Class 32: Computability in Theory and Practice

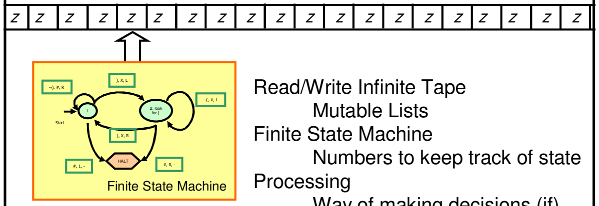
CS150: Computer Science
University of Virginia
Computer Science

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

Menu

- Lambda Calculus Review
- Computability in Theory and Practice
- Learning to Count

Universal Computation



Finite State Machine

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

To prove Lambda Calculus is as powerful as a UTM, we must show we can make everything we need to simulate any TM.

Don’t search for **T**, search for **if**

$$\mathbf{T} \equiv \lambda x\ (\lambda y.\ x)$$

$$\equiv \lambda xy.\ x$$

$$\mathbf{F} \equiv \lambda x\ (\lambda\ y.\ y))$$

$$\mathbf{if} \equiv \lambda pca\ .\ pca$$

Finding the Truth

$\mathbf{T} \equiv \lambda x\ .\ (\lambda y.\ x)$

$\mathbf{F} \equiv \lambda x\ .\ (\lambda y.\ y)$

$\mathbf{if} \equiv \lambda p\ .\ (\lambda c\ .\ (\lambda a\ .\ pca)))$

Is the **if** necessary?

$\mathbf{if}\ \mathbf{T}\ \boldsymbol{M}\ \boldsymbol{N}$

$((\lambda pca\ .\ pca)\ (\lambda xy.\ x))\ \boldsymbol{M}\ \boldsymbol{N}$

$\rightarrow_\beta (\lambda ca\ .\ (\lambda x.(\lambda y.\ x))\ ca))\ \boldsymbol{M}\ \boldsymbol{N}$

$\rightarrow_\beta \rightarrow_\beta (\lambda x.(\lambda y.\ x))\ \boldsymbol{M}\ \boldsymbol{N}$

$\rightarrow_\beta (\lambda y.\ \boldsymbol{M}\ ))\ \boldsymbol{N} \rightarrow_\beta \boldsymbol{M}$

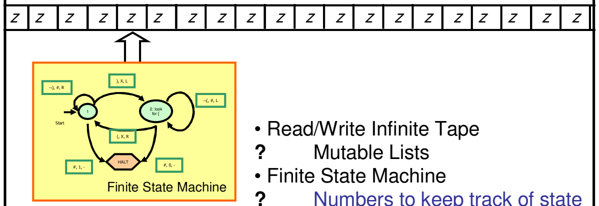
---

and and **or**?

$\mathbf{and} \equiv \lambda x\ (\lambda y.\ \mathbf{if}\ x\ y\ \mathbf{F}))$

$\mathbf{or} \equiv \lambda x\ (\lambda y.\ \mathbf{if}\ x\ \mathbf{T}\ y))$

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

Computability in Theory and Practice

(Intellectual Computability Discussion on TV Video)

Ali G Multiplication Problem

- Input: a list of 2 numbers with up to *d* digits each
- Output: the product of the 2 numbers

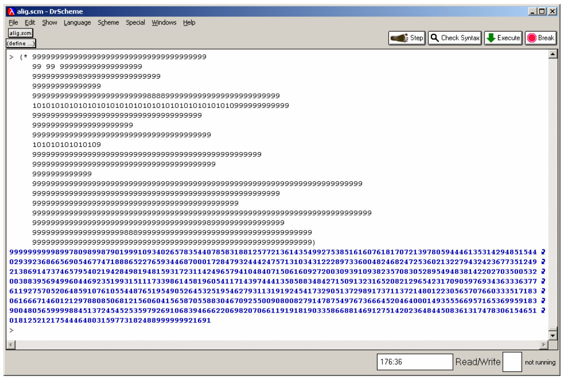
Is it decidable?

Yes – a straightforward algorithm solves it.

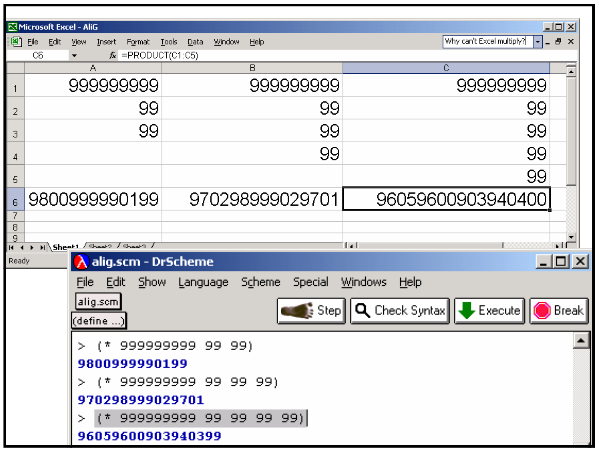
Is it tractable? (how much work?)

Yes – it using elementary multiplication techniques it is $O(d^2)$

Can *real* computers solve it?



```
> (* 999999999999999999999999999999999999
     99 99 999999999999999999
     9999999999899999999999999999
     999999999999999
     99999999999999999999999999888999999999999999999999999
     10101010101010101010101010101010101010101099999999999
     999999999999999999999999999999999999
     9999999999999999999999
     999999999999999999999999999999999999999
     101010101010109
     9999999999999999999999999999999999999999999999999
     9999999999999999999999999999999999999999
     9999999999999
     999999999999999999999999999999999999999999999999999999999999999999999999
     99999999999999999999999999999999999999999999999999999
     999999999999999999999999999999999999999999999
     9999999999999999999999999999999999999999999999999999999999999999999999999
     99999999999999999999999999999999999999999998999999999999999
     99999999999999999999999888999999999999999999999999999999999999999
     9999999999999999999999999999999999999999999999999999999999999999999)
99999999998997809899879019991093402657835440785831881257721361435499275385161607618170721397805944461353142948515442
0293923686656905467747188865227659344687000172847932444247571310343122289733600482468247253602132279432423677351249
2138691473746579540219428498194815931723114249657941048407150616092720030939109382357083052895494838142202703500532
0038839569499604469235199315111733986145819605411714397444135858834842715091323165208212965423170905976934363336377
6119275705206485910761055448765195490526453251954627931131919245417329051372989173711372148012230565707660333517183
0616667146012129788085068121560604156587055883046709255009080082791478754976736664520464000149355566957165369959183
9004805659999884513724545253597926910683946662206982070661191918190335866881469127514202364844508361317478306154651
0181252121754446480315977318248899999921691
>
```



C6 =PRODUCT(C1:C5)

	A	B	C
1	999999999	999999999	999999999
2	99	99	99
3	99	99	99
4		99	99
5			99
6	9800999990199	970298999029701	96059600903940400
7			
8			
9			

```
> (* 999999999 99 99)
9800999990199
> (* 999999999 99 99 99)
970298999029701
> (* 999999999 99 99 99 99)
96059600903940399
```

What about C++?

```
int main (void)
{
  int alig = 999999999;

  printf ("Value: %d\n", alig);
  alig = alig * 99;
  printf ("Value: %d\n", alig);
  alig = alig * 99;
  printf ("Value: %d\n", alig);
  alig = alig * 99;
  printf ("Value: %d\n", alig);
}
```

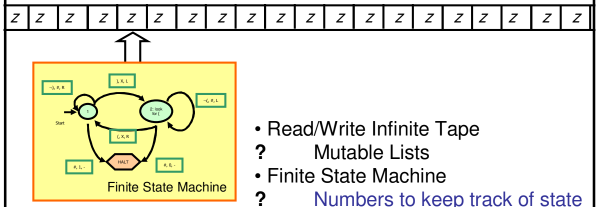
Results from SunOS 5.8:

```
Value: 999999999
Value: 215752093
Value: -115379273
Value: 1462353861
```

Ali G was Right!

- Theory assumes ideal computers:
  - Unlimited, perfect memory
  - Unlimited (finite) time
- Real computers have:
  - Limited memory, time, power outages, flaky programming languages, etc.
  - There are many decidable problems we cannot solve with real computer: the numbers ***do*** matter

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

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

Meaning of Zero

zero? zero

→ T

zero? (succ zero)

→ F

zero? (pred (succ zero))

→ T

Is this enough?

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

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

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? ≡ null?

pred ≡ cdr

succ ≡ $\lambda\ x$. cons F x

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 ≡ $\lambda x.\lambda y.\lambda z.zxy$

cons M N = $(\lambda x.\lambda y.\lambda z.zxy)$ M N

$\rightarrow_\beta (\lambda y.\lambda z.z\text{M}y)$ N

$\rightarrow_\beta \lambda z.z\text{MN}$

car ≡ $\lambda p.p$ T

T ≡ $\lambda x\ .\ \lambda y.\ x$

car (cons M N) ≡ car $(\lambda z.z\text{MN})$ ≡ $(\lambda p.p\ \text{T})\ (\lambda z.z\text{MN})$

$\rightarrow_\beta (\lambda z.z\text{MN})\ \text{T} \rightarrow_\beta \text{TMN}$

$\rightarrow_\beta (\lambda x\ .\ \lambda y.\ x)$ MN

$\rightarrow_\beta (\lambda y.\ \text{M})\text{N}$

$\rightarrow_\beta$ M

---

cdr too!

cons ≡ $\lambda xyz.zxy$

car ≡ $\lambda p.p$ T

cdr ≡ $\lambda p.p$ F

cdr cons $M\ N$

cdr $\lambda z.z\text{MN}$ = $(\lambda p.p\ \text{F})\ \lambda z.z\text{MN}$

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

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

$\rightarrow_\beta \mathbf{T}$

Null and **null?**

null ≡ $\lambda x.\mathbf{T}$

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

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

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

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

$\rightarrow_\beta \mathbf{F}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Arithmetic

zero? ≡ null?

succ ≡ $\lambda x.$ cons **F** x

pred ≡ $\lambda x.x$ **F**

pred 1 = $(\lambda x.x\ \mathbf{F})$ cons **F** null

$\rightarrow_\beta$ (cons F null) F

$\rightarrow$ $(\lambda xyz.zxy$ F null) F

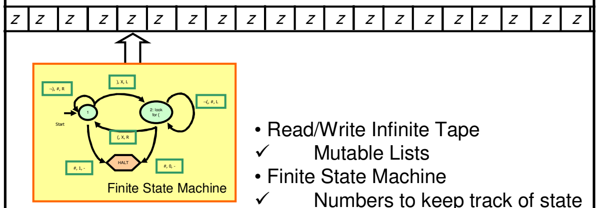
$\rightarrow_\beta$ $(\lambda z.z$ F null) F

$\rightarrow_\beta$ F F null

$\rightarrow_\beta$ null

≡ 0

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

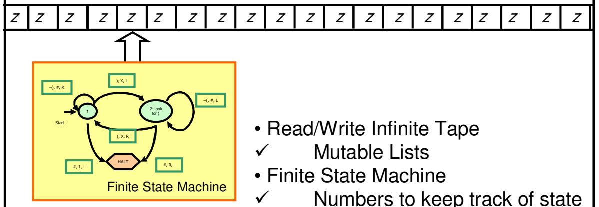
We have this, but we cheated using ≡ to make recursive definitions!

Way to Keep Going

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

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. (CS655 sometimes does.)

Lambda Calculus is a Universal Computer



Finite State Machine

- 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

Universal Computer

- Lambda Calculus can simulate a Turing Machine
  - Everytime 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

Charge

- Exam 2 out Friday
  - Covers through today
  - Links to example exams on the web
  - Review session Wednesday, 7pm
- PS8 Project Ideas due tomorrow (11:59pm)
  - Short email is fine, just explain who your team is and what you plan to do