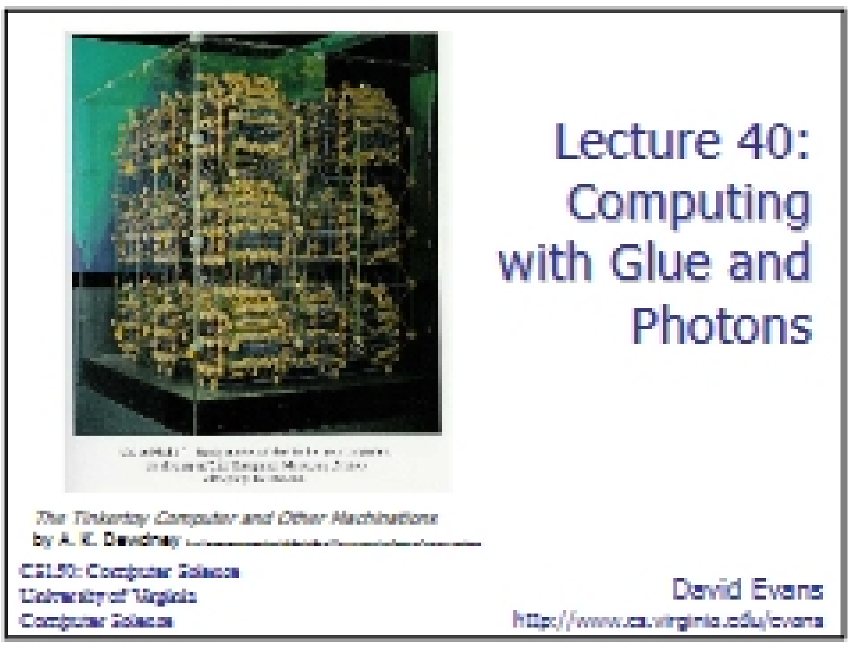
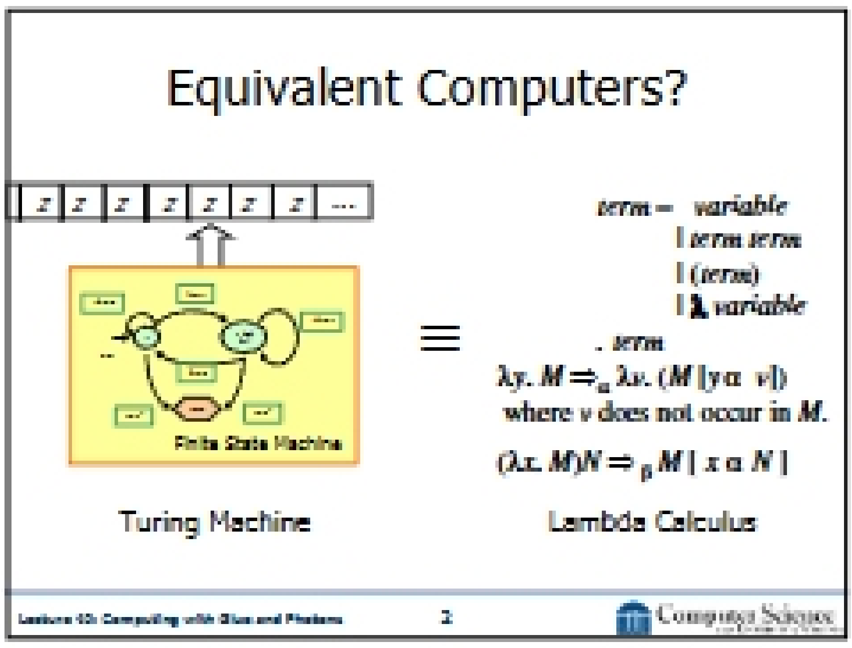


Lecture 40: Computing with Glue and Photons

The Tinkertoy Computer and Other Machinations
by A. K. Dewdney

CS150: Computer Science
University of Virginia
Computer Science

David Evans
http://www.cs.virginia.edu/evans



Equivalent Computers?

≡

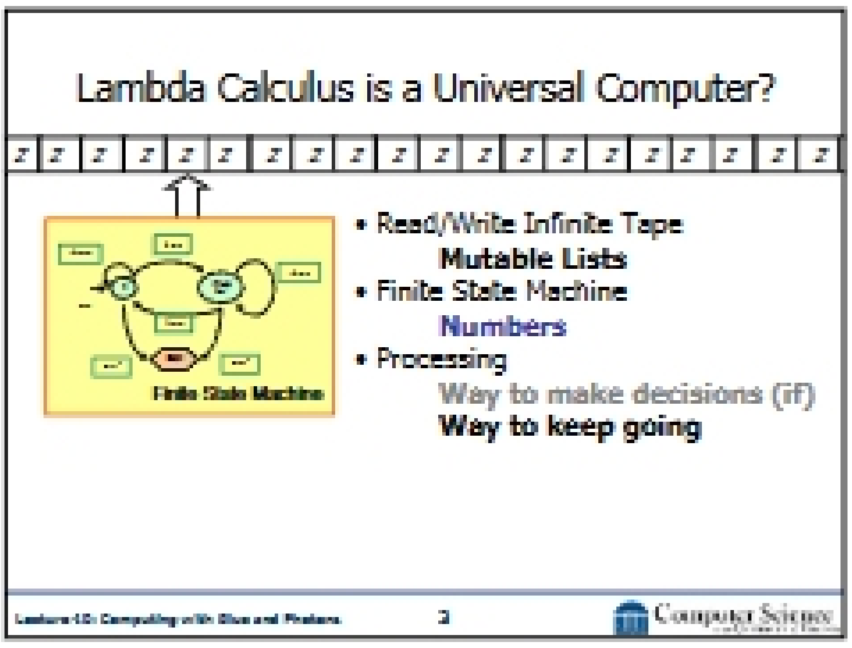
term = *variable*
| *term term*
| (*term*)
| λ *variable* . *term*

$\lambda y.\ M \Rightarrow_\alpha \lambda v.\ (M\,[y\ \alpha\ v])$
where *v* does not occur in *M*.

$(\lambda x.\ M)N \Rightarrow_\beta M\,[\,x\ \alpha\ N\,]$

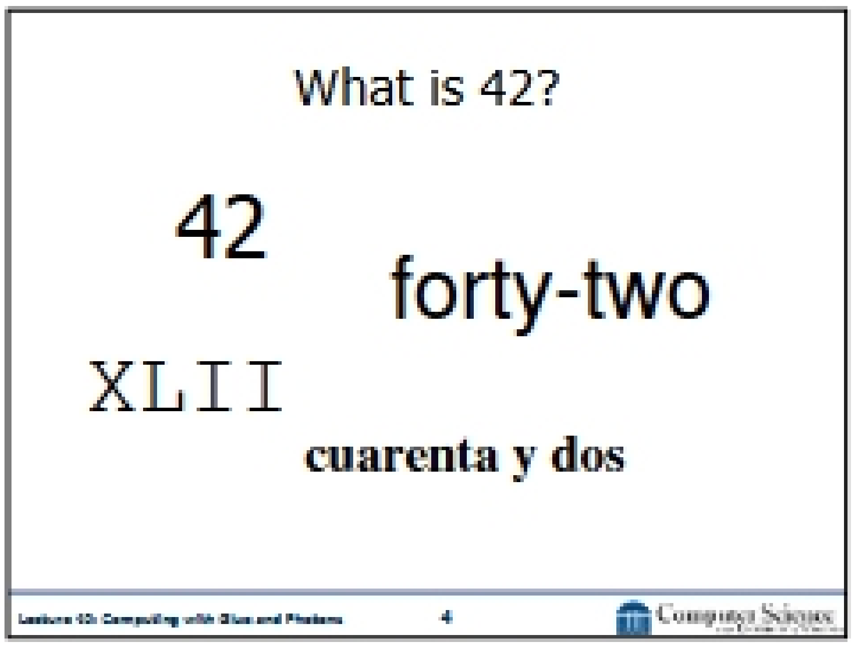
Turing Machine

Lambda Calculus



Lambda Calculus is a Universal Computer?

- Read/Write Infinite Tape
  Mutable Lists
- Finite State Machine
  Numbers
- Processing
  Way to make decisions (if)
  Way to keep going



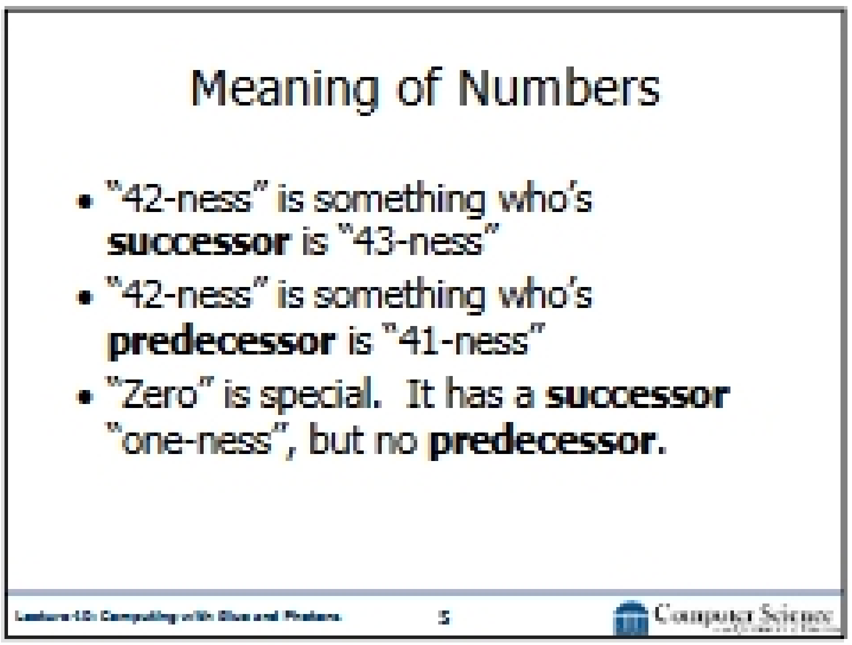
What is 42?

42

forty-two

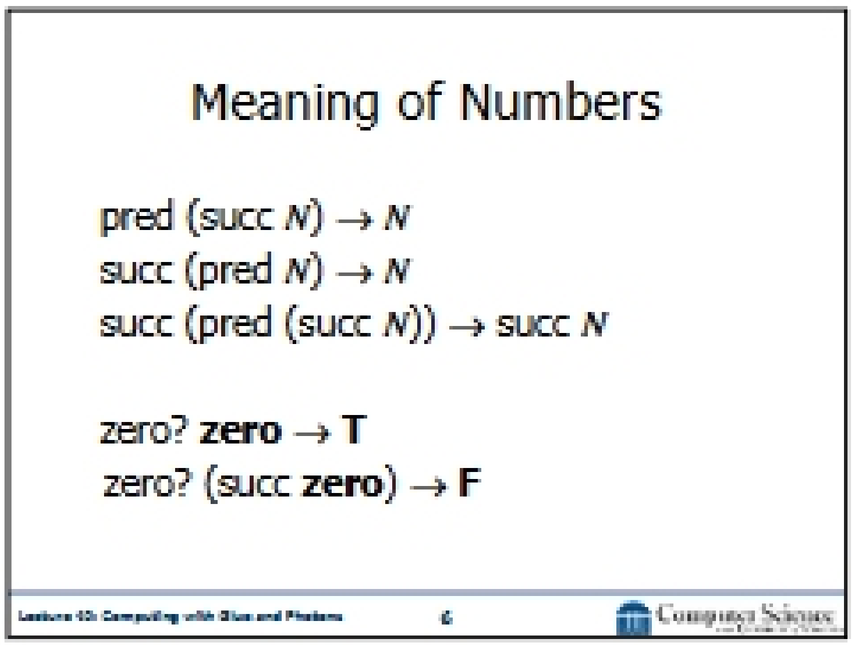
XLII

cuarenta y dos



Meaning of Numbers

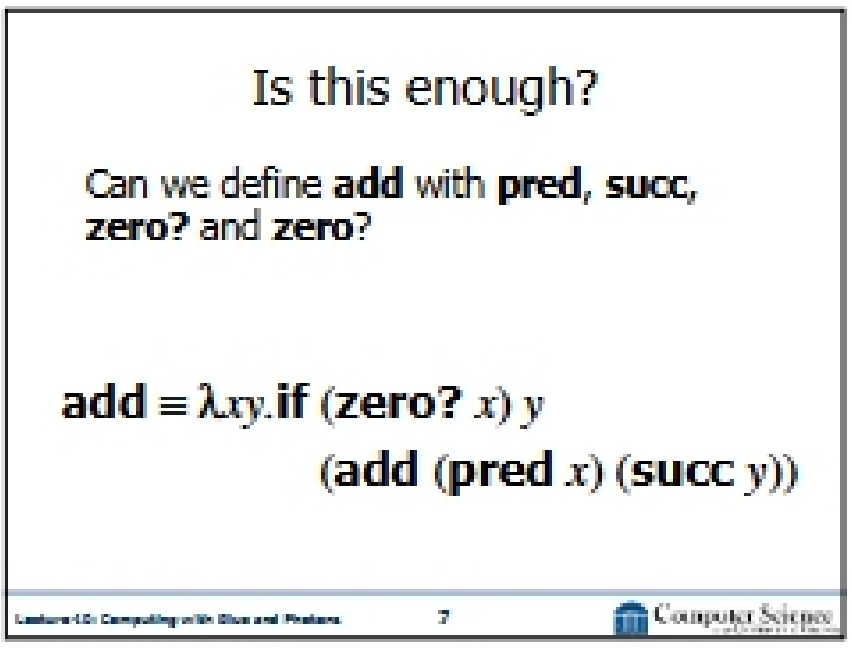
- "42-ness" is something who's **successor** is "43-ness"
- "42-ness" is something who's **predecessor** is "41-ness"
- "Zero" is special. It has a **successor** "one-ness", but no **predecessor.**



Meaning of Numbers

pred (succ *N*) → *N*
succ (pred *N*) → *N*
succ (pred (succ *N*)) → succ *N*

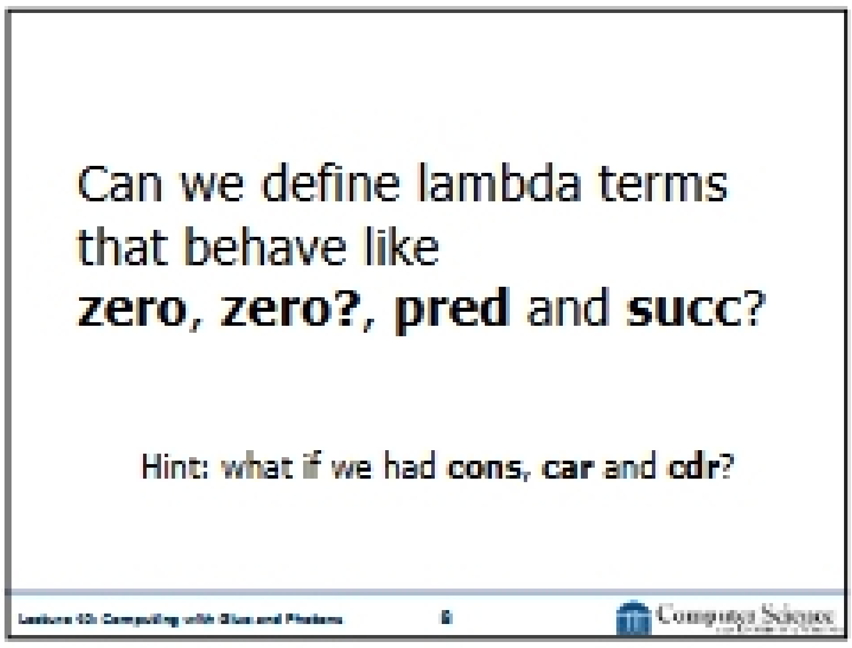
zero? **zero** → **T**
zero? (succ **zero**) → **F**



Is this enough?

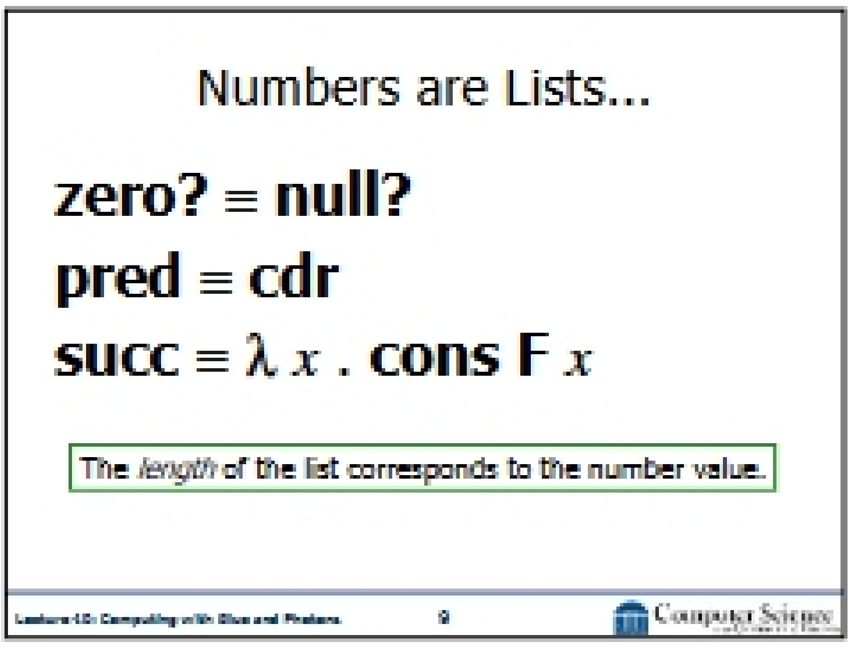
Can we define **add** with **pred**, **succ**, **zero?** and **zero**?

$$\textbf{add} \equiv \lambda xy.\textbf{if}\ (\textbf{zero?}\ x)\ y$$
$$(\textbf{add}\ (\textbf{pred}\ x)\ (\textbf{succ}\ y))$$



Can we define lambda terms that behave like **zero, zero?, pred** and **succ**?

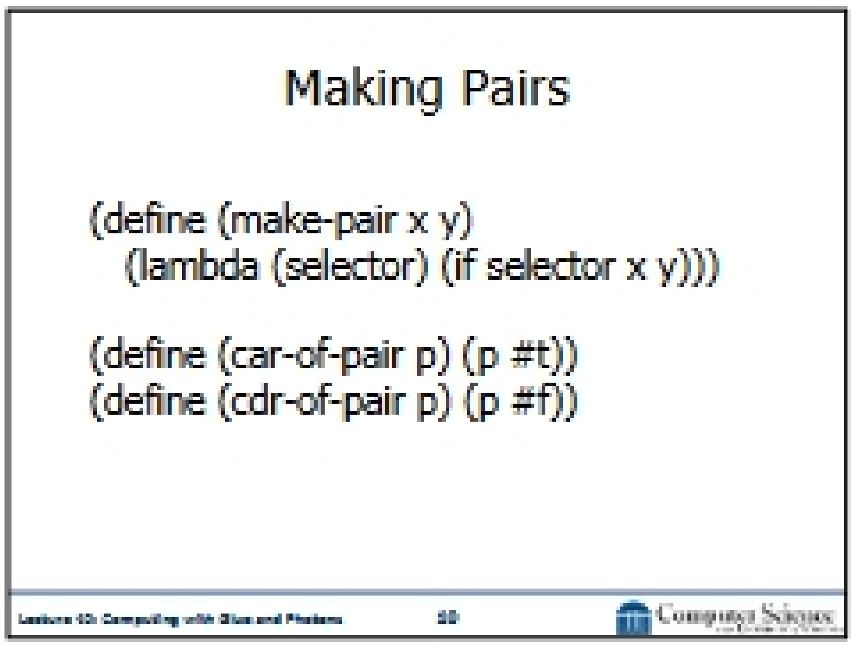
Hint: what if we had **cons**, **car** and **cdr**?



Numbers are Lists...

$$\textbf{zero?} \equiv \textbf{null?}$$
$$\textbf{pred} \equiv \textbf{cdr}$$
$$\textbf{succ} \equiv \lambda\ x\ .\ \textbf{cons}\ \textbf{F}\ x$$

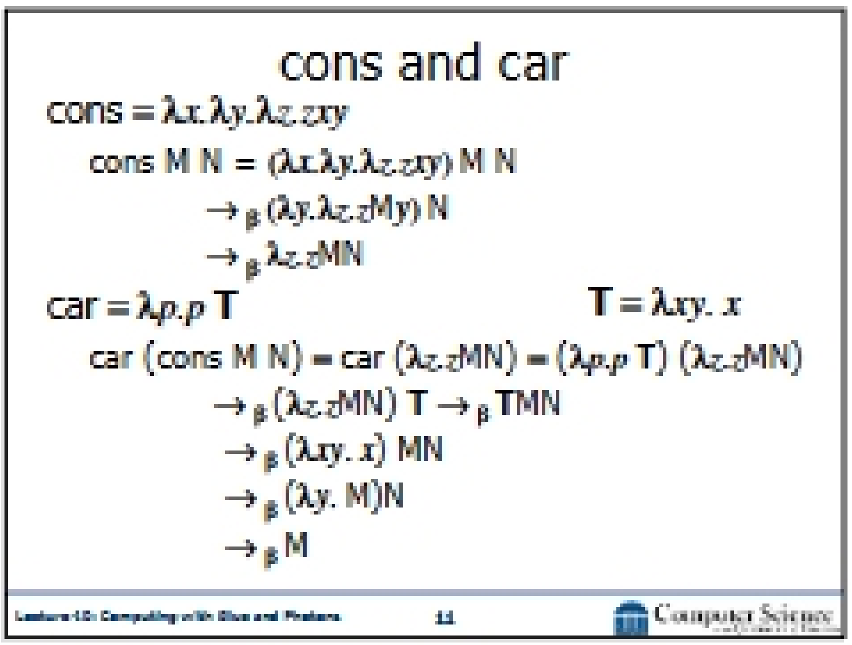
The *length* of the list corresponds to the number value.



Making Pairs

```
(define (make-pair x y)
   (lambda (selector) (if selector x y)))

(define (car-of-pair p) (p #t))
(define (cdr-of-pair p) (p #f))
```



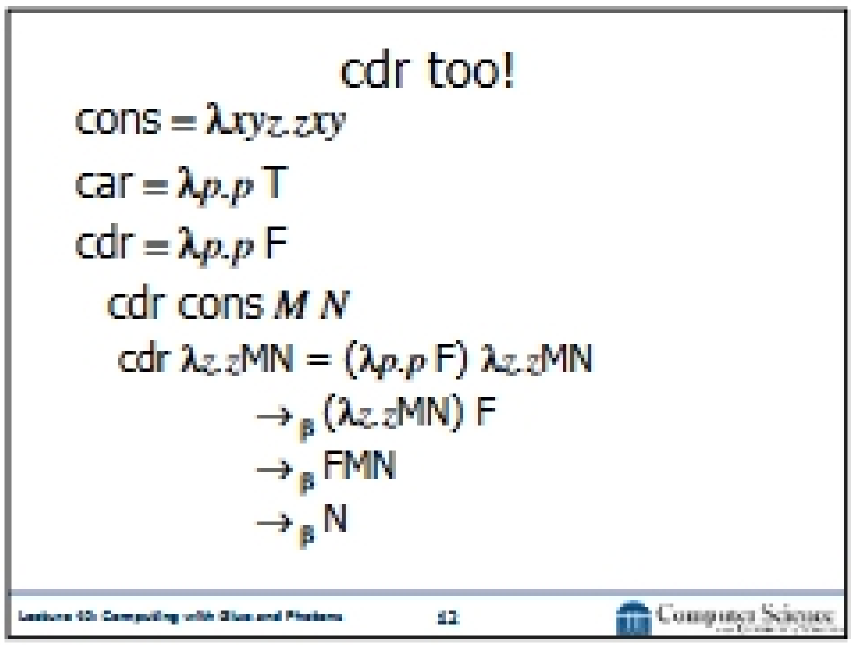
cons and car

$\text{cons} = \lambda x.\lambda y.\lambda z.zxy$

$$\text{cons M N} = (\lambda x.\lambda y.\lambda z.zxy)\ \text{M N}$$
$$\rightarrow_\beta (\lambda y.\lambda z.z\text{M}y)\ \text{N}$$
$$\rightarrow_\beta \lambda z.z\text{MN}$$

$\text{car} = \lambda p.p\ \textbf{T}$ $\qquad$ $\textbf{T} = \lambda xy.\ x$

$$\text{car (cons M N)} = \text{car}\ (\lambda z.z\text{MN}) = (\lambda p.p\ \text{T})\ (\lambda z.z\text{MN})$$
$$\rightarrow_\beta (\lambda z.z\text{MN})\ \text{T} \rightarrow_\beta \text{TMN}$$
$$\rightarrow_\beta (\lambda xy.\ x)\ \text{MN}$$
$$\rightarrow_\beta (\lambda y.\ \text{M})\text{N}$$
$$\rightarrow_\beta \text{M}$$



cdr too!

$\text{cons} = \lambda xyz.zxy$

$\text{car} = \lambda p.p\ \text{T}$

$\text{cdr} = \lambda p.p\ \text{F}$

$\text{cdr cons}\ M\ N$

$$\text{cdr}\ \lambda z.z\text{MN} = (\lambda p.p\ \text{F})\ \lambda z.z\text{MN}$$
$$\rightarrow_\beta (\lambda z.z\text{MN})\ \text{F}$$
$$\rightarrow_\beta \text{FMN}$$
$$\rightarrow_\beta \text{N}$$

Null and **null?**

null = $\lambda x.\mathbf{T}$

null? = $\lambda x.(x\ \lambda y.\lambda z.\mathbf{F})$

null? null $\rightarrow \lambda x.(x\ \lambda y.\lambda z.\mathbf{F})\ (\lambda x.\ \mathbf{T})$

$\rightarrow_\beta (\lambda x.\ \mathbf{T})(\lambda y.\lambda z.\mathbf{F})$

$\rightarrow_\beta \mathbf{T}$

Null and **null?**

null = $\lambda x.\mathbf{T}$

null? = $\lambda x.(x\ \lambda y.\lambda z.\mathbf{F})$

null? (cons M N) $\rightarrow \lambda x.(x\ \lambda y.\lambda z.\mathbf{F})\ \lambda z.z MN$

$\rightarrow_\beta (\lambda z.z\ MN)(\lambda y.\lambda z.\mathbf{F})$

$\rightarrow_\beta (\lambda y.\lambda z.\mathbf{F})\ MN$

$\rightarrow_\beta \mathbf{F}$

Counting

0 = null

1 = cons F 0

2 = cons F 1

3 = cons F 2

...

succ = $\lambda x.$**cons F** x

pred = $\lambda x.$**cdr** x

42 = $\lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y$

$\lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y$

$\lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y$

$\lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y$

$\lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y$

$\lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y$

$\lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y$

$\lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y$

$\lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y$

$\lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y$

$\lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y$

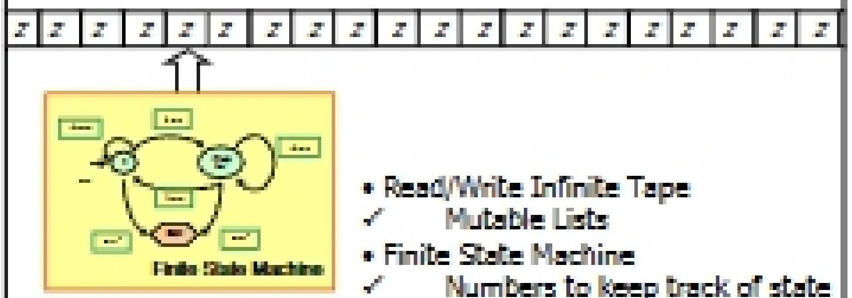
$\lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y$

$\lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y$

$\lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y$

$\lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda x.\mathrm{T}$

Lambda Calculus is a Universal Computer



- Read/Write Infinite Tape
- ✓ Mutable Lists
- Finite State Machine
- ✓ Numbers to keep track of state
- Processing
- ✓ Way of making decisions (if)
- ◐ Way to keep going

We have this, but we cheated using = to make recursive definitions!

Way to Keep Going

$(\lambda f.\ ((\lambda x.f\ (xx))\ (\lambda x.\ f\ (xx))))\ (\lambda z.z)$

$\rightarrow_\beta (\lambda x.(\lambda z.z)(xx))\ (\lambda x.\ (\lambda z.z)(xx))$

$\rightarrow_\beta (\lambda z.z)\ (\lambda x.(\lambda z.z)(xx))\ (\lambda x.(\lambda z.z)(xx))$

$\rightarrow_\beta (\lambda x.(\lambda z.z)(xx))\ (\lambda x.(\lambda z.z)(xx))$

$\rightarrow_\beta (\lambda z.z)\ (\lambda x.(\lambda z.z)(xx))\ (\lambda x.(\lambda z.z)(xx))$

$\rightarrow_\beta (\lambda x.(\lambda z.z)($

$\rightarrow_\beta \ldots$

This should give you some belief that we might be able to do it. We won't cover the details of why this works in this class.