Class 30: Computability in Theory and Practice

CS150: Computer Science
University of Virginia
Computer Science

Greg Humphreys
http://www.cs.virginia.edu/~humper

Menu

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

Universal Computation

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

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

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

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

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

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

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

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

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

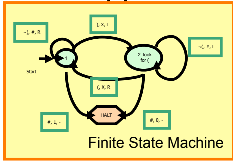
$\rightarrow_\beta (\lambda y.\ \boldsymbol{M}\ ))\ N \rightarrow_\beta \boldsymbol{M}$

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

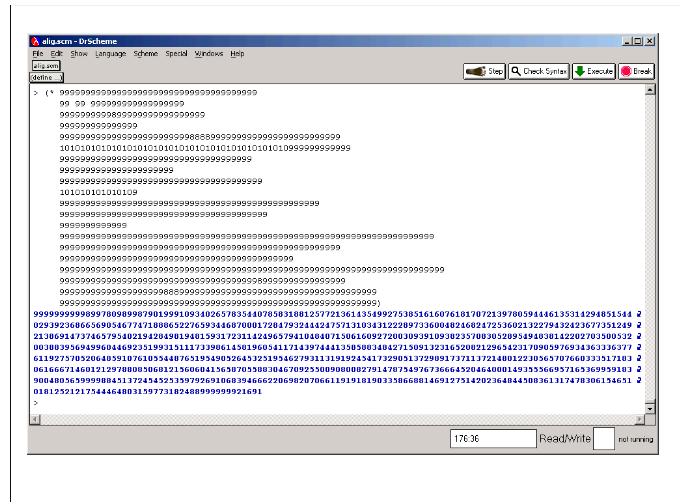
Is it decidable?

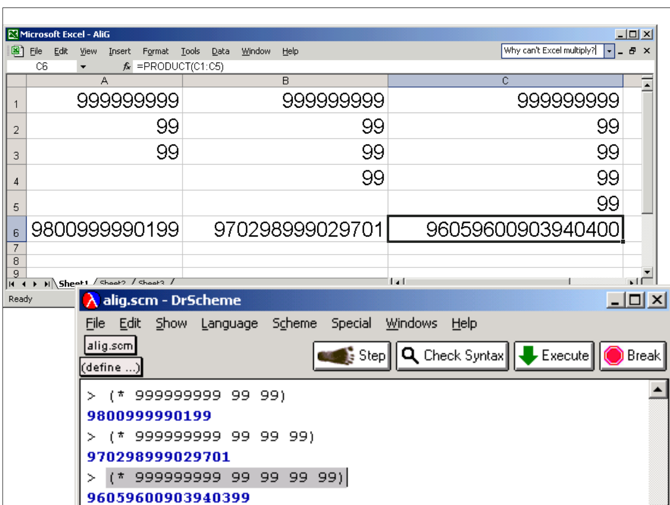
Yes – a straightforward algorithm solves it.

Is it tractable? (how much work?)

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

Can *real* computers solve it?





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?

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

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

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?

add ≡ $\lambda xy.$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? ≡ 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

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

$\rightarrow_\beta \lambda z.z$MN

cons and car

cons $\rightarrow_\beta \lambda z.z$MN

T ≡ $\lambda x\ .\ \lambda y.\ x$

car ≡ $\lambda p.p$ T

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

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

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

$\rightarrow_\beta$ ($\lambda y.$ M)N

$\rightarrow_\beta$ M

cdr too!

cons ≡ λ*xyz.zxy*

car ≡ λ*p.p* T

cdr ≡ λ*p.p* F

cdr cons *M N*

cdr λ*z.z*MN = (λ*p.p* F) λ*z.z*MN

$\rightarrow_{\beta}$ (λ*z.z*MN) F

$\rightarrow_{\beta}$ FMN

Null and **null?**

null ≡ λ*x*.**T**

null? ≡ λ*x*.(*x* λ*y*.λ*z*.**F**)

null? null → λ*x*.(*x* λ*y*.λ*z*.**F**) (λ*x*. **T**)

$\rightarrow_{\beta}$ (λ*x*. **T**)(λ*y*.λ*z*.**F**)

$\rightarrow_{\beta}$ **T**

Null and **null?**

null ≡ λ*x*.**T**

null? ≡ λ*x*.(*x* λ*y*.λ*z*.**F**)

null? (cons M N) → λ*x*.(*x* λ*y*.λ*z*.**F**) λ*z.z*MN

$\rightarrow_{\beta}$ (λ*z.z* MN)(λ*y*.λ*z*.**F**)

$\rightarrow_{\beta}$ (λ*y*.λ*z*.**F**) MN

$\rightarrow_{\beta}$ **F**

Counting

0 ≡ null

1 ≡ cons F 0

2 ≡ cons F 1

3 ≡ cons F 2

…

succ ≡ λ*x*.cons F *x*

pred ≡ λ*x*.cdr *x*

42 = λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y
λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z
xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy.
y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.
(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy)
λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y
λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z
xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy.
y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.
(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy)
λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y
λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z
xy) λxy. y λxy.(λz.z xy) λxy. y λxy.(λz.z xy) λxy. y λx.T

Arithmetic

zero? ≡ null?

succ ≡ λ*x*. cons **F** *x*

pred ≡ λ*x.x* **F**

pred 1 = (λ*x.x* **F**) cons **F** null

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

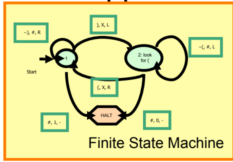
→ (λ*xyz.zxy* F null) F

$\rightarrow_{\beta}$ (λ*z.z* F null) F

$\rightarrow_{\beta}$ F F null

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



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

- 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

Way to Keep Going

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

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

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

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

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

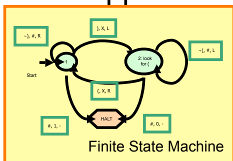
$\rightarrow_\beta\ (\lambda x.($

$\rightarrow_\beta\ \ldots$

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

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

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

- PS7
- Suggestions for what you'd like to learn about?