Lambda Calculus

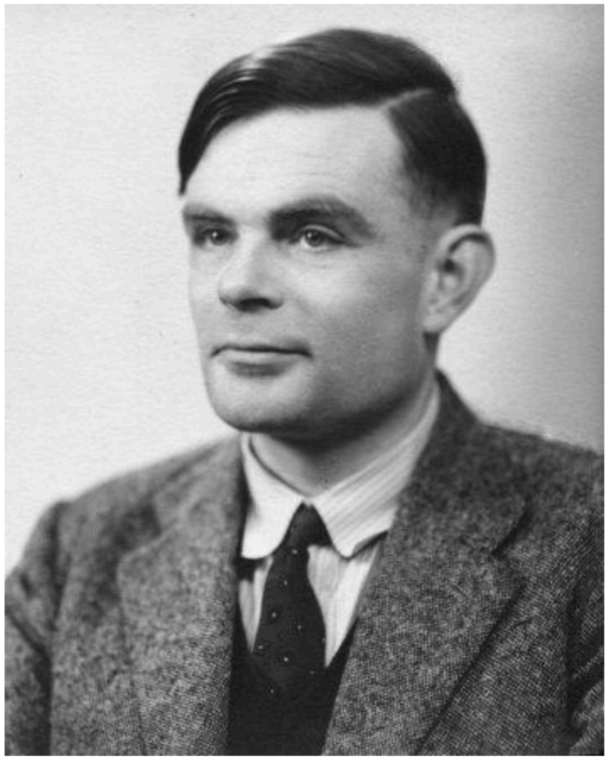
Announcements

cs61a.org/extra.html

Church-Turing Thesis

The Church-Turing Thesis

A function on the natural numbers is computable by a human following an algorithm, ignoring resource limitations, if and only if it is computable by a Turing machine.



Representation

Functions Can Represent Boolean Values

If all we have to work with are functions and call expressions, is there any way to represent other primitive values?

```
t = lambda a: lambda b: a
f = lambda a: lambda b: b

def py_pred(p):
    return p(True)(False)

def f_not(p):
    """Define Not.

    >>> py_pred(f_not(t))
    False
    >>> py_pred(f_not(f))
    True
    """
    return lambda a: lambda b: p(b)(a)
```

Exercise:

```
def f_and(p, q):
    """Define And.

    >>> py_pred(f_and(t, t))
    True
    >>> py_pred(f_and(t, f))
    False
    >>> py_pred(f_and(f, t))
    False
    >>> py_pred(f_and(f, f))
    False
    """
    return __p(q)(f)__
```

```
def f_or(p, q):
    """Define Or.

    >>> py_pred(f_or(t, t))
    True
    >>> py_pred(f_or(t, f))
    True
    >>> py_pred(f_or(f, t))
    True
    >>> py_pred(f_or(f, f))
    False
    """
    return __p(t)(q)__
```

Functions Can Represent Boolean Values

If all we have to work with are functions and call expressions, is there any way to represent other primitive values?

```
t = lambda a: lambda b: a
f = lambda a: lambda b: b

def py_pred(p):
    return p(True)(False)

def f_not(p):
    """Define Not.

    >>> py_pred(f_not(t))
    False
    >>> py_pred(f_not(f))
    True
    """
    return lambda a: lambda b: p(b)(a)
```

```
def f_if(p, a, b):
    """Define If.

    >>> py_pred(f_if(t, t, t))
    True
    >>> py_pred(f_if(t, t, f))
    True
    >>> py_pred(f_if(t, f, t))
    False
    >>> py_pred(f_if(t, f, f))
    False
    >>> py_pred(f_if(f, t, t))
    True
    >>> py_pred(f_if(f, t, f))
    False
    >>> py_pred(f_if(f, f, t))
    True
    >>> py_pred(f_if(f, f, f))
    False
    """
    return _______
```

Lambda Calculus Notation

Lambda Calculus

Variables: single letters, such as **x**

Functions: Instead of **lambda x: x** , write **λx.x** ; Instead of **lambda x, y: x** , write **λxy.x**

Assignment: Write **var f = ...**

Application: Instead of **f(x)** , write **(f x)** ; **f(x)(y)** and **f(x, y)** are both written **(f x y)**

Follow along! http://chenyang.co/lambda/

To type λ, just type \

var I = λx.x

var K = λr.(λs.r)

Are (I I) and I the same?

Are (K I) and I the same?

Are (K K I) and K the same?

Are (K I I) and (K I K) the same?

Are (K I K) and (K (I K)) the same?

What's ((K K) (K K)) the same as?

Can you construct a 4-argument function from K and I?

Boolean Values

Variables: single letters, such as **x**

Functions: Instead of **lambda x: x** , write **λx.x** ; Instead of **lambda x, y: x** , write **λxy.x**

Assignment: Write **var f = ...**

Application: Instead of **f(x)** , write **(f x)** ; **f(x)(y)** and **f(x, y)** are both written **(f x y)**

Follow along! http://chenyang.co/lambda/

To type λ, just type \

var T = λab.a

var F = λab.b

Define **and**, **or**, and **not**!

Define exclusive or:

```
xor(False, False) -> False
xor(False, True)  -> True
xor(True,  False) -> True
xor(True,  True)  -> False
```