# CS312 Notes

September 17, 2024

## Contents

## 1 Theory Of Computation Introduction

The 3 componenets of problem solving

1. Unknowns
2. Data
3. Conditions

To solve a problem we need to find a way of determining the unknowns from given data such that conditions of the problem are satisfied.

The traditional areas of the theory of computation (TOC)

- Automata
  - Provide problem solving devices
- Computability
  - Provide framework that can characterize devices by their computing power
- Complexity
  - Provide framework to classify problems acording to time/space complexity of the toold used to solve them

## 1.1 Automata (Automaton)

- Abstration of computing devices
- How much memory can be used?
- What operations can be performed?

## 1.2 Computability

- Study different computing models and identify the most powerful ones
- Range of problems
- Problems can be undecidable or uncomputatble
  - The halting problem

## 1.3 Complexity

- Computing problems range from easy to hard; sorting is easier than scheduling
- Question
  - What makes some problems computationally hard or others easy?

## 1.4 Problem Abstration

Data

- Abstracted as a word in a given alphabet

Conditions

- Abstracted as a set of words called a language

Unknowns

- A boolean variable: true if a word is in the language or false other wise

### 1.4.1 Abstration of Data

- $\Sigma$: alphabet, a finite, nonempty set of symbols
- $\Sigma^*$: all words of a finite length built up using $\Sigma$
- Rules: (1) the empty word ($\epsilon$) is in $\Sigma^*$; (2) if w $\in \Sigma^*$ and a $\in \Sigma$, then aw $\in \Sigma^*$, and (3) nothing else is in $\Sigma^*$

Example: If $\Sigma = \{0,1\}$, then $\Sigma^* = \{\epsilon,0,1,00,01,10,11,000,001,010,011,\ldots\}$.

1. Valid C

```
int my_func() { return 1; };

int main() {
    int var = my_func(1,2,3,4,5,6,7);
    for (;;) {}
    // You cannot just simply change the syntax of a for loop
    for(;) {}
}
```

2. Invalid C++

```
int my_func() { return 1; };

int main() {
    int var = my_func(1,2,3,4,5,6,7);
    for (;;) {}
    // You cannot just simply change the syntax of a for loop
    for(;) {}
}
```

## 2 Finite Automata

### 2.1 Formal Language

- Some set of strings over a give alphabet
- How do you specify a language?
- How do you recognize strings in a language?
- How do you translate the language?

### 2.2 Abstraction of Problems

1. Data - word in a given alphabet
   - $\Sigma$ alphabet, a finite non-empty set of symbols
   - $\Sigma^*$ all words of finite length built-up using $\Sigma$
2. Conditions - Set of words called a language
   - Any subset $L \subseteq \Sigma^*$ is a formal language
3. Unknown - a boolean variable that is true, if word is in language; false, otherwise.
   - Given $w \in \Sigma^*$ and $L \subseteq \Sigma^*$, is $w \in L$?

## 2.3 Formal Definition

- Simplest computational model also referred to as a finite-state machine or finite automaton (FA)

- Representations: graphical, tabular, and mathmatical

- A finite automaton is a 5-tuple (Q,$\Sigma$,$\delta$,$q_0$,F), where Q is a finite set of states, $\Sigma$ is a finite set of symbols (alphabet), the transition function $\delta$ maps Q X $\Sigma$ to Q, $q_0 \in$ Q is the start (initial) state, and F $\subseteq$ Q is the set of accept (final) states

- Used to design embedded systems, or compilers

### 2.3.1 Example

If the machine is in a start state, where the initial state is an accept state, that means that our FA can accept an empty string $\epsilon$

## 2.4 DFA

Deterministic Finite Automata

## 2.5 Applications

- Parsers for compilers

- Pattern recognition

- Speech processing and OCR

- Financial planning and market prediction

## 2.6 FA Computation

- Automaton $M_1$ receives input symbols one-by-one (left to right)

- After reading each symbol, $M_1$ moves from one state to another along the transition that has that symbol as its label

- When $M_1$ reads the last symbol of the input, it produces the output: accept if $M_1$ is in an accept state, or reject if $M_1$ is not in an accept state

## 2.7 Language Recognition

- If L is the set of all strings that an FA M accepts, we say that L is the language of the machine M and write L(M) = L

- An automaton may accept several strings, but it always recognizes only one language

- If a machine accepts no strings, it still recognizes one language, namely the empty language 0

The machines are recognizing words in the language
Any given automaton only recognizes specifically one language

## 2.8 Formal Definition of Acceptance

- LEt $M = (Q,\Sigma,\delta,q_0,F)$ be an FA and $w = a_1a_2\dots a_n$ be a string over $\Sigma$. We say M accepts w if a sequence of states $r_0r_1\dots r_n$ exist in Q such that

  - $r_0 = q_0$ (where machine starts)

  - $\delta(r_i,a_{i+1}) = r_{i+1}$, $i=0,1,\dots,n-1$,(transitions based on $\delta$)

  - $r_n \in F$ (input accepted)

## 2.9 Regular Languages

- We say that FA recognizes the language L if L = {w | M accepts w}
- A language is called a **regular** language, if there exists an FA that recognizes it
- Q: how do you design/build an FA

## 2.10 FA Design Approach

1. Identify finite pieces of information you need, i.e., the states (possibilities)
2. Identify the condition (or alphabet) to change from one state to another
3. Idenitfy the starting and final/accept states
4. Add missing transitions

## 2.11 Example

Let $M_1 = (Q, \Sigma, \delta, q_1, F)$, $Q = \{q_1, q_2, q_3\}$, $\Sigma = \{0,1\}$, and $F = \{q_2\}$. Let's define a transition functoin $\delta$ for $M_1$ and then draw the resulting (graph-based) **state transition diagram** for $M_1$

DFA, this table is $Q \times \Sigma \rightarrow Q$
$q_1$ is the start state
$q_2$ is the accept

| | 0 | 1 |
|---|---|---|
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_3$ | $q_2$ |
| $q_3$ | $q_2$ | $q_2$ |

->, node distance=3cm, every state/.style=thick, fill=gray!10, initial text=,

[state, initial] (q1) $q_1$; [state, accepting, right of=q1] (q2) $q_2$; [state, right of=q2] (q3) $q_3$; (q1) edge[loop above] node0 (q1) (q1) edge[above] node1 (q2) (q2) edge[loop above] node1 (q2) (q2) edge[bend left, above] node0 (q3) (q3) edge[bend left, below] node0, 1 (q2);

### 2.11.1 Notes on Example

L($M_1$) = ?
L($M_1$) = A
A = {w | w contains at least one 1 AND an event number of 0's following the last 1}

## 2.12 Example 2

$\delta$ Q X $\Sigma \rightarrow$ Q

| | 0 | 1 |
|---|---|---|
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_1$ | $q_2$ |

[state, initial] (q1) $q_1$; [state, accepting, right of=q1] (q2) $q_2$; (q1) edge[loop above] node0 (q1) (q1) edge[bend left, above] node1 (q2) (q2) edge[loop above] node1 (q2) (q2) edge[bend left, below] node0 (q1)

L($M_2$) = B = { w | w ends in a 1 }

### 2.12.1 Expanstion on Above $M_3$

[state, initial, accepting] (q1) $q_1$; [state, right of=q1] (q2) $q_2$; (q1) edge[loop above] node0 (q1) (q1) edge[bend left, above] node1 (q2) (q2) edge[loop above] node1 (q2) (q2) edge[bend left, below] node0 (q1)

Language of $M_3$ = C = { w | w ends in a 0 OR w is empty }

### 2.12.2 What does this give us?

If we flip the accept and initial state, we generate the complement of the machine (flip the meaning)

### 2.13 Last DFA Example

Q={s,$q_1$,$q_2$,$r_1$,$r_2$}
Σ={a,b}
F = {$q_1$,$r_1$}

Δ chart

| | a | b |
|---|---|---|
| s | $q_1$ | $r_1$ |
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_1$ | $q_2$ |
| $r_1$ | $r_2$ | $r_1$ |
| $r_2$ | $r_2$ | $r_1$ |

[state, initial] (s) $s$; [state, accepting, right of=s] (q1) $q_1$; [state, accepting, left of=s] (r1) $r_1$; [state, below of =q1] (q2) $q_2$; [state, below of =r1] (r2) $r_2$; (s) edge[] nodea (q1) (s) edge[] nodeb (r1) (q1) edge[loop above] nodea (q1)
{ w | starts with 'a' AND ends with 'a' }

# 3 Regular Languages

Let A and B be languages
Union: A B = { x | x ∈ A ∨ x ∈ B }
Concatenation: A ˆ B = { xy | x ∈ A ∧ y ∈ B }
Star: $A^*$ = { $x_1x_2\ldots x_k$ | k >= 0 ∧ $x_i$ ∈ A, 0 <= i <= k }

#### 3.0.1 Is $\epsilon$ always a member of $A^*$ regarless of the language A?

Yes

#### 3.0.2 What is another name for the language of A ˆ $A^*$?

$A^+$

### 3.0.3 Closures of Regular Languages

Theorem: Class of regular languages is closed under intersection. (Proof: Use cross-product construction of states)
Theorem: Class of regular languages is closed under complementation (Proof: swap accept/non-accept states and show FA recognizes the complement)

## 3.1 Nondeterminism

NFA or nondeterministic finite automata

- Every stop of a FA computation follows in a unique way from the proceeding step; a deterministic computation

- Nondeterministic computation - choices exist for the next state; a nondeterministic FA (NFA)

- Ways to introduct nondeterminism

  - more choices for next state (zero, one, many)

  - State may change to another state without reading any symbol

### 3.1.1 Formal Definition

a 5-tuple (Q, $\Sigma$, $\delta$, $q_0$, F), where Q is a finite set of states, $\Sigma$ is a finite set of symbols (alphabet), the transition function $\delta$ maps Q x $\Sigma$ $\{\epsilon\}$ to P(Q), $q_0 \in Q$ is the start (initial) state, and $F \subseteq\ Q$ is the set of accept (final) states.

Notice that the range of the transition function $\delta$ for an NFA is the power set of Q P(Q)

### 3.1.2 Formal Definition of Acceptance (NFA)

Let N k (Q, $\Sigma$, $\delta$, $q_0$, F) be an NFA and w $= y_1y_2 \ldots y_n$ be a string over $\Sigma_\epsilon = \Sigma\epsilon$. We say N accepts w if a sequence of states $r_0$,$r_1$,...,$r_m$ exist in Q such that

1. $r_0 = q_0$

2. $\delta(r_i, y_{i+1}) = r_{i+1}$ for $i = 0,1,\ldots,m-1$

3. $r_m \in F$

### 3.1.3 NFA Motivation

- For some problems they are much easier to construct than a DFA

- NFA may actually be smaller than a DFA that performs the same task; but NFA computation is usually more expensive

- Every NFA can be converted into an equivalent DFA (in theory, every NFA has an equivalent DFA t orecognize the same language)

- NFAs can be used to show that regular languages are closed under union, concatenation, and star operations

Epsilon transitions happen without reading anything, allowing you to go either direction

### 3.1.4 DFA/NFA Equivalence

Let N = ($Q$, $\Sigma$, $\delta$, $q_0$, F) be the NFA that recognizes the language A and construct the DFA M that also recognizes A. Define M = ($Q'$, $\Sigma$, $\delta'$, $q_0'$, $F'$).

$E(R) = R \{q \in Q \mid r_1 \in R, r_2, \ldots, r_k \in Q, r_{i+1} \in \delta(r_i, \epsilon), r_k = q\}$

## 4 Nonregular Languages

R is a regular expression if

1. a for some $a \in \Sigma$
2. $\epsilon$ (language contains only the empty string)
3. 0 (language has no strings)
4. ($R_1$ $R_2$), where $R_1$, $R_2$ are regular expressions
5. ($R_1$ ˆ $R_2$), where $R_1$, $R_2$ are regular expressions
6. ${R_1}^*$, where $R_1$ is a regular expression

### 4.1 Language Example

Is $B = \{0^n 1^n | n \geq 0\}$ a regular language?

- No because one single machine cannot possible match the infinte states

#### 4.1.1 Warning

Just because a language might seem to require unbounded (infinite) memory t orecognize it - it could still be regular

Suppose you have the following two languages: C = {w | w has an equal number of 0s and 1s} and D = {w | w has an equal number of 01 and 10 substrings}

2nd is regular?

### 4.2 Example

Language D = {w|w has an equal number of 01 and 10 substrings}

We can generally create a regular language if the constraints are ordering Generally non-regular if we have to do some sort of counting without ordering

## 5 Pumping Lemma

- All strings in a language can be pumped if they are at least as long as a certain value called the pumping length

- Another interpretation: every string contains a section that can be repeated any number of itmes with the resulting string remaining in the language

### 5.1 Example

sqrt(sqrt(sqrt(. . . sqrt(x). . . )))

### 5.2 Lemma

If A is a regular language, then there exists a number p (the pumping length) where, if s is any string in A of length of at least p, then s may be divided into three pices, s = xyz subject too

If you take any string out of the language, at least length p, then I can take that string, cut it into pieces, 3 pieces, prefix x, suffix z,

1. $\forall$ i $\geq$ 0, $\mathrm{xy^iz} \in$ A

    - p is an integer

    - You can also remove all y's, 0 y's

2. $|y| > 0$

   - The actual substring must have some chars
   - x and z can both be empty strings

3. $|xy| \leq p$

   - the length of x+y cannot be bigger than p

## 5.3 Pumping Lemma Proof Construction

- Let M = $(Q,\Sigma,\delta,q_1,F)$ be a DFA that recognizes the language A. Assign a pumping length p to the number of states of M.
  - P (pumping length) is the number of states in M, which is finite
  - If you have finite number of states (p), and it is way bigger than p, then we will have to loop
- Show that any string $s \in A$, $|s| \geq p$ may be broken into xyz satisfying the three PL conditions
  -
- If there are no strings in A of length at least p, then the PL is true because all three conditions hold for all strings of length at least p (if there are NO such strings)

## 5.4 Proof

Recognizes A and let p be the size of Q, let $s = s_1 s_2 \ldots s_n$ be a string over $\Sigma$ with $n \geq p$ and $r = r_1 r_2 \ldots r_{n+1}$ be the sequence of states encountered while processing

### 5.4.1 Example

We know that n+1 $\geq$ p+1, why?

- because n $\geq$ p

The among the first p+1 elements in the sequence r. . . , there must be a repeating state, say $r_j$,$r_k$ what principle is this based on?

- Pigeon hole principle

Let $r_k$ be the recurring state among the first p+1 states in the sequence starting with $r_1$, so k $\leq$ p+1. Let x = $s_1s_2\ldots s_{j\text{-}1}$, y = $s_js_{j+1}\ldots s_{k\text{-}1}$, and z = $s_ks_{k+1}\ldots s_n$

So X takes M from $r_1$ to $r_j$, Y takes M from $r_j$ to $r_j$ and Z takes M from $r_j$ to $r_{n+1}$; recall that $r_j$ == $r_k$ and that $r_{n+1}$ is an accept state
Therefore:

- M must accept $xy^iz$, for i $\geq$ 0 (**Condition 1**)

- Since j $\neq$ k then |y| > 0 (**Condition 2**)

- Since k $\leq$ p + 1 then |xy| $\leq$ p (**Condition 3**)

## 5.5 Technique

- Assume the language is regular, and assume a contradiction

- PL guarantees existence of a pumping length p such that all strings of length p or greater (in A) can be pumped

- Find s $\in$ A, |s| $\geq$ p that cannot be pumped; consider all the ways of dividing s int ox,y,z and show that for each division, at least one of the

PL conditions fail to hold

- pumping length = number of states in a minimalist machine

### 5.5.1 Example

B = $\{0^n 1^n \mid n \geq 0\}$ is not regular

Assume B is regular and let p be the pumping length for B. Choose s = $0^p 1^p \in B$ so that clearly s $\geq$ p. By the PL, we can partition s = xyz such that for all i $\geq$ 0, $xy^iz \in B$. Let's consider three possible cases for the contents of substring y

Think about what the string is, in our case s = 00. . . 011. . . 1, where our length is now 2p, p 0's followed by p 1's.

| x | y | z |
|---|---|---|
| 000 | 0011 | 111 |

1. Options
   (a) Y consists of only 0's
      - x | y | z
      - 0000|. . . 0|111.11
      - Then S' = xyyz
      - Then S' has more 0's than 1's, therefore it is no longer in the language
      - Hence, xyyz $\notin$ B, a violation of condition 1 of the PL
      - Y consists only of 0's. Then S' = xyyz has more 0's than 1's since $|y| > 0$ by condition 2. Hence, xyyz $\notin$ B. This is a violation of condition 1 of the PL.

(b) Y consists of only 0's

- Y consists only of 1's. Then S' = xyyz has more 1's than 0's since $|y| > 0$ by condition 2. Hence, xyyz $\notin$ B. This is a violation of condition 1 of the PL.

(c) Y consists of 0's and 1's

- Y consists of 0's and 1's. Then s' = xyyz may have the same number of 0's and 1's, but they are out of order (i.e some 1's occur before 0's), and so s' = xyyz $\notin$ B. This is a violation of condition 1 of the PL.

## 5.6 How might the proof above be simplified using condition 3 of the PL to constrain the contents of the substring y?

According to condition 3, we must have $|xy| <= p$.

We picked s = 0000000|1111111, both of length p
xy is already p length, therefore the language only contains 0's.
y could only contain 0's and we have contradiction in case 1 for all y

### 5.6.1 Example

i = 2
New string is xz

## 5.7 Conditions

- If a language is regular, one should be able to take any string out of that language, partition it into 3 parts XYZ, with the Y componenet having something in it, then you can pump that string

- $\forall$ i $\geq$ 0, $XY^iZ$ has to be in the language, if i = 0 we are pumping down, if i > 1 we are pumping up

- We only have to find one i that breaks the conditions, because the condition says $\forall$ i's

- if i = 1, then we are not pumping, that is just the same string

- When i $\geq$ 2 then we are pumping up

- When i 0 then we are pumping down

### 5.8 Example Again

Use the PL to prove that the language E = $\{0^i1^j \mid i > j\}$ is not regular.

Assume E is regular and let p be the pumping length for E. Choose s = $0^{p+1}1^p \in$ E so that clearly s > p. By the PL, we can partition s = xyz such that for all i >= 0, $xy^iz \in$ E. By condition 3 of the pumping lemma, we must have $|xy| \leq p$. Therefore, y must contain only 0's. Is it possible for s' = xyyz to be in the language? Adding an additional copy of y increases the number of 0's which does not violate the constraints of the language. But setting i = 0 does violate the condition since we would now have $0^p$ and $1^p$, or more generally i <= j. Subsequently s' would $\notin$ E, given this contradiction of conditoin 1 of the PL, we can conclude E is not regular.

Start with condition 3 of the PL to determine the contents of substring y and see if you can get a violation of Condition 1 of the PL for any choice of y.

### 5.9 Another example

#### 5.9.1 Language 1

$L_1 = \{a^iba^j \mid 0 \leq i < j\}$

- ba

- Assume $L_1$ is regular with pumping length p

- Choose s = $a^pba^{p+1}$
- State cond 1, s = xyz $\forall$ i $\geq$ 0 xy'z $\in$ $L_1$
- Use condition 3 (|xy| $\leq$ p), and condition 2 (|y| > 0)
- y contains only a's
- Pumping up violates the constraint i < j, because xyyz would result in i is now greater than j, which means j is no longer strictly greater than i
- Therefore s' $\notin$ $L_1$, a's before b are greater than or equal to those after b

## 6 Context Free Languages

We have shown that L = $\{0^n1^n \mid n \geq 0\}$ cannot be specified by a FA or regular expression; Context-Free Grammars (or CFGs) provide a more powerful way to specify languages. A CFG is a 4-tuple (V, $\Sigma$, R, S), where

- V is a finite set of symbols (variable or nonterminals)
- $\Sigma$ is a finite set of symbols disjoint from V (terminals)
- R is a finite set of specification rules of the form *lhs* $\rightarrow$ *rhs*, *lhs* $\in$ V, *rhs* $\in$ (V $\Sigma$), and S $\in$ V is the start variable

### 6.1 Example

CFG $G_1$ has the following specification rules

$$A \rightarrow 0A1$$

$$A \rightarrow B$$

$$B \rightarrow \#$$

The start variable for $G_1$ is A.
What are the nonterminals?

- A, B

What are the terminals?

- 0, 1, #

## 6.2 Language Specification

1. Write down start variable; *lhs* of first spec rule

2. Find variable that is written down and a rule whose *lhs* is that variable; replace the written down variable with the *rhs* of that rule

3. Repeat Step 2 until no variables remain in string

### 6.2.1 Example

Use the CFG $G_1$ (above) to derive the string 000#111. Show derivation and corresponding parse tree

## 6.3 Direct Derivation

If u,v,w $\in$ (V $\Sigma$)$^*$, i.e., are strings of variables and terminals, and A $\rightarrow$ w $\in$ R is a grammar rule, then we say that uAv yields uwv or uAv $\Rightarrow$ uwv. Alternatively, uwv is directly derived from uAv using the rule A $\rightarrow$ w.

## 6.4 Derivation

$$S \rightarrow aSb$$

$$S \rightarrow SS$$

$$S \rightarrow \epsilon$$

## 6.5 Applications

- Compiler design and implementation
- Programming language specificatoin
- Scanner, parsers, and code generators

## 6.6 Design Techniques

1. CFG Design Technique
    - Many CFGs are unions of simpler CFGs
    - Combination involves putting all the rules together and adding the new rules

- $s \rightarrow s_1|s_2|\ldots|s_k$
  - where the variables $s_i$, $1 \leq i \leq k$, are the start variables of the invidivual grammars and the s is a new variable

```
int a = 5;

return a + a a;
```

### 6.6.1 Design a Language Example

CFG G = $(\{S,B\}, \{a,b\}, \{S \rightarrow aSB|B|\epsilon, B \rightarrow bB|b\} S)$

$\{a^n b^m \mid n <= m\}$

## 6.7 Ambiguous Grammar

Consider the CFG $G_5$ that has the rules $E \rightarrow E + E \mid E * E \mid (E) \mid a$.

For this out by parsing left to right

### 6.7.1 Derivation Order

It is possible for 2 derivations to produce hte same derivation becuase htey differ in the order

Leftmost derivation, replace the leftmost nonterminal first
Rightmost Derivation, replace the rightmost nonterminal at each step

### 6.7.2 Example

$\Sigma = \{a,b,c\}$
$A = \{a^i b^j c^k \mid i = j \vee j = k \ \ (i,j,k \geq 0)\}$

Rules

- $\mathrm{S} \rightarrow \mathrm{E_{ab}C} \mid \mathrm{AE_{bc}}$
- $\mathrm{E_{ab}} \rightarrow \mathrm{aE_{ab}b} \mid \epsilon$