Lecture 10

Deterministic Finite Automata

CS 241: Foundations of Sequential Programs
Fall 2014

Troy Vasiga et al
University of Waterloo

Review

- formal languages give a theoretical basis for communication and organizing processes
- terminology (alphabet, word, language)
- specification vs. recognition
- studying language levels that increase in power/complexity
- regular languages are composed of union, concatenation and repetition

Recognizers: Finite Automata

Regular languages can be recognized by *finite automata*.
We begin with *deterministic finite automata*, also called DFAs.

- states
- transitions
- start state
- final states

DFA summary

- a.k.a. finite state machines
- start state
- final/accepting states
- implicit error state
- accepted and rejected words
- $L(M)$ – the language recognized by DFA M
- Notice that $L(M) = L(M')$ even though $M \neq M'$

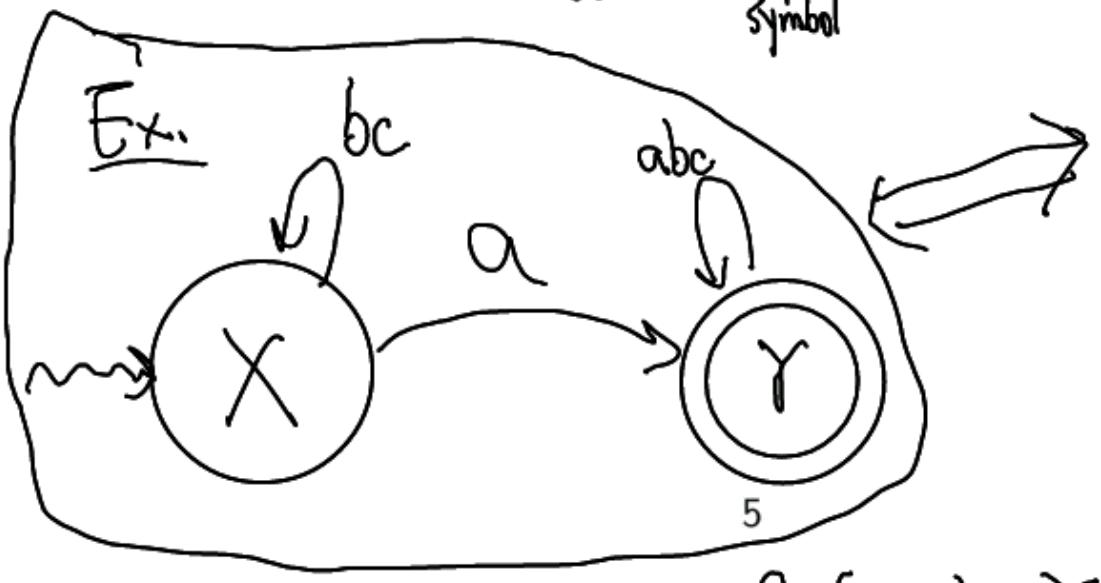
Formal Definition

A DFA is a 5-tuple $(\Sigma, Q, q_0, A, \delta)$ where

- finite alphabet Σ
- finite set of states Q
- start state q_0
- set of final/accepting states $A \subseteq Q$
- transition function: $\delta : Q \times \Sigma \to Q$

DFA Spec

δ: state, input symbol → state



$\Sigma = \{a, b, c\}$

$Q = \{X, Y\}$

$q_0 = X$

$A = \{Y\}$

$\delta(X, a) = Y$ $\delta(X, b) = X$

$\delta(X, c) = X$

$\delta(Y, a) = Y$ $\delta(Y, b) = Y$

$\delta(Y, c) = Y$

DFA Interpreter Algorithm

Input: A word $w = w_1 w_2 ... w_n$ $w_i \in \Sigma$
Output: true if accepted, false if rejected

```
state ← q0
for i in 1 ... n
    state ← δ(state, w_i)
return state ∈ A
```

Implementing DFAs

Need to implement the transition function somehow

Idea (Bad): nested ifs

running time of δ: $O(|Q| \cdot |\Sigma|)$

```
if state = Q1
    if input = Σ1
    if input = Σ2
    ⋮
if state = Q2
    if input = Σ1
    ⋮
```

Idea (good):

$O(1)$

input \ state	A	B	C	X	Error
a	B	X	C		Error
1					Error
2			Error		Error
3					Error

Explicit Error

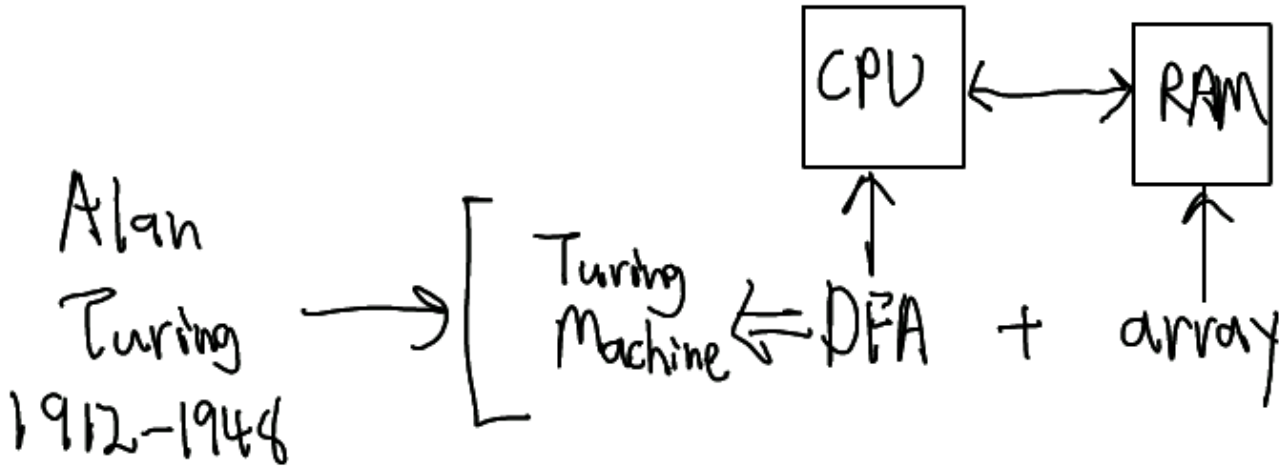
look-up table with 2 indexes; returns next state

Where are DFAs used?

① lexer

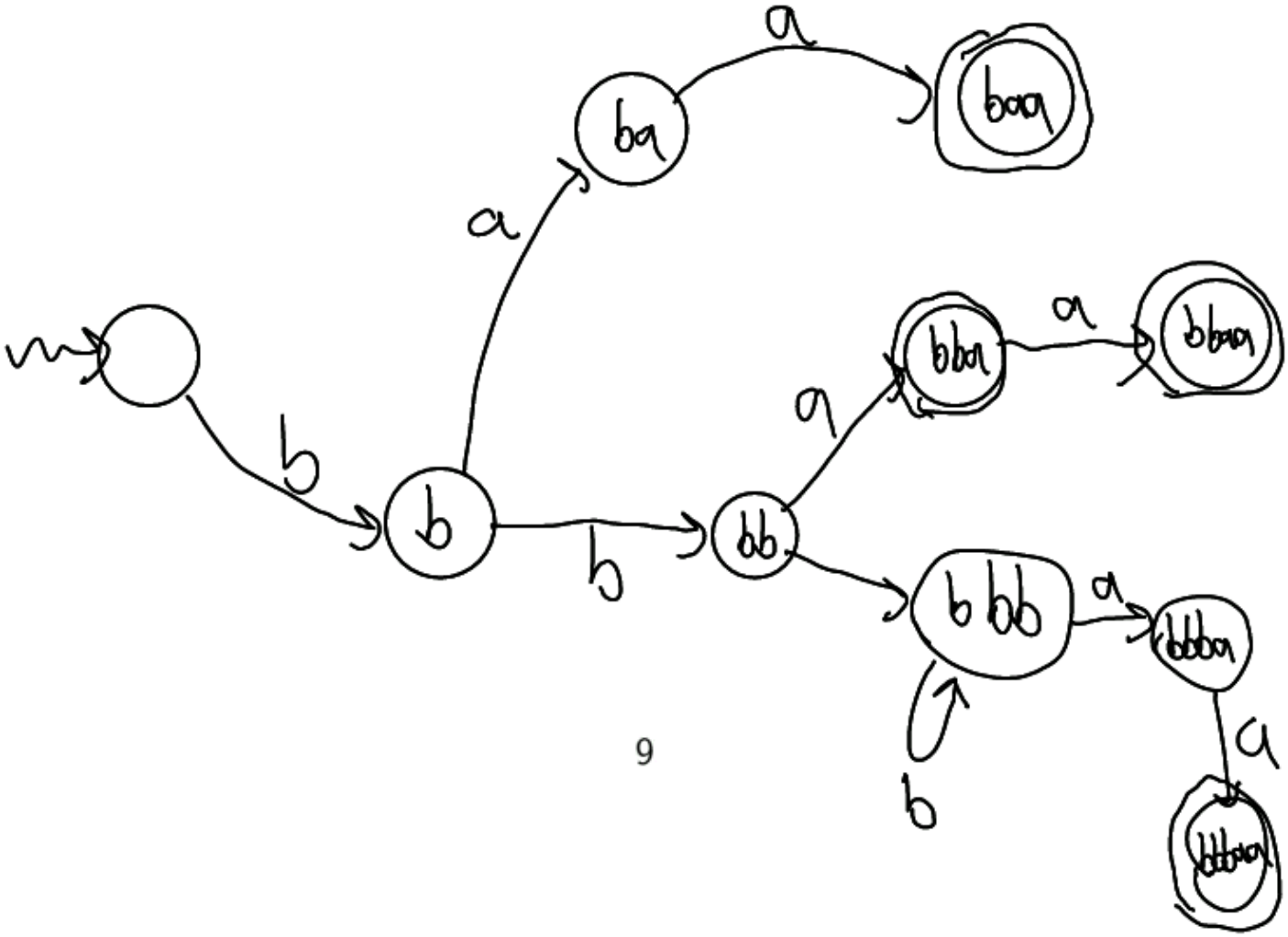
② CPU = DFA

is a



NFAs

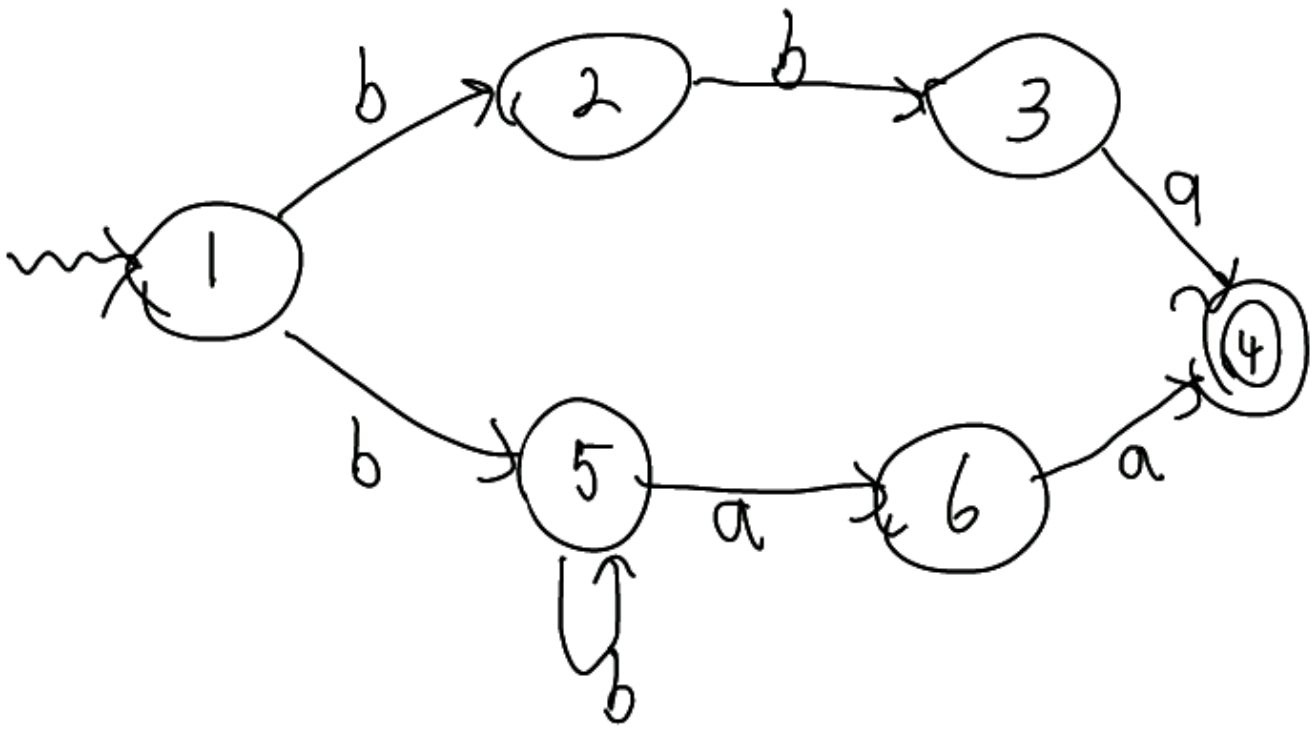
- $L = \{bba, baa, bbaa, bbbaa, bbbbaa, ...\}$ which is either 2 b's followed by an a, or 1 or more b's followed by 2 a's
- try to derive this using a DFA



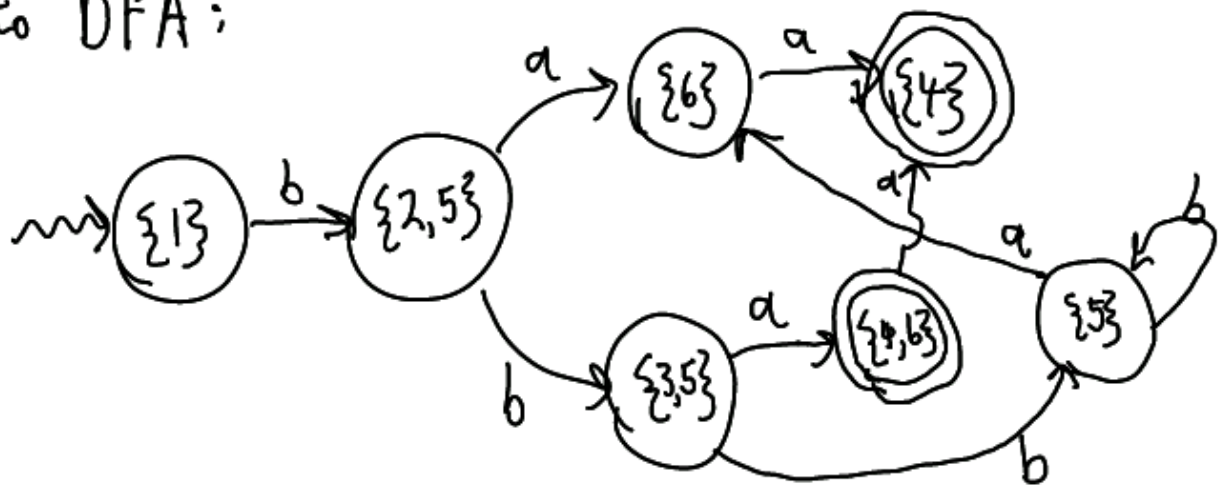
NFAs

- $L = \{bba, baa, bbaa, bbbaa, bbbbaa, ...\}$ which is either 2 b's followed by an a, or 1 or more b's followed by 2 a's
- try to derive using a nice NFA

non-deterministic



change to DFA:



NFA definition

2^Q = power set of Q

Same as a DFA with the following change:

$$T : Q \times \Sigma \to 2^Q$$

That is, we can be in a set of states, and thus T is a *relation* instead of a function.

ex: $Q = \{A, B, C\}$

$2^Q = \{\{\}, \{A, B, C\}, \{A\}, \{B\}, \{C\}, \{A, B\}, \{A, C\}, \{B, C\}\}$

$|2^Q| = 2^{|Q|}$

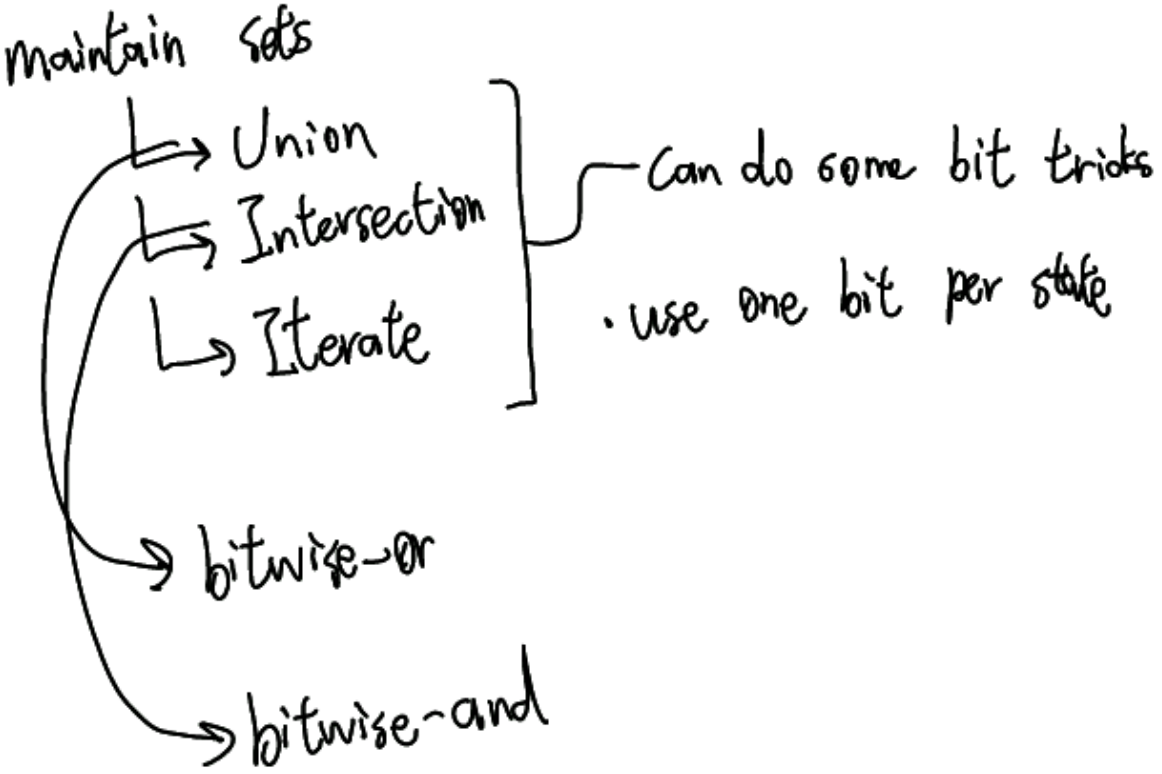
NFA interpreter

Input: $w_1 w_2 \ldots w_n$

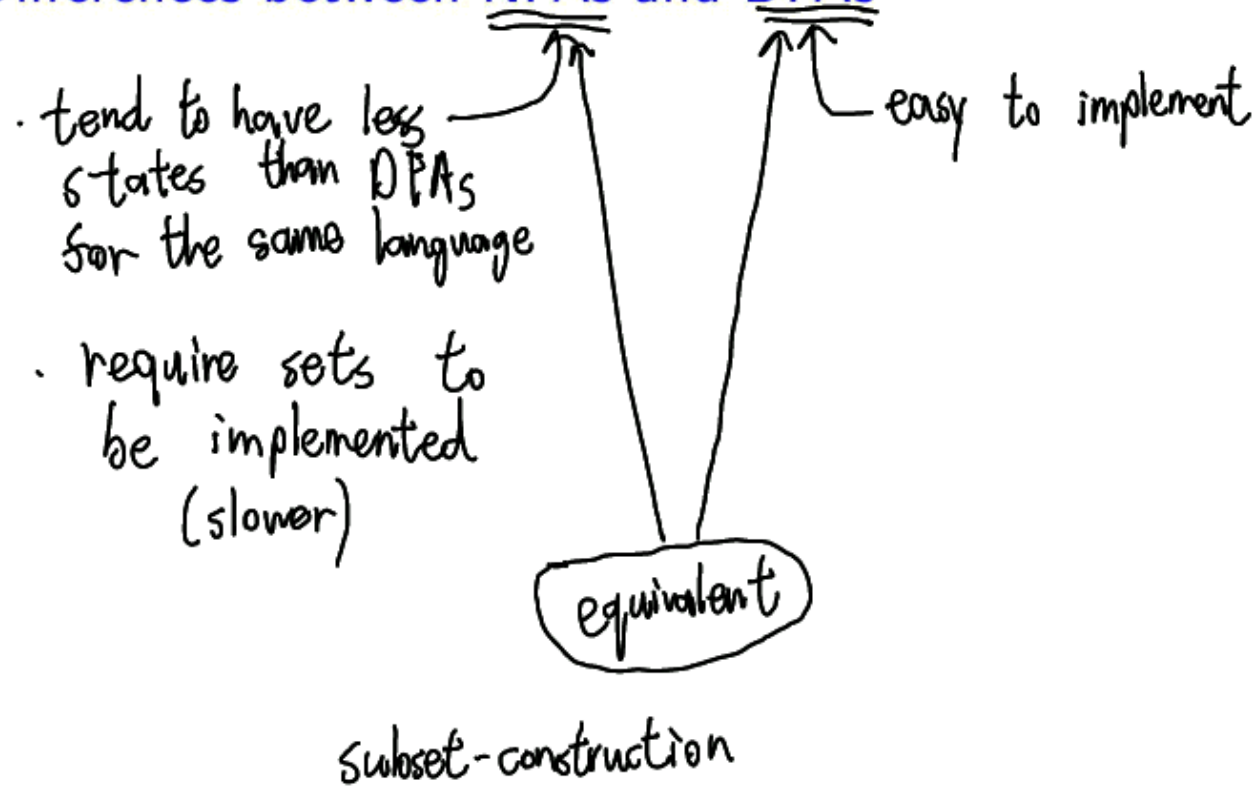
```
states ← {q0}
for i in 1...n
    s' = {}
    for each s in states:
        s' = T(s, w_i) ∪ s'
    states ← s'
return A ∩ states ≠ {}
```

Implementing an NFA interpreter

Messy!



Differences between NFAs and DFAs



A few words about the subset construction

Q: Given my current set of states, where do I go on the current input.

Transducers

Add an output alphabet, and each transition can output a single character from that output alphabet.

Killer app for Finite Automata

Scanner: see `asm.*`