Automata Midterm 1

tyang27

February 2019

1 Regular Languages

- Σ - alphabet, finite set of symbols
- String over Σ - finite sequence of symbols from alphabet
- ε - empty string
- Language over Σ - set of all possible strings over Σ
- Machine accepts string if consumes it and lands in a final state
  - Let $w = w_1 w_2 \dots w_n$
  - M accepts w if there is a sequence of states $r_1, r_2, \dots, r_n \in Q$ if:
  - Valid start state - $r_0 = q_0$
  - Valid transitions - $\delta(r_i, w_{i+1}) = r_{i+1}$
  - Ends in final state - $r_n \in F$
- Machine recognizes language A if it accepts all strings in it, and does not accept any string outside of it, implies that A is a regular language
- Union - $A \cup B = \{x | x \in A \text{ or } x \in B\}$
- Concatenation - $A \circ B = \{xy | x \in A, y \in B\}$
- Kleene star - $A^* = \{x_1 x_2 \dots x_k | k \geq 0, x_i \in A\}$

1.1 Deterministic Finite Automata

$M = (Q, \Sigma, \delta, q_0, F)$

- Q = set of states
- Σ = alphabet, set of symbols
- $\delta : Q \times \Sigma \to Q$ = transition function, maps states, symbols to new states
- $q_0 \in Q$ = start state
- $F \subseteq Q$ = set of final states

1.2 Nondeterministic Finite Automata

$M = (Q, \Sigma, \delta, q_0, F)$

- Q = states
- Σ = alphabet
- $\delta : Q \times \Sigma_\varepsilon \to P(Q)$ = transition function, maps states, symbols/varepsilon to set of new states
- $q_0 \in Q$ = start state
- $F \subseteq Q$ = set of final states

Modified delta transition allows us to 1) varepsilon jump to states, 2) be at multiple states at once, and 3) have dead states

1.3 Regular Expressions

Base cases:

- $a \in \Sigma$
- ε
- $\emptyset$

Operations:

- $R_1 \cup R_2$
- $R_1 \circ R_2$
- R_1^*

1.4 Converting between DFA/NFA/RE

1.4.1 DFA to NFA

Trivial

1.4.2 NFA to DFA

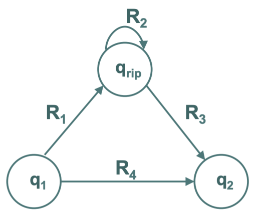
Subset construction

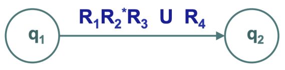
- $Q' = P(Q)$ - NFA states are subset of DFA states
- $\Sigma' = \Sigma$
- $\delta'(R, a) = \bigcup_{r \in R} E(\delta(r, a))$ if $R \in Q'$ - For each NFA state (a subset), we follow the DFA transition of each element in the set (a new subset). Apply varepsilon transition to the result of original transition.
- $q_0' = \{q_0\}$
- $F' = \{R \in Q' | \exists r \in R, r \in F\}$ - Final NFA states contain some original final state

1.4.3 NFA to RE

Ripping method using GNFA.

1. Add new start state that varepsilon jumps to original start state
2. Add new final state that all original final states varepsilon jump to
3. Replace commas with unions

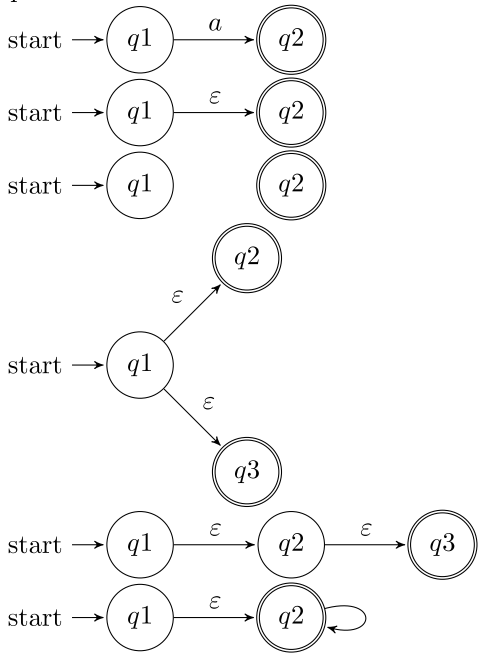


4. 

5. Rip rip rip! Using the formula above.

1.4.4 RE to NFA

Use process similar to induction.



1.5 Closure

1.5.1 Union using NFA

- $Q = Q_1 \cup Q_2 \cup \{q_{new}\}$
- $\Sigma = \Sigma_1 = \Sigma_2$
- $\delta(r, a) = \begin{cases} \delta_1(r, a) & \text{if } r \in Q_1 \\ \delta_2(r, a) & \text{if } r \in Q_2 \\ \{q_{01}, q_{02}\} & \text{if } r = q_{new}, a = \varepsilon \\ \emptyset & \text{if } r = q_{new}, a \neq \varepsilon \end{cases}$

- $q_0 = q_{new}$
- $F = F_1 \cup F_2$

1.6 Pumping Lemma

1.6.1 Formal

If A is a regular language, then there exists an integer p where if s from A is any string of length at least p, then s may be divided into three pieces, $s = xyz$, such that:

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

1.6.2 Informal

Given a regular language A, the pumping lemma holds. The pumping lemma says that if we have a long string (a string with length greater than the number of states), we know that there must be some loop in the machine, and we can partition the machine into three pieces, xyz, where y is the loop.

- If we pump the loop 0 or more times, it is still in the language.
- The loop exists, in the sense that it has at least one state.
- The leadup to the loop and the loop combined fit into the machine.

1.7 Proving nonregularity

FSOC, assume that A is regular. Then, the properties of pumping lemma apply, enumerate. Consider a long string $s = xyz$ (choose s) with an arbitrary partition up to p that satisfies pumping lemma. If we pump it more or less (choose i), then it is no longer in the language.

1.8 Other notes

- To show that reversed strings are closed under regular languages, we can just swap the arrows in a DFA, but cannot do so in NFA. This is because NFA has dead states.

2 Context Free Languages

2.1 Properties

- Regular languages are a subset of context free languages.
- Consistency - generates all strings in language.
- Consistency - generates only strings in language.
- Ambiguous - string derived from grammar in fundamentally different ways. Show that either 1) give different parse trees or 2) different leftmost derivations.

2.2 Context Free Grammar

$G = (V, \Sigma, R, S)$

- V - variables, finite set of symbols, typically caps
- Σ - terminals, finite set of symbols, disjoint from variables ($V \cap \Sigma = \emptyset$), typically lowercase
- R - finite set of rules, e.g. $A \rightarrow B$
- $S \in V$ - start symbol

Start with S, derive by repeating rules until no more variables, only terminals.

2.2.1 Parse Tree

- Leaves are terminals.
- Internal nodes are variables.
- Branches correspond to concatenation

2.2.2 Chomsky Normal Form

Equivalent to CFG. Basically, CFG with certain conditions. Conditions:

- $A \to BC$
- $A \to a$
- B, C cannot be S
- Only S can go to ε

2.3 Converting from RE to CFG

- $a = S \to a$
- $\varepsilon = S \to \varepsilon$
- $\emptyset = S \to S$
- $R_1 \cup R_2 = S \to S_1 | S_2$
- $R_1 \circ R_2 = S \to S_1 S_2$
- $R_1^* = S \to S_1 S$

2.4 Push Down Automata

$P = (Q, \Sigma, \Gamma, \delta, q_0, F)$

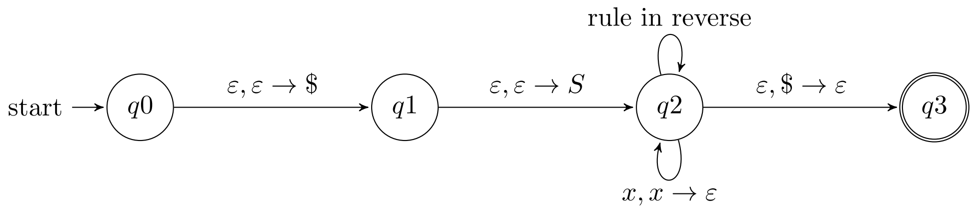
- Q = finite set of states
- Σ = alphabet, finite set of symbols
- Γ = stack alphabet, finite set of symbols
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$ = transition function, e.g. $a, b \to c$
- q_0 = start state
- $F \subseteq Q$ = set of final states

2.4.1 Useful transitions

- symbol a, pop $b \to$ push c = take arrow if symbol in string is a and top of stack is b. Pop b off stack and push c.
- $a, \varepsilon \to \varepsilon$ = consume symbol, ignore stack
- $\varepsilon, \varepsilon \to b$ = consume no symbol, push onto stack
- $\varepsilon, b \to \varepsilon$ = consume no symbol, pop off of stack
- $\varepsilon, \varepsilon \to \varepsilon$ = consume no symbol, ignore stack

2.5 Converting from CFG to PDA

- The general idea is that we place an empty marker on the stack and on top of that, the reverse of all possible generations. We put rules that consume the input $x, x \to \varepsilon$ to check if we can uncover the \$, meaning that it is a valid string.

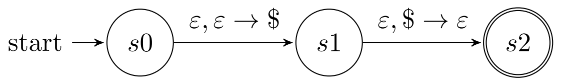

start
$q0$
$\varepsilon, \varepsilon \to \$$
$q1$
$\varepsilon, \varepsilon \to S$
rule in reverse
$q2$
$x, x \to \varepsilon$
$\varepsilon, \$ \to \varepsilon$
$q3$

2.6 Converting from PDA to CFG

A bit too involved, but know that it is possible.

2.7 Other notes

- Stacks lack notion of emptiness, so often use \$ symbol emptiness


start
$s0$
$\varepsilon, \varepsilon \to \$$
$s1$
$\varepsilon, \$ \to \varepsilon$
$s2$