Lecture 37: A Universal Computer

Turing Machine: FSM + Infinite Tape
- **Start:**
  - FSM in Start State
  - Input on Infinite Tape
  - Tape head at start of input
- **Step:**
  - Read current input symbol from tape
  - Follow transition rule from current state on input
    - Write symbol on tape
    - Move L or R one square
    - Update FSM state
- **Finish:** Transition to halt state

Adding
- **Input on tape:**
  
  \[
  \ldots \#n_1n_2\ldots n_k\#n_1\ldots n_m\ldots\# \ldots
  \]
  - Number represented in binary
- **Output:**
  
  \[
  \ldots \#r_1\#r_2\ldots \#
  \]
  where \( r = n + m \)

Can we implement addition with a TM?

Adder TM (Start)
Lecture 37: Universal Computing Machines

Turing Machine

\[
\text{TuringMachine} ::= < \text{Alphabet}, \text{Tape}, \text{FSM} > \\
\text{Alphabet} ::= \{ \text{Symbol}^* \} \\
\text{Tape} ::= < \text{LeftSide}, \text{Current}, \text{RightSide} > \\
\text{OneSquare} ::= \text{Symbol} | \# \\
\text{Current} ::= \text{OneSquare} \\
\text{LeftSide} ::= [ \text{Square}^* ] \\
\text{RightSide} ::= [ \text{Square}^* ] \\
\text{FSM} ::= < \text{States}, \text{TransitionRules}, \text{InitialState}, \text{HaltingStates} > \\
\text{States} ::= \{ \text{StateName}^* \} \\
\text{InitialState} ::= \text{StateName} \\
\text{HaltingStates} ::= \{ \text{StateName}^* \} \\
\text{TransitionRules} ::= \{ \text{TransitionRule}^* \} \\
\text{TransitionRule} ::= < \text{StateName}, \text{OneSquare}, \text{StateName}, \text{OneSquare}, \text{Direction} > \\
\text{Direction} ::= \text{L}, \text{R} \\
\]

Everything to left of \text{LeftSide} is \#, Everything to right of \text{RightSide} is \#.

Example Turing Machine

\[
\text{TuringMachine} ::= < \text{Alphabet}, \text{Tape}, \text{FSM} > \\
\text{Alphabet} ::= \{ (, ), X \} \\
\text{Tape} ::= \{ \#, 1, #, X, R \} \\
\text{FSM} ::= < \{ 1, 2, \text{HALT} \}, \{ (, ), X \}, \{ (, ) \}, \{ 1, 2 \} > \\
\text{InitialState} ::= 1 \\
\text{HaltingStates} ::= \{ \text{HALT} \} \\
\text{TransitionRules} ::= \{ < 1, 1, \#, R >, < 1, \#, \text{HALT}, 1, \#, R >, < 1, \#, \#, R >, < 1, \#, R >, < 2, 1, X, R >, < 2, \#, \text{HALT}, 0, \#, R >, < 2, \#, \#, L, R > \}
\]

Enumerating Turing Machines

- Now that we've decided how to describe Turing Machines, we can number them
  - TM-5023582376 = balancing parens
  - TM-57239683 = even number of 1s
  - TM-3523796834721038296738259873 = Photomosaic Program
  - TM-3672349872381692309875823987609823712347823 = WindowsXP

Not the real numbers – they would be much bigger!
Universal Turing Machine

- Number of TM
- Input Tape
- Output Tape for running TM-P in tape I

Can we make a Universal Turing Machine?

Yes!

- People have designed Universal Turing Machines with
  - 4 symbols, 7 states (Marvin Minsky)
  - 4 symbols, 5 states
  - 2 symbols, 22 states
  - 18 symbols, 2 states
  - 2 states, 5 symbols (Stephen Wolfram)
- No one knows what the smallest possible UTM is

Church-Turing Thesis

- Any mechanical computation can be performed by a Turing Machine
- There is a TM-n corresponding to every computable problem
- We can any “normal” (classical mechanics) computer with a TM
  - If a problem is in polynomial time on a TM, it is in polynomial time on an iMac, Cray, Palm, etc.
  - But maybe not a quantum computer! (later class)

Universal Language

- Is Scheme/Charme/Python as powerful as a Universal Turing Machine?
  - Yes: show we can simulate a UTM with a Scheme program
- Is a Universal Turing Machine as powerful as Scheme/Charme/Python?
  - Can we simulate a Scheme interpreter with a TM?

Complexity in Scheme

- Special Forms
  - if, cond, define, etc.
- Primitives
  - Numbers (infinitely many)
  - Booleans: #t, #f
  - Functions (+, -, and, or, etc.)
- Evaluation Complexity
  - Environments (more than ½ of our eval code)
  - Can we get rid of all this and still have a useful language?
Lecture 37: Universal Computing Machines

λ-calculus

Alonzo Church, 1940
(LISP was developed from λ-calculus, not the other way round.)

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

What is Calculus?

• In High School:
  \[ \frac{d}{dx} x^n = nx^{n-1} \] [Power Rule]
  \[ \frac{d}{dx} (f + g) = \frac{d}{dx} f + \frac{d}{dx} g \] [Sum Rule]

Calculus is a branch of mathematics that deals with limits and the differentiation and integration of functions of one or more variables...

Real Definition

• A calculus is just a bunch of rules for manipulating symbols.
• People can give meaning to those symbols, but that’s not part of the calculus.
• Differential calculus is a bunch of rules for manipulating symbols. There is an interpretation of those symbols corresponds with physics, slopes, etc.

Lambda Calculus

• Rules for manipulating strings of symbols in the language:
  \[ \text{term} = \text{variable} \]
  | term term
  | (term)
  | λ variable . term
• Humans can give meaning to those symbols in a way that corresponds to computations.

Why?

• Once we have precise and formal rules for manipulating symbols, we can use it to reason with.
• Since we can interpret the symbols as representing computations, we can use it to reason about programs.

Evaluation Rules

α-reduction (renaming)
\[ \lambda y. M \rightarrow_{\alpha} \lambda y. (M [y \alpha \beta]) \]
where \( \beta \) does not occur in \( M \).

β-reduction (substitution)
\[ (\lambda x. M)N \rightarrow_{\beta} M [ x \alpha N ] \]
Charge

- Project Descriptions due before midnight tonight
- Exam 2 due Friday at 12:02 pm (beginning of class)
- Friday’s class: student talks about research and industry