CMPT 413
Computational Linguistics

Anoop Sarkar

`http://www.cs.sfu.ca/~anoop`

Formal Languages: Recap

- Symbols: a, b, c
- Alphabet : finite set of symbols $\Sigma = \{a, b\}$
- String: sequence of symbols   bab
- Empty string: ε Define: $\Sigma^{\varepsilon} = \Sigma \cup \{\varepsilon\}$
- Set of all strings: Σ^* cf. *The Library of Babel*, Jorge Luis Borges
- (Formal) Language: a set of strings

  $\{ a^n b^n : n > 0 \}$

Formal Grammars

- A formal grammar is a concise description of a formal language
- A formal grammar uses a specialized syntax
- For example, a **regular expression** is a concise description of a regular language

  $(a|b)^*abb$: is the set of all strings over the alphabet $\{a, b\}$ which end in *abb*

Regular Expressions: Definition

- Every symbol of $\Sigma \cup \{ \varepsilon \}$ is a regular expression
- If r_1 and r_2 are regular expressions, so are
  - Concatenation: $r_1 r_2$
  - Alternation: $r_1 | r_2$
  - Repetition: r_1*
- Nothing else is.
  - Grouping re’s: e.g. aa|bc vs. ((aa)|b)c

Regular Expressions: Examples

- Alphabet { V, C } V: vowel C: consonant
- A set of consonant-vowel sequences (CV|CCV)*
- All strings that do not contain "VC" as a substring C*V*
- Need a decision procedure: does a particular regular expression (regexp) accept an input string
- Provided by: Finite State Automata

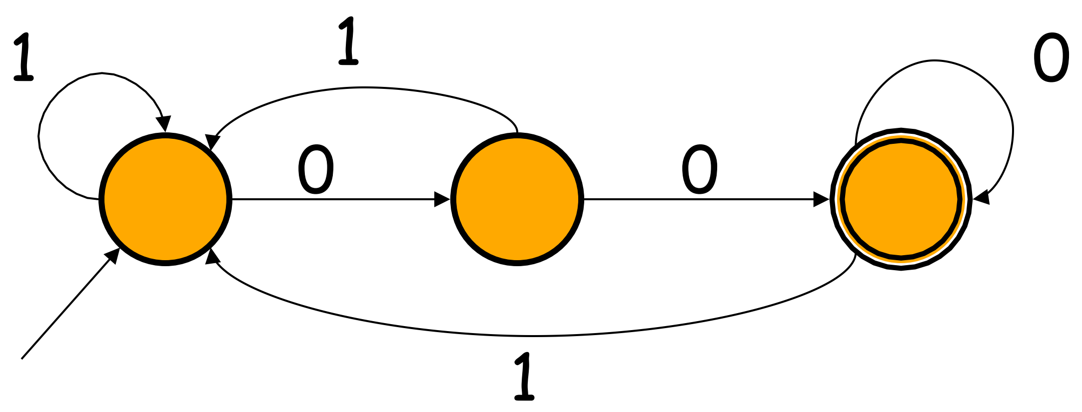
Finite Automata: Recap

- A set of states S
  - One start state q_0, zero or more final states F
- An alphabet $\sum$ of input symbols
- A transition function:
  - $\delta: S \times \Sigma \Rightarrow S$
- Example: $\delta(1, a) = 2$

A single state is deterministically chosen and so this kind of FA is called a **DFA** (*deterministic finite-state automata*)

Finite Automata: Example

- What regular expression does this automaton accept?



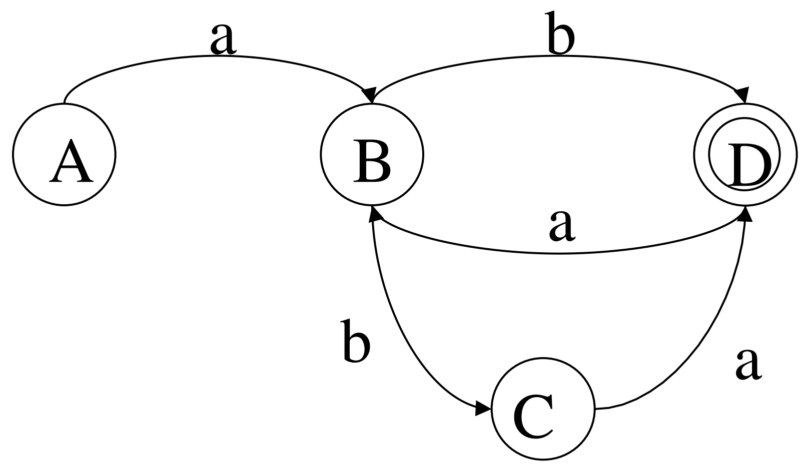
Answer: (0|1)*00

NFAs

- NFA: like a DFA, except
  - A transition can lead to more than one state, that is, $\delta: S \times \Sigma \Rightarrow 2^S$
  - One state is chosen non-deterministically
  - Transitions can be labeled with ε, meaning states can be reached without reading any input, that is,

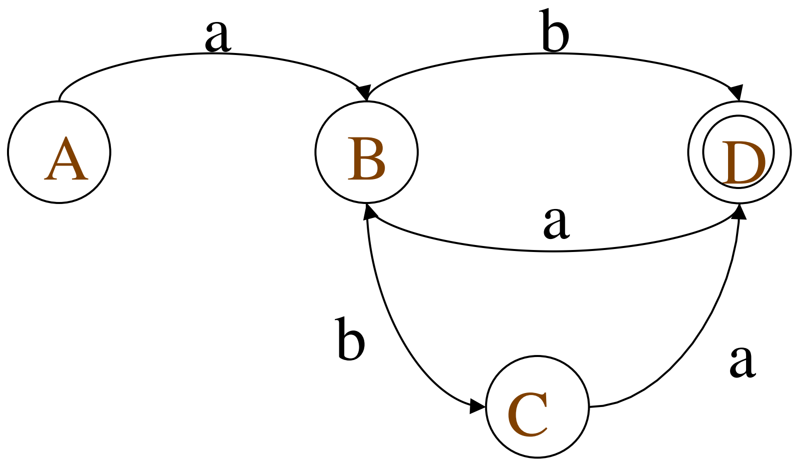
$$\delta: S \times \Sigma \cup \{ \varepsilon \} \Rightarrow 2^S$$

Recognition of strings (NFAs)



- Input string: aba#
- Recognition problem: Is input string in the language generated by the NFA?
- Recognition (without conversion to DFA) is also called *simulation* of NFA

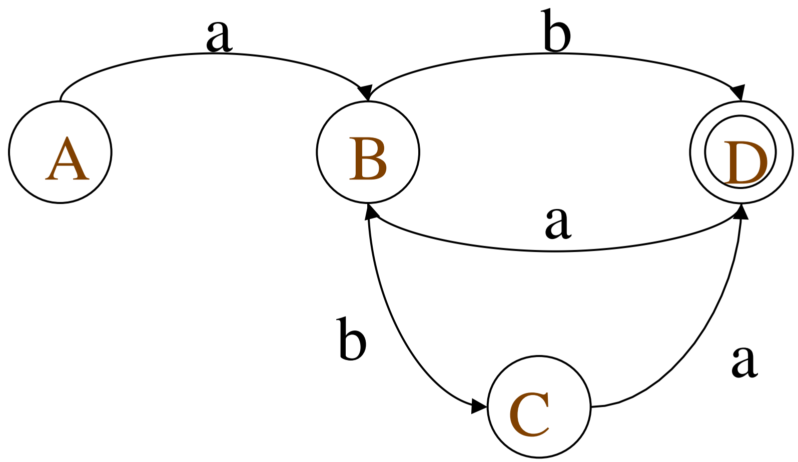
Recognition of strings (NFAs)



q is the transition function for the NFA

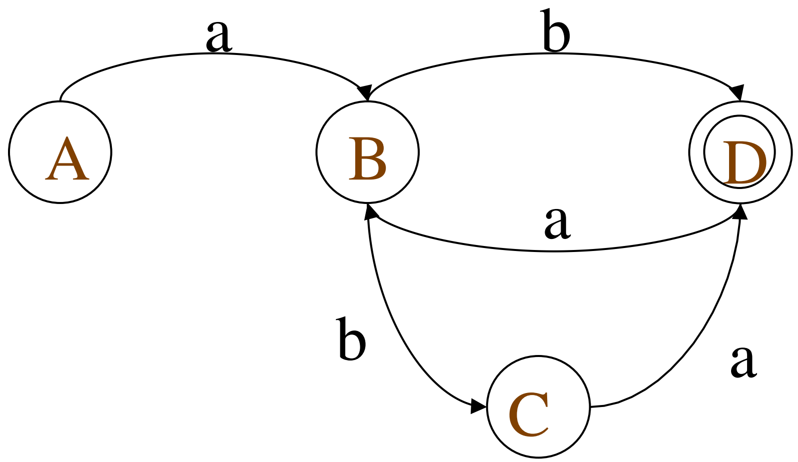
- Input tape: $_0$ a $_1$ b $_2$ a $_3$ # $_4$
- Start State: A   Agenda: { (A, 0) }
- Pop (A, 0) from Agenda
- q(A, a) = B,   Agenda: { (B, 1) }
- Pop (B, 1) from Agenda
- q(B, b) = { D, C }   Agenda: { (D, 2), (C, 2) }

Recognition of strings (NFAs)



- Input tape: $_0$ a $_1$ b $_2$ a $_3$ # $_4$
- Pop (D, 2) from Agenda
- q(D, a) = { B }   Agenda: { (B, 3), (C, 2) }
- Pop (B, 3) from Agenda: B is not a final state
- Pop (C, 2) from Agenda: if Agenda empty, reject
- q(C, a) = { D }   Agenda: { (D, 3) }

Recognition of strings (NFAs)



- Input tape: $_0$ a $_1$ b $_2$ a $_3$ # $_4$
- Pop (D, 3) from Agenda
- Is (D, 3) an accept item?
- Yes: D is a final state **and** 3 is index of the end-of-string marker #
- Return accept

Recognition of strings (NFAs)

```
function NDRecognize (tape[], q):
    Agenda = { (start-state, 0) }
    Current = (state, index) = pop(Agenda)
    while (true) {
        if (Current is an accept item) return accept
        else Agenda = Agenda ∪ GenStates(q, state, tape[index])
        if (Agenda is empty) return reject
        else Current = (state, index) = pop(Agenda)
    }
function GenStates (q, state, index):
    return { (q', index) : for all q' = q(state, ε) } ∪
           { (q', index+1) : for all q' = q(state, tape[index+1]) }
```


what if the input to this algorithm is a DFA?

Algorithms for FSMs
(finite-state machines)

- Recognition of a string in a regular language: is a string accepted by an NFA?
- Conversion of regular expressions to NFAs
- Determinization: converting NFA to DFA
- Converting an NFA into a regular expression
- Other useful *closure* properties: union, concatenation, Kleene closure, intersection