Math 322

P: string$_1$ → string$_2$

Phrase Structure Grammars: Set of Productions

No restrictions on Non-terminals being replaced by strings

Type 1 Context Sensitive Grammars

Productions: $l\,A\,r \to l\,ab\,r$ (ab: string?; length 1 or more)

$aAB \to bC$ (not type 1)

(Outside of $S \to \varepsilon$ these productions make strings of same length or longer)

Type 2 Context Free Grammars

Productions: $A \to$ expression of one or more symbols (style non-term)

(Note: $S \to \varepsilon$ is still OK)

Type 3 Regular Grammars

Productions: $A \to a$ or $A \to aB$

Regular Expressions: Made by $(aw), (b|r)$ Kleene Closure

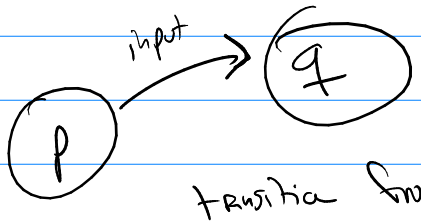
concatenation union

Finite State Machines

→ Q ≡ set of states

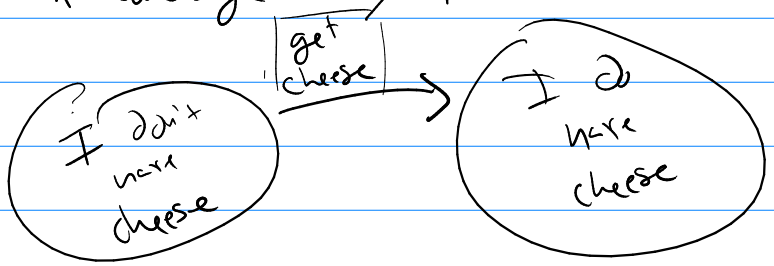
Σ ≡ input alphabet

as digraph



transition from state p to state q

states ≡ knowledge



inputs cause transitions from one state to another.

examples ① State machine with output

② State machine without output

(Finite-State Automata)

With Output

$$M = (|Q|, |\Sigma|, |O|, \delta, \gamma, s_0 \in Q)$$

Q: states; Σ: input sym; O: output sym; δ: transition functions; γ: output function; s_0: start state

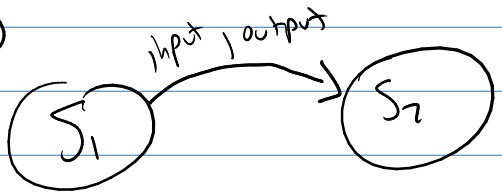
ex's

Candy Machine

$M = (Q, \Sigma, O, \delta, \gamma, S_0 \in Q)$

- Q: states
- Σ: input sym.
- O: output sym
- δ: transition
- γ: output function
- $S_0 \in Q$: start state

Useful when your machine has "knowledge" and returns things.



$\delta(S_1, \text{input}) = S_2$

$\gamma(S_1, \text{input}) = \text{output}$

ex 3 penny candy machine

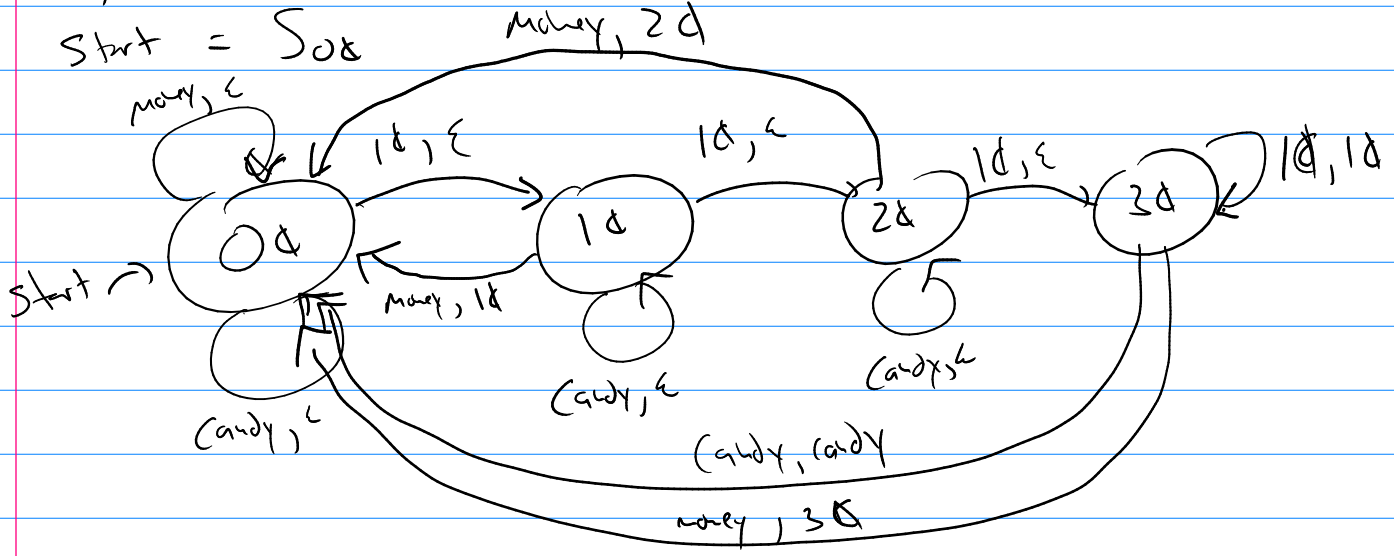
$Q = \{ S_{0¢}, S_{1¢}, S_{2¢}, S_{3¢} \}$

$\Sigma = \{ 1¢, \text{candy}, \text{money} \}$

$O = \{ \varepsilon, 1¢, 2¢, 3¢, \text{candy} \}$

δ, γ

start $= S_{0¢}$



Input: 1, 1, M, 1, C, C

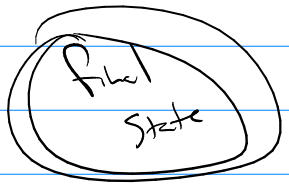
States: 0, 1, 2, 0, 1, 1, 1

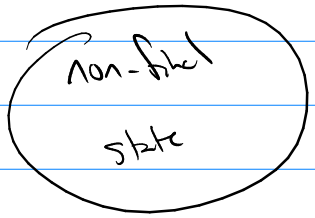
output: ε, ε, 2, ε, C, ε

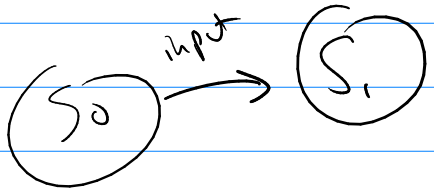
Finite State Automata (without output)

$$M = (Q, \Sigma, s_0 \in Q, F \subseteq Q, \delta)$$

F: final states





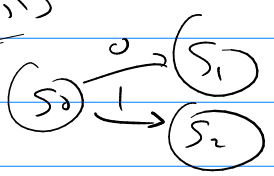


$\delta(s_0, \text{input}) = s_1$

useful: string recognition: if input string ends in final state = M recognizes string.

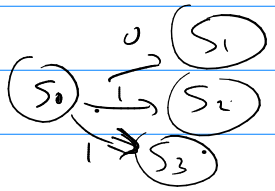
(A) Deterministic (DFA)

$\Sigma = \{0, 1\}$



$$\delta : Q \times \Sigma \to Q$$

(B) Non-Deterministic (NFA)



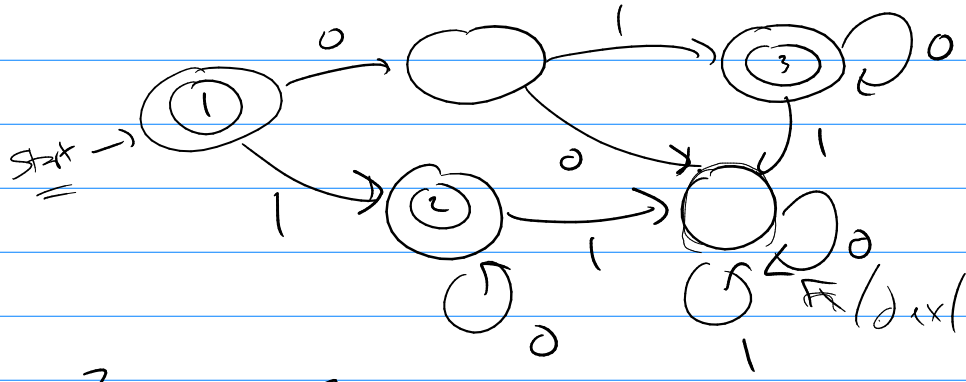
$$\delta : Q \times \Sigma \cup \{\varepsilon\} \to P(Q)$$

$\{s_2, s_3\}$

thm: For $L = L(NFA)$ there exists a DFA such that $L = L(DFA) = L(NFA)$

Ex3 L(DFA) , L(NDA)

DFA



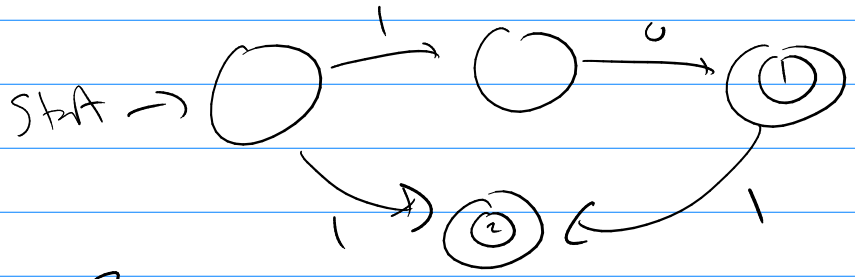
L(M) = ?

Finals:

① (Start) ε
② 10*
③ 010*

L(M) = ε | 10* | 010*

NFA



Finals ①? 10
②? 1 or 101

L(M) = 10 | 1 | 101

Thm Every language generated by a regular expression is recognized by an NFA.

Thm Every language generated by a regular grammar is recognized by an NFA.

$A \to aB$ | $(S_1) \xrightarrow{a} (S_2)$ | $A \to a$ | $(S_1) \xrightarrow{a} ((S_2))$

So regular language (NF) recognized by NFA.

type 3 So regular languages ≡ Finite State Automata

What about other languages?

type 2 → Context Free (NFA (+) pushdown)

type 1 → Context Sensitive (NFA (+) linear bounded type memory)

Any Phrase Structure Grammar?

NFA (+) infinite type memory

or Turing Machines!