CS 4510: Automata and Complexity 08/23/2022-09/15/2022

Lecture EXAM 1: Compiled Notes (Lecture 1-8)

Lecturer: Zvi Galil *Author: Austin Peng*

Contents

1 Lecture 1: Introduction

1.1 Topics To Be Familiar With:

1.1.1 Sets

- sets are elements drawn from some universe:
  - $\mathbb{Z}$ (integers)
  - $\mathbb{N}$ (natural numbers)
  - $\mathbb{R}$ (real numbers)
  - $\mathbb{C}$ (complex numbers)
- set operations:
  - union: $A \cup B$
  - intersection: $A \cap B$
  - complement: $\overline{A}$
  - Cartesian product: $A \times b$ (pairs)
  - power: A^k (k-tuples)

1.1.2 Functions

- functions are mappings from one set to another:
  - $f : D \to R$
    $f(x) = x^2$
  - $f : Z \times Z \to Z$
    $f(x, y) = x + y$

1.1.3 Relations

- relation R can be written as xRy
  - **reflexive**: xRx
  - **symmetric**: $xRy \implies yRx$
  - **transitive**: xRy and $yRz \implies xRz$
  - **equivalence**: if relation R is reflexive, symmetric, and transitive

1.1.4 Graphs

- a graph $G = (V, E)$ has vertices V and edges E
- a graph may be directed or undirected

1.1.5 Strings

- "abc" is a string
- a string is a sequence of symbols from an alphabet Σ
- Σ^* is the set of all strings over Σ
- Σ^* contains the empty string ε, all strings of length 1, all strings of length 2, etc.

1.1.6 Boolean Logic

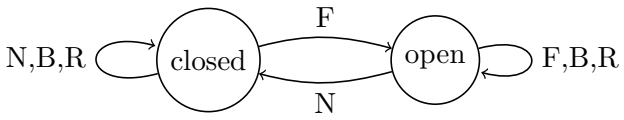
- 3 main operations: "and" ($\wedge$), "or" ($\vee$), "not" ($\neg$)
- these operations are performed on variables and constants (true and false)

1.1.7 Proofs

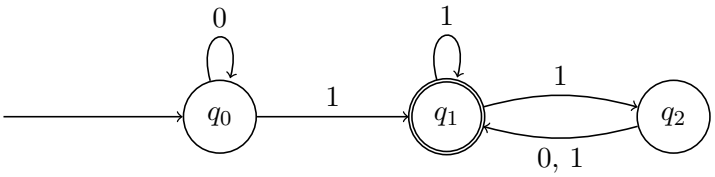
- mathematics consists of definitions, statements/lemmas/theorems, and proofs
- 3 common types of proofs:
  - proof by construction
  - proof by contradiction
  - proof by induction
- these operations are performed on variables and constants (true and false)

1.2 Finite Automata and (Regular Languages)

Consider an automatic door with the inputs $\{\text{F(ront)}, \text{N(either)}, \text{B(oth)}, \text{R(ear)}\}$. The door has 2 states: open and closed. The door will only open if there is someone in front and not in the rear (door swings inward). If the door is open, but nobody is there, the door will close.



Example 1. M_1



- q_0 is the starting state
- q_1 is the accepting statements

- all transitions ($\delta = 0, 1$) are defined for all states
- let x be a binary string 1101, M_1 will accept x because it ends at q_1
- $L(M_1)$ = all x such s.t M_1 accepts x if x has a 1 AND either ends with a 1 or with an even number of 0s

1.2.1 Formal Definition Of A Deterministic Finite Automaton

Definition 1. A deterministic finite automaton is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set called the states
- Σ is a finite set called the alphabet
- $\delta : Q \times \Sigma \to Q$ is the transition function
- q_0 is the initial state
- $F \subseteq Q$ is the accepting states

Note: $\delta(q, a) = p$ means that if DFA is in state q and sees a, it goes to state p

1.2.2 Formal Definition Of Computation

Definition 2. A **computation** of a deterministic finite automaton (DFA) M on an input string $x = x_1, ..., x_n \in \Sigma^*$ is a sequence of states $q_0, ..., q_n \in Q$ s.t. for $i = 0, ..., n-1$ $\delta(q_i, x_{i+1}) = q_{i+1}$. We say that M accepts x if $q_n \in F$. If $q_n \notin F$, then we say M rejects x.

1.2.3 Formal Definition Of A Language

Definition 3. The **language** of M is defined as the set $\{x \in \Sigma^* : M \text{ accepts } x\}$. A language is a set of strings.

Example 2. $\emptyset, \{\varepsilon\}$ are both langauges.

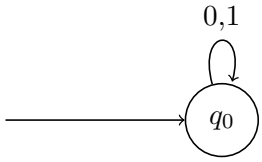
Definition 4. A language L is called **regular** if there is a DFA accepting L.

Example 3. $L_w = \{w\}$ is regular. $L'_w = \{\text{all strings that end in } w\}$ is also regular.

2 Lecture 2: Deterministic Finite Automata

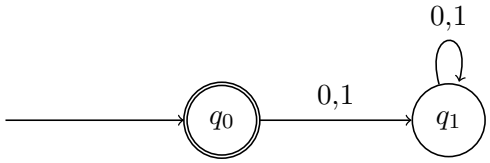
2.1 DFA Examples

Example 4. M_2



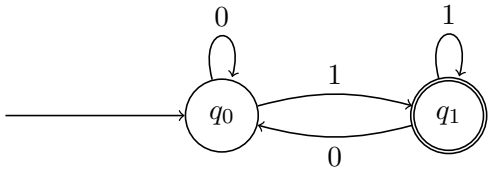
$L(M_1) = \varphi$. The language is empty because there are no accepting states.

Example 5. M_2



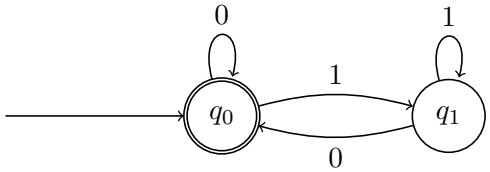
$L(M_2) = \{\varepsilon\}$.
Note the difference between M_1 and M_2. They recognize different langauges.

Example 6. M_3



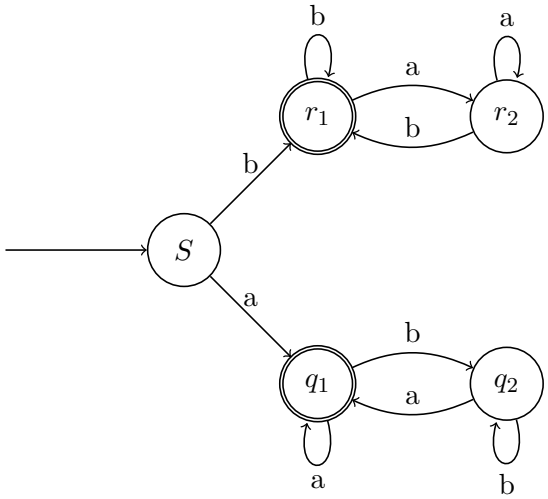
$L(M_3) = \{w | w \text{ ends in a } 1\}$

Example 7. M_4



$L(M_4) = \{\varepsilon \cup$ strings ending with $0\}$. Note this is the complement of $L(M_3)$.

Example 8. M_5



$L(M_5)$ is the set of all strings that start and end with the same character. Note: $\Sigma = \{a, b\}$

2.2 Applications Of DFA

2.2.1 Modular Arithmetic

Let $w \in \{0,1\}^*$ (aka any binary string). We define $\overline{w}$ to be the value of the string as a binary number. Then, for $w \in \{0,1\}^*$ and $a \in \{0,1\}$, we have the following properties:

- $\overline{a} = a$
- $\overline{wa} = 2\overline{w} + a$

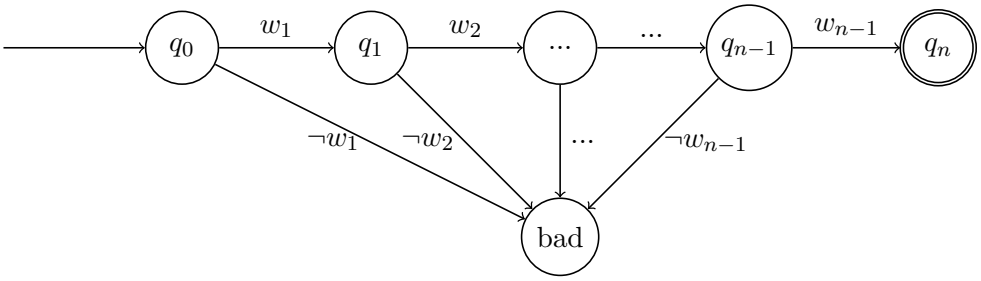
We can use a DFA to recognize modular arithmetic. For the following example, we will consider the following transition table of $\overline{w}$ mod 3. Note that the start state of our transition table is marked with an arrow.

input a / $\overline{w}$ mod 3	$\overline{w0}$ mod 3	$\overline{w1}$ mod 3	state
0 (state q_0)	0	1	$\rightarrow q_0$
1 (state q_1)	2	0	q_1
2 (state q_2)	1	2	q_2

If we set the accepting state to be q_1 then this DFA will accept exactly those strings which are $\equiv 1$ modulo 3 (aka congruent to 1 modulo 3).

2.2.2 String Matching: Recognizing A Single String

For a string w, we can create a DFA for the language $L_w\{w\}$ as follows:


q_0
w_1
q_1
w_2
...
...
q_{n-1}
w_{n-1}
q_n
$\neg w_1$
$\neg w_2$
...
$\neg w_{n-1}$
bad

2.2.3 String Matching: Recognizing A Suffix

Let L'_w be the set of strings that end in w. An example string from this language is $1101001 \in L'_{001}$, because it ends in 001. We can use the following transition table:

Q \ input	0	1
$\rightarrow$ bad	q_0	bad
q_0	q_{00}	bad
q_{00}	q_{00}	q_{001}
q_{001}	q_0	bad

We define q_{001} to be our only accepting state.

In the general case, we need to keep track of the longest suffix seen so far. We will use the states $\{\text{bad}, q_0, ..., q_n\}$.

The DFA will be in state q_i if $w_1...w_i$ is the longest suffix of the input seen so far that is a prefix of w. If we are in state q_i, then we have to see $n - i$ more symbols until we find the string. The transition function is defined as follows:

- $\delta(q_{i-1}, w_i) = q_i$
- $\delta(q_{i-1}, a \neq w_i) = q_j$, where $w_1 w_2 ... w_j$ is the largest prefix of w that is a suffix of the current input (including a).

3 Lecture 3: Operations On Languages

3.1 Operations On Languages

3.1.1 The Regular Operations: Union, Concatenation, Kleene Star

Definition 5. Let A and B be languages. We define the regular operations union, concatenation, and Kleene star as follows:

- **union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
- **concatenation**: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$
- **Kleene star**: $A^* = \{x_1 x_2 ... x_n \mid k \geq 0 \text{ and each } x_i \in A\}$
  - $A^* = \bigcup_{k \geq 0} A^k$

Note: $A^+ = A^* - \{\varepsilon\}$.

Example 9.

$$A = \{\text{good}, \text{bad}\}, B = \{\text{boy}, \text{girl}\}$$

$$A \cup B = \{\text{good}, \text{bad}, \text{boy}, \text{girl}\}$$

$$A \circ B = \{\text{goodboy}, \text{goodgirl}, \text{badboy}, \text{badgirl}\}$$

$$A^* = \{\varepsilon, \text{good}, \text{bad}, \text{goodgood}, \text{goodbad}, \text{badgood}, \text{badbad}, ...\}$$

Theorem 1. The class of regular languages is closed under the union operation. That is, if A and B are regular languages, so is $A \cup B$.

Proof. Let M_1 recognize A_1, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and M_2 recognize A_2, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

Construct M to recognize $A_1 \cup A_2$, where $M = (Q, \Sigma, \delta, q, F)$.

1. $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$
   - this set Q is essentially $Q_1 \times Q_2$, the Cartesian product of sets Q_1 and Q_2
   - it is the set of all pairs of states: the first from Q_1 and the second from Q_2
2. $\Sigma = \Sigma$, the same as in M_1 and M_2
   - for simplicity, assume M_1 and M_2 have the same alphabet Σ
   - the theorem remains true if they have different alphabets Σ_1 and Σ_2
   - then modify the proof to let $\Sigma = \Sigma_1 \cup \Sigma_2$
3. δ (transition function) is defined as follows:
   - for each $(r_1, r_2) \in Q$ and each $a \in \Sigma$, let $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
4. q_0 is the pair (q_1, q_2)
5. F is the set of pairs in which either member is an accepting state of M_1 or M_2 as follows:
   - $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$
   - the above expression is the same as $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$
   - Note: $F = (F_1 \times F_2)$ IS NOT CORRECT! (takes intersection instead of union)

□

3.2 Other Regular Operations

3.2.1 Complement

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

Proof. Let L be a regular language, then some finite automaton M recognizes L.

Let $\overline{M}$ be the same as M, but with the accepting and non-accepting states interchanged. Then $\overline{M}$ accepts a string x if and only if M does not accept x. So, $L(\overline{M}) = \overline{L}$.

□

3.2.2 Intersection

Theorem 3. If A_1 and A_2 are regular languages, then so is $A_1 \cap A_2$.

Proof. Let $M_1 = (Q^1, \Sigma, \delta^1, q_0^1, F^1)$ decide A_1 and $M_2 = (Q^2, \Sigma, \delta^2, q_0^2, F^2)$ decide A_2.

We construct the automaton $M = (Q, \Sigma, \delta, q_0, F)$ as follows:

- $Q = Q^1 \times Q^2$ (each state in M is a pair of states in M_1 and M_2)
- Σ is the same shared alphabet as M_1 and M_2
- $\delta((r_1, r_2), x) = (\delta^1(r_1, x), \delta^2(r_2, x))$
- $q_0 = (q_0^1, q_0^2)$
- $F = F^1 \times F^2$ (both M_1 and M_2 must be in an accepting state for M to accept)

□

3.2.3 Set Difference

Theorem 4. If A and B are regular languages, then so is $A_1 \backslash A_2 = \{x \mid x \in A \text{ and } x \notin B\}$.

Proof. Note: $A \backslash B = A \cap \overline{B}$

Since regular languages are closed under intersection and complement, regular languages are closed under subtraction.

□

3.2.4 Symmetric Difference

Theorem 5. If A and B are regular langauges, then so is $A \oplus B$.

Proof. Note: $A \oplus B = (A \cup B) \backslash (A \cap B)$

Since regular languages are closed under union, intersection, and subtraction, regular languages are closed under symmetric difference.

□

3.3 Closed Operations

A set S is closed under an operation $\cdot$ if for every $a, b \in S, a \cdot b \in S$. That is, if we apply the operation to any two element in the set, we another element in the same set.

- $\mathbb{N}$ is closed under addition
- $\mathbb{N}$ is not closed under subtraction (ex. $3 - 5 = -2 \notin \mathbb{N}$)
- $\mathbb{Z}$ is closed under addition, subtraction, multiplication, but not division.

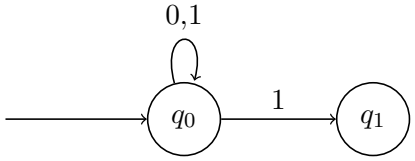
- $\mathbb{Q}$ is closed under addition, subtraction, multiplication, and $\mathbb{Q}\backslash\{0\}$ is closed under division.
- $\mathbb{R}$ is not closed under square root, but $\mathbb{R}^+$ is closed under square root.
  - $\mathbb{R}$ has negative numbers, while $\mathbb{R}^+$ does not.
- $\mathbb{C}$ is closed under square root (because it includes imaginary numbers).

4 Lecture 4: Non-Determinism

4.1 Non-Determinism

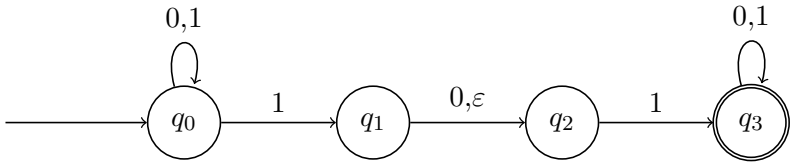
- Non-determinism looks strange and impractical, and in some sense it is, but it is very important.
- You will find out at the end of the course why the most important problem in computer science involves non-determinism and non-deterministic computation.
- Our definition of finite automaton so far is deterministic. This means, when we are at some state, and we see a symbol, we go to exactly one other state. Also, from every state, the transition is well defined for every symbol in the alphabet.
- 2 main differences between a non-deterministic finite automaton (NFA) and deterministic finite automaton (DFA).
  - from a state, when we see a symbol, we can go to 0 or more states (in a DFA, when we see a symbol, we go to exactly 1 state)
  - we have "ε"-transitions, which are transitions that we can take without seeing any symbol.
- The starting, accepting, and states all work the same. The big difference is in δ (transition function).

Example 10. Consider the following NFA:



- If we are at q_0 and see a 1, we do not have to go to q_1.
- Rather, think of it like we are in many states at once. (referred to as "guessing")
- In other words, we say that from q_0, we can go to several states.
- The automaton is a genius and knows where to go. It has the foresight to think, "I see 1 several times, I stay at q_0, but I will transition to q_1 at just the right time."
- The choice of the 1 to transition is because the automaton can see the future.
- Note: this is abstract and not practical

Example 11. Consider the following NFA:



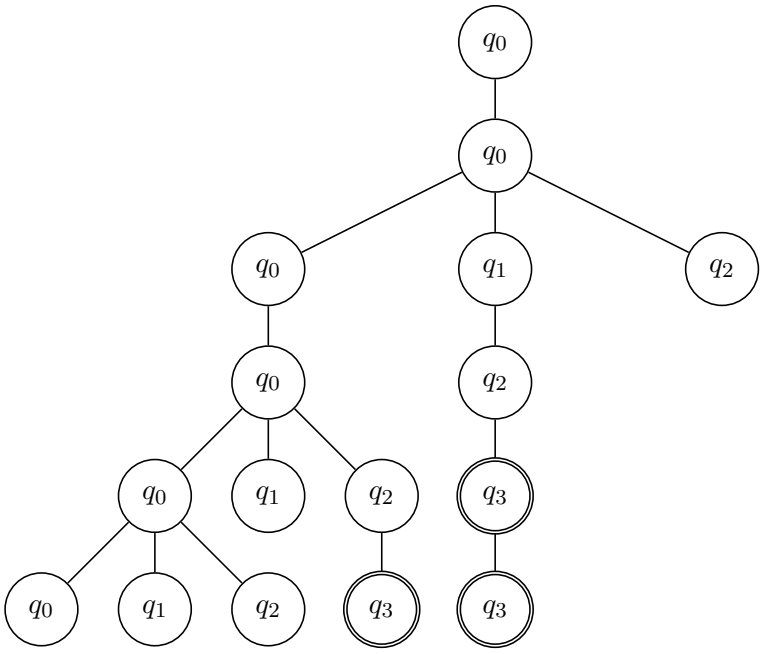
Suppose at q_1 and we see a 1. There is no transition with 1, so we "abort". At q_1 we see ε and move to q_2. Note: we always see ε, so without doing anything at q_1, we "slide" to q_2.

Below is a transition table describing the NFA.

Q \ symbol	0	1	ε
$\rightarrow q_0$	q_0	q_0, q_1	-
q_1	q_2	-	q_2
q_2	-	q_3	-
q_3	q_3	q_3	-

Think of split timelines every time we have to go to multiple states. From q_0 seeing 1, we can go back to q_0 or go to q_1. Note that we accept the input if at least one of the timelines ends at an accepting state.

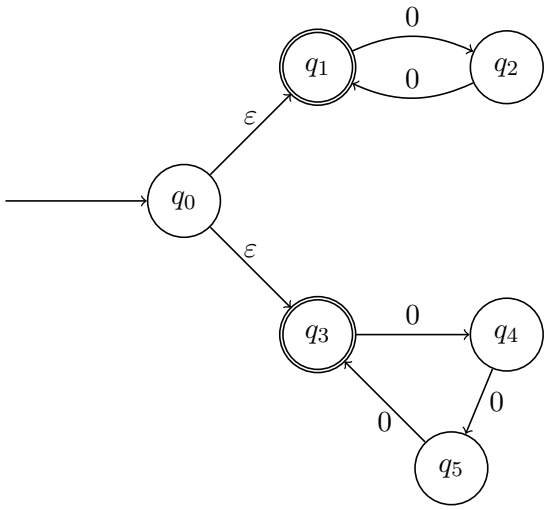
Below is a decision parse tree for input $x = 01011$



The nodes at each layer represent the states we are in at that time (given the input). At q_0, we see 0 and can only go back to q_0 At q_1, we see 1 and can go to q_0, q_1, or slide with ε to q_2. And repeat for the remaining binary values in the input string x. Once we finish, as long as one state in the final depth is accepting, the string x is accepted. This is the reason why we call this "guessing".

4.2 Unary Language

Example 12. M_1



This NFA guesses if the input string length is divisible by either 2 or 3 by guessing to go higher q_1 or lower q_3.

4.3 Nondeterministic Finite Automaton

4.3.1 Formal Definition Of A Nondeterministic Finite Automaton

Definition 6. A **nondeterministic finite automaton** is a 5-tuple $Q, \Sigma, \delta, q_0, F$ where:

- Q is a finite set of states
- Σ is a finite alphabet
- $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is the transition function, where $\mathcal{P}(Q)$ is the power set of Q
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accepting states

Note: the only change in the definition compared to a DFA is the transition function δ. δ is now a set of states.

We say M accepts w if w can be written as $w = y_1 ... y_n$ such that $y_i \in \Sigma_\varepsilon$ and there are states $r_0, R_1, ..., r_m$ such that $r_0 = q_0, r_m \in F,$ and $r_{i+1} \in \delta(r_i, y_{i+1})$ for $0 \leq i \leq m - 1$.

4.3.2 Equivalence Of NFAs and DFAs

Corollary 1. A language is regular if an only if some nondeterminstic finite automaton recognizes it.

Theorem 6. Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

Proof Idea. As a **corollary**, the set of languages NFAs can accept (N) is equal to the set of languages that DFAs can accept (D). The first part of the proof is to note that every DFA is an NFA, so $D \subseteq N$. Then to show that $N \subseteq D$, we say for each NFA $N_i \in N$ we can construct an equivalent DFA, implying that $N_i \in D$. Together this implies that $N = D$.

Note: in terms of power of these two computing structures, this proof shows that there is no difference. But in terms of size and simplicity, NFAs have the advantage (since an equivalent DFA must have at least 2^k states).

Theorem 7. Let L_k be the regular language over $\{0, 1\}$ which contains all strings which have a 1 as the k'th character from the right end of the string. Any DFA that recognizes L_k must contain at least 2^k states.

Proof. Suppose that D is a DFA for L_k containing strictly fewer than 2^k states. Then, by the pigeonhole principle, there must be a state q such that two different binary strings of length $k, x,$ and y, both cause the machine to end in state q. But if x and y are different, there is at least one index i such that $x[i] = 0$ and $y[i] = 1$ or vice versa. But then $x0^{i-1}$ and $y0^{i-1}$ cause the machine to end in the same state, yet one must be accepted and the other must be rejected. This is a contradiction, so our assumption that there was a DFA for L_k with strictly fewer than 2^k states is incorrect.

□

5 Lecture 5: NFA To DFA

5.1 Equivalence Of NFAs and DFAs (cont.)

Theorem 8. Every NFA has an equivalent DFA.

Proof. Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A. We construct DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A. Before doing the full construction, first consider the easier case when N has no ε arrows. We will take ε into account later.

1. $Q' = \mathcal{P}(Q)$ (the set of subsets of Q)

   Every state of M is a set of states of N.

2. Σ (the alphabet) doesn't change

3. For $R \in Q'$, and $a \in \Sigma$, let $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$

   If R is a state of M, it is also a set of states of N. When M reads a symbol a in state R, it shows where A takes each state in R. Because each state may go to a set of states, we take the union of all these sets.

   $$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$

4. $q_0' = \{q_0\}$

   M starts in the state corresponding to the collection containing just the start state of N.

5. $F' = \{R \in Q' \mid R \text{ contains an accepting state of } N\}$

   The machine M accepts if one of the possible states that N could be in at this point is an accepting state.

Now consider the ε arrows. For any state R of M, we define $E(R)$ to be the collection of states that can be reached from members of R by going only along ε arrows, including members of R themselves. Formally, for $R \subseteq Q$, let

$$E(R) = \{q \mid q \text{ can be reached from } R \text{ by traveling along 0 or more } \varepsilon \text{ arrows}\}$$

Then we modify the transition function of M to place additional fingers on all states that can be reached by going along ε arrows after every step. Replacing $\delta(r, a)$ by $E(\delta(r, a))$ achieves this. Finally, we need to modify the start state of M to move the fingers initially to all possible states that can be reached from the start state of N along the ε arrows.

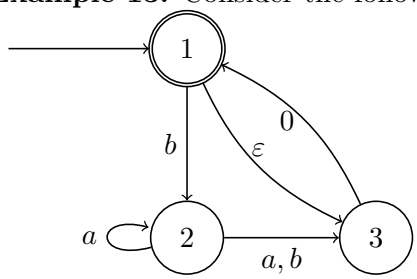
The changes mentioned above to account for ε arrows are shown below:

3. $\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some } r \in R\}$

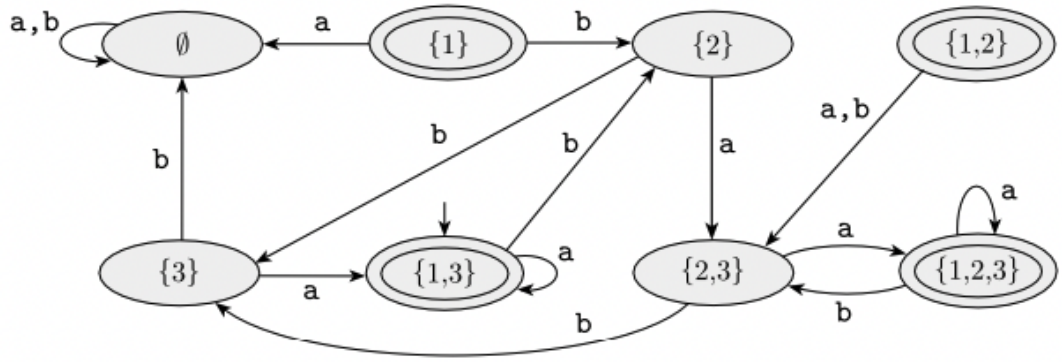
4. $q_0' = E(\{q_0\})$

□

Example 13. Consider the following NFA N:



Note that the DFA will have 8 states, one for each subset of the states of N. The DFA and its transitions are shown below:



The NFA's start state is 1, so the DFA's start state is $E(\{1\}) = \{1, 3\}$ (the set of states reachable from 1 by travelling along ε arrows and 1 itself). The NFA's accepting state is 1, so the DFA's accepting states are all sets of states that include 1: $\{\{1\}, \{1, 2\}, \{1, 3\}, \{1, 2, 3\}\}$

As for D's transition function, each of D's states goes to one place on input a and one place on input b (by definition of DFA). We will illustrate a few.

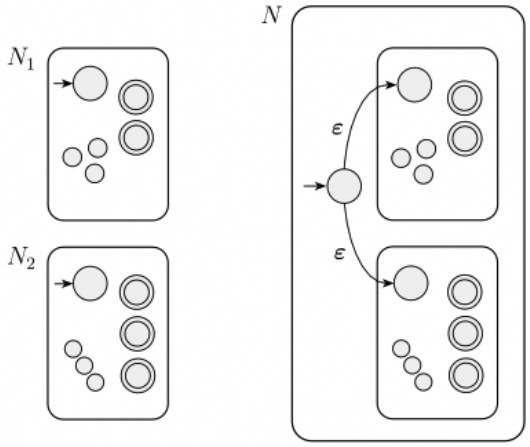
- in D, state $\{2\}$ goes to $\{2, 3\}$ on input a because in N, state 2 goes to both 2 and 3 on input a.
- in D, state $\{1\}$ goes to $\emptyset$ on input a because no a arrows exit it.
- in D, state $\{1, 2\}$ goes to $\{2, 3\}$ on input a because in N, state 1 goes nowhere on input a and state 2 goes to both 2 and 3 on input a

NFA with n states $\rightarrow$ DFA with 2^n states.

5.2 Closure Under Regular Operations

Theorem 9. The class of regular languages is closed under the union operation.

Proof Idea. We have regular languages A_1 and A_2 and want to prove that $A_1 \cup A_2$ is regular. The idea is to take 2 NFAs, N_1 and N_2, that accept A_1 and A_2 respectively, and combine them into a single new NFA N.



Proof. Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1 and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$.

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$

   The states of N are all the states of N_1 and N_2 with the addition of a new start state q_0.

2. The state q_0 is the start state of N.

3. The set of accepting states $F = F_1 \cup F_2$.

   The accepting states of N are all the accepting states of N_1 and N_2. That way, N accepts if either N_1 or N_2 accepts.

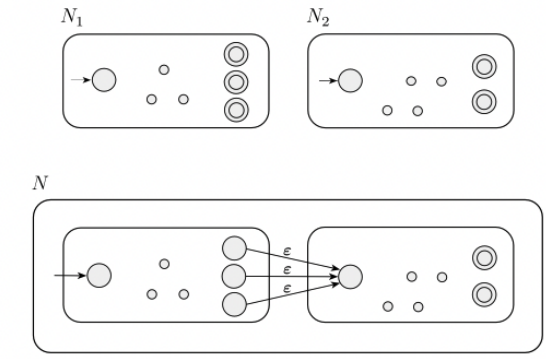
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \\ \delta_2(q,a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$

□

Theorem 10. The class of regular languages is closed under the concatenation operation.

Proof Idea. We have regular languages A_1 and A_2 and want to prove that $A_1 \circ A_2$ is regular. The idea is to take 2 NFAs, N_1 and N_2, and combine them into a single new NFA N (like how we did in the union closure proof, but with a few changes).



Proof. Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1 and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \circ A_2$.

1. $Q = Q_1 \cup Q_2$

   The states of N are all the states of N_1 and N_2.

2. The state q_1 is the start state of N_1.

3. The set of accepting states F_2 are the same as the accepting states of N_2.

4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

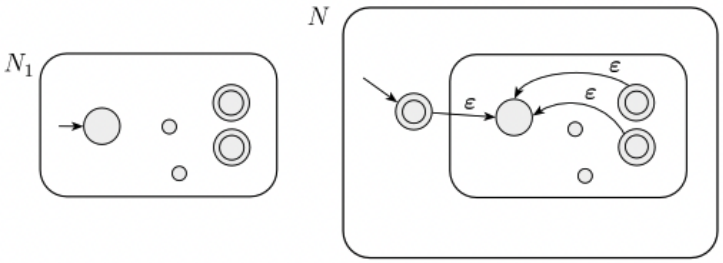
$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\} & q \in F_1 \text{ and } a \in \varepsilon \\ \delta_2(q,a) & q \in Q_2 \end{cases}$$

□

Theorem 11. The class of regular languages is closed under the Kleene star operation.

Proof Idea. We have a regular language A and want to prove that A_1^* also is regular. We take an NFA N_1 for A_1 and modify it to recognize A_1^* as shown. The resulting NFA N will accept its input whenever it can be broken into several pieces and N accepts each piece.

We can construct N like N_1 with additional ε arrows returning to the start state from the accepting states. This way when the processing gets to the end of a piece that N_1 accepts, the machine N has the option of jumping back to the start state to try to read another piece that N_1 accepts.



Proof. Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1. Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize A_1^*.

1. $Q = \{q_0\} \cup Q_1$

   The states of N are the states of N_1 plus a new start state.

2. The state q_0 is the new start state.

3. $F = \{q_0\} \cup F_1$

   The accepting states are the old accepting states plus the new start state.

4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ and } a \notin \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1 \text{ and } a \in \varepsilon \\ \{q_1\} & q = q_0 \text{ and } a \in \varepsilon \\ \emptyset & q = q_0 \text{ and } a \notin \varepsilon \end{cases}$$

□

6 Lecture 6: Regular Expressions

6.1 Equivalence Of NFAs and DFAs (cont.)

Theorem 12. Every NFA has an equivalent DFA.

Proof. Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A. We construct DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A. Before doing the full construction, first consider the easier case when N has no ε arrows. We will take ε into account later.

1. $Q' = \mathcal{P}(Q)$ (the set of subsets of Q)

   Every state of M is a set of states of N.

2. Σ (the alphabet) doesn't change

3. For $R \in Q'$, and $a \in \Sigma$, let $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$

   If R is a state of M, it is also a set of states of N. When M reads a symbol a in state R, it shows where A takes each state in R. Because each state may go to a set of states, we take the union of all these sets.

   $$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$

4. $q_0' = \{q_0\}$

   M starts in the state corresponding to the collection containing just the start state of N.

5. $F' = \{R \in Q' \mid R \text{ contains an accepting state of } N\}$

   The machine M accepts if one of the possible states that N could be in at this point is an accepting state.

Now consider the ε arrows. For any state R of M, we define $E(R)$ to be the collection of states that can be reached from members of R by going only along ε arrows, including members of R themselves. Formally, for $R \subseteq Q$, let

$$E(R) = \{q \mid q \text{ can be reached from } R \text{ by traveling along 0 or more } \varepsilon \text{ arrows}\}$$

Then we modify the transition function of M to place additional fingers on all states that can be reached by going along ε arrows after every step. Replacing $\delta(r, a)$ by $E(\delta(r, a))$ achieves this. Finally, we need to modify the start state of M to move the fingers initially to all possible states that can be reached from the start state of N along the ε arrows.

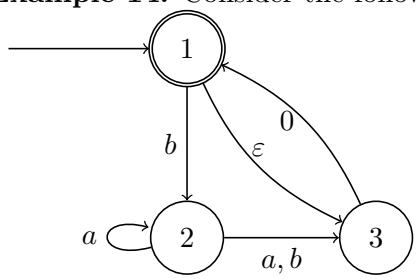
The changes mentioned above to account for ε arrows are shown below:

3. $\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some } r \in R\}$

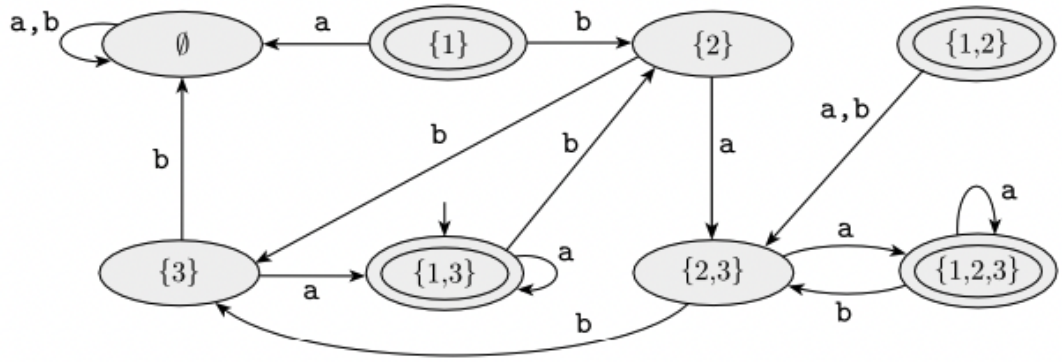
4. $q_0' = E(\{q_0\})$

□

Example 14. Consider the following NFA N:



Note that the DFA will have 8 states, one for each subset of the states of N. The DFA and its transitions are shown below:



The NFA's start state is 1, so the DFA's start state is $E(\{1\}) = \{1, 3\}$ (the set of states reachable from 1 by travelling along ε arrows and 1 itself). The NFA's accepting state is 1, so the DFA's accepting states are all sets of states that include 1: $\{\{1\}, \{1, 2\}, \{1, 3\}, \{1, 2, 3\}\}$

As for D's transition function, each of D's states goes to one place on input a and one place on input b (by definition of DFA). We will illustrate a few.

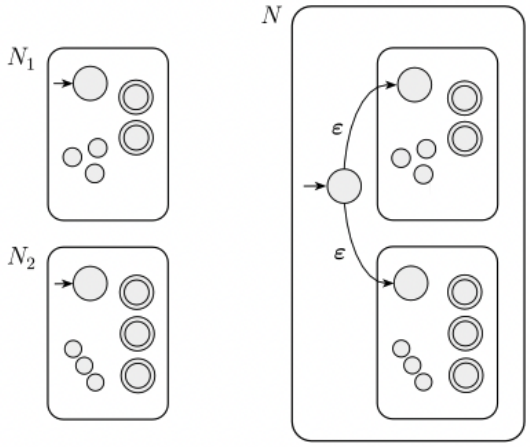
- in D, state $\{2\}$ goes to $\{2, 3\}$ on input a because in N, state 2 goes to both 2 and 3 on input a.
- in D, state $\{1\}$ goes to $\emptyset$ on input a because no a arrows exit it.
- in D, state $\{1, 2\}$ goes to $\{2, 3\}$ on input a because in N, state 1 goes nowhere on input a and state 2 goes to both 2 and 3 on input a

NFA with n states $\rightarrow$ DFA with 2^n states.

6.2 Closure Under Regular Operations

Theorem 13. The class of regular languages is closed under the union operation.

Proof Idea. We have regular languages A_1 and A_2 and want to prove that $A_1 \cup A_2$ is regular. The idea is to take 2 NFAs, N_1 and N_2, that accept A_1 and A_2 respectively, and combine them into a single new NFA N.



Proof. Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1 and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$.

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$

   The states of N are all the states of N_1 and N_2 with the addition of a new start state q_0.

2. The state q_0 is the start state of N.

3. The set of accepting states $F = F_1 \cup F_2$.

   The accepting states of N are all the accepting states of N_1 and N_2. That way, N accepts if either N_1 or N_2 accepts.

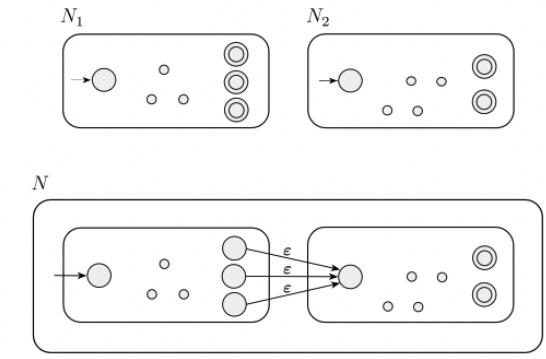
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \\ \delta_2(q,a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$

□

Theorem 14. The class of regular languages is closed under the concatenation operation.

Proof Idea. We have regular languages A_1 and A_2 and want to prove that $A_1 \circ A_2$ is regular. The idea is to take 2 NFAs, N_1 and N_2, and combine them into a single new NFA N (like how we did in the union closure proof, but with a few changes).



Proof. Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1 and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \circ A_2$.

1. $Q = Q_1 \cup Q_2$

   The states of N are all the states of N_1 and N_2.

2. The state q_1 is the start state of N_1.

3. The set of accepting states F_2 are the same as the accepting states of N_2.

4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

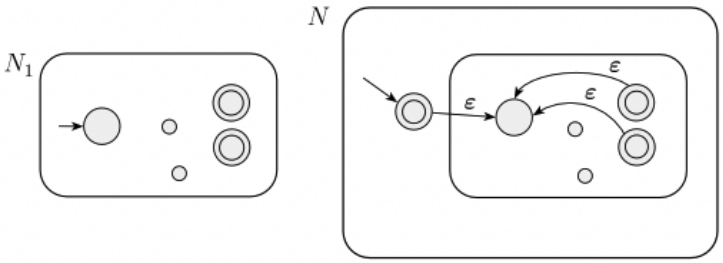
$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\} & q \in F_1 \text{ and } a \in \varepsilon \\ \delta_2(q,a) & q \in Q_2 \end{cases}$$

□

Theorem 15. The class of regular languages is closed under the Kleene star operation.

Proof Idea. We have a regular language A and want to prove that A_1^* also is regular. We take an NFA N_1 for A_1 and modify it to recognize A_1^* as shown. The resulting NFA N will accept its input whenever it can be broken into several pieces and N accepts each piece.

We can construct N like N_1 with additional ε arrows returning to the start state from the accepting states. This way when the processing gets to the end of a piece that N_1 accepts, the machine N has the option of jumping back to the start state to try to read another piece that N_1 accepts.



Proof. Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1. Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize A_1^*.

1. $Q = \{q_0\} \cup Q_1$

   The states of N are the states of N_1 plus a new start state.

2. The state q_0 is the new start state.

3. $F = \{q_0\} \cup F_1$

   The accepting states are the old accepting states plus the new start state.

4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ and } a \notin \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1 \text{ and } a \in \varepsilon \\ \{q_1\} & q = q_0 \text{ and } a \in \varepsilon \\ \emptyset & q = q_0 \text{ and } a \notin \varepsilon \end{cases}$$

□

7 Lecture 7: Nonregular Languages

7.1 Example: Nonregular Language

Example 15. Consider the language $L = \{O^n 1^n \mid n \geq 0\}$.

If we attempt to find a DFA that recognizes L, we find that the machine needs to remember how many 0s have been read so far. Because the number of 0s is not limited, the machine will have to track an unlimited number of possibilities (but this can't be done with finite states). Therefore, L is not a regular language.

Example 16. Consider the language $A = \{w \mid w \text{ has an equal number of 0s and 1s}\}$ and $B = \{w \mid w \text{ has an equal number of occurrences of 01 and 10 as substrings}\}$.
A is not regular but B is. Below, we will use the pumping lemma to show how to prove that certain languages are not regular.

7.2 Pumping Lemma For Regular Languages

Theorem 16. If A is a regular language, then there is a number p (pumping length) where if s is any string in A of length $\geq p$, then s may be divided into 3 pieces, $s = xyz$, satisfying the following conditions:

- for each $i \geq 0, xy^i z \in A$
- $|y| > 0$
- $|xy| \leq p$

Proof. If A is regular then there exists a DFA M accepting A.

Let $M = (Q, \Sigma, \delta, q_1, F)$ be a DFA recognizing A and p be the number of states of M.

Let $s = s_1 s_2 ... s_n$ be a string in A of length n, where $n \geq p$. Let $r_1, ..., r_{n+1}$ be the sequence of states that M enters while processing s, so $r_{i+1} = \delta(r_i, s_i)$ for $1 \leq i \leq n$. This sequence has length $n + 1$, which is at least $p + 1$. Among the first $p + 1$ elements in the sequence, 2 must be the same state, by the pigeonhole principle. Call the first of these r_j and the second r_l. Because r_l occurs among the first $p+1$ places in a seuqnce starting at r_1, we have $l \leq p+1$.

Now let $x = s_1 \cdots s_j - 1, y = s_j \cdots s_{l-1}, z = s_l \cdots s_n$. As x takes M from r_1 to r_j, y takes M from r_j to r_j, and z take sM from r_j to r_{n+1}, which is an accepting state so M must accept $xy^i z$ for $i \geq 0$. We know that $j \neq l$, so $|y| > 0$, and $l \leq p + 1$, so $|xy| \leq p$, and have satisfied all conditions of the pumping lemma.

□

7.2.1 Examples: Using Pumping Lemma To Prove Nonregularity

Example 17. Consider the language B be the language $\{0^n1^n \mid n \geq 0\}$.

Proof. Assume that B is regular.

Let p be the pumping length given by the pumping lemma. Choose s to be a string 0^p1^p. Because s is a member of B and s has length more than p, the pumping lemma guarantees that s can be split into 3 pieces, $s = xyz$, where for any $i \geq 0$ the string xy^iz is in B. Let us consider 3 cases to show that this result is impossible:

1. The string y consists only of 0s. For example, $xyyz$ (when $i = 2$) has more 0s than 1s, so it is in the language B. This breaks condition 1 of the pumping lemma.
2. The string y consists only of 1s. This also breaks condition 1.
3. The string y consists of both 0s and 1s. The string $xyyz$ may have the same number of 0s and 1s, but they are out of order. So the string is not in B and breaks condition 1 of the pumping lemma.

□

Example 18. Let $C = \{w \mid w\text{has an equal number of 0s and 1s}\}$. Prove that C is not regular.

Proof. Assume that C is regular.

Let p be the pumping length given by the pumping lemma. Choose s to be the string 0^p1^p. Because s is a member of C and s has length more than p, the pumping lemma guarantees that s can be split into 3 pieces, $s = xyz$, where for any $i \geq 0$ the string xy^iz is in C.

Let x and z be the empty string and y be the string 0^p1^p. Then xy^iz always has an equal number of 0s and 1s and is in C.

BUT THIS IS WRONG! CONDITION 3 OF THE PUMPING LEMMA COMES INTO PLAY.

By pumping s, it must be divided so that $|xy| \leq p$. That means the string we selected $s = 0^p1^p$ cannot be pumped. If $|xy| \leq p$, then y must consist only of 0s, so $xyyz \notin C$.

s cannot be pumped and therefore C is not regular.

□

Example 19. Consider the language $D = \{ww \mid w \in \{0,1\}^*\}$. Prove that D is not regular.

Proof. Assume that F is regular.

Let s be the string 0^p10^p1. Because s is a member of F and s has length more than p, the pupming lemma guarantees that s can be split into 3 pieces $s = xyz$ satisfying the pumping lemma.

Condition 3 of the pumping lemma is important again. We cannot let x and z be of length 0 because of condition 3. This means that y must consist of only 0s, so $xyyz \notin D$.

□

Example 20. Consider the language $\tilde{L_3} = \{w_1w_2 \mid w_1 \neq w_2; w_1, w_2 \in \Sigma^*\}$.

Then $L_3 = (\Sigma\Sigma)^* - \tilde{L_3} = (\Sigma\Sigma)^* \cap \tilde{L_3}$

7.2.2 Example: Pumping Lemma On Unary Languages

Example 21. Consider the language $E = \{1^{n^2} \mid n \geq 0\}$. E essentially contain all strings of 1s whose lengths is a perfect square. We will show that E is not regular.

Proof. By contradiction:

Assume that E is regular. Let p be the pumping length given by the pumping lemma. Let s be the string 1^{p^2}. Because s is a member of E and s has length at least p, the pumping lemma guarantees that s can be split into $s = xyz$, where for any $i \geq 0$, the string xy^iz is in D.

Consider the 2 strings xyz and xy^2z. These strings differ from each other by a single repetition of y and their lengths differ by the length of y. By condition 3 of the pumping lemma, $|xy| \leq p$ and thus $|y| \leq p$. We have $|xyz| = p^2$ and so $|xy^2z| \leq p^2 + p$. But $p^2 + p < p^2 + 2p + 1 = (p+1)^2$. Condition 2 implies that y is not an empty string so $|xy^2z| > p^2$. The length of xy^2z lies strictly between the consecutive perfect squares p^2 and $(p+1)^2$. So the length cannot be a perfect square, see that $xy^2z \notin E$, and conclude that E is not regular.

□

7.2.3 Example: Pumping Down

Example 22. Consider the language $F = \{0^j 1^j \mid i > j\}$. Show that F is not regular.

Proof. Assume that F is not regular.

Let p be the pumping length of F given by the pumping lemma. By condition 3, y consists only of 0s. Let's example the string $xyyz$ to see whether it can be in F. Adding an extra copy of y increases the number of 0s. But F contains all strings in 0^*1^* that have more 0s and 1s, so increasing the number of 0s will still give the string in F. There is no contradiction. Try the following.

The pumping lemma states that $xy^iz \in F$ even when $i = 0$, so let's consider the string $xy^0z = xz$. Removing string y decreases the number of 0s in s. Remember than s has just one more 0 than 1. Therefore, xz cannot have more 0s than 1s, so it cannot be a member of F. This is a contradiction.

□

8 Lecture 8: Context-Free Languages (CFLs)

8.1 Context-Free Grammars (CFGs)

- A grammar consists of collection of **substitution rules**, called **productions**.
- Each rule appears as a line in the grammar, comprising a symbol and a string separated by an arrow
- The symbol is called a **variable**.
- The string consists of variables and other symbols called **terminals**.
- The variables are often represented by capital letters. The terminals are often represented by lowercase letters, numbers, or special symbols.
- One variable is designated as the **start variable**.

You use a grammar to describe a language by generating each string of that language in the following manner.

1. Write down the start variable. It is the variable on the left-hand side of the top rule, unless specified otherwise.
2. Find a variable that is written down and a rule that starts with that variable. Replace the written down variable with the right-hand side of that rule.
3. Repeat step 2 until no variables remain.

Example 23. Consider the grammar G_1:

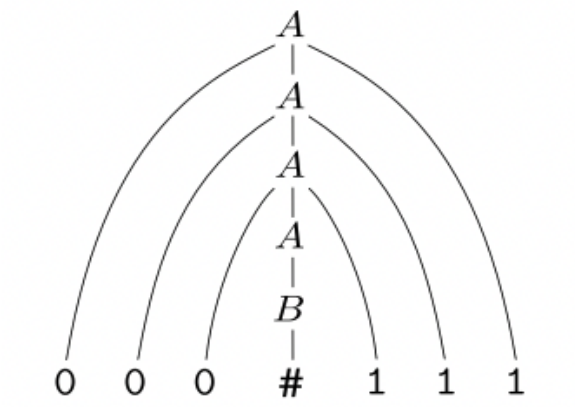
$$A \to 0A1$$
$$A \to B$$
$$B \to \#$$

- $0, 1, \#$ are terminals
- let A be the start symbol
- $A \to 0A1 \to 00A11 \to 000B111 \to 000\#111$ is a derivation
- $000\#111$ is a string of terminals

We can say that G_1 generates $0^3\#1^3$.

Define $L(G)$ to be all strings generated by G. We can prove $L(G_1) = \{0^n\#1^n \mid n \geq 0\}$. Then $L(G_1)$ is a CFL because it is generated by a CFG. We say that $S = 000\#111 \in L(G_1)$.

The following is a parse tree that describes the derivation of S.



Pulling all the leaves from the tree (shown by thin lines), we can see the string that is generated. Every word that is generated/derived from the start state has a parse tree. Grammars and parse tress come from linguistics.

For example, English has a grammar and can be parsed.

8.1.1 Formal Definition Of A Context-Free Grammar

Definition 7. A **context-free grammar** is a 4-tuple (V, Σ, R, S), where:

- V is a finite set called the **variables**
- Σ is a finite set, disjoint from V, called the **terminals**
- R is a finite set of **rules**, with each rule being a variable and a string of variables and terminals, shown below

$$\text{variable} \rightarrow \text{string}$$

- $S \in V$ is the start variable

If u, v, w are strings of variables and terminals, and $A \rightarrow w$ is a rule of the grammar, we say that uAv yields uwv, written $uAv \Rightarrow uwv$. Say that u derives v, written $u \Rightarrow v$, if $u = v$ or if a sequence $u_1, u_2, ..., u_k$ exists for $k \geq 0$ and

$$u \Rightarrow u_1 \Rightarrow ... \Rightarrow u_k \Rightarrow v$$

Example 24. Consider the grammar $G_2 = (\{S\}, \{a, b\}R, S)$. The set of rules R is

$$S \rightarrow aSb \mid SS \mid \varepsilon$$

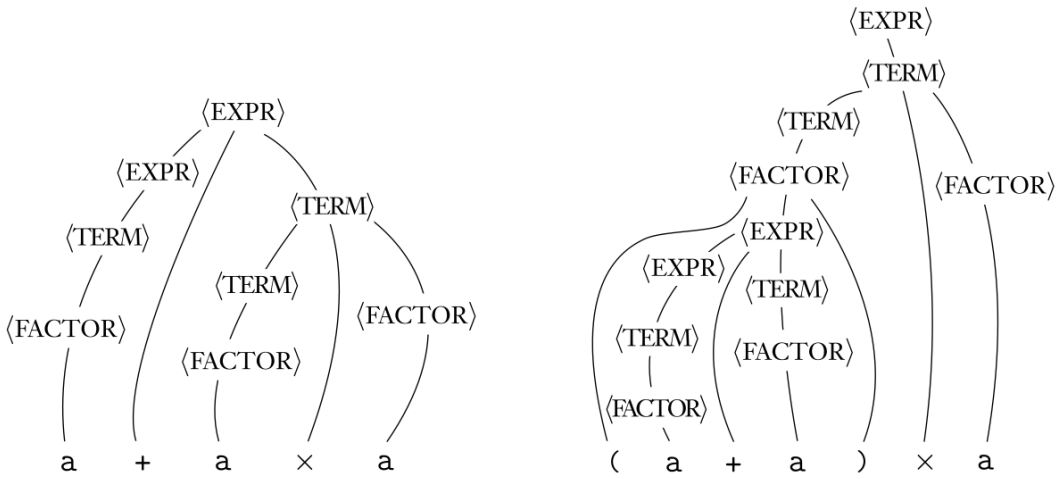
This grammar generates strings such as $abab$, $aaabbb$, and $aabaab$. Think of a as '(' and b as ')'. Viewed in this way, $L(G_3)$ is the language of all strings of properly nested (balanced) parentheses. Note that the language may contain the empty string ε.

Example 25. Consider the grammar $G_3 = (V, \Sigma, R, \langle EXPR \rangle)$.

V is $\{\langle EXPR \rangle, \langle TERM \rangle, \langle FACTOR \rangle\}$ and Σ is $\{a, +, \times, (,)\}$

$$\begin{aligned} \langle EXPR \rangle &\to \langle EXPR \rangle + \langle TERM \rangle \mid \langle TERM \rangle \\ \langle TERM \rangle &\to \langle TERM \rangle \times \langle FACTOR \rangle \mid \langle FACTOR \rangle \\ \langle FACTOR \rangle &\to (\langle EXPR \rangle) \mid a \end{aligned}$$

The two strings $a + a \times a$ and $(a + a) \times a$ are both generated with grammar G_3. The parse trees are shown below:



8.1.2 Designing Context-Free Grammars

Many CFLs are unions of simpler CFLs.

Example 26. To get a grammar for the language $\{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\}$, first construct the grammar.

$S_1 \to 0S_1 1 \mid \varepsilon$ for language $\{0^n 1^n \mid n \geq 0\}$ $S_2 \to 1S_2 0 \mid \varepsilon$ for language $\{1^n 0^n \mid n \geq 0\}$

Then add the rule $S \to S_1 \mid S_2$ to give the grammar:

$$\begin{aligned} S &\to S_1 \mid S_2 \\ S_1 &\to 0S_1 1 \mid \varepsilon \\ S_2 &\to 1S_2 0 \mid \varepsilon \end{aligned}$$

Constructing a CFG for a language that happens to be regular is easy if you can first construct a DFA for that language. You can convert any DFA into an equivalent CFG by:

1. Make a variable R_i for each state q_i of the DFA.
2. Add the rule $R_i \to aR_j$ to the CFG if $\delta(q_i, a) = q_j$ is the transition in the DFA.
3. Add the rule $R_i \to \varepsilon$ if q_i is an accepting state of the DFA.
4. Make R_0 the start variable of the grammar, where q_0 is the start state of the machine

Certain CFLs contain strings with two substrings that are "linked" in the sense that a machine for such a language would need to remember an unbounded amount of information about one of the substrings to verify that it corresponds properly to the other substring.

Example 27. Consider the language $\{0^n1^n \mid n \geq 0\}$.

The machine would need to remember the number of 0s in order to verify that it equals the number of 1s. You can construct a CFG to handle this situation by using a rule of the form $R \to uRv$, which generates strings wherein the portion containing the u's correspond to the portion containing the v's.

In more complex languages, the strings may contain certain structures that appear recursively as part of other (or the same) structures. An example is Example 3, where the grammar generates arithmetic expressions. Any time a symbol a appears, an entire parenthesized expression might appear recursively instead. To do this, put the variable symbol generating the structure in the location of the rules corresponding to where that structure may recursively appear.

8.1.3 Nondeterminism With CFGs

CFGs are in some sense nondeterministic because at each step, we can choose:

1. which rule to apply
2. which variable to apply the rule

Example 28. Consider the string $0AB1A0$.

We can eliminate nondeterminism for (2) with leftmost derivation (applying the rule to the leftmost variable). However we cannot entirely eliminate nondeterminism because there can be multiple choices for the rule from a variable such as $A \to w_1 \mid w_2 \mid w_3 \mid w_4$.

8.1.4 Ambiguity

If the grammar generates the same string in several ways, we say that the string is derived **ambiguously**. If a grammar generates some string ambiguously, we say that the grammar is **ambiguous**.

More formally, if $w \in L(G)$, then $S \overset{*}{\Rightarrow} w$. Sometimes $S \overset{*}{\Rightarrow} w$ in more than 1 way, and if this happens with G, then G is ambiguous.

Example 29. Let $L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$.

It can be shown that L is generated by a CFG, and L is inherently ambiguous. Consider the following grammar G_5 for L:

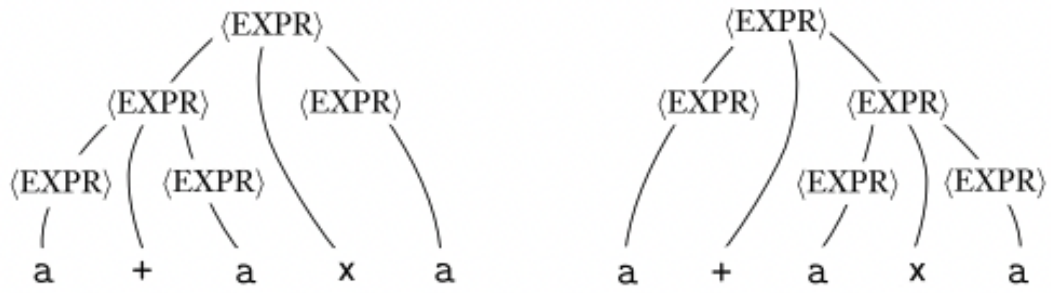
$$\begin{aligned} S &\to S_1 C \mid A S_2 \\ S_1 &\to a S_1 b \mid \varepsilon \\ S_2 &\to b S_2 c \mid \varepsilon \\ A &\to aA \mid \varepsilon \\ C &\to Cc \mid \varepsilon \end{aligned}$$

A string such as *aabbcc* can be generated through multiple different paths.

Example 30. Consider grammar $G_4 = (V, \Sigma, R, \langle EXPR \rangle)$.

$$\langle EXPR \rangle \to \langle EXPR \rangle + \langle EXPR \rangle \mid \langle EXPR \rangle \times \langle EXPR \rangle \mid (\langle EXPR \rangle) \mid a$$

The following is two parse trees for the string $a + a \times a$ in grammar G_4.



Note that grammar G_4 does not capture the usual precedence relations ($\times$ before $+$).