Non-normalizing terms

Some λ-terms don't have a normal form

$$\omega := (\lambda x.x\,x)(\lambda x.x\,x)$$

$$\begin{aligned}\omega &\rightsquigarrow (x\,x)[x := \lambda x.x\,x]\\ &= (\lambda x.x\,x)(\lambda x.x\,x)\\ &= \omega\end{aligned}$$

it reduces to itself forever

Terms may even grow bigger and bigger:

$$\omega_3 := (\lambda x.x\,x\,x)(\lambda x.x\,x\,x)$$

Fixed Points

The λ-calculus can encode all recursively computable functions.

The trick to do it is the **fixed point combinator**

$$Y := \lambda f.(\lambda x.f(x\,x))(\lambda x.f(x\,x))$$

(it is inspired by the never-ending term ω)

$$Yf \rightsquigarrow^{*} f(Yf)$$

not exactly (try it) but essentially

The combinator Y is used to implement general recursion: iterate a function f until we get a result

(With Church numerals we can iterate a function a fixed number of times)

Example: Factorial Function

Recursive definition:

fact n = if (iszero n)
  then 1
  else n • (fact (n-1))

We define a one-step operator

Use a variable f for recursive call

hfact := λf. λn.
  if (iszero n)
    $\underline{1}$
    (mult n (f (pred n)))

The full factorial is the fixed point of the one-step operation

fact := Y hfact

Try to apply it to a numeric value to see what happens.

(fact can be defined without Y, using Church numerals. Try.)

Functions that are not structurally recursive, may not terminate

Example:

$$\text{hail } n = \begin{cases} 0 & \text{if } n=0 \text{ or } n=1 \\ \text{hail}(n/2)+1 & \text{if } n>1 \text{ even} \\ \text{hail}(3n+1)+1 & \text{if } n>1 \text{ odd} \end{cases}$$

Nobody knows if this function terminates for every n

(Collatz Conjecture)

"Completely out of reach of present day mathematics"

It has a recursive call to a larger argument: $3n+1$

it can't be programmed using simple recursion

But we can do it with a higher-order step function and the Y combinator

$$\text{hhail} := \lambda f.\lambda n. \begin{cases} 0 & \text{if } n\leq 1 \\ f(n/2)+1 & \text{if } n>1 \text{ even} \\ f(3n+1)+1 & \text{if } n>1 \text{ odd} \end{cases}$$

(Try to write it formally in λ-calcul

$$\text{hail} := Y \text{ hhail}$$

Infinite Data Structures

Streams = Infinite Sequences

$$a_0 \triangleleft a_1 \triangleleft a_2 \triangleleft \cdots$$

We suggested the representation

$$\lambda f.\, f a_0 (f a_1 (f a_2 \cdots))$$

But it's a bit difficult to define the tail function:

$$\mathrm{tail}\,(a_0 \triangleleft a_1 \triangleleft \cdots) = a_1 \triangleleft a_2 \triangleleft \cdots$$

A more convenient representation:

$$\langle a_0, \langle a_1, \langle a_2, \cdots \rangle\rangle\rangle$$

$$a \triangleleft \sigma := \langle a, \sigma \rangle = \lambda x.\, x\, a\, \sigma$$

$$\mathrm{head}\ \sigma := \mathrm{first}\ \sigma = \sigma\ \mathrm{true}$$

$$\mathrm{tail}\ \sigma := \mathrm{second}\ \sigma = \sigma\ \mathrm{false}$$

Try to define conversion combinators between the two representations

Example:

A term that encodes the stream of natural numbers

$$0 \triangleleft 1 \triangleleft 2 \triangleleft 3 \triangleleft \cdots$$

First define the stream starting from a given number

$$\mathrm{from}\ n = n \triangleleft (n+1) \triangleleft (n+2) \triangleleft \cdots$$

Define a term from such that

$$\mathrm{from}\ n \rightsquigarrow^{*} \langle n, \mathrm{from}\ (n+1) \rangle$$

We can do it as a fixed point

$$\mathrm{hfrom} := \lambda f.\ \lambda n.\ \langle n, f\ (\mathrm{succ}\ n) \rangle$$

$$\mathrm{from} := Y\ \mathrm{hfrom}$$

Finally, the stream of natural numbers:

$$\mathrm{nats} := \mathrm{from}\ \overline{0}$$