Theory of Computing Nice Notes

Cara Bennett

Fall 2022

Contents

1 Regular Languages **2**
- 1.1 Properties . . . 2
- 1.2 Finite Automata . . . 2
- 1.3 Regular Expressions . . . 3
- 1.4 Examples . . . 3
- 1.5 Non-Regular Languages . . . 4

2 Context-Free Languages **4**
- 2.1 Properties . . . 4
- 2.2 Context-Free Grammars . . . 4
- 2.3 Pushdown Automata . . . 4
- 2.4 Examples . . . 5
- 2.5 Non-Context-Free Languages . . . 5

3 Recognizable Languages **5**
- 3.1 Properties . . . 6
- 3.2 Turing Machines . . . 6
  - 3.2.1 Variants . . . 6
  - 3.2.2 Universal Turing Machine . . . 6
- 3.3 Non-Recognizable Languages . . . 7

4 Decidable Languages **7**
- 4.1 Properties . . . 7
- 4.2 Enumerators . . . 7
- 4.3 Non-Decidable Languages . . . 8
  - 4.3.1 Rice's Theorem . . . 8
- 4.4 Reducibility . . . 9

5 Complexity Theory **9**
- 5.1 Time Complexity . . . 9
- 5.2 Space Complexity . . . 9
- 5.3 NP . . . 10
- 5.4 Reducibility . . . 10
- 5.5 NP-Complete . . . 11
  - 5.5.1 Satisfiability Problem . . . 11
  - 5.5.2 Coloring . . . 11
  - 5.5.3 Hamiltonian Cycles . . . 12

1 Regular Languages

Definition 1.1 (Language)**.** A **language** is a set of strings.

Definition 1.2 (Regular)**.** A language is **regular** if some finite automaton recognizes it.

1.1 Properties

Theorem 1. Every finite language is regular.

Definition 1.3 (Complement)**.** The **complement** of a language A is $\overline{A} = \{w : w \notin A\}$.

Theorem 2. The class of regular languages is closed under the complement operation.

Definition 1.4 (Regular Operations)**.** Let A and B be languages. The **regular operations** (in the order of precedence) are

1. **Star:** $A^* = \{x_1 x_2 \dots x_k : k \geq 0 \text{ and each } x_i \in A\}$.
2. **Concatenation:** $AB = \{xy : x \in A \text{ and } y \in B\}$.
3. **Union:** $A \cup B = \{x : x \in A \text{ or } x \in B\}$.

Theorem 3. The class of regular languages is closed under the regular operations.

Proof. Assume A and B are regular; then there exists some NFA M_A and M_B which recognize A and B, respectively. We will construct an NFA M which recognizes each operation:

1. **Star:** Create a new start state q_0 which transitions to the start state of M_A on input ε. The start state q_0 is also an accept state. Add additional ε transitions to the accept states of M_A which go to the start states of M_A. Then M accepts its input whenever the input can be broken into pieces which would be accepted by M_A.
2. **Concatenation:** Alter the transition function so that the accept states of M_A have additional ε arrows that nondeterministically allow branching to M_B. The accept states of M are the accept states of M_B. The start state of M is the start state of M_A.
3. **Union:** Create a new start state q_0. Alter the transition function so that q_0 goes to the start states of M_A and M_B on ε. The accept starts of M are the accept states of M_A and M_B.

□

1.2 Finite Automata

Definition 1.5 (Finite Automaton)**.** A **(deterministic) finite automaton (DFA)** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

1. Q is a finite set called the **states**,
2. Σ is a finite set called the **alphabet**,
3. $\delta : Q \times \Sigma \to Q$ is the **transition function**,
4. $q_0 \in Q$ is the **start state**, and
5. $F \subseteq Q$ is the **set of accept states**.

Definition 1.6 (Nondeterministic Finite Automata)**.** A **nondeterministic finite automaton (NFA)** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Theorem 4. Every NFA has an equivalent DFA.

Proof. Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA. We will construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ which simulates N. The main idea is that M will have states corresponding to each subset of states in N. The transition function of M will sends a state to the state corresponding to the subset of states which N would go to. Formally,

1. $Q' = \mathcal{P}(Q)$,
2. $\delta'(R, a) = \{q \in Q : q \in \delta(r, a) \text{ for some } r \in R\}$ for $R \in Q'$ and $a \in \Sigma^*$,
3. $q_0' = \{q_0\}$, and
4. $F' = \{R \in Q' : R \text{ contains an accept state of } N\}$.

Some additional details are needed to account for ε transitions. □

1.3 Regular Expressions

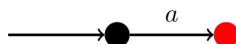
Definition 1.7 (Regular Expression)**.** A **regular expression** R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $R_1 \cup R_2$, where R_1 and R_2 are regular expressions,
5. $R_1 R_2$, where R_1 and R_2 are regular expressions, or
6. R_1^*, where R_1 is a regular expression.

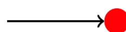
Theorem 5. A language is regular if and only if some regular expression describes it.

Proof. ($\Longleftarrow$) If a language is recognized by a NFA, then it is regular. We will show that we can construct an NFA for any regular expression. Consider the six cases in the definition of a regular expression.

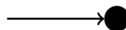
1. Then $L(R) = \{a\}$, and the following NFA recognizes $L(R)$:



2. Then $L(R) = \{\varepsilon\}$, and the following NFA recognizes $L(R)$:



3. Then $L(R) = \emptyset$, and the following NFA recognizes $L(R)$:



4-6. Use the constructions described in the proof of Theorem 3.

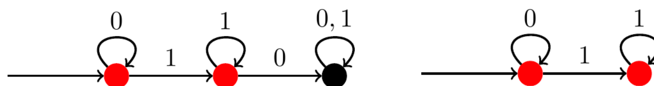
□

1.4 Examples

Example 1.1. Let $L_1 = \{0^k 1^j : k \geq 0 \text{ and } j \geq 0\}$ be the set of binary strings consisting of a string of 0s followed by a string of 1s. A regular expression describing L_1 is

$$0^* 1^*.$$

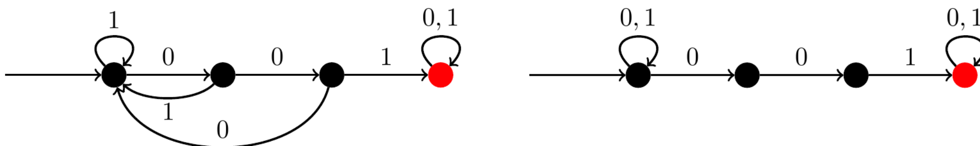
The following DFA (left) and NFA (right) recognize L_1:



Example 1.2. Let $L_2 = \{w : 001 \in w\}$ be the set of binary strings containing 001. A regular expression describing L_2 is

$$(0 \cup 1)^* 001 (0 \cup 1)^*.$$

The following DFA (left) and NFA (right) recognize L_2:



1.5 Non-Regular Languages

Theorem 6 (Pumping Lemma)**.** If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Proof. Let $M = (Q, \Sigma, \delta, q, F)$ be a DFA recognizing A. Set $p = |Q|$. Let $s = s_1 s_2 \dots s_n$ be a string $s \in A$ with $|s| \geq p$. (If no such string exists, then the theorem is vacuously true). Let $r = r_0, r_1, \dots, r_n$ be the sequence of states such that $r_0 = q$ and $r_{i+1} = \delta(r_i, s_{i+1})$. Since $|r| > p$, there must exist j, l such that $r_j = r_l$ and $0 \leq j < l \leq p$. Set $x = s_1 \dots s_{j-1}$, $y = s_j \dots, s_{l-1}$, and $z = s_l \dots, s_n$. □

2 Context-Free Languages

Definition 2.1 (Context-Free)**.** A language is **context-free** if it can be generated by some context-free grammar.

2.1 Properties

Theorem 7. The class of context-free languages is closed under union, concatenation, and star, but **not** intersection or complement.

Theorem 8. Every regular language is context free.

Theorem 9. The intersection of a regular language and a context-free language is a context-free language.

2.2 Context-Free Grammars

Definition 2.2 (Context-Free Grammar)**.** A **context-free grammar (CFG)** is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the **variables**,
2. Σ is a finite set, disjoint from V, called the **terminals**,
3. R is a finite set of **rules**, with each rule being a variable and a string of variable and terminals, and
4. $S \in V$ is the start variable.

2.3 Pushdown Automata

Definition 2.3 (Pushdown Automaton)**.** A **pushdown automaton (PDA)** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

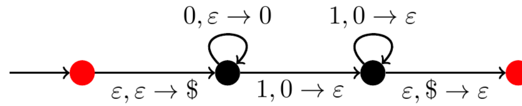
Theorem 10. A language is context free if and only if some pushdown automata recognizes it.

2.4 Examples

Example 2.1. Let $L_1 = \{0^k 1^k : k > 0\}$ be the set of words starting with k 0s followed by k 1s. Then a context free grammar which generates L_1 is:

$$S \to 0S1 \mid \varepsilon.$$

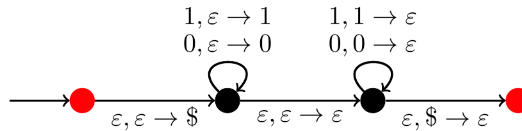
A pushdown automaton which recognizes L_1 is:



Example 2.2. Let $L_2 = \{ww^R : w \in (0 \cup 1)^*\}$ be the set of binary strings which are palindromes and have even length. Then a context free grammar which generates L_2 is:

$$S \to 0S0 \mid 1S1 \mid \varepsilon.$$

A pushdown automaton which recognizes L_2 is:



2.5 Non-Context-Free Languages

Theorem 11 (Pumping Lemma for Context-Free Languages)**.** If A is a context-free language, then there is a number p (the pumping length) where, if s is a string in A of length at least p, then s may be divided into five pieces $s = uvxyz$ satisfying the conditions

1. for each $i \geq 0$, $uv^i xy^i z \in A$,
2. $|vy| > 0$, and
3. $|vxy| \leq p$.

Proof. Let G be a CFL and A be the language generated by G. Let s be a very long string in A. Since $s \in A$, it can be generated by G and has a parse tree. The parse tree for s must be very tall since s is very long. Then there must exist some symbol R which appears twice. This repetition allows us to replace the subtree under the second occurrence of R with copies of the subtree of the first occurrence and still have a legal parse tree. □

3 Recognizable Languages

Definition 3.1 (Recognizable)**.** A language is **recognizable** or **recursively-enumerable** if some Turing machine recognizes it.

Definition 3.2 ($\mathcal{RE}$)**.** The class of recursively-enumerable languages is denoted $\mathcal{RE}$.

3.1 Properties

Theorem 12. The class $\mathcal{RE}$ is closed under union, intersection, concatenation, and star, but **not** complement.

Theorem 13. Every context-free language is recursively enumerable.

3.2 Turing Machines

Definition 3.3. A **Turing machine** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where

1. Q is the finite set of states,
2. Σ is the input alphabet not containing the **blank symbol** $*$,
3. Γ is tape alphabet, where $* \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in F$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

3.2.1 Variants

Theorem 14. Every one-way tape Turing machine has an equivalent two-way tape Turing machine.

Proof. Let M be a two-way tape TM. We will show how to convert M into an equivalent one-way tape TM, by simulating M with S. Assign a number to each position in the tape of S (on the right) as such:

...	-3	-2	-1	0	1	2	3	...

$\longrightarrow$

0	1	-1	2	-2	3	-3	...

While on the positive side of M's tape, moving once to the right [resp. left] corresponds to moving twice to the right [resp. left] on S's tape. While on the negative side of M's tape, moving once to the right [resp. left] corresponds to moving twice to the left [resp. right] on S's tape. The only exceptions occur at positions 0 and 1, which have special cases. $\square$

Definition 3.4 (k-tape Turing Machine)**.** A **k-tape Turing machine** is a Turing machine with k tapes and k heads for reading and writing. The transition function is $\delta : Q \times \Gamma^k \to G \times \Gamma^k \times \{L, R\}^k$.

Definition 3.5 (Nondeterministic Turing Machine)**.** A **nondeterministic Turing machine** is a Turing machine that has a transition function of the form $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$.

Definition 3.6 (Stay Turing Machine)**.** A **stay Turing machine** is a Turing machine with transition function $\delta : Q \times \Gamma \to G \times \Gamma \times \{L, R, S\}$ where S means to stay put.

Theorem 15. Every non-stay, deterministic, one-tape Turing machine has an equivalent:

- nondeterministic Turing machine,
- k-tape Turing machine, and
- stay Turing machine.

3.2.2 Universal Turing Machine

Definition 3.7 (Universal Turing Machine)**.** The **universal Turing machine** is a Turing machine which recognizes every other Turing machine.

Theorem 16. There exists a 3-tape universal Turing machine U such that on input $w\#s$ where $w, s \in \Sigma^*$ and M_w is a Turing machine, U simulates M_w on s.

3.3 Non-Recognizable Languages

Example 3.1. The set of all Turing machines is countable since the set of all strings Σ^* is countable and each Turing machine has an encoding into a string. Then the set of all recursively-enumerable languages $L_1, L_2, L_3, \ldots$ is countable. Create the following table which denotes the words contained in each language:

	ε	0	1	00	01	10	10	11	$\cdots$
L_1	n	n	n	n	n	n	n	n	
L_2	n	y	n	y	y	n	n	n	
L_3	y	y	y	y	y	y	y	y	
$\vdots$									

Let L' be the language containing the complement of the diagonal; $w_i \in L' \iff w_i \notin L_i$. Then L' is not in the table, so L' must not be recursively enumerable.

Example 3.2 (Tiling Problem)**.** A tile is a square with a color on each side. A kit is a set of tiles. The problem is: given a kit, is it possible to construct a complete tiling of the plane using every tile? This problem is not in $\mathcal{RE}$.

Theorem 17 (Diagonal Language)**.** The diagonal language $L_d = \{w : M_w \text{ is a TM and } w \notin L(M_w)\} \notin \mathcal{RE}$.

Proof. Suppose $L_d \in \mathcal{RE}$. Then there exists a TM M_w with code w which recognizes L_d. So $L(M_w) = L_d$. If $w \in L_d$, then $w \notin L(M_w) = L_d$, a contradiction. It follows that $w \notin L_d$, meaning M_w is not a TM or $w \in L(M_w) = L_d$. Neither of these cases are possible, creating another contradiction. □

4 Decidable Languages

Definition 4.1 (Decidable)**.** A language is **decidable** or **recursive** if some Turing machine recognizes it and halts on all inputs.

Definition 4.2 ($\mathcal{R}$)**.** The class of recursive languages is denoted $\mathcal{R}$.

4.1 Properties

Theorem 18. The class $\mathcal{R}$ is closed under union, intersection, complement, concatenation, and star.

Theorem 19. Every recursive language is recursively-enumerable: $\mathcal{R} \subseteq \mathcal{RE}$.

Theorem 20. A language $L \in \mathcal{R}$ if and only if $L \in \mathcal{RE}$ and $\overline{L} \in \mathcal{RE}$.

Proof. ($\Longrightarrow$) Assume $L \in \mathcal{R}$. Then $\overline{L} \in \mathcal{R}$ since $\mathcal{R}$ is closed under complement. Then $L \in \mathcal{RE}$ and $\overline{L} \in \mathcal{RE}$ since $\mathcal{R} \subseteq \mathcal{RE}$.

($\Longleftarrow$) Assume $L \in \mathcal{RE}$ and $\overline{L} \in \mathcal{RE}$. Then there exists TMs M_1 and M_2 which recognize L and $\overline{L}$. Construct a TM M which runs M_1 and M_2 in parallel, and accepts if M_1 accepts and rejects if M_2 accepts. Then M decides L, so $L \in \mathcal{R}$. □

Corollary 1. If $L \in \mathcal{RE}$ and $L \notin \mathcal{R}$, then $\overline{L} \notin \mathcal{RE}$.

4.2 Enumerators

Definition 4.3 (Enumerator)**.** An **enumerator** E is a Turing machine with an attached printer; the language enumerated by E is the collection of all the strings that it eventually prints out.

Theorem 21. A language is in $\mathcal{RE}$ if and only if there exists an enumerator which enumerates all the words in the language.

Proof. ($\implies$) Assume that the TM M recognizes A. Let $s_1, s_2, s_3 \dots$ be a list of all possible strings in Σ^*. The enumerator E does the following: "For $i = 1, 2, 3, \dots$, run M for i steps on each input $s_1, s_2, \dots, s_i$. If M accepts s_j for any j, then print the string." Since any string accepted by M will eventually be printed, E enumerates A.

($\impliedby$) Assume that the enumerator E enumerates A. The TM M does the following: "On input w, run E. Every time E prints a string, compare it with w. If the string equals w, then accept." Then M recognizes A. □

Theorem 22. A language is in $\mathcal{R}$ if and only if there exists an enumerator which enumerates all the words in the language in increasing order.

Proof. ($\implies$) Assume that the TM M decides A. Let $s_1, s_2, s_3 \dots$ be a list of all possible strings in Σ^* in increasing order. The enumerator E does the following: "For $i = 1, 2, 3, \dots$, run M on s_i. If M accepts, then print s_i." This prints the words in A in increasing order, so E enumerates A.

($\impliedby$) Assume that the enumerator E enumerates A. The TM M does the following: "On input w, run E. Every time E prints a string, compare it with w. If the string equals w, then accept. If the string is greater than w, then reject." This is guaranteed to halt, so M decides A. □

4.3 Non-Decidable Languages

Example 4.1 (Post-Correspondence Problem)**.** A pair is $\binom{w_1}{w_2}$ where $w_1, w_2 \in \Sigma^*$. A kit is a collection of pairs. The problem is: given a kit, is it possible to find a sequence of pairs such that the concatenation of words on top equals the concatenation of the words on bottom? This problem is in $\mathcal{RE} \setminus \mathcal{R}$.

Theorem 23. Let $A_{TM} = \{w\#s : M_w \text{ is a TM and } s \in L(M_w)\}$. Then $A_{TM} \in \mathcal{RE} \setminus \mathcal{R}$.

Proof. ($A_{TM} \in \mathcal{RE}$) The following Turing machine recognizes A_{TM}: "On input $w\#s$, check whether M_w is a TM. If not, then reject. Otherwise, simulate M_w on input s using a UTM. If M_w enters an accept state, then accept."

($A_{TM} \notin \mathcal{R}$) Suppose $A_{TM} \in \mathcal{R}$ and M is a TM which decides A_{TM}. Let M' be a TM which does the following: "On input $w\#s$, check whether M_w is a TM. If not, then reject. Otherwise, feed $w\#w$ to M. Reject if and only if M accepts." Then M' decides L_d, which is a contradiction since $L_d \notin \mathcal{RE}$. □

Theorem 24. Let $HALT_{TM} = \{w\#s : M_w \text{ is a TM and } M_w \text{ halts on } s\}$. Then $HALT_{TM} \in \mathcal{RE} \setminus \mathcal{R}$.

Proof. ($HALT_{TM} \in \mathcal{RE}$) The following Turing machine recognizes $HALT_{TM}$: "On input $w\#s$, check whether M_w is a TM. If not, then reject. Otherwise, simulate M_w on input s using a UTM. If M_w enters an accept or reject state, then accept."

($HALT_{TM} \notin \mathcal{R}$) Suppose $HALT_{TM} \in \mathcal{R}$ and M is a TM which decides $HALT_{TM}$. Let M' be at TM which does the following: "On input $w\#s$, run M on $w\#s$. If M rejects, then M' rejects. If M accepts, then M' runs $w\#s$ on the UTM and accepts or rejects according to the decision of the UTM." Then M' decides A_{TM}, a contradiction. □

4.3.1 Rice's Theorem

Definition 4.4 (Language Property)**.** A **non-trivial language property** of $\mathcal{RE}$ languages is a subset $P \subseteq \mathcal{RE}$ such that $P \neq \emptyset$ and $P \neq \mathcal{RE}$.

Theorem 25 (Rice)**.** For every non-trivial property P of $\mathcal{RE}$ languages

$$L_P = \{w : M_w \text{ is a TM and } L(M_w) \in P\} \notin \mathcal{R}.$$

4.4 Reducibility

Definition 4.5. A function $f : \Sigma^* \to \Sigma^*$ is a **computable function** if some Turing machine, on every input w, halts with just $f(w)$ on its tape.

Definition 4.6. A language A is **mapping reducible** to language B, written $A \leq_M B$, if there is a computable function $f : \Sigma^* \to \Sigma^*$, where for every w,

$$w \in A \iff f(w) \in B.$$

Theorem 26. If $A \leq_M B$ and $B \in \mathcal{R}$ [resp. $B \in \mathcal{RE}$], then $A \in \mathcal{R}$ [resp. $A \in \mathcal{RE}$].

Theorem 27. If $A \leq_M B$ and $A \notin \mathcal{R}$ [resp. $A \notin \mathcal{RE}$], then $B \notin \mathcal{R}$ [resp. $B \notin \mathcal{RE}$].

5 Complexity Theory

Definition 5.1 (Big-O)**.** Let f and g be functions. If positive integers c and n_0 exist such that $f(n) \leq c \cdot g(n)$ for every integer $n \geq n_0$, then $\boldsymbol{f(n) = O(g(n))}$.

5.1 Time Complexity

Definition 5.2 (Time Bounded)**.** Let $t : \mathbb{Z}^+ \to \mathbb{Z}^+$. A Turing machine M is $t(n)$ **time bounded** if M makes at most $t(n)$ steps on any input of length at most n.

Definition 5.3 (Time Complexity Class)**.** The **time complexity class TIME**$(t(n)) \subseteq \mathcal{R}$ is the collection of all languages L that are recognizable by an $O(t(n))$ time-bounded Turing machine.

Definition 5.4 (P)**.** The class **P** is the collection of languages that are recognizable in polynomial time on a deterministic, single-tape Turing machine:

$$\mathrm{P} = \bigcup_{k \geq 1} \mathrm{TIME}(n^k).$$

Theorem 28. Every $t(n)$ time k-tape Turing machine has an equivalent $O(t^2(n))$ time one-tape Turing machine.

Theorem 29. Every $t(n)$ time nondeterministic one-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic one-tape Turing machine.

5.2 Space Complexity

Definition 5.5 (Space Bounded)**.** Let $s(n) : \mathbb{Z}^+ \to \mathbb{Z}^+$. A Turing machine M is $s(n)$ **space bounded** if M uses at most $s(n)$ space on the work tapes on any input of length at most n.

Definition 5.6 (Space Complexity Class)**.** The **space complexity class SPACE**$(t(n)) \subseteq \mathcal{R}$ is the collection of all languages L such that are recognizable by an $O(t(n))$ space-bounded Turing machine.

Definition 5.7 (PSPACE)**.** The class **PSPACE** is the collection of languages that are recognizable in polynomial space on a deterministic, single-tape Turing machine:

$$\mathrm{PSPACE} = \bigcup_{k \geq 1} \mathrm{SPACE}(n^k).$$

Theorem 30 (Space-Time)**.** If $L \in \mathrm{SPACE}(s(n))$, then there exists a constant c where $L \in \mathrm{TIME}(c^{s(n)})$.

Corollary 2. For any function $t(n)$, $\mathrm{TIME}(t(n)) \subseteq \mathrm{SPACE}(t(n))$.

5.3 NP

Definition 5.8 (co-)**.** The **complement** co$X$ of a collection of languages X is

$$\text{co}X = \{L : \overline{L} \in X\}.$$

This is distinct from $\overline{X} = \{L : L \notin X\}$.

Definition 5.9 (Nondeterministic Time Complexity Class)**.** The **nondeterministic time complexity class NTIME**$(t(n))$ is the collection of all languages L that are recognizable by an $O(t(n))$ time bounded nondeterministic Turing machine.

Definition 5.10 (NP)**.** The class **NP** is the collection of languages that are recognizable in polynomial time on a nondeterministic Turing machine:

$$\text{NP} = \bigcup_{k \geq 1} \text{NTIME}(n^k).$$

Theorem 31 (Witness)**.** A language $L \in \text{NP}$ if and only if there exists a language L' consisting of pairs (x, y) where $x \in L \iff (x, y) \in L'$ for some y satisfying $|y| \leq |x|^c$ for some $c > 0$.

Conjecture 1. A problem can be decided in polynomial time if and only if it can be verified in polynomial time:

$$\text{P} = \text{NP}.$$

Conjecture 2. A problem and its complement can be verified in polynomial time if and only if the problem can be decided in polynomial time:

$$\text{P} = \text{NP} \cap \text{coNP}$$

5.4 Reducibility

Definition 5.11 (Polynomial Time Computable Function)**.** A function $f : \Sigma^* \to \Sigma^*$ is a **polynomial time computable function** if some polynomial time Turing machine M exists that halts with just $f(w)$ on its tape, when started on any input w.

Definition 5.12 (Polynomial Time Reducible)**.** A language A is **polynomial time reducible** or **Karp-reducible** to language B, written $A \leq_{\text{P}} B$, if a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ exists, where for every w,

$$w \in A \iff f(w) \in B.$$

Theorem 32 (Properties of Karp-Reducibility)**.** Let A and B be languages. Then:

1. If $A \leq_{\text{P}} B$ and $B \in \text{P}$, then $A \in \text{P}$.
2. If $A \leq_{\text{P}} B$ and $B \in \text{NP}$, then $A \in \text{NP}$.
3. If $A \leq_{\text{P}} B$, then $\overline{A} \leq_{\text{P}} \overline{B}$.
4. If $A \leq_{\text{P}} B$ and $B \in \text{coNP}$, then $A \in \text{coNP}$.
5. If $A \leq_{\text{P}} B$ and $B \in \text{NP} \cap \text{coNP}$, then $A \in \text{NP} \cap \text{coNP}$.
6. If $A \leq_{\text{P}} B$ and $B \leq_{\text{P}} C$, then $A \leq_{\text{P}} C$.

Proof. We prove each property separately:

1. Since $A \leq_{\text{P}} B$, there exists a n^k time bounded TM denoted M_1 that computes f such that $w \in A \iff f(w) \in B$. Since $B \in \text{P}$, there exists a n^l time bounded TM denoted M_2 that decides B. Let M be a TM which does the following: "On input x, compute $f(x)$ using M_1. Then run M_2 on input $f(x)$. If M_2 accepts, then accept; otherwise, reject." Then M decides A and M is n^{kl} time bounded. So $A \in \text{P}$.
2. Same as the proof for 1, but with a NTM instead of a TM.
3. Since $A \leq_{\text{P}} B$, there exists a n^k time bounded TM denoted M_1 that computes f such that $w \in A \iff f(w) \in B$. Then $w \notin A \iff f(w) \notin B$, meaning $w \in \overline{A} \iff f(w) \in \overline{B}$. So $\overline{A} \leq_{\text{P}} \overline{B}$.

4. By property 3, $\overline{A} \leq_{\mathrm{P}} \overline{B}$. Note that coNP $= \{L : \overline{L} \in \mathrm{NP}\}$. It follows that $\overline{B} \in \mathrm{NP}$. By property 2, $\overline{A} \in \mathrm{NP}$ so $A \in \mathrm{coNP}$.
5. By property 2, $A \in \mathrm{NP}$. By property 4, $A \in \mathrm{coNP}$. Therefore, $A \in \mathrm{NP} \cap \mathrm{coNP}$.
6. Since $A \leq_{\mathrm{P}} B$, there exists a n^k time computable function f such that $w \in A \iff f(w) \in B$. Since $B \leq_{\mathrm{P}} C$, there exists a n^j time computable function g such that $w \in B \iff f(w) \in C$. Let $h(x) = g(f(x))$. Then h is a n^{kl} time computable function and $w \in A \iff h(w) \in C$, meaning $A \leq_{\mathrm{P}} C$.

□

5.5 NP-Complete

Definition 5.13 (NP-Hard)**.** A language L is **NP-hard** if $L' \leq_{\mathrm{P}} L$ for all $L' \in \mathrm{NP}$.

Definition 5.14 (NP-Complete)**.** A language L is **NP-complete** if $L \in \mathrm{NP}$ and L is NP-hard.

Theorem 33. If $A \leq_{\mathrm{P}} B$ and A is NP-hard, then B is NP-hard.

Theorem 34. If there exists an NP-complete language L with $L \in \mathrm{P}$, then $\mathrm{NP} = \mathrm{P}$.

5.5.1 Satisfiability Problem

Definition 5.15 (SAT)**.** A Boolean formula is **satisfiable** if some assignment of 0s and 1s to the variables makes the formula evaluate to 1. The **satisfiability problem** is

$$\mathrm{SAT} = \{\phi : \phi \text{ is a satisfiable Boolean formula}\}.$$

Definition 5.16 (3-SAT)**.** A **literal** is a Boolean variable or a negated Boolean variable. A **clause** is several literals connected with $\vee$s. A Boolean formula is in **conjuctive normal form (cnf)**, if it is made of clauses connected with $\wedge$s. It is a **3cnf-formula** if all the clauses have three literals. Then

$$3\text{-SAT} = \{\phi : \phi \text{ is a satisfiable 3cnf-formula}\}.$$

Theorem 35. SAT is NP-complete.

Corollary 3. 3-SAT is NP-complete.

5.5.2 Coloring

Definition 5.17 (k-COLOR)**.** The set k**-COLOR** contains graphs G such that there exists a proper coloring of G with k colors.

Theorem 36. 3-COLOR is NP-complete.

Corollary 4. 4-COLOR is NP-complete.

Proof. (4-COLOR $\in$ NP) A proper 4-coloring of G is a witness.

(4-COLOR is NP-hard) We will show 3-COLOR $\leq_{\mathrm{P}}$ 4-COLOR. Given a graph G, let $f(G)$ be the same graph with an additional vertex which is adjacent to every other vertex in G. Then G is 3-colorable if and only if $f(G)$ is 4-colorable. □

Definition 5.18 (MAXSTABLE)**.** The set **MAXSTABLE** contains (G, k) if G is a graph with an independent set U such that $|U| = k$.

Corollary 5. MAXSTABLE is NP-complete.

Proof. (MAXSTABLE $\in$ NP) An independent set of size k is a witness.

(MAXSTABLE is NP-hard) We will show that 3-COLOR $\leq_P$ MAXSTABLE. Given a graph G, let $f(G) = (G', |V(G)|)$ where G' is composed of three copies of G in which copies of the same vertex are mutually adjacent. Then G is 3-colorable if and only if G' has an independent set of size $|V(G)|$. $\square$

Definition 5.19 (MAXCLIQUE)**.** The set **MAXCLIQUE** contains (G, k) if G is a graph with a clique of size k.

Corollary 6. MAXCLIQUE is NP-complete.

Proof. (MAXCLIQUE $\in$ NP) An clique of size k is a witness.

(MAXCLIQUE is NP-hard) We will show that MAXSTABLE $\leq_P$ MAXCLIQUE. Let $f((G, k)) = (\overline{G}, k)$ where $\overline{G}$ is the complement graph of G. Note that G contains an independent set of size k if and only if $\overline{G}$ contains a clique of size k. $\square$

5.5.3 Hamiltonian Cycles

Definition 5.20 (HAMCYCLE, HAMPATH)**.** The set **HAMCYCLE** [resp. **HAMPATH**] contains G if there is a cycle [resp. path] which contains every vertex of G exactly once.

Theorem 37. HAMPATH is NP-complete.

Corollary 7. HAMCYCLE is NP-complete.

Proof. (HAMCYCLE $\in$ NP) A cycle containing every vertex exactly once is a witness.

(HAMCYCLE is NP-hard) We will show that HAMPATH $\leq_P$ HAMCYCLE. Given a graph G, let $f(G)$ be the graph G with additional vertex v such that v is adjacent to every vertex in $V(G)$. Then G has a Hamiltonian path if and only if $f(G)$ has a Hamiltonian cycle. $\square$