CS 252:

Advanced Programming Language Principles

Lambda Calculus

Prof. Tom Austin

San José State University

Minimum complete programming language?

WARNING: I expect you to remember every construct of this language for exams

Lambda Calculus expressions

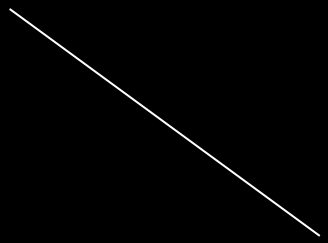
`e`	`::=`	*expressions:*
	`x`	variables
|	`(λx.e)`	lambda abstractions
|	`e e`	function application

We could have just said "function",
but we want to sound cool

Lambda Calculus values

`v ::=` *values:*

`(λx.e)` lambda abstractions



When our program finishes running, it returns some complex function as its "value"

Function application

Suppose we have a function:

`(λx.E)`

Where E is some complex expression.

How do we evaluate:

`(λx.E) v` → `E[x->v]`

v replaces x wherever it occurs in E

Small step semantics for λ-calculus

(in-class)

Operational Semantics

[Ctxt1]

$$\frac{e1 \rightarrow e1'}{e1\ e2 \rightarrow e1'\ e2}$$

[Ctxt2]

$$\frac{e2 \rightarrow e2'}{(\lambda x.e)\ e2 \rightarrow (\lambda x.e)\ e2'}$$

[Call]

$$(\lambda x.e)\ v \rightarrow e[x \rightarrow v]$$

Example: Identity Function

```
        (λx.x)  (λa.λb.a)
→       x[x->(λa.λb.a)]
→   (λa.λb.a)
```

When should we evaluate function arguments?

Strict Evaluation Strategies

Evaluate function arguments first

- *Call-by-value:*
  copy of the parameter is passed
- *Call-by-reference:*
  implicit reference is passed

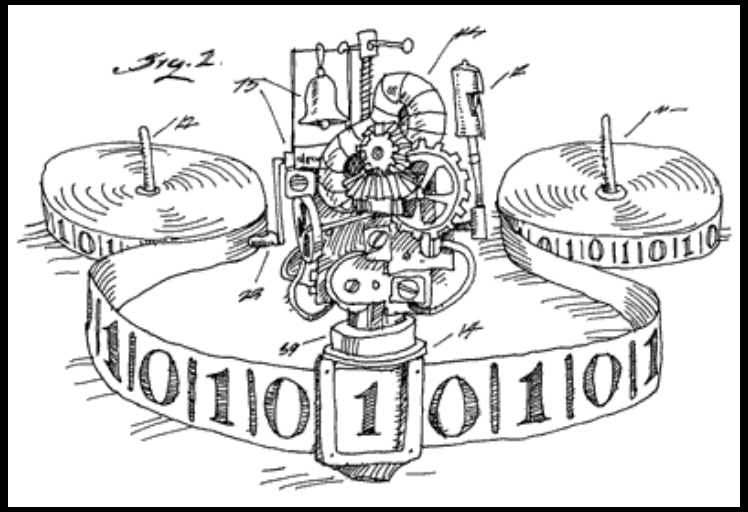
Lazy Evaluation Strategies

Substitute arguments in function body

- *Call-by-name:*
  re-evaluate the argument each time
- *Call-by-need:*
  memoizes parameter value after use

How powerful is this language?

The lambda-calculus is *Turing complete*.



You can also implement the λ-calculus w/ a Turing machine

In other words, the two are **equal in power**

Translating λ-calc to Haskell

Lambda-calculus	Haskell
`x`	`x`
`(λx.e)`	`(\x -> e)`
`e e`	`e e`

Extending the lambda calculus
(in class)

Lab: Develop new features in the Lambda Calculus using Haskell

Details on Canvas.

Starter code is available on the course website.