Lecture 13 – Intro to Complexity theory

NTIN071 Automata and Grammars

Jakub Bulín (KTIML MFF UK)

Spring 2024

** Adapted from the Czech-lecture slides by Marta Vomlelová with gratitude. The translation, some modifications, and all errors are mine.*

Recap of Lecture 12

- the Diagonal language L_D is not recursively enumerable
- the Universal language L_U, the Universal TM: simulate any M on any w
- recursive languages are closed under complement
- Post's theorem: L recursive iff both $L, \overline{L}$ are RE
- L_U, $\overline{L_D}$ are recursively enumerable but not recursive
- reductions between decision problems
- the Halting problem is undecidable
- (Rice's thm: nontriv. properties of programs are undecidable)
- Undecidable problems about context-free grammars
- Source of undecidability: Post's correspondence problem

Summary of Lecture 13

- time complexity
-

Chapter 5: Intro to Complexity

Time complexity

Asymptotic notation

Big-O notation: Let $f, g : \mathbb{N} \to \mathbb{R}^+$. We say that $f(n) \in O(g(n))$, if there exist $C, n_0 \in \mathbb{N}^+$ such that

$$(\forall n \geq n_0)\ f(n) \leq C \cdot g(n)$$

e.g. $\limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty$. In that case we say that $g(n)$ is an [asymptotic] upper bound [up to a constant multiple] for $f(n)$.

Note: Often the imprecise term 'upper bound' is used; sometimes you will encounter $f(n) = O(g(n))$.

For example, $f(5n^3 + 2n^2 + 22n + 6) \in O(n^3)$ with $n_0 = 10,\ C = 6$.

Little-o notation: $f(n) \in o(g(n))$, if for all $c > 0$ there exists $n_0 \in \mathbb{N}^+$ so that $(\forall n \geq n_0)\ f(n) < c \cdot g(n)$, i.e. $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. Then we say $f(n)$ is [asymptotically] dominated by $g(n)$.

Analogously for $\geq$ instead of $\leq$: Ω, ω.

Classes of time complexity

Definition

Let M be a Turing machine that halts on every input. The time complexity of M is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of computation steps for inputs of length n.

Definition

For $t : \mathbb{N} \to \mathbb{R}^+$, $\mathrm{TIME}(t(n))$ is the class of all languages decidable by a TM of time complexity in $O(t(n))$ (i.e., always halts and for $|w| = n$ correctly answers in at most $O(t(n))$ steps).

NB: Here we mean the standard, single-tape, deterministic TM.

Example

Example ($L = \{0^i 1^i \mid i \geq 0\}$ is in $\mathrm{TIME}(n^2)$)

1. check if the input is $0^i 1^j$, if a 0 follows a 1, reject (time $O(n)$)
2. return to the beginning: hidden in the constant $O(2n) = O(n)$
3. go through the 0s, in time $O(n^2)$
   3.1 rewrite the next 0 to X
   3.2 find the first 1, rewrite to X
   3.3 return to the beginning
4. if no more 0s, check that no more 1s remain and accept (if 1 found, reject) (time $O(n)$)

$\vdots$

Can we do it faster?

Can we do it faster?

Idea: "compare the binary representations of i and j", $\log n$ bits, for each bit need to traverse through the word

Example ($L = \{0^i 1^i \mid i \geq 0\}$ is also in $\mathrm{TIME}(n \log n)$)

1. check if the input is $0^i 1^j$ and even length (time $O(n)$)
2. iterate while there are 0s, in time $O(n \log n)$
   2.1 rewrite every other 0 to X, then every other 1 to X
   2.2 check if the number of remaining 0s+1s is even, if not, reject
3. if no more 0s, check that no more 1s and accept (time $O(n)$)

$\vdots$

Can we do it even faster?

Time complexity and regular languages

Can we do it even faster? Not really.

Theorem

Every language decidable in time $o(n \log n)$ [on a single-tape, deterministic TM] is regular.

[We omit the proof.]

Multi-tape TM

Example (Multi-tape TM for $L = \{0^i 1^i \mid i \geq 0\}$)

- copy 0s to Tape 2
- at first 1, switch state; erase 1 from Tape 1 & 0 from Tape 2
- accept if both tapes are erased

Lemma

Every multi-tape Turing Machine with time complexity $t(n)$ is equivalent to a [single-tape] Turing Machine with time complexity $O(t^2(n))$.

Proof: Simulation of n steps of a k-tape TM can be done in $O(n^2)$ moves since one step takes $4n + 2k$ moves (heads at most $2n$ fields apart, read, write, move head marks). $\square$

Nondeterministic time complexity

The time complexity of a **nondeterministic** Turing machine that always halts is defined analogously: $f(n)$ is the maximum number of steps in **any branch** of the computation tree.

Definition

For $t : \mathbb{N} \to \mathbb{R}^+$, $\mathrm{NTIME}(t(n))$ is the class of all languages decidable by a nondeterm. TM of time complexity in $O(t(n))$.

(An NTM decides L if halts on all inputs and recognizes L.)

Theorem

Any nondeterministic TM of time complexity $t(n) \geq n$, has a deterministic equivalent of time complexity in $2^{O(t(n))}$.

Corollary

If $t(n) \geq n$, then $\mathrm{NTIME}(t(n)) \subseteq \mathrm{TIME}(2^{O(t(n))})$.

Proof

Recall the construction: BFS of the computation graph, keep a queue of configurations to process.

- At most d possible transitions for any $(q, X) \in (Q \setminus F) \times \Gamma$.
- So after k steps at most d^k configurations.
- Processing one configuration can be 'hidden' in the constant.
- Therefore the simulation is in time:

$$O(t(n)d^{t(n)}) = 2^{O(t(n))}$$

- We need to simulate multiple tapes, but:

$$(2^{O(t(n))})^2 = 2^{O(2t(n))} = 2^{O(t(n))}$$

□

P vs. NP

The class P

Definition

Let P (also PTIME) be the class of all languages decidable in polynomial time by a [single-tape, deterministic] Turing machine:

$$\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k)$$

- Path in a graph
- Primality of an integer (Agrawal, Kayal, Saxena 2002)
- Linear programming
- Horn-SAT

(The last two are P-complete under $\mathrm{LOGSPACE}$ reductions.)

Theorem (*CFL* $\subseteq$ *P*)

Every context free language belongs to P.

Proof: Take a ChNF grammar for L. Given input ω, run the CYK algorithm (polynomial, in $O(n^3)$). $\square$

The class NP: verifier-based definition

Definition

A verifier for a language L is an algorithm V such that:

$$L = \{w \mid \text{there exists a finite string } c \text{ such that } V \text{ accepts } \langle w, c\rangle\}$$

Such a c is called a certificate. It can be over any alphabet!

Complexity of verifiers is only considered wrt. the length of w: a polynomial verifier must halt in time $O(|w|^k)$ for some $k > 0$. Then we can assume the cerficate has polynomial length (otherwise the verifier cannot even read it).

Definition

NP is the class of all languages that have a polynomial verifier.

That is, there is an algorithm that works in time polynomial in $|w|$ and when given $w \in L$ and a cerficate c validates that c is a valid certificate for $w \in L$.

Hamiltonian path

A Hamiltonian path in a directed graph G is a directed path P that visits each vertex of G exactly once.

$$\mathrm{HAMPATH} = \{\langle G\rangle \mid G \text{ contains a Hamiltonian path}\}$$

- complexity for graphs can be measured just wrt. $|V|$ ($|E|$ is at most quadratic, thus polynomial)
- the certificate is the path (sequence of vertices)
- the algorithm verifies that the sequence is indeed a path containing each vertex exactly once; this can be easily done in polynomial time wrt. $|V|$
- for $\overline{\mathrm{HAMPATH}}$ we do not know whether a polynomial verifier exists (we only know the problem is in $\mathrm{EXPTIME}$)

The class *NP*: nondeterminism-based definition

Definition

NP is the class of all languages that have a polynomial verifier.

Theorem

$\mathrm{NP} = \bigcup_k \mathit{NTIME}(n^k)$.

Idea: convert a verifier to a nondeterministic TM, and vice versa

- $\Rightarrow$ the NTM guesses the certificate, then simulates the verifier
- $\Leftarrow$ the verifier takes as a certificate the accepting path of the NTM (more precisely, the sequence of nondeterministic choices that leads to acceptance), then simulates the NTM

Proof

$\mathrm{NP} \subseteq \bigcup_k NTIME(n^k)$: Let $L \in \mathrm{NP}$ and take a polynomial verifier V for L, say it works in time $C \cdot |\omega|^k$. Construct an NTM M:

Given input ω:

- nondeterministically guess a certificate c (of $|c| \leq C \cdot |\omega|^k$)
- simulate V on input $\langle \omega, c \rangle$
- if V accepted, M accepts

$\bigcup_k NTIME(n^k) \subseteq \mathrm{NP}$: Let $L \in NTIME(n^k)$, i.e., $L = L(M)$ for an NTM M working in time $O(n^k)$. Construct a polynomial verifier V:

Given input $\langle w, c \rangle$, interpret c as sequence of choices: $c_i = j$ means "at step i use jth possible transition" (order as in $\mathrm{code}(M)$)

- simulate M on input w
- at each step i choose the c_ith possible transition
- accept if this computation path leads to acceptance $\square$

Example: CLIQUE is in NP

$\mathrm{CLIQUE} = \{\langle G, k\rangle \mid G \text{ is a graph which contains } K_k \text{ as a subgraph}\}$

Polynomial verifier for CLIQUE: input $\langle\langle G, k\rangle, c\rangle$

- interpret the certificate c as a list of vertices
- check that c contains k vertices
- check that c induces a complete subgraph of G

Nondeterministic TM deciding CLIQUE: input $\langle G, k\rangle$

- nondeterministically choose a k-element subset $c \subseteq V$
- check that c induces a complete subgraph of G

Polynomial-time reductions and NP-completeness

Polynomial-time reducibility

Recall the notion of reduction between decision problems. Now we additionally require that the algorithm is polynomial-time:

A [total] function $f : \Sigma^* \to \Delta^*$ is polynomial-time computable, if there exists a [deterministic] Turing Machine M and $C, k > 0$ such that for each $\omega \in \Sigma^*$, M halts in at most $C \cdot |\omega|^k$ steps with $f(\omega) \in \Delta^*$ being the non-blank contents of its tape.

Definition

A language $A \subseteq \Sigma^*$ is polynomial-time reducible to a language $B \subseteq \Delta^*$, $A \leq_P B$, if there exists a polynomial-time computable function $f : \Sigma^* \to \Delta^*$ such that for all $\omega \in \Sigma^*$:

$$\omega \in A \;\Leftrightarrow\; f(\omega) \in B$$

Then we call f a polynomial-time reduction from A to B.

Example: Hamiltonian path from source to target

- $\mathrm{HAMPATH} = \{\langle G \rangle \mid G \text{ contains a Hamiltonian path}\}$
- $st\text{-}\mathrm{HAMPATH} = \{\langle G, s, t \rangle \mid G \text{ has a H. path from } s \text{ to } t\}$

Example

HAMPATH and st-HAMPATH are polynomial-time interreducible, i.e. each polynomial-time reduces to the other.

The reduction $\mathrm{HAMPATH} \leq_P st\text{-}\mathrm{HAMPATH}$:

Given G create G' by adding new vertices s, t and all edges from s to V_G and from V_G to t; define $f(\langle G \rangle) = \langle G', s, t \rangle$

$$\langle G \rangle \in \mathrm{HAMPATH} \;\Leftrightarrow\; \langle G', s, t \rangle \in st\text{-}\mathrm{HAMPATH}$$

The reduction $st\text{-}\mathrm{HAMPATH} \leq_P \mathrm{HAMPATH}$: construct G' by adding new vertices s', t', edges $s' \to s, t \to t'$; $f(\langle G, s, t \rangle) = \langle G' \rangle$

Example: 3SAT is polynomial-time reducible to CLIQUE

A propositional formula is in CNF if it is a conjunction of clauses, and 3-CNF if each clause contains exactly 3 literals.

- $\mathrm{SAT} = \{\langle \phi \rangle \mid \varphi$ is a satisfiable CNF formula$\}$
- $\mathrm{3SAT} = \{\langle \phi \rangle \mid \varphi$ is a satisfiable 3-CNF formula$\}$

Theorem

$\mathrm{3SAT}$ *is polynomial-time reducible to* CLIQUE.

Proof: Vertices are occurrences of literals (three vertices per clause). Include all edges except for:

- between vertices from the same clause
- between a variable and its negation (x and $\neg x$)

Set $k = \#$clauses. Note: Exactly one literal per clause selected. $\square$

Exercise: 3SAT is polynomial-time interreducible with SAT.