CISC 4090
Theory of Computation

Decidability

Professor Daniel Leeds
dleeds@fordham.edu
JMH 332

"Turing recognizable" vs. "Decidable"

Language is **Turing recognizable** if some Turing machine recognizes it

- Also called "recursively enumerable"

Machine that halts on all inputs is a **decider**. A decider that recognizes language L is said to **decide** language L

Language is **Turing decidable,** or just **decidable**, if some Turing machine decides it

2

Not all problems can be solved

- Good to know when you might not find an answer
- Get perspective on limits of computation

3

Decidable problems for regular languages

- Does DFA D accept string s?
- Is L(D) of DFA empty?
- Are two DFAs D1 and D2 equivalent?

Specify DFA on input TM,
determine control algorithm to run DFA specified on tape

4

Arbitrary DFA D accepts string w

Language: A_{DFA}={(D,w) | D is DFA that accepts w}

Theorem: A_{DFA} is decidable
Proof idea:

~StartQ#AcceptQ#δ#CurrentState#w~

- Define machine M that simulates D on w
- If simulation ends in an accept, accept; else, reject

Note: control states in M cannot be states in D
M needs to run arbitrary D

5

A_{DFA} decider Proof Outline

DFA D described as string: 5-tuple

Use marks on tape to track
- current state in simulated D
- current symbol read from w

Implement transition function of D for current D state and input w
- D's transition δ is **different** from TM M's transition δ

6

Arbitrary NFA N accepts string w

Language: A_{NFA}={(N,w) | N is DFA that accepts w}

Theorem: A_{NFA} is decidable
Proof idea 1:

~StartQ#AcceptQ#δ#CurrentState#w~

- Define machine M that simulates N on w
  - May use extra tape(s) to keep track of breadth first search for multiple transition options
- If simulation ends in an accept, accept; else, reject

7

Arbitrary NFA N accepts string w

Language: A_{NFA}={(N,w) | N is DFA that accepts w}

Theorem: A_{NFA} is decidable
Proof idea 2:

~StartQ#AcceptQ#δ#CurrentState#w~

- Design transducer that converts NFA to equivalent DFA
  - Decidable: finite number of steps: breadth first search of DFA states based on transitions from start state and from resulting states; at most 2^k states if $|Q_{NFA}|=k$
- Ran A_{DFA} on the equivalent DFA we just defined above – decidable as proven in previous slide

8

Arbitrary DFA D accepts **no** strings

E_{DFA}={ D | D is DFA with L(D)={} } is decidable language

Proof idea:

- Is there any way to reach accept from start?
- Think of graph-marking

Proof

- Mark start state of DFA D
- Repeat until no new states
  - Mark any state that past-marked states transition to
- If an accept state is marked, REJECT; else, accept

9

Two DFAs are equivalent

EQ_{DFA} = {(A,B) | A and B are DFAs and L(A)=L(B)} is decidable language

Proof idea:

- Construct new DFA C from A and B; C accepts only strings accepted by either A or B, but not both
- Check if C's language is empty (last slide)

10

A_{CFG} is decidable – Proof

For CFG G and string w, determine if G generates w

Idea 1: Simulate G to go through all derivations

- May never terminate

Idea 2: Note |w|=n; 2n-1 steps from CNF rules to each string
Produce all words of lengths n

- Breadth-first search of finite depth is fixed

12

B_{CFG} is a decidable language (or NE_{CFG})

- For CFG G, determine if there is any terminal string generated by G

13

B_{CFG} is a decidable language

- For CFG G, determine if there is any terminal string generated by G
- Mark all variables that generate terminals
- Repeated loop:
- Mark all variables that have only previously-marked variables on its rules right sides
- If mark S, ACCEPT; otherwise reject

<u>S</u> -> <u>AB</u>
<u>A</u> -> An | <u>x</u>
<u>B</u> -> yB | <u>d</u>

14

EQ_{CFG} is not a decidable language

- Regular expressions closed under complement and intersection
- CFLs not closed under complement and intersection
- We will prove non-decidable languages later

16

The Halting Problem

Key theorem to theory of computation

Addressing unsolvable problems

Unsolvable: Software verification

- For arbitrary computer program P and precise specification of program's behavior S, determine if P fulfills S

17

Halting Problem specified

A_{TM} = {(M,w) | M is a TM and M accepts w}

- If M loops forever on w, our TM for A_{TM} must reject w

- This problem is Turing recognizable, but not decidable!

18

Contradiction

Assume A_{TM} is decidable

H decides A_{TM}

- Input <M,w> causes H to accept if M accepts w, otherwise H rejects

Define a TM D that calls H on <M,<M>>, then outputs opposite answer to H

- D rejects if M accepts <M>; D accepts if M does not accept <M>

Run D on itself

- D(<D>) = accept if D does not accept <D>; reject if D accepts <D>

Contradiction!

Detour: Cantor diagonalization

Comparing sizes of two infinite sets

- What is larger: set of even positive integers or set of all strings in (0U1)*

Diagonalization: two sets have same size if each element of set A can be compared with one element of set B

From CISC 1400: Can you define bijection from set A to set B?

Example pairing

N = {1,2,3,4,…} and E={2,4,6,8,…}

- N and E have "same size" because there exists bijection from N to E
- f(x)=2x

Set is **countable** if either it is finite or if it has same size as N

Q is countable

Let Q = {m/n: m,n ∈ N}, positive rational numbers

Follow diagonal, skipping redundant values

1/1	1/2	1/3	1/4	1/5	1/6
2/1	2/2	2/3	2/4	2/5	2/6
3/1	3/2	3/3	3/4	3/5	3/6
4/1	4/2	4/3	4/4	4/5	4/6
5/1	5/2	5/3	5/4	5/5	5/6
6/1	6/2	6/3	6/4	6/5	6/6

Concatenating infinite set of finite lists produces countable list

Take countable steps along diagonal line to reach each number in Q

Real numbers are uncountable

Real numbers have infinite number of decimal places

Proving uncountability
- Presume we have a list of *n* real numbers
- Generate new real number x **not** in current list
  - Pick i^{th} decimal value of x to be different from i^{th} decimal value of element i in list of real numbers
- At end, x will not be in list

R(1)	1.<u>5</u>32532
R(2)	0.3<u>5</u>2144
R(3)	5.24<u>4</u>525
R(4)	9.327<u>4</u>31
R(5)	5.3663<u>2</u>4
R(6)	4.45932<u>2</u>
⋮	⋮
x	3.646311

Uncountability implications

There are uncountably many languages

There are countably many Turing machines

Some languages are not Turing recognizable

Are these statements true?

"There are countably many Turing machines"

Each TM is captured by finite string <M>$\in \Sigma^*$
- Σ^* is countable – add number of strings of length 0, length 1, length 2, … (like Q)

"There are uncountably many languages"

Represent L as binary sequence
- 1 for each accepted string, 0 for each reject string
- Infinite number of strings – infinite sequence of 0/1s
- Set of possible binary sequences is uncountable (like R)

"Some languages are not Turing decidable"

Set of TMs is countable

Set of Languages is uncountable

Each TM has one language

Some languages not recognized by any TM

Back to the Halting Problem

A_{TM}={<M,w> | M is a TM and accepts w}

- Proof by contradiction
- **Consider diagonalization**

Implications

A_{TM}={(M,w) | M is a TM and M accepts W} is **not** decidable

Some algorithms are decidable

A_{TM} **is** Turing recognizable – just **simulate** M on machine

Co-Turing Recognizable

Language is co-Turing recognizable if it is the complement of a Turing-recognizable language

Theorem: Language is decidable if it is Turing-recognizable and co-Turing recognizable

Thus, for any undecidable language L, either L or L' is not Turing-recognizable

- Is A_{TM}' Turing-recognizable?

Reducibility

If A reduces to B, solution to B will solve A
Example: A: Navigate NYC   B: Reading a map

If A reduces to B

- A is no harder than B
- A could be easier than B

Reduction and decidability

If A is reducable to B and B is decidable

- A is decidable

If A is reducible to B and A is undecidable

- B must be undecidable

$HALT_{TM}$ is undecidable

We can reduce A_{TM} (TM **accepts** w) to $HALT_{TM}$ (TM **halts** on w)

A_{TM} is undecidable, thus $HALT_{TM}$ is undecidable

Proof by contradiction:

- Assume $HALT_{TM}$ is decidable – TM R
- Use R to construct TM S to decide A_{TM}
- S definition:
  - If R **does not indicate halting** for <M,w>, reject w
  - If R **does indicate halt**, simulate M on w