Class 32: Computability in Theory and Practice

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

Universal Computation
- 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 \( \text{if} \)

\[
T \equiv \lambda x \ (\lambda y. \ x) \\
\equiv \lambda xy. \ x \\
F \equiv \lambda x \ (\lambda y. \ y)) \\
\text{if} \equiv \lambda pca . \ pca
\]

Finding the Truth

\[
T = \lambda x . (\lambda y. x) \\
F = \lambda x . (\lambda y. y) \\
\text{if} = \lambda p . (\lambda c . (\lambda a . pca))) \\
\text{is the if necessary?} \\
\text{if} T M N \\
(\lambda pca . pca) (\lambda xy. x) M N \\
\rightarrow_\beta (\lambda ca . (\lambda x. (\lambda y. x)) ca) M N \\
\rightarrow_\beta \rightarrow_\beta (\lambda x. (\lambda y. x) M N \\
\rightarrow_\beta (\lambda y. M)) N \rightarrow_\beta M
\]

and and or?

\[
\text{and} \equiv \lambda x \ (\lambda y. \text{if} \ y \ y \ F)) \\
\text{or} \equiv \lambda x \ (\lambda y. \text{if} \ x \ T \ y))
\]
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++?

```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

\[
\begin{align*}
\text{pred (succ } N) & \rightarrow N \\
\text{succ (pred } N) & \rightarrow N \\
\text{succ (pred (succ } N)) & \rightarrow \text{succ } N 
\end{align*}
\]

Meaning of Zero

\[
\begin{align*}
\text{zero? zero} & \rightarrow T \\
\text{zero? (succ zero)} & \rightarrow F \\
\text{zero? (pred (succ zero))} & \rightarrow T 
\end{align*}
\]
Is this enough?
• Can we define add with pred, succ, zero? and zero?

\[
\text{add} \equiv \lambda x. y. \text{if} \ (\text{zero? } x) \ y \\
\quad (\text{add} \ (\text{pred} x) \ (\text{succ} y))
\]

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...

\[
\begin{align*}
\text{zero?} & \equiv \text{null} \\
\text{pred} & \equiv \text{cdr} \\
\text{succ} & \equiv \lambda x. \text{cons} \ F \ x
\end{align*}
\]

Making Pairs

\[
\begin{align*}
\text{(define (make-pair x y) (lambda (selector) (if selector x y)))} \\
\text{(define (car-of-pair p) (p #t))} \\
\text{(define (cdr-of-pair p) (p #f))}
\end{align*}
\]

cons and car

\[
\begin{align*}
\text{cons} & \equiv \lambda x. \lambda y. \lambda z. zxy \\
\text{cons } M \ N & = (\lambda x. \lambda y. \lambda z. zxy) \ M \ N \\
& \rightarrow_\beta (\lambda x. \lambda z. zMy) \ N \\
& \rightarrow_\beta \lambda z. zMN \\
\text{car} & \equiv \lambda p. p \ T \\
\text{car } (\text{cons } M \ N) & = \text{car} \ (\lambda z. zMN) = (\lambda x. \lambda y. x) \ (\lambda z. zMN) \\
& \rightarrow_\beta (\lambda x. \lambda y. x) \ MN \\
& \rightarrow_\beta (\lambda y. M) N \\
& \rightarrow_\beta M
\end{align*}
\]

cdr too!

\[
\begin{align*}
\text{cons} & \equiv \lambda x. y. z. zxy \\
\text{car} & \equiv \lambda p. p \ T \\
\text{cdr} & \equiv \lambda p. p \ F \\
\text{cdr } \text{cons } M \ N & = (\lambda p. p \ F) \ (\lambda z. zMN) \\
& \rightarrow_\beta (\lambda z. zMN) \ F \\
& \rightarrow_\beta \text{FMN} \\
& \rightarrow_\beta N
\end{align*}
\]
Null and null?

\[
\begin{align*}
null & \equiv \lambda x. T \\
null? & \equiv \lambda x. (x \ y \ z \ F)
\end{align*}
\]

\[
\begin{align*}
null? \ null & \rightarrow \lambda x. (x \ y \ z \ F) (\lambda x. T) \\
& \rightarrow_\beta (\lambda x. T)(\lambda y. z. F) \\
& \rightarrow_\beta T
\end{align*}
\]

Counting

\[
\begin{align*}
0 & \equiv \ null \\
1 & \equiv \ cons \ F \ 0 \\
2 & \equiv \ cons \ F \ 1 \\
3 & \equiv \ cons \ F \ 2 \\
\ldots
\end{align*}
\]

\[
\begin{align*}
succ & \equiv \lambda x. cons \ F \ x \\
pred & \equiv \lambda x. cdr \ x
\end{align*}
\]

Arithmetic

\[
\begin{align*}
\text{zero?} & \equiv \ null? \\
n\text{succ} & \equiv \lambda x. \ cons \ F \ x \\
n\text{pred} & \equiv \lambda x. F \\
n1 & \equiv (\lambda x. F) \ cons \ F \ \null \\
& \rightarrow_\mu (\cons \ F \ \null) \ F \\
& \rightarrow_\mu (\lambda xy. z. F \ \null) \ F \\
& \rightarrow_\mu (\lambda z. F \ \null) \ F \\
& \rightarrow_\mu F \ \null \\
& \rightarrow_\mu \ null \\
& \equiv \ 0
\end{align*}
\]

Lambda Calculus is a Universal Computer

\[
\begin{align*}
42 & \equiv \lambda xy. (\lambda z. x y) \ \lambda x y. \ y \lambda x. (\lambda z. x y) \ \lambda x y. \ y \\
& \lambda x. (\lambda z. x y) \ \lambda x y. \ y \lambda x. (\lambda z. x y) \ \lambda x y. \ y \\
& \lambda x. (\lambda z. x y) \ \lambda x y. \ y \lambda x. (\lambda z. x y) \ \lambda x y. \ y \\
& \lambda x. (\lambda z. x y) \ \lambda x y. \ y \lambda x. (\lambda z. x y) \ \lambda x y. \ y \\
& \lambda x. (\lambda z. x y) \ \lambda x y. \ y \lambda x. (\lambda z. x y) \ \lambda x y. \ y \\
& \lambda x. (\lambda z. x y) \ \lambda x y. \ y \lambda x. (\lambda z. x y) \ \lambda x y. \ y \\
& \lambda x. (\lambda z. x y) \ \lambda x y. \ y \lambda x. (\lambda z. x y) \ \lambda x y. \ y \\
& \lambda x. (\lambda z. x y) \ \lambda x y. \ y \lambda x. (\lambda z. x y) \ \lambda x y. \ y \\
& \lambda x. (\lambda z. x y) \ \lambda x y. \ y \lambda x. (\lambda z. x y) \ \lambda x y. \ y \\
& \lambda x. (\lambda z. x y) \ \lambda x y. \ y \lambda x. (\lambda z. x y) \ \lambda x y. \ y
\end{align*}
\]
Way to Keep Going

\[(\lambda f. ((\lambda x.f(xx)) (\lambda x.f(xx)))) (\lambda z.z)\]
\[\rightarrow_\beta (\lambda x.(\lambda z.z)(xx)) (\lambda x.(\lambda z.z)(xx))\]
\[\rightarrow_\beta (\lambda z.z) (\lambda x.(\lambda z.z)(xx)) (\lambda x.(\lambda z.z)(xx))\]
\[\rightarrow_\beta (\lambda z.z) (\lambda x.(\lambda z.z)(xx)) (\lambda x.(\lambda z.z)(xx))\]
\[\rightarrow_\beta (\lambda z.z) (\lambda z.z) (\lambda x.(\lambda z.z)(xx)) (\lambda x.(\lambda z.z)(xx))\]
\[\rightarrow_\beta (\lambda z.z) (\lambda z.z) (\lambda x.(\lambda z.z)(xx)) (\lambda x.(\lambda z.z)(xx))\]

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

- 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