Amazing fact #3: λ-calculus is Turing-complete!

- But the λ-calculus is too weak, right?
  - No multiple arguments!
  - No numbers or arithmetic!
  - No booleans or if!
  - No data structures!
  - No loops or recursion!

Multiple arguments: currying

- **Encode** multiple arguments via curried functions, just as in regular ML

$$\lambda(x_1, x_2).\ e \quad \Rightarrow \quad \lambda x_1.\ (\lambda x_2.\ e) \quad (\equiv \lambda x_1\ x_2.\ e)$$

$$f(e_1, e_2) \quad \Rightarrow \quad (f\ e_1)\ e_2$$

Church numerals

- **Encode** natural numbers using stylized lambda terms

  $$zero \equiv \lambda s.\ \lambda z.\ z$$

  $$one \equiv \lambda s.\ \lambda z.\ s\ z$$

  $$two \equiv \lambda s.\ \lambda z.\ s\ (s\ z)$$

  $$\ldots$$

  $$n \equiv \lambda s.\ \lambda z.\ s^n\ z$$

- A unary encoding using functions
  - No stranger than binary encoding

Arithmetic on Church numerals

- Successor function:
  take (the encoding of) a number,
  return (the encoding of) its successor
  - I.e., add an *s* to the argument's encoding

    $succ \equiv \lambda n.\ \lambda s.\ \lambda z.\ s\ (n\ s\ z)$

    $succ\ zero \rightarrow_\beta$
    $\lambda s.\ \lambda z.\ s\ (zero\ s\ z) \rightarrow_\beta^*$
    $\lambda s.\ \lambda z.\ s\ z\ =\ one$

    $succ\ two \rightarrow_\beta$
    $\lambda s.\ \lambda z.\ s\ (two\ s\ z) \rightarrow_\beta^*$
    $\lambda s.\ \lambda z.\ s\ (s\ (s\ z))\ =\ three$

Addition

- To add x and y, apply *succ* to y x times
  - Key idea: x is a function that, given a function and a base, applies the function to the base x times
    - "a number is as a number does"

$$plus \equiv \lambda x.\ \lambda y.\ x\ succ\ y$$

$$\begin{aligned}&plus\ two\ three \rightarrow_\beta^* \\ &\quad two\ succ\ three \rightarrow_\beta^* \\ &\quad succ\ (succ\ three) \ = \ five\end{aligned}$$

- Multiplication is repeated addition, similarly