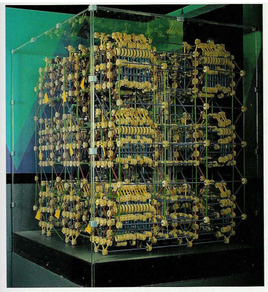


Color Plate 1 Early model of the Tinkertoy computer, on display at The Computer Museum, Boston, also plays tic-tac-toe.

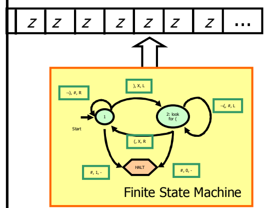
Lecture 40: Computing with Glue and Photons

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?



Turing Machine

$\equiv$

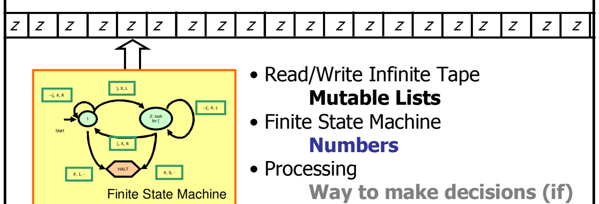
term = *variable*
| *term term*
| (*term*)
| **λ** *variable* . *term*

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

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

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) $\rightarrow N$
succ (pred N) $\rightarrow N$
succ (pred (succ N)) $\rightarrow$ succ N

zero? **zero** $\rightarrow$ **T**
zero? (succ **zero**) $\rightarrow$ **F**

Is this enough?

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

add $\equiv \lambda xy.$**if** (**zero?** x) y
(**add** (**pred** x) (**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...

zero? $\equiv$ **null?**

pred $\equiv$ **cdr**

succ $\equiv \lambda\ x$. **cons 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

cons $\equiv \lambda x.\lambda y.\lambda z.zxy$

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

car $\equiv \lambda p.p$ **T** **T** $\equiv \lambda xy.\ x$

car (cons M N) $\equiv$ car $(\lambda z.z\mathrm{MN}) \equiv (\lambda p.p\ \mathbf{T})\ (\lambda z.z\mathrm{MN})$
$\rightarrow_\beta (\lambda z.z\mathrm{MN})\ \mathbf{T} \rightarrow_\beta \mathbf{T}\mathrm{MN}$
$\rightarrow_\beta (\lambda xy.\ x)$ MN
$\rightarrow_\beta (\lambda y.\ \mathrm{M})\mathrm{N}$
$\rightarrow_\beta$ M

cdr too!

cons $\equiv \lambda xyz.zxy$

car $\equiv \lambda p.p$ T

cdr $\equiv \lambda p.p$ F

cdr cons $M\ N$

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

Null and **null?**

null ≡ $\lambda x.\mathbf{T}$

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

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

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

$\to_\beta \mathbf{T}$

Null and **null?**

null ≡ $\lambda x.\mathbf{T}$

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

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

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

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

$\to_\beta \mathbf{F}$

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

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

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

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

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

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

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

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

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

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

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

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

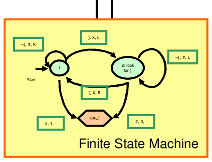
$\lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y\ \lambda xy.(\lambda z.z\ xy)\ \lambda xy.\ y$

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

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

Lambda Calculus is a Universal Computer

z z z z z z z z z z z z z z z z z z z z



- 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

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

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

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

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

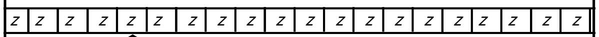
$\to_\beta\ (\lambda z.z)\ (\lambda\ x.(\lambda z.z)(xx))\ (\lambda\ x.(\lambda z.z)(xx))$

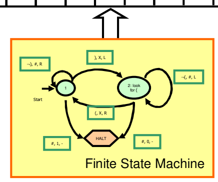
$\to_\beta\ (\lambda x.(\lambda z.z)(x$

$\to_\beta$ ...

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.

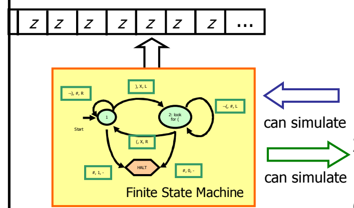
Lambda Calculus is a Universal Computer





- 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

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

Universal Computer

- Lambda Calculus can simulate a Turing Machine
  - Everything a Turing Machine can compute, Lambda Calculus can compute also
- Turing Machine can simulate Lambda Calculus (we didn't prove this)
  - Everything Lambda Calculus can compute, a Turing Machine can compute also
- Church-Turing Thesis: this is true for any other mechanical computer also

Normal Steps

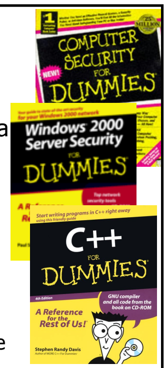
- Turing machine:
  - Read one square on tape, follow one FSM transition rule, write one square on tape, move tape head one square
- Lambda calculus:
  - One beta reduction
- Your PC:
  - Execute one instruction (?)
    - What one instruction does varies

Generalized Normal Steps

- Require a constant amount of time
- Perform a fixed amount of work
  - Read/write a constant amount of stuff
  - Make a constant number of decisions
  - Localized
  - Cannot scale (indefinitely) with input size

What about "non-mechanical" computers?

Quantum Physics for Dummies



- Light behaves like both a wave and a particle at the same time
- A single photon is in many states at once
- Can't observe its state without forcing it into one state
- Schrödinger's Cat
  - Put a live cat in a box with cyanide vial that opens depending on quantum state
  - Cat is both dead and alive at the same time until you open the box

Quantum Computing

- Feynman, 1982
- Quantum particles are in all possible states
- Can try lots of possible computations at once with the same particles
- In theory, can test all possible factorizations/keys/paths/etc. and get the right one!
- In practice, very hard to keep states entangled: once disturbed, must be in just one possible state

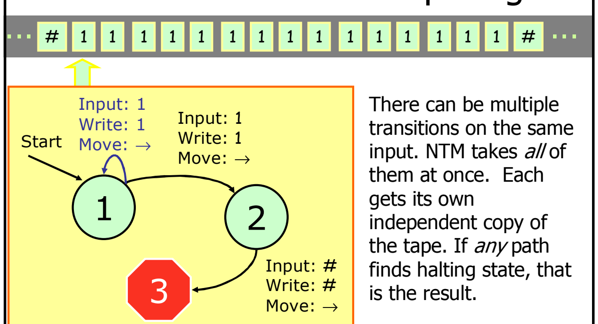
Qubit

- Regular bit: either a 0 or a 1
- Quantum bit: 0, 1 or in between
  - p% probability it is a 1
- A single qubit is in 2 possible states at once
- If you have 7 bits, you can represent any one of 2^7 different states
- If you have 7 qubits, you have 2^7 different states (at once!)

Quantum Computers Today

- Several quantum algorithms
  - Shor's algorithm: factoring using a quantum computer
- Actual quantum computers
  - 5-qubit computer built by IBM (2001)
  - Implemented Shor's algorithm to factor:
    - "World's most complex quantum computation" **15** (= 5 * 3)
  - D-Wave 16-qubit quantum computer (2007)
    - Solves Sudoku puzzles
- To exceed practical normal computing need > 50 qubits
  - Adding another qubit is more than twice as hard

Nondeterministic Computing



There can be multiple transitions on the same input. NTM takes *all* of them at once. Each gets its own independent copy of the tape. If *any* path finds halting state, that is the result.

Two Ways of Thinking about Nondeterminstic Computing

- Omniscient (all-knowing): machine always guesses right (the right guess is the one that eventually leads to a halting state)
- Omnipotent (all-powerful): machine can split in two every step, all resulting machines execute on each step, if one of the machines halts its tape is the output

Computability

Is a nondeterministic TM more **powerful** than a deterministic TM?

No! We can simulate a nondeterminstic TM with a regular TM.

Efficiency

Is a nondeterministic TM **faster** than a deterministic TM?

Unknown! This is the most famous open problem in CS.

Charge

- Friday's class: P versus NP (the nondeterministic TM question)
- Qualification for Monday's presentations
  - Send me a URL for your site before 11:59pm Friday
  - Basic functionality should be working
  - You can keep developing after this (if something breaks, you won't be disqualified, but be smart and keep a copy of what works!)