Lambda Calculus
Part II

Overview

- Computing with Lambda
- How to represent
  - Booleans, Naturals
- Combinators
  - I
  - K
  - S
- Recursion
  - Y

Lambda Calculus

- As a programming language
  - We must be able to express
    - Constants
      - Booleans
      - Naturals
    - Elementary operations
      - Successor
      - Addition
      - Multiplication
  - We must be able to deal with recursion

Booleans

- How to represent true & false.
  - Hint
    - Think about them as functions
    - What are boolean used for ?
- Branching

Naturals

- What we need
  - A function to represent ZERO
  - A function to get the successor of a natural *n*
  - A function to get the predecessor of a natural *n*
  - A function to test for ZERO
- Church numerals
  - Zero: $\lambda s.\lambda z.z$
  - One: $\lambda s.\lambda z.s\ z$
  - Two: $\lambda s.\lambda z.s\ (s\ z)$
  - Three: $\lambda s.\lambda z.s\ (s\ (s\ z))$

Successor

- How to define successor ?
  - Natural n $= \lambda s.\lambda z.\underbrace{s\ (s\ (s\ ....\ (s}_{n \text{ of them}}\ z))...)$
  - Natural (n+1) $= \lambda s.\lambda z.\underbrace{s\ (s\ (s\ (s\ ....\ (s}_{n+1 \text{ of them}}\ z))...))$
  - Successor

IsZero

- Testing for Zero
  - Input
    - A Natural: n
  - Output
    - True if n is zero
    - False in all other cases otherwise
  - Idea
    - Make an induction on the church numeral
      - For zero $\lambda s.\lambda z.z$ return true. How ?
      - For others: return false, always.

Pairing & Lists

- How to define a Pair ?
  - A function
    - Takes two inputs
    - Returns something that denotes a pair (a function!)
  - Objective
    - We should be able to ask a pair for
      - Its first element
      - Its second element
  - Suggestions ?

Predecessor

- How to define *n-1* from *n* ?
  - Question
    - Can we eliminate applications within a numeral directly ?

Predecessor

- How to define *n-1* from *n* ?
  - Question
    - Can we eliminate applications within a numeral directly ?
  - Idea
    - Form a function that computes <n,n+1> from <n-1,n>
    - Apply the function n times starting with <?,zero>
    - What is the final pair ?

Predecessor

- How to define *n-1* from *n* ?
  - Question
    - Can we eliminate applications within a numeral directly ?
  - Idea
    - Form a function that computes <n,n+1> from <n-1,n>
    - Apply the function n times starting with <?,zero>
    - What is the final pair ?

```
Pair   = λx.λy.λz.z x y
First  = λp.p true
Second= λp.p false
np     = λp. Pair (Second p) (succ (Second p))
pred   = λn. First (n np (Pair zero zero))
```

Addition

- Objective
  - Compute m+n from naturals m,n
- Idea ?

Multiplication

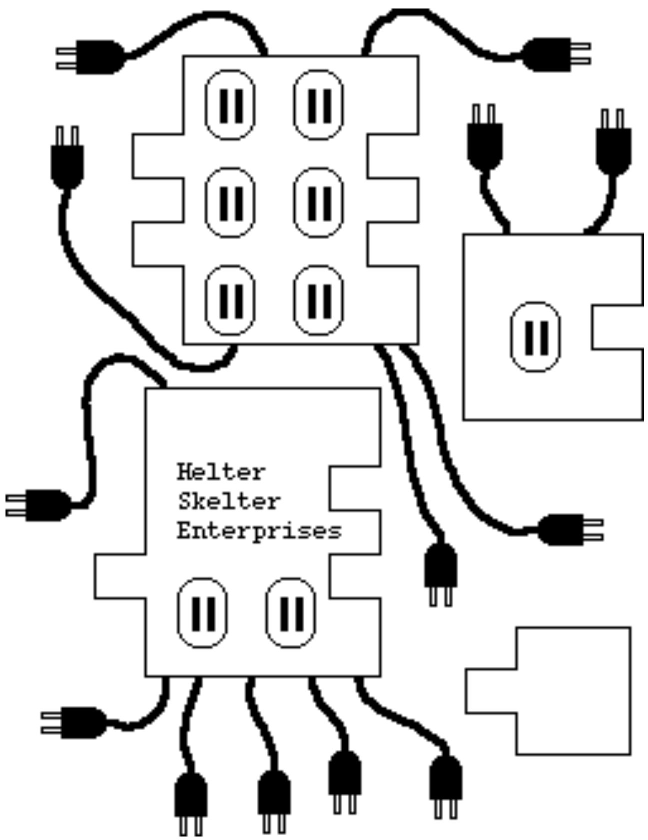
- Objective
  - Compute m*n from naturals m,n
- Idea ?

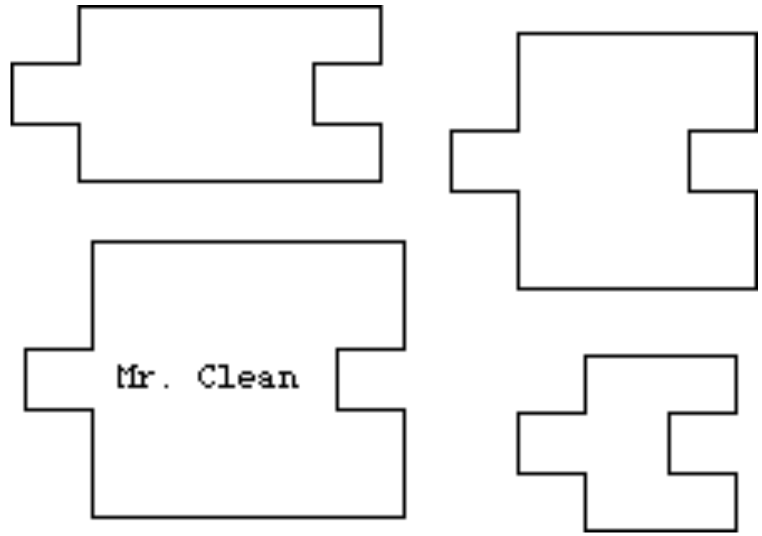
Overview

- Computing with Lambda
- How to represent
  - Booleans, Naturals
- Combinators
  - I
  - K
  - S
- Recursion
  - Y

Combinators

- What are these things?





Combinator I

- Purpose
  - Provide the Identity transformation
    - Take a lambda expression as input
    - Produce the lambda expression unchanged
  - Definition

```
I   = λx.x
```

Combinator K

- Purpose
  - Create a constant *function*
    - Take as input an expression
    - Produce as output a function that returns the expression
  - Definition

```
K   = λx.λy.x
```

Combinator S

- Purpose
  - Provide a general form of composition.
  - Definition

```
S  = λx.λy.λz.x z (y z)
```

Expressiveness

- What does SKK simplify to ?

```
I   = λx.x
K   = λx.λy.x
S   = λx.λy.λz.x z (y z)
```

Recursion

- Is achieved with one more combinator
  - Y
- Topic of next lecture is exclusively recursion