Computable = $\lambda$-definable

**Theorem.** A partial function is computable if and only if it is $\lambda$-definable.

We already know that

- Register Machine computable
  - Turing computable
  - partial recursive.

Using this, we break the theorem into two parts:

- every partial recursive function is $\lambda$-definable
- $\lambda$-definable functions are RM computable
Recall:

Representing primitive recursion

If \( f \in \mathbb{N}^n \rightarrow \mathbb{N} \) is represented by a \( \lambda \)-term \( F \) and \( g \in \mathbb{N}^{n+2} \rightarrow \mathbb{N} \) is represented by a \( \lambda \)-term \( G \), we want to show \( \lambda \)-definability of the unique \( h \in \mathbb{N}^{n+1} \rightarrow \mathbb{N} \) satisfying \( h = \Phi_{f,g}(h) \)

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

\[
\Phi_{f,g}(h)(\bar{a}, a) \triangleq \begin{cases} 
  f(\bar{a}) & \text{if } a = 0 \\
  g(\bar{a}, a - 1, h(\bar{a}, a - 1)) & \text{else}
\end{cases}
\]
Representing primitive recursion

If $f \in \mathbb{N}^n \rightarrow \mathbb{N}$ is represented by a $\lambda$-term $F$ and $g \in \mathbb{N}^{n+2} \rightarrow \mathbb{N}$ is represented by a $\lambda$-term $G$, we want to show $\lambda$-definability of the unique $h \in \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ satisfying $h = \Phi_{f,g}(h)$

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

Strategy:

- show that $\Phi_{f,g}$ is $\lambda$-definable;

$\lambda z \exists x. \text{If}(\text{Eq}_o x)(F \overrightarrow{x})(G \overrightarrow{x}(\text{pred}_x)(z \overrightarrow{x}(\text{pred}_x)))$
Representing primitive recursion

If \( f \in \mathbb{N}^n \rightarrow \mathbb{N} \) is represented by a \( \lambda \)-term \( F \) and 
\( g \in \mathbb{N}^{n+2} \rightarrow \mathbb{N} \) is represented by a \( \lambda \)-term \( G \),
we want to show \( \lambda \)-definability of the unique 
\( h \in \mathbb{N}^{n+1} \rightarrow \mathbb{N} \) satisfying 
\[ h = \Phi_{f,g}(h) \]
where \( \Phi_{f,g} \in (\mathbb{N}^{n+1} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N}^{n+1} \rightarrow \mathbb{N}) \) is given by...

Strategy:

- show that \( \Phi_{f,g} \) is \( \lambda \)-definable;
- show that we can solve fixed point equations 
  \[ X = MX \]
  up to \( \beta \)-conversion in the \( \lambda \)-calculus.
Curry’s fixed point combinator $Y$

$Y \triangleq \lambda f. (\lambda x. f(xx))(\lambda x. f(xx))$

$Y M \Rightarrow M(\lambda x. M(xx))(\lambda x. M(xx))$

$\Rightarrow M(\lambda x. M(xx))$

So for all $\lambda$-terms $M$ we have $Y M =_\beta M(Y M)$
## Origins of \( Y \)

<table>
<thead>
<tr>
<th>Naive set theory</th>
<th>( \lambda ) calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Russell set</strong></td>
<td><strong>( R \triangleq \lambda x. \neg (x \in x) )</strong></td>
</tr>
<tr>
<td>( R \triangleq { x</td>
<td>\neg (x \in x) } )</td>
</tr>
<tr>
<td>Naive set theory</td>
<td>$\lambda$ Calculus</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td><strong>Russell set:</strong></td>
<td></td>
</tr>
<tr>
<td>$R \triangleq { x \mid \neg (x \in x) }$</td>
<td>$R \triangleq \lambda x. \neg \text{true}(x , x)$</td>
</tr>
<tr>
<td><strong>Russell's Paradox:</strong></td>
<td></td>
</tr>
<tr>
<td>$R \in R \iff \neg (R \in R)$</td>
<td>$RR =_\beta , \neg \text{true}(RR , R)$</td>
</tr>
</tbody>
</table>
# Origins of Y

<table>
<thead>
<tr>
<th>Naive set theory</th>
<th>$\lambda$ calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Russell set:</strong></td>
<td>$R \triangleq \lambda x. \neg \forall R \in R$</td>
</tr>
<tr>
<td>$R \triangleq { x \mid \neg (x \in x) }$</td>
<td></td>
</tr>
<tr>
<td><strong>Russell’s Paradox:</strong></td>
<td>$\forall R \in R \iff \neg (R \in R)$</td>
</tr>
<tr>
<td>$\forall R \in R \iff \neg (R \in R)$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>$Y \neg \forall = \beta RR = (\lambda x. \neg \forall (x \in x))(\lambda x. \neg \forall (x \in x))$</td>
<td></td>
</tr>
</tbody>
</table>
## Origins of \( Y \)

<table>
<thead>
<tr>
<th>Naive set theory</th>
<th>( \lambda ) calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Russell set:</strong></td>
<td></td>
</tr>
<tr>
<td>( R \triangleq { x \mid \neg (x \in x) } )</td>
<td>( R \triangleq \lambda x. \neg \text{not}(x \in x) )</td>
</tr>
<tr>
<td><strong>Russell’s Paradox:</strong></td>
<td></td>
</tr>
<tr>
<td>( R \in R \iff \neg (R \in R) )</td>
<td>( RR = \beta \neg \text{not}(RR \in R) )</td>
</tr>
</tbody>
</table>

\[
Y \neg \text{not} = \beta RR = (\lambda x. \neg \text{not}(xx))(\lambda x. \neg \text{not}(xx))
\]

\[
Yf = (\lambda x. f(xx))(\lambda x. f(xx))
\]

\[
Y = \lambda f. (\lambda x. f(xx))(\lambda x. f(xx))
\]
Curry’s fixed point combinator $\mathbf{Y}$

\[ \mathbf{Y} \triangleq \lambda f. (\lambda x. f(xx)) (\lambda x. f(xx)) \]

satisfies $\mathbf{Y} M \rightarrow (\lambda x. M(xx)) (\lambda x. M(xx))$
Curry’s fixed point combinator \( \mathbf{Y} \)

\[
\mathbf{Y} \triangleq \lambda f. (\lambda x. f(x x))(\lambda x. f(x x))
\]

satisfies

\[
\mathbf{Y} M \rightarrow (\lambda x. M(x x))(\lambda x. M(x x))
\]

\[
\rightarrow M((\lambda x. M(x x))(\lambda x. M(x x)))
\]

hence

\[
\mathbf{Y} M \rightarrow M((\lambda x. M(x x))(\lambda x. M(x x))) \iff M(\mathbf{Y} M).
\]

So for all \( \lambda \)-terms \( M \) we have

\[
\mathbf{Y} M =_{\beta} M(\mathbf{Y} M)
\]
Turing's fixed point combinator

$$\Theta \triangleq AA$$

where

$$A \triangleq \lambda xy. y(xyx)$$
Turing's fixed point combinator

\[ \Theta \triangleq AA \]

where \( A \triangleq \lambda xy. y(xx) \)

\[ \Theta M = AAM = (\lambda xy. y(xx))AM \]
Turing’s fixed point combinator

\[ \Theta \triangleq AA \]

where
\[ A \triangleq \lambda xy. y(xxy) \]

\[ \Theta M = AAM = (\lambda xy. y(xxy))AM \]

\[ \rightarrow M(AAM) \]

\[ = M(\Theta M) \]
Representing primitive recursion

If $f \in \mathbb{N}^{n} \rightarrow \mathbb{N}$ is represented by a $\lambda$-term $F$ and $g \in \mathbb{N}^{n+2} \rightarrow \mathbb{N}$ is represented by a $\lambda$-term $G$, we want to show $\lambda$-definability of the unique $h \in \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ satisfying $h = \Phi_{f,g}(h)$

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

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

We now know that $h$ can be represented by

$$\mathcal{Y}(\lambda z \vec{x} x. \text{If}(\text{Eq}_0 x)(F \vec{x})(G \vec{x}(\text{Pred} x)(z \vec{x}(\text{Pred} x))))).$$
Representing primitive recursion

Recall that the class \textbf{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 $\lambda$-definability so far, we have: every $f \in \text{PRIM}$ is $\lambda$-definable.

So for $\lambda$-definability of all recursive functions, we just have to consider how to represent minimization. Recall...
Minimization

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

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

so $\mu^n f(\vec{x}) = g(\vec{x}, 0)$ where in general $g(\vec{x}, x)$ satisfies

$$
g(\vec{x}, x) = \begin{cases} 
0 & \text{if } f(\vec{x}, x) = 0 \text{ then } x \\
\text{else } g(\vec{x}, x+1) & \end{cases}
$$
Minimization

Given a partial function \( f \in \mathbb{N}^{n+1} \rightarrow \mathbb{N} \), define

\[ \mu^n f \in \mathbb{N}^n \rightarrow \mathbb{N} \]

by

\[ \mu^n f(\vec{x}) \triangleq \text{least } x \text{ such that } f(\vec{x}, x) = 0 \text{ and for each } i = 0, \ldots, x - 1, f(\vec{x}, i) \text{ is defined and } > 0 \]

(undefined if there is no such \( x \))

Can express \( \mu^n f \) in terms of a fixed point equation:

\[ \mu^n f(\vec{x}) \equiv g(\vec{x}, 0) \text{ where } g \text{ satisfies } g = \Psi_f(g) \]

with \( \Psi_f \in (\mathbb{N}^{n+1} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N}^{n+1} \rightarrow \mathbb{N}) \) defined by

\[ \Psi_f(g)(\vec{x}, x) \equiv \text{if } f(\vec{x}, x) = 0 \text{ then } x \text{ else } g(\vec{x}, x + 1) \]
Representing minimization

Suppose \( f \in \mathbb{N}^{n+1} \rightarrow \mathbb{N} \) (totally defined function) satisfies \( \forall \vec{a} \ \exists a \ (f(\vec{a}, a) = 0) \), so that \( \mu^n f \in \mathbb{N}^n \rightarrow \mathbb{N} \) 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
\[
\text{if } (f(\vec{a}, a) = 0) \text{ then } a \text{ else } g(\vec{a}, a + 1).
\]

So if \( f \) is represented by a \( \lambda \)-term \( F \), then \( \mu^n f \) is represented by
\[
\lambda \vec{x}.Y(\lambda z \vec{x}. \text{If}(\text{Eq}_0(F \vec{x} x)) x (z \vec{x}. (\text{Succ} x))) \vec{x} 0
\]
Recursive implies $\lambda$-definable

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

Hence every (total) recursive function is $\lambda$-definable.

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

**Theorem.** A partial function is computable if and only if it is \lambda\text{-definable.

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

- code \lambda\text{-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) \beta\text{-reduction.}
Computable = $\lambda$-definable

**Theorem.** A partial function is computable if and only if it is $\lambda$-definable.

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

- code $\lambda$-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) $\beta$-reduction.
Numerical coding of \( \lambda \)-terms

Fix an enumeration \( x_0, x_1, x_2, \ldots \) of the set of variables. For each \( \lambda \)-term \( M \), define \( ^\ast M \in \mathbb{N} \) by

\[
^\ast x_i = ^\ast [0, i] \\
^\ast \lambda x_i \cdot M = ^\ast [1, i, ^\ast M] \\
^\ast MN = ^\ast [2, ^\ast M, ^\ast N]
\]

(where \( ^\ast [n_0, n_1, \ldots, n_k] \) is the numerical coding of lists of numbers from p 43).
Computable = $\lambda$-definable

**Theorem.** A partial function is computable if and only if it is $\lambda$-definable.

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

- code $\lambda$-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) $\beta$-reduction.

The details are straightforward, if tedious.
Summary

- Formalization of intuitive notion of algorithm in several equivalent ways (cf. "Church-Turing Thesis")

- Limitative results: undecidable problems, uncomputable functions

  "programs as data" + diagonalization