CS423 Finite Automata & Theory of Computation

MW 8:00 - 9:20 in Blow 333

Prof. Weizhen Mao, wm@cs.wm.edu
General Information

- Office Hours: On Zoom. MW 3:00 - 4:30 or by email.
- Grader: TBD
- Prerequisites/background: Linear algebra, Data structures and algorithms, and Discrete math
Use of Blackboard

▶ Announcements
▶ Homework
▶ Solution sets
▶ Lecture notes
▶ Recorded lectures
▶ Grades (HWs, Midterm, Final)
▶ Check at least weekly
Lecture Notes (in various forms and places)
- Lecture slides: http://www.cs.wm.edu/~wm/
- Lecture notes posted on BB (A high-level summary)
- Recorded lecture posted on BB
- Your own class notes

Course Organization
- Automata theory: 50%
- Computability theory: 25%
- Complexity theory: 25%
Grading

► 10 homework assignments: 35%
► Midterm: 25%
► Final: 40%

Grading Policy

► [90, 100]: A or A-
► [80, 90): B+, B, or B-
► [70, 80): C+, C, or C-
► [60, 70): D+, D, or D-
► [0, 60): F
Homework Submission Policy and Accommodation

- Your homework needs to be typeset in LaTeX with all figures drawn nicely and inserted. The pdf file is expected to be submitted to BB before or on the due date.
- However, since we are in a pandemic semester, I will allow one late day for each homework to all students who need the extra time.
- In calculating the final grades, homework with the lowest grade will be dropped.
- Additional extensions may be permitted for illness and family emergency. Requests must be made prior to the due date.
Homework Completion Policy

▶ Homework must be typeset in LaTeX. Nothing should be handwritten, including diagrams and tables
▶ Empty-hand policy when you discuss homework problems with your classmates
▶ List your collaborators for every homework problem
▶ Cite all references you used to solve a problem
▶ In no case you should copy verbatim from other people’s work without proper attribution, as this is considered plagiarism, thus a violation of the Honor Code
Exams

▶ Midterm: Timed, on BB, and likely in mid-March
▶ Final Exam: 9:00-12:00, May 11

Other Important Dates

▶ Jan. 26: First day of class
▶ Jan. 26 - Feb. 4: Add/Drop
▶ Mar. 9 - 11: Midterm on BB
▶ Mar. 12 - 20: Spring Break
▶ Mar. 28: Last day to withdraw
▶ May 6: Last day of class
▶ May 11: Final exam
▶ May 20 - 22: Commencement
Writing Assignments for CSci423W

▶ A paper of 2 to 3 pages about a someone pioneering the field of ToC and one of his/her major research contributions, written in the IEEE standard format for CS conferences. The title of the paper should be in the format of “name and result”, e.g., “Alan Turing and the Enigma”, or “Stephen Cook and the Satisfiability Problem”, or “Rivest, Shamir, and Adleman and the RSA Cryptosystem”. More details after Spring break.

▶ The class will be divided into ten groups, each being responsible for the production of one homework solution set. More details later.
Honor Code

▶ "As a member of the William and Mary community, I pledge on my honor not to lie, or steal, either in my academic or personal life. I understand that such acts violates the Honor Code and undermine the community of trust, of which we are all stewards."

▶ Academic honesty, the cornerstone of teaching and learning, lays the foundation for lifelong integrity. The Honor Code is, as always, in effect in this course. Please go to the "Honor System" page in the wm.edu site to read the complete honor code document.
Student Accessibility Services
William & Mary accommodates students with disabilities in accordance with federal laws and university policy. Any student who feels they may need an accommodation based on the impact of a learning, psychiatric, physical, or chronic health diagnosis should contact Student Accessibility Services staff at 757-221-2512 or at sas@wm.edu to determine if accommodations are warranted and to obtain an official letter of accommodation. For more information, please go to www.wm.edu/sas
Please inform me privately of any accommodations you have arranged with the Office of SAS.

My goal: I will try my best to be understanding, flexible, fair, and helpful, while creating a high quality learning experience for all of you.
Covid Semester

- For this course in spring 2022, here is the way that we will address student absences: a student’s absence itself will not be part of the student’s grade; I will also record corresponding lectures and upload them to Blackboard so that the student can access the missing ones; the student should also communicate with me on how to handle possible delay of homework etc.

- For this course in spring 2022, here is the way we will address instructor absence: If only a couple of lectures are missed, I will make up some of the missing ones through recorded videos; if too many lectures are missed, the department will get another instructor involved.
1.1 Three areas of ToC (*Sipser 0.1*)
Theory of Computation is to study the fundamental capabilities and limitations of computers. It contains three areas.

- **Automata theory**: Models of computation. Seeking a precise but concise definition of a computer.

- **Computability theory**: What can and cannot a computer do? Computationally unsolvable versus computationally solvable problems. Determining whether a problem is unsolvable by computer algorithms.

- **Complexity theory**: What can a computer do efficiently? Computationally hard versus computationally easy problems. For example, factorization versus sorting. Cryptography needs hard problems such as factorization to ensure security.
1.2 Sets and Languages (Sipser 0.2)
Languages are central concepts in automata theory.

- **Alphabet** $\Sigma$: Finite and nonempty, e.g., $\Sigma = \{0, 1\}$ and $\Sigma = \{a, b, \ldots, z\}$.

- **String** (or **word**), e.g., $w = 01110$; empty string $\varepsilon$; length of a string, $|w|$; concatenation of two strings $w_1w_2$; reverse of a string $w^R$, and substring of a string.

- **Language**: A language is a set of strings, e.g., $\{\varepsilon\}$, $\emptyset$, $\Sigma$, $A = \{w \mid w \text{ has an equal number of 0s and 1s}\} = \{\varepsilon, 01, 10, 0011, 0110, 1010, \ldots\}$, and $B = \{0^n1^n \mid n \geq 1\} = \{01, 0011, 000111, \ldots\}$. Note $B \subset A$. 

Regular operators: Let $A$ and $B$ be two languages.

- **Union:** $A \cup B = \{ x \mid x \in A \text{ or } x \in B \}$.
- **Concatenation:** $A \cdot B = \{ xy \mid x \in A \text{ and } y \in B \}$.

**Example:** If $A = \{01, 10\}$ and $B = \{0, 1\}$, then $AB = \{010, 011, 100, 101\}$.

- **Star:**
  
  $A^* = \{ x_1 x_2 \cdots x_k \mid \text{all } k \geq 0 \text{ and each } x_i \in A \text{ for } i = 1, 2, \ldots, k \}$.  

  Note $\varepsilon \in A^*$

**Example:** If $A = \{0, 1\}$, then $A^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 111, \ldots\}$

**Example:** If $A = \{01, 11\}$, then $A^* = \{\varepsilon, 01, 11, 0101, 1101, \ldots, 11110111, \ldots\}$
More operators

- **Intersection:** \( A \cap B = \{ x \mid x \in A \text{ and } x \in B \} \)
- **Complement:** \( \bar{A} \) is the set of all elements under consideration that are not in \( A \). Usually, \( \bar{A} = \Sigma^* - A \).
- **Difference:** \( A - B = A \cap \bar{B} \).

**Power of language:**
\[
A^k = \{ x_1 x_2 \cdots x_k \mid x_i \in A \text{ for } i = 1, 2, \cdots, k \} = AA^{k-1}.
\]

**Observation:**
\[
\begin{align*}
A^0 &= \{ \epsilon \} \\
A^* &= A^0 \cup A^1 \cup A^2 \cup \cdots \text{ (zero or more)} \\
A^+ &= A^1 \cup A^2 \cup \cdots \text{ (one or more)} \\
A^* &= A^+ \cup \{ \epsilon \}
\end{align*}
\]

**Quiz 1:** If \( A = \{ 1, 01 \} \), then \( A^2 = \)?

**Quiz 2:** If \( A = \{ 01, 11, 0 \} \), then \(|A^3| = \)?

**Claim:** If \(|A| = n\), then \(|A^k| = |A|^k = n^k\). True or false?

**Quiz 3:** \( A^+ = A^* - \{ \epsilon \} \)?
Why languages, not problems:

Meta Claim: Computational problem $\rightarrow$ decision problem.
Example: Traveling Saleman problem (TSP)
Input: $G = (V, E, w)$
Goal: Find a tour (cycle) with minimum total weight

Example: Decision problem for TSP
Input: $G = (V, E, w)$ and $B \geq 0$
Question: Is there a tour in $G$ such that the total weight of the tour is no more than $B$?

Decision problem: Given an input $x$, does $x$ satisfy property $P$, or is $x \in \{y | y$ satisfies $P\}$?
Input data of any form, such as matrix, graph, list, etc., can be coded into strings

Membership in a language: Given a language $A$ and a string $w \in \Sigma^*$, is $w$ a member of $A$, or is $w \in A$?
1.3 Proof techniques (Sipser 0.4)

▶ By definition: Convert terms in the hypothesis to their definitions.

▶ By construction: To prove the existence of $X$, construct it.

▶ By contradiction: $H \rightarrow C$ is equivalent to $\overline{C} \rightarrow \overline{H}$ or $\overline{C} \land H \rightarrow T$, where $T$ is an axiom, a proven truth/fact.

**Example**: The number of primes is infinite.

**Example**: $\sqrt{2}$ is irrational.

▶ By induction: Used to prove a statement $S(n)$, $\forall n \geq c$.

The logic behind the method:

$S(n)$ for $n \geq c$ iff $S(c) \land \forall k(S(k) \rightarrow S(k + 1))$.

The proof includes (1) basis step, (2) inductive hypothesis, and (3) inductive step.

**Example**: For $n \geq 1$, $\sum_{i=1}^{n} i^2 = \frac{1}{6} n(n + 1)(2n + 1)$. 
A Quick Review of Languages

- Alphabet $\Sigma$, string or word, substring, language
- Common operations borrowed from set theory: union, intersection, difference
- New operations: concatenation, complement, star, power
Finite automata are the simplest computational models for computers with an extremely limited amount of memory.

Use of automata theory in software applications includes: study of the behavior of digital circuits, lexical analyzer in compilers, text pattern matching, and verification of finite-state systems.

They are designed to accept some strings, therefore to recognize a language, which is the set of accepted strings.

Given an input string, a finite automaton reads the string, one symbol at a time, from left to right. Using a transition function, the FA changes from one state to another based on what symbol is being read. Eventually, it reads all symbols in the string. If at this time the FA enters a final state, then the FA halts and the string is accepted.
Example 1 of a DFA

- Alphabet: $\Sigma = \{0, 1\}$.

- Four states: $q_0, q_1, q_2, q_3$, in which $q_0$ is the start state and $q_3$ is a final state.

- Transition function $\delta$:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\rightarrow q_0$</td>
<td>$q_0$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_0$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_0$</td>
<td>$q_3$</td>
</tr>
<tr>
<td>$*q_3$</td>
<td>$q_3$</td>
<td>$q_3$</td>
</tr>
</tbody>
</table>

- What does $\delta$ tell us?

$\delta(q_0, 0) = q_0$, $\delta(q_0, 1) = q_1$
$\delta(q_1, 0) = q_0$, $\delta(q_1, 1) = q_2$
$\delta(q_2, 0) = q_0$, $\delta(q_2, 1) = q_3$
$\delta(q_3, 0) = q_3$, $\delta(q_3, 1) = q_3$
Example 1 (continued)

Figure 1: A DFA that accepts strings with substring 111

Try $w_1 = 100111010$ and $w_2 = 0011010001$

The language recognized/accepted is $L_1 = \{ w \in \{0, 1\}^* | w$ contains substring 111$\}$
2.2 DFA \textit{(Sipser 1.1, 35-44)}

- DFA $M = (Q, \Sigma, \delta, q_0, F)$, where
  - $Q$ is a finite set of states
  - $\Sigma$ is an alphabet
  - $q_0 \in Q$ is the start state
  - $F \subseteq Q$ is a set of accept/final states
  - $\delta : Q \times \Sigma \rightarrow Q$ is a transition function, where $\delta(q, a) = p$ is the next state of $M$ if the current state is $q$ and the current symbol is $a$

- Components of a DFA
  - A tape of squares, with the same length of the input string
  - A control unit that keeps track of the current state and follows the $\delta$ function
  - The head that reads and moves to the right, one square at a time
  - The DFA accepts the input string if the head reaches the end of the tape and the control unit sees the final state
Extending $\delta$ to $\hat{\delta}: Q \times \Sigma^* \rightarrow Q$: For any $q \in Q$ and any $w = xa \in \Sigma^*$, define $\hat{\delta}(q, x)$ recursively as below:

- $\hat{\delta}(q, \varepsilon) = q$ and
- $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$

Language recognized by DFA $M$: $L(M) = \{w | \hat{\delta}(q_0, w) \in F\}$.

A language is called a regular language if some DFA recognizes it.

How does a DFA accept a string? A path in the diagram that starts from $q_0$ and ends at $q_f \in F$ such that the concatenation of the symbols on the path matches the input string.
Example 2
Let $L_2 = \{ w \in \{0, 1\}^* \mid w$ has even numbers of 0s and 1s $\}$
Construct DFA $M$ such that $L(M) = L_2$.

Figure 2: A DFA that accepts strings of even number of 0s and 1s
Example 3 (Sipser p. 36): A DFA $M$ is given below:

$$
\begin{array}{c|cc}
\delta & 0 & 1 \\
\hline
q_0 & q_0 & q_1 \\
q_2 & q_1 & q_1 \\
\end{array}
$$

*Figure 3: Describe the language of the DFA*
Example 3 (continued)

The DFA in this example accepts strings that have at least one 1 and an even number of 0s after the last 1.

Reminder: More on DFA design in Sipser pp. 37-44.
Example 4 Design a DFA that accepts
\[ L_4 = \{ w \in \{0, 1\}^* | w \text{ contains substring 001} \} \]

**Figure 4:** A DFA that accepts strings with substring 001
Example 5 Give a DFA that accepts $L_5 = \{ w \in \{0, 1\}^* \mid \text{numerical value of } w \text{ is a multiple of } 3 \}$, e.g., for string 0110, its numerical value is 6, then it is in the language, but for 101 with a numerical value of 5, it is not a multiple of 3, thus is not in the language.

Figure 5: A DFA that accepts strings with numerical value that is a multiple of 3
Example 5 (continued)
Assume $x$ is portion of the input string being read so far.
Let $[x]$ be the numerical value of $x$, e.g., if $x = 0010$, then $[x] = 2$.
Entering state $q_i$ means that the string read so far has a numerical value that has a remainder of $i$ when divided by 3.

- Case 0: The current state is $q_0$, then $[x] = 3k$. If the next symbol is 0, then $[x0] = 2 \cdot 3k = 6k$, a multiple of 3. So the next state should be $q_0$. But if the next symbol is 1, then $[x1] = 2 \cdot 3k + 1 = 6k + 1$, with a remainder of 1 when divided by 3. So the next state should be $q_1$.

\[ \delta(q_0, 0) = q_0, \, \delta(q_0, 1) = q_1 \]

- Case 1: Similar to Case 0. $\delta(q_1, 0) = q_2, \, \delta(q_1, 1) = q_0$

- Case 2: Similar to above. $\delta(q_2, 0) = q_1, \, \delta(q_2, 1) = q_2$
Definitions of $\delta : Q \times \Sigma \rightarrow Q$ and $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$

The language of DFA $M$:
$L(M) = \{ w \in \{0,1\}^* : \hat{\delta}(q_0, w) \in F \}$

A language accepted by a DFA is a regular language.

In designing a DFA with $n$ nodes and alphabet $\Sigma$, the following two properties must be satisfied:

1. each node must have $|\Sigma|$ out-going arcs
2. the total number of arcs must be $n \cdot |\Sigma|$

A dead-end state may be added to a DFA to satisfy the above properties.

An example of including a dead-end state in a DFA that accepts strings that start with a 01:
2.3 NFA (Sipser 1.2, pp. 47-54)

NFA $N = (Q, \Sigma, \delta, q_0, F)$, where

- $Q$ is a finite set of states,
- $\Sigma$ is an alphabet,
- $q_0 \in Q$ is the start state,
- $F \subseteq Q$ is a set of accept/final states, and
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$ is a transition function, where $\delta(q, a) = P$ is the set of states that $N$ may enter if the current state is $q$ and the current symbol is $a$. In the case of $\delta(q, \varepsilon) = P$, $N$ ignores the current symbol and goes from $q$ to any state in $P$ without reading any symbol.
- \(\varepsilon\)-closure: For any \(P \subseteq Q\), \(E(P)\) is the set of all states reachable from any state in \(P\) via \(\geq 0\) \(\varepsilon\)-transitions.

- Extending \(\delta\) to \(\hat{\delta}: Q \times \Sigma^* \rightarrow 2^Q\): For any \(q \in Q\) and any \(w = xa \in \Sigma^*\), define
  - \(\hat{\delta}(q, \varepsilon) = E(\{q\})\) and
  - \(\hat{\delta}(q, w) = E(\bigcup_{i=1}^{k} \delta(p_i, a))\) if \(w = xa\) and \(\hat{\delta}(q, x) = \{p_1, \ldots, p_k\}\).

- Language of NFA \(N\): \(L(N) = \{w | \hat{\delta}(q_0, w) \cap F \neq \emptyset\}\).

- How does an NFA accept a string? Among all paths from \(q_0\) to \(q_f \in F\), there is some path such that the concatenation of symbols on the path matches the string.
Example 1: Language of strings that contain substring 111

Figure 6: An NFA that accepts strings with substring 111
Example 2: $A = \{w \in \{0, 1\}^* | w \text{ has 101 or 11 as substrings}\}$

![Diagram of NFA](image)

Figure 7: An NFA that accepts strings with substrings 101 or 11

Clarification:

- Consider string 001011. There are at least two paths from $q_0$ to $q_3$
- Examples of $\varepsilon$-closure (or E-closure): $E(\{q_0\}) = \{q_0\}$; $E(\{q_0, q_1\}) = \{q_0, q_1, q_2\}$
Example 3:

\[ B = \{ w \in \{0, 1\}^* | \text{w has a 1 in the 3rd position from the right end} \} \]

Figure 8: An NFA that accepts strings with a 1 in the third position from the right end
Example 4: An NFA that accepts decimal numbers (a number that may have + or − preceding it, but must have a decimal point, e.g., .123, 23., +1.2, -1.0).

Figure 9: An NFA that accepts strings that are decimal numbers with or without signs
2.4 DFAs $\Leftrightarrow$ NFAs (Sipser 1.2, pp.54-58)

Subset construction method: Given NFA $N = (Q, \Sigma, \delta, q_0, F)$, construct a DFA $M = (Q', \Sigma, \delta', q'_0, F')$ such that $L(M) = L(N)$.

$\blacktriangleright$ $Q' = 2^Q$ (power set), i.e., $Q'$ contains all subsets of $Q$. Note that if $|Q| = n$ then $|Q'| = 2^n$. This is just the worst case. Since many states in $M$ are inaccessible or dead-end states and thus may be thrown away, so in practice, $|Q'|$ may be much less than $2^n$.

$\blacktriangleright$ $q'_0 = E(\{q_0\})$.

$\blacktriangleright$ $F' = \{ R \in Q' | R \cap F \neq \emptyset \}$.

$\blacktriangleright$ For each $R \in Q'$ and each $a \in \Sigma$, $\delta'(R, a) = E(\bigcup_{p \in R} \delta(p, a))$.

**Definition**: Any language that can be accepted by DFA or NFA is called a regular language (RL).

**Theorem**: The equivalence of DFAs, NFAs, and RLs.
Converting an NFA to a DFA with subset construction

Example 1:

![NFA and DFA conversion diagram]

Figure 10: Converting an NFA to DFA with subset construction
Example 2:

Figure 11: Converting an NFA to a DFA
Example 3:

Figure 12: Converting an NFA to a DFA with subset construction
Example 3: (continued)
Without using the subset construction method, can a DFA be designed to accept all strings that has a 1 in the third position from the right end?

Figure 13: Design the same DFA from scratch
**Example 4**: A bad case for the subset construction: \( |Q_N| = n + 1 \) and \( |Q_D| = 2^n \).

![Diagram of states](image)

**Figure 14**: A case that converting an NFA to a DFA causes an exponential increasing of states in the DFA
2.5 Closure Properties of RL’s (*Sisper 1.1, pp. 44-47 and 1.2 pp. 58-63*)

- Union: If $A$ and $B$ are regular, so is $A \cup B$.
- Concatenation: If $A$ and $B$ are regular, so is $AB$.
- Star: If $A$ is regular, so is $A^*$. (Need a new start state.)
Complementation: If $A$ is regular, so is $\overline{A}$ (which is $\Sigma^* - A$).

Intersection: If $A$ and $B$ are regular, so is $A \cap B$ (which is $\overline{A \cup B}$).

Difference: If $A$ and $B$ are regular, so is $A - B$ (which is $A \cap \overline{B}$).

Reverse: If $A$ is regular, so is $A^R$.

Homomorphism: If $A$ is regular, so is $h(A)$ (which is $\{h(w)|w \in A\}$ for a homomorphism $h: \Sigma \rightarrow (\Sigma')^*$).

(Discuss later)

Inverse homomorphism: If $A$ is regular, so is $h^{-1}(A)$ (where $h^{-1}(A) = \{w|h(w) \in A\}$).
Example: Prove that $A = \{w \in \{a, b\}^* | w$ is of odd length and contains an even number of $a$’s$\}$ is regular.

Let $A_1 = \{w | w$ is of odd length$\}$ Let $A_2 = \{w | w$ has an even number of $a$’s$\}$

Since DFAs exist to accept $A_1$ and $A_2$, both are RLs

$A = A_1 \cap A_2$. By the CP of RLs under intersection, $A_1 \cap A_2$ is RL. So $A$ is RL

![DFA for A1](image1)

![DFA for A2](image2)

Figure 15: DFAs for $A_1$ and $A_2$
3.1 Definition of REs \((\textit{Sipser 1.3 (pp. 63-66)})\)

Regular expressions (REs) are to represent regular languages. Let \(L(R)\) be the language that regular expression \(R\) represents. A recursive definition is given below:

▶ **Basis**: \(\varepsilon\) and \(\emptyset\) are REs, and \(L(\varepsilon) = \{\varepsilon\}\) and \(L(\emptyset) = \emptyset\). For any \(a \in \Sigma\), \(a\) is an RE and \(L(a) = \{a\}\).

▶ **Induction**: If \(R_1\) and \(R_2\) are REs, then

▶ \(R_1 \cup R_2\) is an RE, with \(L(R_1 \cup R_2) = L(R_1) \cup L(R_2)\),

▶ \(R_1 R_2\) is an RE, with \(L(R_1 R_2) = L(R_1) L(R_2)\),

▶ \(R_1^*\) is an RE, with \(L(R_1^*) = (L(R_1))^*\), and

▶ \((R_1)\) is an RE, with \(L((R_1)) = L(R_1)\).
Remark:

- Precedence order for regular-expression operators: Star, concatenation, and finally union. () may override this order.
- Use of $R^+$ and $R^k$.
- Algebraic laws:
  - $R_1 \cup R_2 = R_2 \cup R_1$, $(R_1 \cup R_2) \cup R_3 = R_1 \cup (R_2 \cup R_3)$, and $(R_1 R_2) R_3 = R_1 (R_2 R_3)$.
  - $\emptyset \cup R = R \cup \emptyset = R$, $\varepsilon R = R \varepsilon = R$, $\emptyset R = R \emptyset = \emptyset$, and $R \cup R = R$.
  - $R_1 (R_2 \cup R_3) = R_1 R_2 \cup R_1 R_3$ and $(R_1 \cup R_2) R_3 = R_1 R_3 \cup R_2 R_3$.
  - $(R^*)^* = R^*$, $\emptyset^* = \varepsilon$, $R^+ = RR^* = R^* R$, and $R^* = R^+ \cup \varepsilon$. 
3.2 Understanding REs

- RE is a pattern for all strings in a RL. The goal is to make the RE as simple and readable as possible. Consider the following examples to simplify REs

- $1 \cup 10^* \Rightarrow 10^*$
- $(0^*1^*)^* \Rightarrow (0 \cup 1)^*$
- $((0 \cup 1)(0 \cup 1)^*)^* \Rightarrow (0 \cup 1)^*$
Given a language, design its RE

Example 1: \{w \text{ has no substring } 10\}: 0^*1^*
Example 2: \{w \text{ has even number of } 1\text{'s}\}: (0^*10^*10^*)^* \cup 0^*
Example 3: \{w \text{ has odd length}\}: ((0 \cup 1)(0 \cup 1))^*(0 \cup 1)
Example 4: \{w \text{ has a } 1 \text{ in 3rd or 2nd position from right end}\}:
\[(0 \cup 1)^*1(0 \cup 1)(0 \cup 1) \cup (0 \cup 1)^*1(0 \cup 1) \Rightarrow\]
\[(0 \cup 1)^*1(0 \cup 1)((0 \cup 1) \cup \epsilon) \Rightarrow\]
\[(0 \cup 1)^*1(0 \cup 1)(0 \cup 1 \cup \epsilon)\]
Example 5: A language of strings that consist of alternating 0s and 1s: (01)^* \cup (10)^* \cup 0(10)^* \cup 1(01)^*.
Example 7: \( D = \{ w \text{ has odd number of alternating blocks of 0s and 1's} \} \) (Note: \( \varepsilon \notin D \))

- \( 0|1|0 \in D, 0|1|0|1 \notin D, 11|00|111 \in D, 0|11|000|11|0|1 \notin D \)
- Observation 1: Odd number of blocks implies strings in \( D \) must start and end with the same symbol.
- RE for \( D \) based on ob.1: \( 0(0 \cup 1)^*0 \cup 1(0 \cup 1)^*1 \cup 0 \cup 1 \)
- Observation 2: Draw boundaries between blocks. Near each boundary, we see substring 01 or 10. A string in \( D \) must have equal number of substrings of 01 and 10.
- RE for \( D \) based on ob.2: \( 0^+(1^+0^+)^* \cup 1^+(0^+1^+)^* \)
**Example 8:** \( E = \{ w \mid \text{In } w, \text{ each } 1 \text{ is immediately preceded by a } 0 \text{ and followed by a } 0 \} \)

- Look for substring 010
- \(001000100010010 \in E\). Rewrite the string as
  \((001)(0001)(00001)(001)(0) \Rightarrow (0^+1)(0^+1)(0^+1)(0^+1)(0^+)\)
- \(RE: (0^+1)^*0^+ \cup \varepsilon \) or equally correct, \(0^+(10^+)^* \cup \varepsilon\)
Proof of Closure Properties of RLs
Union, concatenation and star:

- Union
- Concat
- Star

An example:

- $N^*$
- $N_1$
- $N_2$

$N_1$ and $N_2$ are not accepted by $N^*$.
**RE ⇒ NFA** *(Sipser 1.3 pp. 67-69)*

Since REs are defined recursively, it is suitable to construct the equivalent NFAs recursively.

- **Basis**: NFAs for simple RE: ε, ∅, and a for a ∈ Σ.
- **Induction**: Given the NFAs for REs $R_1$ and $R_2$, what are the NFAs for $R_1 \cup R_2$, $R_1 R_2$, and $R_1^*$?

**Example 1**: Converting RE $(0^*10^*10^*)^*$ to an NFA (which NFA is correct?)

![Diagram](image)

**Figure 16**: Converting a RE to an NFA
Example 2: Converting RE $(00^*1)^*(\epsilon \cup 10^*)$

Figure 17: Converting a RE to an NFA

Theorem: DFA, NFA, and RE are equivalent ways to accept or represent RLs. In particular, \( \text{RE} \Rightarrow \text{NFA}, \text{NFA} \Rightarrow \text{DFA}, \text{DFA} \Rightarrow \text{RE} \) (didn’t discuss in our class)
4.1 Regular versus nonregular languages

- $A = \{0^*1^*\}$
- $B = \{0^n1^n | n \geq 0\}$
- $C = \{w \in \{0, 1\}^* | w \text{ has an equal number of 0s and 1s}\}$
- $D = \{w \in \{0, 1\}^* | w \text{ has an equal } \# \text{ of substrings 01 and 10}\}$
Proving nonregularity by pumping lemma (Sisper 1.4 pp. 77-82)

**Theorem** (The pumping lemma for regular languages): For any regular language $A$, there exists a constant $p$ (whose value depends on $A$) such that $\forall s \in A$ with $|s| \geq p$, $s$ can be partitioned into three substrings $s = xyz$ such that

1. $|y| > 0$;
2. $|xy| \leq p$; and
3. $\forall i \geq 0$, string $xy^i z \in A$.

(Note: $xy^0 z = xz$, $xy^1 z = xyz$, $xy^2 z = xyyz$)
How to use the pumping lemma to prove that a language $A$ is not regular:

1. Assume that $A$ is regular by contradiction. (2) Then the pumping lemma applies to $A$. (3) Let $p$ be the constant in the pumping lemma. (three steps to start the proof)

- Select $s \in A$ with $|s| = f(p) \geq p$.
- By the pumping lemma, $\exists x, y, z$ such that $s = xyz$ with $|y| > 0$, $|xy| \leq p$ and $xy^i z \in A$ for any $i \geq 0$.
- For any $x, y, z$ such that $s = xyz$, $|y| > 0$, and $|xy| \leq p$, find $i \geq 0$ such that $xy^i z \not\in A$. A contradiction!
Example 1 (Sipser p. 80): Prove that $B = \{0^n1^n | n \geq 0\}$ is not regular.

- Assume $B$ is RL. Then PL applies to $B$. Let $p$ be the constant in PL.
- Select $s = 0^p1^p \in B$ with $|s| = 2p > p$
- By PL, $s = \underbrace{0 \ldots 0}_{p}1 \ldots 1 = xyz$, $|y| > 0$ and $|xy| \leq p$.
- Since $y \neq \epsilon$ and $|xy| \leq p$, then $y = 0^+$
- Choose $i = 0$. Then $xy^0z = xz = 0^{p'}1^p$ for $p' < p$
- So $xy^0z \notin B$. A contradiction to PL. So $B$ is non-R.
Example 2 (Sipser p. 81): Prove that \( F = \{ww \mid w \in \{0, 1\}^*\} \) is not regular.

- Assume \( F \) is RL. Then PL applies to \( F \). Let \( p \) be the constant in PL.
- Select \( s = 0^p10^p1 \in F \) with \( |s| = 2p + 2 > p \)
- By PL, \( s = \overbrace{0\ldots0}^p 1 \overbrace{0\ldots0}^p 1 = xyz, \ |y| > 0, \ |xy| \leq p \)
- Since \( y \neq \varepsilon \) and \( |xy| \leq p \), then \( y = 0^+ \)
- Choose \( i = 2 \). Then \( xy^2z = xyyz = 0^{p'}10^p1 \) for \( p' > p \)
- So \( xy^2z \notin F \). A contradiction to PL. So \( F \) is non-R.
Example 3: Prove that $A = \{1^r | r \text{ is a prime} \}$ is not regular.

- Some strings in $A$: 11, 111, 11111, 1111111, etc..
- Assume $A$ is RL. Then PL applies. Let $p$ be the constant.
- Select $s = 1^q$, where $q$ is a prime and $q \geq p$
- By PL, $s = \underbrace{1\ldots1}_q = xyz$, $x = 1^{h_1}$ ($h_1 \geq 0$), $y = 1^{h_2}$ ($h_2 > 0$),
  
  $z = 1^{q-h_1-h_2}$
- By PL, $\forall i \geq 0$, $xy^i z = 1^{h_1+i \cdot h_2+(q-h_1-h_2)} = 1^{(i-1)h_2+q} \in A$
- Choose $i = q+1$. Then $xy^i z = xy^{q+1} z = 1^{q(h_2+1)} \not\in A$
- A contradiction to PL. So $A$ is non-R.
Example 4: (Sipser p. 82) Prove that $D = \{1^{n^2} | n \geq 1\}$ is non-R.

- Some strings in $D$: 1, 1111, 111111111, etc.
- Assume $D$ is RL. Then PL applies. Let $p$ be the constant
- Select $s = 1^{p^2} \in D$, $|s| = p^2 \geq p$
- By PL, $s = 1^{p^2} = xyz$, $x = 1^{h_1}$ ($h_1 \geq 0$), $y = 1^{h_2}$ ($h_2 > 0$),
  $$z = 1^{p^2 - h_1 - h_2}, \text{ and } h_2 \leq |xy| \leq p$$
- By PL, $\forall i$, $xy^i z \in D$
- Choose $i = 2$. Consider $|xy^2 z|$
  $$= h_1 + 2h_2 + (p^2 - h_1 - h_2) = h_2 + p^2.$$ A perfect square?
- Some algebra:
  $$p^2 = 0 + p^2 < (h_2 + p^2) \leq p + p^2 < 1 + 2p + p^2 = (p + 1)^2$$
  $$p^2 < |xy^2 z| < (p + 1)^2.$$ So $|xy^2 z|$ is not a perfect square.
  Then $xy^2 z \notin D$. A contradiction to PL. So $D$ is non-R.
Example 5: Prove that \( A = \{10^n1^n \mid n \geq 0\} \) is not regular.

- Assume \( A \) is RL. The PL applies. Let \( p \) be the constant.
- Select \( s = 10^p1^p \in A. \ |s| = 2p + 1 > p \)
- By PL, \( s = 10\ldots01\ldots1 = xyz, \ |y| > 0, \ |xy| \leq p \)
- By PL, \( \forall i, \ xy^iz \in A. \) Consider two cases for \( y \).
  - Case 1. \( y \) contains the first 1: \( x = \varepsilon, \ y = 10^* \).
    Choose \( i = 0. \ xy^0z = xz = 0^p'1^p \not\in A, \) for \( p' \leq p \)
  - Case 2. \( y \) does not contain the first 1: \( x = 10^*, \ y = 0^+ \).
    Choose \( i = 0. \ xy^0z = xz = 10^p'1^p \not\in A, \) for \( p' < p \)
- For both cases, we have found contradiction to PL. So \( A \) is non-R.
Example 6: Prove that \( A = \{(01)^a0^b | a > b \geq 0 \} \) is not regular.

▶ Assume \( A \) is RL. The PL applies. Let \( p \) be the constant.

▶ Select \( s = (01)^p0^{p-1} \in A. \) \(|s| = 3p - 1\).

▶ By PL, \( s = 01\ldots010\ldots0 = xyz, \) \(|y| > 0, \) \(|xy| \leq p\)

▶ Consider \( y \) which is entirely in \((01)^p\).

▶ Case 1: Even length, i.e., \( y = 01\ldots01\) or \( y = 10\ldots10\).
  Choose \( i = 0 \) to remove at least a substring 01 or 10, violating \( a > b \)

▶ Case 2: Odd length, i.e., \( y = 01\ldots10\) or \( y = 10\ldots01\) or \( y = 0 \) or \( y = 1 \).
  Choose \( i = 2 \) to create substrings of 00 or 11, which is not allowed.

▶ In each case listed, we can find an \( i \) such that \( xy^iz \not\in A \). A contradiction to the PL. So \( A \) is non-R.
4.2 Prove nonregularity by closure properties
To prove that \( A \) is non-regular, assume it is regular. Find a regular language \( B \) and a language operator that preserves regularity, and then apply the operator on \( A \) and \( B \) to get a regular language \( C \). If \( C \) is known to be non-regular, a contradiction is found.

Example 1: Prove that
\[ C = \{ w \in \{0, 1\}^* | w \text{ has an equal } \# \text{ of 0s and 1s} \} \] is not regular.

- Assume \( C \) is regular
- Let \( B = \{0^*1^*\} \). \( B \) is known to be RL
- Let \( D = C \cap B = \{0^n1^n\} \). \( D \) is known to be non-R
- But \( D \) is regular by CP under intersection
- A contradiction! So \( C \) is non-R
Example 2: Prove that $A = \{0^m1^n \mid m \neq n\}$ is not regular.

- Assume $A$ is RL
- $\{0^*1^*\} = \{0^n1^n\} \cup A$ (two disjoint sets)
- $\{0^n1^n\} = \{0^*1^*\} - A$
- Since $\{0^*1^*\}$ and $A$ are both RL, the difference of the two RLs is still RL by CP.
- So $\{0^n1^n\}$ is RL. A contradiction.
- $A$ is non-R
Example 3: Prove that $A = \{a^mb^n c^{m+n} | m, n \geq 0\}$ is non-R

About homomorphism:

$h : \Sigma \rightarrow (\Sigma_1)^*$, e.g., $h(a) = 01$, $h(b) = 0$, $h(c) = \varepsilon$

$h : \Sigma^* \rightarrow (\Sigma_1)^*$, e.g., $h(ab) = 010$

$h : h(A) = \{h(w) | \forall w \in A\}$

- Assume $A$ is RL.
- Define homomorphism $h$ such that $h(a) = 0$, $h(b) = 0$, $h(c) = 1$

\[
h(A) = \{0^m0^n1^{m+n}\} = \{0^{m+n}1^{m+n}\} = \{0^n1^n\}\]

- By CP under homomorphism, since $A$ is RL, so is $\{0^n1^n\}$
- A contradiction. So $A$ is non-R
5.1 Context-free grammars *(Sipser 2.1 pp. 100-105)*

- CFG $G = (V, \Sigma, R, S)$, where $V$ is the set of variables, $\Sigma$ is the set of terminals (alphabet), $R$ is the set of rules in the form of $V \rightarrow (V \cup \Sigma)^*$ (head $\rightarrow$ body), and $S \in V$ is the start variable.

- The CFG that generates all palindromes (strings that read the same forward and backward) over $\{0, 1\}$ is $G = (\{S\}, \{0, 1\}, R, S)$, where $R$ contains $S \rightarrow 0S0|1S1|0|1|\varepsilon$.

- Any language that can be generated by a CFG is called context-free.
Let $u, v, w$ be strings in $(V \cup \Sigma)^*$. If $A \rightarrow w$ is a rule, then $uAv$ yields $uwv$, written $uAv \Rightarrow uv$. We say $u$ derives $v$, written $u \Rightarrow^* v$, if $\exists u_1, \ldots, u_k \in (V \cup \Sigma)^*$ such that $u \Rightarrow u_1 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v$. Here, $\Rightarrow$ means one step and $\Rightarrow^*$ means zero or more steps.

Leftmost and rightmost derivations: $\Rightarrow_{lm}$, $\Rightarrow_{lm}^*$, $\Rightarrow_{rm}$, $\Rightarrow_{rm}^*$.

The language of a CFG $G$, $L(G) = \{ w \in \Sigma^* | S \Rightarrow^* w \}$. $L(G)$ is said to be a CFL.
Some Simple CFGs and their CFLs

Example 1: \( L = \{0^n1^n | n \geq 0\} \) a context-free language. It can be generated by the following context-free grammar.

\[
S \rightarrow 0S1 | \varepsilon.
\]

Example 2: Given a CFG \( G \), describe \( L(G) \).

\[
S \rightarrow AA, \quad A \rightarrow AAA | bA | Ab | a
\]

Leftmost derivation: \( S \Rightarrow AA \Rightarrow bAA \Rightarrow baA \Rightarrow baa \)
Rightmost derivation: \( S \Rightarrow AA \Rightarrow Aa \Rightarrow bAa \Rightarrow baa \)

\( L(G) = \{w \in \{a, b\}^* | w \text{ has an even (nonzero) number of } a's\} \).

Example 3: A CFG for simple expressions in programming languages:

\[
S \rightarrow S + S | S * S | (S) | l
l \rightarrow la | lb | l0 | l1 | a | b
\]
5.2 Parse trees and ambiguity

Figure 18: Parse trees and ambiguity

(a) Parse tree and derivation $S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 0011$

(b) and (c) Two parse trees (or derivations) for string $a + b \ast a$. 

71
Parse trees

- A parse tree is a tree representation for a derivation, in which each interior node is a variable, each leaf node is either a terminal, or $\varepsilon$, and if an interior node is a variable $A$ and its children are $X_1, \ldots, X_k$, then there must be a rule $A \rightarrow X_1 \cdots X_k$.

- Yield of a parse tree: Concatenation of the leaf nodes in a parse tree rooted at the start variable.

- Four equivalent notions:
  1. $S \Rightarrow^* w$;
  2. $S \overset{lm}{\Rightarrow^*} w$;
  3. $S \overset{rm}{\Rightarrow^*} w$; and
  4. A parse tree with root $S$ and yield $w$. 

72
Ambiguity in grammars and languages (Sipser 2.1 pp. 105-106)

▶ A CFG $G = (V, \Sigma, R, S)$ is ambiguous if there is $w \in \Sigma^*$ for which there are at least two parse trees (or leftmost derivations).

▶ Grammar $G$: $S \rightarrow S + S | S \cdot S | (S) | I$ and $I \rightarrow la|lb|l0|l1|a|b$ is ambiguous since $a + b \cdot a$ has two parse trees.

▶ Some ambiguous grammars have an equivalent unambiguous grammar. For example, an unambiguous grammar for the simple expressions is $G': S \rightarrow S + T | T$, $T \rightarrow T \cdot F | F$, $F \rightarrow (S) | I$, and $I \rightarrow la|lb|l0|l1|a|b$. 
A context-free language is said to be inherently ambiguous if all its grammars are ambiguous.

There is no algorithm to determine whether a given CFG is ambiguous. There is no algorithm to remove ambiguity from an ambiguous CFG. There is no algorithm to determine whether a given CFL is inherently ambiguous.
Chomsky normal form (Sipser 2.1 pp. 106-109)

The Chomsky Normal Form (CNF): Any nonempty CFL without $\varepsilon$ has a CFG $G$ in which all rules are in one of the following two forms: $A \rightarrow BC$ and $A \rightarrow a$, where $A, B, C$ are variables, and $a$ is a terminal. Note that one of the uses of CNF is to turn parse trees into binary trees.
5.3 More CFGs design

Example 1: \( \{a^m b^n c^{m+n} | m, n \geq 0\} \)
- Rewrite the pattern as \( a^m b^n c^n c^m \)
- \( S \to aSc | T, T \to bTc | \varepsilon \)

Example 2: \( \{a^m b^m c^n d^n | m, n \geq 0\} \cup \{a^m b^n c^n d^m | m, n \geq 0\} \)
- \( S \to S_1 | S_2 \)
- \( S_1 \to AB, A \to aAb | \varepsilon, B \to cBd | \varepsilon \)
- \( S_2 \to aS_2 d | C, C \to bCc | \varepsilon \)
**Example 3:** \( \{0^m 1^n | m \neq n\} \)

- Rewrite the language as \( \{0^m 1^n | m < n\} \cup \{0^m 1^n | m > n\} \), which is \( \{0^m 1^{n-m} 1^m\} \cup \{0^n 0^{m-n} 1^n\} \)
- \( S \rightarrow S_1 | S_2 \)
- \( S_1 \rightarrow 0S_1 1 | A, \ A \rightarrow 1A | 1 \)
- \( S_2 \rightarrow 0S_2 1 | B, \ B \rightarrow 0B | 0 \)
**Example 5:** $L = \{ a^i b^j c^k \in \{ a, b, c \}^* \mid i + j \neq k \}$

The grammar will pair $a$ and $c$ until running out one of the two. Then the grammar will consider the following cases.

$L = L_1 \cup L_2 \cup L_3$

- **Case 1:** $i = k$. Then $j \neq 0$, i.e., $j \geq 1$. So $L_1 = \{ a^i b^+ c^i \}$
- **Case 2:** $i > k$. Then $j \geq 0$. So $L_2 = \{ a^k a^+ b^* c^k \}$
- **Case 3:** $i < k$. And $j \neq k - i$. So $L_3 = \{ a^i b^j c^{k-i} c^i \}$

$S \to aSc \mid S_1 \mid S_2 \mid S_3$

$S_1 \to bS_1 \mid b$ (Case 1: To generate $b^+$)

$S_2 \to aS_2 \mid S_2 b \mid a$ (Case 2: To generate $a^+ b^*$)

$S_3 \to bS_3 c \mid S_1 \mid C, C \to cC \mid c$ (Case 3: To generate $b^i$ and $c^{k-i}$ s.t. $j \neq k - i$)
Example 6: \( L = \{a^i b^j \mid i \neq j \text{ and } 2i \neq j\} \).

Draw an x-axis and mark two points, one for \( i \) and one for \( 2i \).
These two points divides the x-axis into three intervals: \( j < i \), \( i < j < 2i \), and \( j > 2i \).

\[
\begin{array}{c c c}
L1 & L2 & L3 \\
\hline
j<i & i<j<2i & j>2i \\
\end{array}
\]

Figure 19: Intervals that \( j \) falls in

\( L = L_1 \cup L_2 \cup L_3: S \rightarrow S_1 \mid S_2 \mid S_3 \)

\( L_1 = \{a^i b^j \mid j < i\}: S_1 \rightarrow aS_1 b \mid A, A \rightarrow aA \mid a \)

\( L_2 = \{a^i b^j \mid i < j < 2i\}: S_2 \rightarrow aS_2 b \mid aTb, T \rightarrow aTbb \mid abb \)

\( L_3 = \{a^i b^j \mid j > 2i\}: S_3 \rightarrow aS_3 bb \mid B, B \rightarrow Bb \mid b \)
Example 7: $L = \{ x\#y \mid x, y \in \{0, 1\}^*, |x| \neq |y| \}$

Consider two cases: $|x| < |y|$ and $|x| > |y|$.

$S \rightarrow 0S0 \mid 0S1 \mid 1S0 \mid 1S1 \mid T\# \mid \#T$

$T \rightarrow 0T \mid 1T \mid 0 \mid 1$
6.1 PDAs (*Sipser 2.2 pp. 102-114*)

- PDA = NFA + Stack (still with limited memory but more than that in FAs)
- PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where
  - $Q$: A finite set of states
  - $\Sigma$: A finite set of input symbols (input alphabet)
  - $\Gamma$: A finite set of stack symbols (stack alphabet)
  - $\delta$: The transition function from $Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\})$ to $2^{Q \times (\Gamma \cup \{\epsilon\})}$
  - $q_0$: The start state
  - $F$: The set of final states
What does $\delta(q, a, X) = \{(p, Y)\}$ mean? If the current state is $q$, the current input symbol is $a$, and the stack symbol at the top of the stack is $X$, then the automaton changes to state $p$ and replace $X$ by $Y$.

What if $\epsilon$ replaces $a$, or $X$, or $Y$? For example,

$\delta(q, \epsilon, X) = \{(p, Y)\}$: No cursor move. $X$ replaced by $Y$ (pop + push).
$\delta(q, a, \epsilon) = \{(p, Y)\}$: Push $Y$
$\delta(q, a, X) = \{(p, \epsilon)\}$: Pop $X$
$\delta(q, \epsilon, \epsilon) = \{(p, Y)\}$: No cursor move. Push $Y$
$\delta(q, a, \epsilon) = \{(p, \epsilon)\}$: No stack change
$\delta(q, \epsilon, X) = \{(p, \epsilon)\}$: No cursor move. Pop $X$
$\delta(q, \epsilon, \epsilon) = \{(p, \epsilon)\}$: No change except state

The state diagram of PDAs: For transition $\delta(q, a, X) = \{(p, Y)\}$, draw an arc from state $q$ to state $p$ labeled with $a, X \rightarrow Y$. 

82
Instantaneous description (ID) of a PDA: \((q, w, \gamma)\) represents the configuration of a PDA in the state of \(q\) with the remaining input of \(w\) yet to be read and the stack content of \(\gamma\). (The convention is that the leftmost symbol in \(\gamma\) is at the top of the stack.)

Binary relation \(\vdash\) on ID’s: \((q, aw, X\beta) \vdash (p, w, Y\beta)\) if \(\delta(q, a, X)\) contains \((p, Y)\). \(\vdash\) represents one move of the PDA, and \(\vdash^*\) represents zero or more moves of the PDA.

Language of a PDA \(M\) (or language recognized by \(M\)) is \(L(M) = \{ w | (q_0, w, \varepsilon) \vdash^* (f, \varepsilon, \gamma) \text{ for } f \in F \}\).

How does a PDA check the stack is empty? At the beginning of any computation, it pushes a special symbol \($\) to the initially empty stack by having transition \(\delta(q_0, \varepsilon, \varepsilon) = \{(q, $)\}.\)
Example 1 (Sipser p. 112): A PDA that recognizes \( \{0^n1^n | n \geq 0 \} \).

Figure 20: An example of a PDA
Example 2 (Sipser p. 114): A PDA that recognizes 
\[ \{ a^i b^j c^k \mid i, j, k \geq 0, i = j \text{ or } i = k \} = \{ a^n b^n c^* \} \cup \{ a^n b^* c^n \} \].

Figure 21: Another PDA

Example: How the PDA in the above example accepts input \textit{aabbc}.  

Theorem: The equivalence of PDA, CFG, and CFL
7.1 Proving non-CFLs by pumping lemma (Sipser 2.3 (pp. 125-129))

**Theorem 2.34** (The pumping lemma for CFLs)
Let $A$ be a CFL. Then there exists a constant $p$ such that $\forall s \in A$ with $|s| \geq p$, we can write $s = uvxyz$ such that

1. $|vy| > 0$; (not allow $v = y = \varepsilon$)
2. $|vxy| \leq p$; and
3. $\forall i \geq 0$, string $uv^ixy^iz \in A$.

Recall in the PL for RLs, $s$ is partitioned into $x, y, z$ satisfying

1. $|y| > 0$;
2. $|xy| \leq p$; and
3. $\forall i \geq 0$, string $xy^iz \in A$
How to use the pumping lemma to prove that a language is not Context-free?

▶ Assume that $A$ is context-free by contradiction. Then the pumping lemma applies to $A$. Let $p$ be the constant in the pumping lemma. (Always begin the proof with these three steps.)

▶ Select $s \in A$ with $|s| = f(p) \geq p$. (This may be tricky. But start with an intuitive simple string that uses $p$ in the length.)

▶ By the pumping lemma, $s = uvxyz$ with (1)$|vy| > 0$, (2)$|vxy| \leq p$, and (3)$uv^i xy^i z \in A, \forall i \geq 0$.

▶ Prove for ANY $u, v, x, y, z$ such that $s = uvxyz$, $|vy| > 0$, and $|vxy| \leq p$, find $i \geq 0$ such that $uv^i xy^i z \notin A$. A contradiction to (3) in PL!
Example 2.36 (Sipser p.128): Prove \( B = \{ a^n b^n c^n | n \geq 0 \} \) is non-CF.

Pf. Assume B is CF. Then PL applies. Let \( p \) be the constant.

- Select \( s = a^p b^p c^p = a \cdots b \cdots c \cdots \in B \), with \( |s| = 3p > p \).
- By PL, \( s = uvxyz \) with \( |vy| > 0 \) and \( |vxy| \leq p \).
- Consider what \( vxy \) can be. Imagine \( vxy \) is a sliding window that moves left to right within \( s \).
- Case 1: \( vxy \) contains one symbol type (\( a^+, b^+, \) or \( c^+ \))
- Case 2: \( vxy \) contains two symbol types (\( a^+ b^+ \) or \( b^+ c^+ \))
- Choose \( i = 0 \) for both cases. Then \( uv^0 xy^0 v \not\in B \) for both.
- A contradiction to the PL!
Example 2.38 (Sipser p. 129): Prove $D = \{ww | w \in \{0, 1\}^*\}$ is non-CF.

Pf. Three sentences to start the proof. Then,

- Select $s = 0^p 10^p 1 = 0 \cdots 010 \cdots 01 \in D$, $|s| = 2p + 2 > p$.
- By PL, $s = uvxyz$ with $|vy| > 0$ and $|vxy| \leq p$
- We consider the following partition:
  $s = 0^{p-1} \cdot 0 \cdot 1 \cdot 0 \cdot 0^{p-1} 1$, where $u = 0^{p-1}, v = 0, x = 1, y = 0$, and $z = 0^{p-1} 1$.
- For any $i \geq 0$, $uv^i xy^i z = 0^{p-1} \cdot 0^i \cdot 1 \cdot 0^i \cdot 0^{p-1} 1 = 0^{p-1+i} \cdot 1 \cdot 0^{i+p-1} \cdot 1 \in D$
- No contradiction. Need to choose a different $s$.
- Try $s = 0^p 1^p 0^p 1^p$ and $i = 0$
  (Exercise or read p. 129 bottom)
Example 3: Prove that $A = \{0^j1^j^2\}$ is non-CF.

- Select $s = 0^p1^p^2 = uvxyz \in A$, where $|vy| > 0$ and $|vxy| \leq p$.
- $vxy = 0^+, 1^+, \text{or } 0^+1^+$ (three cases)
- For the first two cases, choose $i = 0$ to shrink the 0 and 1 blocks, respectively, thus making the string $uxz \notin A$
- Case 3: $vxy = 0^+1^+$.
  - Case 3.1: $v$ or $y$ contains both 0 and 1, i.e., $v$ or $y = 0^+1^+$. Let $i = 2$. Then $uv^2xy^2z$ contains substring $0^+1^+0^+1^+$, thus not in $A$.
  - Case 3.2: $v$ and $y$ do not contain both 0 and 1, which means that $v$ and $y$ each contain at most one symbol type, i.e. $v = 0^{|v|}$ and $y = 1^{|y|}$. (Note: $v$ or $y$ may be $\varepsilon$ but not both)
Continue with Case 3.2

- Let $i = 2$. Then $uv^2 xy^2 z = 0^{p+|v|} . 1^{p^2+|y|}$
- Is $(p + |v|)^2 = (p^2 + |y|)$?
- Is $p^2 + 2p|v| + |v|^2 = p^2 + |y|$?
- Is $2p|v| + |v|^2 = |y|$? Or Left = Right?
- If $|y| = 0$ and $|v| \neq 0$: Left $>$ Right
- If $|y| \neq 0$ and $|v| = 0$: Left $<$ Right
- If $|y| \neq 0$ and $|v| \neq 0$:
  
  Left $= 2p|v| + |v|^2 > p \geq |vxy| \geq |y| = $ Right
  
  So Left $>$ Right

- So for all combinations of $|v|$ and $|y|$, Left $\neq$ Right. So $uv^2 xy^2 z \notin A$

- A contradiction!
Example 4: Prove that $L = \{a^ib^jc^id^j \mid i, j \geq 0\}$ is non-CF.

Assume $L$ is CF. Then the PL applies to $L$. Let $p$ be the constant.

Select $s = a^pb^pc^pd^p$. $s \in L$ and $|s| = 4p > p$.

By PL, $s = uvxyz$ with $|vy| > 0$ and $|vxy| \leq p$.

Since $|vxy| \leq p$, $v$ and $y$ cannot contain both $a$’s and $c$’s, nor can it contain both $b$’s and $d$’s. Further $|vy| > 0$. We have $uv^0xy^0z = uxz \notin L$, because it either contains fewer $a$’s than $c$’s, or fewer $c$’s then $a$’s, or fewer $b$’s than $d$’s, or fewer $d$’s than $b$’s.

A contradiction to the PL.

So $L$ is non-CF.
7.2 Proving non-CFLs by closure properties

- Closed under union: If $A$ and $B$ are CF, so is $A \cup B$.
  
  Proof: CFG $G_A$ with $S_A \to \cdots$ and CFG $G_B$ with $S_B \to \cdots$.
  Define a CFG that generates $A \cup B$ as $S \to S_A \mid S_B$ plus the grammars $G_A$ and $G_B$.

- Closed under concatenation: If $A$ and $B$ are context-free, so is $AB$.
  
  Proof: CFG $G_A$ with $S_A \to \cdots$ and CFG $G_B$ with $S_B \to \cdots$.
  Define a CFG that generates $AB$ as $S \to S_A S_B$ plus the grammars $G_A$ and $G_B$.

- Closed under star: If $A$ is context-free, so is $A^*$.
  
  Proof: Consider CFG $G$ with $S_1 \to \cdots$. Define a CFG that generates $A^*$ as $S \to SS_1 \mid \varepsilon$ plus the grammar $G$.  

93
Closed under reverse: If $A$ is context-free, so is $A^R$.

Not closed under intersection: Consider $A = \{a^n b^n c^m\}$ and $B = \{a^m b^n c^n\}$.

Not closed under complementation: Note that $A \cap B = \overline{A} \cup \overline{B}$.

Not closed under difference: Note that $\overline{A} = \Sigma^* - A$. 
- Intersect with a regular language: If $A$ is context-free and $B$ is regular, then $A \cap B$ is context-free.
- Difference from a regular language: If $A$ is context-free and $B$ is regular, then $A - B$ is context-free. Note that $A - B = A \cap \overline{B}$. 
How can closure properties of CFLs be used to prove that a given language is non-CF?

**Example 1:** $A = \{ w \in \{a, b, c\}^* | n_a(w) = n_b(w) = n_c(w) \}$ is non-CF. (Note: $n_a(w)$ is defined to be the number of $a$'s in $w$.)

- Assume $A$ is CF. Let $B = \{ a^n b^n c^n \}$, a RL.
- $A \cap B = \{ a^n b^n c^n \}$ must be CF by the closure property of the intersection of a CFL and a RL.
- But we proved before that $\{ a^n b^n c^n \}$ is non-CF.
- A contradiction.
- So $A$ must be non-CF.
How can closure properties of CFLs be used to prove that a given language is CF?

**Example 2:** \( B = \{ a^i b^j c^k \mid j > i + k \} \) is CF.

- Since \( j > i + k \), let \( j = i + k + h \) for some \( h > 0 \).
- \( a^i b^j c^k = a^i \cdot b^{i+k+h} \cdot c^k = a^i \cdot b^{i+h} \cdot b^k \cdot c^k = (a^i b^i)(b^+)(b^k c^k) \)
- This is the concatenation of three CFLs.
- By the closure property of CFLs under concatenation, \( B \) is CF.
About the midterm:
True/False. Cheat sheet only. 60min + 30min. 25%. Available Friday afternoon to Monday evening. Sample questions below.

- **DFA/NFA/RL:**
  - A DFA can have more than one final state.
  - The union of any infinite number of regular languages is regular.
  - Any superset (subset) of a regular language is regular.

- **RE:**
  - \((0^*1^*)^* = (0 \cup 1)^*\)
  - \((R_1 \cup R_2)^* R_2 = (R_1^* R_2)^*\)
  - \(L((0101 \cup 01 \cup \varepsilon)^*) \subset L((01)^* \cup \varepsilon)\)

- **CFG/CFL:**
  - Any regular language is context-free.
  - The language generated by CFG \(G\), where \(S \rightarrow aaSb | aSb | a | \varepsilon\), is ambiguous.
  - Any superset (subset) of a CFL is CF (non-CF)
Computability Theory: An introduction

- A study of capability and limitation of computers, or equivalently, what they can do and what they cannot.
- Given a problem, can it be solved at all?
- The set of all problems can be divided into two subsets. One subset contains all solvable problems, such as sorting, finding shortest path in a graph. The other subset contains those unsolvable problems.
- Computability Theory is to study techniques to prove if a given problem is solvable or unsolvable.

![Figure 22: Solvable vs. unsolvable](image)

99
8.1 Unsolvable problems

- A problem is said to be unsolvable/undecidable if it cannot be solved/decided by any algorithm.
- Most interesting problems are optimization problems (OPT)
- Decision problems (DEC) ask a yes-no question.
- Example: The Traveling Salesman Problem (TSPOPT)
  Visit every city and go back home.
  Input: An edge-weighted graph $G = (V, E, w)$
  Output: A tour (simple cycle of all vertices) with min total weight

- Corresponding decision problem (TSPDEC)
  Input: $G = (V, E, w)$ and $B \geq 0$
  Question: Is there a tour in $G$ with total weight $\leq B$?
Meta Claim: DEC is no harder than its corresponding OPT

So, to study hardness of an OPT, we focus on its DEC.

Any DEC is actually a language since the yes-no question in DEC can be interpreted as asking membership of a string in a language.

Example: Prime (DEC)
Input: An integer $x \geq 2$
Question: Is $x$ a prime? (This is a yes/no question.)

$L_{\text{prime}} = \langle x \rangle | x \text{ is prime}$ (This is a language)
$L_{\text{prime}}$ is actually the language of all prime numbers encoded in binary representation.
Encoding anything to a binary string:

- Integer $x$ to binary string $< x >$
- Graph $G$ to $< G >$
- Matrix $M$ to $< M >$
- List $L$ to $< L >$

Revisit TSPDEC and its corresponding language:

- Input (or Instance): $G$ and $B \geq 0$
  Question: Does $G$ contain a tour with the total weight $\leq B$?
- $L_{TSPDEC} = \{ < G, B > | \text{There is a tour with total weight } \leq B \}$
  Is string $< G, B >$ a member of language $L_{TSPDEC}$?
The number of languages over a non-unary alphabet is uncountably infinite. So is the number of DECs (or decision problems).

However, the number of programs that a computer can use to solve problems is countably infinite. Therefore, there are more problems than there are programs. Thus, there must be some unsolvable problems.
An unsolvable (or undecidable) problem:
The famous Halting Problem (by Turing):
  - Input: Any Turing Machine $M$ and any string $s$
  - Question: Does $M$ halt on $s$?

The modern version:
  - Input: Any program $P$ and any input $I$
  - Output: “Yes” if $P$ terminates on $I$ and “No” otherwise.
  - Or Question: Does $P$ terminate on $I$?

![Diagram]

Figure 23: Does $P$ terminate/halt on $I$?
8.2 Turing machine (Sipser 3.1, pp. 165-175)

- A Turing machine includes a control unit, a read-write head, and a one-way infinite tape.

![Figure 24: Picture of a Turing Machine](image-url)

Tape and tape squares | Infinitely long
---|---
0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | B | B

δ(q, 1) = (p, X, L)
TM \( M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}) \), where

- **Q**: The finite set of states for the control unit.
- **\( \Sigma \)**: An alphabet of input symbols, not containing the “blank symbol”, \( B \).
- **\( \Gamma \)**: The complete set of tape symbols. \( \Sigma \cup \{B\} \subset \Gamma \).
- **\( \delta \)**: The transition function from \( Q \times \Gamma \) to \( Q \times \Gamma \times D \), where \( D = \{L, R\} \).
- For example, \( \delta(q, 0) = (p, X, L) \) and \( \delta(p, Y) = (q, B, R) \).
- **\( q_0 \)**: The start state.
- **\( q_{\text{accept}} \)**: The accept state.
- **\( q_{\text{reject}} \)**: The reject state.
Configuration: Use a string to describe the look of a TM at a certain time, instead of drawing a picture of the TM. For example, string $X_1 \cdots X_{i-1} q X_i \cdots X_n$ gives a description (snapshot) of the TM at a time, when the current state is $q$, the tape content is $X_1 \cdots X_n$, and the head is scanning (pointing to) $X_i$. Such a string is called the configuration of the TM at a certain time.
How a TM changes its configurations:

- If $\delta(q, X_i) = (p, Y, L)$, then
  $$X_1 \cdots X_{i-1} q X_i \cdots X_n \vdash X_1 \cdots X_{i-2} p X_{i-1} Y X_{i+1} \cdots X_n.$$  

- If $\delta(q, X_i) = (p, Y, R)$, then
  $$X_1 \cdots X_{i-1} q X_i \cdots X_n \vdash X_1 \cdots X_{i-1} Y p X_{i+1} \cdots X_n.$$  

Figure 25: Transitions applied on configurations
Three important configurations:
(1) Starting configuration $q_0w$,
(2) accepting configuration $uq_{accept}v$,
(3) rejecting configuration $uq_{reject}v$,
where (2) and (3) are called the halting configurations.

Language of a Turing machine $M$ (or language recognized/accepted by $M$) is
$L(M) = \{ w \in \Sigma^* | q_0w \vdash_\ast \alpha q_{accept} \beta \text{ for any } \alpha, \beta \in \Gamma^* \}.$

Note: To produce $\vdash$, type ”backslash vdash” in the math mode.

For any given input, a TM has three possible outcomes: accept, reject, and loop. Accept and reject mean that the TM halts on the given input, but loop means that the TM does not halt on the input.
TRL: A language \( A \) is Turing-recognizable if there is a TM \( M \) such that \( A = L(M) \). In other words,

- \( \forall w \in A, M \) accepts \( w \) by entering \( q_{\text{accept}} \).
- \( \forall w \notin A, M \) does not accept (i.e., it may reject or loop).

TDL: A language \( A \) is Turing-decidable if there is a TM \( M \) such that \( A = L(M) \) and \( M \) halts on all inputs. In other words,

- \( \forall w \in A, M \) accepts \( w \).
- \( \forall w \notin A, M \) rejects \( w \).

Such TMs are a good model for algorithms.

**Figure 26: TRLs vs. TDLs**

110
How to design a TM that recognizes/accepts a language?

Example 1: Give a implementation-level description of a TM $M$ that accepts $\{0^n1^n | n \geq 0\}$, i.e., $L(M) = \{0^n1^n | n \geq 0\}$

Idea: $w = 000111 \Rightarrow X00Y11 \Rightarrow XX0YY1 \Rightarrow XXXYYYY$

$M =$ "On input string $w = 0^n1^n$

1. If $w = \varepsilon$, accept
2. Mark the first 0 with $X$, move right to mark the first 1 with $Y$
3. Move left to find the leftmost 0. If no 0, accept, else go to stage 2"
Example 1: (more) Define a TM that accepts \( \{0^n1^n \mid n \geq 0 \} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( 0 )</th>
<th>( 1 )</th>
<th>( X )</th>
<th>( Y )</th>
<th>( B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>((q_1, X, R)^1)</td>
<td>-</td>
<td>-</td>
<td>((q_3, Y, R)^8)</td>
<td>((q_a, B, R)^0)</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>((q_1, 0, R)^2)</td>
<td>((q_2, Y, L)^4)</td>
<td>-</td>
<td>((q_1, Y, R)^3)</td>
<td>-</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>((q_2, 0, L)^6)</td>
<td>-</td>
<td>((q_0, X, R)^7)</td>
<td>((q_2, Y, L)^5)</td>
<td>-</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>((q_3, Y, R)^9)</td>
<td>((q_a, B, R)^{10})</td>
</tr>
<tr>
<td>( q_a )</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 27: Transition diagram for TM
Example 2: Give an implementation-level description of a TM $M$ that decides $\{0^n1^n | n \geq 0\}$.

$M =$ ”On any input string $w \in \{0, 1\}^*$

1. If $w \neq 0^*1^*$, reject

2. Sweep left to right. If no 0 and 1 are found, accept. If only 0 is found or 1 is found, but not both, reject. If both 0 and 1 are found, go to stage 3

3. Mark the leftmost 0 with X. Move head to right to find and mark the first 1 with Y

4. Move head to left end, and then go to stage 2”
Example 3.7 (Sipser p. 171): Give a TM $M$ that decides $A = \{0^{2^n} | n \geq 0\} = \{0, 00, 0000, 00000000, \cdots \}$.

Consider the following strings to figure out an algorithm (TM): 
(1) odd length, e.g., $w_1 = 00000 \Rightarrow 0X0X0$; 
(2) even length, e.g., 
$w_2 = 00000000 \Rightarrow 0X0X0X0X \Rightarrow 0XXX0XXX \Rightarrow 0XXXXXXX$; 
$w_3 = 000000 \Rightarrow 0X0X0X$

TM $M = $"On input string $w \in \{0\}^*$:
1. Sweep left to right, crossing off every other 0
2. If in stage 1 the tape contained a single 0, accept (e.g., $w_2$)
3. If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, reject (e.g., $w_1$ and $w_3$)
4. Move head to the left end of the tape
5. Go to stage 1"
8.4 Variations of TMs (*Sipser 3.2 (pp. 148-159)*)

- TM with multi-tapes (and multi-heads) 
  \( (\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R\}^k) \).
- TM with multi-strings (and multi-heads).
- TM with multi-heads.
- TM with multi-tracks.
- TM with two-way infinite tape.
- TM with multi-dimensional tape.
- Nondeterministic TM’s \( (\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times D}) \).

Consider a move in NTM, \( \delta(q_3, X) = \{(q_5, Y, R), (q_3, X, L)\} \).

How does the NTM know which step it should take? One way to look at this is: the NTM is the "luckiest possible guesser" and it always picks a transition that eventually leads to an accepting state if there is such a transition.
**Theorem:** The equivalent computing power of the above TM’s:

For any language $L$, if $L = L(M_1)$ for some TM $M_1$ with multi-tapes, multi-strings, multi-heads, multi-tracks, two-way infinite tape, multi-dimensional tape, or nondeterminism, then $L = L(M_2)$ for some basic TM $M_2$. 
**Theorem**: The equivalent computing speed of the above TM’s except for nondeterministic TM’s:

For any language $L$, if $L = L(M_1)$ for some TM $M_1$ with multi-tapes, multi-strings, multi-heads, multi-tracks, two-way infinite tape, or multi-dimensional tape in a polynomial number of steps, then $L = L(M_2)$ for some basic TM $M_2$ in a polynomial number of steps (with a higher degree).

Or in other words, all reasonable models of computation can simulate each other with only a polynomial loss of efficiency.

Note: The speed-up of a nondeterministic TM vs. a basic TM is exponential.
The Church-Turing Thesis:

Any reasonable attempt to model mathematically algorithms and their time performance is bound to end up with a model of computation and associated time cost that is equivalent to Turing machines within a polynomial. (The power of TM.)
Nondeterministic TMs

- \( \delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times D} \).

- Consider a move in NTM, \( \delta(q_3, X) = \{(q_5, Y, R), (q_3, X, L)\} \). How does the NTM know which step it should take?

- One way to look at this is that the NTM is the "luckiest possible guesser" and it always picks a transition that eventually leads to an accepting state if there is such a transition.

- The other way is to imagine that the NTM branches into many copies, each of which follows one of the possible transition.

- DTM (path) versus NTM (tree): See the wiki page for "Nondeterministic Turing Machine".
**Theorem:** A TDL is also a TRL, but not vice versa.

**Theorem:** About $A$ and $\overline{A}$:

1. If $A$ is Turing-decidable, so is $\overline{A}$.
2. If $A$ and $\overline{A}$ are both Turing-recognizable, then $A$ is Turing-decidable. (See Theorem 4.22, p.210)
3. For any $A$ and $\overline{A}$, we have one of the following possibilities:
   (1) Both are Turing-decidable;
   (2) Neither is Turing-recognizable;
   (3) One is Turing-recognizable but not decidable, the other is not Turing-recognizable.

![Figure 28: A language and its complement](image-url)
Some closure properties:
TRLs and TDLs are both closed under

- Union
- Intersection
- Concatenation
- Star

In addition, TDLs are closed under complement, and TRLs are closed under homomorphism.
Examples to prove closure properties:

Example 1: If $L_1$ and $L_2$ are TD, so is $L_1 \cup L_2$.

Pf: Let TM $M_1$ and TM $M_2$ decide $L_1$ and $L_2$, respectively. Then we have the following TM $M$ to decide $L_1 \cup L_2$.

TM $M =$ "On input $w$:

1. Run $M_1$ on $w$
2. If $M_1$ accepts, accept
3. else run $M_2$ on $w$
4. If $M_2$ accepts, accept
5. else reject"
**Example 2:** If $L_1$ and $L_2$ are TR, so is $L_1 \cup L_2$.

**Pf:** Let TM $M_1$ and TM $M_2$ recognize $L_1$ and $L_2$, respectively. Then we have the following TM $M$ to recognize $L_1 \cup L_2$.

TM $M = "$On input $w \in L_1 \cup L_2$

1. Run $M_1$ and $M_2$ alternately on $w$, one step at a time
2. If either accepts, **accept**"
Example 3: If $L$ is TD, so is $L^*$.  

Pf: Let TM $M$ decide $L$. Then we have the following TM $M^*$ to decide $L^*$.  

Note: $w \in L^*$ if $w = w_1 w_2 \cdots w_k$ for some $k \in [1, |w|]$, where $w_i \in L$ for $i = 1, \cdots, k$.

TM $M^*$ = ”On input $w$

1. If $w = \varepsilon$, accept  
2. $\forall k = 1, 2, \cdots, |w|$  
3. $\forall$ partitions of $w$ into $k$ substrings, i.e., $w_1, w_2, \cdots, w_k$  
4. Run $M$ on $w_1, w_2, \cdots, w_k$  
5. If $M$ accepts $w_i$, $\forall i = 1, \cdots, k$, accept  
6. reject”

124
Example 4: If $L$ is TR, so is $L^*$

**Pf:** Let TM $M$ recognize $L$. Then we have the following NTM $N$ to recognize $L^*$

NTM $N =$ "On input $w \in L^*$

1. Nondeterministically generate (guess) a partition of $w$ into $w_1, w_2, \ldots, w_k$
2. Run $M$ on $w_1, w_2, \ldots, w_k$
3. If $M$ accepts $w_i$, $\forall i = 1, 2, \ldots, k$, accept"
9.1 A binary encoding scheme for TMs

- TM $\Leftrightarrow$ binary number.
  
  $Q = \{q_1, q_2, \ldots, q_{|Q|}\}$ with $q_1$ to be the start state, $q_2$ to be the accept state, and $q_3$ to be the reject state.

  $\Gamma = \{X_1, X_2, \ldots, X_{|\Gamma|}\}$.

  $D = \{D_1, D_2\}$ with $D_1$ to be $L$ and $D_2$ to be $R$.

  A transition $\delta(q_i, X_j) = (q_k, X_l, D_m)$ is coded as $0^i10^j10^k10^l10^m$.

  A TM is coded as $C_111C_211$ $\cdots$ $11C_n$, where each $C$ is the code for a transition.

- An example: $\delta(q_2, X_3) = (q_1, X_4, D_1)$ can be coded as $001000101000010$

- An example: $000010010100100$ is the encoding of $\delta(q_4, X_2) = (q_1, X_2, D_2)$
- TM $M$ with input $w$ is represented by $\langle M, w \rangle$ and encoded as $\langle M \rangle 111w$.

- Using similar schemes, we can encode DFA, NFA, PDA, RE, and CFG into binary strings.
9.2 Decidable languages (Sipser 4.1, pp. 194-201)

▶ $A_{DFA} = \{ < B, w > | B \text{ is a DFA that accepts string } w \}$. 
▶ $A_{NFA} = \{ < B, w > | B \text{ is an NFA that accepts string } w \}$. 
▶ $A_{REX} = \{ < R, w > | R \text{ is a RE that generates string } w \}$. 
▶ $E_{DFA} = \{ < B > | B \text{ is a DFA and } L(B) = \emptyset \}$. 
▶ $EQ_{DFA} = \{ < B_1, B_2 > | B_1 \text{ and } B_2 \text{ are DFAs and } L(B_1) = L(B_2) \}$. 
▶ $A_{CFG} = \{ < G, w > | G \text{ is a CFG that generates string } w \}$. 
▶ $E_{CFG} = \{ < G > | G \text{ is a CFG and } L(G) = \emptyset \}$. 
▶ Every CFL is decidable.

Note: The proofs of these TDLs can be found in Sipser’s book.
Diagonalization: A proof method

- The size of an infinite set: Countably infinite (or countable) and uncountably infinite (or uncountable).

- A set $A$ is countable if there is a 1-1 correspondence with $N = \{1, 2, 3, \ldots\}$ (the set of natural numbers).

- The following sets are countable.
  1. The set of even (or odd) numbers
  2. The set of rationale numbers
  3. The set of binary strings
  4. The set of TMs

- But, the set of languages is uncountable. (How to prove this?)

- So, there are more languages than there are TMs. So there must be languages that are non-TRL.
**Example** (Sipser p. 205): $R$, the set of real numbers, is uncountable.

**Pf:** Assume $R$ is countable. Then $R = \{r_1, r_2, \ldots \}$.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$f(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>3. 14159…</td>
</tr>
<tr>
<td>$r_2$</td>
<td>55. 55555…</td>
</tr>
<tr>
<td>$r_3$</td>
<td>0. 12345…</td>
</tr>
<tr>
<td>$r_4$</td>
<td>0. 50000…</td>
</tr>
<tr>
<td>…</td>
<td>…   …</td>
</tr>
</tbody>
</table>

1. Consider the fractions after the decimal point of each real number. Focus on the diagonal, 1, 5, 3, 0, ….
2. Define a new real number $r = 0.d_1d_2d_3d_4\cdots$, where $d_i$ can be anything but the $i$th number in the diagonal, e.g., $r = 0.2401\cdots$.
3. So $r$ cannot be $r_1, r_2, r_3, r_4, \cdots$, thus $r$ can not be in the table.
4. Contradiction to that all real numbers are in the table.
Consider the binary alphabet.
Order and label strings: $\varepsilon, 0, 1, 00, 01, 10, 11, \ldots$. Let $w_i$ be $i$th string in the above lexicographic ordering.
Order and label TMs: $M_1, M_2, M_3, \ldots$. Let $M_i$ be the TM whose code is $w_i$, i.e. $< M_i > = w_i$. In case $w_i$ is not a valid TM code, let $M_i$ be the TM that immediately rejects any input, i.e., $L(M_i) = \emptyset$.

<table>
<thead>
<tr>
<th>$\varepsilon$</th>
<th>0</th>
<th>1</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
<th>000</th>
<th>$\vdots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$</td>
<td>$w_2$</td>
<td>$w_3$</td>
<td>$w_4$</td>
<td>$w_5$</td>
<td>$w_6$</td>
<td>$w_7$</td>
<td>$w_8$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$M_1$</td>
<td>$M_2$</td>
<td>$M_3$</td>
<td>$M_4$</td>
<td>$M_5$</td>
<td>$M_6$</td>
<td>$M_7$</td>
<td>$M_8$</td>
<td>$\vdots$</td>
</tr>
</tbody>
</table>

For any string $w_i$, there is a TM $M_i$.
For any TM $M_i$, there is a string $w_i$, where $< M_i > = w_i$. 

131
Define diagonalization language $A_D = \{ w_i | w_i \notin L(M_i) \}$.

The corresponding decision problem:
Input: Any binary string $w_i$
Question: Is $w_i$ not accepted by $M_i$?

Prove that $A_D$ is non-TR (not a TRL).

Proof:
1. Suppose, by contradiction, $A_D$ is TR, i.e., there is a TM $M$ such that $A_D = L(M)$.
2. Then $M = M_i$ with code $w_i$ for some $i$.
3. $w_i \in A_D$ iff $w_i \notin L(M_i)$ by definition of $A_D$.
4. $w_i \in A_D$ iff $w_i \in L(M_i)$ by $A_D = L(M_i)$.
5. A contradiction within the two iff statements.
A universal TM:

- Each TM (among those discussed) can only solve a single problem, however, a computer can run arbitrary algorithms. Can we design a general-purposed TM that can solve a wide variety of problems just as a computer?
- Theorem: There is a universal TM $U$ which simulates an arbitrary TM $M$ with input $w$ and produces the same output.

$$\text{TM } U = \text{"On input } < M, w > \text{, Run } M \text{ on } w\"$$

- TM $U$ is an abstract model for computers just as TM $M$ is a formal notion for algorithms.

Figure 29: The universal Turing machine
A review of some new concepts from the last lecture

- Encoding of TM, DFA, NFA, PDA, RE, Graph, Matrix, list, etc. to binary strings: e.g., $<M>$, $<M,w>$, $<M_1,M_2>$

- Infinite sets: Countable vs. uncountable. Compare the set of TMs (countable) vs. the set of languages (uncountable). There are languages without a TM to accept/recognize.

- Diagonalization: a method of self-referencing, used to demonstrate certain properties are inherently contradictory. For example, Russell’s paradox and Barber’s paradox

- Correspondence between binary strings and Turing machines, i.e., for any $w_i$, there is a $M_i$ and for any $M$, there is $i$ s.t. $<M>=w_i$. Thus $M$ can be renamed as $M_i$

- The diagonalization language $A_D = \{w_i \mid w_i \not\in L(M_i)\}$ non-TR.

- Universal TM $U$: An abstract model for computers
Let $A_{TM} = \{ <M, w> | M \text{ accepts string } w \}$

Or equivalently $A_{TM} = \{ <M, w> | w \in L(M) \}$

Or equivalently as a decision problem

Input: A TM $M$ and a string $w$

Question: Is $w$ accepted by $M$?

$A_{TM}$ is called the universal language.

$A_{TM}$ is TR since it can be recognized by TM $U$.

TM $U = \text{"On input } <M, w> \in A_{TM}\text{, run } M \text{ on } w\text{. If } M \text{ accepts } w, \text{ accept"}
\( A_{TM} = \{ <M, w> | w \in L(M) \} \) is non-TD. (By C&C)

1. Assume that \( A_{TM} \) is decided by TM \( T \).
   
   Input to \( T \) is \( <M, w> \)
   
   Output from \( T \) is **accept** if \( w \in L(M) \) and **reject** if \( w \not\in L(M) \)

2. On input \( <M, w> \), \( T \) accepts \( <M, w> \) iff \( M \) accepts \( w \).
   (We can also say, \( T \) rejects \( <M, w> \) iff \( M \) rejects \( w \).)

3. Define TM \( D \) as follows:

4. Observe that \( D \) accepts \( <M> \) iff \( T \) rejects \( <M, <M>> \).

5. Feed \( <D> \) to \( D \).

6. From steps 4 and 2, \( D \) accepts \( <D> \) iff \( T \) rejects \( <D, <D>> \) iff \( D \) rejects \( <D> \).

$A_{TM} = \{ < M, w > | w \in L(M) \}$ is non-TD.

Proof by contradiction:

1. Assume that $A_{TM}$ is decided by TM $T$.
   Input to $T$ is $< M, w >$
   Output from $T$ is accept if $w \in L(M)$ and reject if $w \notin L(M)$

2. On input $< M, w >$, $T$ accepts $< M, w >$ iff $M$ accepts $w$.
   (We can also say, $T$ rejects $< M, w >$ iff $M$ rejects $w$.)

3. Define TM $D =$ "On input $< M >$
   Run $T$ on $< M, < M >>$
   If $T$ accepts, reject
   If $T$ rejects, accept"

4. Observe that $D$ accepts $< M >$ iff $T$ rejects $< M, < M >>$.

5. Feed $< D >$ to $D$.

6. From steps 4 and 2, $D$ accepts $< D >$ iff $T$ rejects $< D, < D >>$ iff $D$ rejects $< D >$.

10.1 A summary of terminology in Computability Theory

- Language, Decision Problem, Problem
- TM, Algorithm, Solution
- Decide, Solve, (Decidable, Solvable)
- Undecidable, Unsolvable
- Accept, Recognize, (Acceptable, Recognizable)
10.2 A review of some languages and corresponding decision problems

- \( A_D = \{ w_i \mid w_i \not\in L(M_i) \} \) (non-TR)
  - Input: Any binary string \( w_i \)
  - Question: Is \( w_i \) not accepted by \( M_i \)?

- \( A_{TM} = \{ < M, w > \mid w \in L(M) \} \) (TR but non-TD)
  - Input: TM \( M \) and string \( w \)
  - Question: Does \( M \) accept \( w \)?
  - It is undecidable whether TM \( M \) accepts string \( w \) for any given \( M \) and \( w \).

- \( HALT_{TM} = \{ < M, w > \mid M \text{ halts on } w \} \) (TR but non-TD)
  - Input: TM \( M \) and string \( w \)
  - Question: Does \( M \) halt on \( w \)?
  - It is undecidable whether TM \( M \) halts string \( w \) for any given \( M \) and \( w \).
10.3 Reducibility or Reduction (Sipser 5 (pp. 216-220))

We say that problem $A$ reduces (or is reducible) to problem $B$, written as $A \leq B$, if we can use a solution (TM) to $B$ to solve $A$ (i.e., if $B$ is decidable/solvable, so is $A$).

We may use reducibility to prove undecidability as follows:

1. Let $A$ be non-TD, such as $A_D$ or $A_{TM}$. Wish to prove $B$ is non-TD.
2. Assume $B$ is TD. Then there exists a TM $M_B$ to decide $B$.
3. If we can use $M_B$ as a sub-routine to construct a TM $M_A$ that decides $A$, then $A$ is TD. We have a contradiction.
4. The construction of TM $M_A$ using TM $M_B$ establishes that $A$ reduces to $B$, i.e., $A \leq B$. ($A$ is no harder than $B$)
5. Corollary 5.23 (Sipser p. 236):
   If $A \leq B$ and $A$ is non-TD, then $B$ is non-TD.
10.4 A proof of the non-TD $A_{TM}$ by reduction

Proof sketch:
1. Assume $A_{TM}$ is TD, by contradiction.
2. Let TM $S$ decide $A_{TM}$, by the definition of a TDL.
3. Try to construct a TM $D$ that decides $A_D$. The construction will include TM $S$. This shows $A_D$ is TD.
4. A contradiction since we know $A_D$ is non-TR.

Note: This proof uses the TM $S$ for $A_{TM}$ to build a TM $D$ for $A_D$, i.e., $A_D \leq A_{TM}$.

Recall two languages:
1. $A_{TM} = \{ <M, w> | w \in L(M) \}$.
2. $A_D = \{ w_i | w_i \notin L(M_i) \}$. ($A_D$ is non-TR)
Prove that $A_{TM}$ is non-TD by reduction

1. Assume $A_{TM}$ is TD, by contradiction.

2. Let TM $S$ decide $A_{TM}$, i.e.,

$$S(<M, w>) = \begin{cases} 
\text{accept} & w \in L(M) \\
\text{reject} & w \notin L(M) 
\end{cases}$$

3. Construct a TM $D$ that decides $A_D$, a non-TRL.

   TM $D =$”On input $w_i$
   
   Run $S$ on $<M_i, w_i>$
   
   If $S$ accepts, **reject** else **accept**”

4. Why does $D$ decide $A_D$?

   $S$ accepts $<M_i, w_i>$ iff $w_i \in L(M_i)$ iff $w_i \notin A_D$ iff $D$ rejects $w_i$. So $S$ accepts iff $D$ rejects.

5. So $A_D$ is TD. A contradiction.
10.5 The halting problem (Theorem 5.1 (pp. 216-217))

▶ \( \text{HALT}_{TM} = \{ < M, w > | M \text{ halts on string } w \} \).

▶ \( A_{TM} = \{ < M, w > | M \text{ accepts } w \} \)

▶ \( A_{TM} \subseteq \text{HALT}_{TM} \)

▶ \( \text{HALT}_{TM} \) is TR since it can be recognized by TM \( U \).

▶ **Theorem 5.1** \( \text{HALT}_{TM} \) is non-TD.
  (Will show \( A_{TM} \leq \text{HALT}_{TM} \))

**Theorem** \( \text{HALT}_{TM} \) is non-TD. Prove by reduction from \( A_{TM} \), i.e., \( A_{TM} \leq \text{HALT}_{TM} \)
1. Assume TM $R$ decides $\text{HALT}_{TM}$. Then $R$ accepts $<M,w>$ iff $M$ halts on $w$. Construct TM $S$ to decide $A_{TM}$.

$TM \ S = \"On \ input \ <M,w> \nRun \ R \ on \ <M,w> \nif \ R \ rejects, \ reject \nif \ R \ accepts, \ run \ M \ on \ w \ until \ it \ halts \n\quad \text{if} \ M \ accepts, \ \textbf{accept}; \ \text{else} \ \textbf{reject}\"$

2. Why does $S$ accept $A_{TM}$?
$R$ rejects $<M,w> \Rightarrow M$ doesn’t halt on $w \Rightarrow M$ doesn’t accept $w \Rightarrow <M,w> \notin A_{TM} \Rightarrow S$ rejects
$R$ accepts $<M,w> \Rightarrow M$ halts on $w$ (accepts or rejects? Need to run $M$ on $w$ to find out)
$M$ accepts $w \Rightarrow <M,w> \in A_{TM}$

3. Since we constructed a TM $S$ that decides $A_{TM}$ using TM $R$, so $A_{TM}$ is TD. A contradiction to that $A_{TM}$ is proved to be non-TD.
10.6 Other non-TD problems (*Sipser 5.1 (pp. 216-220)*)

The following problems about Turing machines are non-TD:

- Whether \( L(M) = \emptyset \) for any TM \( M \).
  
  \[ E_{TM} = \{ < M > | L(M) = \emptyset \} \]
  
  \[ NE_{TM} = \{ < M > | L(M) \neq \emptyset \} \] (complement of \( E_{TM} \))

- Whether \( L(M_1) = L(M_2) \) for any two TMs \( M_1 \) and \( M_2 \).
  
  \[ EQ_{TM} = \{ < M_1, M_2 > | L(M_1) = L(M_2) \} \]

- Whether \( L(M) \) is finite for any TM \( M \)

  \[ FINITE_{TM} = \{ < M > | L(M) \text{ is finite} \} \]

- Whether \( \varepsilon \in L(M) \) for any TM \( M \).

  \[ ESTRING_{TM} = \{ < M > | \varepsilon \in L(M) \} \]

- Whether \( L(M) = \Sigma^* \) for any TM \( M \).

  \[ ALL_{TM} = \{ < M > | L(M) = \Sigma^* \} \]

**Rice’s Theorem**: Every nontrivial property of the TRLs (or TMs) is undecidable.
Theorem 5.2 Prove that $E_{TM} = \{ < M > \mid L(M) = \emptyset \}$ is non-TD.

Proof: (1) Assume that $E_{TM}$ is decidable by TM $R$.

$$R(<M>) = \begin{cases} 
\text{accept} & L(M) = \emptyset \\
\text{reject} & L(M) \neq \emptyset 
\end{cases}$$

(2) Use $R$ to construct TM $S$ that decides $A_{TM}$, i.e., $A_{TM} \leq E_{TM}$.

TM $S =$ "On input $<M, w>$, 

- Construct TM $M_1 =$ "On input $x$
  
  If $x \neq w$ reject else run $M$ on $w$"
  
- Run $R$ on $<M_1>$. 
  
- If $R$ accepts, reject; and if $R$ rejects, accept"

(3) Why does $S$ decide $A_{TM}$? $L(M_1) = \emptyset$ if $M$ does not accept $w$; and $L(M_1) = \{w\}$ if $M$ accepts $w$. I.e., $L(M_1) = \emptyset$ iff $w \not\in L(M)$. So $R$ accepts $<M_1>$ iff $L(M_1) = \emptyset$ iff $w \not\in L(M)$ iff $S$ rejects.

(4) TM $S$ decides the non-TD $A_{TM}$. A contradiction.
A graphical explanation of the undecidability proof of $E_{TM}$

Figure 30: Reduction from $A_{TM}$ to $E_{TM}$

Important questions to answer:

▶ Input: how to define $M_1$ (the input to $R$) using $<M, w>$ (the input to $S$)?

▶ Output: how the output from $R$ implies the output from $S$?

Goal: Design $M_1$ such that the output from $R$ defines that of $S$. 

147
Prove that $NE_{TM} = \{ <M> | L(M) \neq \emptyset \}$ is TR but non-TD.

1. To prove $NE_{TM}$ is TR, we give a NTM $N$ to recognize $NE_{TM}$.

   NTM $N$ = "On input $<M> \in NE_{TM}$
   
   - Guess a string $w$
   - Run $M$ on $w$
   - If $M$ accepts, accept"

We can also use a deterministic TM to recognize $NE_{TM}$.

TM $D$ = "On input $<M> \in NE_{TM}$

Recall the binary sequence $w_1, w_2, w_3, \ldots$

- Systematically generates strings: $\varepsilon, 0, 1, 00, 01, \ldots$
- for $i = 1, 2, 3, \ldots$
  - Run $M$ on $w_1, \ldots, w_i$, each for $i$ steps
- If in the loop above, $M$ ever accepts some $w_j$, then accept"
An explanation of the TM $D$ that recognizes $NE_{TM}$:

Assume $w_9$ is accepted by $M$ in 7 steps.
Assume $w_{10}$ is accepted by $M$ in 12 steps.

$i = 1$: Run $M$ on $w_1$ for 1 step;
$i = 2$: Run $M$ on $w_1, w_2$ each for 2 steps;
$i = 3$: Run $M$ on $w_1, w_2, w_3$ each for 3 steps;
......
$i = 9$: Run $M$ on $w_1, w_2, \cdots, w_9$ for 9 steps; (accepted)
......
$i = 12$: Run $M$ on $w_1, w_2, \cdots, w_{10}, \cdots, w_{12}$ for 12 steps (accepted)
To prove \( NE_{TM} = \{< M > | L(M) \neq \emptyset\} \) is non-TD, assume it is decided by TM \( R \). Then \( R \) accepts \( < M > \) iff \( L(M) \neq \emptyset \).

Construct a TM \( S \) that decides the undecidable \( A_{TM} \). Then a contradiction.

\[ TM \ S = \text{"On input } < M, w > \]

1. Construct TM \( M_1 = \text{"On input } x \)
   
   If \( x \neq w \), reject else Run \( M \) on \( w \)”

2. Run \( R \) on \( < M_1 > \)

3. If \( R \) accepts, accept; else reject”

Why does \( S \) accept \( A_{TM} \)?

\( L(M_1) = \emptyset \) if \( w \not\in L(M) \) and \( L(M_1) = \{w\} \) if \( w \in L(M) \). In other words, \( L(M_1) \neq \emptyset \) iff \( w \in L(M) \).

\( R \) accepts \( < M_1 > \) iff \( L(M_1) \neq \emptyset \) iff \( w \in L(M) \) iff \( < M, w > \in A_{TM} \)

iff \( S \) accepts \( < M, w > \).

So \( A_{TM} \) is TD. A contradiction.
About $E_{TM}$ and its complement $NE_{TM}$

We proved: $E_{TM}$ is non-TD. $NE_{TM}$ is TR.

Recall the theorem on page 120. For $A$ and $\bar{A}$,

1. Both are TD; (Both are TR)
2. Neither is TR;
3. One is TR but non-TD, the other is non-TR

We immediately have the following results.

(1) $NE_{TM}$ is non-TD (If $NE_{TM}$ is TD, so is $E_{TM}$)
(2) $E_{TM}$ is non-TR (If $E_{TM}$ is TR, both $E_{TM}$ and $NE_{TM}$ are TD)
Theorem 5.4 \( EQ_{TM} = \{ < M_1, M_2 > \mid L(M_1) = L(M_2) \} \) is non-TD. Reduce from \( E_{TM} = \{ < M > \mid L(M) = \emptyset \} \).

1. Assume \( EQ_{TM} \) is decided by TM \( R \).

\[ R(< M_1, M_2 >) = \begin{cases} \text{accept} & \text{if } L(M_1) = L(M_2) \\ \text{reject} & \text{if } L(M_1) \neq L(M_2) \end{cases} \]

2. Construct TM \( S \) that decides the undecidable \( E_{TM} \).

TM \( S = \) "On input \( < M > \)

Construct TM \( M_1 = \) "On input \( x \)

Immediately reject" //\( L(M_1) = \emptyset \)

Run \( R \) on \( < M_1, M > \)

If \( R \) accepts, accept; else reject”

3. Why does \( S \) decides \( E_{TM} \)? \( R \) accepts \( < M_1, M > \) iff \( L(M_1) = L(M) \) iff \( L(M) = \emptyset \) iff \( S \) accepts \( < M > \).

4. \( S \) decides \( E_{TM} \). So \( E_{TM} \) is TD. A contradiction.
10.7 Post’s correspondence problem (PCP) (Sipser 5.2)

INPUT: $P = \{ \frac{t_1}{b_1}, \frac{t_2}{b_2}, \ldots, \frac{t_k}{b_k} \}$, where $t_1, t_2, \ldots, t_k$ and $b_1, b_2, \ldots, b_k$ are strings over alphabet $\Sigma$. ($P$ is a collection of dominos, each containing two strings, with one stacked on top of the other.)

QUESTION: Does $P$ contain a match? Or, is there $i_1, i_2, \ldots, i_l \in \{1, 2, \ldots, k\}$ with $l \geq 1$ such that $t_{i_1} t_{i_2} \cdots t_{i_l} = b_{i_1} b_{i_2} \cdots b_{i_l}$?

Equivalently, defined as a language, we have $L_{PCP} = \{ <P> | P \text{ is an instance of PCP with a match} \}$.

For input $P_1 = \{ \frac{b}{ca}, \frac{a}{ab}, \frac{ca}{a}, \frac{abc}{c} \}$, sequence $2, 1, 3, 2, 4$ indicates a match. Since $\frac{a}{ab} \frac{b}{ca} \frac{ca}{a} \frac{a}{ab} \frac{abc}{c}$, top=bottom=abcaaaabc

For $P_2 = \{ \frac{abc}{ab}, \frac{ca}{a}, \frac{acc}{ba} \}$, there is no match since all top strings are longer than bottom strings.

PCP is non-TD for the binary alphabet.
A Summary of Computability Theory
1. Definitions and concepts:
   ▶ Turing machine, how it works, its language, its encoding, Church-Turing Thesis
   ▶ TRL and TDL, properties, how M accepts/decides a language, implementation-level description
   ▶ Reduction, the meaning of $A \leq B$ ($A$ is no harder than $B$), use reduction to prove undecidability
2. Various proofs:
   ▶ A language is TR/TD (prove by definition)
   ▶ A language is non-TR/non-TD (prove by a combination of contradiction, construction, and reduction)
   ▶ Many examples to learn from
Complexity Theory:

- Computability Theory is the study of what can or cannot be computed by a TM/algorithm, among all problems.
- Complexity Theory is the study of what can or cannot be computed *efficiently* by a TM/algorithm, among all decidable/solvable problems.
- For the set of all solvable problems, it is further classified into various **complexity classes** based on the efficiency of algorithms solving these problems.
- Complexity Theory is the study of the definition and properties of these classes.
Figure 31: Three complexity classes if $P=NP$ or $P \neq NP$
11.1 The class of $P$ (Sipser 7.2)

- **Definition:** $P$ is the class of problems solvable in polynomial time (number of steps) by deterministic TMs. Polynomial $O(n^c)$, where $n$ is input size and $c$ is a constant. Problems in $P$ are "tractable" (not so hard).

- **Why use polynomial as the criterion?**
  - If a problem is not in $P$, it often requires unreasonably long time to solve for large-size inputs.
  - $P$ is independent of all models of computation, except nondeterministic TM.

- **Problems in $P$:** Sorting, Searching, Selecting, Minimum Spanning Tree, Shortest Path, Matrix Multiplication, etc.

- **Review of asymptotic notation:** $O$, $\Omega$, $\Theta$

- **Examples of polynomial and polylog functions:** $O(1)$, $O(n)$, $O(n^2)$, $O(n^d)$, $O(\log n)$, $O((\log n)^c)$, $O(n^3 \log n)$, $O(n^c (\log n)^d)$
11.2 The class of NP \((Sipser 7.3)\)

- An NTM is an unrealistic (unreasonable) model of computing which can be simulated by other models with an exponential loss of efficiency. It is a useful concept that has had great impact on the theory of computation.

- NTM \(N = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\), where \(\delta : Q \times \Gamma \rightarrow 2^P\) for \(P = Q \times \Gamma \times \{L, R\}\).

- \(\delta(q, X)\) is a set a moves. Which one to choose? This is nondeterminism. The computation can be illustrated by a tree, with each node representing a configuration.
Nondeterminism can be viewed as a kind of parallel computation wherein multiple independent processes or threads can be running concurrently. When a nondeterministic machine splits into several choices, that corresponds to a process forking into several children, each then proceeding separately. If at least one process accepts, then the entire computation accepts.

Time complexity of nondeterministic TMs (NTMs): Let \( N \) be an NTM that is a decider (where all computation paths halt in the tree for any input). The time complexity of \( N \), \( f(n) \), is the maximum number of steps that \( N \) uses on any computation path for any input of length \( n \). In other words, \( f(n) \) is the maximum height of all computation trees for all input of length \( n \).
An unreasonable model of computation:

**Theorem:** Every $T(n)$-time multi-tape TM has an equivalent $O(T^2(n))$-time single-tape TM.

**Theorem:** Every $T(n)$-time single-tape NTM has an equivalent $O(2^{O(T(n))})$-time single-tape DTM.

**Definition:** NP is the class of problems solvable in polynomial time by nondeterministic TMs.

**Another definition of nondeterministic TMs (algorithms):**
- Guessing phase: Guess a solution (always on target).
- Verifying phase: Verify the solution.
Example: TSP (DEC) is in **NP**.

**INSTANCE:** An edge-weighted graph $G(V, E, w)$ and a bound $B \geq 0$.

**QUESTION:** Is there a tour (a cycle that passes through each node exactly once) in $G$ with total weight no more than $B$?

Define the following NTM $N$ to solve TSP in polynomial time.

**NTM $N$** = ”On input $\langle G, B \rangle$"

1. Nondeterministically **guess** a tour $T$ in $O(|V|)$
2. **Verify** if $T$ includes every node once in $O(|V|)$
3. Compute $\text{sum} \leftarrow \sum_{e \in T} w(e)$ in $O(|V|)$
4. **Verify** if $\text{sum} \leq B$. If true, answer **yes**; else **No** in $O(1)$

Note 1: The time complexity of $N$ is $O(|V|)$.

Note 2: If the answer to the **QUESTION** is ”Yes”, $N$ guarantees that the right $T$ will be guessed in step 1.
Note 3: The acceptance of an input by a nondeterministic machine is determined by whether there is an accepting computation among all, possibly exponentially many, computations.

In the above proof, if there is a solution, i.e., a tour with total weight no more than $B$, it will always be generated by the Turing machine. This is like that a nondeterministic machine has a guessing power.

A tour can only be found by a deterministic machine in exponential time, however, it can be found by a nondeterministic machine in just linear steps. Any nondeterministic proof should always contain two stages: Guessing and verifying.

What needs to be guessed? What needs to be verified? What is the time complexity?
Example: Graph Coloring (GC) is in \textbf{NP}.

INSTANCE: Graph \( G = (V, E) \), and \( B \geq 0 \)

QUESTION: Is there a coloring scheme of the nodes that uses no more than \( B \) colors such that no two nodes connected by an edge are given the same color?

NTM \( N = \) "On input < \( G, B \) >

1. Guess a coloring scheme (in polynomial time) \( c : V \rightarrow C \)
2. Verify if (1) \( |C| \leq B \) and (2) \( \forall (u, v) \in E, c(u) \neq c(v) \)
3. if true, answer \( \text{yes} \); else answer \( \text{no} \)

So we have a \textbf{nondeterministic} algorithm (or TM) that \textbf{guesses} a coloring scheme (or function) and \textbf{verifies} that (1) for any \( (u, v) \in E, c(u) \neq c(v) \) and that (2) the number of colors used is no more than \( B \), and further, all these can be done in polynomial time of \( O(|V|) + O(|E|) \). So GC is in \textbf{NP}.

163
Theorem: \( P \subseteq NP \). (Two possibilities: \( P \subset NP \) or \( P = NP \))
Any deterministic TM is a special case of nondeterministic TMs.

Theorem: Any \( \Pi \in NP \) can be solved by a deterministic TM in time \( O(c^{p(n)}) \) for some \( c > 0 \) and polynomial \( p(n) \).

Open problem: \( P = NP ? \)
The west wall bricks on the CS building at Princeton, 1989:
x1010000x
x0111101x
x1001110x
x1001110x
x1010000x
x0111111x
Definition of **Polynomial Reduction** $\leq_p$ (cf. $\leq_m$ and $\leq$)

Let $\Pi_1$ and $\Pi_2$ be two decision problems, and $\{I_1\}$ and $\{I_2\}$ be sets of instances for $\Pi_1$ and $\Pi_2$, respectively. We say there is a polynomial reduction from $\Pi_1$ to $\Pi_2$, or $\Pi_1 \leq_p \Pi_2$, if there is $f : \{I_1\} \rightarrow \{I_2\}$ such that

(1) $f$ can be computed in polynomial time and

(2) $I_1$ has a “yes” solution if and only if $f(I_1)$ has a “yes” solution.

Theorem: If $\Pi_1 \leq_p \Pi_2$, then $\Pi_2 \in \mathbf{P}$ implies $\Pi_1 \in \mathbf{P}$.

Theorem: If $\Pi_1 \leq_p \Pi_2$ and $\Pi_2 \leq_p \Pi_3$, then $\Pi_1 \leq_p \Pi_3$.

Remark: $\leq_p$ means “no harder than”.

![Figure 32: Polynomial reduction $\Pi_1 \leq_p \Pi_2$](image)
11.4 The class of NPC (Sipser 7.4)

- Definition 1: \textbf{NPC} (\textbf{NP}-complete) is the class of the hardest problems in \textbf{NP}

- Definition 2: \( \Pi \in \textbf{NPC} \) if \( \Pi \in \textbf{NP} \) and \( \forall \Pi' \in \textbf{NP}, \Pi' \leq_p \Pi \).

- Definition 3: \( \Pi \in \textbf{NPC} \) if \( \Pi \in \textbf{NP} \) and \( \exists \Pi' \in \textbf{NPC} \) such that \( \Pi' \leq_p \Pi \).

- Theorem: If \( \exists \Pi \in \textbf{NPC} \) such that \( \Pi \in \textbf{P} \), then \( \textbf{P} = \textbf{NP} \).

- Theorem: If \( \exists \Pi \in \textbf{NPC} \) such that \( \Pi \not\in \textbf{P} \), then \( \textbf{P} \neq \textbf{NP} \).
Some most important classes: Definitions and proofs

- **P**: class of problems solvable in polynomial-time by DTM. To prove $\Pi \in P$, design a polynomial-time algorithm.

- **NP**: class of problems solvable in polynomial-time by NTM. To prove $\Pi \in NP$, design a polynomial-time nondeterministic algorithm of two steps: guess and verify.

- **NPC**: class of all hardest problems in **NP**. To prove $\Pi \in NPC$, prove (1) $\Pi \in NP$ and (2) $\exists \Pi' \in NPC$ s.t. $\Pi' \leq_p \Pi$

- **NP-hard**: A problem $X$ is NP-hard, if there is an NP-complete problem $Y$, such that $Y$ is reducible to $X$ in polynomial time. (Note $X$ does not need to be in **NP**)  

Possible relations among **P**, **NP**, **NP-complete**, **NP-hard**: 
$P \subset NP$, $NP \cap NP\text{-}hard = NP\text{-}complete$, $P \cup NP\text{-}complete \neq \emptyset$
Satisfiability (SAT):

INSTANCE: A boolean formula $\phi$ in CNF with variables $x_1, \ldots, x_n$ and clauses $c_1, \ldots, c_m$

QUESTION: Is $\phi$ satisfiable? (Is there a truth assignment $A$ to $x_1, \ldots, x_n$ such that $\phi$ is true?)

$L_{SAT} = \{< \phi > | \exists A \text{ that satisfies } \phi \}$

Example of an instance for SAT:

- Variables: $x_1, x_2, x_3, x_4$
- Literals: Any variables and their negations, such as $x_1, \overline{x_3}$
- Clauses: $c_1 = x_1 \lor \overline{x_2} \lor x_3$, $c_2 = x_1 \lor x_2$, $c_3 = \overline{x_1} \lor x_2 \lor \overline{x_3} \lor x_4$
- Function/formula: $\phi = c_1 \land c_2 \land c_3$
- The instance $\phi$ is T by assignment $x_1 = T, x_2 = x_3 = x_4 = F$. Note: Many assignments satisfy $\phi$, but we only need one.
Cook’s Theorem: SAT $\in$ \textbf{NPC}. (Need to prove (1) SAT $\in$ \textbf{NP} and (2) $\forall \Pi \in \textbf{NP}$, $\Pi \leq_p \text{SAT}$.)

**First Step:** How to prove SAT is in \textbf{NP}?
NTM $N =$ ”On input $< \phi >$ in CNF
1. Guess a truth assignment $A$ $O(n)$
2. Verify if $\phi = T$ under $A$ $O(n + m)$
3. If $T$, accept; else reject”

SAT is solvable by a NTM in polynomial time, thus in \textbf{NP}.

**Second step:** How to prove $\forall \Pi \in \textbf{NP}$, $\Pi \leq_p \text{SAT}$, or equivalently, for any polynomial-time NTM $M$, $L(M) \leq_p L_{\text{SAT}}$?

Will not discuss this proof. But if interested, go to the final six pages of this slide set for details.
11.5 NP-complete problems (Sipser 7.5, pp.310-322)

How to prove $\Pi_2$ is $\textbf{NP}$-complete:

- Show that $\Pi_2 \in \textbf{NP}$.
- Choose a known $\textbf{NP}$-complete $\Pi_1$.
- Construct a reduction $f$ from $\Pi_1$ to $\Pi_2$.
- Prove that $f$ is a polynomial reduction by showing (1) $f$ can be computed in polynomial time and (2) $\forall I_1$ for $\Pi_1$, $I_1$ is a yes-instance for $\Pi_1$ if and only if $f(I_1)$ is a yes-instance for $\Pi_2$.

**Figure 33:** Polynomial reduction $\Pi_1 \leq_p \Pi_2$
Seven basic \textbf{NP}-complete problems.

- **3-Satisfiability (3SAT):** (Reduced from SAT)
  
  INSTANCE: A formula $\alpha$ in CNF with each clause having three literals.
  
  QUESTION: Is $\alpha$ satisfiable?

- **3-Dimensional Matching (3DM):** (Reduced from 3SAT)
  
  INSTANCE: $M \subseteq X \times Y \times Z$, where $X$, $Y$, $Z$ are disjoint and of the same size.
  
  QUESTION: Does $M$ contain a matching, which is $M' \subseteq M$ with $|M'| = |X|$ such that no two triples in $M'$ agree in any coordinate?

- **PARTITION:** (Reduced from 3DM)
  
  INSTANCE: A finite set $A$ of numbers.
  
  QUESTION: Is there $A' \subseteq A$ such that $\sum_{a \in A'} a = \sum_{a \in A - A'} a$?
Vertex Cover (VC): (Reduced from 3SAT)
INSTANCE: A graph $G = (V, E)$ and $0 \leq k \leq |V|$.
QUESTION: Is there a vertex cover of size $\leq k$, where a vertex cover is $V' \subseteq V$ such that $\forall (u, v) \in E$, either $u \in V'$ or $v \in V'$?

Hamiltonian Circuit (HC): (Reduced from VC)
INSTANCE: A graph $G = (V, E)$.
QUESTION: Does $G$ have a Hamiltonian circuit, i.e., a tour that passes through each vertex exactly once?

CLIQUE: (Reduced from VC)
INSTANCE: A graph $G = (V, E)$ and $0 \leq k \leq |V|$.
QUESTION: Does $G$ contain a clique (complete subgraph) of size $\geq k$?
Figure 34: Seven basic NP-complete problems
Test of your understanding of the complexity classes

▶ If $\Pi_1 \leq_p \Pi_2$ and $\Pi_1 \in \text{NP}$, is $\Pi_2 \in \text{NP}$? Maybe.
▶ If $\Pi_1 \leq_p \Pi_2$ and $\Pi_1, \Pi_2 \in \text{NPC}$, is $\Pi_2 \leq_p \Pi_1$? Yes.
▶ If $\Pi_1 \leq_p \Pi_2$ and $\Pi_1 \not\in \text{NP}$, is $\Pi_1 \in \text{P}$? No.
Example to prove reduction: KNAPSACK Problem

INSTANCE: $U = \{u_1, \ldots, u_n\}$, $W$ (max weight knapsack holds), functions $w : U \rightarrow R^+$ and $v : U \rightarrow R^+$, bound $B \geq 0$.

QUESTION: Is there $U' \subseteq U$ s.t. $\sum_{u_i \in U'} w(u_i) \leq W$, and $\sum_{u_i \in U'} v(u_i) \geq B$?

Show PARTITION $\leq_p$ KNAPSACK.

For any instance $\{a_1, \ldots, a_n\}$ in PARTITION, define the following instance for KNAPSACK:

- $U = \{u_1, \ldots, u_n\}$
- $w(u_i) = a_i$ and $v(u_i) = a_i$, for all $i$
- $W = B = \frac{1}{2} \sum_{i=1}^{n} a_i$

Show that (1) the above construction can be done in polynomial time and (2) there is a partition $A' \subseteq A$ iff there is a subset $U'$ s.t. $\sum_{u_i \in U'} w(u_i) \leq \frac{1}{2} \sum_{i=1}^{n} a_i$ and $\sum_{u_i \in U'} v(u_i) \geq \frac{1}{2} \sum_{i=1}^{n} a_i$. 

175
Example to prove reduction: Hitting Set (HS) problem

INSTANCE: Set \( S \), collection \( C \) of subsets of \( S \), \( K \geq 0 \)

QUESTION: Does \( S \) contain a HS \( S' \) of size \( \leq K \)? \((S' \) is a HS if \( S' \) contains at least one element from each subset in \( C \).)

\( VC \leq_p HS. \)

Instance for VC: \( G = (V, E) \) and \( B \geq 0 \)

Instance for HS: \( S = V \), \( C = \{\{u, v\} | \forall e = (u, v) \in E\} \), \( K = B \)

Goal: \( G \) has a VC of size \( \leq B \) iff \( S \) has a HS of size \( \leq K \).

Figure 35: An example of reduction from VC to HS
Proof techniques, languages, complexity classes

- Contradiction, construction, reduction, polynomial reduction, and diagonalization
- RL, CFL, TDL, TRL, non-TR (Chomsky hierarchy)
- DFA/NFA, PDA, TM(decides), TM(accepts), NTM(guesses)
- Closure properties for the above sets of languages
- Pumping Lemmas for RL and CFL
- P, NP, NPC, NP-Hard (relation based on $P \neq NP$, $P=NP$)
- TD, non-TD, TR, non-TR
- Prove a language is TD, TR, non-TD, or non-TR
- Prove a DEC is in NP.
- A good understanding of $A_D$, $A_{TM}$, $HALT_{TM}$, SAT, 3SAT, VC, PARTITION, HC, and those languages and DECs discussed in examples.
Review of Computability Theory

- TDL: How to prove a language is TDL
- The closure properties of TDLs: union, intersection, concatenation, star, complement
- A TM that decides (accept, reject)
- TRL: How to prove a language is TRL
- The closure properties of TRLs: union, intersection, concatenation, star, homomorphism
- A TM that accepts/recognizes
- A language and its complement: Three scenarios (both TD, one TR but non-TD other non-TR, both non-TR)
- Important languages: $A_D$, $A_{TM}$, $HALT_{TM}$, etc.
- Reduction $A \leq B$ is to show any TM that decides $B$ can be used to define a TM that decides $A$. ($A$ is no harder than $B$ or $B$ is at least as hard as $A$.)
Review of Complexity Theory

- Three classes: P, NP, NPC (Also NP-hard)
- Polynomial reduction $A \leq_p B$ is to show any algorithm that solves $B$ can be used to define an algorithm that solves $A$. ($A$ is no harder than $B$ or $B$ is at least as hard as $A$.)
- Important NP-complete problems: SAT, 3SAT, VC, HC, PARTITION, 3DM, CLIQUE, COLOR, KNAPSACK, HS
Some sample problems

- The universal language, $A_{TM}$, is a proper (non-equal) subset of the halting language, $HALT_{TM}$.
- The Post Correspondence Problem is decidable for the unary alphabet.
- If $L$ is TR but non-TD, then $\overline{L}$ is non-TR.

![Venn diagram for TD, TR, and non-TR]

**Figure 36:** Venn diagram for TD, TR, and non-TR

- If $A$ is non-TD and $A \leq C$ and $D \leq C$, then $D$ must be non-TR.
A problem that can be solved by a deterministic TM in $2^n$ steps must be intractable.

If $\Pi_1$ is not in $\textbf{NP}$, then $\Pi_1$ may still be in $\textbf{P}$

If $A \leq_p B$, then it is possible that $A \in P$ but $B \not\in \textbf{NP}$.

In order to prove a problem $X$ is NP-hard, one needs to develop a polynomial reduction from $X$ to a known NP-hard problem.

Prove that $L = \{<M>| M \text{ accepts } 01101\}$ is non-TD

Prove that 3DM is in NP

Prove there is a polynomial reduction from SAT to 3SAT
A proof that 3SAT is NP-complete:
First, 3SAT is obvious in \textbf{NP}.
Next, we show that \text{SAT} \leq_p 3\text{SAT}.
Given any instance of SAT, \(f(x_1, \ldots, x_n) = c_1 \land \cdots \land c_m\), where \(c_i\) is a disjunction of literals. To construct an instance for 3SAT, we need to convert any \(c_i\) to an equivalent \(c'_i\), a conjunction of clauses with exactly 3 literals.

Case 1. If \(c_i = z_1\) (one literal), define \(y_i^1\) and \(y_i^2\). Let \(c'_i = (z_1 \lor y_i^1 \lor y_i^2) \land (z_1 \lor y_i^1 \lor \neg y_i^2) \land (z_1 \lor \neg y_i^1 \lor y_i^2) \land (z_1 \lor \neg y_i^1 \lor \neg y_i^2)\).

Case 2. If \(c_i = z_1 \lor z_2\) (two literals), define \(y_i^1\). Let \(c'_i = (z_1 \lor z_2 \lor y_i^1) \land (z_1 \lor z_2 \lor \neg y_i^1)\).

Case 3. If \(c_i = z_1 \lor z_2 \lor z_3\) (three literals), let \(c'_i = c_i\).
Case 4. If $c_i = z_1 \lor z_2 \lor \cdots \lor z_k$ ($k > 3$), define $y_i^1, y_i^2, \ldots, y_i^{k-3}$. Let $c'_i = (z_1 \lor z_2 \lor y_i^1) \land (\lnot y_i^1 \lor z_3 \lor y_i^2) \land (\lnot y_i^2 \lor z_4 \lor y_i^3) \land \cdots \land (\lnot y_i^{k-3} \lor z_{k-1} \lor z_k)$.

If $c_i$ is satisfiable, then there is a literal $z_l = T$ in $c_i$. If $l = 1, 2$, let $y_i^1, \ldots, y_i^{k-3} = F$. If $l = k - 1, k$, let $y_i^1, \ldots, y_i^{k-3} = T$. If $3 \leq l \leq k - 2$, let $y_i^1, \ldots, y_i^{l-2} = T$ and $y_i^{l-1}, \ldots, y_i^{k-3} = F$. So $c'_i$ is satisfiable.

If $c'_i$ is satisfiable, assume $z_l = F$ for all $l = 1, \ldots, k$. Then $y_i^1, \ldots, y_i^{k-3} = T$. So the last clause $(\lnot y_i^{k-3} \lor z_{k-1} \lor z_k) = F$. Therefore, $c'_i$ is not satisfiable. Contradiction.

The instance of 3SAT is therefore $f'(x_1, \ldots, x_n, \ldots) = c'_1 \land \cdots \land c'_m$, and $f$ is satisfiable if and only if $f'$ is satisfiable.
Cook’s Theorem: SAT is \textbf{NP}-complete.

Proof. SAT is clearly in \textbf{NP} since a NTM exists that guesses a truth assignment and verifies its correctness in polynomial time. Now we wish to prove \(\forall \Pi \in \textbf{NP}, \Pi \leq_p \text{SAT}\), or equivalently, for any polynomial-time NTM \(M\), \(L(M) \leq_p L_{\text{SAT}}\).

For any NTM \(M\), assume \(Q = \{q_0, q_1(\text{accept}), q_2(\text{reject}), \ldots, q_r\}\) and \(\Gamma = \{s_0, s_1, s_2, \ldots, s_v\}\). Also assume that the time is bounded by \(p(n)\), where \(n\) is the length of the input.

We wish to prove that there is a function \(f_M : \Sigma^* \rightarrow \{\text{instances of SAT}\}\) such that \(\forall x \in \Sigma^*, x \in L(M)\) iff \(f_M(x)\) is satisfiable. In other words, we wish to use a Boolean expression \(f_M(x)\) to describe the computation of \(M\) on \(x\).
Variables in $f_M(x)$:
— State: $Q[i, k]$. $M$ is in $q_k$ after the $i$th step of computation (at time $i$).
— Head: $H[i, j]$. Head points to tape square $j$ at time $i$.
— Symbol: $S[i, j, l]$. Tape square $j$ contains $s_l$ at time $i$.
(Assume the tape is one-way infinite and the leftmost square is labeled with 0.)

For example, initially $i = 0$. Assume the configuration is $q_0abba$. Let $s_0 = B$, $s_1 = a$, and $s_2 = b$. Therefore, we set the following Boolean variables to be true: $Q[0, 0]$, $H[0, 0]$, $S[0, 0, 1]$, $S[0, 1, 2]$, $S[0, 2, 2]$, $S[0, 3, 1]$ and $S[0, j, 0]$ for $j = 4, 5, \ldots$. A configuration defines a truth assignment, but not vice versa.
Clauses in \( f_M(x) \):

— At any time \( i \), \( M \) is in exactly one state.
  \( Q[i,0] \lor \cdots \lor Q[i,r] \) for \( 0 \leq i \leq p(n) \).
  \( \neg Q[i,k] \lor \neg Q[i,k'] \) for \( 0 \leq i \leq p(n) \) and \( 0 \leq k < k' \leq r \).

— At any time \( i \), head is scanning exactly one square.
  \( H[i,0] \lor \cdots \lor H[i,p(n)] \) for \( 0 \leq i \leq p(n) \).
  \( \neg H[i,j] \lor \neg H[i,j'] \) for \( 0 \leq i \leq p(n) \) and \( 0 \leq j < j' \leq p(n) \).

— At any time \( i \), each square contains exactly one symbol.
  \( S[i,j,0] \lor \cdots \lor S[i,j,v] \) for \( 0 \leq i \leq p(n) \) and \( 0 \leq j \leq p(n) \).
  \( \neg S[i,j,l] \lor \neg S[i,j,l'] \) for \( 0 \leq i \leq p(n) \), \( 0 \leq j \leq p(n) \) and \( 0 \leq l < l' \leq v \).
— At time 0, $M$ is in its initial configuration. Assume
\[ x = s_{l_1} \cdots s_{l_n}. \]

$Q[0, 0]$.  
$H[0, 0]$.  
$S[0, 0, l_1], \ldots, S[0, n-1, l_n]$.  
$S[0, j, 0]$ for $n \leq j \leq p(n)$.

— By time $p(n)$, $M$ has entered $q_1$ (accept)). (If $M$ halts in less than $p(n)$ steps, additional moves can be included in the transition function.)  

$Q[p(n), 1]$.  

187
Configuration at time $i \rightarrow$ configuration at time $i + 1$. Assume
$
\delta(q_k, s_l) = (q_k', s_{l'}, D), \text{ where } D = -1, 1.
$
If the head does not point to square $j$, symbol on $j$ is not changed from time $i$ to time $i + 1$.

\[
H[i, j] \lor \neg S[i, j, l] \lor S[i + 1, j, l]
\]
for $0 \leq i \leq p(n)$, $0 \leq j \leq p(n)$, and $0 \leq l \leq v$.

If the current state is $q_k$, the head points to square $j$ which contains symbol $s_l$, then changes are made accordingly.

\[
\neg H[i, j] \lor \neg Q[i, k] \lor \neg S[i, j, l] \lor H[i + 1, j + D],
\]
\[
\neg H[i, j] \lor \neg Q[i, k] \lor \neg S[i, j, l] \lor Q[i + 1, k'], \text{ and}
\]
\[
\neg H[i, j] \lor \neg Q[i, k] \lor \neg S[i, j, l] \lor S[i + 1, j, l'],
\]
for $0 \leq i \leq p(n)$, $0 \leq j \leq p(n)$, $0 \leq k \leq r$, and $0 \leq l \leq v$. 
Let $f_M(x)$ be the conjunction of all the clauses defined above. Then $x \in L(M)$ iff there is an accepting computation of $M$ on $x$ iff $f_M(x)$ is satisfiable. $f_M$ can be computed in polynomial time since $|f_M(x)| \leq (\text{number of clauses}) \times (\text{number of variables}) = O(p(n)^2) \times O(p(n)^2) = O(p(n)^4)$. So there is a polynomial reduction from any language in $\textbf{NP}$ to SAT. So SAT is $\textbf{NP}$-complete.