Review of Computability and Complexity

Lecture 2

Purandar Bhaduri
pbhaduri@iitg.ernet.in

Indian Institute of Technology Guwahati

January 1, 2018

Outline

GIAN
GLOBAL INITIATIVE OF ACADEMIC NETWORKS

Part I

Turing Machines and Computability

GLOBAL INITIATIVE OF ACADEMIC NETWORKS

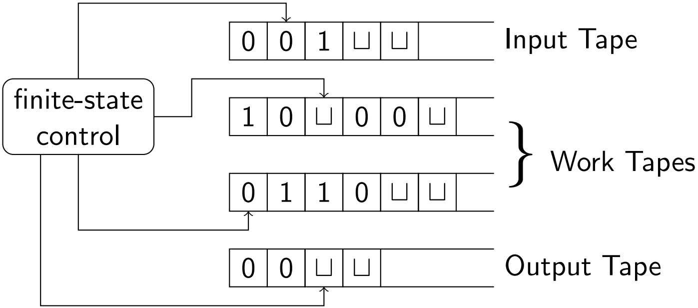
Decision Problems

In this part we will focus on decision problems — problems where we expect a yes/no answer on an input

- Examples: Is x composite? Is a vertex in T reachable from a vertex in S in the graph G?
- Non-examples: Find the factors of x. Find a path from a vertex in S to a vertex in T in graph G.

Will find it convenient to think of decision problems as a set/language, namely, the set of inputs on which the answer is supposed to be "yes".

Turing Machine



- Has a read-only input tape, write-only output tape, and k read/write work tapes.

Computation

- Initially: input stored on input tape, and all other tapes are blank, with the head pointing to the first cell of each tape and the finite control in an initial state.
- One Step: Based on current state of finite control, and symbols under the heads on input tape and work-tapes
  - Change state of finite control
  - Write new symbols on the cells scanned by heads on work tape, and output tape
  - Move tape heads on input tape and work-tapes left or right. If something was written on output tape, then move its head right.[1]

[1] Moving left of leftmost cell leaves the head position unchanged.

(Deterministic) Turing Machine

Formal Definition

A TM with k-work-tapes is $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}})$ where

- Q is a finite set of control states
- Σ is a finite set of input symbols
- $\Gamma \supseteq \Sigma$ is a finite set of tape symbols. Also, a blank symbol $\sqcup \in \Gamma \setminus \Sigma$
- $q_0 \in Q$ is the initial state
- $q_{\mathrm{acc}} \in Q$ is the accept state
- $q_{\mathrm{rej}} \in Q$ is the reject state, where $q_{\mathrm{rej}} \neq q_{\mathrm{acc}}$
- $\delta : (Q \setminus \{q_{\mathrm{acc}}, q_{\mathrm{rej}}\}) \times \Gamma^{k+1} \to Q \times \{\mathrm{L}, S, \mathrm{R}\} \times (\Gamma \times \{\mathrm{L}, S, \mathrm{R}\})^k \times (\Gamma \cup \{\epsilon\})$ is the transition function.

Configurations

Configurations or instantaneous discriptions are all the information needed to capture the "system state" during a computation. This is the control state, contents of all the work-tapes, and positions of all heads.

- Formally, on input of length n, a configuration $\mathrm{C} \in Q \times \{0, \ldots n-1\} \times (\Gamma^*\{*\}\Gamma\Gamma^*)^k$, where $*$ denotes the position of a work-tape head.

Computation

- $\mathrm{C}_1 \vdash \mathrm{C}_2$ denotes that TM moves from configuration C_1 to C_2 in one step.
    - For example, if $\delta(q, a, b) = (q', \mathrm{R}, (c, \mathrm{L}))$ and the input string is w, then

      $$(q, i, \alpha d * b\beta) \vdash (q', i+1, \alpha * dc\beta)$$

      provided $w_i = a$.
    - Formal definition of $\mathrm{C}_1 \vdash \mathrm{C}_2$ skipped.
- $\mathrm{C}_1 \vdash^* \mathrm{C}_2$ denotes that the TM moves from C_1 to C_2 in zero or more steps. i.e., $\mathrm{C}_1 = \mathrm{C}_2$ or there exist $\mathrm{C}'_1, \ldots, \mathrm{C}'_n$ such that $\mathrm{C}_1 = \mathrm{C}'_1$, $\mathrm{C}_2 = \mathrm{C}'_n$ and $\mathrm{C}'_i \vdash \mathrm{C}'_{i+1}$

GLOBAL INITIATIVE OF ACADEMIC NETWORKS

Acceptance

An input w is accepted by M (or M answers "yes" on w) if M reaches an accepting configuration (its control state is q_{acc}) from the initial configuration (with input w).
In other words, M rejects or does not accept w (i.e., answers "no") if either

- M reaches a halting configuration with state q_{rej} [explicit rejection]
- M never halts [implicit rejection]

Language Recognition and Function Computation

- The language accepted/recognized by M, denoted as $L(M)$, is the set of inputs w that M accepts.
- A Turing machine M is said to compute a (partial) function f if and only if whenever f is defined on input w, M halts with $f(w)$ written on its output tape.
- A function f is said to be computable if there is some Turing machine M such that M computes f.

Nondeterministic Turing Machine

Deterministic: Given current state, input symbol, and work-tape symbols, there is only one possibility for the next state, symbols to be written, and direction in which to move tape heads

- On any input string, the machine has only one "execution"

Non-deterministic: Current state, input symbol and work-tape symbols do not uniquely determine the next step

- Transition Function: $\delta : (Q \setminus \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}) \times \Gamma^{k+1} \to \mathcal{P}(Q \times \{\mathsf{L}, S, \mathsf{R}\} \times (\Gamma \times \{\mathsf{L}, S, \mathsf{R}\})^k \times (\Gamma \cup \{\epsilon\}))$.
- Input w is accepted if on some computation, M reaches an accepting configuration.

Expressive Power

- Deterministic and Non-deterministic Turing machines have the same expressive power: For every non-deterministic machine N, there is a deterministic TM $\mathrm{det}(N)$ such that $L(N) = L(\mathrm{det}(N))$.
- Number of work-tapes don't affect expressive power: For every TM M with k-work-tapes, there is a TM $\mathrm{single}(M)$ with 1 work-tape that accepts the same language.
- Changing tape alphabet does not affect computational power.

Church-Turing Thesis

Why Turing machines?

Alonzo Church

Alan Turing

Anything solvable using a mechanical procedure can be solved using a Turing machine

Recursively Enumerable and Recursive Languages

Definition (Recursively Enumerable)

L is recursively enumerable/semi-decidable if there is a TM M such that $L = L(M)$

- If $w \notin L$ then M may not halt on w

Definition (Recursive)

L is recursive/decidable if there is a TM M that halts on all inputs and $L = L(M)$

- L is undecidable if it is not decidable

Proposition

If L is recursive then L is recursively enumerable.

Examples of Recursive and Recursively Enumerable Languages

Observation

The following statements hold

- *There are languages L such that* $L \notin RE$
  - *Example,* $L_d = \{\langle M \rangle \mid \langle M \rangle \notin L(M)\}$
- *There is a language L such that* $L \in RE \setminus REC$
  - *Example,* $L_{\text{halt}} = \{\langle M \rangle \mid M \text{ halts on } \epsilon\}$

where REC denotes the class of recursive languages and RE denotes the class of recursively enumerable languages

Part II

Computational Complexity

Efficiency

- How do we measure the efficiency of a computational solution?
  - Many possible metrics could be used: running time, memory requirements, security, amount of communication, use of shared resource, quality of user interface, maintainability of code, etc.
  - We will focus on running time and memory requirements.
- Goal: To develop a framework to compare solutions, and quantify computational difficulty in a platform independent manner.

Measuring Computational Resources

- Computational resources needed to solve a problem depend on the size of the input
  - Computing "4+5" is easier than computing "1298959829494+4128393208157"
- Therefore, time and space used by an algorithm/TM are measured as function of input size
- Time/memory requirement of an algorithm could either measure resource needs on "worst" input or "average" input; we will focus on worst case analysis here.

Time and Space Bounded Computation

Definition

A Turing machine is said to run in time $t(n)$ if on any input u, all computations of M on u take at most $t(|u|)$ steps

Definition

A Turing machine is said to use at most space $s(n)$ if on any input u, all computations of M on u use at most $s(|u|)$ cells of the work-tapes

- Only work-tape cells written on at least once is counted
- Work-tape cells can be reused

Time and Space Complexity Classes

Definition

- $L \in \mathrm{DTIME}(t(n))$ iff there is a deterministic TM M that runs in time $t(n)$ and $L = L(M)$
- $L \in \mathrm{NTIME}(t(n))$ iff there is a non-deterministic TM M that runs in time $t(n)$ and $L = L(M)$
- $L \in \mathrm{DSPACE}(s(n))$ iff there is a deterministic TM M that uses space $s(n)$ and $L = L(M)$
- $L \in \mathrm{NSPACE}(s(n))$ iff there is a non-deterministic TM M that runs in time $s(n)$ and $L = L(M)$

Linear Speedup and Compression

Constants don't matter

Theorem

If $L \in DTIME(t(n))$ (or $L \in NTIME(t(n))$) and $c > 0$ is any constant, then $L \in DTIME(ct(n) + n)$ ($L \in NTIME(ct(n) + n)$, respectively).

Theorem

If $L \in DSPACE(s(n))$ (or $L \in NSPACE(s(n))$) and $c > 0$ is any constant, then $L \in DSPACE(cs(n))$ ($L \in NSPACE(cs(n))$, respectively).

The Big Oh!

Definition

- $f(n) = O(g(n))$ if there are constants c, n_0 such that for $n > n_0$, $f(n) \leq cg(n)$. We say $g(n)$ is an asymptotic upper bound.
- $f(n) = \Omega(g(n))$ if there are constants c, n_0 such that for $n > n_0$, $f(n) \geq cg(n)$. We say $g(n)$ is an asymptotic lower bound.

Robust Complexity Classes

- Complexity bounds for a problem should be platform independent
  - Small changes to model (like reducing tapes, or changing alphabet) should not affect the relevance of the results
  - Results should be valid even if one considers computational models other than Turing machines.
- Complexity classes should be closed under function composition
- Complexity classes should capture "interesting" real-world problems

Invariance Thesis

Any effective, mechanistic procedure can be simulated on a Turing machine using the same space (if space $\geq \log n$) and only a polynomial slowdown (time $\geq n$).

- Time (or memory) analysis of a pseudo-code/TM is a reliable indicator of the running time in any platform (upto a polynomial slowdown).

Alonzo Church

Common Complexity Classes

- L = DSPACE($\log n$)
  - Example problems: Multiplication of numbers, Boolean formula evaluation, reachability in undirected graphs
- NL = NSPACE($\log n$)
  - Example problems: Reachability in directed graphs
- P $= \cup_k$DTIME(n^k)
  - Example problems: Linear programming, convex programming, Boolean circuit evaluation
  - Cobham-Edmonds Thesis: P is the collection of all problems that have "efficient" computational solutions

Common Complexity Classes (continued)

- $\mathrm{NP} = \cup_k \mathrm{NTIME}(n^k)$
  - Example problems: Satisfiability, decision versions of many optimization problems
- $\mathrm{PSPACE} = \cup_k \mathrm{DSPACE}(n^k)$
  - Example problems: Solving games like Go
- $\mathrm{EXP} = \cup_k \mathrm{DTIME}(2^{n^k})$

Invariance Thesis: A Caveat

Invariance theses may not hold! In 1994, Peter Shor gave a polynomial time algorithm to factor two numbers on a "Quantum Computer". No efficient factoring algorithm is know for traditional computers! However, we haven't managed to build a "real" quantum computer yet.

Reachability

Problem

Given a directed graph $G = (V, E)$ and sets of vertices $S, T \subseteq V$, are there vertices $s \in S$ and $t \in T$ such that there is a path from s to t.

Solution

```
Set Marked := ∅
Queue Q := S
Marked := Marked ∪ S
while Q is not empty do
    t := Q.dequeue()
    if t ∈ T return ``yes''
    for each (t,u) ∈ E
        if u ∉ Marked
            Marked := Marked ∪ {u}
            Q := enqueue(Q,u)
return ``no''
```

Complexity Analysis of Reachability

Breadth first search solves the Reachability problem. Breadth first search runs in linear time, and uses linear memory. Hence

- Reachability $\in$ P
- Reachability $\in$ PSPACE

Reachability $\in$ NL

Suppose input to the Reachability problem is $G = (V, E)$, S, and T and suppose $\mid V \mid = m$.

$\texttt{current} := \textbf{choose from } \texttt{S}$
$\texttt{pathlength} := 0$
$\textbf{while } \texttt{current} \notin \texttt{T} \textbf{ and } \texttt{pathlength} < m \textbf{ do}$
$\quad \texttt{next} := \textbf{choose from } \{v \in V \mid (\texttt{current}, v) \in E\}$
$\quad \texttt{current} := \texttt{next}$
$\quad \texttt{pathlength} := \texttt{pathlength + 1}$
$\textbf{if } \texttt{current} \in \texttt{T} \textbf{ return } \texttt{``yes''}$

Time to Space

Proposition

$DTIME(t(n)) \subseteq DSPACE(t(n))$ *and*
$NTIME(t(n)) \subseteq NSPACE(t(n))$

Proof.

You can write in at most $t(n)$ cells in $t(n)$ steps. $\square$

Space to Time

Proposition

$DSPACE(s(n)) \subseteq DTIME(n \cdot 2^{O(s(n))})$ *and*
$NSPACE(s(n)) \subseteq NTIME(n \cdot 2^{O(s(n))})$ [a]

[a] Provided $s(n)$ is a proper complexity function.

Proof.

- If M uses at most $s(n)$ space, then it has $n \cdot 2^{O(s(n))}$ configurations.
  - A configuration has state + input head position + work-tape contents
- If M accepts w of length n, it does so within $n \cdot 2^{O(s(n))}$ steps
  - Any computation longer than $n \cdot 2^{O(s(n))}$ is cycling. □

Consequences

$$\mathrm{L} \subseteq \mathrm{P} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}$$
$$\mathrm{NL} \subseteq \mathrm{NP} \subseteq \mathrm{NPSPACE} \subseteq \mathrm{NEXP}$$

Deterministic Classes to Nondeterministic Classes

Proposition

$DTIME(t(n)) \subseteq NTIME(t(n))$ *and*
$DSPACE(s(n)) \subseteq NSPACE(s(n))$.

Proof.

Because deterministic Turing machines are special nondeterministic Turing machines, namely, those that have exactly one transition from every configuration. $\square$

Nondeterministic Classes to Deterministic Classes

Proposition

$NTIME(t(n)) \subseteq DTIME(2^{t(n)})$

Theorem (Savitch)

$NSPACE(s(n)) \subseteq DSPACE((s(n))^2)$, *provided* $s(n) \geq \log n$.

Combining everything . . .

$$\mathsf{L} \rightarrow \mathsf{NL} \rightarrow \mathsf{P} \rightarrow \mathsf{NP} \longrightarrow \begin{array}{c}\mathsf{PSPACE}\\ =\\ \mathsf{NPSPACE}\end{array} \longrightarrow \mathsf{EXP}$$

Relationship between Complexity Classes. $\rightarrow$ indicates containment, though whether it is strict is unknown.

In addition . . .

L $\neq$ PSPACE and NL $\neq$ PSPACE due to space hierarchy theorem, and P $\neq$ EXP due to time hierarchy theorem.

Comparing Computational Problems

First Ideas

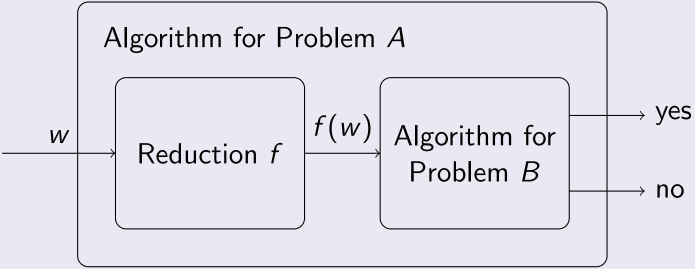
Problem A is at most as hard as problem B, if an algorithmic solution for B (with some resource bounds) can be used to obtain an algorithmic solution to solve A (with similar resource bounds)

- Checking an invariant property for a program (with Boolean variables) is no harder than solving the reachability problem on directed graphs.

Many-one Reductions: Informal View

Capturing the Relative Difficulty of Problems

A reduction from A to B is a function f such that for any input x, solving A on x is the same as solving B on $f(x)$.



Thus, A is no harder than B.

Logspace Reductions

Definition

A logspace reduction from A to B is a logspace computable function $f : \Sigma^* \to \Sigma^*$ such that

$$u \in A \text{ iff } f(u) \in B$$

A is said to be logspace reducible to B and is denoted by $A \leq_{\mathsf{L}} B$.

Properties of Reductions

Proposition

If $A \leq_L B$ *then* $\overline{A} \leq_L \overline{B}$.

Proposition

If $A \leq_L B$ *and* $B \leq_L C$ *then* $A \leq_L C$.

Proposition

If $A \leq_L B$ *and* $B \in \mathcal{C}$ *then* $A \in \mathcal{C}$, *where* $\mathcal{C} \in \{L, NL, P, NP, PSPACE, EXP\}$

Hardness and Completeness

Hardest Problems in a Class

Intuition

The most difficult problem in a collection of problems $\mathcal{C}$ is a problem that is at least as difficult as every other problem in $\mathcal{C}$.

Definition

Let $\mathcal{C} \in \{\mathsf{L}, \mathsf{NL}, \mathsf{P}, \mathsf{NP}, \mathsf{PSPACE}, \mathsf{EXP}\}$

- L is said to be $\mathcal{C}$-hard iff for every $L' \in \mathcal{C}$, $L' \leq_{\mathsf{L}} L$
- L is said to be $\mathcal{C}$-complete iff $L \in \mathcal{C}$ and L is $\mathcal{C}$-hard

Hardness as a way to argue optimality

- Suppose L is $\mathcal{C}_1$-hard, and suppose we believe $\mathcal{C}_1$ to be not contained in $\mathcal{C}_2$. Then it is unlikely that L belongs to $\mathcal{C}_2$.
- Justification: Suppose (for contradiction) $L \in \mathcal{C}_2$. Consider an arbitrary problem $A \in \mathcal{C}_1$. Since L is $\mathcal{C}_1$-hard, $A \leq_{\mathsf{L}} L$. Further, since $L \in \mathcal{C}_2$, $A \in \mathcal{C}_2$. Therefore, $\mathcal{C}_1 \subseteq \mathcal{C}_2$, which we believe to be unlikely.

Reachability is NL-hard

Let L be some problem in NL and M be a NL algorithm solving L. Need to show that $L \leq_{\mathrm{L}}$ Reachability.

- Goal: Construct a function f computable in L such that for any input w, $f(w) = (G, S, T)$ with the property that $w \in L$ iff some vertex in T is reachable from a vertex in S in graph G.
- Reduction: G is the "configuration graph" of M on input w, i.e., vertices are configurations of M on input of length $|w|$, and there is an edge from C_1 to C_2 iff $\mathrm{C}_1 \vdash \mathrm{C}_2$. $S = \{\mathrm{C}_0\}$, where C_0 is the initial configuration with input w, and T is the set of accepting configurations.

Reachability is NL-hard

Correctness of reduction

- If $w \in L$ then w is accepted by M. There is a computation $\mathrm{C}_0 \vdash^* \mathrm{C}_1$, where C_1 is accepting. Thus, there is a path from C_0 ($\in S$) to C_1 ($\in T$) in the configuration graph.
- Suppose there is a path from C_0 to some $\mathrm{C}_1 \in T$ in G then by definition of G $\mathrm{C}_0 \vdash^* \mathrm{C}_1$. Thus, M accepts w, and so $w \in L$.

Reachability is NL-hard

Reduction computable in L

- Each configuration of M is a string of length $O(\log n)$ ($|w| = n$). Thus, the set of vertices of G can be output using log space by enumerating all such strings.
- Accepting configurations ($= T$) are all configurations with state q_{acc}. Thus, they can be enumerated using log space. The initial configuration ($= S$) is fixed string that can be written out without using any memory.
- Finally, the set of edges can be output as follows. Each pair of configurations is listed; such pairs are of length $O(\log n)$ and so can be stored on the work tape. For each pair, the machine will read the string encoding this pair, and check if they correspond to one step of M. If so the pair is output.

References

- Michael Sipser, *Introduction to the Theory of Computation*. Third Edition, Cengage Learning, 2012.
- Mahesh Viswanathan and Sayan Mitra, Lecture Slides on *"Turing Machines, Languages and Decidability"* and *"Complexity Classes"* from the course *ECE/CS 584: Embedded and Cyber-Physical Systems Verification* at UIUC, Fall 2004.