Lecture 29: Universal Turing Machines

CSCI 81
Spring, 2012

Kim Bruce

TM Programming Tips

- Divide work into different phases/subroutines
- Controller has arbitrarily large“finite memory”.
- Squares can be “marked” and “unmarked” in finitely many ways.
- Take advantage of TM extensions.

TM Variants

- Showed last time that various strengthenings don't help in computability (though do in speed).
  - Adding extra tapes
  - Making non-deterministic

TM's

- So far built “dedicated machines”.
  - Only run one program
  - Specified by transition on states
- Can TM's be general-purpose computers?
  - Can we create a “universal” TM with an arbitrary program and have it execute the program?
  - What kind of program?

UTM

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

Defining UTM

- Two steps:
  - Define encoding for arbitrary TM
  - Describe operation when given input of TM M and input string w

Encoding TM

- States: Let i = ⌈log2(|K|)⌉
- Number states sequentially as i bit numbers letting start state be 0…0.
- For each state t, let t' be its associated number.
  - If t is halting state y, assign code yt'
  - If t is halting state n, assign code nt'
  - If t any other state, assign code qt'

Example Encoding States

- Suppose M has 9 states. $\lceil \log_2(9) \rceil = 4$
- Let s' = q0000,
- Remaining states (where y is 3 and n is 4):
  - q0001, q0010, y0011, n0100, q0101, q0110, q0111, q1000

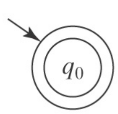
Encoding Tape Alphabet

- Encode in form ak where k is j = ⌈log2(|Γ|)⌉ bit number
- Example: Γ = {❑, a, b, c}. j = 2.
  - ❑ ⇒ a00
  - a ⇒ a01
  - b ⇒ a10
  - c ⇒ a11

Transitions

- The transitions:
  - (state, input, state, output, move)
- Example: (q000,a000,q110,a000,→)
- Specify s as q000.
- Specify M as a list of transitions.

Special Case



Encode as (q0)

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

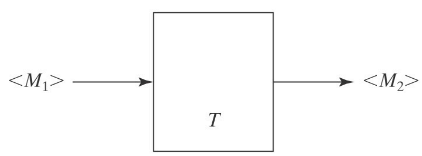
- Theorem: There exists an infinite lexicographic enumeration of:
  1. All syntactically valid TMs.
  2. All syntactically valid TMs with specific input alphabet Σ.
  3. All syntactically valid TMs with specific input alphabet Σ and specific tape alphabet Γ.

Proof

- Fix Σ = {(,), a, q, y, n, 0, 1, comma, →, ←}, ordered as listed. Then:
  - Lexicographically enumerate the strings in Σ*.
  - As each string s is generated, check to see whether it is a syntactically valid Turing machine description. If it is, output it.
  - To restrict enumeration to symbols in Σ & Γ, check, in step 2, that only alphabets of appropriate sizes allowed.
  - Can now talk about the ith Turing machine

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.
  - 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 Diophantine solution: Undecidable. (Y. Matiyasevich, J. Robinson).

Solns to 1 & 10 resulted in awards of Fields Medals

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

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