Lecture 19: Other Models of Computability

CSCI 101
Spring, 2019

Kim Bruce

Create Universal TM

- Input:
  - program inputString
  - where program is TM description
- Output
  - result of executing program on inputString

Hmwk: Dove-Tailing

- Suppose want to find if any input s.t. M stops on w.
- If just go through all inputs one at a time, may get stuck if some computation doesn't finish.
- Dove-tailing is sneaky method to make sure we have a chance to try every possibility without getting stuck in an infinite computation.

Trying all inputs

- List all possible inputs w0, w1, w2, ...
  - countably infinite
- Plan:

Dove-tailing

  - Run M for one step on w0
  - Run M for two steps on each of w0, w1
  - Run M for three steps on each of w0, w1, w2
  - ...
  - If M steps in 134,543 steps on w 123 then will eventually find it. If accepts nothing then run forever.

Finish Universal ™

Encoding TM

- Showed how to encode TM description using binary to represent states and input symbols.

Encoding Example

Consider M = ({s, q, h}, {a, b, c}, {❑, a, b, c}, δ, s, {h}):

state	symbol	δ
s	❑	(*q*,❑,, →)
s	a	(*s*,b,→)
s	b	(*q*,a, ←)
s	c	(*q*,b, ←)
q	❑	(*s*,a,, →)
q	a	(*q*,b,→)
q	b	(*q*,b, ←)
q	c	(*h*,a, ←)

state/symbol	representation
s	q00
q	q01
h	h10
❑	a00
a	a01
b	a10
c	a11

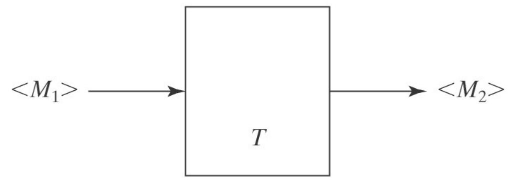
<M> = (q00,a00,q01,a00,→), (q00,a01,q00,a10,→),
(q00,a10,q01,a01, ←), (q00,a11,q01,a10,←),
(q01,a00,q00,a01,→), (q01,a01,q01,a10,→),
(q01,a10,q01,a11,←), (q01,a11,h11,a01,←)

Enumerating TMs

- Theorem: There exists an infinite lexicographic enumeration of:
  1. All syntactically valid TMs.
  2. All syntactically valid TMs with specific input alphabet Σ.
  3. All syntactically valid TMs with specific input alphabet Σ and specific tape alphabet Γ.

Side note

- Can talk about algorithmically modifying TM's:



- Example: Make an extra copy of input and then run <M> on new copy.

Specifying UTM

- On input <M, w>, U must:
  - Halt iff M halts on w.
  - If M is a deciding or semideciding machine, then:
    - If M accepts, accept.
    - If M rejects, reject.
  - If M computes a function, then U(<M, w>) must equal M(w).

Implementation

- ... as a 3-tape TM:
  - Tape 1: M's tape.
  - Tape 2: <M>, the "program" that U is running.
  - Tape 3: M's state.

Implementation

	<M----	----	----	----M,	w----	----	----w>		
	1	0	0	0	0	0	0		
❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
	1	0	0	0	0	0	0		
	❑	❑	❑	❑	❑	❑	❑		
	1	0	0	0	0	0	0		

- Initialization of U:
  - Copy <M> onto tape 2 (and erase from tape 1).
  - Look at <M>, figure out # of states, and write the encoding of state s on tape 3.
- After initialization:

	❑	❑	❑	❑	<w----	----	----w>		
	0	0	0	0	1	0	0		
❑	<M----	----	----	----M>	❑	❑	❑	❑	❑
	1	0	0	0	0	0	0		
	q	0	0	0	❑	❑	❑		
	1	❑	❑	❑	❑	❑	❑		

Simulation

- Simulate the steps of M :
  1. Until M would halt do:
     1.1. Scan tape 2 for a transition matching the current state, input pair.
     1.2. Perform the associated action, by changing tapes 1 and 3 (state). If necessary, extend the tape.
     1.3. If no matching quintuple found, halt. Else loop.
  2. Report the same result M would report.
- How long does U take?

Universal FSM??

- Can we write FSM, M, that accepts
  L = {<F, w> : F is a FSM, and w $\in$ L(F) }?

How big is UTM?

- The first constructed by Turing.
- Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine
- Minsky (1960): 7-state 6-symbol machine.
- Watanabe (1961): 8-state 5-symbol machine.
- Minsky (1962): 7-state 4-symbol machine.
- Rogozhin (1996) 4-state 6-symbol machine
- Wolfram & Reed(2002): 2-state 5-symbol machine.
- Smith & Wolfram(2007): 2-state 3-symbol machine.
- No 2-state 2-symbol UTM exists.

What is more powerful?

- Are we done? Is there more powerful model?
- Lots of languages we can't recognize with TM's
  - Countably infinite number of Turing machines since we can lexicographically enumerate all the strings that correspond to syntactically legal Turing machines.
  - There is an uncountably infinite number of languages over any nonempty alphabet.
  - Many more languages than Turing machines.

Historical Context

- David Hilbert's lecture to 1900 International Congress of Mathematics in Paris.
- Presented 23 problems to influence course of 20th century mathematics (only 10 at meeting)

CS & Logic Relevant:

1. Continuum hypothesis: Is there a set with cardinality between that of integers and reals?

2. Prove that the axioms of arithmetic are consistent.

10. Find an algorithm to determine whether a given polynomial Diophantine equation with integer coefficients has an integer solution.

All Had Surprising Results

1. Continuum hypothesis: Independent of axioms of set theory (K. Gödel & P. Cohen)

2. Consistency of arithmetic: Not provable from within arithmetic (K. Gödel)

10. Find an algorithm to determine solutions to Diophantine equations: Undecidable. (Y. Matiyasevich, J. Robinson).

Hilbert Again

- Entscheidungsproblem posed by David Hilbert in 1928.
  - Find an algorithm that will take as input a description of a formal language and a mathematical statement in the language and produce as output either "True" or "False" according to whether the statement is true or false.
- If find an algorithm, then no problem, but ...
  - how do you show there is no such algorithm?

What is an algorithm?

- Alonzo Church (w/S. Kleene) 1936: λ-calculus
- Alan Turing 1936: Turing machine
- Negative answer to the Entscheidungsproblem
  - Church 1935-36
  - Turing (independently) 1936-37 -- reducing to Halting Problem
  - Both influenced by Gödel's proof of incompleteness of predicate logic & Number Theory

Church-Turing Thesis

- All formalisms powerful enough to describe everything we think of as a computational algorithm are equivalent.
- Can't prove it because don't have a list of all possible formalisms.
  - But have shown it for all proposed formalisms.

Proposed Formal Models

- Modern computers (with unbounded memory)
- Lambda calculus
- Partial recursive functions
- Tag systems (FSM plus FIFO queue)
- Unrestricted grammars:
  - $aSa \rightarrow B$

Proposed Formal Models

- Post production systems
- Markov algorithms
- Conway's Game of Life
- One dimensional cellular automata
- DNA-based computing
- Lindenmayer systems
- While language

Partial Recursive Functions

- Are built up from:
  - Constant fcns: for each n, k, $c_n(x_1,\ldots,x_k) = n$
  - Successor: $S(x) = x + 1$
  - Projection: $P^k{}_i(x_1,\ldots,x_k) = x_i$
- Using composition:
  - If $g_1(x_1,\ldots,x_k),\ldots, g_m(x_1,\ldots,x_k)$, $h(x_1,\ldots,x_m)$ are fcns, define $f(x_1,\ldots,x_k) = h(g_1(x_1,\ldots,x_k),\ldots, g_m(x_1,\ldots,x_k))$

Partial Recursive Functions

- Primitive recursion:
  - Given the k-ary function $g(x_1,\ldots,x_k)$ and k+2 -ary function $h(y,z,x_1,\ldots,x_k)$, define $f(y, x_1,\ldots,x_k)$ where
    - $f(0,x_1,\ldots,x_k) = g(x_1, \ldots, x_k)$
    - $f(y+1,x_1,\ldots,x_k) = h(y, f(y,x_1,\ldots,x_k), x_1, \ldots, x_k)$
- Minimization:
  - Given $f(y, x_1,\ldots,x_k)$, define $h(x_1,\ldots,x_k) = \mu z.(f(z, x_1,\ldots,x_k) = 0)$ *where* $\mu z.(R(z,y))$ is the least $z \geq 0$ s.t. $R(z,y)$.

Example

- Informally:
  - plus(0,n) = n
  - plus(m+1,n) = S(plus(m,n))
- More formally:
  - $plus(0,n) = P^1{}_1(n)$
  - plus(m+1,n) = h(m,plus(m,n),n) where
    - $h(x,y,z) = S(P^3{}_2(x,y,z))$

Defining Functions

- In math and LISP/Scheme/Racket:
  - f(n) = n * n
  - (define (f n) (* n n))
  - (define f (<u>lambda (n) (* n n)</u>))
  - λ n -> n * n *in SML*
- In lambda calculus
  - λn. n * n *anonymous functions*
  - ((λn. n * n) 12) (which evaluates to 144)