could write two terms A and N such that

$$\begin{array}{rcl} \mathsf{A}\ [t]\ [u] & =_\beta & [t\ u] \\ \mathsf{N}\ [t] & =_\beta & [[t]] \end{array}$$

(A is simply $\lambda tu.\lambda abc.b\ t\ u$, whereas N is defined as the fixpoint of a function defined by pattern matching on the representation $[t]$ of t)
Then define

$$\begin{array}{rcl} w & \equiv & \lambda x.f\ (\mathsf{A}\ x\ (\mathsf{N}\ x)) \\ z & \equiv & w\ [w] \end{array}$$

Then z is a fixpoint for f.

$$\begin{array}{rcllcl} z \equiv w\ [w] & =_\beta & f\ (\mathsf{A}\ [w]\ (\mathsf{N}\ [w])) & & & \\ & =_\beta & f\ (\mathsf{A}\ [w]\ [[w]]) & & & \\ & =_\beta & f\ [w\ [w]] & & \equiv & f\ [z] \end{array}$$

Scott's undecidability theorem

Theorem

1. any two non-empty sets $A, B \subseteq \Lambda$ closed by β-equality are not effectively separable
2. no non-trivial set $A \subseteq \Lambda$ closed by β-equality can be effectively characterized

Scott's undecidability theorem

Definitions

- E is closed by β-equality if $\forall x, y \in \Lambda\ x \in E \wedge x =_\beta y \implies y \in E$
- E is non-trivial if there are $x \in E$ and $y \notin E$
- A and B are effectively separable if there is an effectively characterized set C such that $t \in A \implies t \in C$ and $t \in B \implies t \notin C$
- C is effectively characterized if there is a λ-term f such that $f\ t =_\beta \mathsf{T}$ for any $t \in C$ and $f\ t =_\beta \mathsf{F}$ for any $t \notin C$

(note: in the definition of "effectively characterized" it is of critical importance that the application of the λ-term f to *any* λ-term t is normalizable)

Proof of Scott's theorem

Any two non-empty sets $A, B \subseteq \Lambda$ closed by β-equality are not effectively separable.

Assume there is a separating set C such that $A \subseteq C$ and $B \cap C = \emptyset$, characterized by a λ-term f such that

$$\begin{array}{lcl} t \in C & \Longrightarrow & f\ [t] =_\beta \mathsf{T} \\ t \notin C & \Longrightarrow & f\ [t] =_\beta \mathsf{F} \end{array}$$

Since A and B are not empty, we have two terms $a \in A$ and $b \in B$. Define

$$g \quad \equiv \quad \lambda x.\text{if } f\ x \text{ then } b \text{ else } a$$

Then

$$\begin{array}{lcl} t \in C & \Longrightarrow & g\ [t] =_\beta b \\ t \notin C & \Longrightarrow & g\ [t] =_\beta a \end{array}$$

Proof of Scott's theorem

From the second fixpoint theorem, there is z such that $g\ [z] = z$

$$\begin{array}{lcccc} z \in C & \Longrightarrow & z =_\beta g\ [z] =_\beta b \in B & \Longrightarrow & z \notin C \\ z \notin C & \Longrightarrow & z =_\beta g\ [z] =_\beta a \in A & \Longrightarrow & z \in C \end{array}$$

Contradiction!

Undecidability of β-equality

No algorithm can decide whether two arbitrary λ-terms are β-equal

Assume f is a λ-term such that, for any a and b,
$f\ [a]\ [b]$ equals to $[1]$ if $a =_\beta b$ and to $[0]$ otherwise

Define $A = \{x \mid x =_\beta a\}$

- by definition, A is closed by β-equality
- A is not empty, since it contains a
- $\Lambda \setminus A$ is not empty, because:
    - if a has a normal form, then $\Omega \notin A$
    - if a has no normal form, then $\lambda x.x \notin A$

By Scott's theorem, the set A is not recursive

On the other hand, $f\ [a]$ computes the characteristic function of A
Contradiction.

Exercise: halting problem for the λ-calculus

No algorithm can decide whether an arbitrary λ-term has a normal form

Undecidability of the optimal strategy

Strategy: function $F : \Lambda \to \Lambda$ such that

$$\forall t \in \Lambda \quad t \to_\beta F(t)$$

Optimal strategy: strategy that always picks a shortest path to the normal form (if there is a normal form)

There is no computable optimal strategy

Undecidability of the optimal strategy: idea

Consider the set

$$t_n \equiv (\lambda x.x\mathsf{E}x)\,(\lambda y.y[n](\mathsf{I}\mathsf{I}))$$

of λ-terms, where E enumerates λ-terms with at most one free variable a

Assuming E is already in normal form, for each n we have to choose between:

- reducing $t_n \to_\beta (\lambda y.y[n](\mathsf{I}\mathsf{I}))\;\mathsf{E}\;(\lambda y.y[n](\mathsf{I}\mathsf{I}))$
- reducing $t_n \to_\beta (\lambda x.x\mathsf{E}x)\,(\lambda y.y[n]\mathsf{I})$

However, the best choice differs depending on the normal form of E $[n]$

Optimal strategy: first case

If $\mathsf{E}\ [n] \to_\beta^* \lambda xyz.z$ in k steps then

$$\begin{aligned}(\lambda y.y[n](\mathsf{II}))\ \mathsf{E}\ (\lambda y.y[n](\mathsf{II})) &\to_\beta \mathsf{E}\ [n]\ (\mathsf{II})\ (\lambda y.y[n](\mathsf{II}))\\ &\to_\beta^* (\lambda xyz.z)\ (\mathsf{II})\ (\lambda y.y[n](\mathsf{II}))\\ &\to_\beta^2 \lambda z.z\end{aligned}$$

optimally in $k+3$ steps and

$$\begin{aligned}(\lambda x.x\mathsf{E}x)\ (\lambda y.y[n]\mathsf{I}) &\to_\beta (\lambda y.y[n]\mathsf{I})\ \mathsf{E}\ (\lambda y.y[n]\mathsf{I})\\ &\to_\beta \mathsf{E}\ [n]\ \mathsf{I}\ (\lambda y.y[n]\mathsf{I})\\ &\to_\beta^* (\lambda xyz.z)\ \mathsf{I}\ (\lambda y.y[n]\mathsf{I})\\ &\to_\beta^2 \lambda z.z\end{aligned}$$

optimally in $k+4$ steps

Optimal strategy: second case

If $\mathsf{E}\ [n] \to_\beta^* a$ in k steps then

$$\begin{array}{rcl} (\lambda y.y[n](\mathsf{II}))\ \mathsf{E}\ (\lambda y.y[n](\mathsf{II})) & \to_\beta & \mathsf{E}\ [n]\ (\mathsf{II})\ (\lambda y.y[n](\mathsf{II})) \\ & \to_\beta^* & a\ (\mathsf{II})\ (\lambda y.y[n](\mathsf{II})) \\ & \to_\beta^2 & a\ \mathsf{I}\ (\lambda y.y[n]\mathsf{I}) \end{array}$$

optimally in $k + 3$ steps and

$$\begin{array}{rcl} (\lambda x.x\mathsf{E}x)\ (\lambda y.y[n]\mathsf{I}) & \to_\beta & (\lambda y.y[n]\mathsf{I})\ \mathsf{E}\ (\lambda y.y[n]\mathsf{I}) \\ & \to_\beta & \mathsf{E}\ [n]\ \mathsf{I}\ (\lambda y.y[n]\mathsf{I}) \\ & \to_\beta^* & a\ \mathsf{I}\ (\lambda y.y[n]\mathsf{I}) \end{array}$$

optimally in $k + 2$ steps

Optimal strategy: conclusion

$$t_n \quad \equiv \quad (\lambda x.x\mathsf{E}x)\ (\lambda y.y[n](\mathsf{II}))$$

If F is an optimal strategy, then

- if $\mathsf{E}\ [n] \rightarrow_\beta^* \lambda xyz.z$ then $F(t_n) = (\lambda y.y[n](\mathsf{II}))\ \mathsf{E}\ (\lambda y.y[n](\mathsf{II}))$, and
- if $\mathsf{E}\ [n] \rightarrow_\beta^* a$ then $F(t_n) = (\lambda x.x\mathsf{E}x)\ (\lambda y.y[n]\mathsf{I})$

An optimal strategy thus separates

$$\{n \mid \mathsf{E}\ [n] \rightarrow_\beta^* \lambda xyz.z\} \qquad \text{and} \qquad \{n \mid \mathsf{E}\ [n] \rightarrow_\beta^* a\}$$

However, these two sets are not recursively separable,
since by Scott's theorem

$$\{t \mid t \rightarrow_\beta^* \lambda xyz.z\} \qquad \text{and} \qquad \{t \mid t \rightarrow_\beta^* a\}$$

are not recursively separable.

λ-computability

Basic data and operations

Fixpoints

Decidability

The λ-calculus is a model of computable functions

Decidability, traditional presentation

Definability

A mathematical function $\varphi : \mathbb{N}^p \to \mathbb{N}$ is λ-*definable* if there is a λ-term $f \in \Lambda$ such that

$$\forall n_1, \ldots, n_p \in \mathbb{N}, \quad f[n_1] \ldots [n_p] =_\beta [\varphi(n_1, \ldots, n_p)]$$

By Church-Rosser property, we could also have given the condition

$$\forall n_1, \ldots, n_p \in \mathbb{N}, \quad f[n_1] \ldots [n_p] \to_\beta^* [\varphi(n_1, \ldots, n_p)]$$

Property: the λ*-definable functions are exactly the recursive functions*

Initial recursive functions

Zero $Z(n) = 0$

- $\mathsf{Z} = \lambda x.[0]$

Successor $S(n) = n + 1$

- $\mathsf{S} = \lambda x.\langle \mathsf{F}, x\rangle$

Projection $U_i^p(n_0, \ldots, n_p) = n_i$ with $0 \leqslant i \leqslant p$

- $\mathsf{U}_i^p = \lambda x_0 \ldots x_p.x_i$

Composition of recursive functions

If F, G_1, ..., G_m are recursive then the function H defined by

$$H(\vec{n}) = F(G_1(\vec{n}), \ldots, G_m(\vec{n}))$$

is recursive

Assume F, G_1, ..., G_m are defined by f, g_1, ..., g_m
then H can be defined by

$$h \equiv \lambda \vec{x}.F\ (G_1\ \vec{x}) \ldots (G_m\ \vec{x})$$

Primitive recursion

If F and G are recursive
then the function H defined by

$$\begin{array}{rcl} H(0,\vec{n}) & = & F(\vec{n}) \\ H(k+1,\vec{n}) & = & G(H(k,\vec{n}),k,\vec{n}) \end{array}$$

is recursive

Assume F and G are defined by f and g,
we are looking for an h such that

$$h \quad \equiv \quad \lambda x\vec{y}.\text{if isZ } x \text{ then } f\ \vec{y} \text{ else } g\ (h\ (\mathsf{P}\,x)\ \vec{y})\ (\mathsf{P}\,x)\ \vec{y}$$

Fixpoint theorem: such a term h exists

Minimisation

If F is recursive and is such that

$$\forall \vec{n}\; \exists m\; F(\vec{n}, m) = 0$$

then the function M defined by

$$M(\vec{n}) \quad = \quad \text{the smallest } m \in \mathbb{N} \text{ such that } F(\vec{n}, m) = 0$$

is recursive

Assume F is defined by f, then define

$$m \quad \equiv \quad \lambda \vec{x}.(\Theta\ (\lambda h y.\mathsf{if\ isZ}\ (f\ \vec{x}\ y)\ \mathsf{then}\ y\ \mathsf{else}\ h(\mathsf{S}\ y))\ [0])$$

Summary

We encoded in the λ-calculus:

- the initial functions Z, S and U_i^p
- function composition
- primitive recursion
- minimisation

Therefore, any recursive function is λ-definable

The λ-calculus is Turing-complete

λ-computability

Basic data and operations

Fixpoints

Decidability

The λ-calculus is a model of computable functions

Decidability, traditional presentation

Encoding λ-terms using numbers

Assume a (computable and) injective function $\varphi : \mathbb{N}^2 \to \mathbb{N}$, for instance $\varphi(x, y) \equiv 2^x(2y + 1) - 1$

Assign numbers to all variables: $\{x_0, x_1, x_2, \dots\}$

We deduce a function $\sharp : \Lambda \to \mathbb{N}$ assigning a unique number to each λ-term

$$\begin{array}{rcl} \sharp x_i & = & \varphi(0,\ i) \\ \sharp(t\ u) & = & \varphi(1,\ \varphi(\sharp t,\ \sharp u)) \\ \sharp(\lambda x_i.t) & = & \varphi(2,\ \varphi(i,\ \sharp t)) \end{array}$$

Encoding of a λ-term t: the λ-term t' representing the number n representing the encoded λ-term t

$$[t] \quad \equiv \quad [\sharp t]$$

Remark: this is a new encoding, thus all encoding-dependent theorems have to be proved again.

Enumeration theorem (admitted)

There is a λ-term E such that for any closed λ-term t, $\mathsf{E}\ \lceil t \rceil \to_{\beta}^{*} t$

This is the equivalent of the self-interpreter in the previous presentation. The proof however is far more technical.

Proof of the second fixpoint theorem

The functions φ_A and φ_N defined by

$$\begin{array}{rcl} \varphi_A(\sharp t, \sharp u) & = & \sharp(t\ u) \\ \varphi_N(\sharp t) & = & \sharp[t] \end{array}$$

are recursive. They are thus defined by λ-terms A and N such that

$$\begin{array}{rcl} \mathsf{A}\ [t]\ [u] & =_\beta & [t\ u] \\ \mathsf{N}\ [t] & =_\beta & [[t]] \end{array}$$

Define

$$\begin{array}{rcl} w & \equiv & \lambda x.f\ (\mathsf{A}\ x\ (\mathsf{N}\ x)) \\ z & \equiv & w\ [w] \end{array}$$

Then z is a fixpoint for f.

$$\begin{array}{rcll} z \equiv w\ [w] & =_\beta & f\ (\mathsf{A}\ [w]\ (\mathsf{N}\ [w])) & \\ & =_\beta & f\ (\mathsf{A}\ [w]\ [[w]]) & \\ & =_\beta & f\ [w\ [w]] & \equiv\ \ f\ [z] \end{array}$$

Scott's undecidability theorem (stated using general vocabulary of recursive functions)

Theorem

1. any two non-empty sets $A, B \subseteq \Lambda$ closed by β-equality are not recursively separable;
2. any non-trivial set $A \subseteq \Lambda$ closed by β-equality is not recursive.

Definitions

- E is closed by β-equality if $\forall x, y \in \Lambda\ x \in E \wedge x =_\beta y \implies y \in E$
- E is non-trivial if there are $x \in E$ and $y \notin E$
- A and B are recursively separable if there is a recursive set C such that $A \subseteq C$ and $B \cap C = \emptyset$
- C is recursive if its characteristic function is recursive

Proof of Scott's theorem

Any two non-empty sets $A, B \subseteq \Lambda$ closed by β-equality are not recursively separable.

Assume there is a recursive set C such that $A \subseteq C$ and $B \cap C = \emptyset$. Its characteristic function is realized by a λ-term f such that

$$\begin{array}{lcl} t \in C & \Longrightarrow & f\ [t] =_\beta [1] \\ t \notin C & \Longrightarrow & f\ [t] =_\beta [0] \end{array}$$

Since A and B are not empty, we have two terms $a \in A$ and $b \in B$. Define

$$g \quad \equiv \quad \lambda x.\text{if isZ } (f\ x) \text{ then } b \text{ else } a$$

Then

$$\begin{array}{lcl} t \in C & \Longrightarrow & g\,[t] =_\beta b \\ t \notin C & \Longrightarrow & g\,[t] =_\beta a \end{array}$$

Proof of Scott's theorem

From the second fixpoint theorem, there is z such that $g\,[z] = z$

$$\begin{array}{lcccc} z \in C & \Longrightarrow & z =_\beta g\,[z] =_\beta b \in B & \Longrightarrow & z \notin C \\ z \notin C & \Longrightarrow & z =_\beta g\,[z] =_\beta a \in A & \Longrightarrow & z \in C \end{array}$$

Contradiction!

Undecidability results...

are proved exactly as in the previous section, now that Scott's theorem is established for this other representation of λ-terms.

Summary

- systematic way to encode data-structures in λ-calculus
  - identify a data with the function that uses it for computing
- existence of fixpoints combinators (equivalence or reduction)
- λ-calculus can represent any recursive function on natural numbers
- Scott's indecidability theorem : non-trivial sets that are closed by β-equality are not recursive

Homework

1. Prove that there exists no λ-term h such that $h\,[t] = \mathsf{T}$ for any $t \in \Lambda$ with a normal form and $h\,[t] = \mathsf{F}$ for any $t \in \Lambda$ with no normal form.
2. Using the encoding of algebraic datatypes, and one of the already defined encodings of numbers, propose an encoding of lists, and of the nth function.
3. In your encoding, prove that $\mathrm{nth}\; k\; \ell = \mathrm{nth}\; (k+1)\; (t : \ell)$.