cse103-notes

Dec 09, 2020

Contents:

CHAPTER 1

Introduction

1.1 Algorithmic Problems

- **Decision problems: yes/no**
  - Given a graph G, is G connected?
  - Given a graph G, is G 3-colorable?
  - Given a boolean formula phi, is phi satisfiable?
  - Given a natural number M, is M a prime?
- **Function problems**
  - Given 2 integers M and N, find the GCD
  - Given a network of cities and distances, find the length of minimal length
- **Enumeration problems**
  - Given a natural number, find its prime factors
  - Given a boolean formula phi, find all/how many satisfying assortments
- **Optimization problems**
  - Given a network of cities and distances, find a tour of minimal length

In general, decision problems are the easiest to solve - and any problem can be expressed as a series of decision problems

1.2 Problem Difficulties

- Unsolvable
- **Solvable**

- Untractable (NP runtime)
- Tractable (polynomial runtime)

CHAPTER 2

Sets

10/5/2020 - 10/12/2020

Set: A collection of distinguishable objects, with unordered, non-repeating elements

Two sets are equal if their elements are equal

2.1 Notation

- $z \in S$ - element member
- $S = \{1, 2, 3\}$ - complete denotation
- $\emptyset$ - empty set
- Z - integers
- R - real numbers
- N - natural numbers (no 0)
- Q - rational numbers
- $A \subseteq B$ **- all elements in A are in B (subset)**
  - $\forall x, x \in A \rightarrow x \in B$
- $A \cap B$ **- all elements in A and B (intersection)**
  - $\{x : x \in A \wedge x \in B\}$
- $A \cup B$ **- all elements in A or B (union)**
  - $\{x : x \in A \vee x \in B\}$
- $A - B$ **- all elements in A but not B (difference)**
  - $\{x : x \in A \wedge x \notin B\}$
- $A \Delta B$ **- all elements in exactly one set (symmetric difference)**

  - $\{x : x \in (A - B) \vee x \in (B - A)\}$

Given a universe of discourse Ω:

- $\bar{A} = \Omega - A$ - all elements not in A (complement)

Demorgan's Laws:

- $\overline{A \cap B} = \bar{A} \cup \bar{B}$
- $\overline{A \cup B} = \bar{A} \cap \bar{B}$

Definition:

- $\{x \in N | \frac{x}{2} \in N\}$ - even number definition by restricted comprehension
- $\{x | P(x)\}$ - unrestricted comprehension

Power Set: The set of all of a set's subsets

2.2 Russel's Paradox

Extraordinary Sets: All sets that include themselves as an element (ex. the set of everything that is not a teacup)

Ordinary Sets: All sets that don't have themselves as a member

Paradox: Does the set of all ordinary sets contain itself?

This is a paradox - which means that the set of all sets cannot exist

2.3 Relations

Ex. $<$ - the set of all ordered pairs (a, b) s.t. $a < b$

- **Cartesian product of 2 sets A, B:** $\{(a, b) | a \in A \wedge b \in B\}$
  - e.g. $\{c, d\} \times \{1, 2, 3\} = \{(c, 1), (c, 2), (c, 3), (d, 1), (d, 2), (d, 3)\}$

Binary Relation

A binary relation on A and B is defined by some subset of $A \times B$ - some examples of binary relations on $N \times N$ are:

- $=$: $\{(1, 1), (2, 2), ...\}$
- $<$: $\{(1, 2), (1, 3), (2, 3), ...\}$

This can be denoted $a < b \rightarrow (a, b) \in <$

2.3.1 Properties

Note: For this notation, the symbol $\sim$ represents an arbitrary relation. This can also be denoted R, but that doesn't look good in LaTeX.

- **Reflexive:** $x \sim x$
  - ex: =, <=
- **Symmetric:** $x \sim y \implies y \sim x$

  - ex: =, but *not* < or <=

- **Transitive:** $x \sim y \land y \sim z \implies x \sim z$
  - ex: =, <, <=
  - but not: $\{(x, y)|x, y \in N \land x = y - 1\}$

If a relation has all 3 properties, it is called an *equivalence relation*

2.4 Functions

A function is a binary relation defined on the cross product of the domain and the codomain.

Given 2 sets A and B, a function f is a binary relation on $A \times B$ s.t. for all x in A, there exists exactly one y in B s.t. $(x, y) \in f$

Notation: $f : A \to B$

2.5 Graph

An undirected graph can be represented as a tuple $G = (V, E)$ where V and E are sets (vertices, edges), where $E \subseteq \{\{x, y\}|x, y \in V \land x \neq y\}$ (set of sets of two vertices)

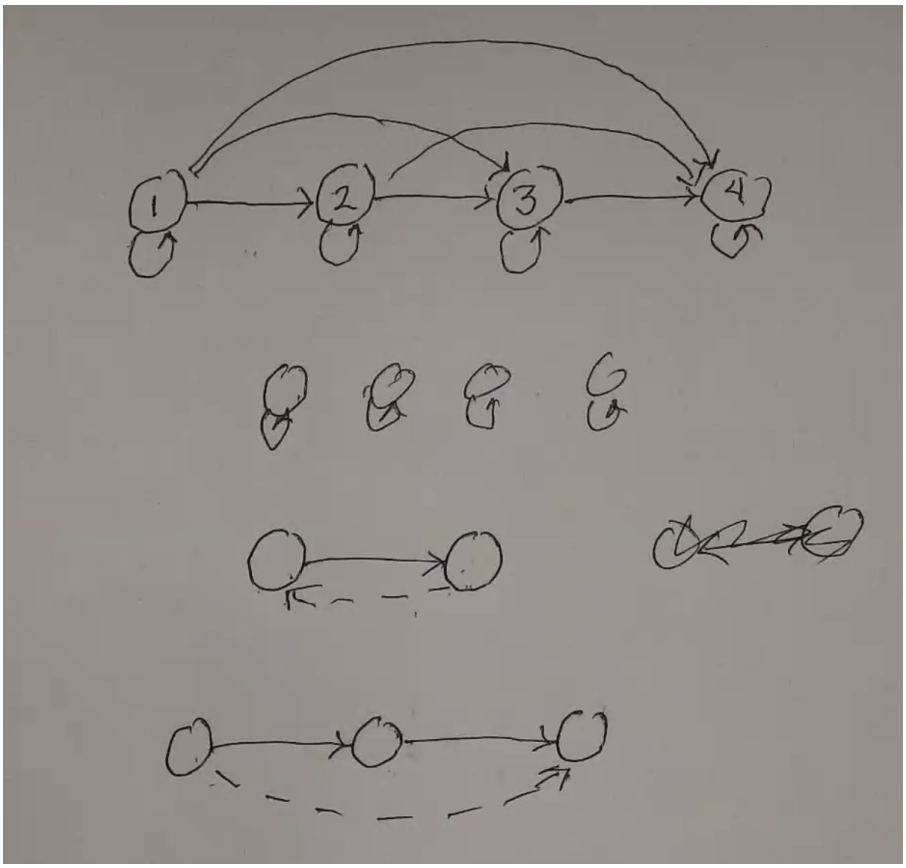
A digraph is similar, but E must use ordered pairs rather than sets to indicate the direction of the edge, and an edge can go to the same vertex. $E \subseteq \{(x, y)|x, y \in V \times V\}$

Ex:

```
(1) --- (2)      V = {1, 2, 3}
 |               E = {{1, 2}, {1, 3}}
(3)
```

```
(1) --> (2)      V = {1, 2, 3}
 ^               E = {(1, 2), (3, 1)}
(3)
```

You can use digraphs to represent relations:



- Reflexive: every vertex has a self-loop
- Symmetric: all arrows must be bi-directional
- Transitive: the "jump" edge must exist (bottom of drawing)

2.6 Strings

Alphabet: Any finite set (usually notated Σ)

A string over Σ is a finite length sequence of elements from Σ

The *length* of a string x $|x|$ is the number of symbols in x

An *empty string* is a unique string of length 0, notated ϵ

A symbol with an exponent (e.g. a^x) is repeated that many times

---

Note: $a^0 = \epsilon$ and $a^{m+1} = a^m a$

Σ^* is the *set of all strings* over the alphabet Σ

Note: $\emptyset^* = \{\epsilon\}$

2.7 Propositional Logic

A *proposition* is a statement that is true or false.

Connectives

- not: $\neg$
- and: $\wedge$
- or: $\vee$
- implies: $\implies$
- iff: $\iff$

Constants

- 0, 1 (false, true)

Variables

- $X = \{P, Q, R, ...\}$

Series of propositions/operations can be modeled using *truth tables* (which I am not going to write here, because tables in RST suck)

Tautology: A proposition that is true in any given state of the universe

Contradiction: A proposition that is false in any given state of the universe

Valid Argument: The conjunction of all givens and the negation of the output is false in all states.

e.g. given the argument:

```
P -> Q
P
---
Q
```

$(P \implies Q) \wedge P \wedge (\neg Q)$ is always false.

2.7.1 Useful Tautologies

```
P -> Q      P -> Q      P -> Q      P or Q
P           not Q       Q -> R      not P
---         ---         ---         ---
Q           not P       P -> R      Q

                        P
P           P and Q     Q
---         ---         ---
P or Q      Q           P and Q
```

2.8 Cardinality

For finite sets, the *cardinality* of a set is the number of elements in the set.

Denoted, given a set A, $|A|$. $|\emptyset| = 0$

For infinite sets:

- **countably infinite: all elements in the set can be put in a 1-to-1 correspondence with natural numbers, or a list of the elem**
  - e.g. natural numbers ($f(m) = m$)
  - even integers ($f(m) = 2m$)
  - integers ($f(m) = (-1)^m \lfloor \frac{m}{2} \rfloor$)
  - strings over the alphabet $\{0, 1\}$
  - rational numbers (map $N \times N$ onto $\frac{p}{q}$ by making a list)
  - the union of any two countable sets
  - strings over any finite alphabet
- **uncountably infinite**
  - e.g. real numbers (diagonalization)
  - set of all languages

Note: Let's go back and look at Σ^* - all strings over an alphabet.

- Σ^* is countably infinite, but
- $P(\Sigma^*)$ is not!

You can use diagonalization to prove that $P(\Sigma^*)$ is uncountably infinite using the same binary argument as real numbers - use 1 to indicate an element's presence in the subset, and 0 to indicate its not

In general, the power set of any set has greater cardinality than that set.

CHAPTER 3

Proofs

10/12/2020 -

Definitions:

- **Statements to be proved or already proved:**
  - Theorem
  - Proposition
  - Lemma
  - Corollary (immediately follows from proof of theorem)
- **Statements assumed to be true:**
  - Axiom
  - Postulate

3.1 Induction

Weak Induction: Prove base cases, then prove inductive steps.

Ex. Prove that the sum of the first *m* odd numbers $S(m)$ is equal to m^2.

- B.S. m = 1, $1 = 1^2$.
- **I.S. Assume** $S(m) = m^2$ **for some** $m \geq 1$**.**
  - $S(m+1) = S(m) + 2(m+1) - 1$
  - $= m^2 + 2m + 2 - 1$
  - $= m^2 + 2m + 1$
  - $= (m+1)^2$.

Strong Induction: Prove all cases less than current case implies current case.

Ex. Prove $\forall m > 1, m$ is divisible by a prime.

Consider any $m > 1$:

1. m is a prime. QED.
2. **m is not a prime:** $m = a * b$**, where** $a, b < m$ **and** $a, b > 1$
    1. By IH, a is divsible by a prime P
    2. $P|a$ and $a|m$, so $P|m$. QED.

3.2 Contrapositive

Rather than proving $p \implies q$, prove $\neg q \implies \neg p$.

Ex. p^2 is even $\implies p$ is even

3.3 Contradiction

Prove that the opposite statement causes a contradiction.

Ex. Prove that $\sqrt{2}$ is irrational.

1. Assume that $\sqrt{2} = \frac{p}{q}, p, q \in N$ with no common factor
2. $p^2 = 2q^2$
3. p^2 is even
4. p is even
5. $p = 2k$
6. $\sqrt{2} = \frac{2k}{q}$
7. $q^2 = \frac{(2k)^2}{2} = 2k^2$
8. q^2 is even
9. q is even
10. $2|p$ and $2|q$, so by contradiction $\sqrt{2}$ is not rational.

3.4 Pidgeonhole Principle

Given *n* containers and *m* items, if m > n, at least one container must have more than one item in it.

Ex. For any *m*, there exists a multiple of *m* that is a sequence of 1s followed by a sequence of 0s in binary.

- For some m, consider the sequence `1 11 111 ... 111...11` where the last item is of length m+1
- if you divide by m, there are only *m* remainders possible: [0..m-1]
- There are m+1 items in the sequence but only m possible remainders, so two items in the sequence will have the same remainder
- Let $a = k_1 m + r, b = k_2 m + r, a > b$

- $\therefore a - b = (k_1 - k_2)m$
- $a - b$ is a multiple of m and of form `111...00`

CHAPTER 4

DFA

Deterministic Finite Automaton (DFA) is a structure:

$M = (Q, \Sigma, \delta, s, F)$, where:

- Q is a finite set of states
- Σ is a finite set of symbols (input alphabet)
- $\delta : Q \times \Sigma \to Q$ **is the transition function**
  - given a current state and input, use delta to find the new state
- $s = q_0 \in Q$ is the start state
- $F \subseteq Q$ are the accept/final states

Defns:

- $x \in \Sigma^*$ if *accepted* by M if M stops in F
- $L(M)$ is the *language of machine M* when it consists of all strings the machine accepts
- $L \subseteq \Sigma^*$ is *regular* if there is a DFA M s.t. $L = L(M)$ (some dfa recognizes it)

DFAs can be represented using a graph/flowchart thing. Final states are represented by double-bordered nodes.

Note: An example of a non-regular language is $\{0^m 1^m | m \geq 1\}$. (e.g. 01, 0011, 000111, etc)

Note: Any finite language is regular, since it can be represented by just a really huge DFA!

4.1 Extended Transition Function

$\hat{\delta} : Q \times \Sigma^* \to Q$

Rather than a transition from one state to the next given a symbol, this function maps a starting state and a string to the result after processing what whole string.

Inductice Defn

- For all $q \in Q, x \in \Sigma^*, a \in \Sigma$:
- $\hat{\delta}(q, \epsilon) = q$
  - the extended transition function of any state and the empty string is the same state
- $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$
  - just the normal transition plus one more

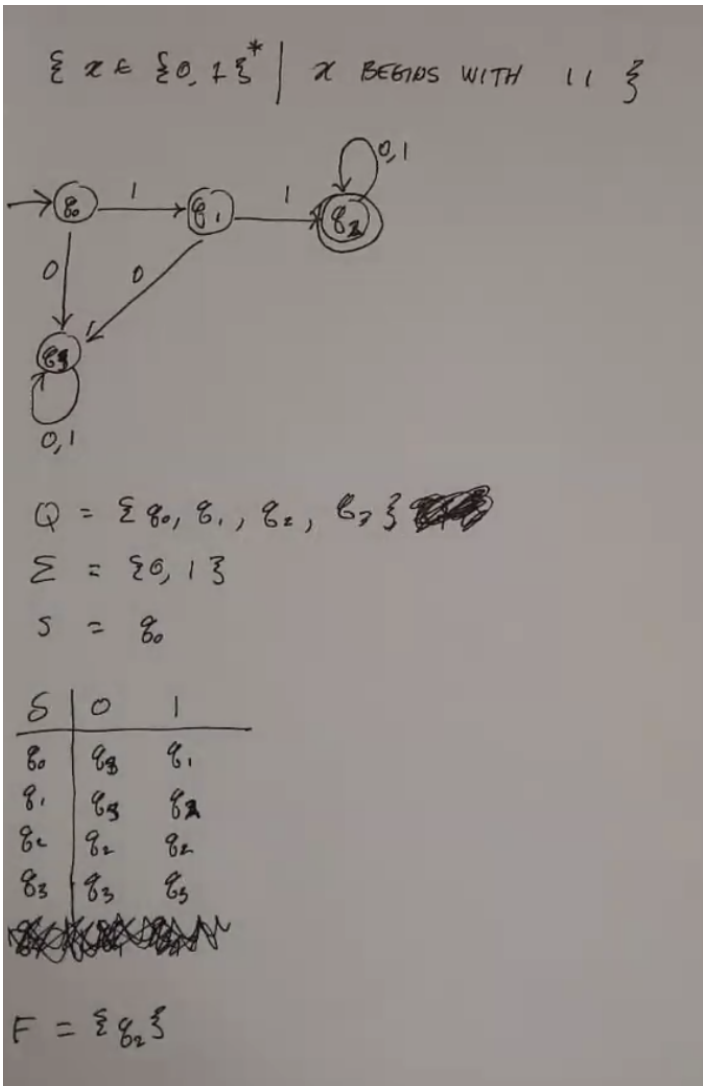
4.2 Thms

- **Given a regular language, the complement of that language is also regular. (let the accept states of the DFA be the rejects**
  - $x \in L(M) \rightarrow \hat{\delta}(s, x) \notin F$
  - $\hat{\delta}(s, x) \in Q - F = F_c$
  - $\hat{\delta}(s, x) \in F_c$
  - $x \in L(M_c)$

4.3 Examples

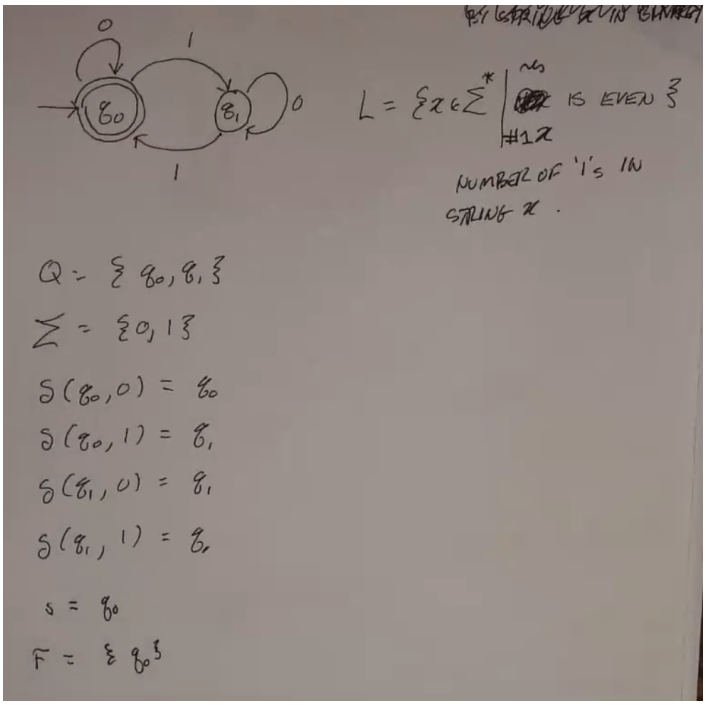
4.3.1 Two Ones

This image shows a DFA that accepts any string starting with two ones.



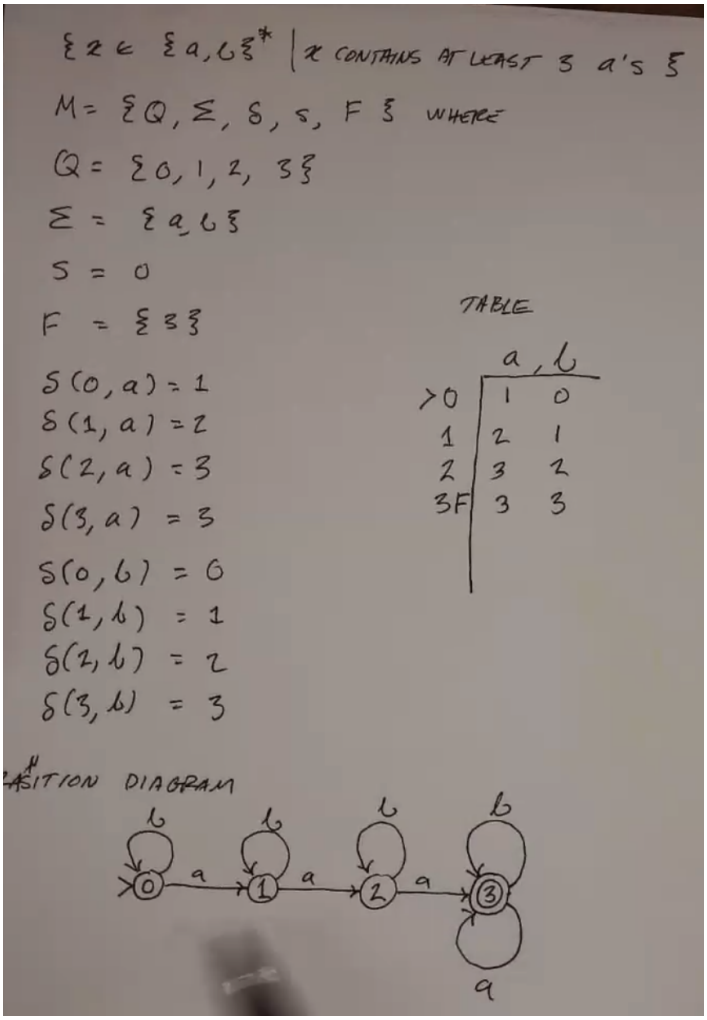
4.3.2 Even Ones

Consider a DFA that accepts any string with an even number of ones. ($\Sigma = \{0, 1\}$)



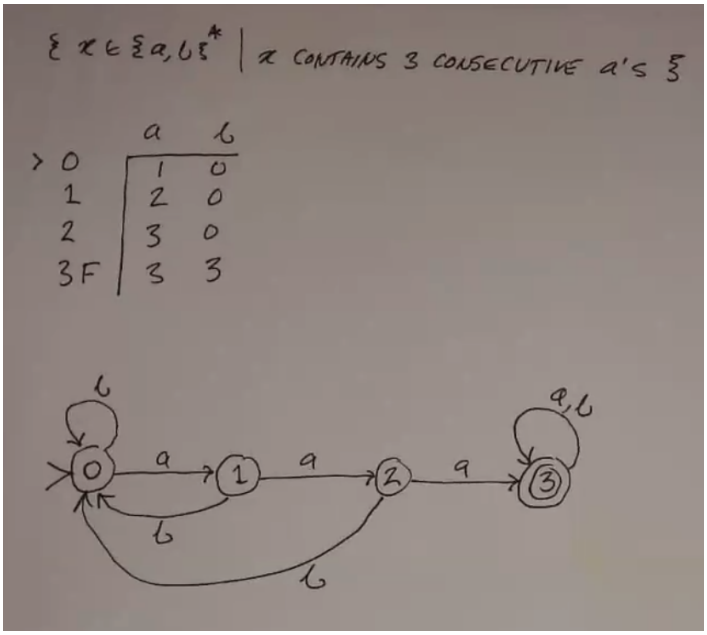
4.3.3 3 As

Consider a DFA that accepts any string that contains at least 3 As. ($\Sigma = \{a, b\}$)



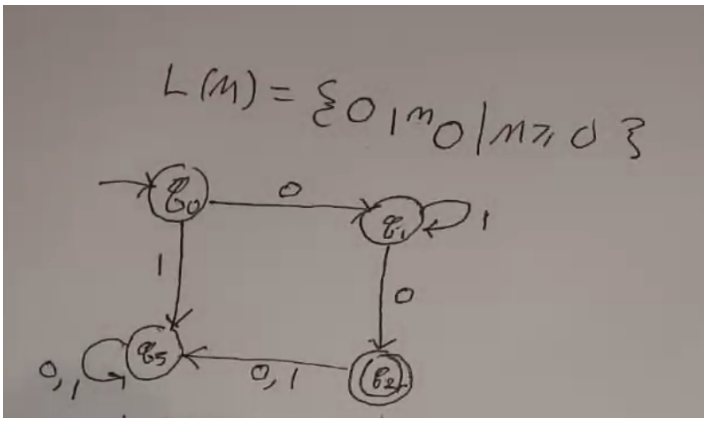
4.3.4 3 Consec As

Consider a DFA that accepts any string that contains at least 3 *consecutive* As. ($\Sigma = \{a, b\}$)



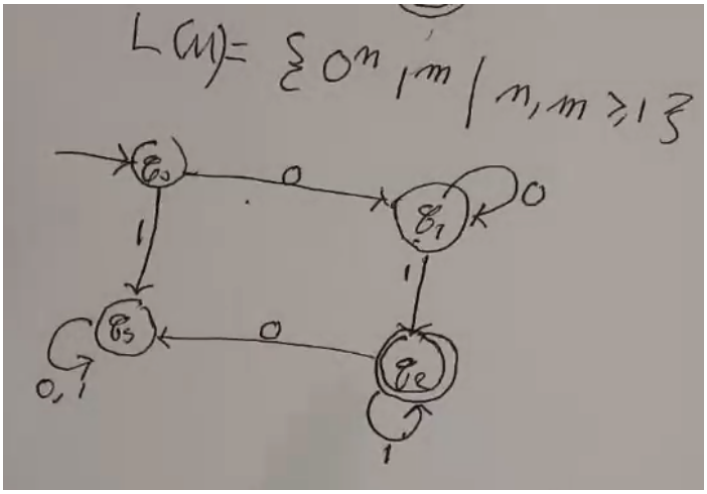
4.3.5 0m0

Design a DFA for the language $L(M) = \{01^n0 | n \geq 0\}$.

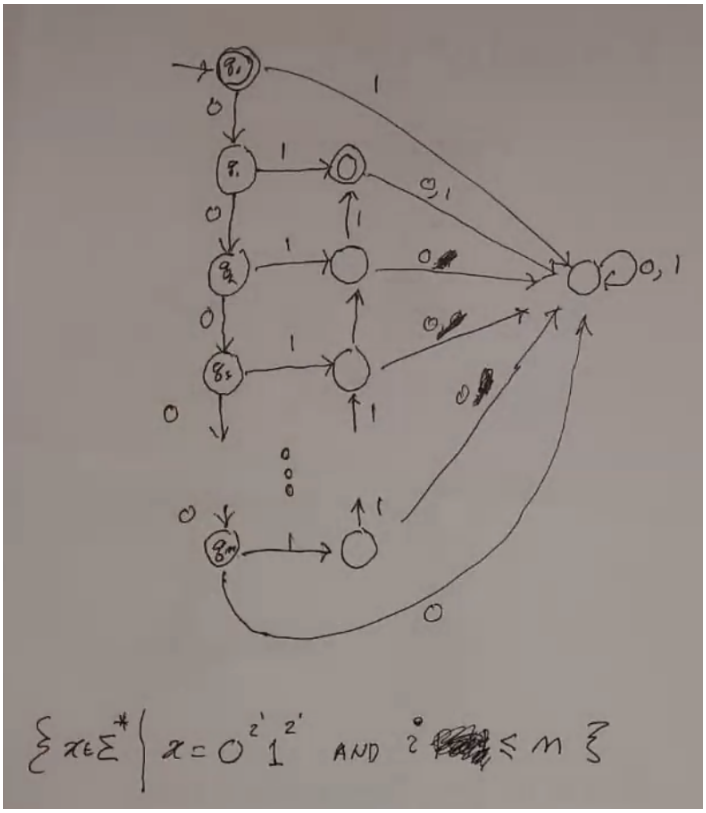


4.3.6 00011

Design a DFA for the language $L(M) = \{0^n1^m | n, m \geq 1\}$.

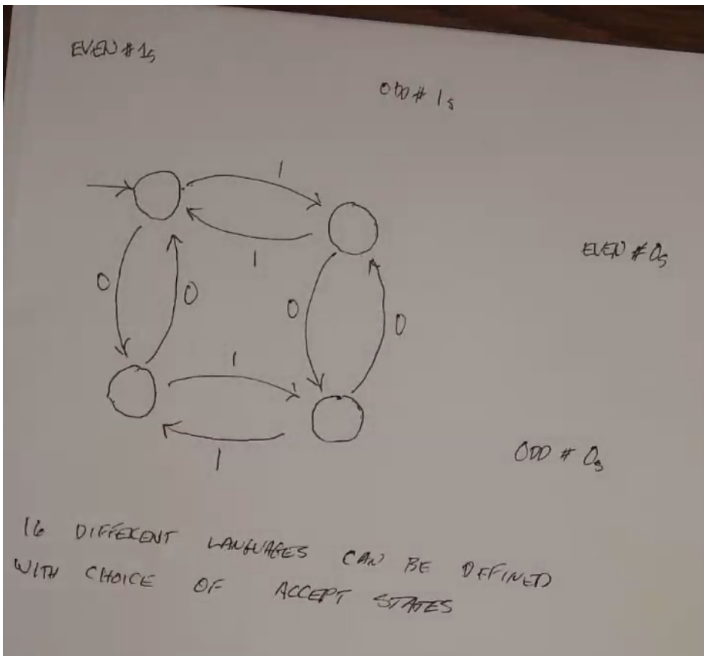


Note: However, $L(M) = \{0^n 1^n | n \geq 1\}$ does not exist. Such a DFA would have to be infinitely large:



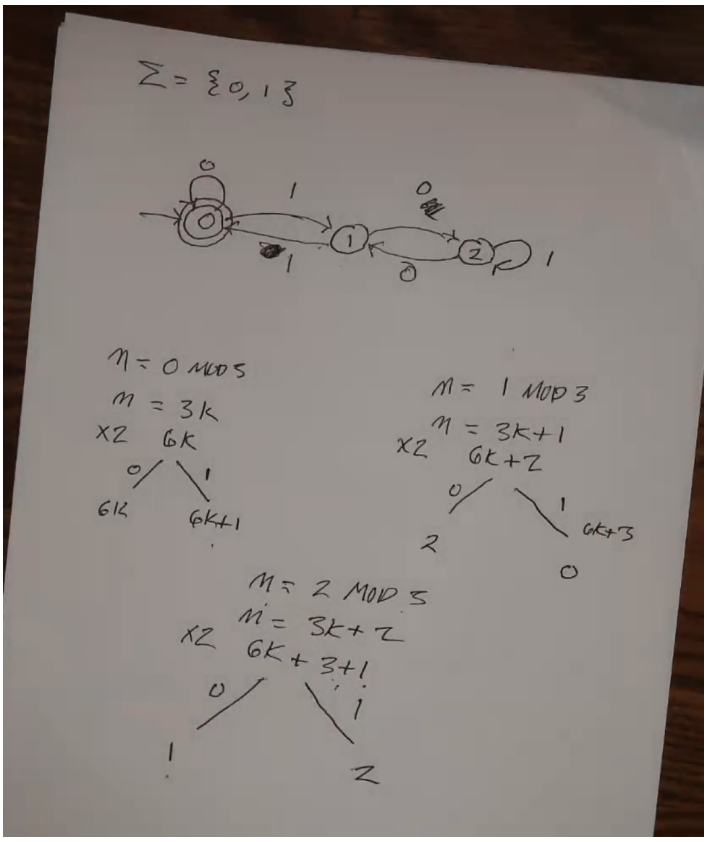
4.3.7 Odds/Evens

This DFA tracks how many 1s and 0s are found in a string. 16 different languages can be defined with choices of accept states:



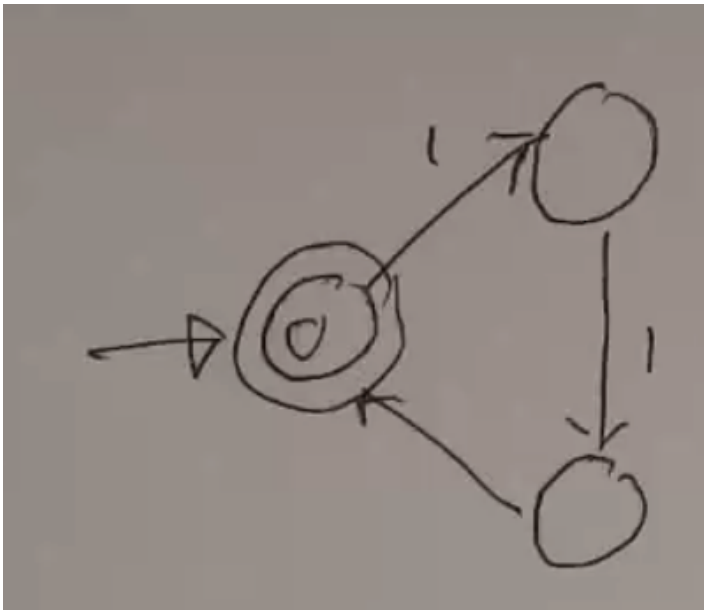
4.3.8 Div3

Design a DFA for the language of all binary numbers that are divisible by 3



4.3.9 Len3

Strings of length multiple of 3.



4.4 Intersection

aka Product Construction

Thm: If languages A and B are regular, then $A \cap B$ is regular.

- there exists $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ with $L(M_1) = A$
- there exists $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ with $L(M_2) = B$
- since A and B are regular, we can build a DFA M_3 s.t. $L(M_3) = A \cap B$.
- let $M_3 = (Q_3, \Sigma, \delta_3, s_3, F_3)$
- $Q_3 = Q_1 \times Q_2 = \{(p, q) | p \in Q_1, q \in Q_2\}$
- $F_3 = F_1 \times F_2 = \{(p, q) | p \in F_1, q \in F_2\}$
- $s_3 = (s_1, s_2)$
- $\delta_3 : Q_3 \times \Sigma \to Q_3$
  - $\delta_3((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$
- **extended transition function:**
  - $\hat{\delta_3}((p, q), \epsilon) = (p, q)$
  - $\hat{\delta_3}((p, q), xa) = \delta_3(\hat{\delta_3}((p, q), x), a)$

Pf: $L(M_3) = L(M_1) \cap L(M_2)$

THM $L(M_3) = L(M_1) \cap L(M_2)$

FOR ALL $x \in L(M_3)$

$x \in L(M_3) \iff \hat{\delta}_3(s_3, x) \in F_3$ DEF OF ACC.

$\iff \hat{\delta}_3((s_1, s_2), x) \in F_1 \times F_2$ DEF OF s_3 AND F_3

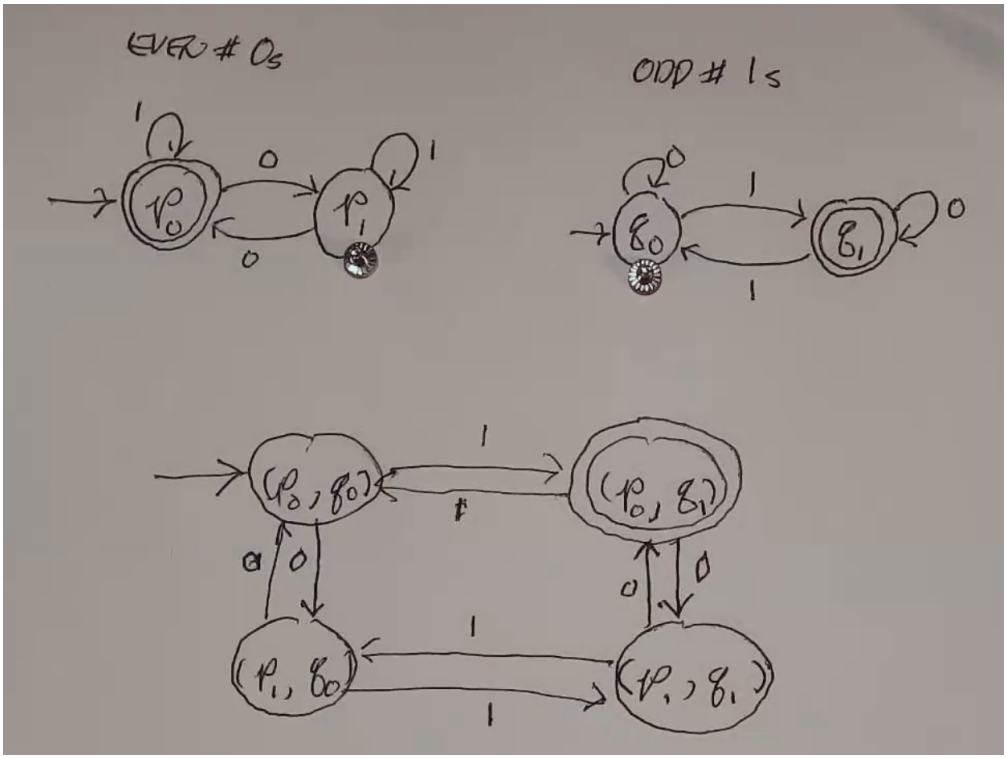
$\iff (\hat{\delta}_1(s_1, x), \hat{\delta}_2(s_2, x)) \in F_1 \times F_2$ LEMMA

$\iff \hat{\delta}_1(s_1, x) \in F_1$ AND $\hat{\delta}_2(s_2, x) \in F_2$ DEF OF CART. PROD.

$\iff x \in L(M_1)$ AND $x \in L(M_2)$ DEF OF ACC

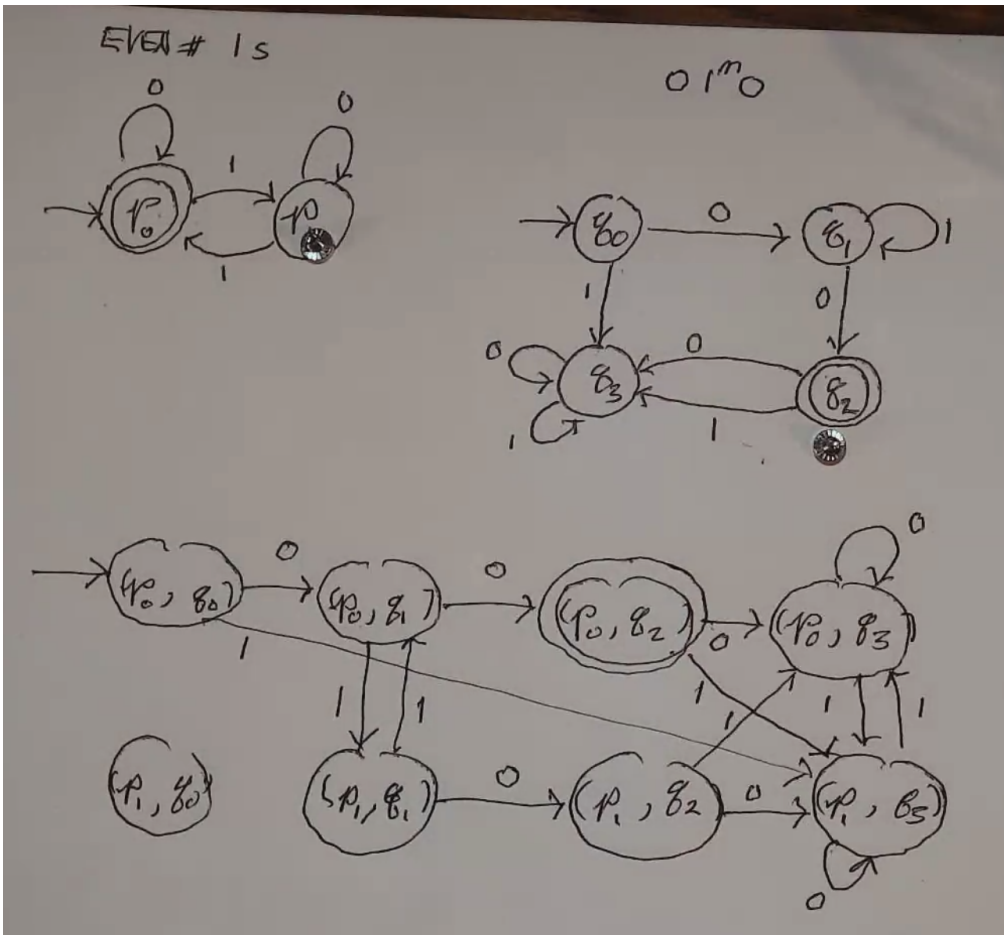
$\iff x \in (L(M_1) \cap L(M_2))$ DEF OF SET INTERSECTION

Ex: Given two machines that accept an even number of 0s and odd number of 1s, the intersection can be constructed as such:



Ex: Even number of 1s and form 01^m0

Note that there is no way into (p_1, q_0).



4.5 Union

Thm: If languages A and B are regular, then $A \cup B$ is regular.

- A is regular $\implies$ $\neg A$ is regular
- B is regular $\implies$ $\neg B$ is regular
- $\neg A$ and $\neg B$ regular $\implies$ $\neg A \cap \neg B$ regular
- $\neg A \cap \neg B$ regular implies $\neg(\neg A \cap \neg B)$ regular
- $\neg(\neg A \cap \neg B)$ regular implies $A \cup B$ regular (demorgans).

CHAPTER 5

NFA

$N = (Q, \Sigma, \delta, s, F)$, where:

- Q is a finite set of states
- Σ is a finite set of symbols (input alphabet)
- $\delta : Q \times \Sigma \to P(Q)$ **is the transition function**
  - the transition function *relation* on $(Q \times \Sigma) \times Q$
  - $\delta(p, a) \in P(Q)$
  - the set of all states N can move to from p in one step under the symbol a
  - $p \to^a q$ in $q \in \delta(p, a)$
  - $\delta(p, a)$ can be empty set
- $s \in Q$ is the start state
- $F \subseteq Q$ are the accept/final states

Extended Transition Function

- $\delta^*(q, \epsilon) = \{q\}$
- $\delta^*(q, a) = \delta(q, a)$
- $\delta^*(q, xa) = \bigcup_{p \in \delta^*(q,x)} \delta(q, a)$

Accept

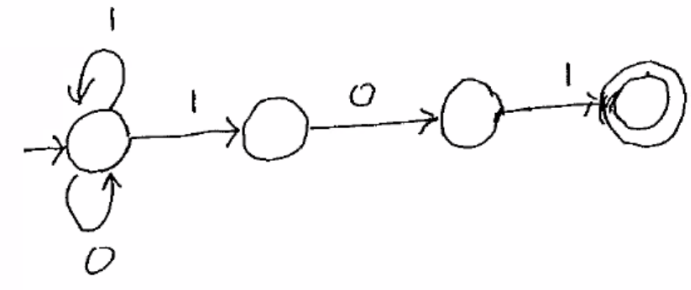
$w \in \Sigma^*$ is accepted if $\delta^*(s, w) \cap F \neq \emptyset$

(a string will be accepted if it's *possible* to accept the string)

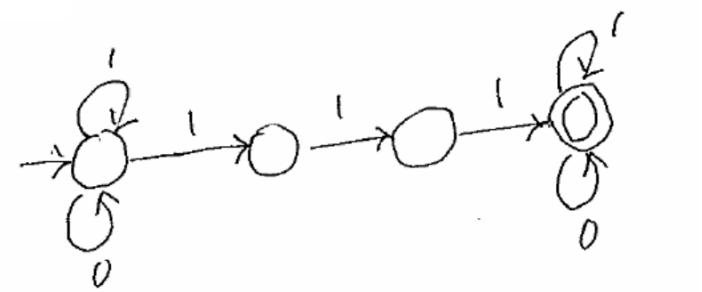
Note: It is trivial to prove that every DFA is also a NFA (if the output if the DFA transition is put into a set).

5.1 Examples

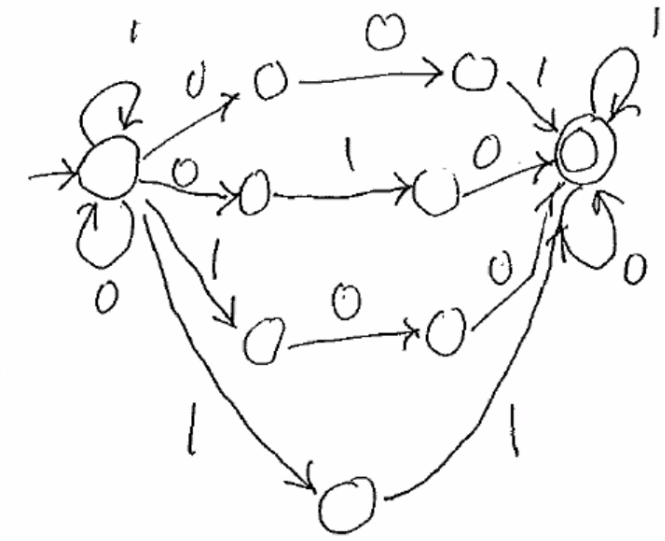
String ends with 101.



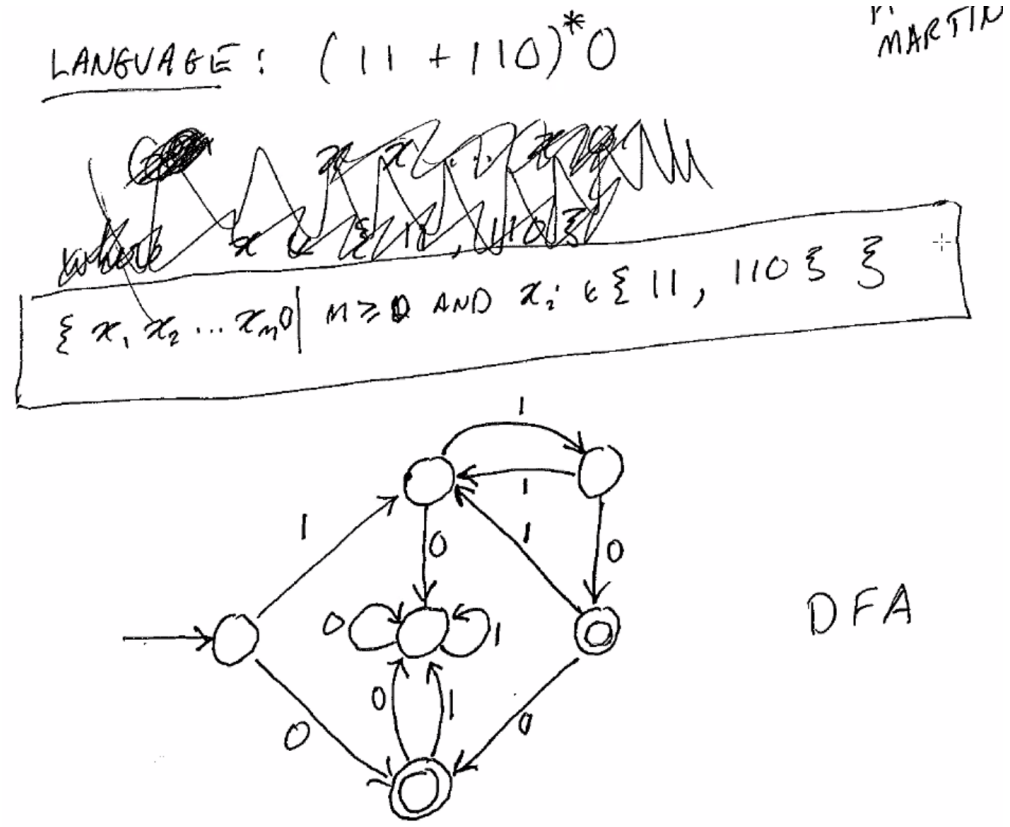
String contains 111.



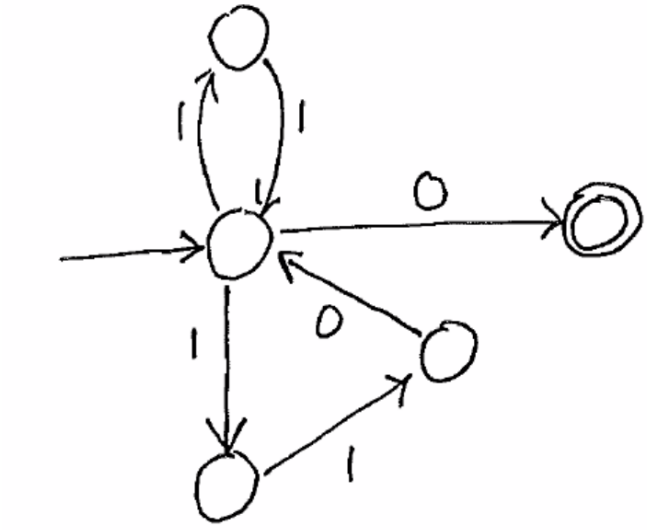
String contains 001 or 010 or 100 or 11.



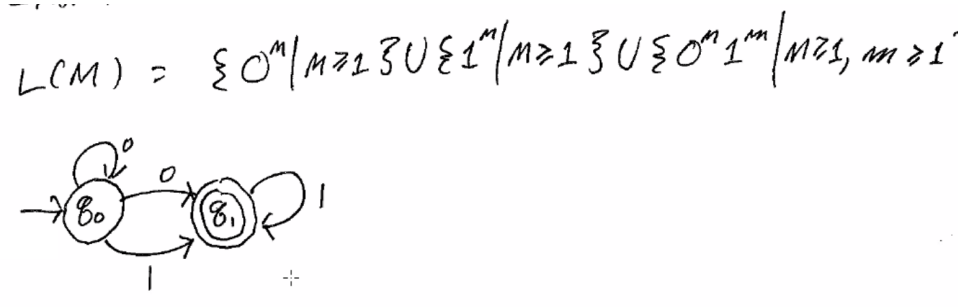
NFAs are easier to design.



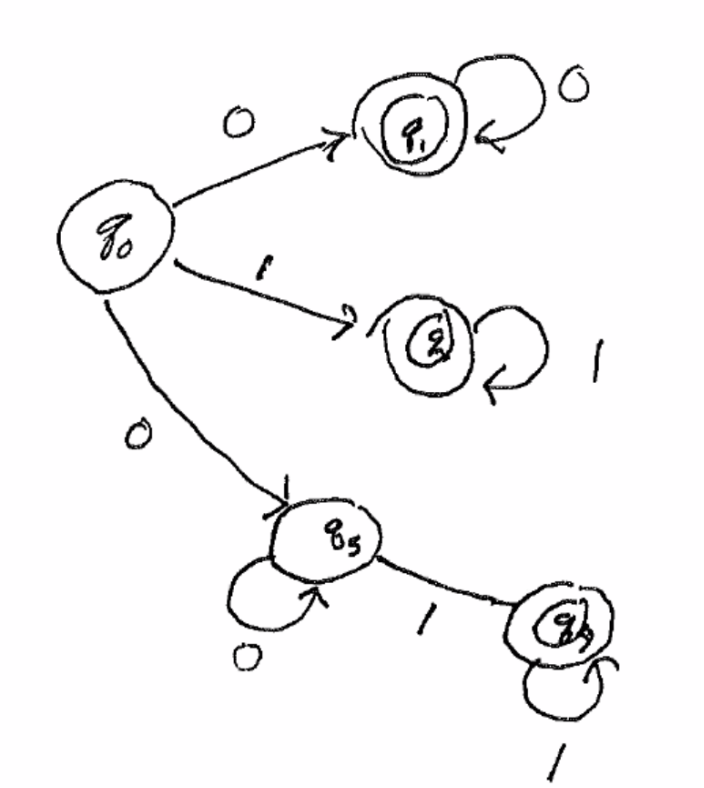
as an NFA:



And a weird one:



or



5.2 Subset Construction

aka the Rabin-Scott theorem

Given a NFA $N = (Q_N, \Sigma, \delta_N, s_n, F_n)$, it is possible to construct a DFA:

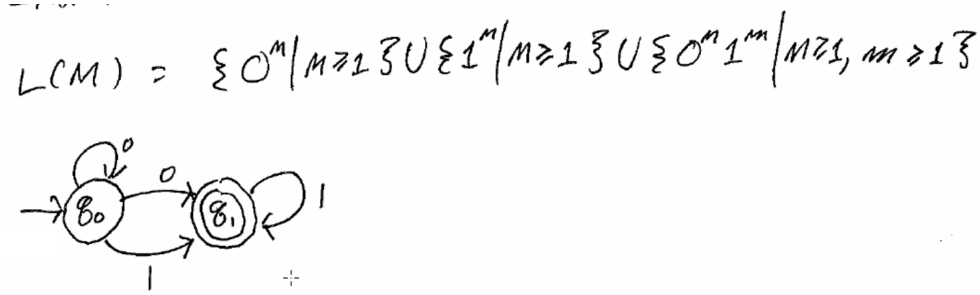
$M = (Q_D, \Sigma, \delta_D, s_D, F_D)$

- $Q_D = P(Q_N)$
- $s_D = \{s_N\}$
- $F_D = \{P \subseteq Q_N | P \cap F_N \neq \emptyset\}$
- $\delta_D : Q_D \times \Sigma \to Q_D$ **(i.e.** $P(Q_N) \times \Sigma \to P(Q_N)$**)**
  - $\delta_D(P, a) = \bigcup_{p \in P} \delta_N(p, a)$ for $P \subseteq Q_N$

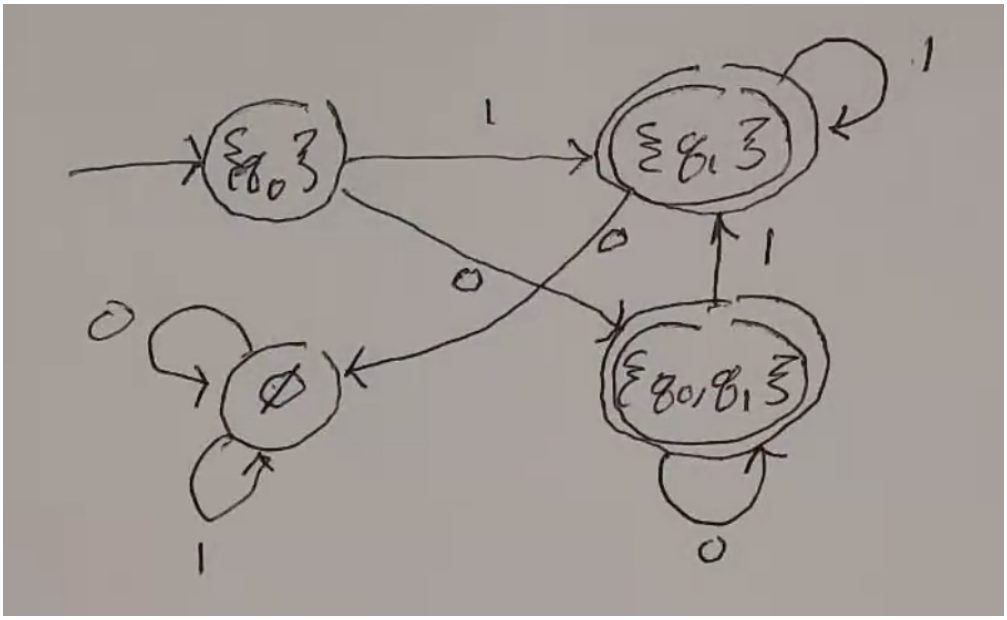
TLDR: the states of the DFA are the sets of states possible at any given point in the NFA.

Ex.

Given this language and NFA:



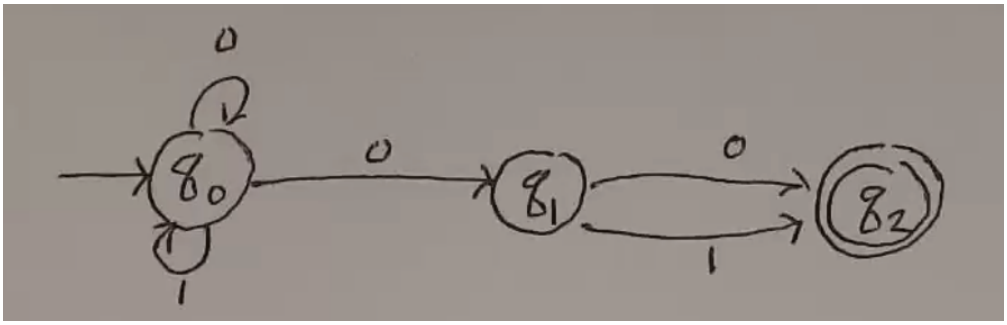
the DFA looks like:



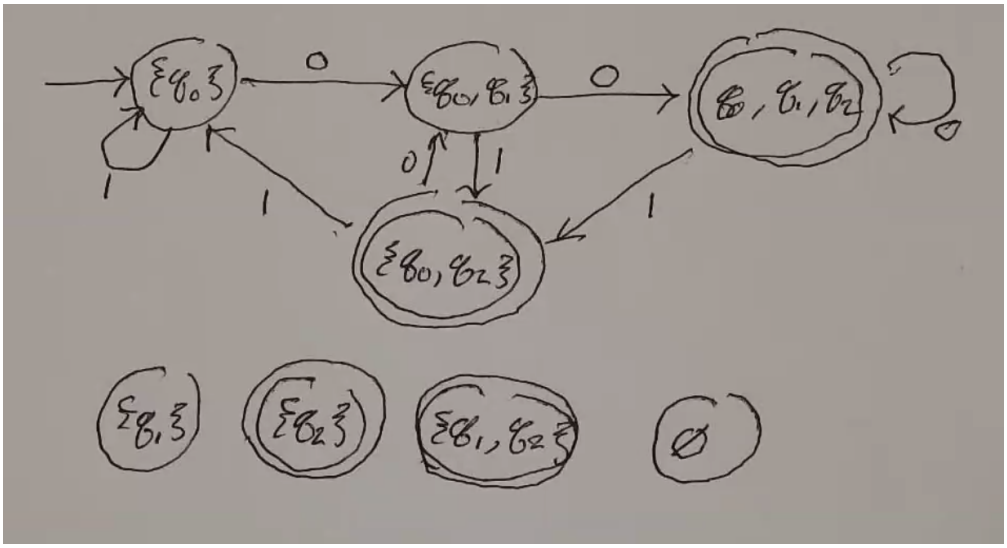
Ex 2.

A language of 0s and 1s, where the second-last symbol is a 0.

NFA:

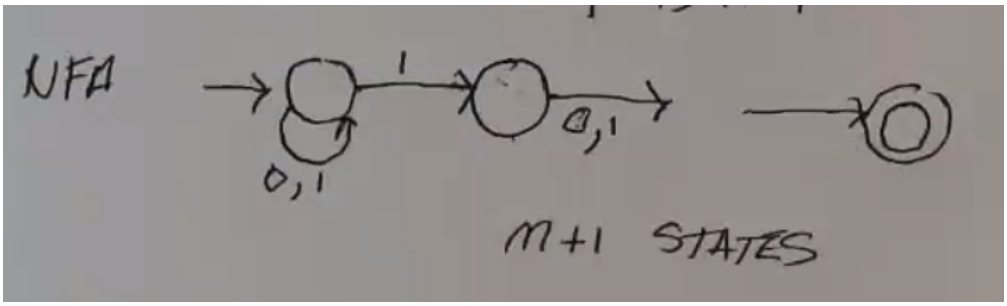


DFA:



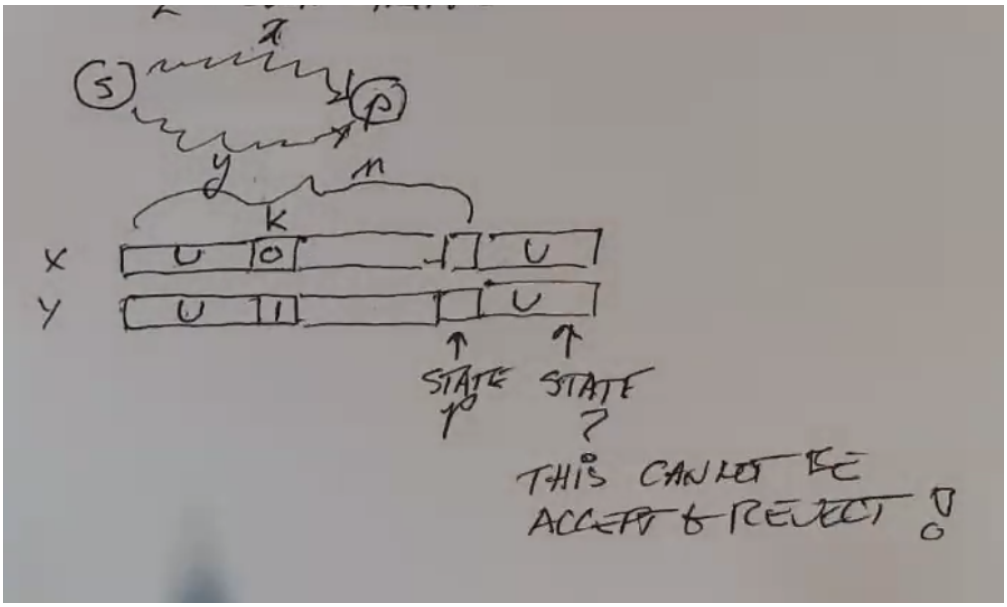
Ex 3.

The family of languages $L_n = \{w \in \{0,1\}^* | \text{ the nth position from end is } 1\}$



The DFA for L_n cannot have less than 2^n states. Pf:

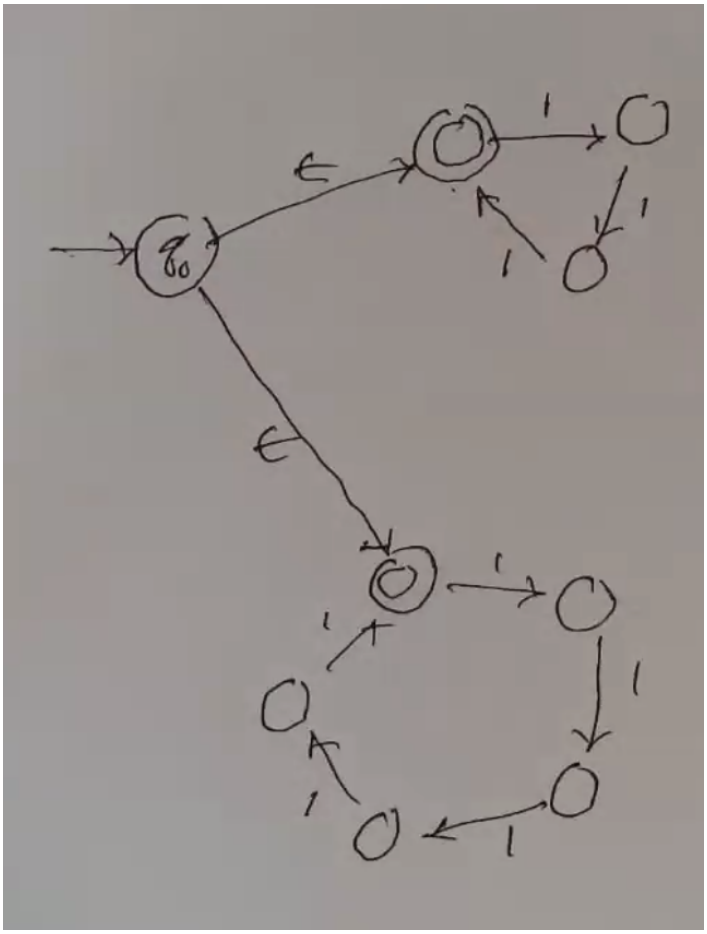
- Assume there is some DFA that recognizes L_n that has less than 2^n states.
- Consider strings of length *n*. There are 2^n such strings.
- Consider two arbitrary strings *x* and *y* that both end in a state *p*.
- Consider the first position those two strings differ *k*.
- Call the same identical part of the string *u*.
- If we redefine *x* and *y* such that *k* is the start and *u* is moved to the end, we get two strings with different characters *n* from the end, only one of which is accepted
- Contradiction!



5.3 Epsilon Moves

Epsilon moves allow us to jump to another state without taking any input.

Consider $L = \{1^n | n \text{ is a multiple of 3 or 5}\}$



While epsilon moves make NFAs easier to design, any NFA with epsilon moves can be rewritten as one without.

Formally:

$M = (Q, \Sigma, \delta, s, F)$, where:

- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to P(Q)$ **is the transition function**
  - $\delta(q, \epsilon) \to P \subseteq P(Q)$ does not move the scan head

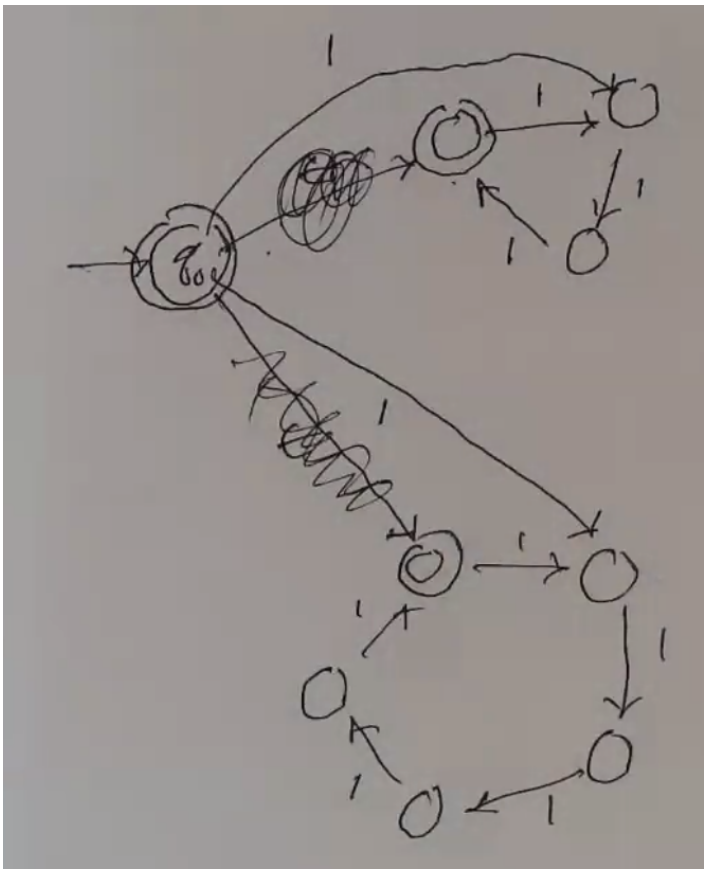
Thm:

For every ϵ-NFA there is an NFA $\hat{M}$ s.t. $L(M) = L(\hat{M})$.

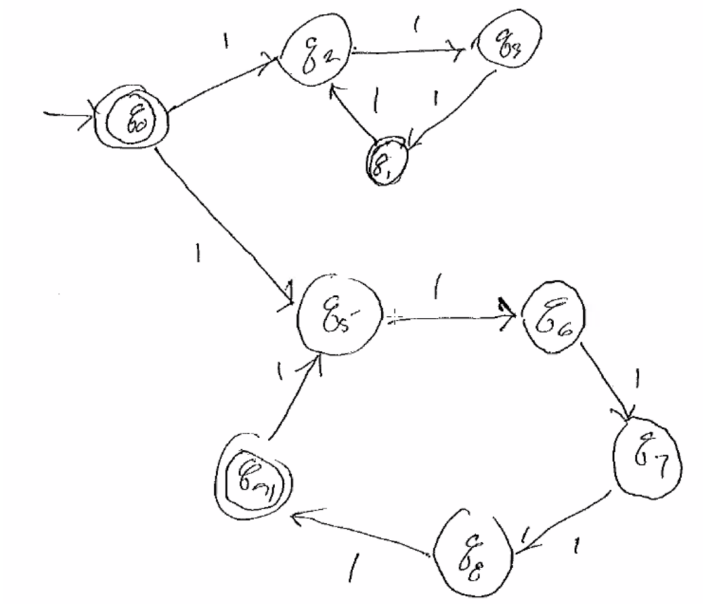
- $\hat{M} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{s}, \hat{F})$, where:
- $\hat{Q} = Q$
- $\hat{s} = s$
- $\hat{\delta} : \hat{Q} \times \Sigma \to P(\hat{Q})$
  - $\hat{\delta}(p, a) = \{q|$ **there are states** p_1, p_2 **s.t.:**
    - $p_1 \in E(p)$
    - $p_2 \in \delta(p_1, a)$
    - $q \in E(p_2)\}$
  - where $E(p)$ is the set of all states reachable from *p* using only epsilon moves.

- new accept states are states that can reach accept states with epsilon moves

Ex:



(cleaner view)

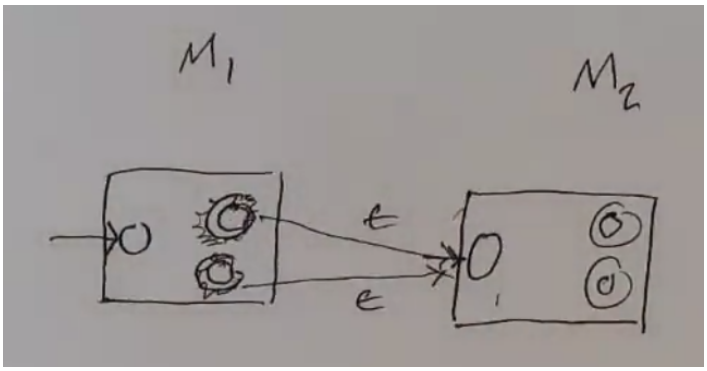


5.4 Closure

Regular languages are closed under:

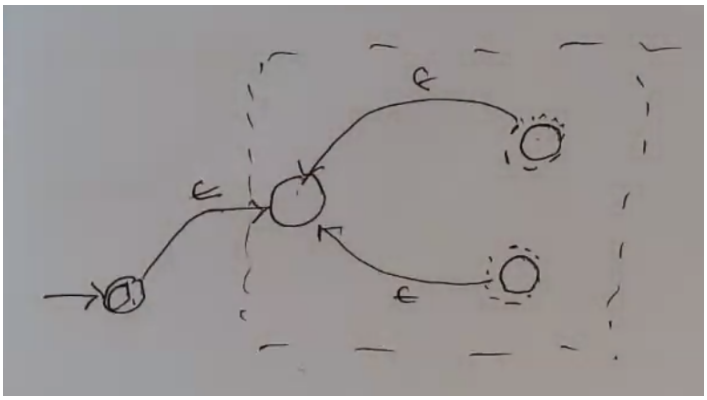
- complement
- union
- intersection
- difference
- concatenation
- kleene star
- quotient
- reversal

5.4.1 Concatenation



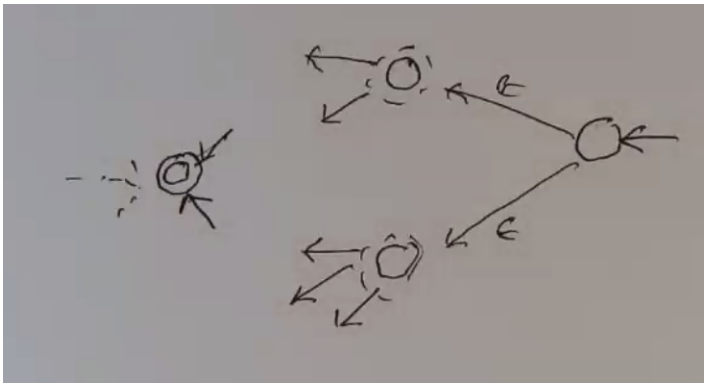
5.4.2 Kleene Star

The set of all strings that can be generated by concatenating 0 or more strings in a set of strings.



5.4.3 Reversal

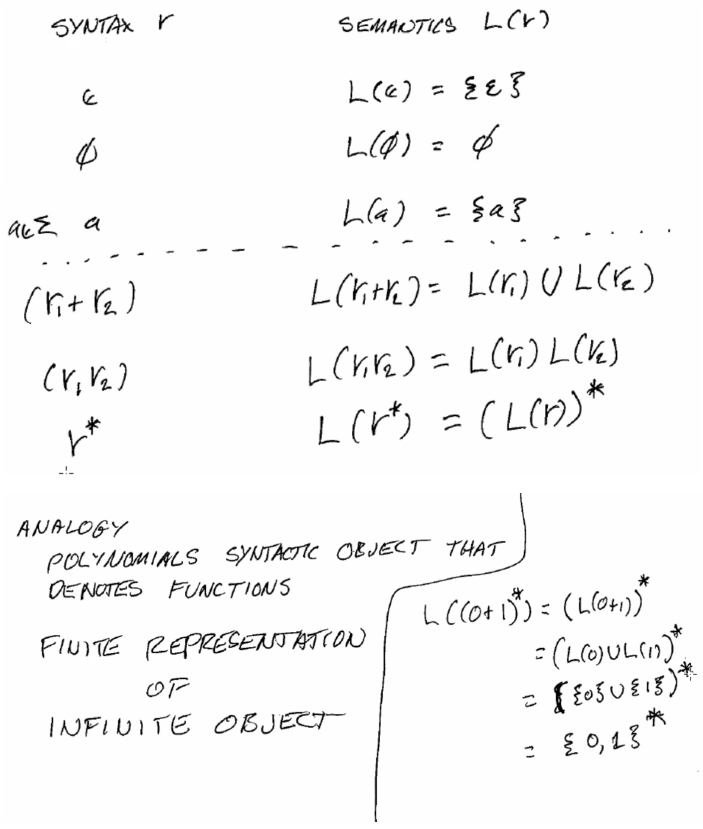
Reverse all arrows, make start state accept state, make accept states the start states (using epsilon).


ε
ε

CHAPTER 6

Regular Expressions

Regular expressions over alphabet Σ:



Ex. Begin with 0, end with 11: `0(0+1)*11`

Ex. Contains at least 2 1s: `(0+1)* 1 (0+1)* 1 (0+1)*` or `0* 1 0* 1 (0+1)*`

Note: `a+` is often used as an abbreviation for `aa*`

Ex. Contains the substring `111`: `(0+1)* 111 (0+1)*`

Ex. Even length: `((0+1)(0+1))*`

Ex. Odd length: `(0+1)((0+1)(0+1))*`

Ex. Strings that don't end with `01`: `e + 1 + (0+1)*0 + (0+1)*11` - it's hard to exclude things!

Ex. Every 0 is followed by at least one 1: `1* (011*)*`

Ex. 3rd symbol from right is 1: `(0+1)* 1 (0+1)(0+1)`

Ex. Contains both 01 and 10. `((0+1)* 01 (0+1)* 10 (0+1)*) + ((0+1)* 10 (0+1)* 01 (0+1)*)`

Ex. Not containing 00. `(1+01)*(0+e)`

Ex. At most 1 00. Consider: `(1+01)*` has no 00 and does not end w/ 0, `(1+10)*` has no 00 and does not start w/ 0. So exactly 1 occurance of 00: `(1+01)* 00 (1+10)*`, and up to 1 is just the combination: `(1+01)*(0+e) + (1+01)* 00 (1+10)*`.

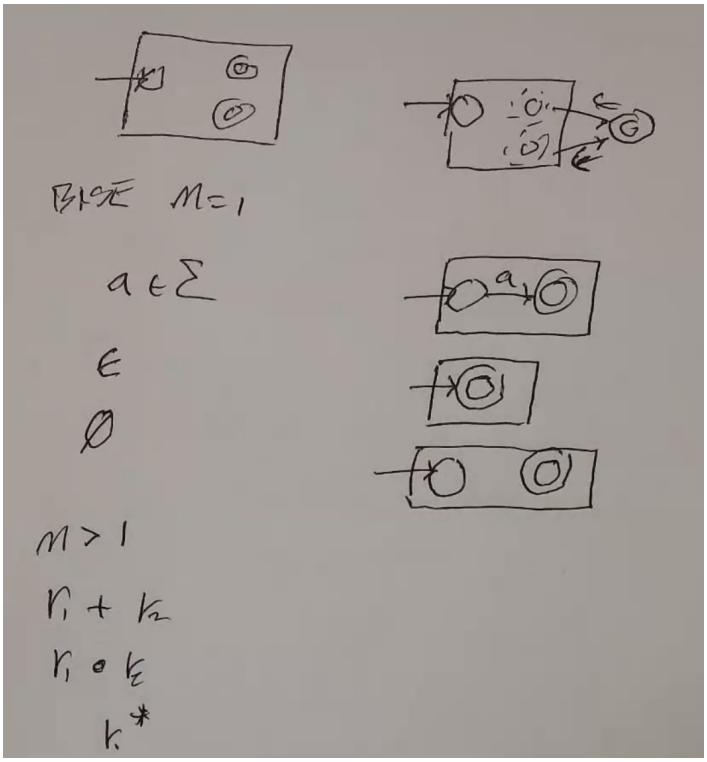
Ex. Not containing 110. `0* (100*)* 111*`

6.1 Kleene's Thm

For every regular expression, there is an equivalent ϵ-NFA.

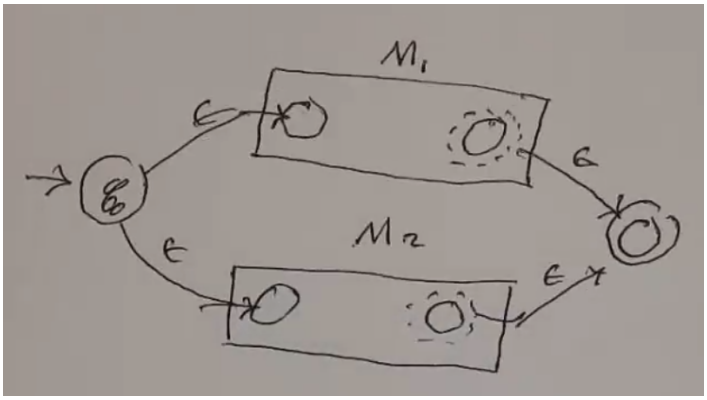
Use strong induction to prove:

1. **NFAs for regexes of length 1:**
    - let the building blocks have a unique accept state



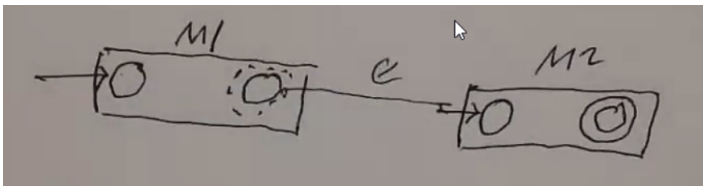
2. **All regexes longer than length 1 are:**
    - two smaller regexes with addition
    - two smaller regexes with concatenation
    - one smaller regex starred
3. IS: prove S(M) is true given that S(i) is true for all i < M.

- **Case:** $r = r_1 + r_2$
    - since r_1, r_2 are smaller than *r*, there exists an ϵ-NFA with a single unique accept state for each. (IH)
    - Let these be M_1 and M_2.
    - By adding a new start state, e-steps from that start state to the start states of of M_1 and M_2, and e-steps from the accept states to a new unique accept state, it is possible to construct an ϵ-NFA to accept $r_1 + r_2$.

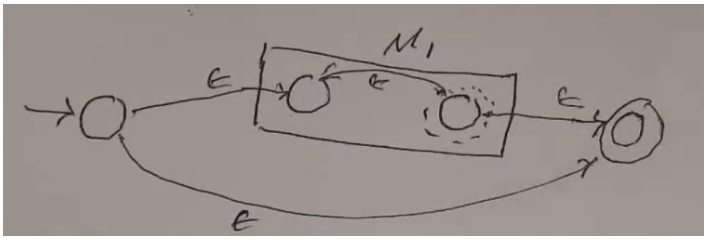


- **Case:** $r = r_1 \cdot r_2$ **(concatenate)**

  - since r_1, r_2 are smaller than *r*, there exists an ϵ-NFA with a single unique accept state for each. (IH)
  - Let these be M_1 and M_2.
  - By adding an e-step from the accept state of M1 to the start state of M2, it is possible to construct an ϵ-NFA to accept $r_1 + r_2$.



- **Case:** $r = r_1*$
  - since r_1 is smaller than *r*, there exists an ϵ-NFA with a single unique accept state. (IH)
  - Let this be M_1.
  - By adding an e-step from the old accept state to the old start state, from the old accept state to a new accept state, and the new start state to a new accept state.
  - QED.



6.1.1 The Other Way Around

Thm. Given a DFA, NFA, or e-NFA, there exists a regex that accepts the language of that FA.

GIVEN DFA $M = (Q, \Sigma, \delta, q_0, F)$, THERE IS A REG. EXPRESSION r S.T. $L(r) = L(M)$.

IDEA OF PROOF:

DECOMPOSE $L(M)$ INTO "SIMPLER" LANGUAGES.

SHOW EACH SIMPLER LANGUAGE CAN BE DENOTED BY A REG. EXPRESSION.

WRITE $L(M)$ AS THE FINITE UNION OF THE SIMPLER LANGUAGES. THIS CAN BE EXPRESSED USING "+".

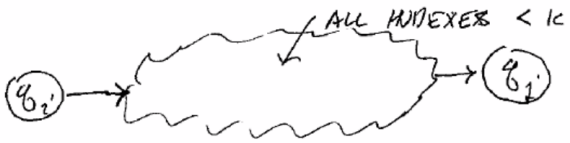
PROOF

LET $Q = \{q_0, q_1, \ldots, q_n\}$

IDEA: USE THE ACCEPTING STATES (F) TO DETERMINE THE SIMPLER LANGUAGES.

Define L_{ij}^k as the set of strings that will move the DFA from q_i to q_j, where all intermediate states' indices are < k.

LET $L_{ij}^k = \{w \in \Sigma^* \mid \hat{\delta}(q_i, w) = q_j$ AND ALL INTERMEDIATE STATES HAVE INDEXES LESS THAN $K\}$



NOTE: i AND/OR j CAN BE $\geq K$, THE RESTRICTION IS ON INTERMEDIATE STATES

Now we can define L(M) as the union of all ways to get from q_0 to an accept state for each accept state:

BY DEF. $L(M) = \{ w \mid \delta^*(q_0, w) \in F \}$

NOW WE CAN DECOMPOSE THIS TO:

$$L(M) = \bigcup_{q_j \in F} L_{0j}^{m+1}$$

WE NEED TO SHOW FOR EACH $q_j \in F$
THERE IS A REG. EXPRESSION DENOTING L_{0j}^{m+1}

CAN NOT USE INDUCTION DIRECTLY ON m
BECAUSE L_{0j}^{m} DOES NOT IN GENERAL COME
FROM SOME L_{0j}^{m-1}.

SOLUTION PROVE SOMETHING MORE GENERAL
STRONGER (INDUCTION LOADING) THAT OUR
DESIRED RESULT IS A SPECIAL CASE OF.

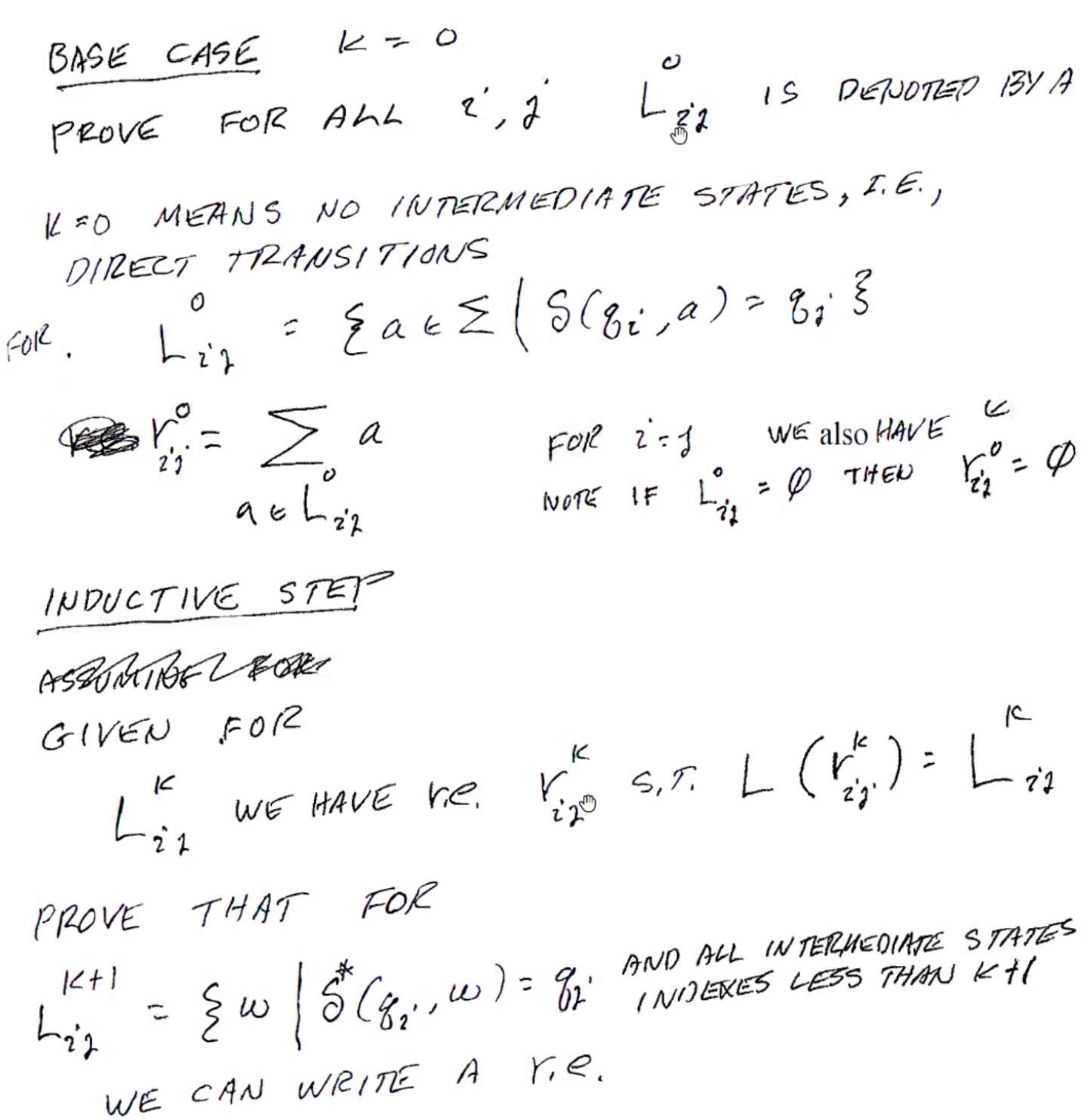
FOR ALL k, FOR ALL i, j
L_{ij}^{k} IS DENOTED BY A REG. EXPRESSION.

PROVE BY INDUCTION ON k.

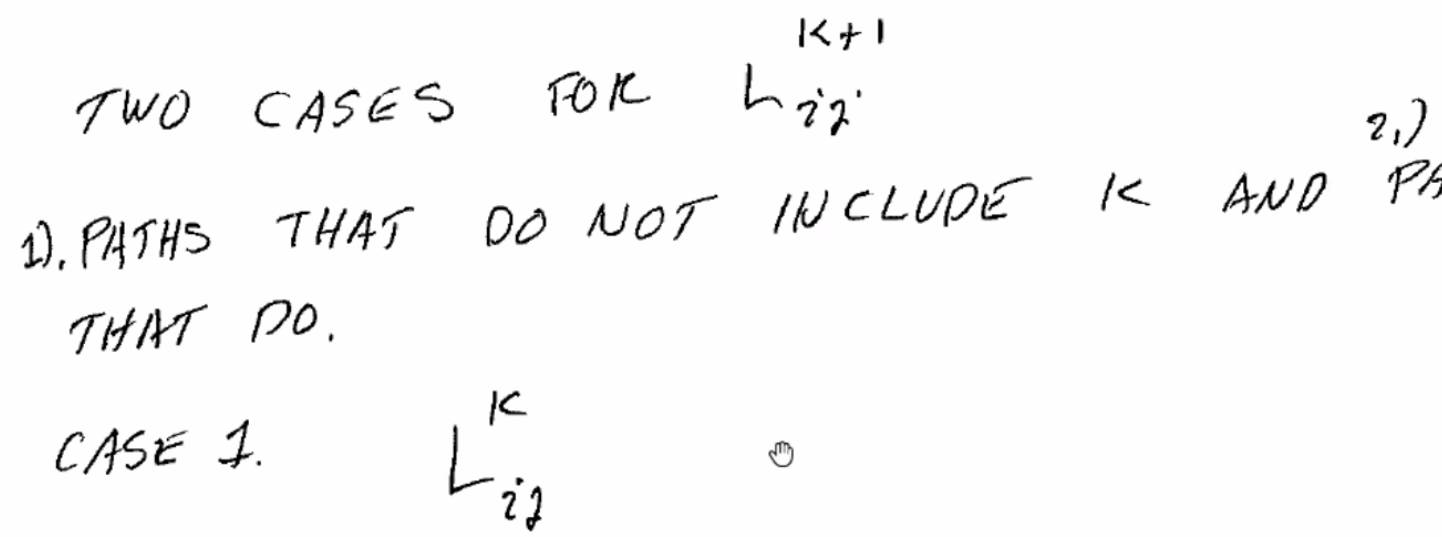
(Above: let's call this regex for L_{ij}^{k} r_{ij}^{k})

The induction:

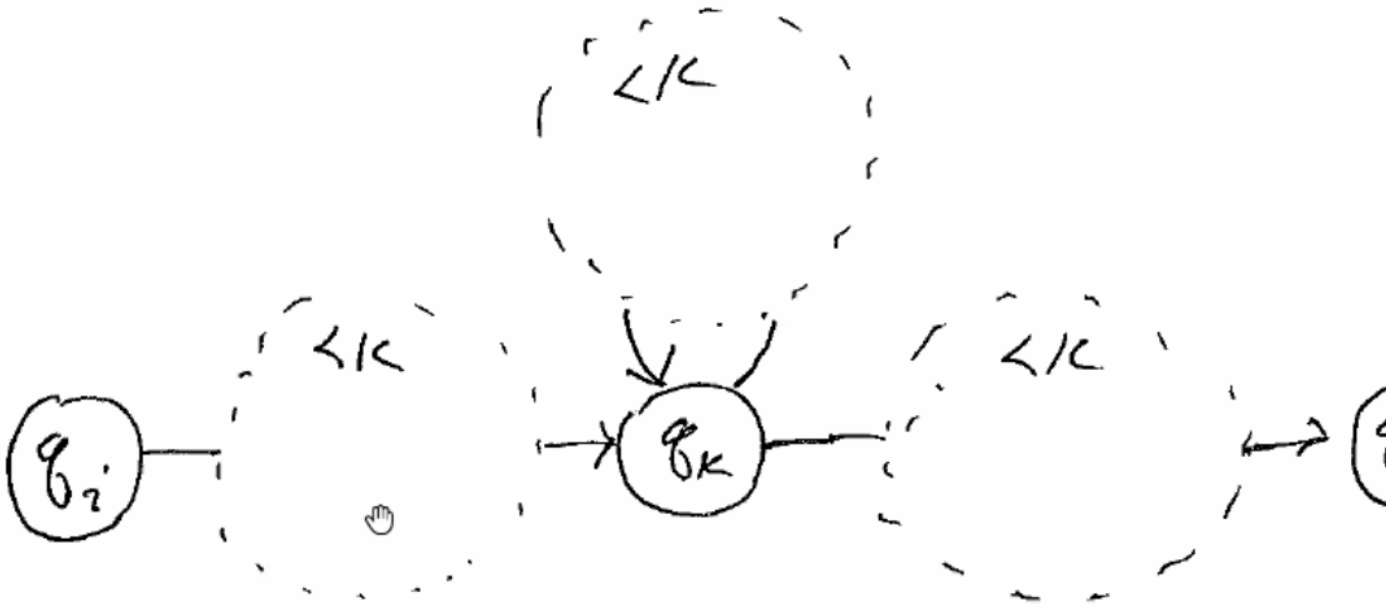
- base case. Since k = 0, this means only direct transitions:

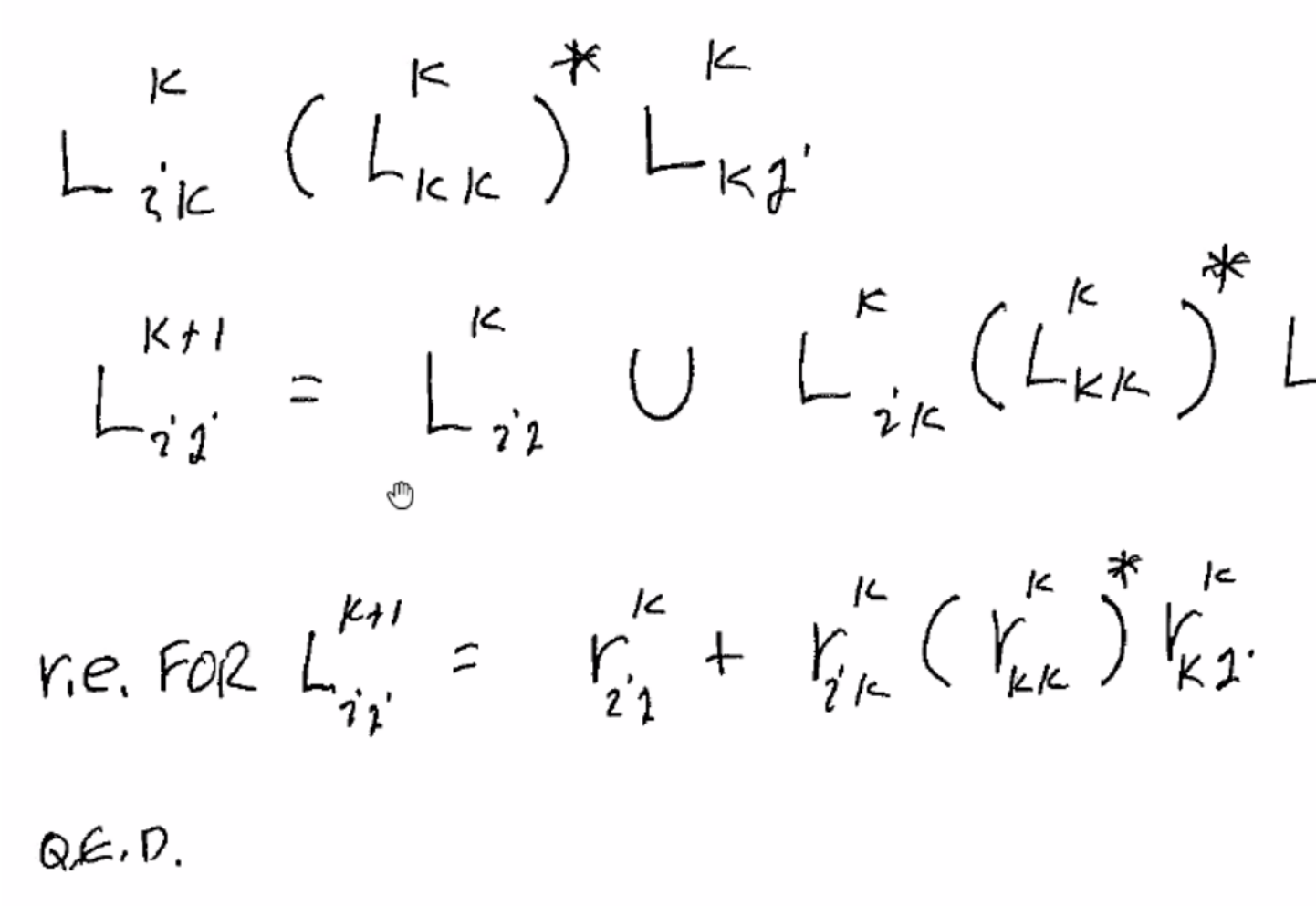


this produces two cases: if the path does not encounter k, or if it does:



And if it does, it must have a first time it enters k and the last time it exits k, and anything in between must be k to k, where all intermediate nodes are < k:



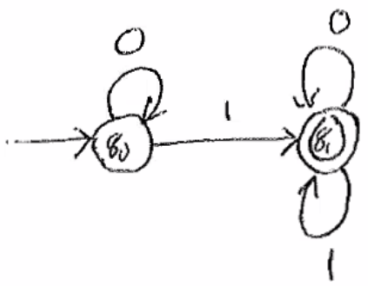


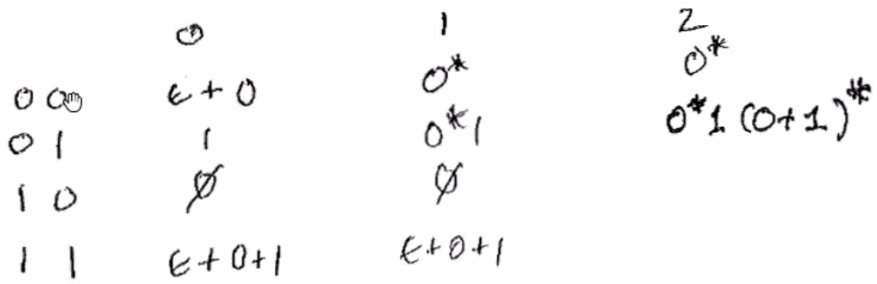
Now, we can expand this regex to:

$\sum_{q_j \in F} r^k_{0j}$.

6.1.2 Example

(left axis: i, j; top axis: k)





See published notes for full work.

CHAPTER 7

Pumping Lemma

If a language L is regular, then:

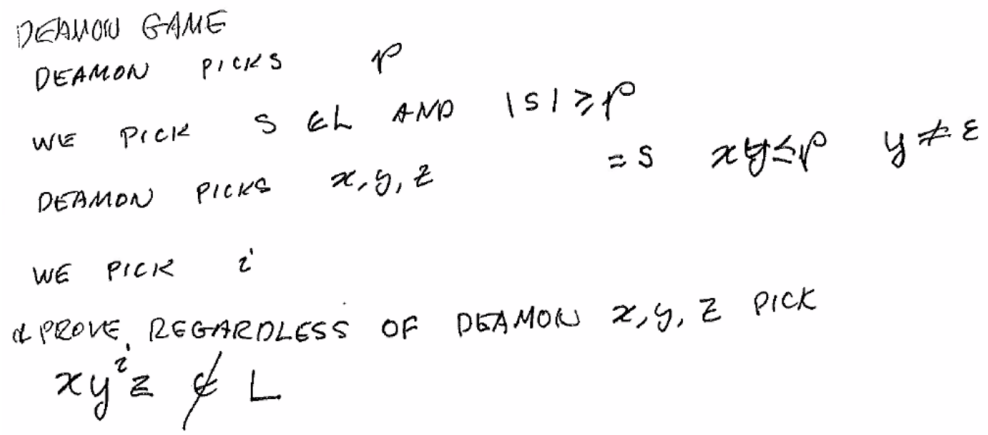
(P): There exists a $p \geq 0$ s.t. for any string $s \in L$ with $|s| \geq p$, there exist strings xyz s.t. $s = xyz, y \neq \epsilon, |xy| \leq p$, and for all $i \geq 0$, the string $xy^iz \in L$.

"there exists a non-empty string (y) within the first *p* characters (3rd constraint) that can be pumped, with the resulting string still being in the language."

The contrapositive of this ($\neg P \implies$ L is not regular) is used to prove that a language is not regular.

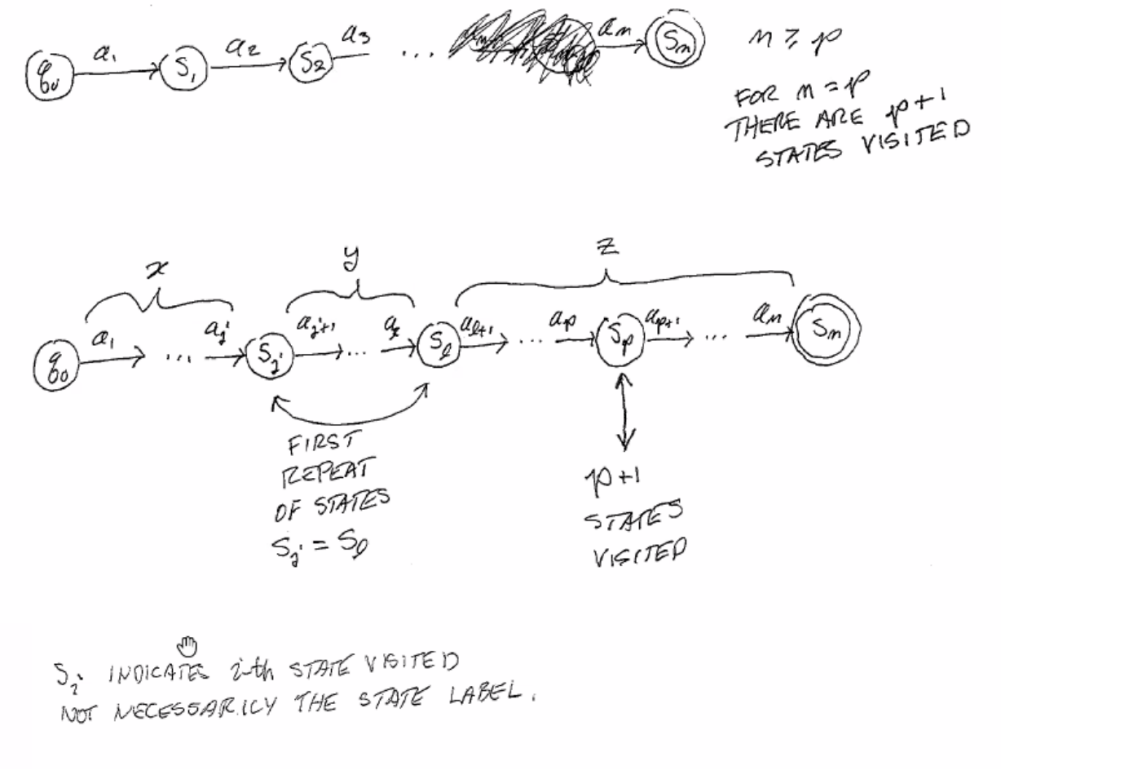
(not P): For all $p \geq 0$ there exists a string $s \in L$ with $|s| \geq p$, and for all x, y, z such that $xyz = s, y \neq \epsilon, |xy| \leq p$ there exists an $i \geq 0$ such that $xy^iz \notin L$.

You can use this adversarial game to prove the contrapositive:



7.1 Proof

Given a DFA with p states, processing any string with length $m \geq p$ means the machine will visit at least p + 1 states. By the pigeonhole principle, at least one state will be revisited.



Notice that the string that causes the DFA to go from s_j to s_l can be repeated infinitely, since they are a loop.

7.2 Examples

7.2.1 Ex 1

$A = \{0^m 1^m | m \geq 0\}$ is not regular.

Use the demon game as a valid proof:

- Demon picks p
- **we pick** $s \in L$ **and** $|s| \geq p$
  - $s = 0^p 1^p$
- demon picks partition x, y, z s.t. $xyz = s, |xy| \leq p, y \neq \epsilon$
- **we show any partition that satisfies these conditions cannot be pumped for some** $i \geq 0$
  - since $|xy| \leq p, y \neq \epsilon$ and we chose $s = 0^p 1^p$, y must be one or more 0s
  - choose $i = 2$: this causes xy^2z to have more 0s than 1s and not be in the language
  - QED

7.2.2 Ex 2

$L = \{w|$ # of 0s = # of 1s$\}$ is not regular.

Abbreviated demon argument:

- p
- $s = 0^p1^p$
- **for any partition** x, y, z **s.t.** $xyz = s, |xy| \leq p, y \neq \epsilon$**: (same argument as before)**
  - y must be made of 1 or more 0s
  - choose $i = 2$: this causes xy^2z to have more 0s than 1s and not be in the language
  - QED

7.2.3 Ex 3

$L = \{1^j0^i|j < i\}$ is not regular

- p
- $s = 1^p0^{p+1}$ (conditions: $s \in L, |s| = 2p + 1 \geq p$)
- **for any partition** x, y, z **s.t.** $xyz = s, |xy| \leq p, y \neq \epsilon$**:**
  - y must be made of 1 or more 1s
  - choose $i = 2$: this causes xy^2z to have $i \geq j$ and not be in the language
  - QED

7.2.4 Ex 4

$L = \{0^i1^j|i > j\}$ is not regular

- Assume L is regular
- so the reverse of L is regular (closure under reverse)
- The reverse of L is not regular (ex 3)
- so L is not regular. QED.

7.2.5 Ex 5

$L = \{ww|w \in \{0, 1\}*\}$ is not regular

- p
- $s = 0^p10^p1$
  - $|s| = 2p + 2 \geq p, s \in L$
- $xyz = 0^p10^p1$ s.t. $|xy| \leq p, |y| > 1$
- **if** $i = 2$**,** $xy^2z \notin L$**.**
  - since then there will be more 0s before the first 1 than before the last one.

7.2.6 Ex 6

Palindrones ($L = \{w | w = w^R\}$) are not regular.

- p
- $s = 0^p 10^p$
  - $|s| = 2p + 1 \geq p, s \in L$
- $xyz = 0^p 10^p$ **s.t.** $|xy| \leq p, |y| > 1$
  - for any i 2, the new string of the form $xy^i z$ will have more 0s before the 1 than after, and will no longer be in the language.

7.2.7 Ex 7

$L = \{0^m 1^n | m \neq n\}$ is not regular

- p
- $s = 0^p 1^{p+p!}$
- $xyz = 0^p 1^{p+p!}$ **s.t.** $|xy| \leq p, |y| > 1$
  - so *y* must be 1 or more 0s
  - let the length of *y* be *k*, so $xyz = 0^{p-k} 0^k 1^{p+p!}$
- **pick** $i = \frac{p!}{k} + 1$
  - $|y| = k$, so $|y^i| = ki = k * \frac{p!}{k} + 1 = p! + k$
- **then** $xyz = 0^{p-k} 0^{p!+k} 1^{p+p!}$
  - $= 0^{p+p!} 1^{p+p!} \notin L$.

7.2.8 Ex 7b

Alternatively, assume L is regular.

- Then $\neg L$ is regular (closed on complement)
- Then $\neg L \cap 0^* 1^*$ is regular (closed on intersection)
- That language is $\{0^m 1^n | m = n\}$, which is not regular - contradiction!

7.2.9 Ex 8

$L = \{1^{n^2} | n \geq 0\}$ is not regular

- p
- $s = 1^{p^2}$
- $xyz = s$ s.t. $|xy| \leq p, |y| > 1$
- **let i = 2, then:**
  - $|xy^2 z| - |xyz| = |y| \leq p$
  - $|xy^2 z| \leq p^2 + p$

- $p^2 < |xy^2z| \leq p^2 + p < (p+1)^2$, so $s \notin L$.

7.2.10 Ex 9

$L = \{0^{2^n} | n \geq 1\}$ is not regular

- p
- $s = 0^{2^p}$
- $xyz = s$ **s.t.** $|xy| \leq p, |y| > 1$
  - let $|x| = a, |y| = b, |z| = c, 0 < b \leq p, a + b = p$
- **let i = 2,** $s' = xy^2z$**, then:**
  - $|xy^2z| = 2^p + b$
  - $2^p + b \leq 2^p + p$
  - $< 2^p + 2^p$
  - $= 2^{p+1}$
  - so $|xy^2z|$ is not a power of 2, so $s' \notin L$.

7.2.11 Ex 10

$L = \{a^{n!} | n \geq 0\}$ is not regular

- p
- $s = a^{p!}$
- $xyz = s$ **s.t.** $|xy| \leq p, |y| > 1$
  - let $|x| = j, |y| = m > 0, |z| = n, j + m + n = p!$
- **pick *i* s.t.** $|xy^iz| \neq q!$ **for any *q***
  - for any *i*, $|xy^iz| = j + im + n = p! + (i-1)m$
  - pick $i = (p+1)! + 1$, then $|xy^iz| = p!(p+1)!m$
  - $= p!(1 + m(p+1))$, prove that this is not a factorial
  - assume $q! = p!(1 + m(p+1))$
  - then dividing both sides by $p!$: $q(q-1)(q-2)...(p+2)(p+1) = (1 + m(p+1))$
  - impossible because left is divisible by $p+1$ and right side leaves remainder of 1.
  - therefore $p!(1 + m(p+1))$ is not a factorial, so $xy^iz \notin L$.

7.2.12 Ex 11

$L = \{1^n | n \text{ is prime}\}$

- p
- $s = 1^{p'}$ where p' is a prime larger than p
- $xyz = s$ **s.t.** $|xy| \leq p, |y| > 1$

  - let $x = 1^a, a \geq 0$
  - let $y = 1^b, b > 0$
  - let $z = 1^c, c \geq 0$
  - where $a + b + c = p'$
  - so the claim is $a + ib + c$ is a prime for all i
- **let** $i = a + 2b + c + 2$
  - then $a + ib + c = (b + 1)(a + 2b + c)$
  - this is a factor of two numbers, and so not a prime
  - therefore $xy^i z \notin L$.

CHAPTER 8

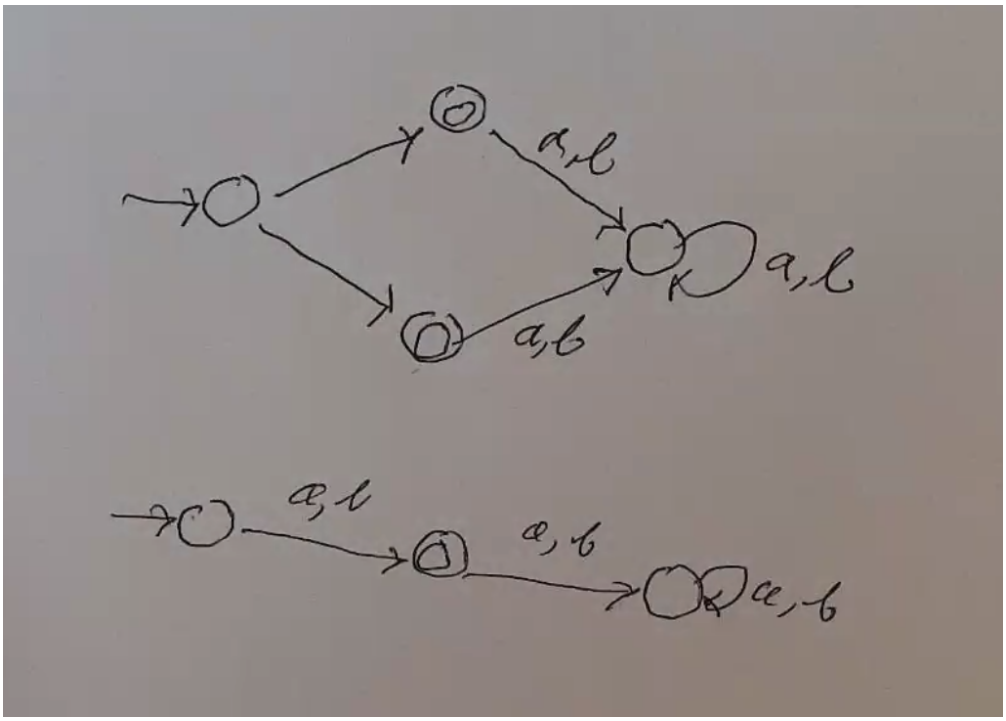
DFAs Extra

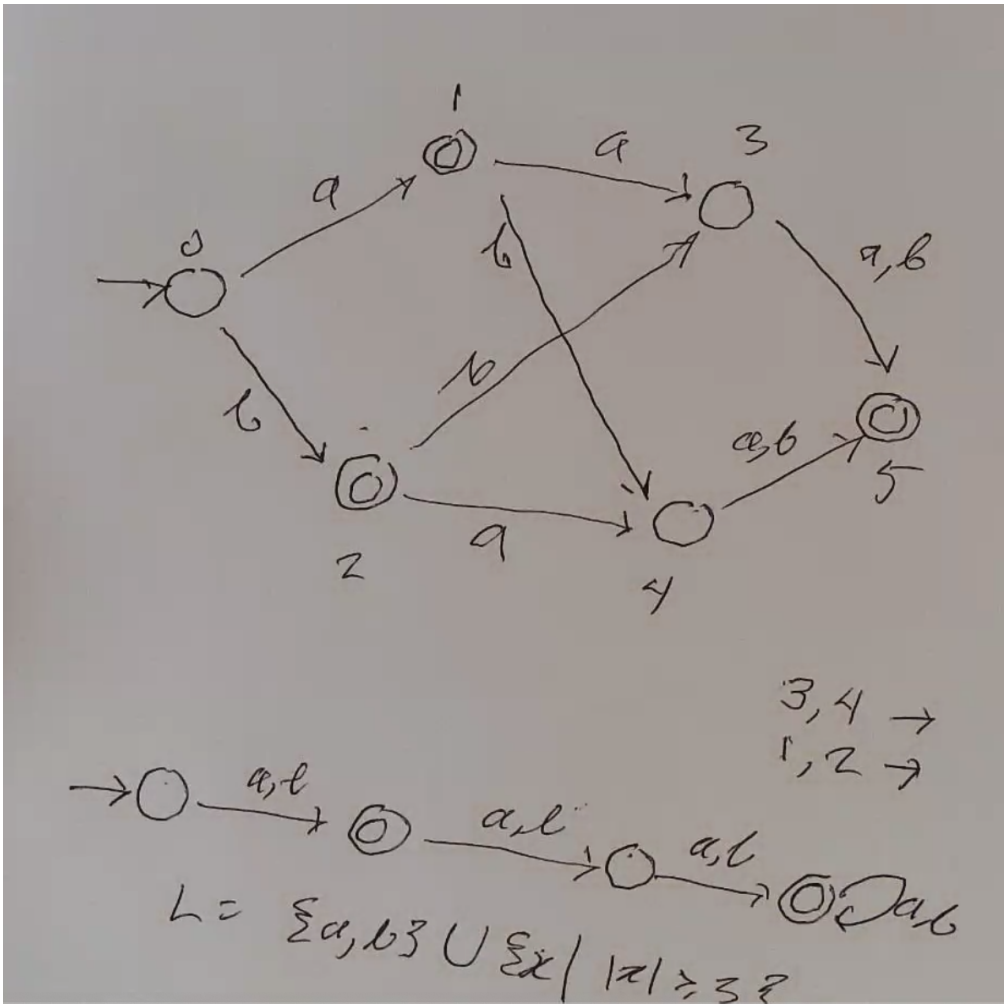
8.1 Minimizing a DFA

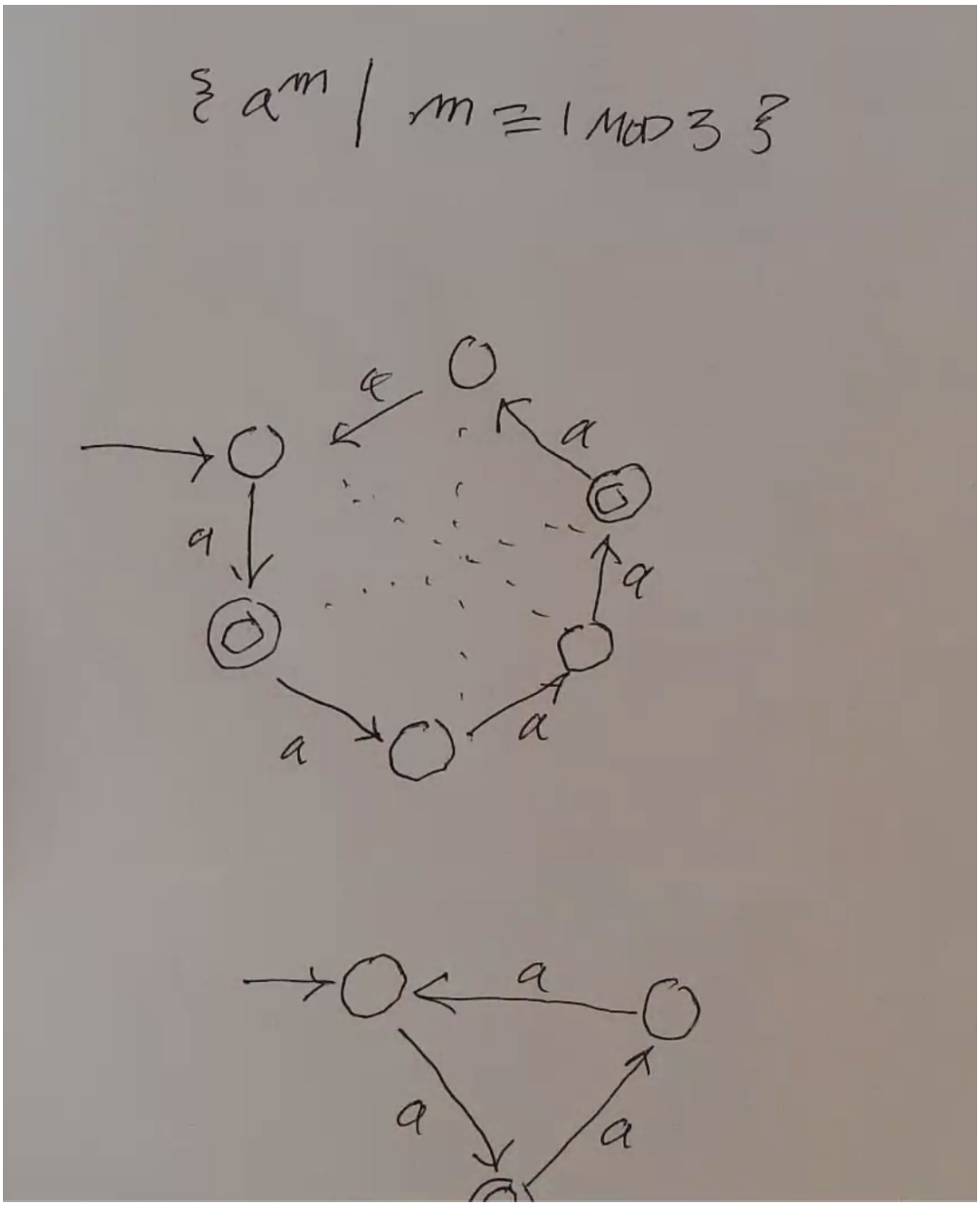
Given a DFA:

1. Remove inaccessible states
2. Collapse equivalent areas

E.g.:

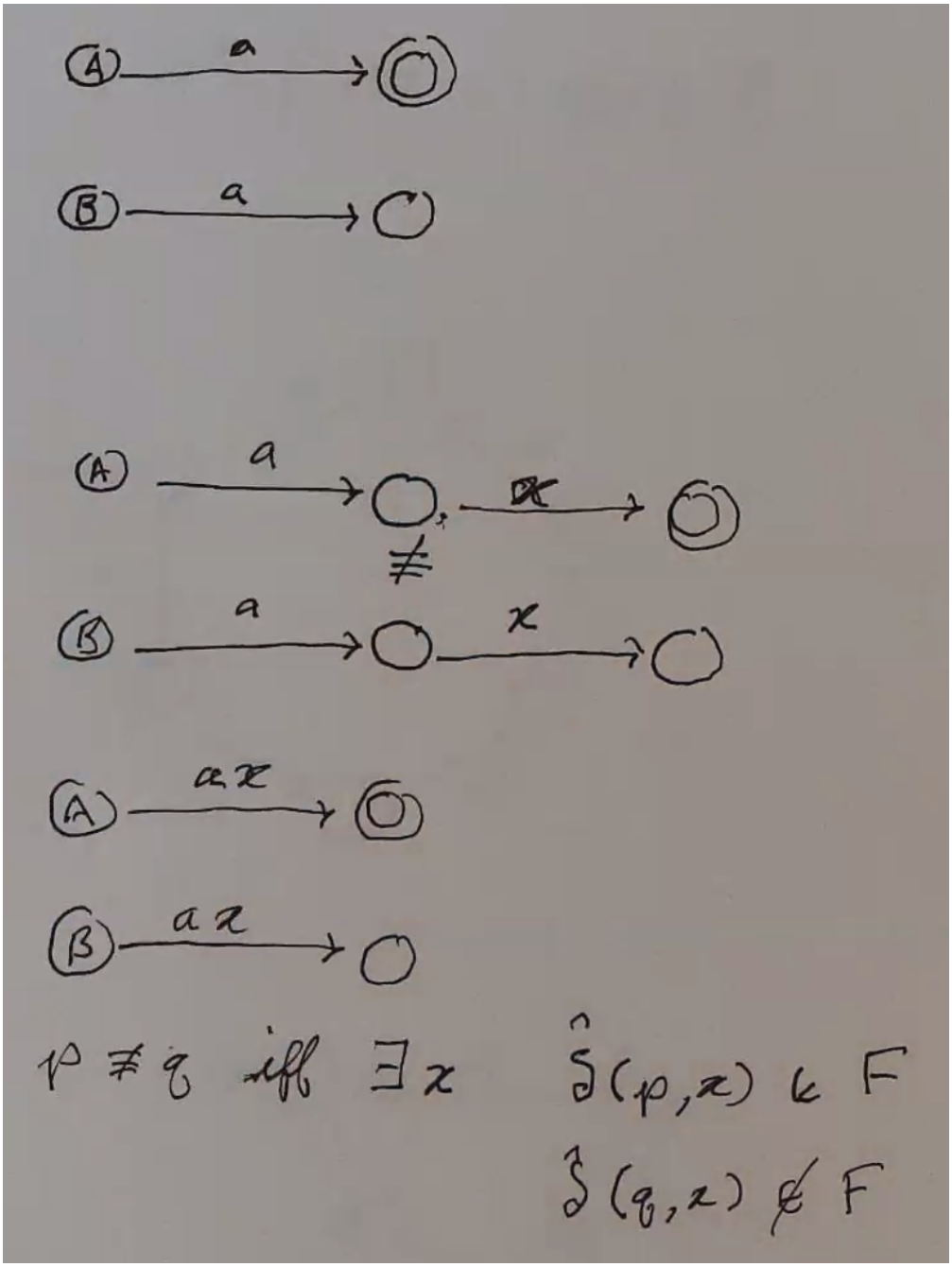



0
1
2
3
4
5
a
b
a,b
3,4 →
1,2 →
L = {a,b}* ∪ {x | |x| ≥ 3}

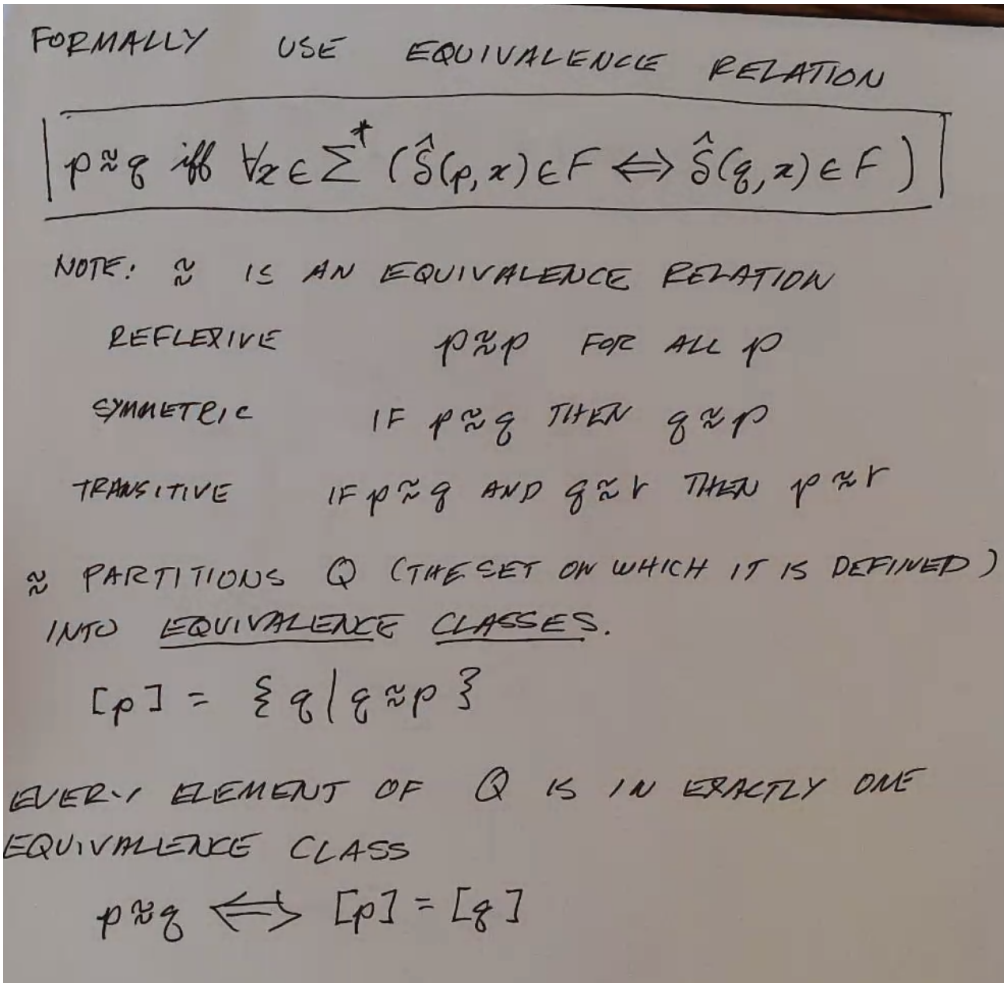


8.1.1 Identifying Equivalent States

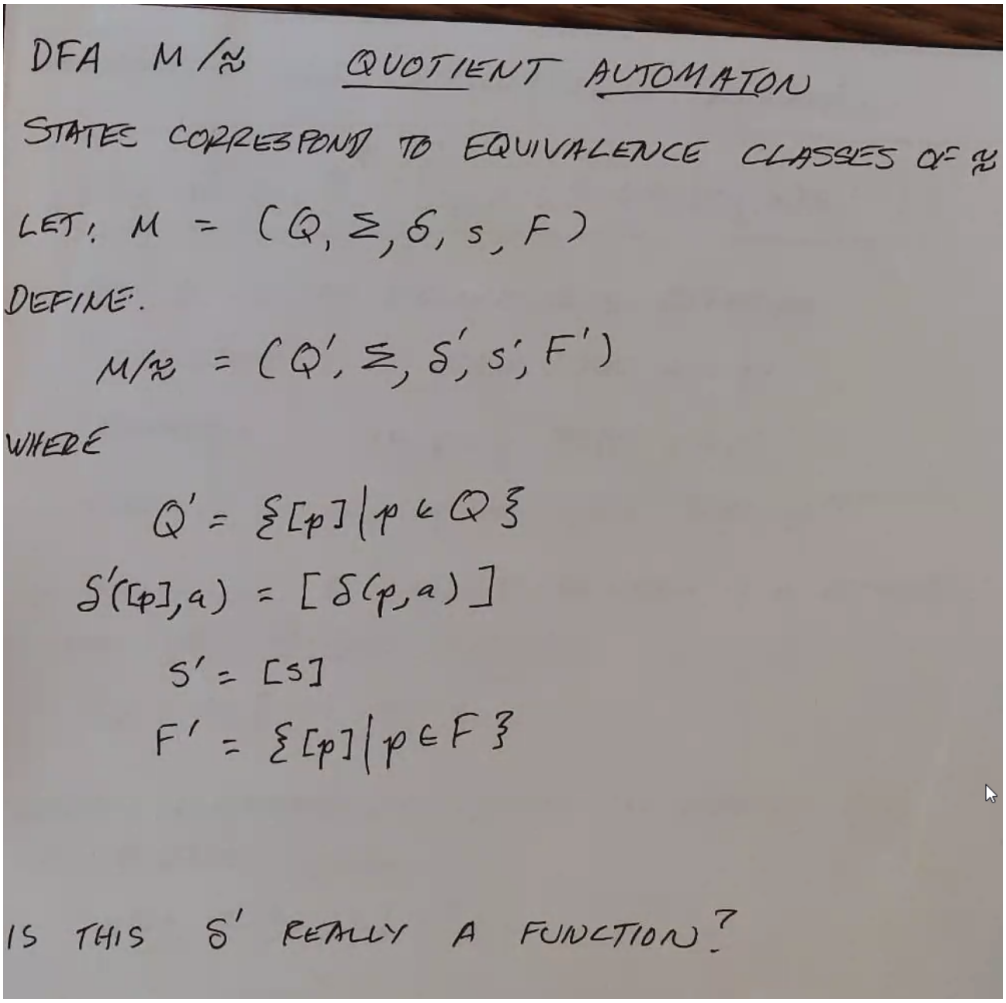
Do this by identifying all states that cannot be equivalent: two states cannot be equivalent if processing the same some string at each state brings you to a different acceptance value



Formally, $p \approx q$ iff $\forall x \in \Sigma^*(\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F)$



You can use these equivalence classes to make a quotient automaton:

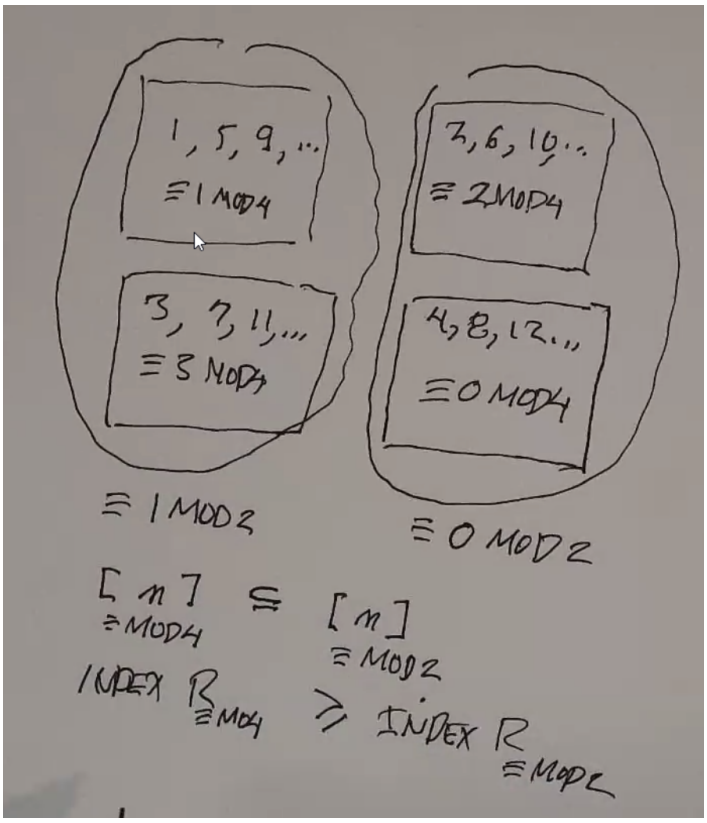


8.1.2 Equivalence Relations

- must be reflexive, symmetric, transitive
- **partitions a set into disjoint parts (equivalence classes)**
  - if R is an equivalence relation on A, $[x]_R = \{y | x \ R \ y\}$
- given $[x]_R$ and $[y]_R$, they are either the same or disjoint
- the union of all equivalence classes of a set is the set
- the *index* of an equivalence relation is the number of equivalence classes

If, for all $x \in A$, $[x]_1 \subseteq [x]_2$ (where 1 and 2 are 2 different relations), then 1 is *finer* than 2.

Therefore, the index of relation 1 will be greater than the index of relation 2.



8.2 Myhill-Nerode

Idea:

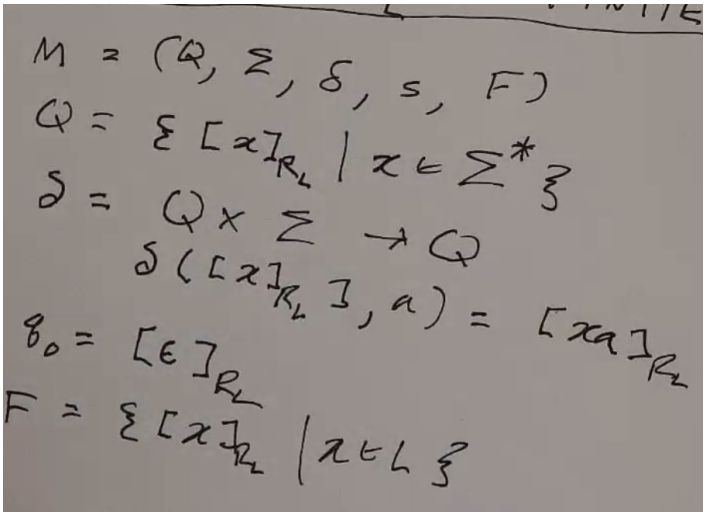
- Let L be any language in Σ^* (so $L \subseteq \Sigma^*$).
- Let R_L be a special equivalence relation on Σ^*.
- $x \; R_L \; y$ **iff** $\forall z \in L, \; (xz \in L \iff yz \in L)$
  - these two strings are equivalent if regardless of what you append to them, they are both either in the language or not

Thm:

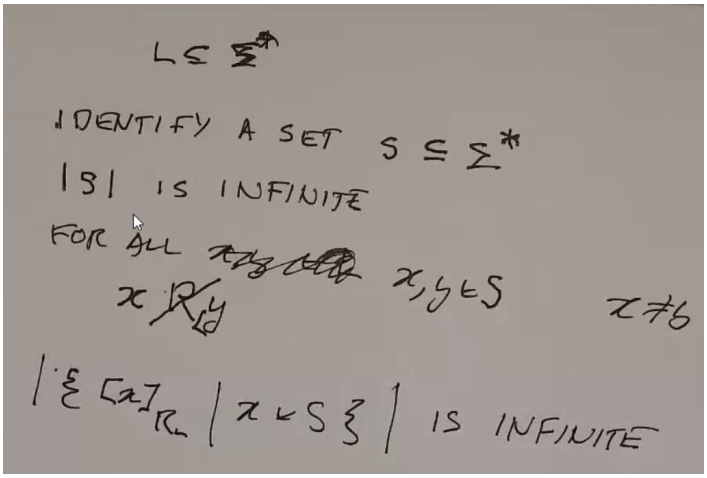
Let $L \subseteq \Sigma^*$. Then the following statements are equivalent:

1. L is regular
2. The index of R_L is finite

$1 \implies 2$ is relatively easy to prove - $2 \implies 1$ (shown here) is harder, but we prove it by constructing a DFA:



If you can find an infinite sized set of all strings in Σ^* such that no two of them are in the same equivalence class, then that language is not regular (since the equivalences classes of that set are a subset of equivalence classes in that language):



8.2.1 Ex 1

Prove that $L = \{0^n 1^n | n \geq 1\}$ is not regular using M-N:

- Let $S = \{0^n | n \geq 1\}$
- $|S|$ is infinite
- **Examine** $0^i, 0^j \in S$ **where** $i \neq j$
  - By appending 1^i to both strings, we get one string in the language and another that is not
  - so all items in this set are in different equivalence classes of R_L
- So the language is not regular.

8.2.2 Ex 2

Prove that $L = \{w \in \Sigma^* | \text{w is a palindrome}\}$ is not regular using M-N:

- Let $S = \{01, 001, 0001, ...\} = \{0^i1 | i \geq 1\}$
- $|S|$ is infinite
- **Examine** $0^i1, 0^j1 \in S$ **where** $i \neq j$
  - By appending 0^i to both strings, we get one string in the language and another that is not
  - so all items in this set are in different equivalence classes of R_L
- So the language is not regular.

8.2.3 Ex 3

$L = \{ww | w \in \Sigma^*\}$

- Let $S = \{0^i1 | i \geq 1\}$
- $|S|$ is infinite
- **Examine** $0^i1, 0^j1$ **where** $i \neq j$
  - **By appending** 0^i1 **to both strings, we get one string in the language and another that is not:**
    - $0^i10^i1 \in L$
    - $0^j10^i1 \notin L$
  - so all items in this set are in different equivalence classes of R_L
  - so the index of R_L is infinite
- So the language is not regular.

8.2.4 Ex 4

$L = \{1^{m!} | m \geq 1\}$

- Let $S = L$
- $|S|$ is infinite
- **Examine** $1^{i!}, 1^{j!}$ **where** $i \neq j$
  - By appending $1^{ii!}$, we get:
  - $1^{i!}1^{ii!} = 1^{(i+1)!} \in L$
  - $1^{j!}1^{ii!} = 1^{\frac{j!}{i!}i! + ii!}$
    - $= 1^{i!(\frac{j!}{i!} + i)}$
    - **proof by contradiction: assume** $i!(\frac{j!}{i!} + i)$ **is some factorial** $q!$
      - $q! = i!(\frac{j!}{i!} + i)$
      - $q(q-1)...(i+1) = \frac{j!}{i!} + i$
      - $= j(j-1)...(i+1) + i$
      - dividing both sides by $i+1$, the remainder on the left is 0 while the remainder on the right is 1
      - so $i!(\frac{j!}{i!} + i)$ is not some factorial.

        - $*$ so $= 1^{i!(\frac{j!}{i!}+i)} \notin L$
    - so all items in this set are in different equivalence classes of R_L
    - so the index of R_L is infinite
- so the language is not regular.

8.2.5 Ex 5

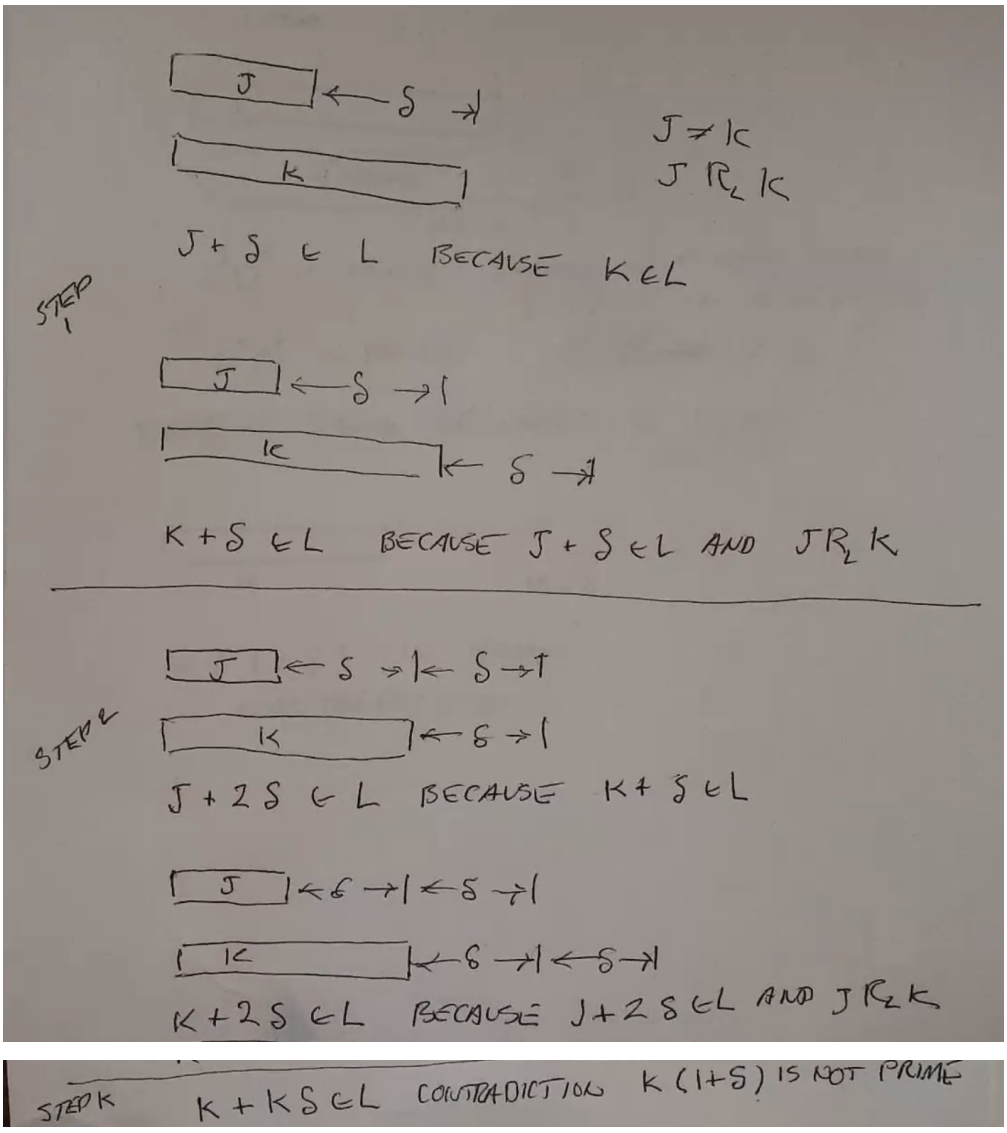
$L = \{a^i b^j k^c | i, j, k \geq 0 \wedge i = 1 \implies j = k\}$

This language cannot be proven irregular using the pumping lemma thm.

- let $S = \{ab^i | i \geq 1\}$
- $|S|$ is infinite
- **Examine ab^i, ab^j where $i \neq j$. Append c^i to both:**
    - $ab^i c^i \in L$
    - $ab^j c^i \notin L$
- so the index of R_L is infinite and L is not regular.

8.2.6 Ex 6

$L = \{0^p | p \text{ is prime}\}$



8.2.7 Ex 7

$L = \{1^n | n \text{ is even}\}$ is regular. Show that the index of R_L is finite.

Note: Intuitively, the 2 equivalence classes of R_L are the even lengths and the odd lengths.

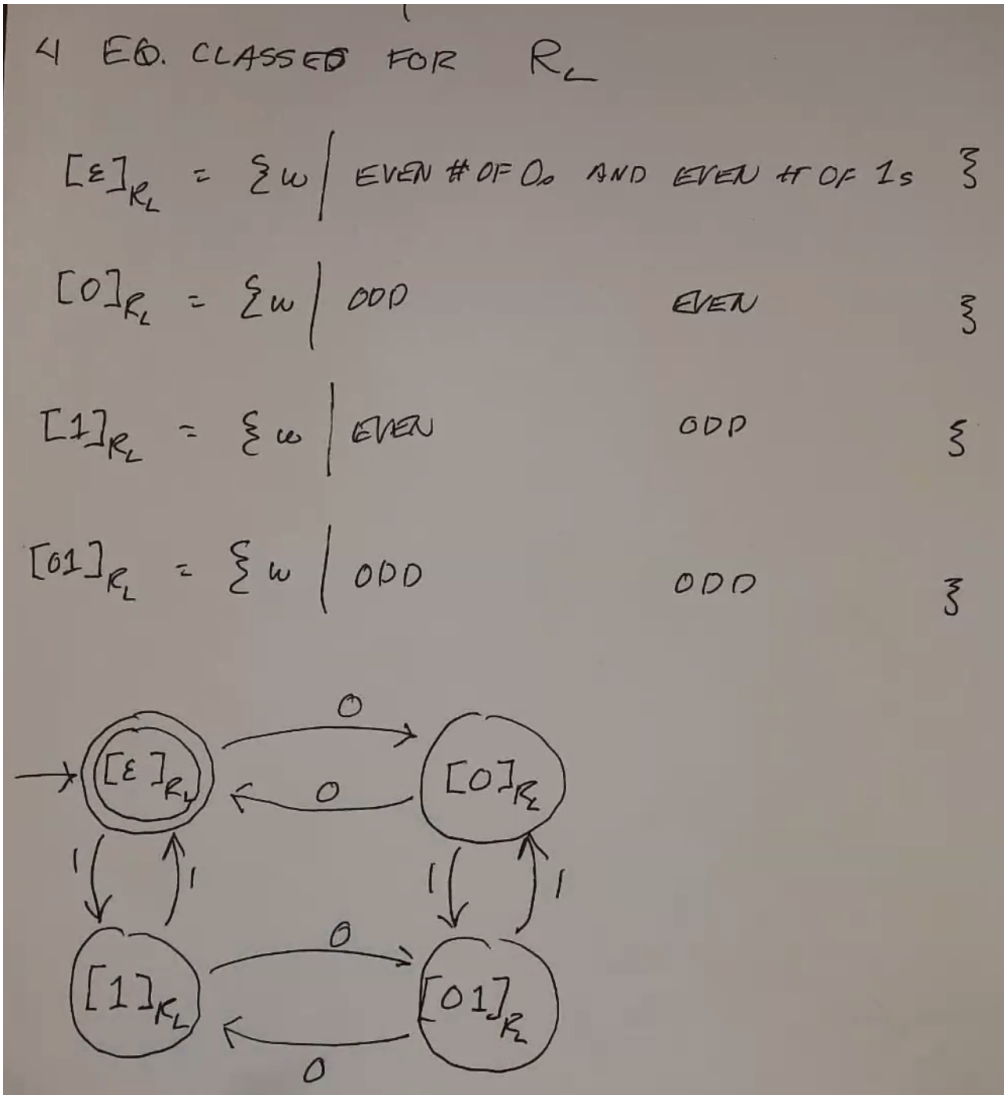
- All strings in Σ^* fall into one of two equivalence classes of R_L:
- **Case 1: Examine $1^j z, 1^k z$ where *j* and *k* are even.**
  - **Case 1: *z* is of even length.**
    - The lengths of both strings will be even (sum of 2 even numbers is an even number)
    - so both strings will be in the language.
  - **Case 2: *z* is of odd length.**
    - The lengths of both strings will be odd (sum of even and odd numbers is an odd number)

        - so both strings will not be in the language.
- **Case 2: Examine $1^j z, 1^k z$ where *j* and *k* are odd.**
    - **Case 1: *z* is of even length.**
        - The lengths of both strings will be odd (sum of even and odd numbers is an odd number)
        - so both strings will not be in the language.
    - **Case 2: *z* is of odd length.**
        - The lengths of both strings will be even (sum of 2 odd numbers is an even number)
        - so both strings will be in the language.
- The index of R_L is finite, so the language is regular.

8.2.8 Additional Comments

Each equivalence class of R_L corresponds to a state in the minimal DFA of the language.

Ex: $L = \{w | w \text{ has an even number of 0s and 1s}\}$



8.3 Two-Way DFAs

aka 2dfa

$M = (Q, \Sigma, \vdash, \dashv, \delta, s, t, r)$

- Q = a finite set of states
- Σ = a finite set (input alphabet)
- $\vdash$ = left end marker ($\notin \Sigma$)
- $\dashv$ = right end marker ($\notin \Sigma$)
- $\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to (Q \times \{L, R\})$
- $s \in Q$ = start state
- $t \in Q$ = unique accept state
- $r \in Q$ = unique reject state

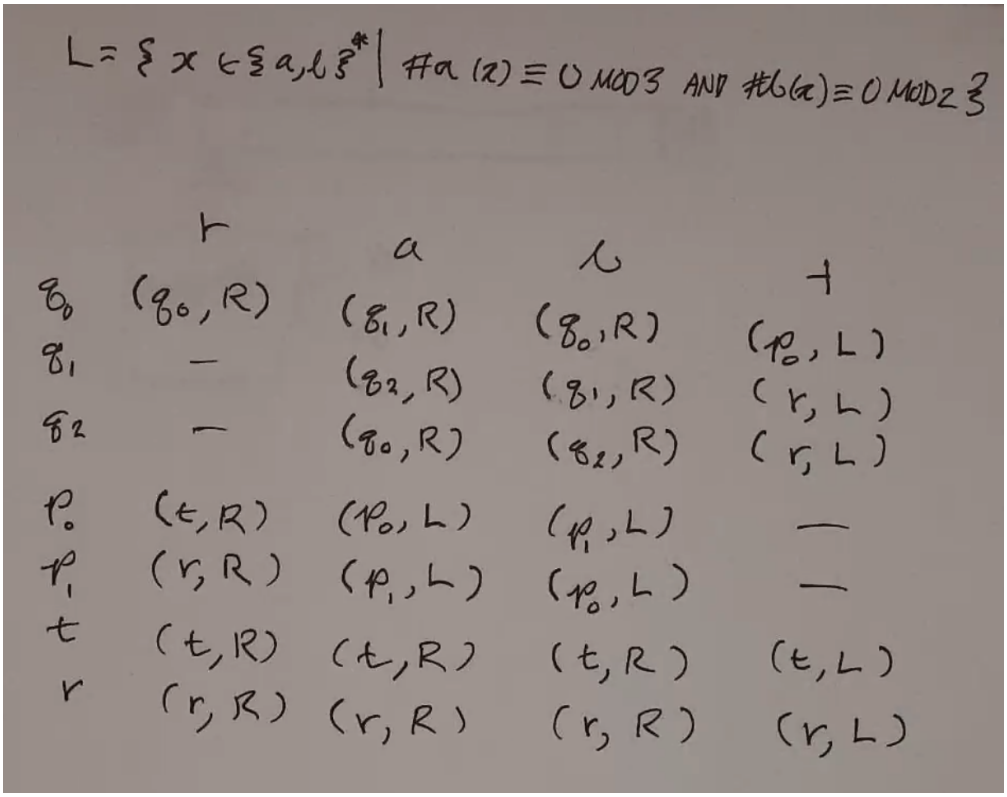
This makes it possible to accept or reject an input without reading the whole thing, or loop forever.

Note: there are some safety mechanisms in place:

- $\forall q \in Q$**, you cannot go off the end of the tape:**
  - $\delta(q, \vdash) = (u, R)$ for some $u \in Q$
  - $\delta(q, \dashv) = (u, L)$ for some $u \in Q$
- $\forall b \in \Sigma \cup \{\vdash\}$**,**
  - $\delta(t, b) = (t, R)$
  - $\delta(t, \dashv) = (t, L)$
  - $\delta(r, b) = (t, R)$
  - $\delta(r, \dashv) = (t, L)$
  - once in *t* or *r*, keep moving right.

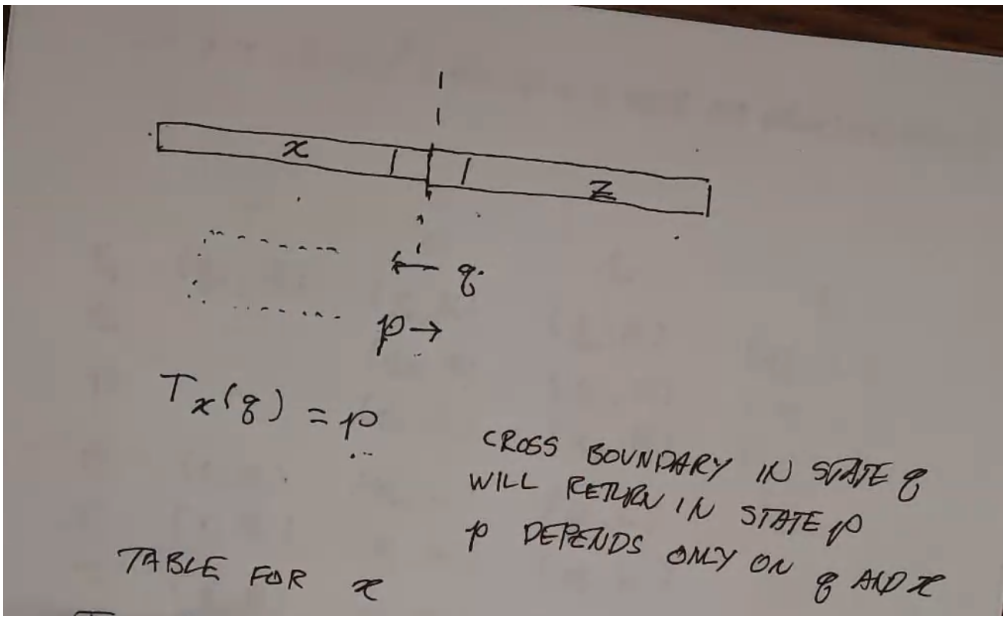
8.3.1 Example

- Scan LtR, counting a's, then RtL counting b's.
- Reject early if encounter right end with invalid num of a's.
- Otherwise make reject/accept decision at left end given number of b's.



8.3.2 Additional Comments

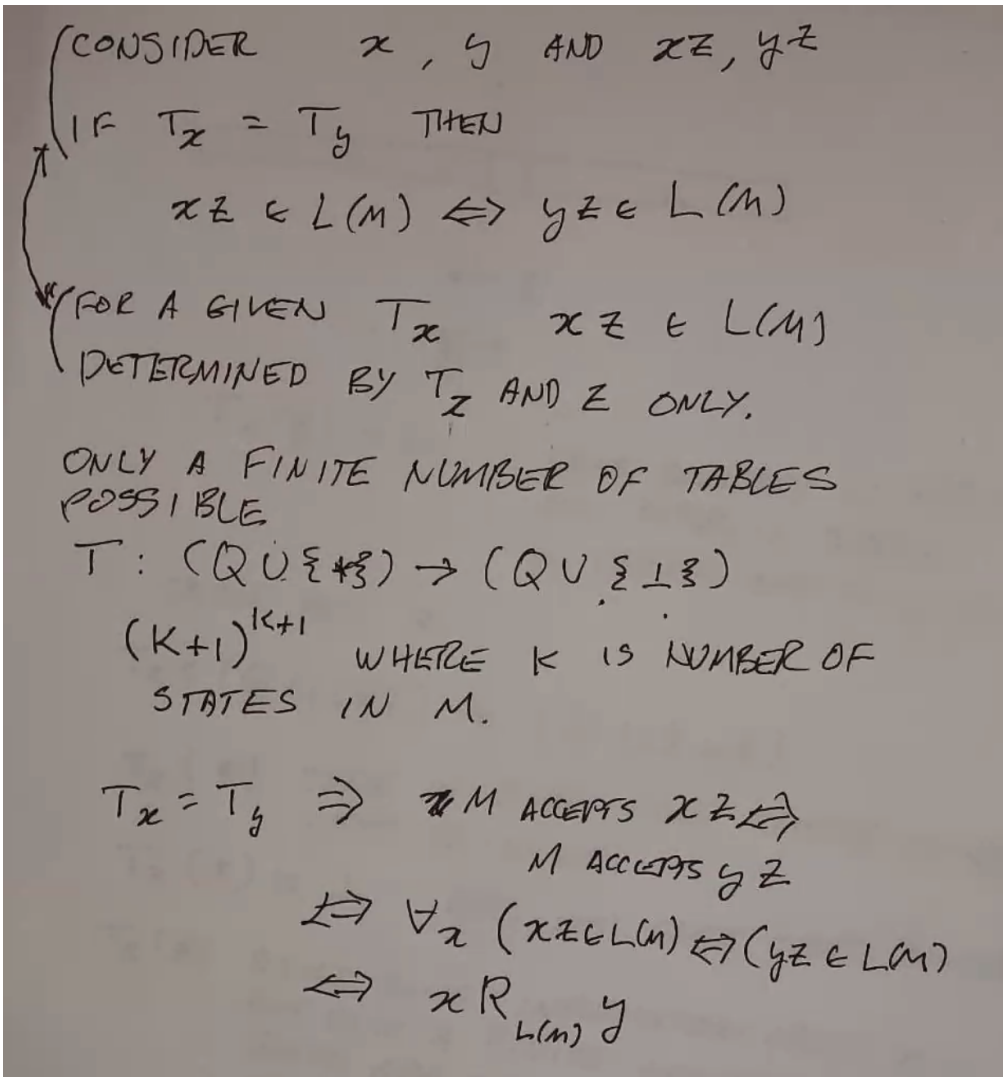
Given some boundary on the tape, assuming you cross that boundary at some point again, the state you are in when you cross the boundary going the opposite direction depends only on tape behind the boundary and the state you crossed in.



There is also a special symbol for never crossing the boundary again:

Warning: There is a missing image here. Please open a pull request if you have the notes that go here.

This has a relationship with the Myhill-Nerode relationship:



Which proves that this machine has an equal amount of power as a DFA!

CHAPTER 9

Context-Free Languages

CFLs are more powerful than regular languages! You can use them to do things like balance parens.

Ex. palindromes, 0^n1^n, balanced parens, etc.

Context-Free Grammars

CFGs are used to generate CFLs.

9.1 Definition

$G = (N, \Sigma, P, S)$

Note: N ("nonterminals") is sometimes V ("variables"), and P ("productions") is sometimes R ("rules")

- N is a finite set (nonterminal symbols/variables)
- Σ **is a finite set (terminal symbols, disjoint from N)**
  - $N \cap \Sigma = \emptyset$
- P is a finite subset of $N \times (N \cup \Sigma)^*$ (productions)
- $S \in N$ is the start symbol/variable

9.2 Convention

- $A, B, C...$ = nonterminals
- $a, b, c...$ = terminals
- $\alpha, \beta, \gamma...$ = strings in $(N \cup \Sigma)^*$
- $A \to \alpha$ for (A, α)

- $A \to \alpha_1, A \to \alpha_2, A \to \alpha_3 = A \to \alpha_1 | \alpha_2 | \alpha_3$

A grammar can be abbreviated as a list of rules. The nonterminals can be found on the left, the rules are listed, the terminals are the states on the right that are not on the left, and the start state is the top nonterminal.

9.3 Examples

Note: For these examples, $\to$ will be expressed as `:=` and ϵ as `e`.

9.3.1 Ex 1

$a^n b^n$

Formally:

- $G = (N, \Sigma, P, S)$
- $N = \{S\}$
- $\Sigma = \{a, b\}$
- $P = \{S \to aSb, S \to \epsilon\}$
- $S = S$

Abbreviated:

```
S := a S b | e
```

Ex. $a^3 b^3$ can be `S := aSb := aaSbb := aaaSbbb := aaabbb`.

9.3.2 Ex 2

Even length palindromes.

```
S := a S a | b S b | e
```

Ex. `aabbaa` can be `S := aSa := aaSaa := aabSbaa := aabbaa`.

9.3.3 Ex 3

Odd length palindromes.

```
S := a S a | b S b | a | b
```

Ex. `aababaa` can be `S := aSa := aaSaa := aabSbaa := aababaa`.

9.3.4 Ex 4

The set of strings over {a, b} such that the reversal of a string is the string with a's and b's swapped

```
S := a S b | b S a | e
```

9.3.5 Ex 5

The set of even length strings

```
S := a T | b T | e
T := a S | b S

OR

S := aaS | abS | baS | bbS | e
```

9.3.6 Ex 6

Even length with two middle symbols the same

```
S := a S a | a S b | b S a | b S b | aa | bb
```

9.3.7 Ex 7

Odd length with first, middle, and last symbols the same

```
S := a T a | b U b
T := a T a | a T b | b T a | b T b | a
U := a U a | a U b | b U a | b U b | b
```

9.3.8 Ex 8

Equal number of a's and b's

```
S := a S b | b S a | S S | e
```

9.3.9 Ex 9

Palindromes

```
S := a S a | b S b | a | b | e
```

9.3.10 Ex 10

Balanced parentheses

```
S := ( S ) | S S | e
```

9.3.11 Ex 11

$\{0,1\}^*$

```
S := 0 S | 1 S | e
```

9.4 Special Constructions

- **Right Linear**
  - `A := x B | x` where $x \in \Sigma^*$
- **Strongly Right Linear**
  - `A := x B | e` where $x \in \Sigma$
- **Left/Strongly Left Linear are similar.**
  - All the linear constructions generate the regular languages!

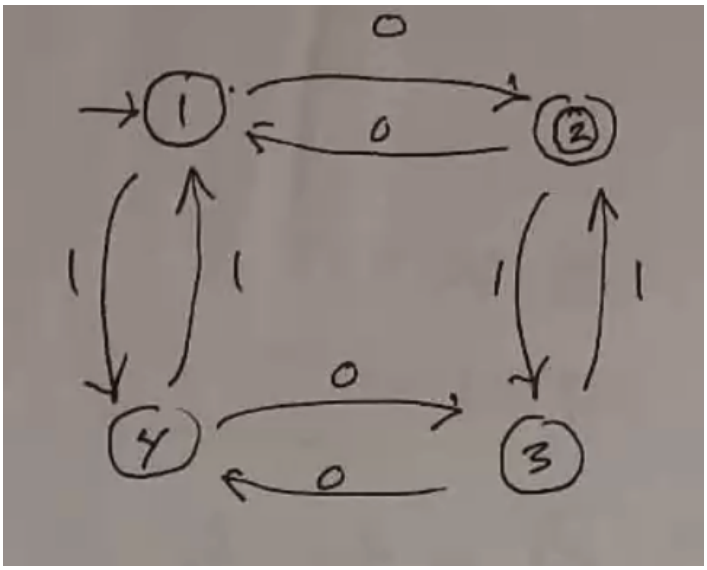
9.4.1 DFA Relation

Since the linear constructions generate the regular languages, they must have an associated DFA:

$M = (Q, \Sigma, \delta, s, F)$

- Make a variable for each state: $V = \{V_i | q_i \in Q\}$
- $\Sigma = \Sigma$
- **For each** $\delta(q_i, a) \to q_j$ **where** $a \in \Sigma$**:**
  - make a rule $V_i \to aV_j$
- **For each** $q_i \in F$**:**
  - make a rule $V_i \to \epsilon$
- $S = V_0$

Ex. Use this method on the language of an odd number of 0s, even number of 1s.




```
V1 := 0 V2 | 1 V4
V2 := 0 V1 | 1 V3 | e
V3 := 0 V4 | 1 V2
V4 := 0 V3 | 1 V1
```

9.5 Closure

Given:

- $G_1 = (N_1, \Sigma, P_1, S_1)$
- $G_2 = (N_2, \Sigma, P_2, S_2)$

9.5.1 Union

- $G = (N, \Sigma, P, S)$
- $N = N_1 \cup N_2 \cup \{S\}$
- $P = P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}$
- $S = S$

9.5.2 Concatenation

- $G = (N, \Sigma, P, S)$
- $N = N_1 \cup N_2 \cup \{S\}$
- $P = P_1 \cup P_2 \cup \{S \to S_1 S_2\}$
- $S = S$

9.5.3 Kleene Star

(Examining G_1^*)

- $G = (N, \Sigma, P, S)$
- $N = N_1 \cup \{S\}$
- $P = P_1 \cup \{S \to \epsilon, S \to S_1 S\}$
- $S = S$

9.5.4 Intersection (Not)

CFLs are *not* closed under intersection! Counterexample:

- $a^n b^n c^n$ is not a CFL
- $\{a^m b^m c^n | m, n \geq 0\}$ **is a CFL**
  - Pf: See CFG 1 below
- $\{a^m b^n c^n | m, n \geq 0\}$ **is a CFL**
  - Pf: See CFG 2 below
- $\{a^m b^n c^n | m, n \geq 0\} \cap \{a^m b^m c^n | m, n \geq 0\} = a^n b^n c^n$ which is not a CFL.

```
CFG 1
S := A B
A := a A b | e
B := c B   | e

CFG 2
S := A B
A := a A   | e
B := b B c | e
```

9.6 Chomsky Normal Form

A special form where all rules are of the form `A := BC | a` (where $A, B, C \in N$ and $a \in \Sigma$).

Thm. For any CFG G, there is a CFG G' in CNF s.t. $L(G') = L(G) - \epsilon$.

9.6.1 Converting to CNF

Step 1: Get rid of epsilon-rules and unit-rules

Converting a CFG to Chomsky Normal Form

Given a CFG, $G = (N, \Sigma, P, S)$, construct a new grammar, $\hat{G} = (N, \Sigma, \hat{P}, S)$ by recursive adding productions to P to form $\hat{P}$, the smallest set of productions containing P that is clos under the following rules:

1. If $A \to \alpha B \beta$ and $B \to \epsilon$ are in $\hat{P}$, then $A \to \alpha\beta$ is in $\hat{P}$.
2. If $A \to B$ and $B \to \gamma$ are in $\hat{P}$, then $A \to \gamma$ is in $\hat{P}$.

After $\hat{P}$ has had enough productions added to be closed under the above rules, remove ϵ-productions and all unit productions.

Step 2: Make everything either go to one terminal or multiple nonterminals

Next for each terminal $a \in \Sigma$, that occurs as part (not all) of a string on the righthand side a production, add a new non-terminal V_a and a new production $V_a \to a$. Then replace all 'a's i those productions with V_a.

Now all productions are of the form:

$$A \to a \quad \text{or} \quad A \to B_1 B_2 B_3 \dots B_k$$

where the B_i are all non-terminals and $k \geq 2$.

Step 3: Make every run of 2+ nonterminals just 2 nonterminals.

Finally do the following until all productions have righthand sides of length 2 or less. Repla productions of the form $A \to B_1 B_2 B_3 \dots B_k$ with two new productions, namely, $A \to B_1 A_1$ a $A_1 \to B_2 B_3 \dots B_k$ where A_1 is a new non-terminal.

Ex.

$L = \{a^n b^n | n \geq 1\}$

```
0.
S := a S b | e

1. add S := a b
S := a S b | e | a b

2. Remove e-productions
S := a S b | a b

3. Give each terminal a nonterminal
S := a S b | a b
A := a
B := b

4. Replace nonterminals on the right
S := A S B | A B
A := a
```

(continues on next page)

(continued from previous page)

```
B := b

5. Replace >2 nonterminals
S  := A S2 | A B
S2 := S B
A  := a
B  := b
```

Ex.

Balanced parentheses.

```
0.
S := ( S ) | S S | e

1. add S := ( ) and remove epsilon
S := ( S ) | S S | ( )

2. make each terminal a nonterminal
S := A S B | S S | A B
A := (
B := )

3. Replace runs of >2 nonterminals
S  := A S2 | S S | A B
S2 := S B
A  := (
B  := )
```

A more complex example can be found on Canvas.

9.7 Pumping Lemma for CFLs

For every CFL L, there exists a $p \geq 0$ s.t. every $z \in L$ of length $\geq p$ can be divided into *five* substrings $z = uvwxy$ such that $vx \neq \epsilon$ and $|vwx| \leq p$, and for all $i \geq 0$, $uv^iwx^iy \in L$.

In English: every sufficiently long string can be divided into 5 segements such that the middle 3 are not too long, and the second and fourth are both not empty, and no matter how much you pump the 2nd and 4th (simultaneously), the string stays in the language.

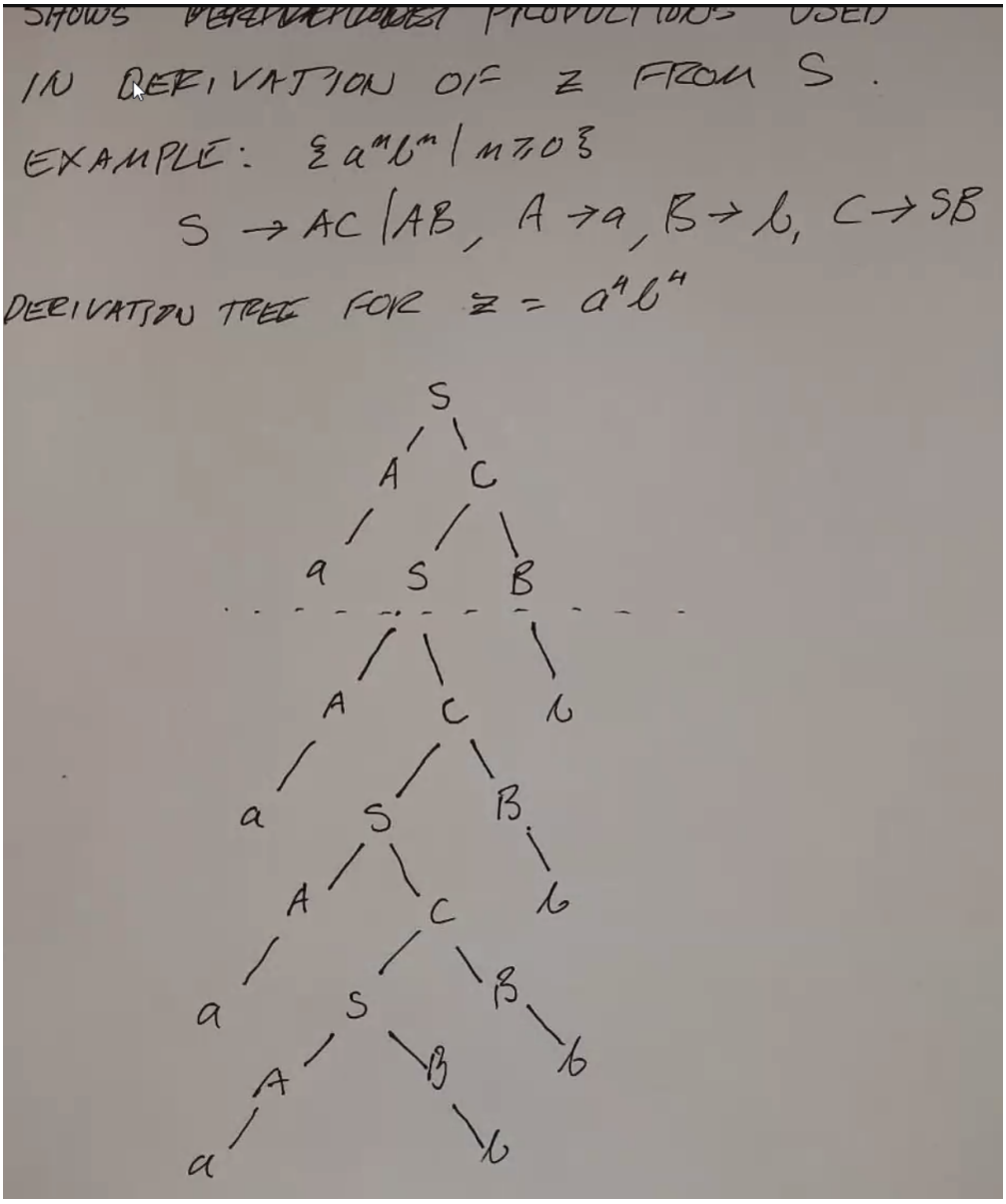
We can use this to prove that a language is not a CFL. We just need to show:

- for all $p \geq 0$
- there exists $s \in L$ of length at least p
- such that for all $uvwxy$ with $z = uvwxy, vx \neq \epsilon, |vwx| \leq p$
- there exists i
- such that $uv^iwx^iy \notin L$.

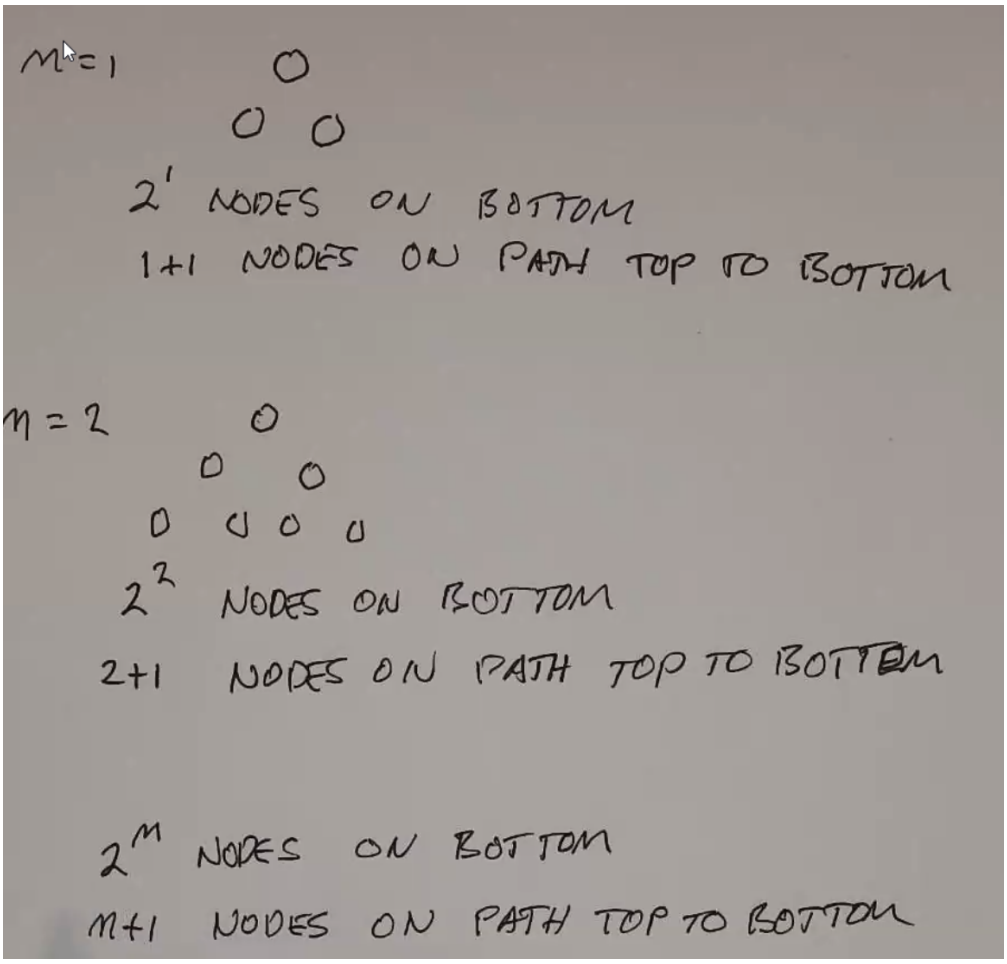
We can use the adversary game again.

9.7.1 Proof

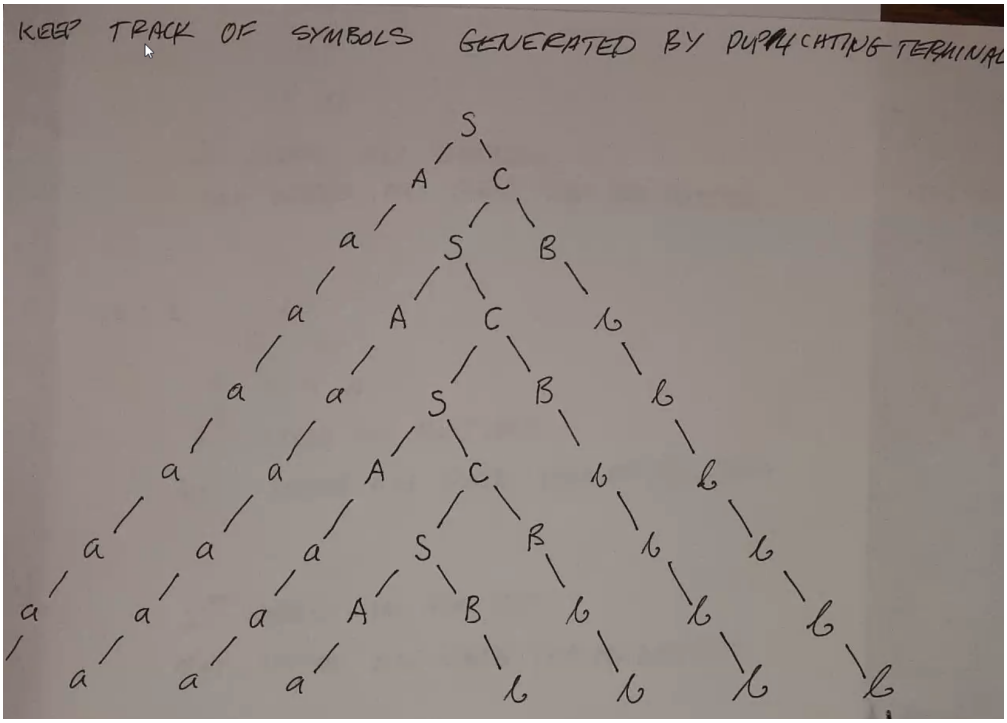
For any CFG, you can make a Chomsky derivation tree:



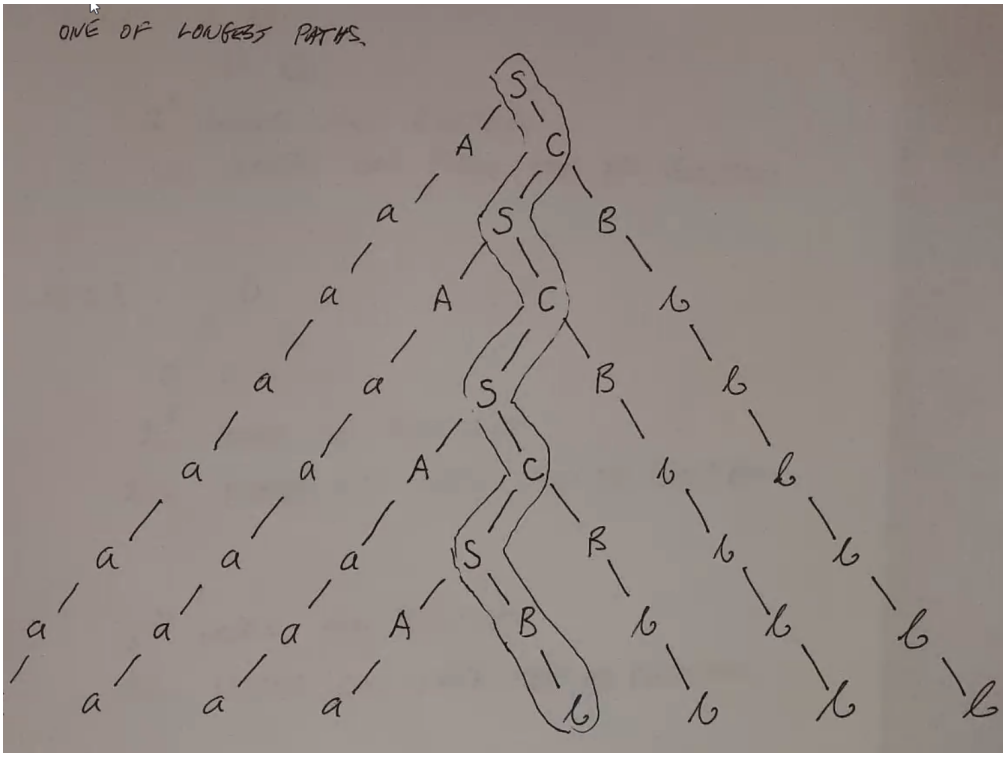
Each level may only double the number of nodes at most, since CNF only allows each nonterminal to go to 2 nodes.



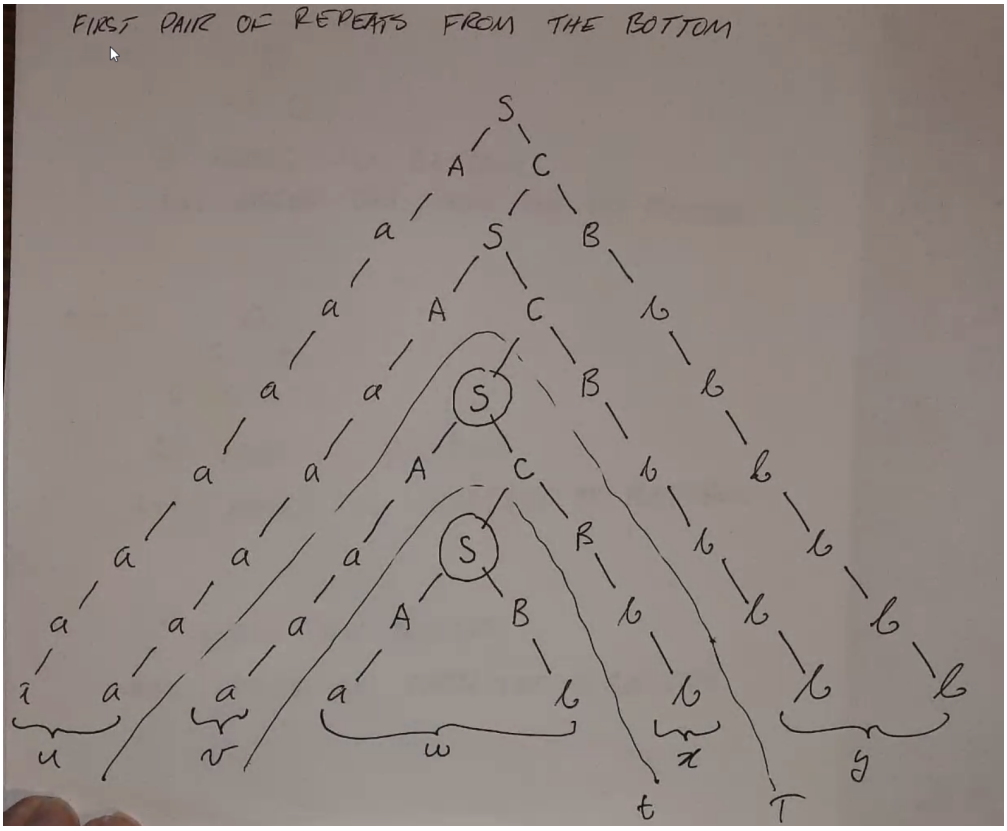
For a derivation tree with 2^n nodes on the bottom, the shortest path to the top must be at least $n + 1$ long.



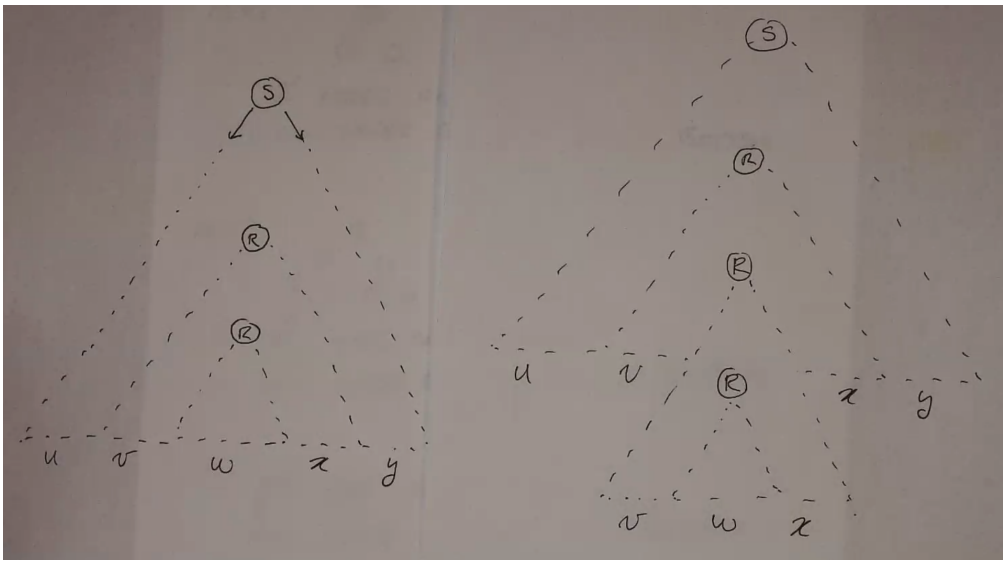
So for a grammar with n variables, the shortest path to the top must be at least $n + 1$ long.



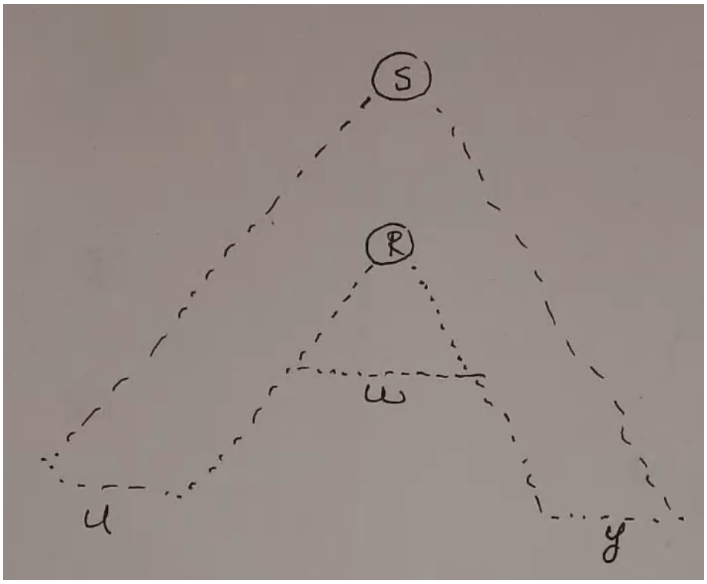
Which means, by the pigeonhole principle, at least 1 variable must be repeated along that path - you can pump along the first repeated variable:



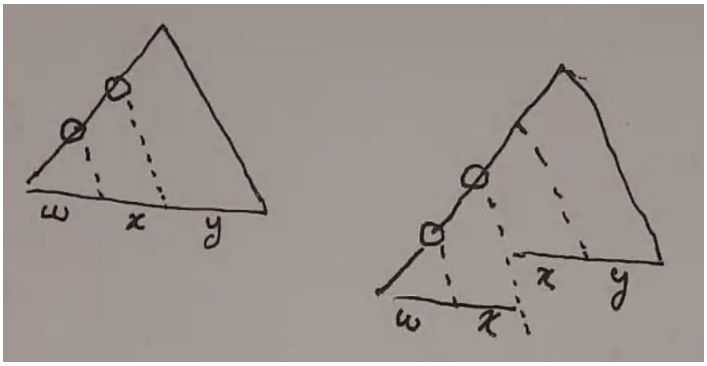
In this diagram, pumping v and x is roughly equivalent to removing the w portion and replacing its child with another of itself.



You can repeat this process to keep pumping. Alternatively, you can pump down by removing the v and x portions:



If either *v* or *x* is empty, it looks like this:

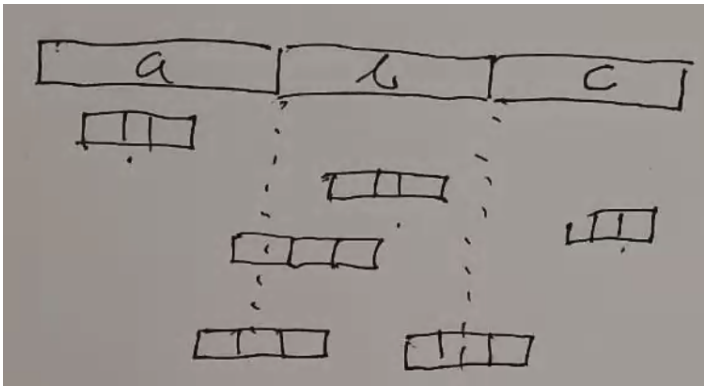


9.7.2 Ex 1

$L = \{a^n b^n c^n | n \geq 0\}$

- p
- $s = a^p b^p c^p$, $s \in L, |s| = 3p > p$.
- $uvwxy = a^p b^p c^p$, $|vwx| \leq p, vx \neq \epsilon$
- Let $i = 2$
- **Case 1:** vwx **is entirely contained within** $a^p, b^p,$ **or** c^p**.**
  - Either *v*, *x*, or both must be made entirely of a's, b's, or c's
  - Pumping that fragment results in more of that letter than the others, and the resulting string is not in the language
- **Case 2: Either *v* or *x* falls on a boundary between letters**
  - Pumping that fragment would result in a mixture of letters, and the resulting string is not in the language
- **Case 3: *v* and *x* are made of two different letters (*w* falls on the boundary)**

  - Pumping those fragments would result in less of the letter that did not contain *v* or *x*, and the resulting string is not in the language.



9.7.3 Ex 2

$L = \{a^n b^n a^n | n \geq 0\}$

- p
- $s = a^p b^p a^p$: $s \in L, |s| = 3p > p$.
- $uvwxy = a^p b^p a^p$: $|vwx| \leq p, vx \neq \epsilon$
- Let $i = 2$
- **Case 1:** vwx **is entirely contained within** a^p **or** b^p
  - Either *v*, *x*, or both must be made entirely of a's or b's
  - Pumping that fragment results in more of that letter than the others, and the resulting string is not in the language
- **Case 2: Either *v* or *x* falls on a boundary between letters**
  - So *v* or *x* contains both a's and b's
  - Pumping that fragment would result in a mixture of letters, and the resulting string is not in the language
- **Case 3: *v* and *x* are made of two different letters (*w* falls on the boundary)**
  - Pumping those fragments would result in less a's in the section that did not contain *v* or *x*, and the resulting string is not in the language.

9.7.4 Ex 3

$\Sigma = \{0, 1\}; L = \{ww | w \in \Sigma^*\}$

- p
- $s = 0^p 1^p 0^p 1^p$: $s \in L, |s| = 4p > p$.
- $uvwxy = 0^p 1^p 0^p 1^p$: $|vwx| \leq p, vx \neq \epsilon$
- **Case 1:** vwx **is entirely on one half of the string**
  - Let $i = 2$

    - **By pumping, at least one symbol on the boundary is pushed over the boundary to the other half**
        - Which means that either the second half would start with 1 (but the first starts with 0!)
        - Or the first half ends with 0 (but the second ends with 1!)
    - the resulting string is not in the language.
- **Case 2: *v* and *x* are on opposite halves (*w* is on the boundary)**
    - let $i = 0$
    - The new string becomes $0^p1^a0^b1^p$
    - Both *a* and *b* cannot both be *p*, since either $a < p$ or $b < p$
    - So the resulting string is not in the language

(. . . what about the boundary?)

9.7.5 Ex 4

(same language as ex 3)

$\Sigma = \{a, b\}; D = \{ww | w \in \Sigma^*\}$

Important note: CFL ∩ CFL is not necessarily a CFL, but CFL ∩ regular language is

Consider $D' = D \cap L(a^*b^*a^*b^*) = \{a^nb^ma^nb^m | n, m \geq 0\}$

- Assume *D* is a CFL
- then *D'* must be (since the intersection of a CFL and a regular language is a CFL)
- however *D'* is not a CFL:
- p
- $s = a^pb^pa^pb^p$: $s \in L, |s| = 4p > p$.
- $uvwxy = a^pb^pa^pb^p$: $|vwx| \leq p, vx \neq \epsilon$
- Let $i = 2$
- **Case 1: *v* or *x* crosses a boundary between a's and b's**
    - By pumping this fragment, the resulting string introduces a mixture
    - no longer of the form $a^nb^ma^nb^m$
- **Case 2: vwx is entirely contained within one section of a's or b's**
    - By pumping this fragment, this changes the number of a's or b's on only one side
    - so the string is not longer in the language
- **Case 3: *v* and *x* are made entirely of different letters (*w* contains the boundary)**
    - **Case 1: the boundary is the middle of the left section**
        - By pumping this fragment, this changes the number of a's, b's, or both on only one side
        - so the string is not longer in the language
    - **Case 2: the boundary is the center**
        - By pumping this fragment, this changes the number of b's on the left but not the right, a's on the right but not the left, or both

        - so the string is not longer in the language
- so D is not a CFL.

9.7.6 Ex 5

$\Sigma = \{a, b\}; L = \{x \in \Sigma^* | x \text{ is not of form } ww, w \in \Sigma^*\}$

L *is* a CFL!

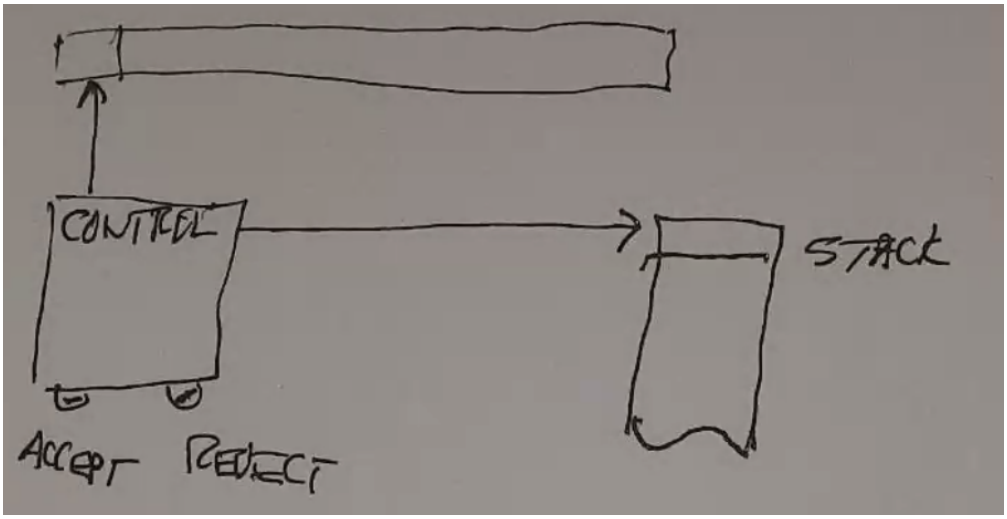
- Odd length strings are certainly not of this form
- **what about even length ones? let $L = a|b$, they must be the form $L^n a L^m L^n b L^m$**
    - the (n+1)th symbol on the left is different from the (n+1)th symbol on the right
    - so at least 1 symbol must not match
    - note: $L^n a L^m L^n b L^m = L^n a L^{m+n} b L^m = L^n a L^n L^m b L^m$

```
S := A | B | A B | B A
A := L A L | a          # odd length strings with a in middle
B := L B L | b          # odd length strings with b in middle
L := a | b
```

CHAPTER 10

Push-Down Automata

PDAs give us a form of memory by introducing a stack, which has infinite space but containing a finite number of elements:



10.1 Formal Definition

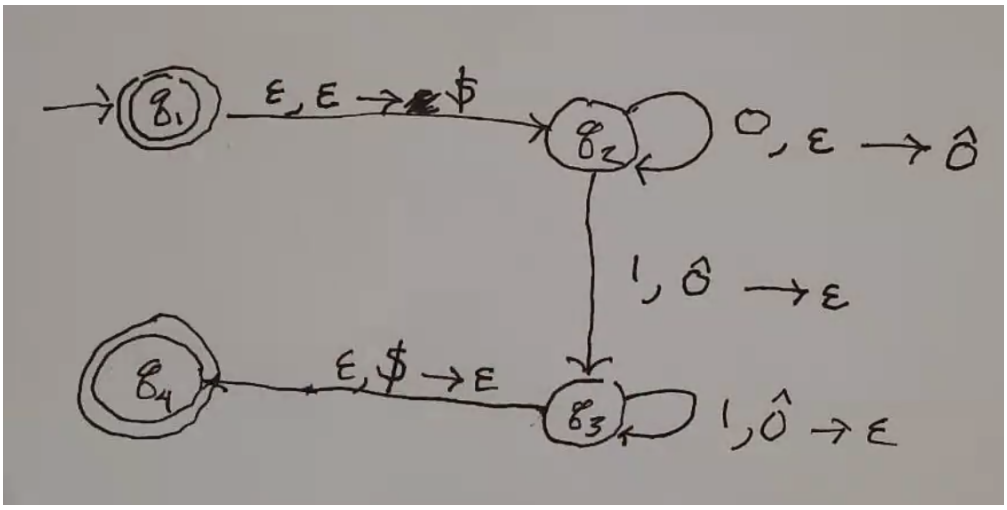
$M = (Q, \Sigma, \Gamma, \delta, s, F)$ where

- Q is a finite set of states
- Σ is a finite set (symbols - input alphabet)
- Γ is a finite set (symbols - stack alphabet)
- $\delta : (Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\})) \to P(Q \times (\Gamma \cup \{\epsilon\}))$
  - Takes a state, maybe something from the input, maybe something from the stack (popped)

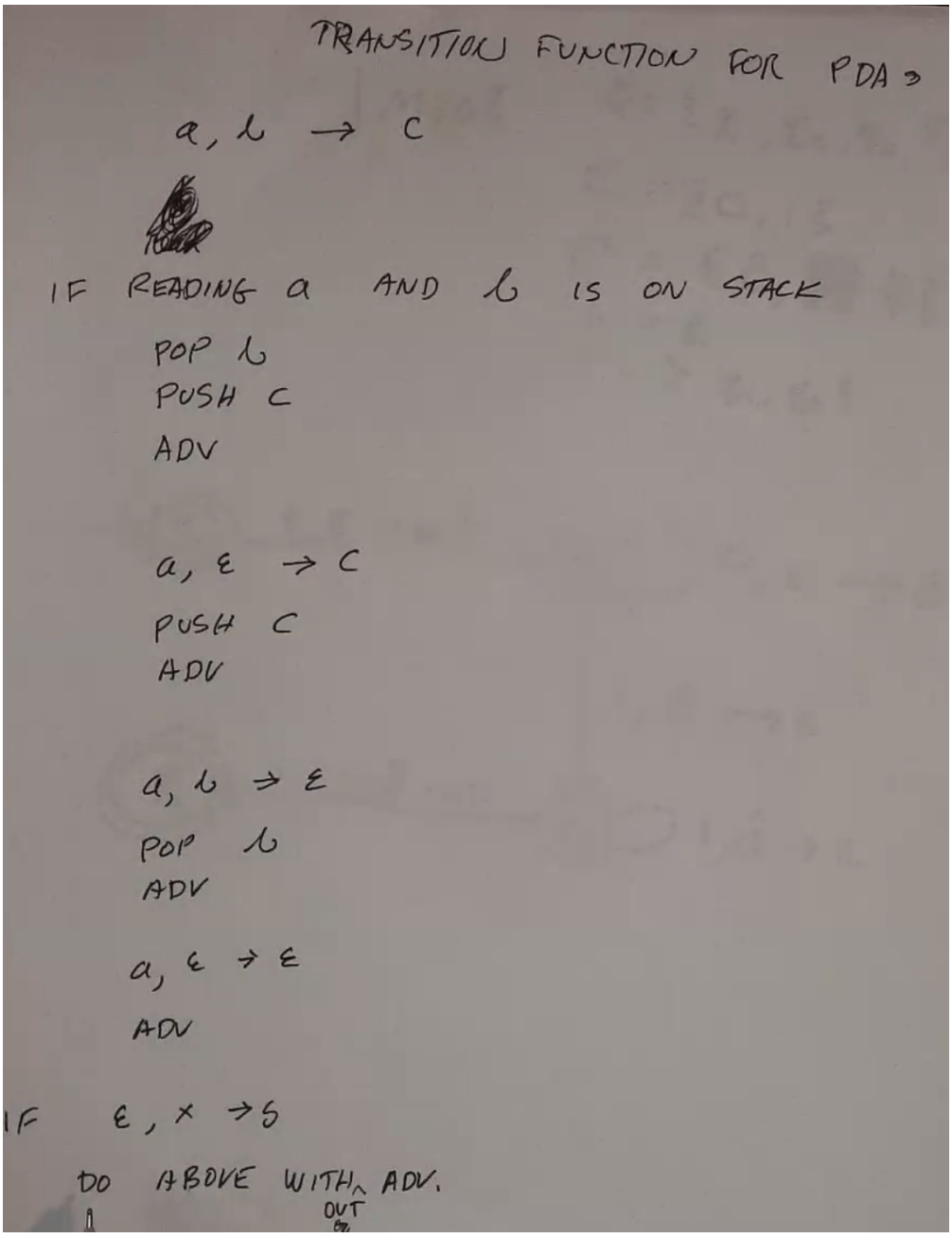
  - Produces some subset of states (non-deterministic) and something to push on stack, optionally
- $s \in Q$ is the start state
- $F \subset Q$ is the accept states

10.1.1 Ex 1

- $L = \{0^n 1^n | n \geq 0\}$
- $Q = \{q_1, q_2, q_3, q_4\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{\hat{0}, \}$
- $s = q_1$
- $F = \{q_1, q_4\}$
- δ is represented by this state diagram:
- Design: Push 0s onto the stack, pop an equal number of 1s.

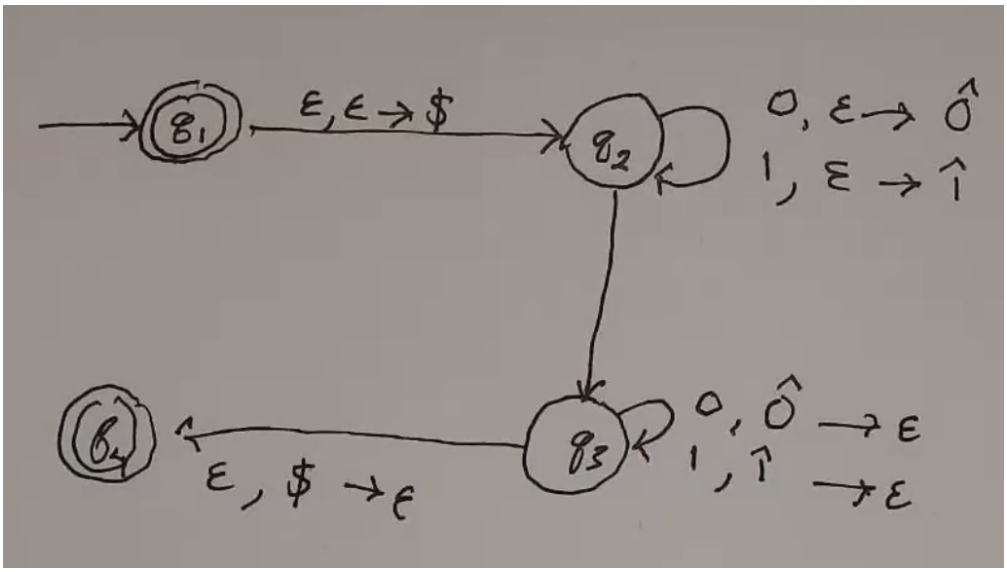


$x, y \to z$ represents whether to read, what to pop, and what to push, if any.



10.1.2 Ex 2

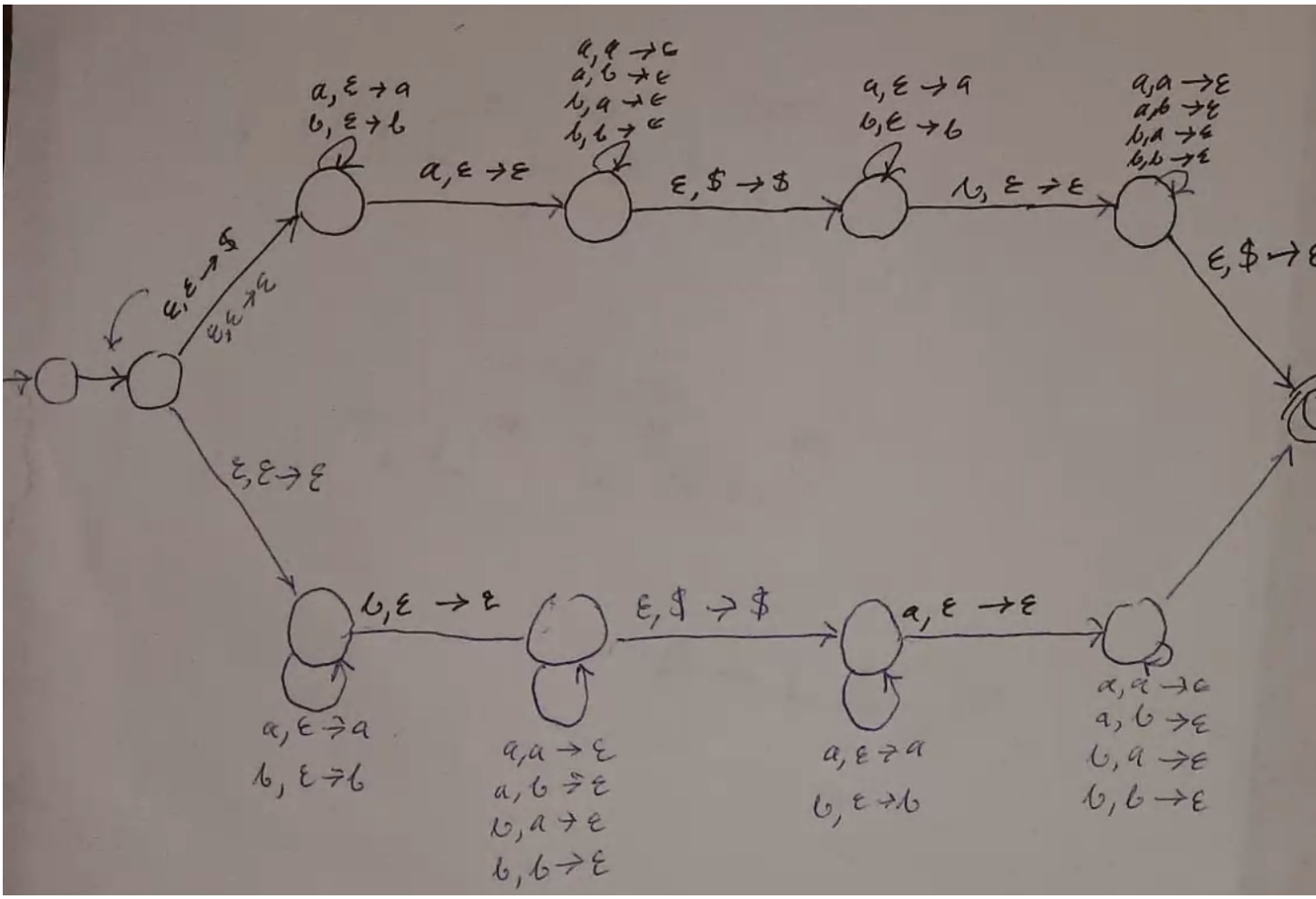
$L = \{ww^R | w \in \{0,1\}^*\}$



Note: The transition $q_2 \to q_3$ is nondeterministic and can be an epsilon move.

10.1.3 Ex 3

The complement of $L = \{ww | w \in \{0,1\}^*\}$ (see CFG ex. 5)



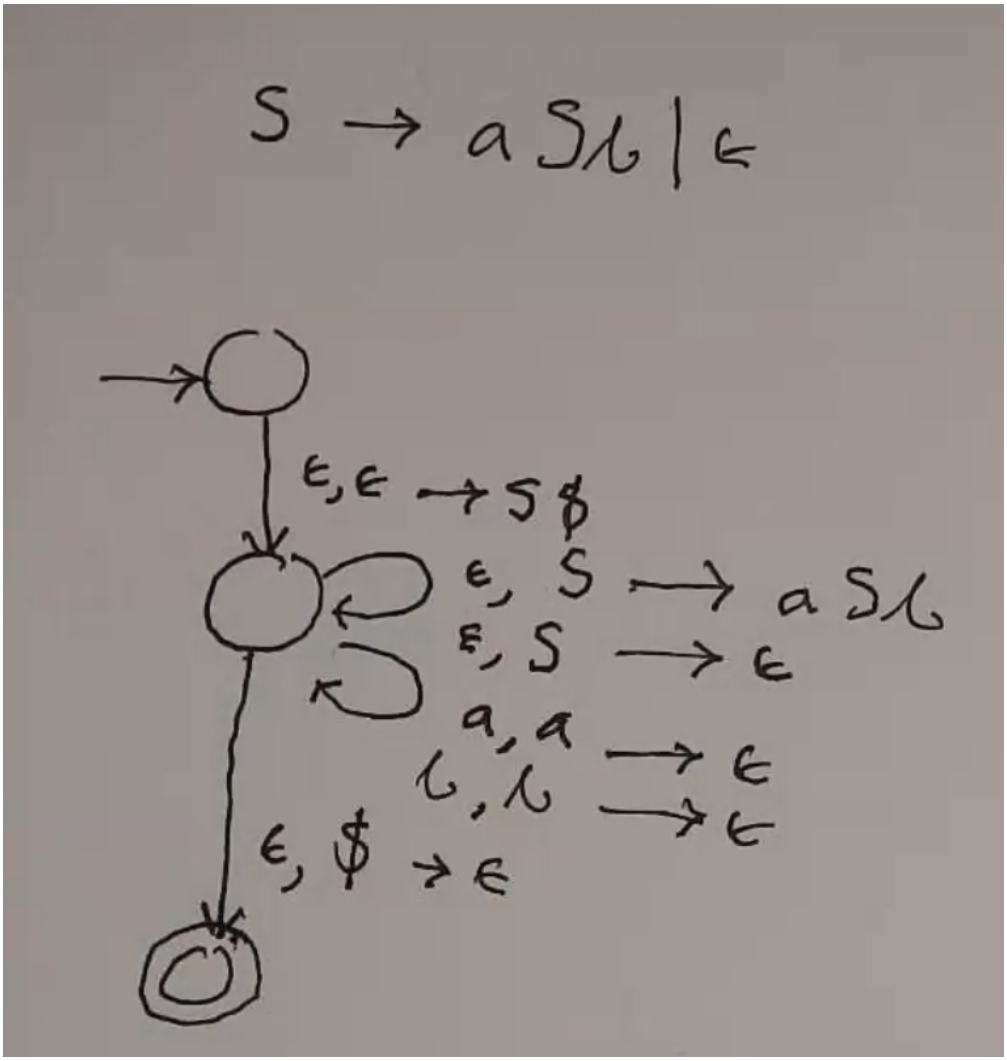
10.2 CFGs

You can make a PDA for any CFG using 3 states:

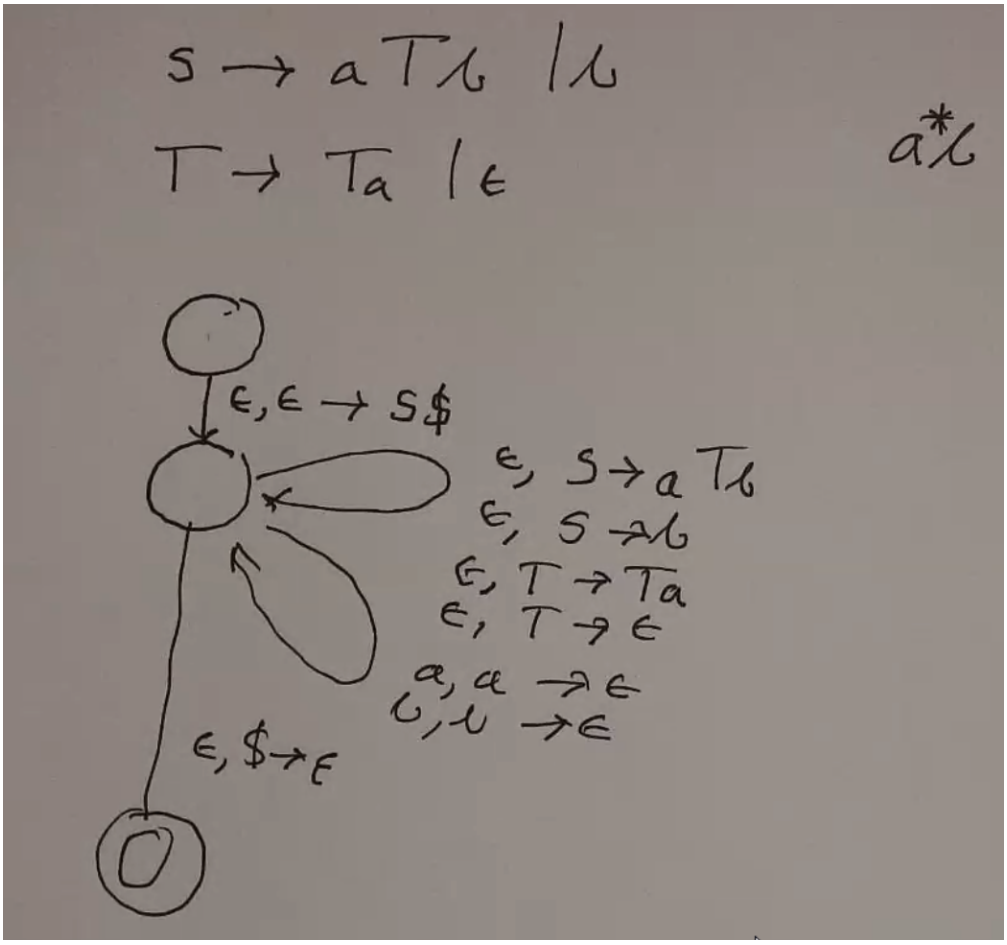
```
Place $ and start in stack
Do repeatedly:
    If var on top:
        Pop it and push right side of rule (*)
    If terminal on top:
        If it matches the stack, advance read head
        If not, fail
    If $ on top:
        Accept
```

Note: $\star$: This means we allow pushing entire strings onto the stack. This can be done character-wise, but it's faster this way.

10.2.1 Ex 1



10.2.2 Ex 2



10.3 CKY Algorithm

Given a CFG, how do you determine whether $x \in L(G)$?

Examine the substrings (grammar should be in CNF):

10.3.1 Ex 1

```
S := AB | BA | SS | AC | BD
A := a
B := b
C := SB
D := SA
```

Is `aabbab` in the language? The table represents ways to get from top to right. Fill in the diagonals (longest first):

```
|a|a|b|b|a|b|
0 1 2 3 4 5 6
```

(continues on next page)

(continued from previous page)

```
+-----+-----+-----+-----+-----+-----+---+
| 0   |     |     |     |     |     |   |
+-----+-----+-----+-----+-----+-----+---+
| {A} | 1   |     |     |     |     |   |
+-----+-----+-----+-----+-----+-----+---+
| {}  | {A} | 2   |     |     |     |   |
+-----+-----+-----+-----+-----+-----+---+
| {}  | {S} | {B} | 3   |     |     |   |
+-----+-----+-----+-----+-----+-----+---+
| {S} | {C} | {}  | {B} | 4   |     |   |
+-----+-----+-----+-----+-----+-----+---+
| {D} | {S} | {}  | {S} | {A} | 5   |   |
+-----+-----+-----+-----+-----+-----+---+
| {S} | {C} | {}  | {C} | {S} | {B} | 6 |
+-----+-----+-----+-----+-----+-----+---+
```

Since it is possible to get from 0 to 6 using the start variable, the string is in the language.

10.3.2 Ex 2

```
S := AB | BC
A := BA | a
B := CC | b
C := AB | a
```

Is `baaba` in the language?

```
|b|a|a|b|a|
0 1 2 3 4 5

+-----------+--------+--------+--------+--------+---+
| 0         | b      |        |        |        |   |
+-----------+--------+--------+--------+--------+---+
| {B}       | 1      | a      |        |        |   |
+-----------+--------+--------+--------+--------+---+
| {A, S}    | {A, C} | 2      | a      |        |   |
+-----------+--------+--------+--------+--------+---+
| {}        | {B}    | {A, C} | 3      | b      |   |
+-----------+--------+--------+--------+--------+---+
| {}        | {B}    | {S, C} | {B}    | 4      | a |
+-----------+--------+--------+--------+--------+---+
| {A, S, C} | {S, A} | {B}    | {A, S} | {A, C} | 5 |
+-----------+--------+--------+--------+--------+---+
```

So the string is in the language.

CHAPTER 11

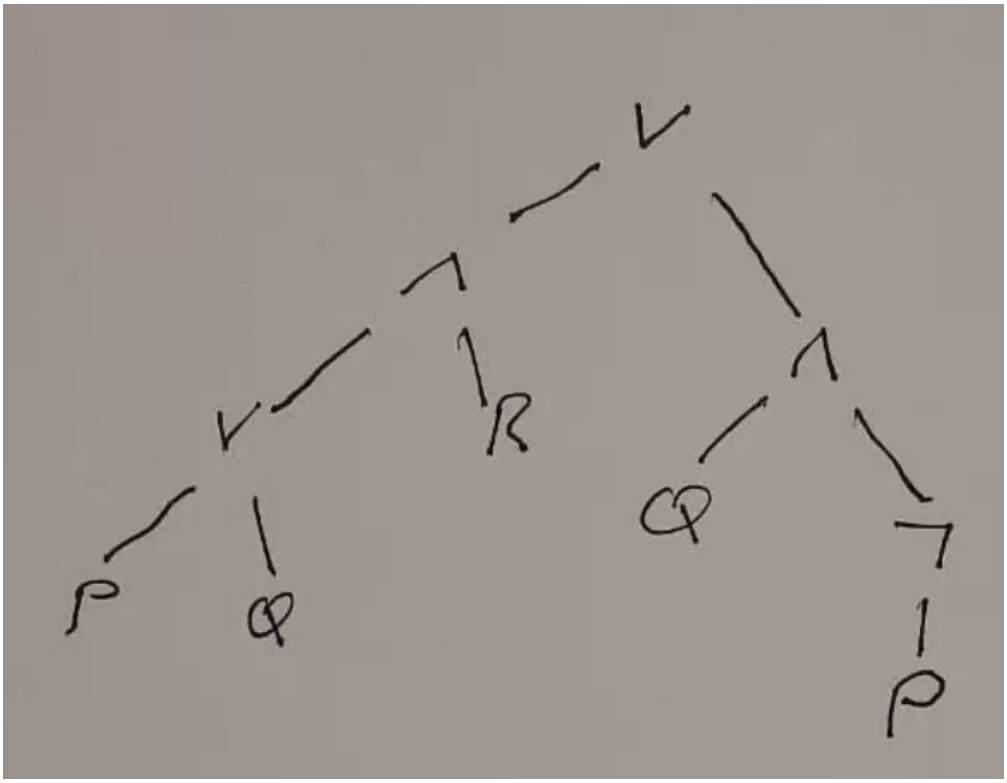
Parsing

Example: Propositional Logic

- Variable: P, Q, R, ...
- Constants: T/F, 0/1, etc.
- Binary operations: $\wedge, \vee, \implies, \iff$
- Unary operations: $\neg$
- Parentheses

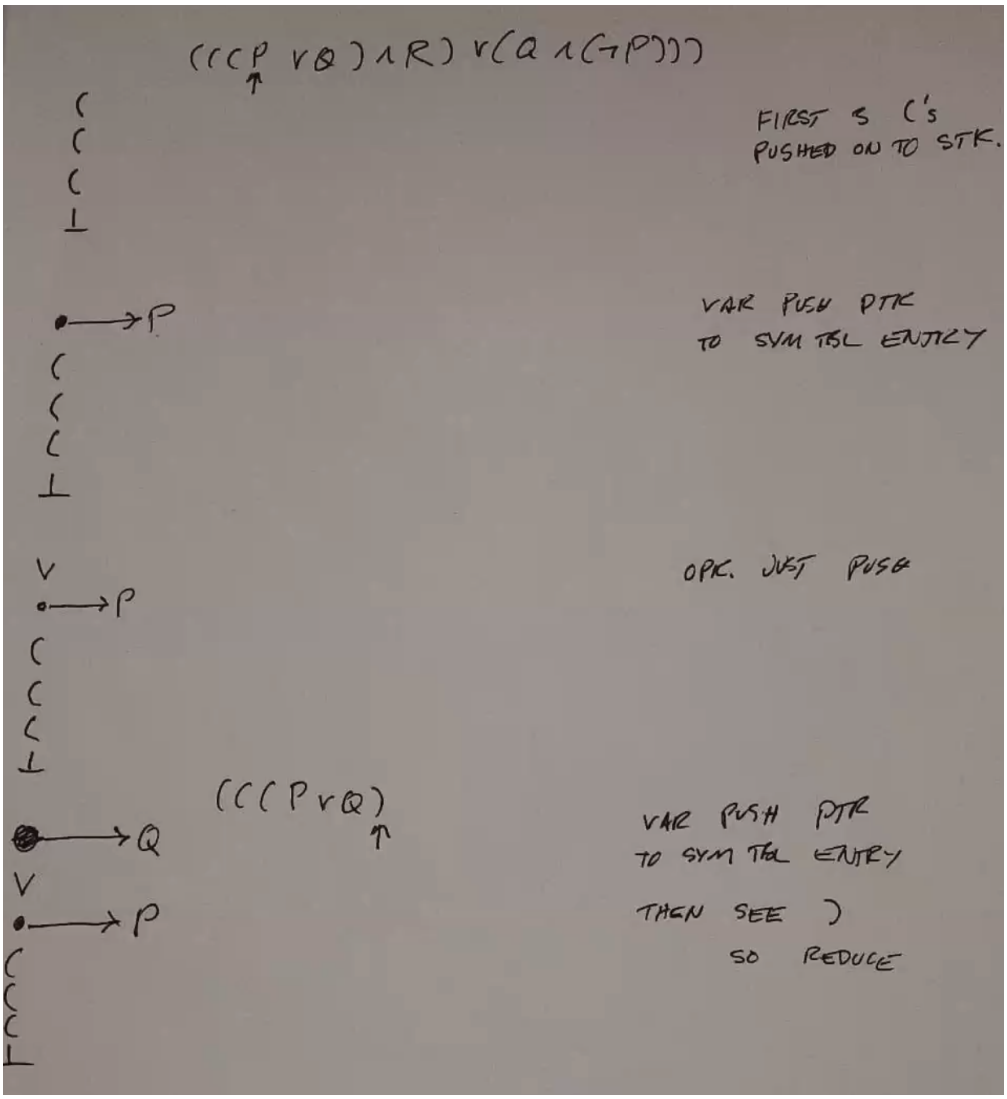
```
E := ( E B E ) | ( U E ) | C | V
B := V | ^ | => | <=>
U := ~
C := 0 | 1
V := P | Q | R | ... | Z
```

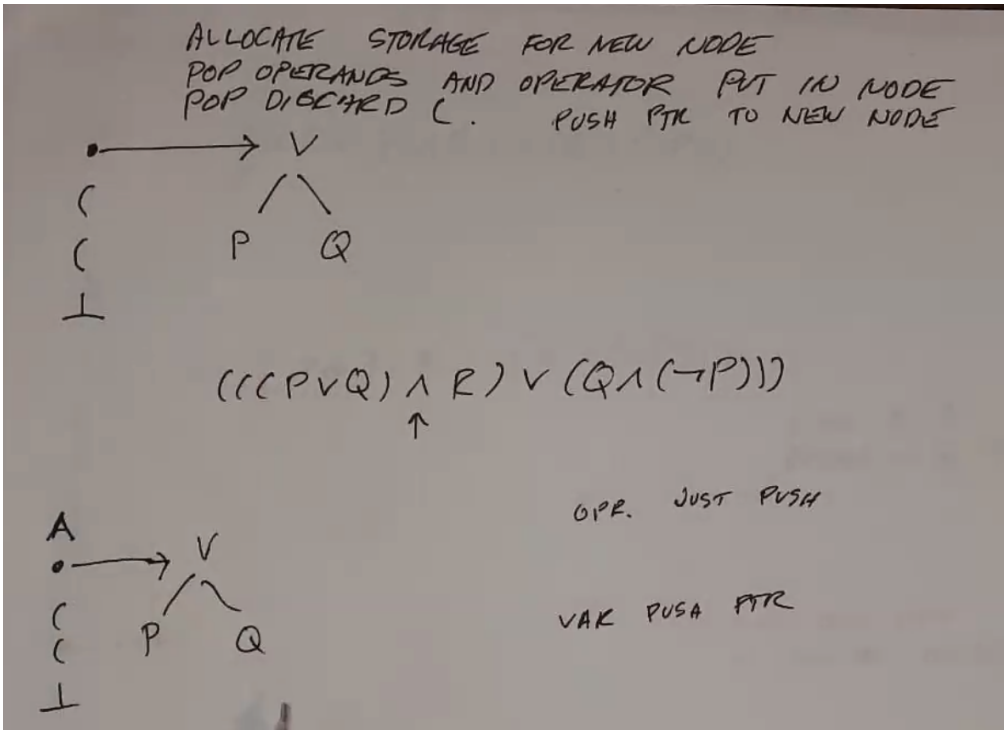
Let's parse: $(((P \vee Q) \wedge R) \vee (Q \wedge (\neg P)))$

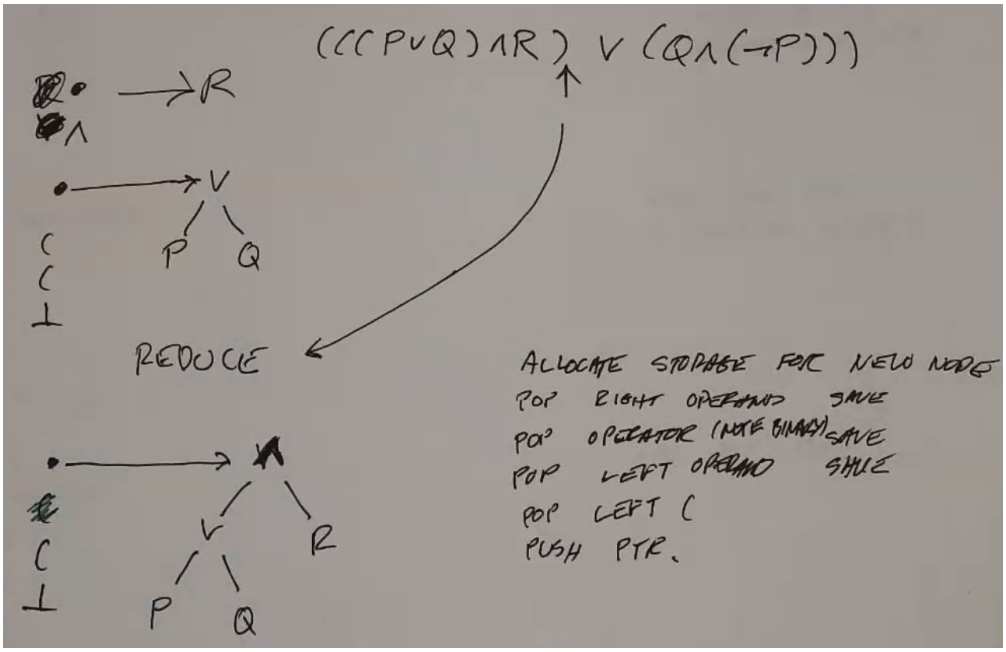


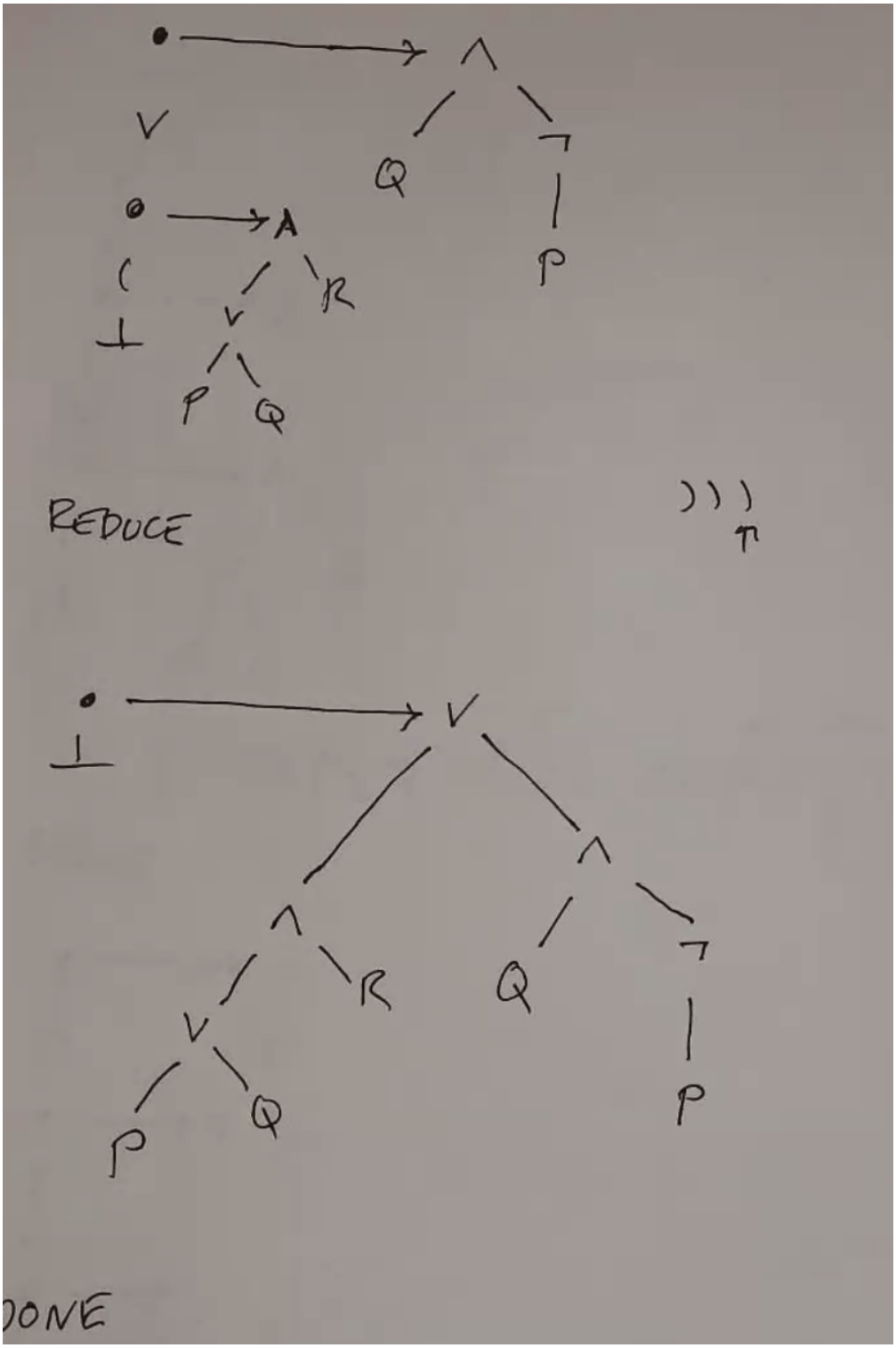
A *simple* parser algorithm (not really robust, or working for other grammars):

- Init stack with $\perp$
- Scan left to right
- If open paren, push
- If operator, push
- If constant, push pointer to it
- If variable, lookup in symbol table, push pointer to it
- **If close paren, reduce:**
  - Allocate storage for node in expression tree
  - Pop object (ptr to right operand, could be const, var, or another node)
  - Pop object (should be the operator), save in node
  - **If operator is binary:**
    - Pop object (ptr to left operand)
  - Pop object (should be left paren)
  - Push pointer to new node


((((P ∨ Q) ∧ R) ∨ (Q ∧ (¬P)))
FIRST 5 (s PUSHED ON TO STK.
VAR PUSH PTR TO SYM TBL ENTRY
OPR. JUST PUSH
(((P ∨ Q)
VAR PUSH PTR TO SYM TBL ENTRY
THEN SEE)
SO REDUCE


ALLOCATE STORAGE FOR NEW NODE
POP OPERANDS AND OPERATOR PUT IN NODE
POP DISCARD (. PUSH PTR TO NEW NODE
V
P Q
(((P∨Q)∧R)∨(Q∧(¬P)))
OPR. JUST PUSH
VAR PUSH PTR
A
V
P Q

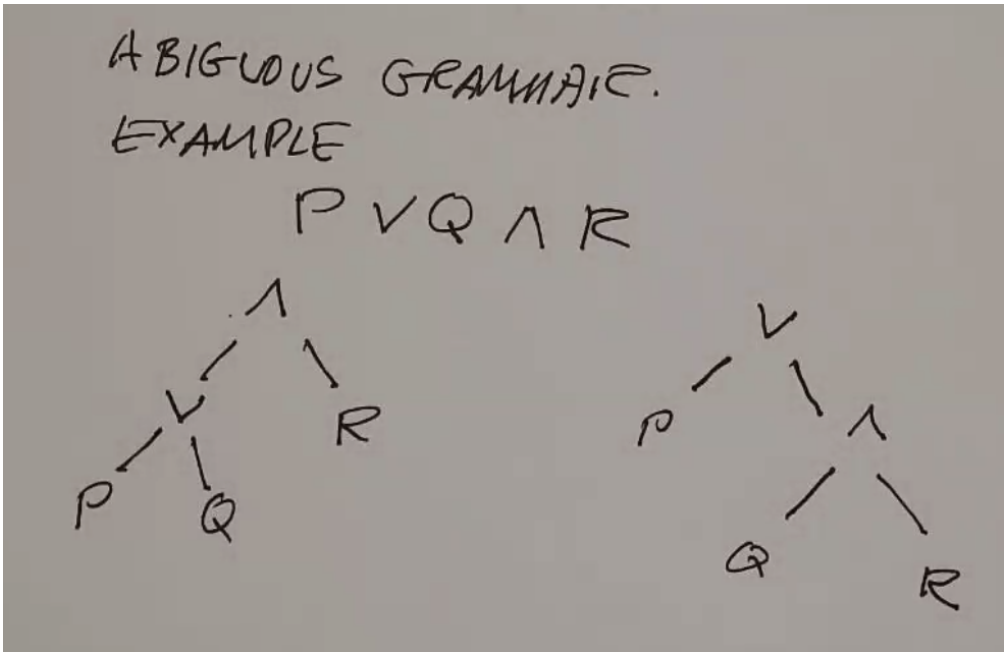

(((P∨Q)∧R)∨(Q∧(¬P)))
R
V
P Q
REDUCE
ALLOCATE STORAGE FOR NEW NODE
POP RIGHT OPERAND SAVE
POP OPERATOR (NOTE BINARY) SAVE
POP LEFT OPERAND SAVE
POP LEFT (
PUSH PTR.
V
R
P Q



11.1 Ambiguity

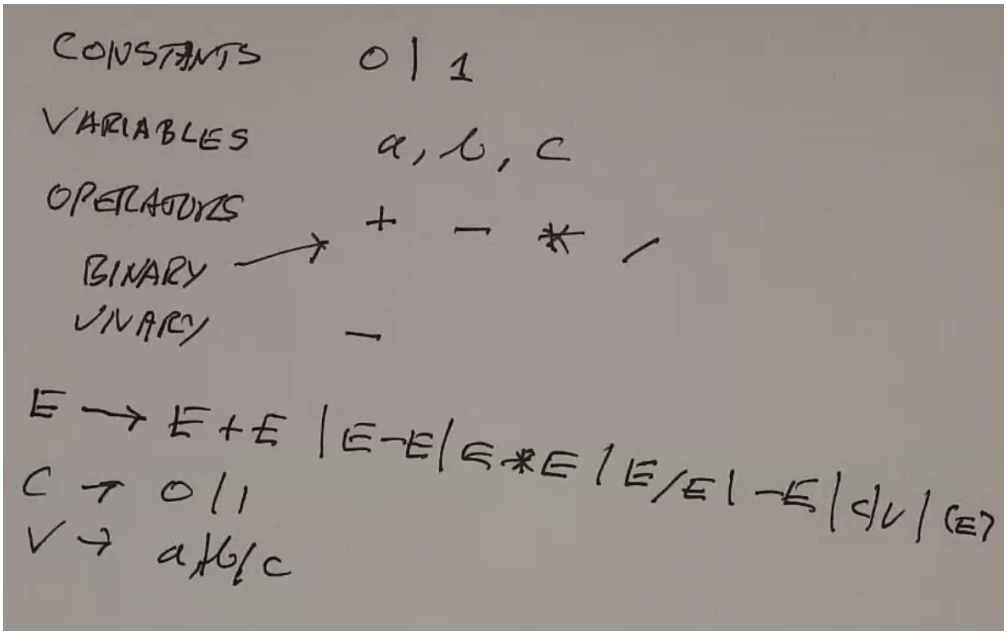
The above example uses parens everywhere so we don't have to worry about order of ops. What if we don't?

```
E := EBE | UE | C | V | (E)
B := V | ^ | => | <=>
U := 0 | 1
V := P | Q | R | ... | Z
```

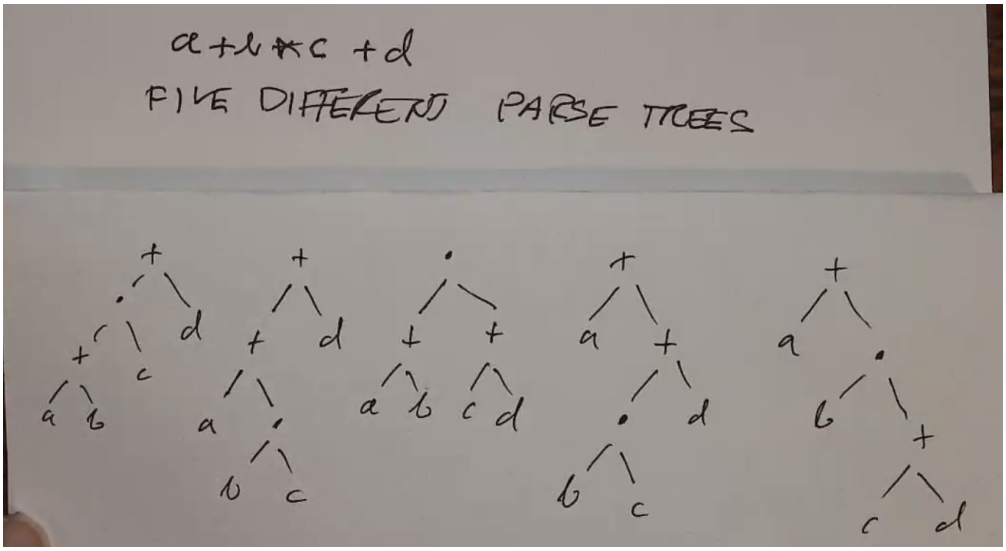


We need to give precedence to some operators. Let's look at a math grammar and apply OoO:

```
E := E+E | E-E | E*E | E/E | -E | C | V | (E)
C := 0 | 1
V := a | b | c
```



There's a lot of ambiguity without precedence:



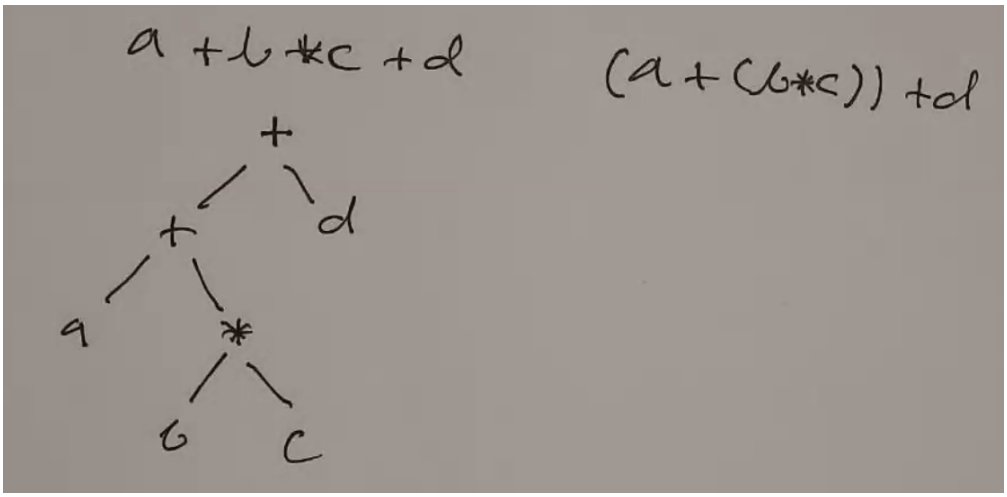
11.1.1 Solution 1: Design

Design the grammar so that there is no ambiguity.

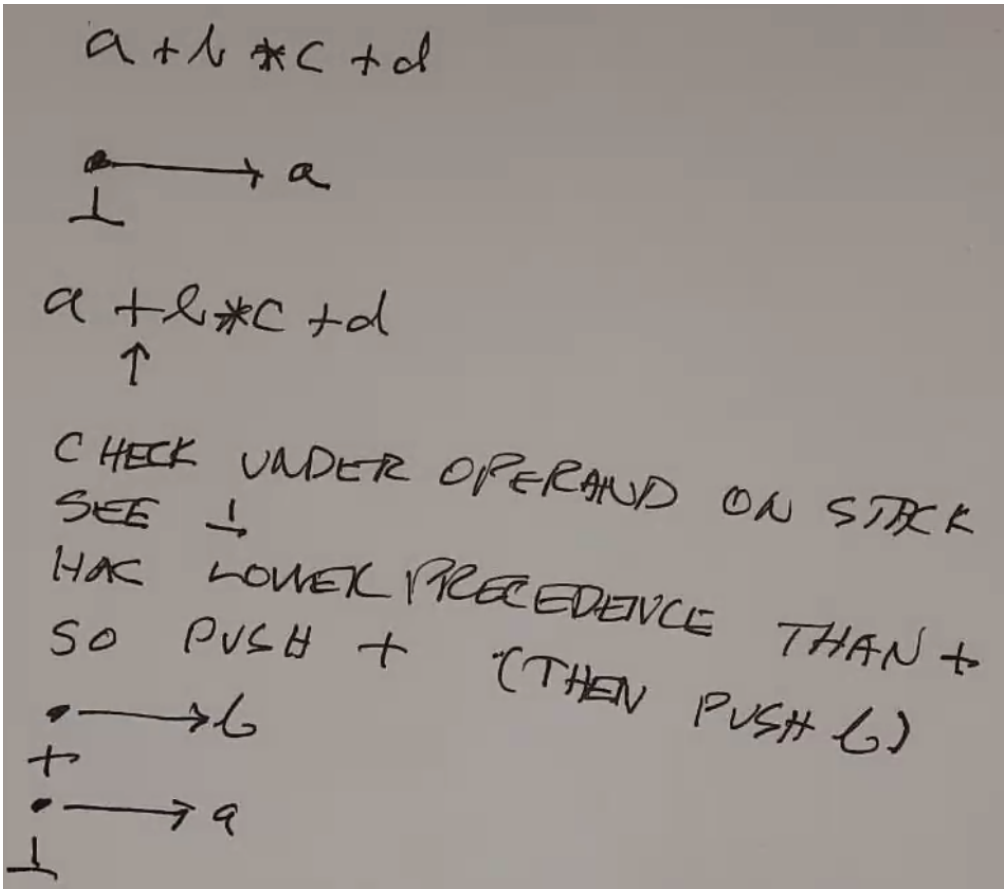
```
E := E+F | E-F | F
F := F*G | F/G | G
G := -G | H
H := C | V | (E)
C := 0 | 1
V := a | b | c
```

This gives the following precedence:

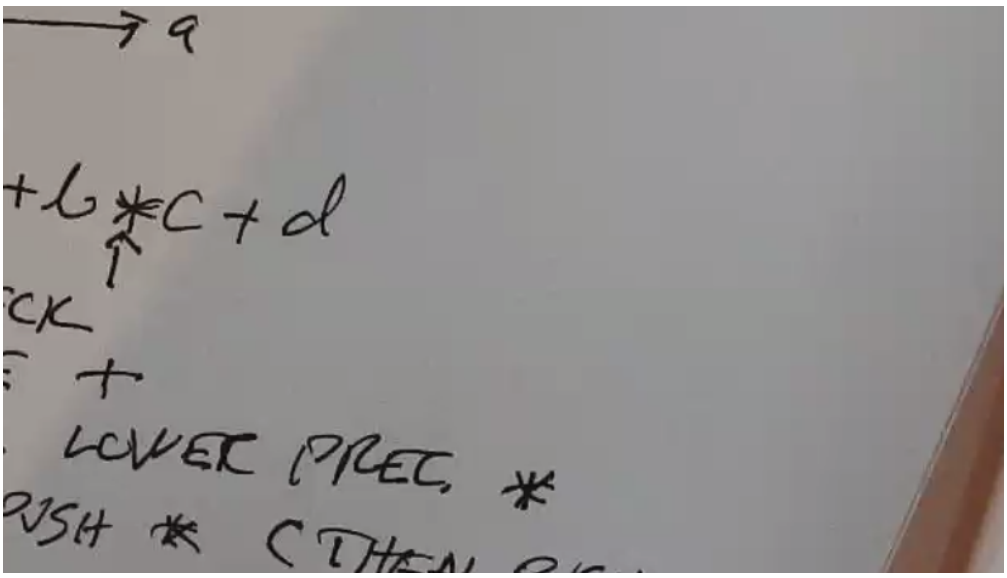
1. unary -
2. $*$, /
3. $+$, $-$
4. (
5. $\perp$



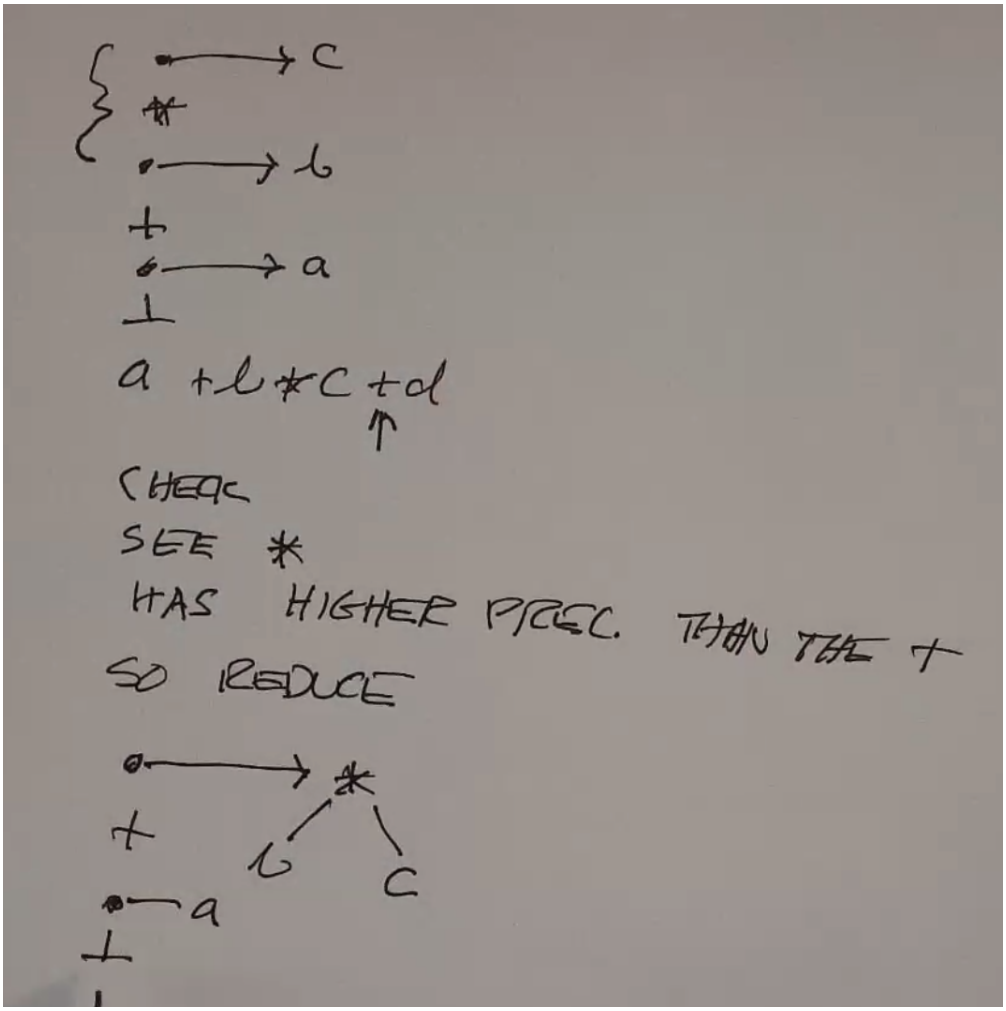
This changes our parsing algorithm to check the operator under the operand on the stack when it reaches an operator - reduce if it has higher or equal precedence, push if not:



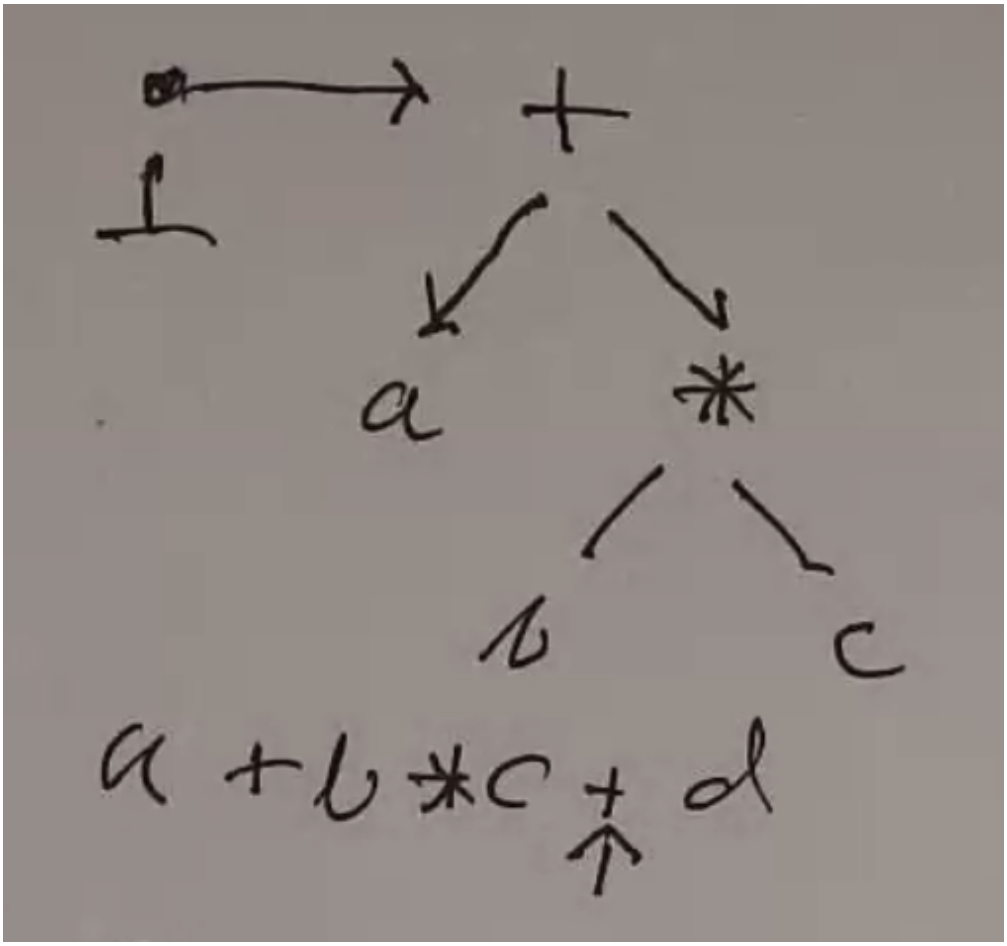
Ok, this screenshot kind of failed, but same thing: push $\star$ then $\circ$:



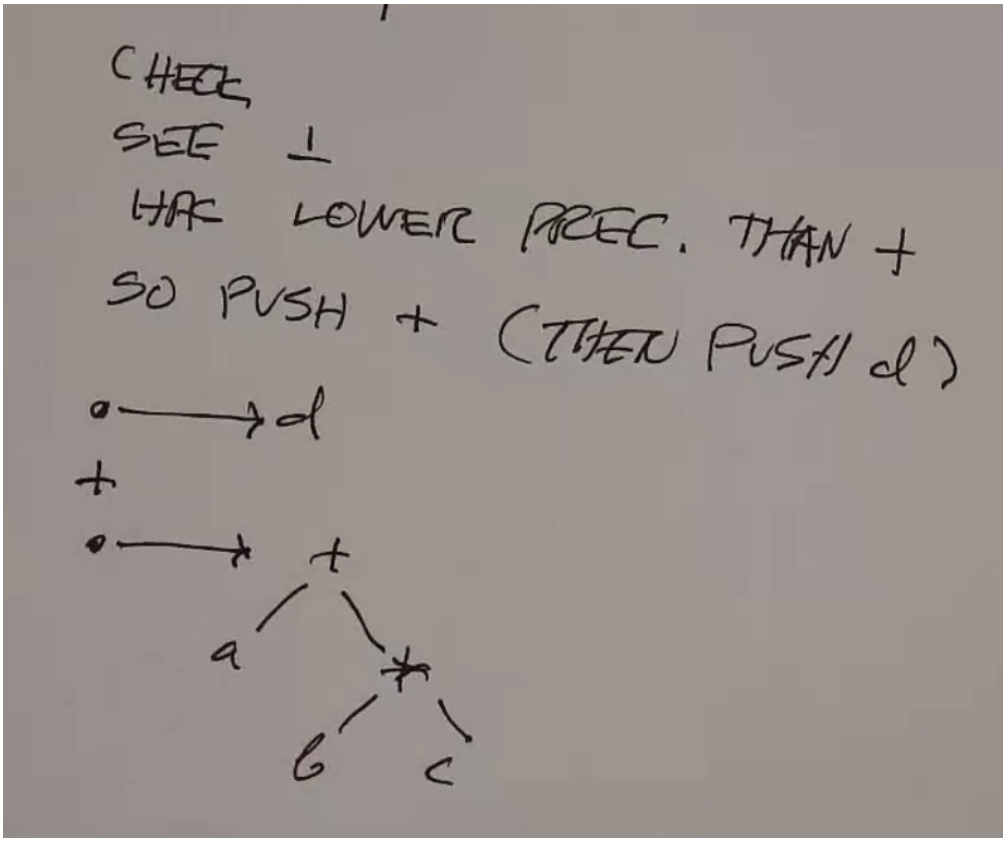
Now, when we parse the next operator, we see that the operator below ($\star$) has higher precedence: so we need to reduce!



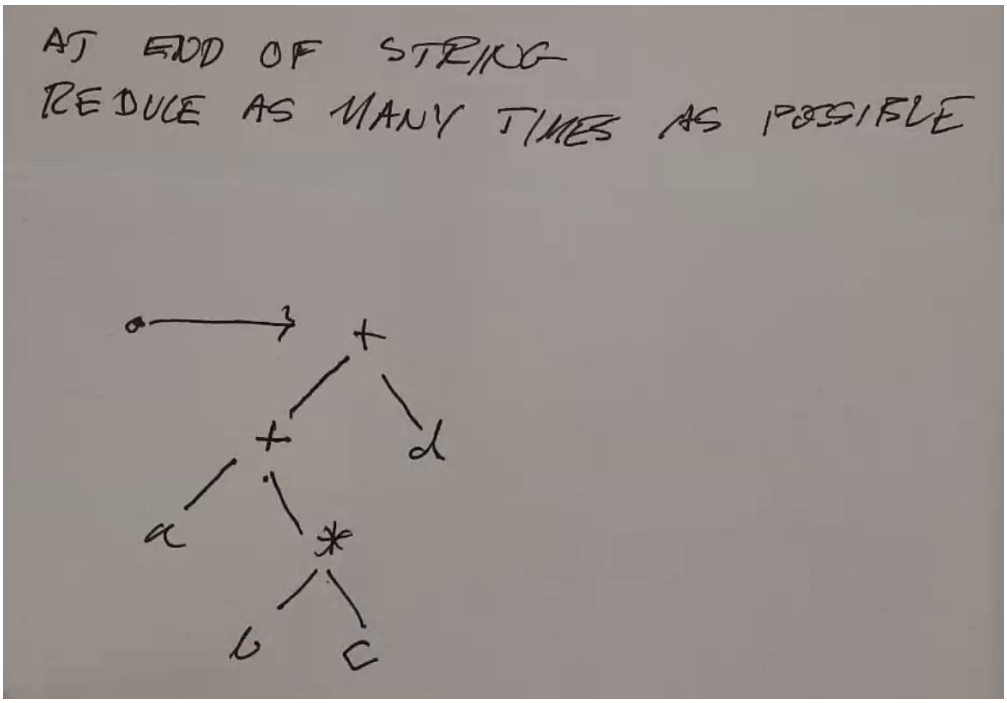
Repeat, now the operand underneath is +, which has equal precedence - so we reduce again (because we're parsing left to right)



And check again, the bottom of the stack has super low precedence so we're fine. Push.



Now we're at the end of the string, so reduce as much as possible (in this case 1 reduce):



CHAPTER 12

Turing Machines

Related:

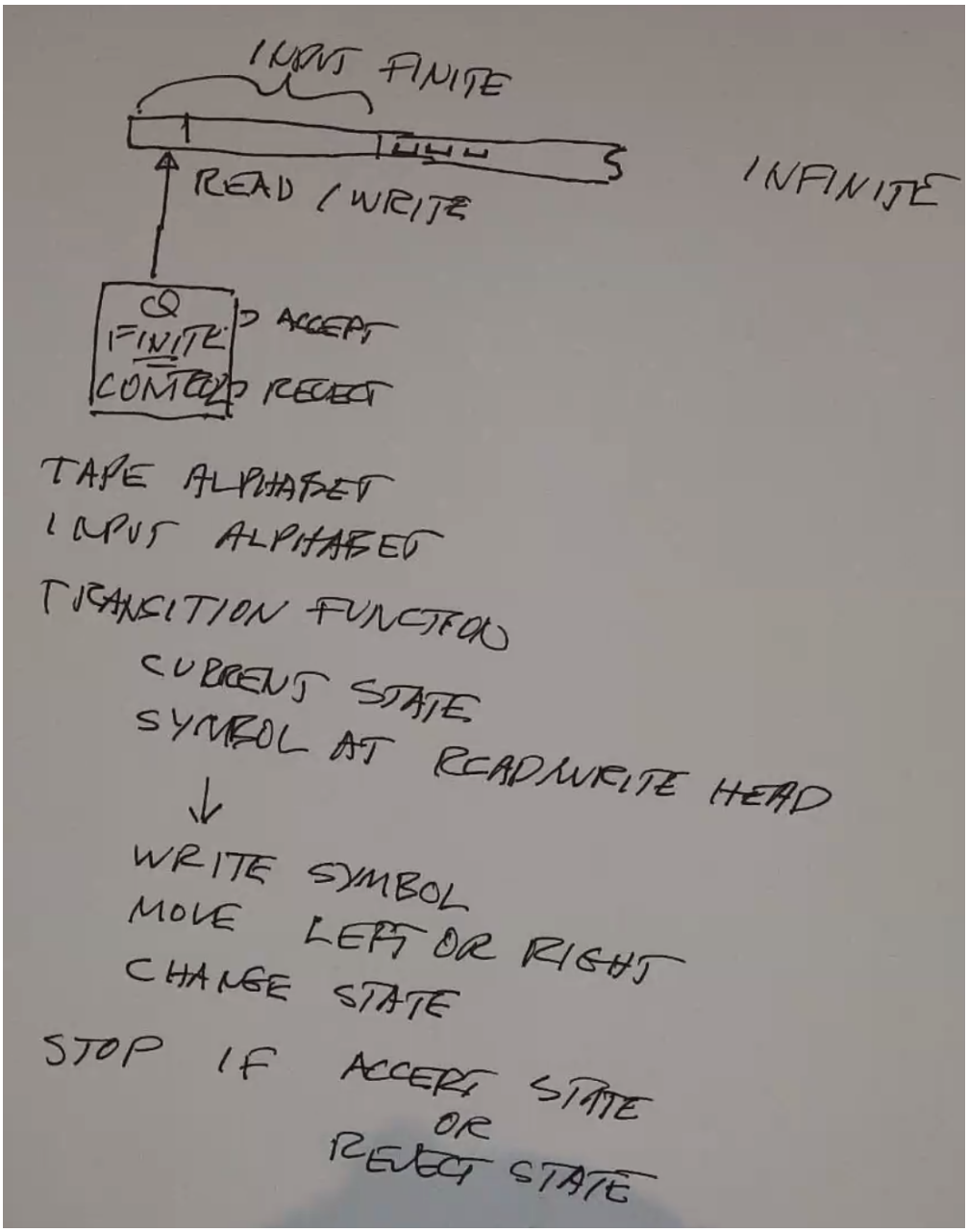
- Turing Machines
- Post systems
- mu-recursive functions
- lambda calc
- combinatory logic

Is it possible to write a program, that given the description of a machine M and an input for M, simulates the execution of the machine on that input? (spoilers: yes, with TMs)

12.1 Definition

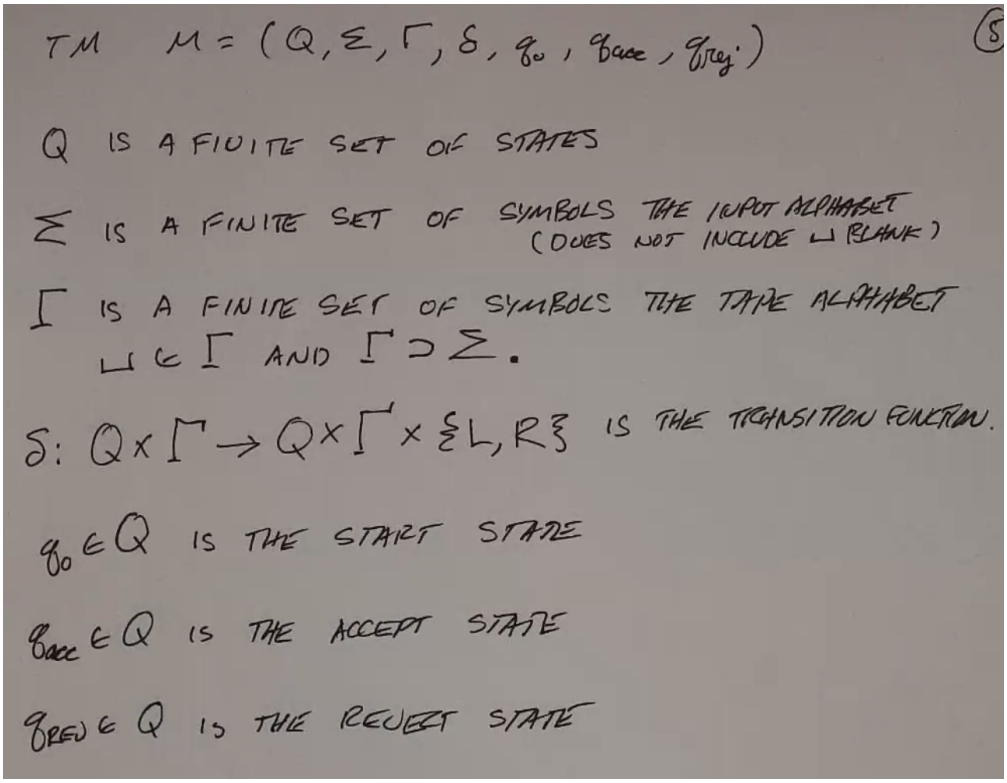
- **infinite tape**
  - but input is finite, rest of tape is blank
- **scanning head**
  - can read cell from tape
  - can write to tape
- **finite control**
  - accept/reject
- **tape alphabet**
  - always includes a special blank symbol, and the input alphabet
- input alphabet
- **transition function**

  - inputs: (current state, symbol at head)
  - outputs: (new state, symbol to write, move l/r)
- stop in accept or reject state

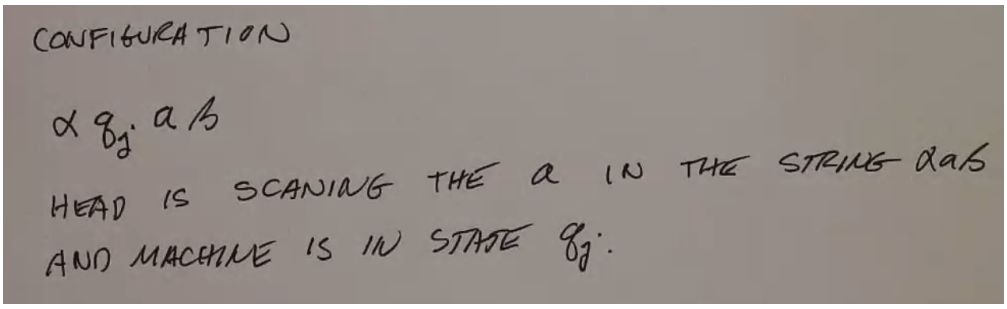


Formally:

- $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$



You can notate the current state of the machine and tape like this:

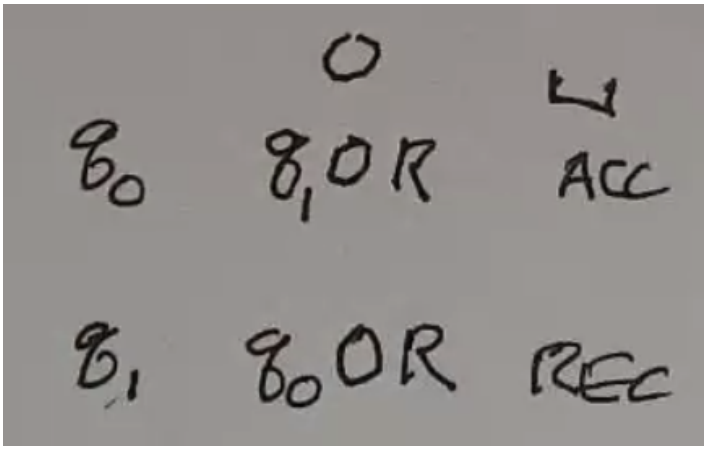


12.2 Examples

Note: The symbol for "empty" on the tape below is notated ⊖.

12.2.1 Ex 1

Even length strings of 0s:

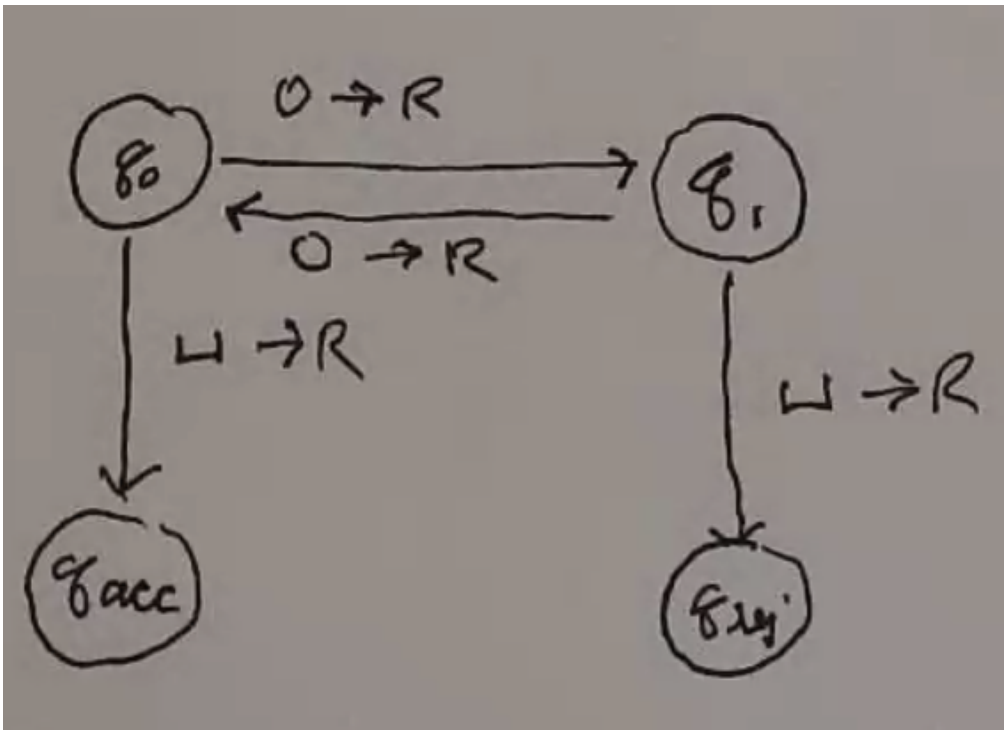


We can use some shorthand:

- If not changing states, omit new state
- If not writing, omit symbol to write

	0	e
q0	q1 R	ACC R
q1	q0 R	REJ r

Alternatively, you can use a state diagram:



12.2.2 Ex 2

Even # of 0s, ignore 1s

	0	1	e
q0	q1 R	R	ACC
q1	q0 R	R	REJ

12.2.3 Ex 3

Add 1 to binary number

- $\Sigma = \{0, 1\}$
- $\Gamma = \{>, 0, 1\}$

Strategy: Move to the end of the tape, then go back and write 0s at each 1 until your carry is fine.

	>	0	1	e
q0	R	R	R	q1, L
q1	REJ	ACC 1 L	0 L	REJ

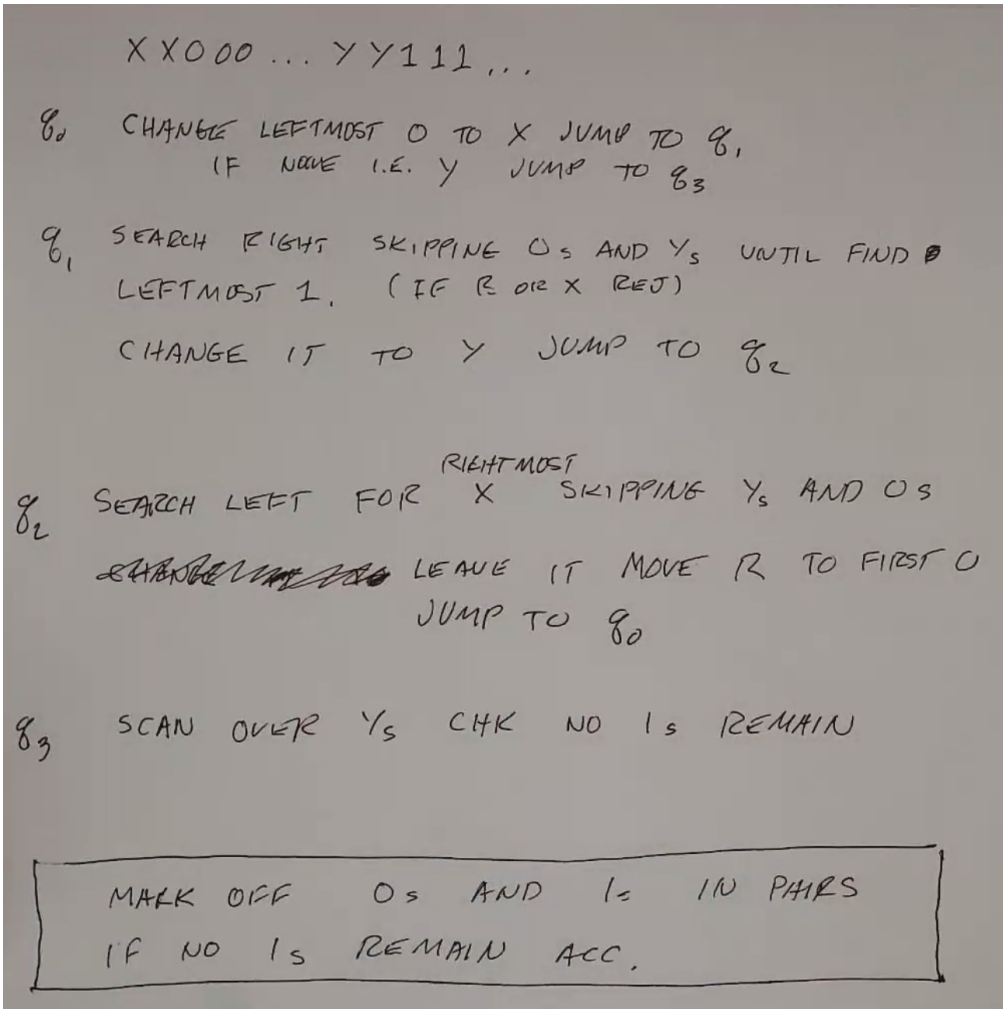
12.2.4 Ex 4

Equal number of 0->1 transitions as 1->0 (assume string starts with 0)

	0	1	e
q0	R	q1 R	ACC
q1	q0 R	R	REJ

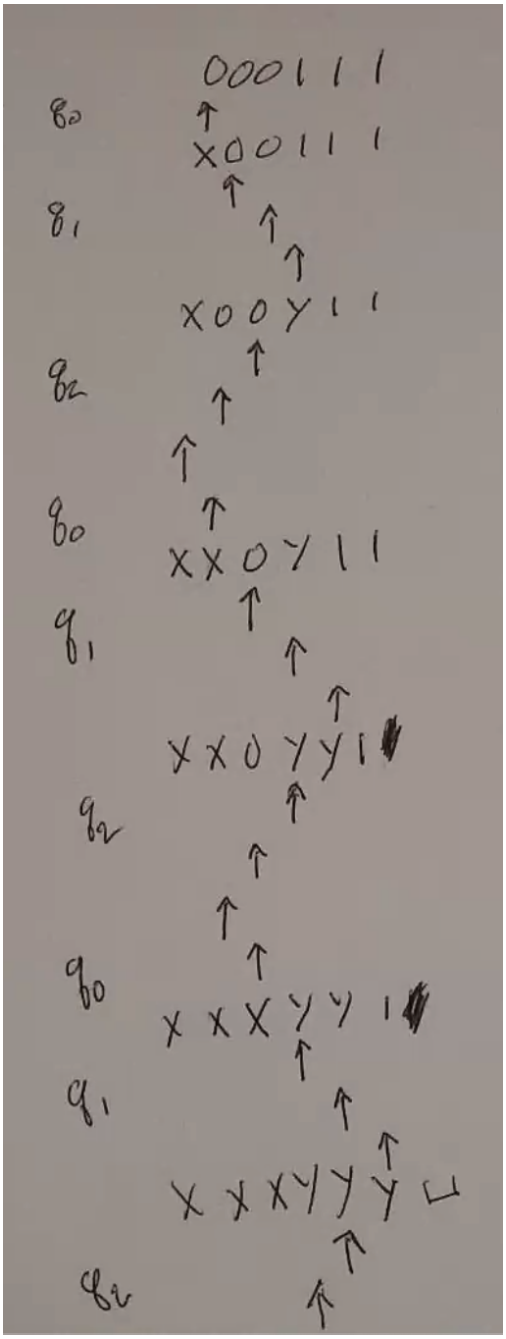
12.2.5 Ex 5

$0^n 1^n$

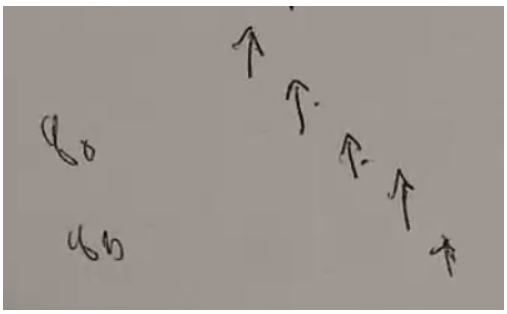


	0	1	X	Y	e
q0	q1 X R	REJ		q3 R	
q1	R	q2 Y L	REJ	R	REJ
q2	L		q0 R	L	
q3	REJ	REJ	REJ	R	ACC

Execution looks like this:



(con't)



12.2.6 Ex 6

Palindromes

- $\Gamma = \{>, 0, 1, e\}$
- A string may look like `>010e`.

	0	1	>	e
s	q0 > R	q1 > R	R	ACC
q0	R	R		q0' L
q0'	q2 e L	REJ	ACC	
q1	R	R		q1' L
q1'	REJ	q2 e L	ACC	
q2	L	L	s R	

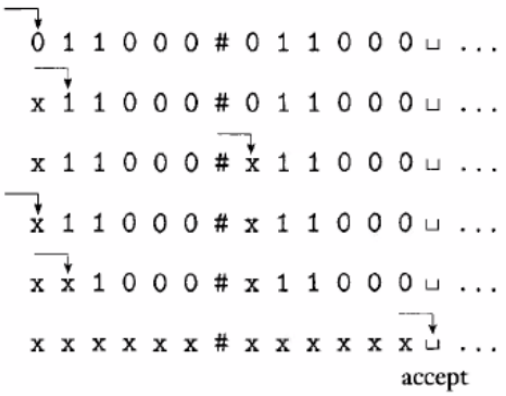
12.2.7 Ex 7

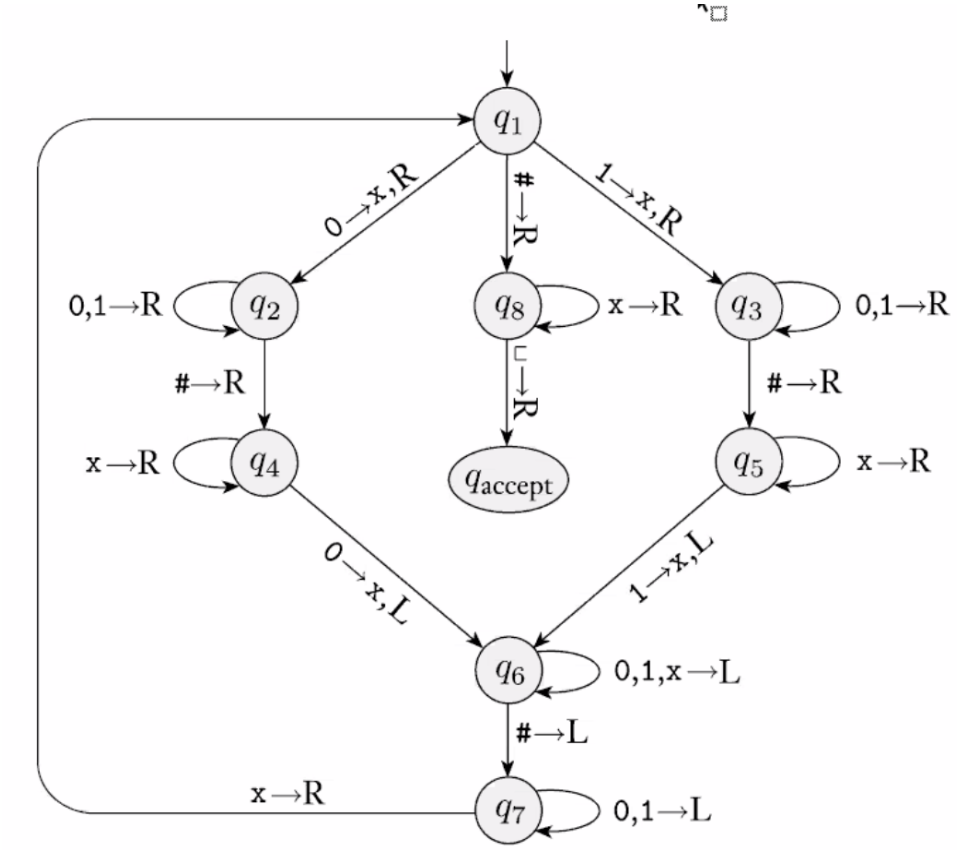
`w#w` - the same string repeated twice with a divider (Sipser's approach)

M_1 = "On input string w:

1. Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do not, or if no # is found, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.
2. When all symbols to the left of the # have been crossed off, check for any remaining symbols to the right of the #. If any symbols remain, *reject*; otherwise, *accept*."

The following figure contains several snapshots of M_1's tape while it is computing in stages 2 and 3 when started on input 011000#011000.





12.2.8 Ex 8

ww - without the boundary

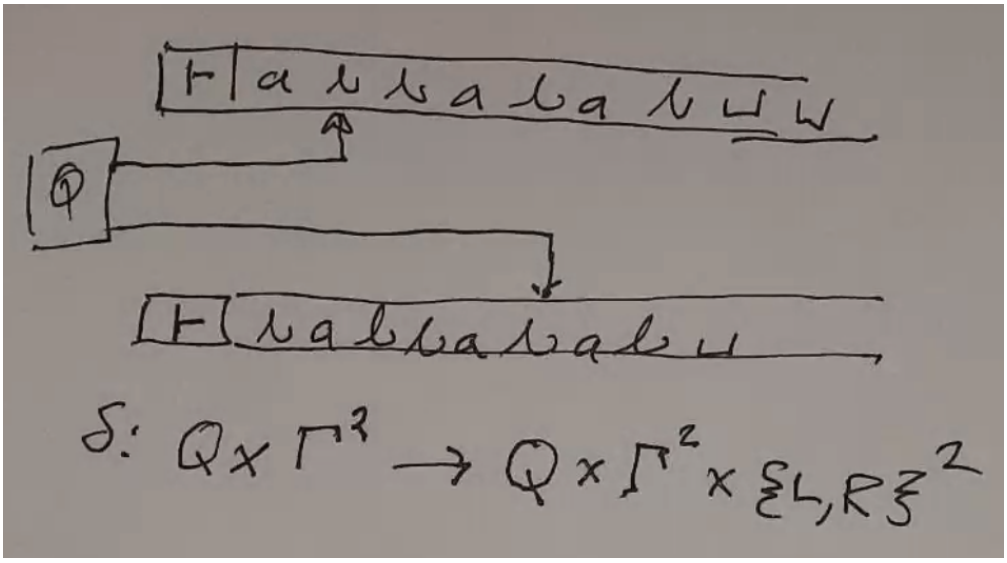
$\Gamma = \{a, b, \dashv, ., \text{similar marks for b}\}$

- **Scan L to R, counting symbols mod 2.**
  - If not even, reject
- When reach end, put down an end marker $\dashv$
- Then repeatedly scan left and right over tape
- When scanning R to L mark first unmarked a or b with á
- When scanning L to R mark first unmarked a or b with à
- Continue until all symbols of input are marked (finds middle of string)
- **Repeatedly scan L to R**
  - Remember and erase first à symbol
  - Check first á matches and erase
  - Reject if no match
- When all symbols erased, reject

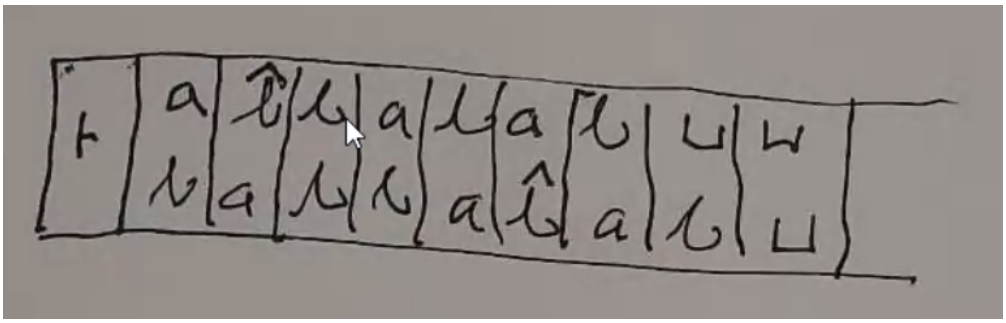
12.3 Variants

12.3.1 Multi-Tape TM

E.g. 2 tapes



This is only just as powerful as a 1-tape TM:



12.3.2 Infinite Tape

Infinite left/right.

12.4 Universal TM

Given:

- initial tape information
- the functional matrix for a TM

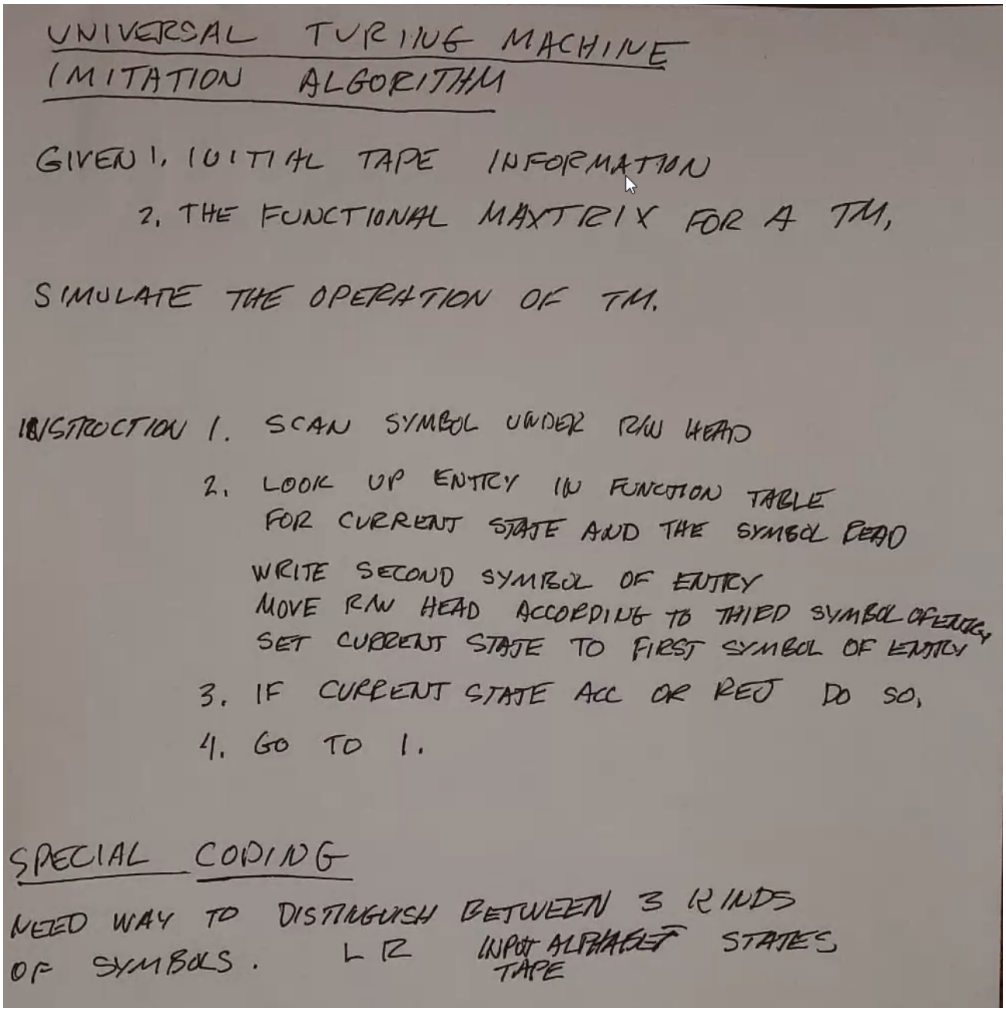
it is possible to simulate the operation of another TM.

1. scan symbol under read/write head
2. **look up entry in function table for current state and the symbol read**

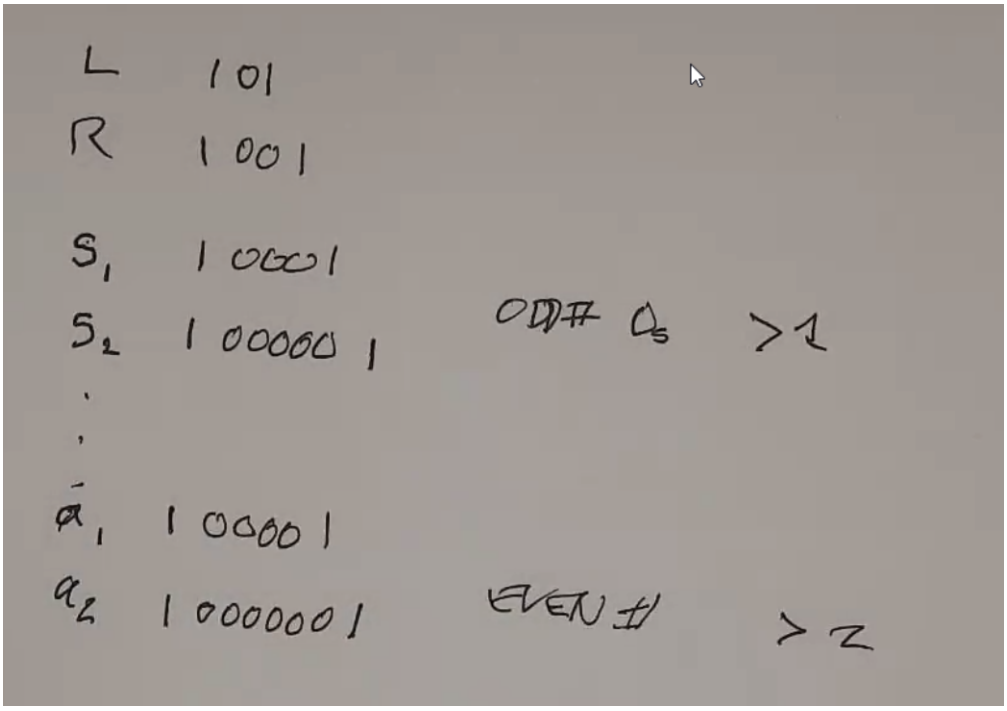
    1. write second symbol of entry
    2. move r/w head according to 3rd symbol entry
    3. set current state to first symbol of entry
3. if current state acc or rej do so
4. goto 1

12.4.1 Special Coding

Need way to distinguish between 3 kinds of symbols: L/R, input/tape alphabet, states

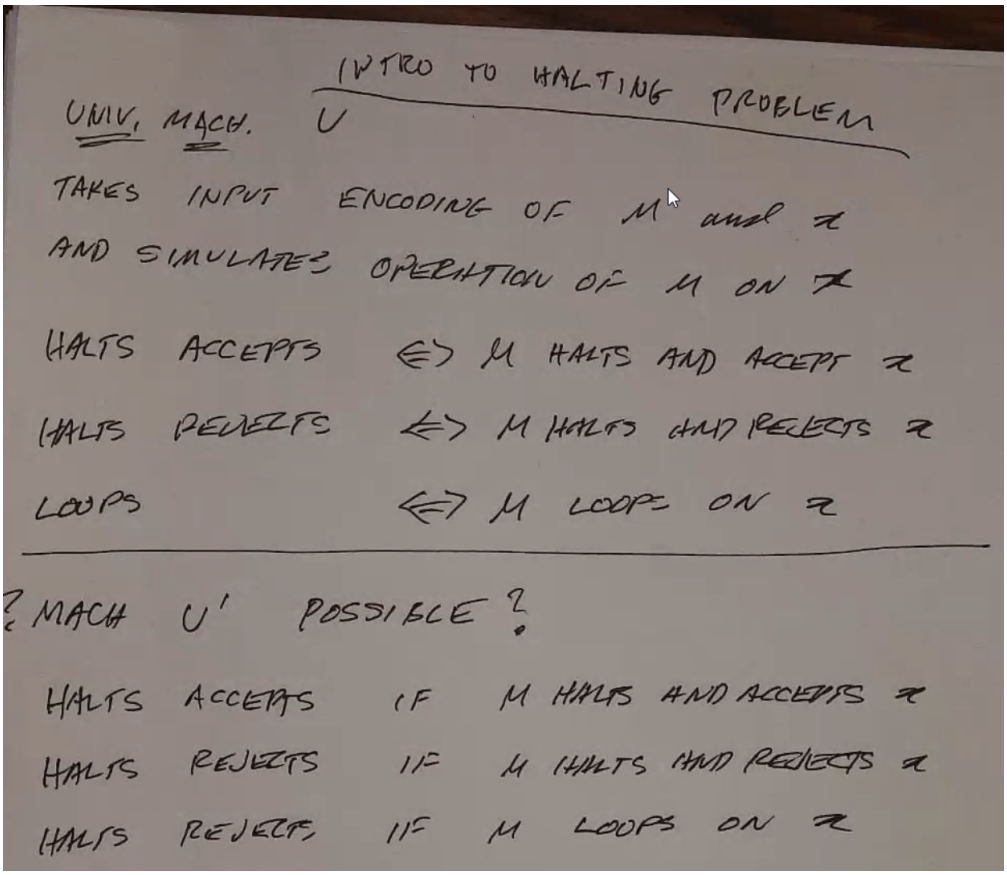


Maybe we can use binary:



12.4.2 Halting Problem

For any universal machine U, it acts the same way on a string as the machine it simulates. Is it possible to make a universal machine that halts and rejects if the simulated machine loops?

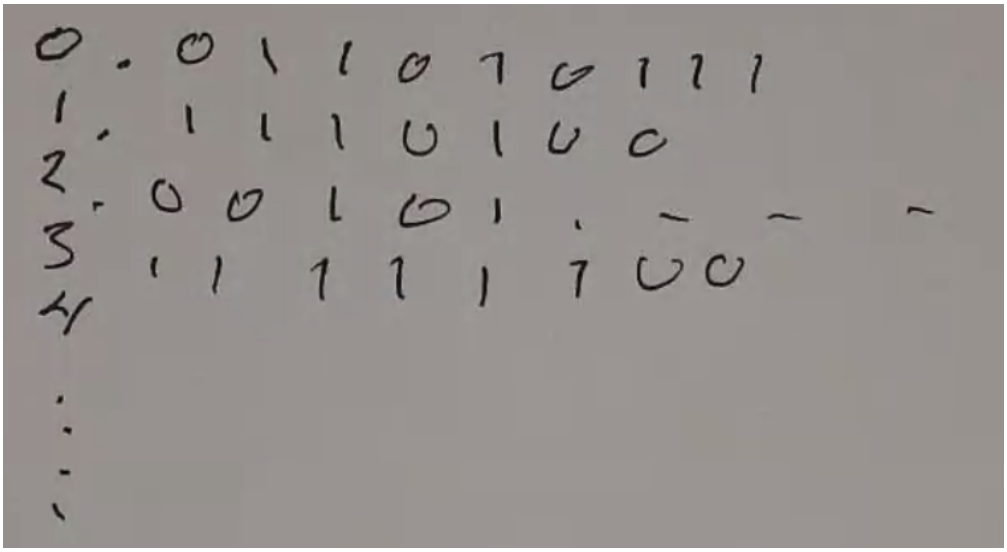


No. No it is not. Use diagonalization:

12.4.2.1 Diagonalization Review

The real numbers are not countable.

- Assume R is countable
- So it is possible to write a list of R
- Consider a list of numbers (in this example, binary decimals):



- **Given this, it is possible to define a real number that is not in the list:**
  - the first digit is the opposite of the first position of the first number in the list
  - the second digit is the opposite of the second position of the second number
  - etc
- so the number must be different from all other numbers in the list
- which means it must not be able to make a list, so R cannot be countable.

12.4.2.2 Proof

- let x be a binary number
- let M_x be the TM with the encoding x
- if x is not a valid TM encoding, M_x halts
- consider the matrix, encoding whether each machine halts on a given input:

	e	0	1	00	01	10	11	...
M_e	H	H	H	H	H	H	H	
M_0								
M_1								
M_00								
M_01								
...								
M_...	H	L	H	L	L	...		

- Assume K exists that can determine, if given some machine M and string x, whether M halts on x

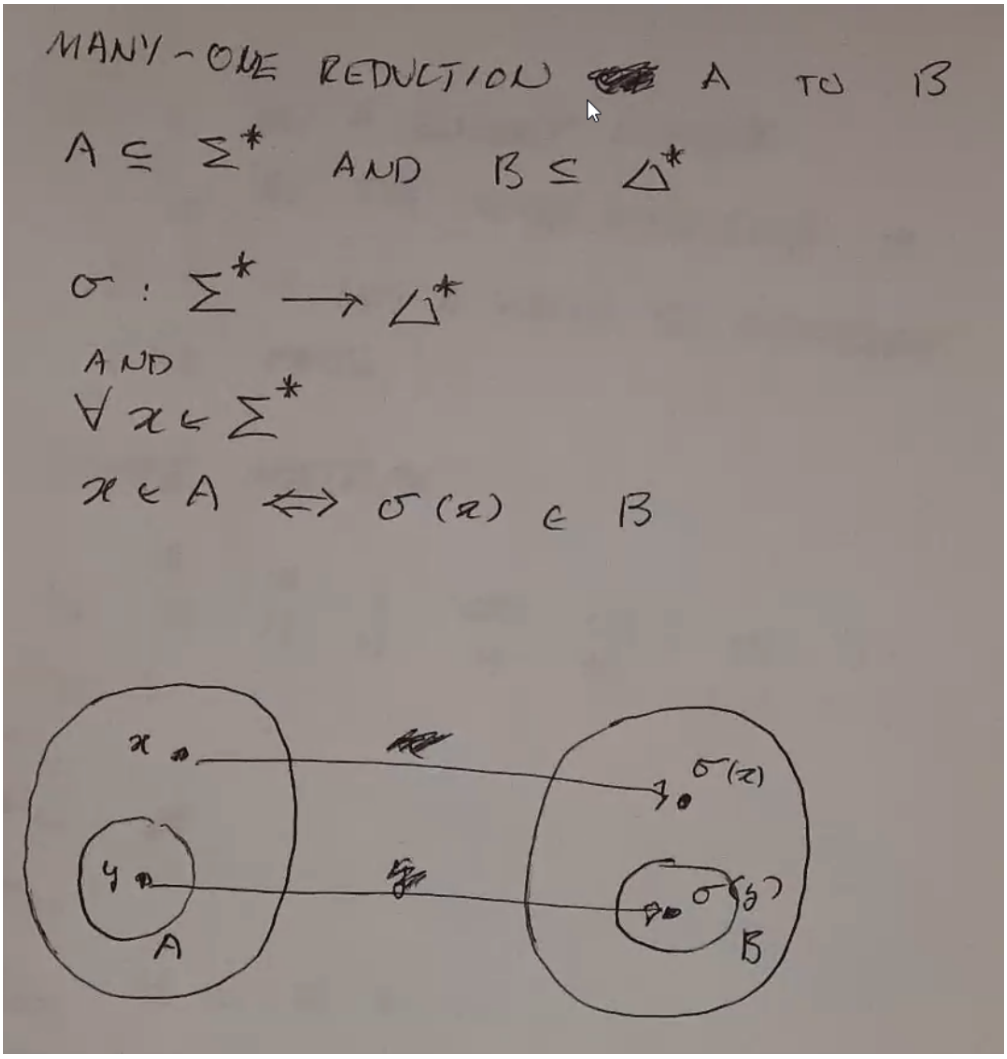
- **Build N using K**
  - On input x:
  - N applies K to M_x, x
  - run K on $M_x x$
  - if K accepts (i.e. M_x halts), N loops
  - if K rejects, N accepts
  - (note: this is the diagonalization argument on the matrix above)
- So N is different on at least one given string for every M_x in the table above
- So we have constructed an impossible machine, since it is not in the list of all possible machines above, so K cannot exist

12.4.2.3 Example

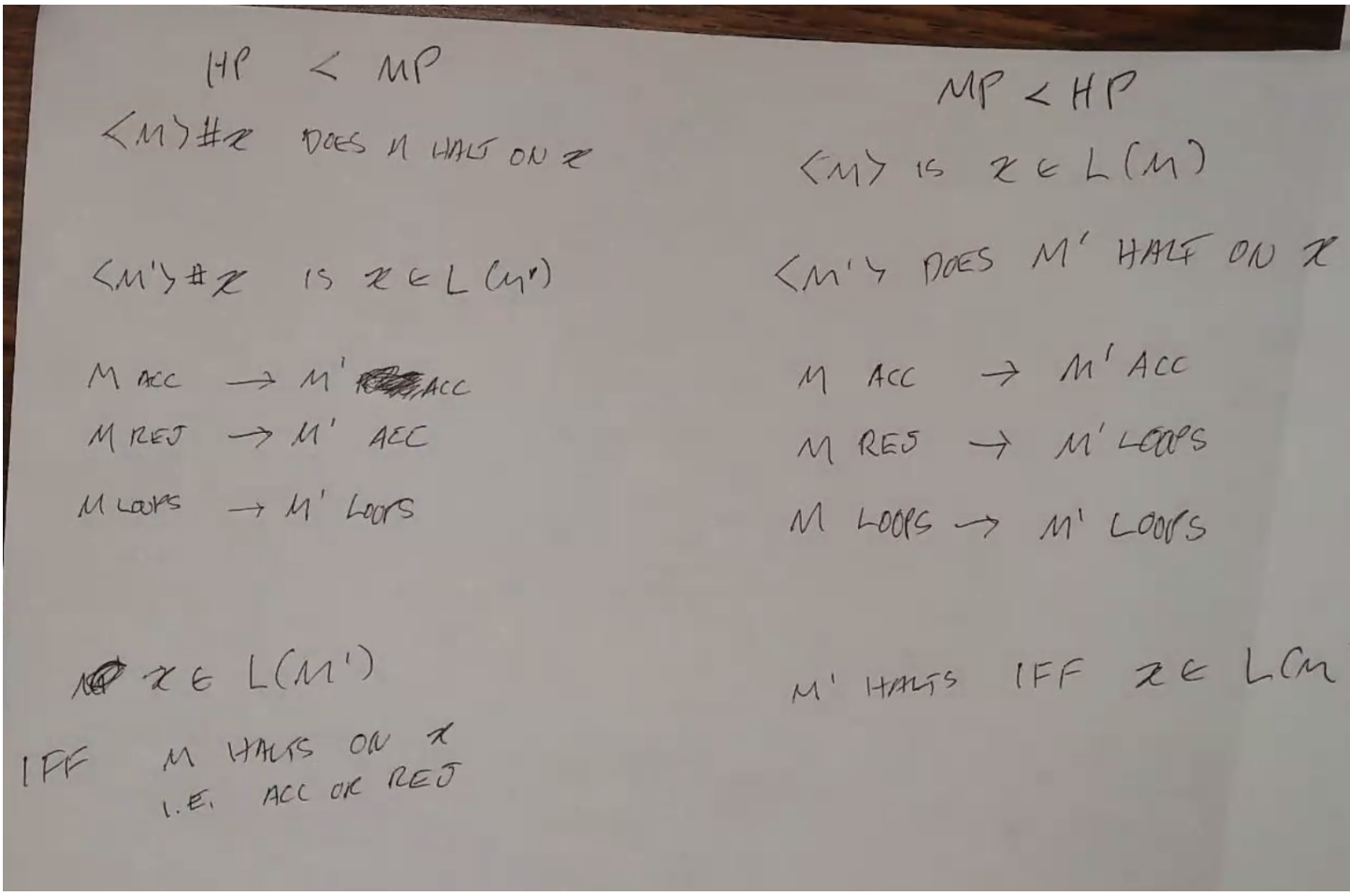
```
m = 999
while m > 1:
    if m % 2 == 0:
        m = m // 2
    else:
        m = 3 * m + 1
```

Is there any value of `m` such that this loop never halts? We don't know!

12.5 Reduction



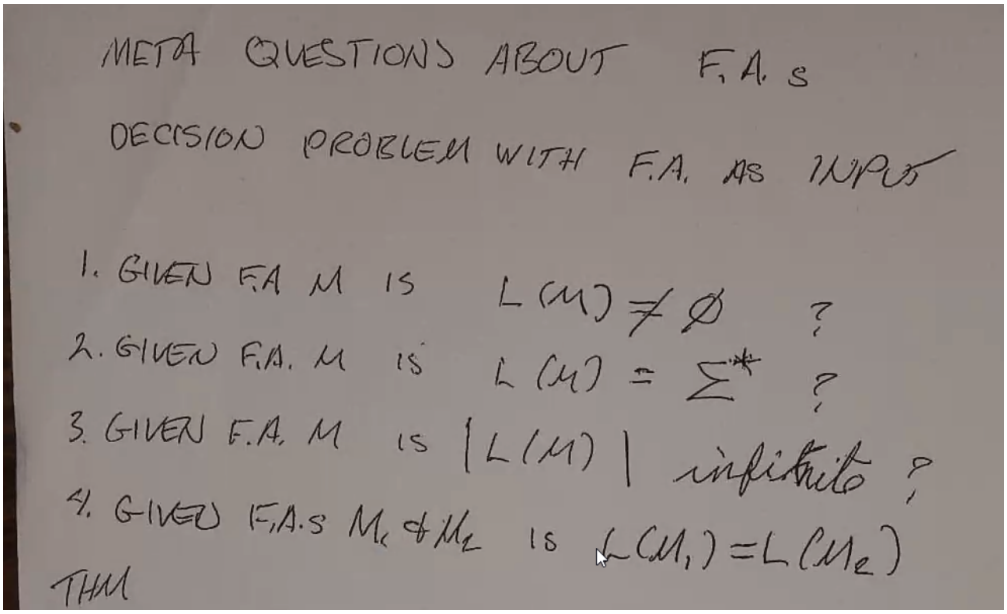
Ex. Halting Problem v. Membership Problem



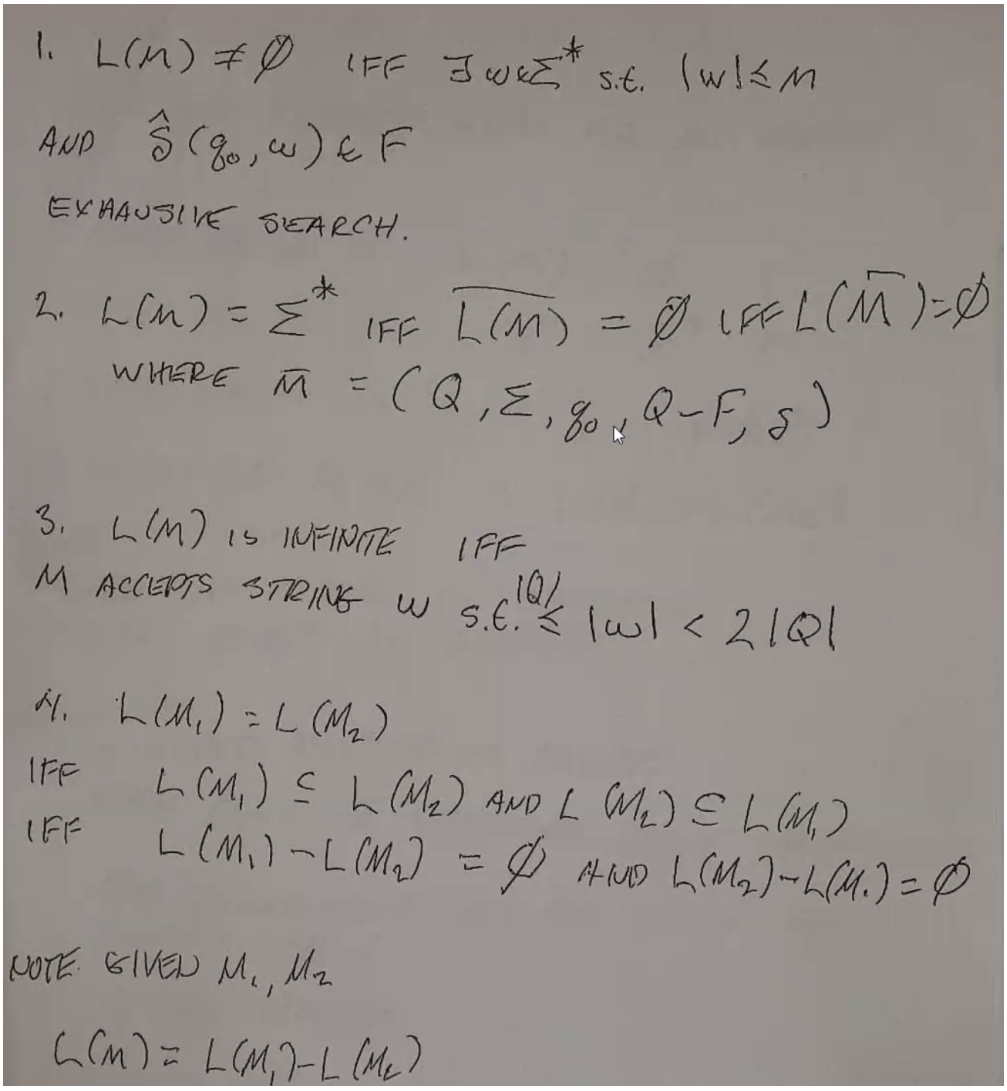
- Left side: reducing the halting problem to the membership problem
- Right side: reducing the membership problem to the halting problem

You can use this to show that the membership problem is not solvable.

12.6 Rice's Theorem



For each of these, there is an algorithm that can solve these problems for finite automata:

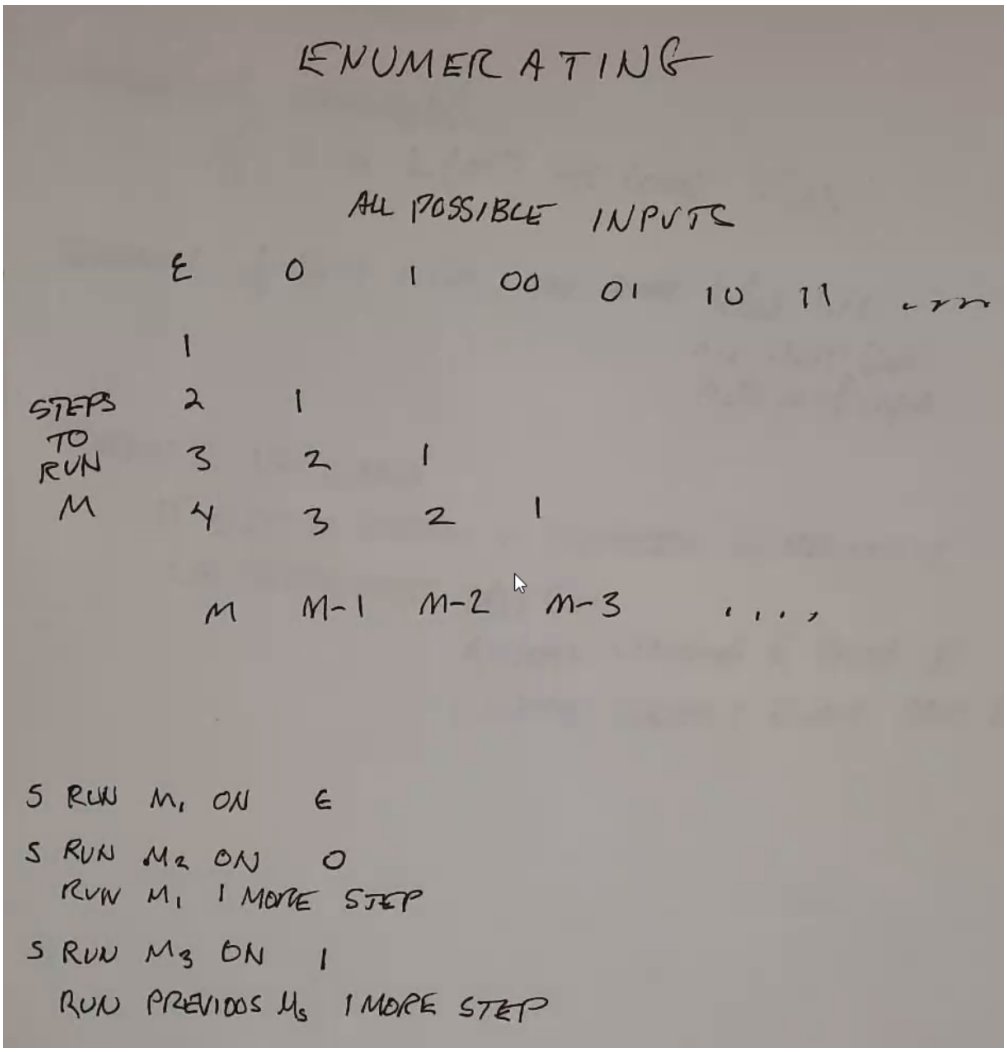


However, these are not solvable for Turing Machines.

12.6.1 Definitions

- **Recursively Enumerable**: if $S = L(M)$ for some TM (i.e. TM doesn't loop)
- **Decidable: a property is decidable if the set of strings with property is recursive**
  - i.e. a total TM accepts strings with prop and rejects without prop

How do we find the set of all strings a TM accepts?

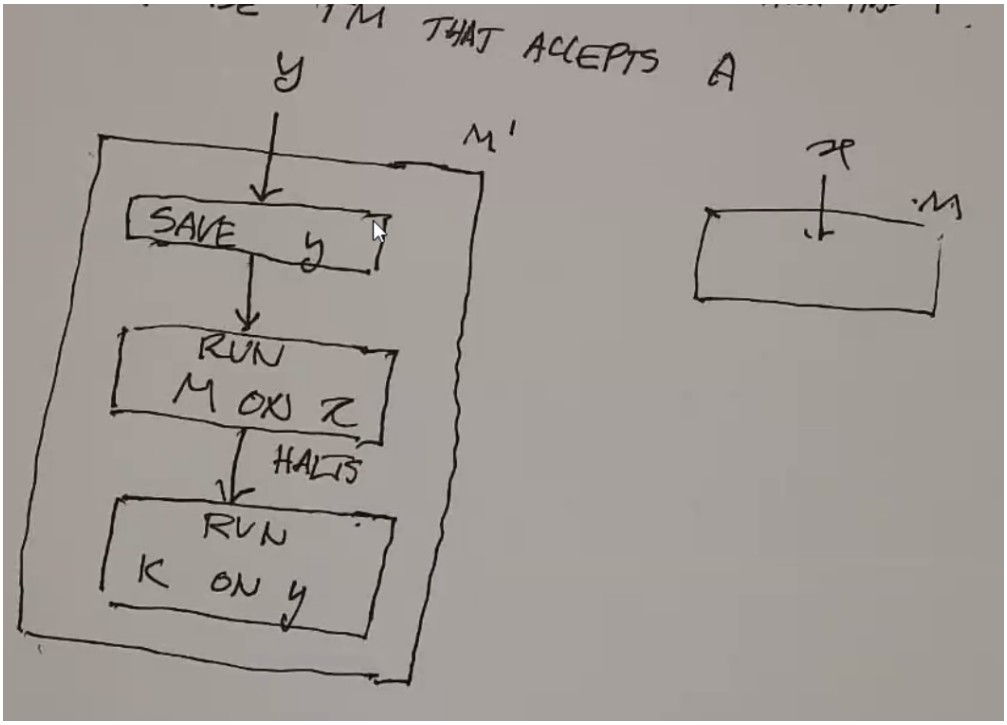


12.6.2 Thm

Any non-trivial property of a recursively enumerable set is undecidable.

12.6.3 Proof

- **Let *p* be a non-trivial property of a recursively enumerable set such that:**
  - $P(A)$ = T or F
  - $P(\emptyset)$ = F
- p is non-trivial $\implies \exists A$ a recursively enumerable set that has p
- let K be a TM that accepts A
- Let M and M' be defined as such:



- above: M halts on $x \implies L(M') = A$
- M does not halt on $x \implies L(M') = \emptyset$
- $L(M') = A \implies P(L(M')) = P(A) = T$
- $L(M') = \emptyset \implies P(L(M')) = P(\emptyset) = F$
- Reversing this, if we can decide the property, then we can tell if M halts on x.
- The halting problem is not solvable, so this problem is not solvable.

12.6.4 Conclusion

regular languages $\subset$ CFLs $\subset$ recursive $\subset$ RE $\subset \Sigma^*$

There are a lot of languages that are not expressible by a TM! There are infinite subsets of Σ^*!

CHAPTER 13

Indices and tables

- genindex
- modindex
- search