Lecture 17: Turing Machine Variants

CSCI 101
Spring, 2019

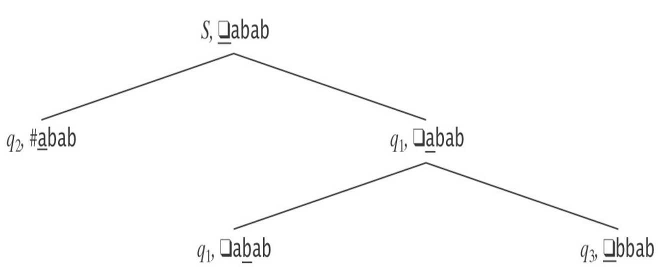
Kim Bruce

Last Time

- Showed multi-tape TM's have no more power than single tape (though can be more efficient)
- Look at other variations.

Nondeterminism

- A nondeterministic TM is a six-tuple (K, Σ, Γ, Δ, s, H) where Δ is a subset of:
  $((K - H) \times \Gamma) \times (K \times \Gamma \times \{\leftarrow, \rightarrow\})$



Deciding

- Let M = (K, Σ, Γ, Δ, s, {y, n}) be a nondeterministic TM, and w be element of Σ*.
  - M accepts w iff at least one of its computations accepts.
  - M rejects w iff all of its computations reject.
  - M decides a language L ⊆ Σ* iff, ∀w:
    - There is a finite number of paths that M can follow on input w,
    - All of those paths halt, and
    - w ∈ L iff M accepts w.

Nondeterministic Programming

- L = {w ∈ {0, 1}* : w is the binary encoding of a *composite* number}. M decides L by doing the following on input w:

non-prime

  - Nondeterministically choose two positive binary numbers such that: 2 ≤ |p|, |q| ≤ |w|. Write them on the tape, after w, separated by ;
    - ❑110011;111;1111❑❑
  - Multiply p and q and put the answer, A, on the tape, in place of p and q.
    - ❑110011;1011111❑❑
  - Compare A and w. If equal, go to y. Else go to n.

Semi-Deciding

- Let M = (K, Σ, Γ, Δ, s, H) be a nondeterministic TM.
- We say that M semi-decides a language L ⊆ Σ* iff for all w ∈ Σ*: w ∈ L iff (s, ❑w) yields at least one accepting configuration.

Example

- Let L = {descriptions of TMs that halt on at least one string}.
  - Let <M> mean the string that describes some TM M.
- S semi-decides L as follows on input <M>:
  - Nondeterministically choose a string w in Σ_M* and write it on the tape.
  - Run M on w
- See later that semi-deciding is best we can do.

Non-deterministic Functions

- M computes a function f iff, ∀w ∈ Σ* :
  - All of M's computations halt, and
  - All of M's computations result in f(w).

Review

- Non-determinism not more powerful for FSM's
  - Subset construction
- PDA's?
- Which is TM more like?

Non-determinism Not More Powerful!

- Theorem: If a nondeterministic TM M decides or semi-decides a language, or computes a function, then there is a standard TM M' deciding or semi-deciding the same language or computing the same function.
- Proof: (by construction). Must do separate constructions for deciding/semi-deciding and for function computation.

Proof Sketch

- Try all possible computation paths
- Because computations may be infinite, need to do breadth first search
- Use 3 tapes
  - 1st for input (never modified)
  - 2nd for computations
  - 3rd for string specifying which of possible instructions to take

Proof Sketch

- Let b be largest number of possible transitions from any configuration.
- Encode computation of length n as n-digit number written in base b:
  - E.g. If b = 3, then 10221 encodes computation of length 5.
- Let E be TM program that takes a number m in base b and returns m+1 in base b.

Computation

- Start with input w on tape 1, 0 on tape 3
- Loop:
  - Copy input w from tape 1 to tape 2
  - Using number n on tape 3 to select steps to take in simulating run on w of length $\log_b$ n.
  - Use E to increase number of tape 3 by 1

Simulating

- Semi-deciding is easy. If any path accepts then stop and accept.
- Deciding is trickier as must be able to reject
  - If any path halts and accepts then accept
  - If tried all paths until they halt and then reject then reject
    - How can you tell?

Deciding

- Write value notHalted on tape 1 telling if any paths of current length haven't halted. Initially false.
  - If path halts and accepts then stop and accept
  - If path halts and rejects then do nothing
  - If path doesn't halt then set notHalted to true
- When increase length of guide string, check value of notHalted.
  - If false, then halt and reject
  - If true then reset to false and continue simulation

Other Variants

- One-way vs two-way infinite tape
- Two dimensional tape
- Multiple-track tape

TM Programming Tips

- Divide work into different phases/subroutines
- Controller has arbitrarily large“finite memory”
  - ... but it can’t depend on the size of the input!
- Squares can be “marked” and “unmarked” in finitely many ways.
- Take advantage of TM extensions.

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

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

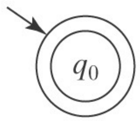
Encoding Tape Alphabet

- 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,←)

Lexicographic Order

- of strings in Σ* as defined in text:
  - if |u| < |v|, then u < v
  - if |u| = |v|, then u < v if u precedes v in dictionary order
- Example of lexicographic order over {0,1}*
  - ε, 0, 1, 00, 01, 10, 11, 000, ...
- Feature: Every string occurs in finite position
  - Dictionary order: ε, 0, 00, 000, ...
  - Never get to 1!

Enumerating TMs

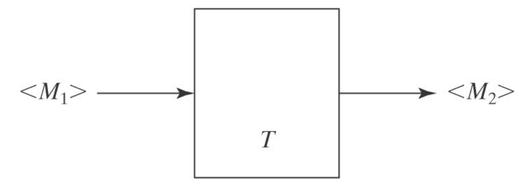
- Theorem: There exists an infinite lexicographic enumeration of:
  1. All syntactically valid TMs.
  2. All syntactically valid TMs with specific input alphabet Σ.
  3. All syntactically valid TMs with specific input alphabet Σ and specific tape alphabet Γ.

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.