Elementary Complexity Theory

Bow-Yaw Wang

Institute of Information Science
Academia Sinica, Taiwan

July 2, 2009

Outline

Turing Machine

- Turing machines are one of the most popular models of computation.
- They are proposed by Alan Turing (a British mathematician).
  - The renowned ACM Turing Award is named after him.
- A Turing machine is a quadruple $M = (K, \Sigma, \delta, s)$ where
  - K is a finite set of states;
  - Σ is a finite set of symbols (also called an alphabet);
    - ★ $\sqcup \in \Sigma$: the blank symbol
    - ★ $\triangleright \in \Sigma$: the first symbol
  - δ is a transition function
    - ★ $\delta : K \times \Sigma \to (K \cup \{halt, yes, no\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}$.
    - ★ Since δ is a function, M is deterministic.

Computation in Turing Machines

- A Turing machine has a tape.
  - Initially, a finite input $x = a_1 a_2 \cdots a_n \in (\Sigma - \{\sqcup\})^*$ following the symbol $\triangleright$ is on the tape.
  - $\triangleright\, a_1 a_2 \cdots a_n \sqcup \sqcup \cdots$
- There is a cursor pointing to a current symbol on the tape
  - Initially, the cursor points to $\triangleright$.
  - $\underline{\triangleright} a_1 a_2 \cdots a_n \sqcup \sqcup \cdots$
- δ is the "program" of the machine.
  - Assume the current state is $q \in K$, the current symbol is $\sigma \in \Sigma$.
  - $\delta(q, \sigma) = (p, \rho, D)$ represents that p is the next state, ρ is the symbol replacing σ, and $D \in \{\leftarrow, \rightarrow, -\}$ is the cursor direction.
  - We assume the $\triangleright$ is never overwritten.
    - ★ That is, for all q and p, $\delta(q, \triangleright) = (p, \rho, \Delta)$ implies $\rho = \triangleright$ and $D = \rightarrow$.

Configurations

- A configuration characterizes the complete description of the current computation.
- A configuration (q, w, u) of a Turing machine consists of a state q, and two strings w and u.
  - q is the current state of the Turing machine.
  - w is the string to the left of the cursor and the current symbol.
  - u is the string to the right of the cursor (possibly empty).
- The initial configuration on input x is therefore $(s, \triangleright, x)$.
- Moreover, we write $(q, w, u) \xrightarrow{M} (q', w', u')$ if (q, w, u) changes to (q', w', u') by one step in M. There are three cases:
  - $\delta(q, \sigma) = (p, \rho, \leftarrow)$, then $(q, x\sigma, y) \xrightarrow{M} (p, x, \rho y)$;
  - $\delta(q, \sigma) = (p, \rho, \rightarrow)$, then $(q, x\sigma, \tau y) \xrightarrow{M} (p, x\rho\tau, y)$;
  - $\delta(q, \sigma) = (p, \rho, -)$, then $(q, x\sigma, y) \xrightarrow{M} (p, x\rho, y)$.

Halting and Acceptance

- The computation in a Turing machine cannot continue only when it reaches the three states: *halt*, *yes*, and *no*.
  - If this happens, we say the Turing machine halts.
  - Of course, a Turing machine may not halt.
- If the state *yes* is reached, we say the machine accepts the input (write $M(x) = yes$).
- If the state *no* is reached, we say the machine rejects the input (write $M(x) = no$).
- If the state *halt* is reached, we define the output of the Turing machine to be the content y of the tape when it halts (write $M(x) = y$).
- If the Turing machine does not halt, we write $M(x) = \nearrow$.

Recursive Languages

- Let $L \subseteq (\Sigma \setminus \{\sqcup\})^*$ be a language.
- Let M be a Turing machine such that for any $x \in (\Sigma \setminus \{\sqcup\})^*$,
  - $x \in L$, then $M(x) = yes$;
  - $x \notin L$, then $M(x) = no$.
- Then we say M decides L.
- If L is decided by some Turing machine, we say L is recursive.
- In other words,
  - M always halts on any input; and
  - M decides whether the input is in the language or not.

Recursively Enumerable Languages

- Let $L \subseteq (\Sigma \setminus \{\sqcup\})^*$ be a language.
- Let M be a Turing machine such that for any $x \in (\Sigma \setminus \{\sqcup\})^*$,
  - $x \in L$, then $M(x) = yes$;
  - $x \notin L$, then $M(x) = \nearrow$.
- Then we say M accepts L.
- If L is accepted by some Turing machine, we say L is recursively enumerable.
- Note that,
  - M may not halt.
  - The input is in the language when when it halts.
- Practically, this is not very useful.
  - We do not know how long we need to wait.

Nondeterministic Turing Machines

- Similar to finite automata, we can consider nondeterministic Turing machines.
- A nondeterministic Turing machine is a quadruple $N = (K, \Sigma, \Delta, s)$ where K is a finite set of states, Σ is a finite set of symbols, and $s \in K$ is its initial state. Moreover,
  - $\Delta \subseteq (K \times \Sigma) \times [(K \cup \{halt, yes, no\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}]$ is its transition relation.
- Similarly, we can define $(q, w, u) \xrightarrow{N} (q', w', u')$.

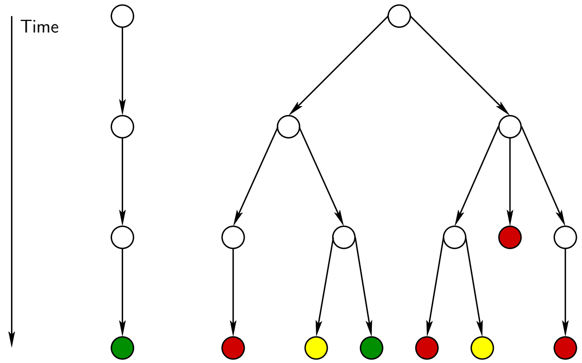
Acceptance

- Let N be a nondeterministic Turing machine.
- Let $L \subseteq (\Sigma \setminus \{\sqcup\})^*$ be a language.
- We say N decides L if for any $x \in \Sigma^*$

$$x \in L \text{ if and only if } (s, \triangleright, x) \xrightarrow{N}{}^{*} (yes, w, u) \text{ for some } w, u.$$

- Since N is nondeterministic, there may be several halting configurations.
  - $(s, \triangleright, x) \xrightarrow{N}{}^{*} (halt, w_0, u_0)$, $(s, \triangleright, x) \xrightarrow{N}{}^{*} (halt, w_1, u_1)$, $(s, \triangleright, x) \xrightarrow{N}{}^{*} (no, w_2, u_2)$, etc.
- However, we need only one halting configuration of the form (yes, w, u) for $x \in L$.
  - Conversely, *all* halting configurations are not of this form if $x \notin L$.

Deterministic and Nondeterministic Computation



- A nondeterministic Turing machine decides language L in time $f(n)$ if it decodes L and for any $x \in \Sigma^*$, $(s, \triangleright, x) \longrightarrow^k (q, u, w)$, then $k \leq f(|x|)$.

P and NP

- Define

$$\textbf{TIME}(f(n)) = \{L : L \text{ can be decided by a TM in time } f(n)\}$$

$$\textbf{NTIME}(f(n)) = \{L : L \text{ can be decided by an NTM in time } f(n)\}$$

- Let

$$\mathbf{P} = \bigcup_{k \in \mathbb{N}} \textbf{TIME}(n^k) \text{ and } \mathbf{NP} = \bigcup_{k \in \mathbb{N}} \textbf{NTIME}(n^k)$$

- We have $\mathbf{P} \subseteq \mathbf{NP}$.
  - However, whether the inclusion is proper is still open.
- In this lecture, we will consider several problems related to logic and discuss their complexity.

Boolean Expressions

- Fix a countably infinite set of Boolean variables $X = \{x_0, x_1, \ldots, x_i, \ldots\}$.
- A Boolean expression is an expression built from Boolean variables with connectives $\neg$, $\vee$, and $\wedge$.
- A truth assignment T is a mapping from Boolean variables to truth values **false** and **true**.
- We say a truth assignment T satisfies a Boolean expression ϕ (write $T \models \phi$) if $\phi[x_0, x_1, \cdots, x_i, \cdots \mapsto T(x_0), T(x_1), \cdots, T(x_i), \cdots]$ evaluates to **true**.

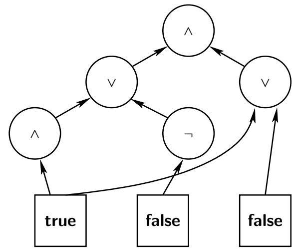
SATISFIABILITY (SAT)

- A Boolean expression ϕ is satisfiable if there is a truth assignment T such that $T \models \phi$.
- SATISFIABILITY (SAT) is the following problem:
  Given a Boolean expression ϕ in conjunctive normal form, is it satisfiable?
- SAT can be decided in $\mathbf{TIME}(n^2 2^n)$ by exhaustive search.
- SAT can be decided in **NP**:
  - Guess a truth assignment nondeterministically;
  - Check whether the truth assignment satisfies all clauses.

Boolean Circuits

- A Boolean circuit is a graph $C = (V, E)$ where $V = \{1, \ldots, n\}$ are the gates of C. Moreover
  - C has no cycles. All edges are of the form (i, j) with $i < j$.
  - All nodes have indegree ≤ 2.
  - Each $i \in V$ has a sort $s(i)$ where $s(i) \in \{\mathbf{false}, \mathbf{true}, \vee, \wedge, \neg, x_0, x_1, \ldots, \}$.
    - ★ If $s(i) \in \{\mathbf{false}, \mathbf{true}, x_0, x_1, \ldots\}$, i has indegree 0 and is an input gate;
    - ★ If $s(i) = \neg$, i has indegree one;
    - ★ If $s(i) \in \{\vee, \wedge\}$, i has indegree two.
  - The gate n has outdegree zero and is called the output gate.
- The semantics of a Boolean circuit is defined as in propositional logic.

CIRCUIT VALUE



- CIRCUIT VALUE is the following problem:
  Given a Boolean circuit C without variable gates, does C evaluate to **true**?
- CIRCUIT VALUE is in **P**.
  - Simply evaluate the gate values in numerical order.

Space Complexity

- A k-tape Turing machine with input and output is a Turing machine M with k tapes. Moreover,
  - M never writes on tape 1 (its read-only input);
  - M never reads on tape k (its write-only output);
  - The other $k-2$ tapes are working tapes.
- A configuration of k-tape Turing machine with input and output is a $2k+1$-tuple $(q, w_1, u_1, \ldots, w_k, u_k)$.
  - The initial configuration on input x is $(s, \triangleright, x, \triangleright, \epsilon, \ldots, \triangleright, \epsilon)$.
- On input x, if $(s, \triangleright, x, \triangleright, \epsilon, \ldots, \triangleright, \epsilon) \xrightarrow{M}{}^{*} (H, w_1, u_1, \ldots, w_k, u_k)$ where $H \in \{halt, yes, no\}$, we say the space required by M on input x is $\sum_{i=2}^{k-1} |w_i u_i|$.
  - Note that the space on input and output tapes does not count.

SPACE$(f(n))$ and **NSPACE**$(f(n))$

- Define

$$\textbf{SPACE}(f(n)) = \left\{ L : \begin{array}{c} L \text{ can be decided by a TM with input} \\ \text{and output within space bound } f(n) \end{array} \right\}.$$

- **NSPACE**(f(n)) is defined similarly.
- Define

$$\textbf{L} = \textbf{SPACE}(\log n).$$

$$\textbf{NL} = \textbf{NSPACE}(\log n).$$

$$\textbf{PSPACE} = \bigcup_{k \in \mathbb{N}} \textbf{SPACE}(n^k)$$

$$\textbf{NPSPACE} = \bigcup_{k \in \mathbb{N}} \textbf{NSPACE}(n^k)$$

"Complements" of Complexity Classes

- Let $L \subseteq \Sigma^*$ be a language.
- The complement of L, write $\overline{L}$, is as follows.

$$x \in \overline{L} \text{ iff } x \notin L.$$

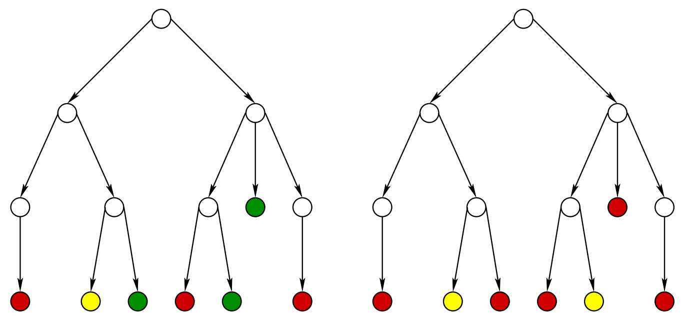
- For any complexity class $\mathcal{C}$, define

$$\textbf{co}\mathcal{C} = \{\overline{L} : L \in \mathcal{C}\}.$$

Complements of Complexity Classes

- For any deterministic complexity class $\mathcal{C}$, we have $\textbf{co}\mathcal{C} = \mathcal{C}$.
  - Let $L \in \mathcal{C}$. There is a TM M deciding L within the resource bound of $\mathcal{C}$. Construct a TM M' by switch the *yes* and *no* states of M. We have $x \in L$ iff $M(x) = yes$ iff $M'(x) = no$. Thus M' decides $\overline{L}$ within the resource bound of $\mathcal{C}$.
- Consider $L = \{\phi : \phi$ is an unsatisfiable Boolean expression $\}$.
- Thus $\overline{L} = \{\phi : \phi$ is a satisfiable Boolean expression $\}$.
  - Strictly speaking, $\overline{L} = \{\phi : \phi$ is not a Boolean expression or ϕ is satisfiable $\}$. But this is a convenient convention.
- Since $\overline{L} \in \textbf{NP}$, we have $L \in \textbf{coNP}$.

NP and **coNP**



- $SAT = \{\phi : \phi$ is a satisfiable Boolean expression $\}$.
  - $\phi \in SAT$ if there is a truth assignment that satisfies ϕ.
- $UNSAT = \{\phi : \phi$ is an unsatisfiable Boolean expression $\}$.
  - $\phi \in UNSAT$ if there is no truth assignment that satisfies ϕ.
  - $\phi \in UNSAT$ if all truth assignments do not satisfy ϕ.

Fallacies Q & A

- Q: Is $\mathbf{P} \subseteq \mathbf{coNP}$?
- A: Yes.
  - Let $L \in \mathbf{P}$. Clearly, $\overline{L} \in \mathbf{P} \subseteq \mathbf{NP}$. Thus $L \in \mathbf{coNP}$.
- Q: Is $\Sigma^* \setminus \mathbf{NP} = \mathbf{coNP}$?
- A: No.
  - Both **NP** and **coNP** are classes of languages (that is, each one is a set of sets of strings). It does not make sense to consider $\Sigma^* \setminus \mathbf{NP}$ or $\Sigma^* \setminus \mathbf{coNP}$.
- Q: Is $2^{\Sigma^*} \setminus \mathbf{NP} = \mathbf{coNP}$?
- A: No.
  - $\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}$.

Relation between Complexity Classes

- Since any Turing machine with input and output is a nondeterministic Turing machine with input and output, it is easy to see the following statements:
  - $\mathbf{TIME}(f(n)) \subseteq \mathbf{NTIME}(f(n))$;
  - $\mathbf{SPACE}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$.
- Moreover, a Turing machine can use at most $f(n)$ space in time $f(n)$. Therefore,
  - $\mathbf{TIME}(f(n)) \subseteq \mathbf{SPACE}(f(n))$;
  - $\mathbf{NTIME}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$.
- Can we establish more relation between these classes?

Nondeterministic Time and Deterministic Space

Theorem

For any "reasonable" non-decreasing function $f(n)$, we have $\mathbf{NTIME}(f(n)) \subseteq \mathbf{SPACE}(f(n))$.

Proof.

Let $L \in \mathbf{NTIME}(f(n))$ and M a NTM decide L in time $f(n)$. On input of size n, a TM M' works as follows:

1. for each sequence of nondeterministic choices of M
2. M' simulates M with time $f(n)$
3. if M accepts, M' accepts
4. if M does not accept, M' erases working tapes

Each sequence of nondeterministic choices of M has length $f(n)$. Moreover, the simulation of M uses at most $f(n)$ space. Hence M is a TM deciding L in space $f(n)$. $\square$

Reachability Method

Theorem

For any "reasonable" non-decreasing function $f(n)$, we have $\textbf{NSPACE}(f(n)) \subseteq \textbf{TIME}(c^{\log n + f(n)})$.

Proof.

Let $L \in \textbf{NSPACE}(f(n))$ and M a k-tape NTM with input and output decide L in space $f(n)$. A configuration of M is of the form $(q, w_1, u_1, \ldots, w_k, u_k)$. Moreover, M does not overwrite the input. A configuration can be represented by $(q, i, w_2, u_2, \ldots, w_k, u_k)$ where i is the index of the cursor on input. Thus there are at most $|K| \times n \times |\Sigma|^{(2k-2)f(n)}$ configurations.

Define the configuration graph of M on input x $G(M, x)$ to be the graph with configurations of M as its nodes. (C_0, C_1) is an edge in $G(M, x)$ if $C_0 \xrightarrow{M} C_1$. Thus $x \in L$ iff there is a path from $(s, \triangleright, x, \triangleright, \epsilon \cdots, \triangleright, \epsilon)$ to some $(yes, w_1, u_1, \ldots, w_k, u_k)$.

Reachability Method

Proof.

Since there is a polynomial-time deterministic algorithm for graph reachability, we can decide if $x \in L$ in time polynomial in the size of the configuration graph. Thus $L \in \textbf{TIME}(c^{\log n + f(n)})$. □

- To be precise, let us describe how the reachability algorithm is used.
- We do not need the adjacency matrix of the configuration graph.
  - It uses too much space unnecessarily.
- Instead, we check whether there is an edge from C_0 to C_1 by simulating M.
- In other words, entries in the adjacency matrix are computed when needed.
  - This is called an on-the-fly algorithm.

Comparing Complexity Classes

Theorem

$\textbf{L} \subseteq \textbf{NL} \subseteq \textbf{P} \subseteq \textbf{NP} \subseteq \textbf{PSPACE} \subseteq \textbf{NSPACE}$.

- We know in fact that $\textbf{L} \subsetneq \textbf{PSPACE}$.
- However, we do not know which of the inclusion is proper.

Nondeterminism in Space Complexity

- For time complexity, we do not know if nondeterminism does increase the expressive power of Turing machines.
  - Otherwise, we would have known $\textbf{P} \subsetneq \textbf{NP}$ or not.
- For space complexity, we know a little bit more.
  - Intuitively, nondeterministic computation does not need more space because space can be reused.

Savitch's Theorem

Theorem

REACHABILITY $\in$ **SPACE**$(\log^2 n)$.

Proof.

Let $G = (V, E)$ with $|V| = n$. For $x, y \in V$ and $i \in \mathbb{N}$, define that $PATH(x, y, i)$ holds if there is a path of length $\leq 2^i$ from x to y. Clearly, x reaches y in G if $PATH(x, y, \lceil \log n \rceil)$ holds. We will construct a TM M that decides $PATH(x, y, i)$.
M decides $PATH(x, y, 0)$ by looking up the adjacency matrix of G.
For $i \geq 1$, M does the following recursively:

1. for all nodes z
2. if $PATH(x, z, i-1)$ holds then
3. if $PATH(z, y, i-1)$ holds then go to *yes*

Savitch's Theorem

Proof.

More precisely, when M is checking z. It puts the tuple $(x, z, i-1)$ on its working tape (line 2). If $PATH(x, z, i-1)$ does not hold, M erases the tuple $(x, z, i-1)$ and tries the next node. If $PATH(x, z, i-1)$ holds, M erases the tuple $(x, z, i-1)$, puts the new tuple $(z, y, i-1)$ on its working tape. If $PATH(z, y, i-1)$ does not hold, M erases the tuple $(z, y, i-1)$ and tries the next node. Otherwise, M goes to the *yes* state.
Observe that at most $\lceil \log n \rceil$ tuples on the working tape. Each tuple uses $3\lceil \log n \rceil$ cells. Hence M uses at most $O(\log^2 n)$ space. □

- Of course, the algorithm is highly inefficient in terms of time.
  - Each recursive call will try all nodes regardlessly.
- On the other hand, it is very efficient in terms of space
  - DFS, for instance, may use $O(n)$ space.

NSPACE = SPACE

Theorem

For any "reasonable" nondecreasing $f(n) \geq \log n$*,*
NSPACE$(f(n)) \subseteq$ **SPACE**$(f^2(n))$.

Proof.

Let L be a language and M an NTM decide L in space $f(n)$. Moreover, $x \in L$ if the initial configuration of M can reach an accepting configuration of M in its configuration graph. Recall that the configuration graph of M has $O(c^{\log n + f(n)}) = O(c^{f(n)})$ nodes (since $f(n) \geq \log n$). Thus there is a TM M' deciding the reachability problem within space $O(\log^2(c^{f(n)})) = O(f^2(n))$. □

- In other words, nondeterminism does not increase the power of TM in terms of space complexity.

CoNSPACE

- For any deterministic complexity class $\mathcal{C}$, we have shown **co**$\mathcal{C} = \mathcal{C}$.
- For nondeterministic complexity classes, it is not clear at all.
  - Recall **NP** and **coNP**.
- However, we will show that **NSPACE** = **coNSPACE**.

Immerman-Szelepscényi Theorem I

Theorem

Given a graph G and a node x, the number of nodes reachable from x in G can be computed by an NTM within space $\log n$.

Proof.

Let $S(k) = \{y : x \longrightarrow^{\leq k} y\}$. We compute $|S(1)|, |S(2)|, \ldots, |S(n-1)|$ iteratively. Clearly $|S(n-1)|$ is what we want. We design an nondeterministic algorithm using four functions. The Main function is:

1. $|S(0)| := 1$
2. for $k = 1, 2, \ldots, n-1$ do $|S(k)| :=$ Count $(|S(k-1)|)$

Observe that only $|S(k-1)|$ is needed to compute $|S(k)|$.

Immerman-Szelepscényi Theorem II

Proof.

To compute $|S(k)|$ from C, we check how many nodes u are in $S(k)$ by invoking InS (k, u, C). The Count (C) function is:

1. $\ell := 0$
2. for $u \in V$ do if InS (k, u, C) then $\ell := \ell + 1$

The InS (k, u, C) function is: (cf the next slide)

1. $m := 0$; reply := false
2. for $v \in V$ do
3.   if GuessInS $(k-1, v)$ then
4. $m := m + 1$
5.   if $(v, u) \in E$ then reply := true
6. if $m < C$ then "give up" else return reply

Immerman-Szelepscényi Theorem III

Proof.

For each node v, we nondeterministically check if $v \in S(k-1)$ (GuessInS $(k-1, v)$). If so, the counter m is incremented by 1. Futhermore, if v can reach u in one step, set reply to true.
After checking all nodes nondeterministically, we will check if we have correctly collect all nodes in $S(k-1)$ by comparing the counter m with C. If so, return the variable reply.

Immerman-Szelepscényi Theorem IV

Proof.

To verify $v \in S(j)$ nondeterministically, it suffices to guess a path of length j. The function GuessInS (j, v) is:

1. $w_0 := x$
2. for $p = 1, \ldots, j$ do
3.   guess $w_p \in V$ and check $(w_{p-1}, w_p) \in E$ (if not, "give up")
4. if $w_j = v$ then return true else "give up."

Observe that only the variables $k, C, \ell, u, m, v, p, w_p, w_{p-1}$ need be recorded. Since the number of nodes is n, $\log n$ space is needed.

□

NSPACE = coNSPACE

Theorem

For any "reasonable" nondecreasing function $f(n) \geq \log n$, $\mathbf{NSPACE}(f(n)) = \mathbf{coNSPACE}(f(n))$.

Proof.

Suppose $L \in \mathbf{NSPACE}(f(n))$ and an NTM M decide L in space $f(n)$. We construct an NTM $\overline{M}$ that decides $\overline{L}$ in space $f(n)$. On input x, $\overline{M}$ runs the nondeterministic algorithm in the previous theorem on the configuration graph of M. If at any time, $\overline{M}$ discovers that M reaches an accepting configuration, $\overline{M}$ halts and rejects x. If $|S(n-1)|$ is computed and no accepting configuration is found, $\overline{M}$ accepts x.
Since the configuration graph of M has $c^{\log|x|+f(|x|)}$ nodes, $\overline{M}$ uses at most $O(f(n))$ space if $f(n) \geq \log n$. □

Reduction

- A language L_0 is reducible to L_1 if there is a function $R : \Sigma^* \to \Sigma^*$ computable by a Turing machine in space $O(\log n)$ such that for all input x,
$$x \in L_0 \text{ if and only if } R(x) \in L_1.$$
- R is called a reduction from L_0 to L_1.
- If R is a reduction computed by a Turing machine M, then for all input x, M halts after a polynomial number of steps.
  - Since M is deterministic, its configurations cannot repeat.
    - ★ Otherwise, M will not halt.
  - There are at most $O(nc^{\log n})$ configurations.

Solving Problems by Reductions

- Assume there is a Turing machine M_1 to decide L_1.
  - That is, on input x
    - ★ M_1 goes to *yes* if $x \in L_1$;
    - ★ M_1 goes to *no* if $x \notin L_1$.
- Further, assume L_0 is reducible to L_1 by R.
- There is a Turing machine M_0 that decides L_0.
  1. On input x, M_0 first computes $R(x)$;
  2. M_0 invokes M_1 on input $R(x)$. There are two cases:
     - ★ If M_1 goes to *yes*, M_0 goes to *yes*;
     - ★ If M_1 goes to *no*, M_0 goes to *no*.
- If there is a reduction from L_0 to L_1 and L_1 is solved, then we can solve L_0 as well.
  - Informally, L_1 is *harder* than L_0.

Completeness

- Let $\mathcal{C}$ be a complexity class (such as $\mathbf{P}, \mathbf{NP}, \mathbf{L}$, etc).
- A language L in $\mathcal{C}$ is called $\mathcal{C}$-complete if any language $L' \in \mathcal{C}$ can be reduced to L.
- Informally, L is $\mathcal{C}$ – *complete* means that it is *hardest* to solve in $\mathcal{C}$.
  - Since any language in $\mathcal{C}$ is reducible to L, solving L means solving any language in $\mathcal{C}$.
- But how can we prove a langauge is $\mathcal{C}$-complete?
  - There are infinitely many languages in $\mathcal{C}$. It is impossible to write down a reduction for each of them.

Table Method

$\triangleright$	0_s	1	1	0	$\sqcup$
$\triangleright$	0	1_{q_0}	1	0	$\sqcup$
$\triangleright$	0	1	1_{q_0}	0	$\sqcup$
$\triangleright$	0	1	1	0_{q_0}	$\sqcup$
$\triangleright$	0	1	1_{q_1}	$\sqcup$	$\sqcup$
$\triangleright$	0	1_{q_1}	1	$\sqcup$	$\sqcup$
$\triangleright$	*no*	1	1	$\sqcup$	$\sqcup$

- Consider a TM $M = (K, \Sigma, \delta, s)$ deciding language L within time n^k.
- Its computation on input x can be seen as a $|x|^k \times |x|^k$ computation table.
  - Its rows are time steps 0 to $|x|^k - 1$.
  - Its columns are contents of the tape.
- Moreover, let us write σ_q to represent that the cursor is pointing at a symbol σ with state q.

Convention in Table Method

- To simplify our presentation, we adopt the following conventions.
  - M has only one tape;
  - M halts on any input x in $|x|^k - 2$ steps;
  - The computation table has enough $\sqcup$'s to its right;
  - M starts with cursor at the first symbol of x;
  - M never visits the leftmost $\triangleright$;
  - M halts with its cursor at the second position and exactly at step $|x|^k$.
    - ★ You should check that these conventions are not at all restrictive.
- Let's use $T(x)$ to represent the computation table on input x.
  - $T_{ij}(x)$ represent the (i, j)-entry of $T(x)$.
- By convention, we have
  - $T_{0j}(x)$ = the j-th symbol of the input x
  - $T_{i0}(x) = \triangleright$ for $0 \leq i < |x|^k$
  - $T_{i,|x|^k-1}(x) = \sqcup$ for $0 \leq i < |x|^k$.

CIRCUIT VALUE is **P**-Complete I

Theorem

CIRCUIT VALUE is **P**-*Complete.*

Proof.

We know CIRCUIT VALUE is in **P**. It remains to show that any $L \in \mathbf{P}$, there is a reduction R from L to CIRCUIT VALUE.
Let M be a TM deciding L in time n^k. Consider the computation table $T(x)$ of M on input x. Observe that $T_{ij}(x)$ only depends on $T_{i-1,j-1}(x)$, $T_{i-1,j}$, and $T_{i-1,j+1}$. If the cursor is not at $T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}$, $T_{i,j} = T_{i-1,j}$. If the cursor is at one of $T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}$, $T_{i,j}$ may be updated. To determine $T_{i,j}$, it suffices to look at $T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}$!

$T_{i-1,j-1}$	$T_{i-1,j}$	$T_{i-1,j+1}$
$T_{i,j-1}$	$T_{i,j}$	$T_{i,j+1}$

CIRCUIT VALUE is **P**-Complete II

Proof.

Let Γ be the set of symbols appearing on $T(x)$. Encode each symbol $\gamma \in \Gamma$ by a bit vector $(s_1, \ldots, s_{\lceil \log |\Gamma| \rceil})$. We thus have a table of binary entries $S_{ij\ell}$ where $0 \leq i, j \leq |x|^k - 1$ and $1 \leq \ell \leq \lceil \log |\Gamma| \rceil$. Moreover, we know $S_{ij\ell}$ is determined by $S_{i-1,j-1,\ell'}, S_{i-1,j,\ell'}, S_{i-1,j+1,\ell'}$. That is, there are Boolean functions $F_1, F_2, \ldots, F_{\lceil \log |\Gamma| \rceil}$ such that

$$S_{ij\ell} = F_\ell(S_{i-1,j-1,1}, \ldots, S_{i-1,j-1,\lceil \log |\Gamma| \rceil}, S_{i-1,j,1}, \ldots, S_{i-1,j+1,\lceil \log |\Gamma| \rceil}).$$

Observe that F_ℓ are determined by M, regardless of x. Moreover, we can think of each F_i as a circuit. Thus we have a circuit C with $3\lceil \log |\Gamma| \rceil$ inputs (for $T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}$) and $\lceil \log |\Gamma| \rceil$ outputs (for $T_{i,j}$).

CIRCUIT VALUE is **P**-Complete III

Proof.

$$\begin{array}{cccc} \underline{C_{0,0}} & \underline{C_{0,1}} & \cdots & \underline{C_{0,|x|^k-1}} \\ \underline{C_{1,0}} & C_{1,1} & \cdots & \underline{C_{1,|x|^k-1}} \\ & & \vdots & \\ \underline{C_{|x|^k-1,0}} & C_{|x|^k-1,1} & \cdots & \underline{C_{|x|^k-1,|x|^k-1}} \end{array}$$

Our reduction $R(x)$ consists of $(|x|^k-1)(|x|^k-1)$ copies of C. The inputs of $R(x)$ are the encoding of the initial configuration. The output of $R(x)$ is to check if $C_{|x|^k-1,1}$ encodes the state "*yes*."
Note that the circuit C is determined by M (and hence not by the input x). The computation of R needs to count up to $|x|^k$ only. Hence the reduction can be performed in $O(\log|x|)$ space. □

Cook's Theorem

Theorem

SAT is **NP***-complete.*

Proof.

Let $L \in \mathbf{NP}$ and M an NTM deciding L in time n^k. Without loss of generality, we assume each step of M is nondeterministic. Moreover, there are exactly two choices in each nondeterministic step.
As in table method, we construct a circuit (and hence a Boolean expression) for the computation table of M. Now the entry $T_{i,j}$ is determined by $T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}$ *and the choice* c_{i-1}. Thus, the circuit C has $3\lceil\log|\Gamma|\rceil + 1$ inputs. M accepts x iff there is a truth assignment to $c_0, c_1, \ldots, c_{|x|^k-1}$ such that $C_{|x|^k-1,1}$ encodes *yes*. □

Graph-Theoretic Problems

- Let $\mathcal{G}$ be a set of finite graphs (called a graph-theoretic property).
- The computational problem related to $\mathcal{G}$ is: given a graph G, to decide whether $G \in \mathcal{G}$.
- It is not hard to encode any input G as a string in Σ^*.
  - For instance, we can represent the adjacency matrix of G by a string.
- A graph-theoretic problem $\mathcal{G}$ corresponds to a language L.
  - $G \in \mathcal{G}$ iff *encoding*$(G) \in L$.
- Consider a set $\mathcal{G}$ expressible in existential second-order logic.
  - That is, there is an existential second-order logic sentence $\exists P_0 \exists P_1 \cdots \exists P_\ell \phi$ such that

$$\mathcal{G} = \{G : G \vDash \exists P_0 \exists P_1 \cdots \exists P_\ell \phi\}.$$

Deciding Graph-Theoretic Properties I

Theorem

Let $\exists P_0 \exists P_1 \cdots \exists P_\ell \phi$ *be an existential second-order sentence. Given a graph* G *as an input, checking* $G \vDash \exists P_0 \exists P_1 \cdots \exists P_\ell \phi$ *is in* **NP**.

Proof.

Assume P_i has arity r_i. Given $G = (V, E)$ with $|V| = n$, an NTM can guess relations $P_i^M \subseteq V^{r_i}$ for $i = 0, \ldots, \ell$. Note that the time for guessing P_i^M is at most n^{r_i}.
After guessing P_i^M's, we have a first-order logic formula ϕ with relations $P_0, P_1, \ldots, P_\ell$. We now show how to decide $(G, P_0^M, \ldots, P_\ell^M) \vDash \phi$ in polynomial time.
We prove by induction on ϕ.

- If ϕ is atomic, we can check it by examining the adjacency matrix or P_i^M.

Deciding Graph-Theoretic Properties II

Proof.

- If $\phi = \neg\psi$, there is a polynomial time algorithm for ψ by inductive hypothesis. We can decide $\neg\psi$ by exchanging the *yes* and *no* states.
- If $\phi = \psi_0 \vee \psi_1$, there are polynomial time algorithms M_0 and M_1 for ψ_0 and ψ_1 respectively. We decide $\psi_0 \vee \psi_1$ by executing M_0 and then M_1 (if necessary).
- $\phi = \psi_0 \wedge \psi_1$ is similar.
- If $\phi = \forall x \psi$, there is a polynomial time algorithm M for ψ. We construct a new model H that assigns x to v and check $H \vDash \psi$ by M. If the answer is "*yes*" for all $v \in V$, we return "*yes*;" otherwise we return "*no*." Since M is polynomial in n and there are n iterations, this case can be performed in polynomial time.

□

Characterizing Graph-Theoretic Properties

- Let Ψ be an existential second-order sentence.
- Clearly, Ψ determines a graph-theoretic property.
  - $\mathcal{G}_\Psi = \{G : G \vDash \Psi\}$.
- We have shown that deciding $G \in \mathcal{G}_\Psi$ is in **NP** for any input graph G.
- Now consider a graph-theoretic property $\mathcal{G}$ that can be decided in **NP**.
- Is there an existential second-order sentence Ψ such that $\mathcal{G} = \mathcal{G}_\psi$?
- If so, we can prove a graph-theoretic property is in **NP** by writing an existential second-order logic formula!
  - We thus say that the fragment of existential second-order logic characterizes graph-theoretic properties in **NP**.

Fagin's Theorem I

Theorem

The class of all graph-theoretic properties expressible in existential second-order logic is equal to **NP**.

Fagin's Theorem II

Proof.

Let $\mathcal{G}$ be a graph property in **NP**. Hence there is an NTM M deciding whether $G \in \mathcal{G}$ in time n^k for some k. We will construct a formula $\exists P_0 \cdots \exists P_\ell \phi$ such that $G \vDash \exists P_0 \cdots \exists P_\ell \phi$ iff $G \in \mathcal{G}$. Consider

$$\begin{aligned}
e(m) &= \exists x_0 \exists x_1 \cdots \exists x_{m-1} \textstyle\bigwedge_{0 \le i < j < m} \neg(x_i = x_j) \\
succ &= \forall x \exists x' \neg(x = x') \wedge S(x, x') \\
unique &= \forall x \forall y \forall y' (S(x,y) \wedge S(x,y') \rightarrow y = y') \\
linear &= \forall x \forall y (S(x,y) \rightarrow \neg S(y,x)) \\
\Phi_S &= e(n) \wedge \neg e(n+1) \wedge succ \wedge unique \wedge linear
\end{aligned}$$

Observe that S is isomorphic to $\{(0,1), (1,2), \ldots, (n-2, n-1)\}$.

Fagin's Theorem III

Proof.

Define $\zeta(x) = \forall y \neg S(y,x)$ ("$x = 0$") and $\eta(x) = \forall y \neg S(x,y)$ ("$x = n-1$"). Let $0 \le x_1, x_2, \ldots, x_k < n$. Write $(x_1, x_2, \ldots, x_k)$ as $\vec{x}$. Observe that any number between 0 and $n^k - 1$ is represented by an $\vec{x}$. We define $S_k(\vec{x}, \vec{y})$ to represent $\vec{y}$ is the successor of $\vec{x}$:

$$
\begin{array}{rcl}
S_1(x_1, y_1) & = & S(x_1, y_1) \\
S_j(x_1, \ldots, x_j, y_1, \ldots, y_j) & = & [S(x_j, y_j) \wedge (x_1 = y_1) \wedge \cdots (x_{j-1} = y_{j-1})] \vee \\
 & & [\eta(x_j) \wedge \zeta(y_j) \wedge S_{j-1}(x_1, \ldots, x_{j-1}, y_1, \ldots, y_{j-1})]
\end{array}
$$

In the inductive definition, $S_j(\vec{x}, \vec{y})$ represents $\vec{y} = \vec{x} + 1$ with $|\vec{x}| = |\vec{y}|$. We have $\vec{y} = \vec{x} + 1$ iff (x_1 and y_1 are MSB's)

- $y_j = x_j + 1$ and $\forall i < j (y_i = x_i)$; or
- $x_j = n - 1$, $y_j = 0$, and $(y_1, \ldots, y_{j-1}) = (x_1, \ldots, x_{j-1}) + 1$.

Fagin's Theorem IV

Proof.

Consider the computation table $T(G)$ of M. For each symbol $\sigma \in \Gamma$ (Γ is the set of symbols on $T(G)$), the relation $T_\sigma(\vec{x}, \vec{y})$ means that the $(\vec{x}, \vec{y})$-entry of $T(G)$ is σ. Moreover, $C_0(\vec{x})$ means that the 0-th nondeterministic choice is made at the step $\vec{x}$. Similarly for $C_1(\vec{x})$. The existential second order sentence is of the form:

$$\exists S \exists T_{\sigma_1} \exists T_{\Sigma_2} \cdots \exists T_{\sigma_\ell} \exists C_0 \exists C_1 \forall \vec{x} \forall \vec{x}' \forall \vec{y} \forall \vec{y}' \forall \vec{y}'' (\Phi_S \wedge \Phi_T \wedge \Phi_\Delta \wedge \Phi_C \wedge \Phi_{yes}).$$

Φ_S is the formula specifying the successor relation S. We now define the remaining subformulae.

Fagin's Theorem V

Proof.

In addition to the conventions used in Table Method, we further assume that the adjacency matrix is spread in the input: we put $n^{k-2} - 1$ $\sqcup$'s between two consecutive entries.

$$
\begin{array}{cccccccccccccc}
\triangleright & 0_s & \sqcup & \cdots & \sqcup & 1 & \sqcup & \cdots & \sqcup & 0 & \sqcup & \cdots & \sqcup \\
\triangleright & 0 & \sqcup_q & \cdots & \sqcup & 1 & \sqcup & \cdots & \sqcup & 0 & \sqcup & \cdots & \sqcup \\
 & & & & & & \vdots & & & & & & \\
\triangleright & yes & \sqcup & \cdots & \sqcup & \sqcup & \sqcup & \cdots & \sqcup & \sqcup & \sqcup & \cdots & \sqcup
\end{array}
$$

Fagin's Theorem VI

Proof.

Φ_T specifies the boundary of computation table $T(G)$.

- When $\vec{x} = 0$
  - If $y_2 = \cdots = y_k = 0$, $T_i(\vec{x}, \vec{y})$ iff $G(y_1, y_2) = i$ for $i = 0, 1$;
  - Otherwise, $T_\sqcup(\vec{x}, \vec{y})$.
- When $\vec{y} = 0$, $T_\triangleright(\vec{x}, \vec{y})$;
- When $\vec{y} = n^k - 1$, $T_\sqcup(\vec{x}, \vec{y})$.

Fagin's Theorem VII

Proof.

Φ_Δ specifies transition relations of M on $T(G)$. Recall

$T_{i-1,j-1} = \alpha$	$T_{i-1,j} = \beta$	$T_{i-1,j+1} = \gamma$
$T_{i,j-1}$	$T_{i,j} = \sigma$	$T_{i,j+1}$

Let c be the nondeterministic choice made at step $i-1$. For each $(T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}, c, T_{i,j})$, we add the following conjunct to Φ_Δ:

$$[S_k(\vec{x}', \vec{x}) \wedge S_k(\vec{y}', \vec{y}) \wedge S_k(\vec{y}, \vec{y}'') \wedge T_\alpha(\vec{x}', \vec{y'}) \wedge T_\beta(\vec{x}', \vec{y}) \wedge T_\gamma(\vec{x}', \vec{y}'') \wedge C_c(\vec{x}')] \to T_\sigma(\vec{x}, \vec{y}).$$

Fagin's Theorem VIII

Proof.

Φ_C specifies the nondeterministic choice at any step.

$$(C_0(\vec{x}) \vee C_1(\vec{x})) \wedge (\neg C_0(\vec{x}) \vee \neg C_1(\vec{x})).$$

Finally Φ_{yes} specifies the accepting configuration.

$$\vec{x} = n^k - 1 \wedge \vec{y} = 1 \to T_{yes}(\vec{x}, \vec{y}).$$

It should be clear that $G \in \mathcal{G}$ iff G satisfies the existential second order sentence constructed above. □

- Spreading the adjacency matrix of the input allows us to have a simple encoding.
  - Otherwise, we have to define $\vec{y} \leq n^2$.
- The formula $S_k(\vec{x}, \vec{y})$ is defined by $S(x_i, y_i)$. Each instance of $S_k(\vec{x}, \vec{y})$ is a new copy.

Fagin's Theorem IX

  - For instance, there are three copies in Φ_Δ.
- Observe that the constructed formula is not in the monadic second order logic.
  - For instance, Φ_S and Φ_Δ define binary relations S and T_σ.

Quantified Boolean Formula

- As we have seen, logic and complexity are closely related.
  - SATISFIABILITY is **NP**-complete (Cook's theorem).
  - Existential second-order logic characterizes **NP** (Fagin's theorem).
- There is yet another connection between logic and complexity.
- The quantified Boolean formula (QBF) problem is the following: Given a Boolean expression ϕ in conjunctive normal form with variables $x_1, x_1, \ldots, x_n$, decide

$$\exists x_1 \forall x_2 \exists x_3 \cdots Q_n x_n \phi?$$

QBF and SATISFIABILITY

- SATISFIABILITY is in fact a subclass of QBF.
  - Let $\phi(x_1, x_2, \ldots, x_n)$ be a Boolean expression in conjunctive normal form with variables $x_1, \ldots, x_n$.
  - $\phi(x_1, x_2, \ldots, x_n)$ is satisfiable iff $\exists x_1 \forall y_1 \exists x_2 \forall y_2 \cdots \exists x_n \phi(x_1, \ldots, x_n)$.
- Since this is a reduction, QBF is **NP**-hard.

QBF is **PSPACE**-Complete I

Theorem

QBF is ***PSPACE****-complete.*

Proof.

Consider any quantified Boolean formula $\exists x_1 \forall x_2 \exists x_3 \cdots Q_n x_n \phi$. Given any truth assignment to $x_1, \ldots, x_n$, we can evaluate ϕ in $O(n)$ space. Moreover, $O(n)$ space is needed to record each assignment. Hence QBF is in **PSPACE**.
Suppose L is a language decided by an NTM M in polynomial space. Thus there are at most 2^{n^k} configurations of M on input $|x| = n$. We thus encode each configuration of M on input x by a bit vector of length n^k.

QBF is **PSPACE**-Complete II

Proof.

Let $A = \{a_1, \ldots, a_{n^k}\}$ and $B = \{b_1, \ldots, b_{n^k}\}$ be sets of Boolean variables. We will construct a quaitified Boolean formula ψ_i with free variables in $A \cup B$ such that $\psi_i(A, B)$ is satisfied by ν iff $(\nu(a_1), \ldots, \nu(a_{n^k})) \xrightarrow{M}{}^{*} (\nu(b_1), \ldots, \nu(b_{n^k}))$ in 2^i steps.
For $i = 0$, $\psi_0(A, B)$ states that

- $a_j = b_j$ for all j; or
- configuration B follows from A in one step.

$\psi_0(A, B)$ can be written in disjunctive normal form with $O(n^k)$ disjuncts, and each disjunct contains $O(n^k)$ literals. That is, $\psi_0(A, B)$ is in fact in disjunctive normal form.

QBF is **PSPACE**-Complete III

Proof.

Inductively, assume we have $\psi_i(A, B)$. Define

$$\psi_{i+1}(A, B) = \exists Z \forall X \forall Y [((X = A \wedge Y = Z) \vee (X = Z \wedge Y = B)) \rightarrow \psi_i(X, Y)]$$

where each of X, Y, Z has fresh n^k variables.
However, ψ_{i+1} is not in the form required by QBF. It is not in prenex normal form. But this is easy to fix. Note that

$$P \rightarrow \exists Z \forall X \forall Y [R(X, Y, Z)] \equiv \exists Z \forall X \forall Y [P \rightarrow R(X, Y, Z)].$$

We can easily transform ψ_{i+1} into its prenex normal form.

QBF is **PSPACE**-Complete IV

Proof.

The other problem is that

$$((X = A \wedge Y = Z) \vee (X = Z \wedge Y = B)) \to \psi_i(X, Y)$$

is not in conjunctive normal form.
Note that the disjunctive normal form is easy to compute. Recall that ψ_0 is in disjunctive normal form. Assume ψ_i is in disjunctive normal form. Our goal is to compute the disjunctive normal form of the following formula:

$$((X \neq A \vee Y \neq Z) \wedge (X \neq Z \vee Y \neq B)) \vee \psi_i(X, Y)$$

QBF is **PSPACE**-Complete V

Proof.

Observe that $(X \neq A) \wedge (X \neq Z)$ is equivalent to the following formula:

$$\begin{array}{llll} \bigvee_{1 \leq i,j \leq n^k} (x_i \wedge \neg a_i \wedge x_j \wedge \neg z_j) & \vee & \bigvee_{1 \leq i,j \leq n^k} (\neg x_i \wedge a_i \wedge x_j \wedge \neg z_j) & \vee \\ \bigvee_{1 \leq i,j \leq n^k} (x_i \wedge \neg a_i \wedge \neg x_j \wedge z_j) & \vee & \bigvee_{1 \leq i,j \leq n^k} (\neg x_i \wedge a_i \wedge \neg x_j \wedge z_j) & \end{array}$$

The disjunctive normal form consists of ψ_i and $16n^{2k}$ disjuncts.
We thus have a reduction from any problem in **PSPACE** to the version of QBF in disjunctive normal form. But this version is precisely the complement of QBF. Hence we have a reduction from any problem in **coPSPACE** to QBF. Since **coPSPACE** = **PSPACE** (Immerman-Szelepscényi Theorem), our reduction is in fact from any problem in **PSPACE** to QBF. □

QBF is **PSPACE**-Complete VI

- If ψ_{i+1} were defined to be $\exists Z[\psi_i(A, Z) \wedge \psi(Z, B)]$, the size of the formula is doubled. The reduction could not be performed in polynomial time.
  - That is why we "reuse" the formula ψ_i.