λ-Definable functions

Definition. $f \in \mathbb{N}^n \rightharpoonup \mathbb{N}$ is λ-definable if there is a closed λ-term F that represents it: for all $(x_1, \ldots, x_n) \in \mathbb{N}^n$ and $y \in \mathbb{N}$

- if $f(x_1, \ldots, x_n) = y$, then $F\,\underline{x_1} \cdots \underline{x_n} =_\beta \underline{y}$
- if $f(x_1, \ldots, x_n)\uparrow$, then $F\,\underline{x_1} \cdots \underline{x_n}$ has no β-nf.

This condition can make it quite tricky to find a λ-term representing a non-total function.

For now, we concentrate on total functions. First, let us see why the elements of **PRIM** (primitive recursive functions) are λ-definable.

Representing primitive recursion

If $f \in \mathbb{N}^n \to \mathbb{N}$ is represented by a λ-term F and

$g \in \mathbb{N}^{n+2} \to \mathbb{N}$ is represented by a λ-term G,

we want to show λ-definability of the unique
$h \in \mathbb{N}^{n+1} \to \mathbb{N}$ satisfying

$$\begin{cases} h(\vec{a}, 0) & = f(\vec{a}) \\ h(\vec{a}, a+1) & = g(\vec{a}, a, h(\vec{a}, a)) \end{cases}$$

or equivalently

$$\begin{aligned} h(\vec{a}, a) = & \; \textit{if } a = 0 \textit{ then } f(\vec{a}) \\ & \; \textit{else } g(\vec{a}, a-1, h(\vec{a}, a-1)) \end{aligned}$$

Representing primitive recursion

If $f \in \mathbb{N}^n \to \mathbb{N}$ is represented by a λ-term F and

$g \in \mathbb{N}^{n+2} \to \mathbb{N}$ is represented by a λ-term G,

we want to show λ-definability of the unique
$h \in \mathbb{N}^{n+1} \to \mathbb{N}$ satisfying $\boxed{h = \Phi_{f,g}(h)}$

where $\Phi_{f,g} \in (\mathbb{N}^{n+1} \to \mathbb{N}) \to (\mathbb{N}^{n+1} \to \mathbb{N})$ is given by

$$\begin{aligned}\Phi_{f,g}(h)(\vec{a}, a) \triangleq\ & \mathit{if}\ a = 0\ \mathit{then}\ f(\vec{a}) \\ & \mathit{else}\ g(\vec{a}, a - 1, h(\vec{a}, a - 1))\end{aligned}$$

Representing primitive recursion

If $f \in \mathbb{N}^n \to \mathbb{N}$ is represented by a λ-term F and

$g \in \mathbb{N}^{n+2} \to \mathbb{N}$ is represented by a λ-term G,

we want to show λ-definability of the unique

$h \in \mathbb{N}^{n+1} \to \mathbb{N}$ satisfying $\boxed{h = \Phi_{f,g}(h)}$

where $\Phi_{f,g} \in (\mathbb{N}^{n+1} \to \mathbb{N}) \to (\mathbb{N}^{n+1} \to \mathbb{N})$ is given by...

Strategy:

- show that $\Phi_{f,g}$ is λ-definable;
- show that we can solve fixed point equations $\boxed{X = M\,X}$ up to β-conversion in the λ-calculus.

Origins of Y

Naive set theory	λ calculus
Russell set: $R \triangleq \{x \mid \neg(x \in x)\}$	$R \triangleq \lambda x.\, \text{not}\,(x\,x)$

$\text{not} \triangleq \lambda b.\, \text{If}\ b\ \text{False}\ \text{True}$

Origins of Y

Naive set theory	λ calculus
Russell set: $R \triangleq \{x \mid \neg(x \in x)\}$	$R \triangleq \lambda x.\ \text{not}\,(x\,x)$
Russell's Paradox: $R \in R \iff \neg(R \in R)$	$R\,R =_\beta \text{not}\,(R\,R)$

Origins of Y

Naive set theory

Russell set :

$$R \triangleq \{x \mid \neg(x \in x)\}$$

Russell's Paradox :

$$R \in R \iff \neg(R \in R)$$

λ calculus

$$R \triangleq \lambda x.\, \mathrm{not}(x\,x)$$

$$R\,R =_\beta \mathrm{not}(R\,R)$$

$$Y\,\mathrm{not} =_\beta R\,R = (\lambda x.\, \mathrm{not}(x\,x))(\lambda x.\, \mathrm{not}(x\,x))$$

Origins of Y

Naive set theory	λ calculus
Russell set: $R \triangleq \{x \mid \neg(x \in x)\}$	$R \triangleq \lambda x.\ \mathrm{not}(x\,x)$
Russell's Paradox: $R \in R \iff \neg(R \in R)$	$R\,R =_\beta \mathrm{not}(R\,R)$

$$Y\,\mathrm{not} =_\beta R\,R = (\lambda x.\ \mathrm{not}(x\,x))(\lambda x.\ \mathrm{not}(x\,x))$$

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

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

Curry's fixed point combinator $\mathbf{Y}$

$$\mathbf{Y} \triangleq \lambda f.\,(\lambda x.\,f(x\,x))(\lambda x.\,f(x\,x))$$

satisfies $\mathbf{Y}\,M \;\rightarrow\; (\lambda x.\,M(x\,x))(\lambda x.\,M(x\,x))$

Curry's fixed point combinator $\mathbf{Y}$

$$\mathbf{Y} \triangleq \lambda f.\,(\lambda x.\, f(x\,x))(\lambda x.\, f(x\,x))$$

satisfies $\mathbf{Y}\,M \;\to\; (\lambda x.\, M(x\,x))(\lambda x.\, M(x\,x))$
$\to\; M((\lambda x.\, M(x\,x))(\lambda x.\, M(x\,x)))$

hence $\mathbf{Y}\,M \twoheadrightarrow M((\lambda x.\, M(x\,x))(\lambda x.\, M(x\,x))) \twoheadleftarrow M(\mathbf{Y}\,M)$.

So for all λ-terms M we have

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

Turing's fixed point combinator

$$\Theta \triangleq A\,A$$

where $A \triangleq \lambda x y.\, y\,(x x y)$

Turing's fixed point combinator

$$\Theta \triangleq A\,A$$

where $A \triangleq \lambda xy.\, y(xxy)$

$$\Theta M = AAM = (\lambda xy.\, y(xxy))\,A\,M$$

Turing's fixed point combinator

$$\Theta \triangleq A\,A$$

where $A \triangleq \lambda xy.\,y(xxy)$

$$\Theta M = AAM = (\lambda xy.\,y(xxy))\,A\,M$$

$$\twoheadrightarrow M(AAM)$$

$$= M(\Theta M)$$

Representing primitive recursion

If $f \in \mathbb{N}^n \to \mathbb{N}$ is represented by a λ-term F and $g \in \mathbb{N}^{n+2} \to \mathbb{N}$ is represented by a λ-term G,

we want to show λ-definability of the unique $h \in \mathbb{N}^{n+1} \to \mathbb{N}$ satisfying $\boxed{h = \Phi_{f,g}(h)}$

where $\Phi_{f,g} \in (\mathbb{N}^{n+1} \to \mathbb{N}) \to (\mathbb{N}^{n+1} \to \mathbb{N})$ is given by

$$\begin{aligned}\Phi_{f,g}(h)(\vec{a}, a) \triangleq\ & \mathit{if}\ a = 0\ \mathit{then}\ f(\vec{a}) \\ & \mathit{else}\ g(\vec{a}, a-1, h(\vec{a}, a-1))\end{aligned}$$

We now know that h can be represented by

$Y(\lambda z \vec{x} x.\, \mathbf{If}(\mathbf{Eq}_0\, x)(F\, \vec{x})(G\, \vec{x}\, (\mathbf{Pred}\, x)(z\, \vec{x}\, (\mathbf{Pred}\, x))))$.

Example

Factorial function $fact \in \mathbb{N} \to \mathbb{N}$ satisfies

$$fact(n) = \text{if } n = 0 \text{ then } 1 \text{ else } n.(fact(n-1))$$

Example

Factorial function $fact \in \mathbb{N} \to \mathbb{N}$ satisfies

$$fact(n) = \text{if } n = 0 \text{ then } 1 \text{ else } n.(fact(n-1))$$

and is λ-definable — it's represented by

$$Fact \triangleq Y(\lambda f x.\, If(Eq_0\, x)\, \underline{1}\, (Mult\, x(f(Pred\, x))))$$

(where $Mult \triangleq \lambda x_1 x_2 f x.\, x_1(x_2 f)x$ represents multiplication).

Representing primitive recursion

Recall that the class **PRIM** of primitive recursive functions is the smallest collection of (total) functions containing the basic functions and closed under the operations of composition and primitive recursion.

Combining the results about λ-definability so far, we have: **every $f \in \mathbf{PRIM}$ is λ-definable**.

So for λ-definability of all recursive functions, we just have to consider how to represent minimization. Recall...

Minimization

Given a partial function $f \in \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$, define $\mu^n f \in \mathbb{N}^n \rightharpoonup \mathbb{N}$ by

$\mu^n f(\vec{x}) \triangleq$ least x such that $f(\vec{x}, x) = 0$ and
for each $i = 0, \ldots, x-1$, $f(\vec{x}, i)$
is defined and > 0
(undefined if there is no such x)

Can express $\mu^n f$ in terms of a fixed point equation:
$\mu^n f(\vec{x}) \equiv g(\vec{x}, 0)$ where g satisfies $g = \Psi_f(g)$
with $\Psi_f \in (\mathbb{N}^{n+1} \rightharpoonup \mathbb{N}) \to (\mathbb{N}^{n+1} \rightharpoonup \mathbb{N})$ defined by

$$\Psi_f(g)(\vec{x}, x) \equiv \textit{if } f(\vec{x}, x) = 0 \textit{ then } x \textit{ else } g(\vec{x}, x+1)$$

Representing minimization

Suppose $f \in \mathbb{N}^{n+1} \to \mathbb{N}$ (totally defined function) satisfies $\forall \vec{a}\, \exists a\, (f(\vec{a}, a) = 0)$, so that $\mu^n f \in \mathbb{N}^n \to \mathbb{N}$ is totally defined.

Thus for all $\vec{a} \in \mathbb{N}^n$, $\mu^n f(\vec{a}) = g(\vec{a}, 0)$ with $g = \Psi_f(g)$ and $\Psi_f(g)(\vec{a}, a)$ given by
$\mathit{if}\ (f(\vec{a}, a) = 0)\ \mathit{then}\ a\ \mathit{else}\ g(\vec{a}, a+1)$.

So if f is represented by a λ-term F, then $\mu^n f$ is represented by

$$\lambda \vec{x}.\mathbf{Y}(\lambda z\, \vec{x}\, x.\, \mathbf{If}(\mathbf{Eq}_0(F\, \vec{x}\, x))\, x\, (z\, \vec{x}\, (\mathbf{Succ}\, x)))\, \vec{x}\, \underline{0}$$

Recursive implies λ-definable

Fact: every partial recursive $f \in \mathbb{N}^n \rightharpoonup \mathbb{N}$ can be expressed in a standard form as $f = g \circ (\mu^n h)$ for some $g, h \in \mathbf{PRIM}$. (Follows from the proof that computable = partial-recursive.)

Hence every (total) recursive function is λ-definable.

More generally, every partial recursive function is λ-definable, but matching up $\uparrow$ with $\nexists\beta{-}\mathbf{nf}$ makes the representations more complicated than for total functions: see [Hindley, J.R. & Seldin, J.P. (CUP, 2008), chapter 4.]

Computable = λ-definable

Theorem. A partial function is computable if and only if it is λ-definable.

We already know that computable = partial recursive $\Rightarrow$ λ-definable. So it just remains to see that **λ-definable functions are RM computable**. To show this one can

- code λ-terms as numbers (ensuring that operations for constructing and deconstructing terms are given by RM computable functions on codes)
- write a RM interpreter for (normal order) β-reduction.

The details are straightforward, if tedious.

Numerical coding of λ-terms

Fix an enumeration $x_0, x_1, x_2, \ldots$ of the set of variables.

For each λ-term M, define $\ulcorner M \urcorner \in \mathbb{N}$ by

$$\ulcorner x_i \urcorner = \ulcorner [0, i] \urcorner$$

$$\ulcorner \lambda x_i . M \urcorner = \ulcorner [1, i, \ulcorner M \urcorner] \urcorner$$

$$\ulcorner M N \urcorner = \ulcorner [2, \ulcorner M \urcorner, \ulcorner N \urcorner] \urcorner$$

(where $\ulcorner [n_0, n_1, \ldots, n_k] \urcorner$ is the numerical coding of lists of numbers from p 43).

Computable = λ-definable

Theorem. A partial function is computable if and only if it is λ-definable.

We already know that computable = partial recursive $\Rightarrow$ λ-definable. So it just remains to see that **λ-definable functions are RM computable**. To show this one can

- code λ-terms as numbers (ensuring that operations for constructing and deconstructing terms are given by RM computable functions on codes)
- write a RM interpreter for (normal order) β-reduction.

The details are straightforward, if tedious.

Summary

- Formalization of intuitive notion of ALGORITHM in several equivalent way

  cf. "Church-Turing Thesis"

- Limitative results: { undecidable problems, uncomputable functions }

"programs as data" + diagonalization