COMPSCI 501: Formal Language Theory

Lecture 30: Midterm Review

Marius Minea
marius@cs.umass.edu

University of Massachusetts Amherst

8 April 2019

Turing Machines, Recognizability, Decidability

A Turing recognizable (by M_1) and $\bar{A}$ Turing recognizable (by M_2)
$\implies$ A decidable

run M_1 and M_2 in lockstep, see which halts first

Enumerating

Turing-recognizable $\Leftrightarrow$ (recursively) enumerable

run for k steps on $s_1, s_2, \dots s_k$ (dovetailing)

Decidable $\Leftrightarrow$ enumerable in lexicographical order

Problems for Recognizers

Acceptance:
$A_M = \{\langle M, w\rangle | M$ is a machine that accepts $w\}$

Emptiness
$E_M = \{\langle M\rangle | M$ is a machine with $L(M) = \emptyset\}$

Universality
$ALL_M = \{\langle M\rangle | M$ is a machine with $L(M) = \Sigma^*\}$

Equivalence
$EQ_M = \{\langle A, B\rangle | A, B$ are machines with $L(A) = L(B)\}$

All decidable for DFA/NFA/REX (convert to minimized DFA)

A_{CFG}, E_{CFG} decidable, ALL_{CFG}, EQ_{CFG} not decidable

A_{TM}, $HALT_{\mathsf{TM}}$, E_{TM}, etc. not decidable

Proving Undecidability of a Language L

Diagonalization (directly)

e.g., for proving A_{TM} undecidable

Reducing from A_{TM}. Example: E_{TM}.

Assume decider D for E_{TM}, build decider for A_{TM}.

Construct a TM M_1 that will either have an empty language or not, depending on whether M accepts w.
won't ever run M_1, but feed as input to D

On input x:
if $x \neq w$, *reject*
otherwise, run A on w $(= x)$, *accept* if A does

Thus, $L(M_1) = \{w\}$ if A accepts w, $\emptyset$ otherwise
Could use D to decide A_{TM}.

More General: Rice's Theorem

Let P be a nontrivial property of a Turing machine:
A *language property*, $L(M_1) = L(M_2) \to (P(M_1) \leftrightarrow P(M_2))$
At least one TM has this property.

Then P is undecidable.

Let MP a Turing machine with that property.
(Assume language nonempty, else pick complement).

Construct M_1 as follows:

On input x:
run A on w, (run forever or *reject* like A does)
run MP on x, *accept* if MP does.

This will either have the same language as MP, or the empty language.
$\implies$ could use decider for P to decide A_{TM}.

Proving a Language is not Turing-recognizable

Similar idea, but reduce from $\overline{A_{\mathsf{TM}}}$.

EQ_{TM} not Turing-recognizable nor co-Turing-recognizable.

On input $\langle M, w\rangle$, construct two machines:
$M_\emptyset$: rejects any input / M_{all}: accepts any input
M_w: accept all/none, according to run of M on w
$M_\emptyset$ not EQ M_w iff M accepts w: $A_{\mathsf{TM}} \le_{\mathsf{m}} \overline{EQ_{\mathsf{TM}}}$, $\overline{A_{\mathsf{TM}}} \le_{\mathsf{m}} EQ_{\mathsf{TM}}$

M_{all} EQ M_w iff M accepts w: $A_{\mathsf{TM}} \le_{\mathsf{m}} EQ_{\mathsf{TM}}$, $\overline{A_{\mathsf{TM}}} \le_{\mathsf{m}} \overline{EQ_{\mathsf{TM}}}$

Mapping reduction: $f(\langle M, w\rangle) = \langle M_\emptyset, M_w\rangle$ or $\langle M_{all}, M_w\rangle$

Homework 4, accepting precisely all strings with even length.

Reduction via Computation Histories

Linear Bounded Automata: A_{LBA} decidable (finite number of configurations), but E_{LBA} is not.

Set of accepting computation histories of a TM can be checked by an LBA.

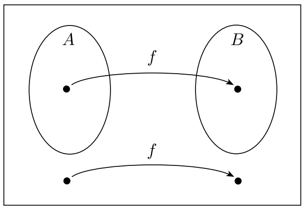
Another use: all strings that are *not* accepting computation histories on a string w.

- can generate with a PDA / CFG $\implies$ ALL_{CFG}= undecidable (deciding $\neq \Sigma^*$ $\Leftrightarrow$ deciding A_{TM}
- can generate via extended regular expressions
  use to prove that equivalence of regular expressions with exponentiation is EXPSPACE-hard.

Mapping Reducibility

Def. A function $f : \Sigma^* \to \Sigma^*$ is a **computable function** if some Turing machine M, on input w, *halts* with just $f(w)$ on tape.

Def. A language A is **mapping reducible** to language B (written $A \leq_{\mathsf{m}} B$) if there is a *computable function* $f : \Sigma^* \to \Sigma^*$ where for every w, $w \in A \Leftrightarrow f(w) \in B$



YES for A means YES for B
NO for A means NO for B

Using Mapping Reducibility

Decidability

If $A \leq_{\mathsf{m}} B$ and B is decidable, then A is decidable.

If $A \leq_{\mathsf{m}} B$ and A is undecidable then B is undecidable.

Turing-recognizability

If $A \leq_{\mathsf{m}} B$ and B is Turing-recognizable, then A is Turing-recognizable.

If $A \leq_{\mathsf{m}} B$ and A is not Turing-recognizable then B is not Turing-recognizable.

Recursion Theorem

A TM can obtain and execute its own description.

Use: e.g., in proofs by contradiction (do something else than description says)

e.g. assume A_{TM} has a decider H
Construct a TM B:
On input w:

1. Obtain own description $\langle B \rangle$
2. Run H on input $\langle B, w \rangle$
3. Do the opposite of H (accept/reject)

Descriptive Complexity

The **minimal description** of a binary string x is the shortest string $\langle M, w \rangle$ where M halts on input w with x on tape.

The **descriptive complexity** (Kolmogorov complexity) is the length of the mininmal description: $\mathsf{K}(x) = |d(x)|$

Def.: A string x is c-**compressible** if $\mathsf{K}(x) \leq |x| - c$.

incompressible = not 1-compressible.
Most strings are close to incompressible.

Incompressible strings are undecidable.

Can only enumerate a finite subset.

Polynomial Verifiers and NP

Def. A **verifier** for a language A is an algorithm V, where

$$A = \{w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}$$

A **polynomial-time verifier** runs polynomial in the length of v.

A language is **polynomially verifiable** if it has a polynomial time verifier.

NP is the class of languages that have polynomial-time verifiers.

equivalent: A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine

NP-completeness

Def. A *language* B is NP-complete iff
1. B is in NP
2. for any A in NP, we have $A \leq_P B$

- If B is NP-complete and $B \leq_P C$, then C is NP-complete
  reduce *known* NP-complete problem B to target C
  reduce target problem C *from* NP-complete problem B
- If B is NP-complete and $B \in$ P, then P = NP

All NP-complete problems are polynomially reducible to one another (the hardest problems in NP)

Time Complexity

Time complexity class TIME($t(n)$) = all languages that are decidable by an $O(t(n))$ (deterministic, single-tape) Turing machine.

A $t(n)$ **multitape** TM has an equivalent $O(t^2(n))$ single-tape TM.

multi-tape *polynomial* ⇒ single-tape *polynomial*

Every $t(n)$ **nondeterministic** TM has an equivalent $2^{O(t(n))}$ deterministic single-tape TM.

nondeterministic *polynomial* ⇒ single-tape *exponential*

Space Complexity

Savitch's Theorem

For any function $f : \mathbb{N} \to \mathbb{R}^+$, where $f(n) \geq n$,
NSPACE($f(n)$) ⊆ SPACE($f^2(n)$)

$\implies$ NPSPACE = PSPACE

PSPACE-completeness: Quantified Boolean Formula = valid ?
also admits log-space reducibility

PATH = $\{\langle G, s, t\rangle \mid G$ is directed graph that has an $s \rightsquigarrow t$ path $\}$
solvable in log space

Model for *sublinear* space: read-only input tape, *work tape* gives space complexity.

Classes L, NL: use constant number of pointers to input tape

Complexity Hierarchies

L ⊆ NL = coNL ⊆ **P** ⊆ NP ⊆ **PSPACE** ⊆ EXPTIME

NL ⊊ PSPACE **P ⊊ EXPTIME**

$f : \mathbb{N} \to N$ that is at least $O(\log n)$ is *space constructible* if there is a $O(f(n))$ space TM that computes $f(n)$ from 1^n

Space Hierarchy Theorem: For *any* space constructible function $f : \mathbb{N} \to \mathbb{N}$, there exists a language that is decidable in $O(f(n))$ space but not $o(f(n))$ space.

SPACE(n^{c_1}) ⊊ SPACE(n^{c_2}) for any real $c_1, c_2 > 0$

$t : \mathbb{N} \to N$ that is at least $O(n \log n)$ is *time constructible* if $t(n)$ is computable in time $O(t(n))$ from 1^n.

Time Hierarchy Theorem: For *any* time constructible function $t : \mathbb{N} \to \mathbb{N}$, there exists a language that is decidable in $O(t(n))$ time but not in time $o(t(n)/\log t(n))$.

TIME(n^{c_1}) ⊊ TIME(n^{c_2}) for any reals $1 \leq c_1 < c_2$

A Classification of NP-Complete Problems

- Optimization problems (find the min or max number of ...)
- Decision problems (is there solution with $\leq k$ or $\geq k$ of ...)

Equivalent in complexity, for a given problem

Satisfiability problems

"Most general": satisfy all constraints

- CIRCUIT-SAT
- SAT
- 3-SAT
- NAE-SAT not-all-equal

Covering Problems

Achieve some global goal with few elements

- Vertex Cover: cover edges with vertices
- Set Cover: cover entire set with subsets
- Hitting Set: cover subsets with elements
- Dominating Set: cover self and neighbor vertices

Packing Problems

Choose many elements while avoiding conflicts

- Independent Set vertices with no edges
- Set Packing non-intersecting subsets

Polynomial

Matching (edges with no common endpoints)
case of bipartite graphs (network flow)

Sequencing problems

- Hamiltonian Path (all nodes)
  reduction to cycle: extra node, connected to all others
- Hamiltonian Cycle
  reduction to path: split a node, add an endpoint to each half
- Traveling Salesman Problem minimum-length tour
  reduce from HAM-CYCLE

Numerical Problems

- Subset-Sum numbers with precise sum

reduce from SAT: construct numbers digit-by-digit

Partitioning / Coloring Problems

- 3-coloring no edge with same-color nodes
- k-coloring

Polynomial

2-coloring (bipartite graph)