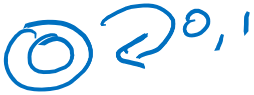
BU CS 332 – Theory of Computation

Lecture 6:

- NFAs -> Regular expressions
- Context-free grammars
- Pumping lemma for CFLs

101 11



Reading:

Sipser Ch 1.3, 2.1, 2.3

HW mean 23.78/30
SD 5.06

HW 3 out, due Tuesday 2/18

Nadya away this week

Mark Bun

February 10, 2020

Regular Expressions – Syntax

A regular expression R is defined recursively using the following rules:

1. ε, $\emptyset$, and a are regular expressions for every $a \in \Sigma$

2. If R_1 and R_2 are regular expressions, then so are
   $(R_1 \cup R_2)$, $(R_1 R_2)$, and (R_1^*)

Examples: (over $\Sigma = \{a, b, c\}$)

ab $\quad$ $(ab^* \cup a^*b)^*$ $\quad$ $\emptyset^*$

Regular Expressions – Semantics

$L(R)$ $=$ the language a regular expression describes

1. $L(\emptyset) = \emptyset$
2. $L(\varepsilon) = \{\varepsilon\}$
3. $L(a) = \{a\}$ for every $a \in \Sigma$
4. $L((R_1 \cup R_2)) = L(R_1) \cup L(R_2)$
5. $L((R_1 R_2)) = L(R_1) \circ L(R_2)$
6. $L\left((R_1^*)\right) = (L(R_1))^*$

Example: $L(a^* b^*) = \{a^m b^n | m, n \geq 0\}$

Regular Expressions Describe Regular Languages

Theorem: A language A is regular if and only if it is described by a regular expression

Theorem 1: Every regular expression has an equivalent NFA

Last time

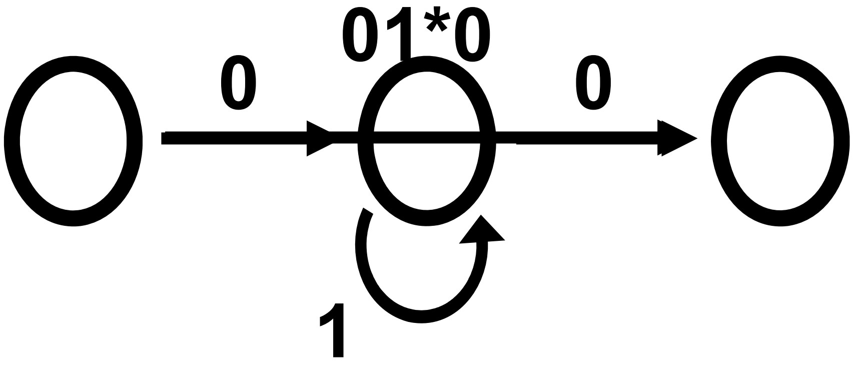
Theorem 2: Every NFA has an equivalent regular expression

Today

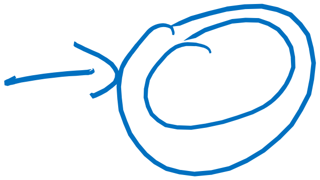
NFA -> Regular expression

Theorem 2: Every NFA has an equivalent regex

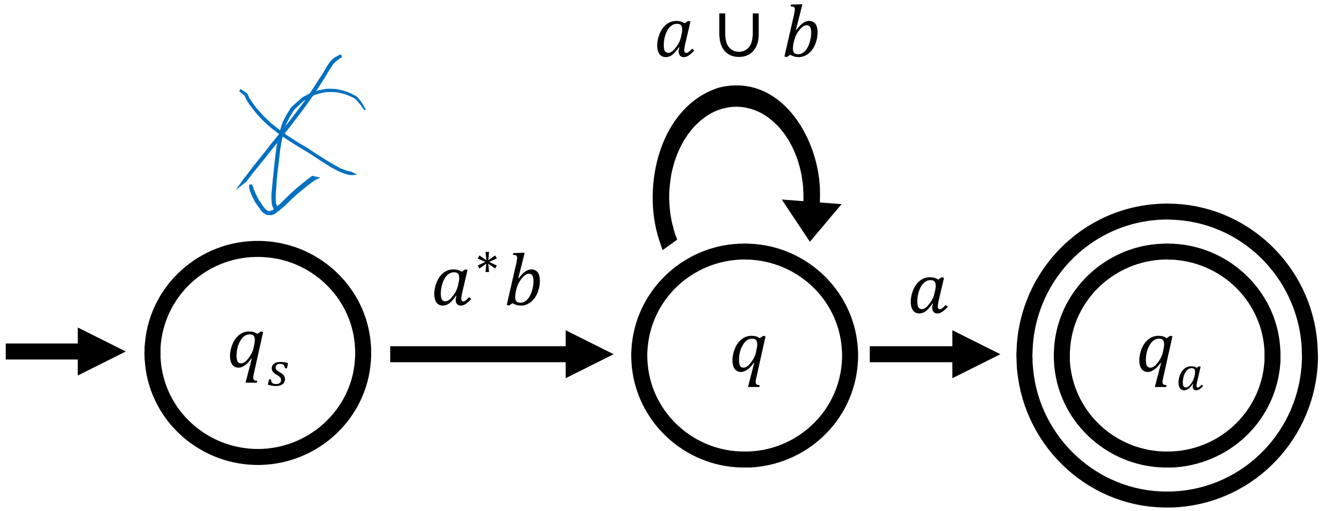
Proof idea: Simplify NFA by “ripping out” states one at a time and replacing with regexes



Generalized NFAs



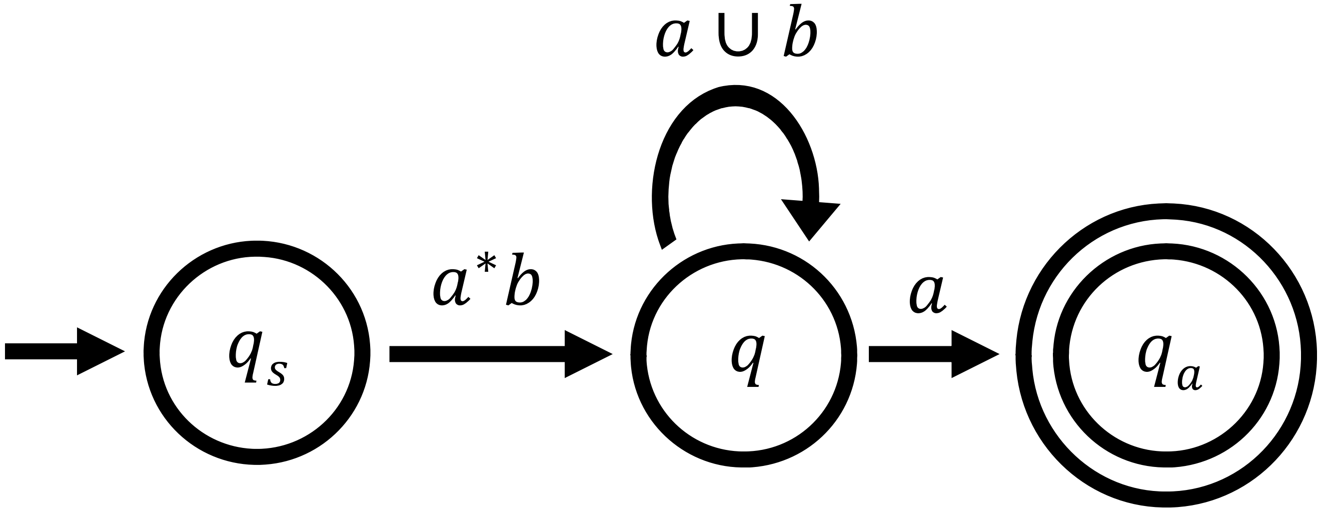
- **Every transition is labeled by a regex**
- One start state with only outgoing transitions
- Only one accept state with only incoming transitions
- Start state and accept state are distinct



Generalized NFA Example

$\phi \neq \epsilon$

$L(\phi^*) = \{\epsilon\}$

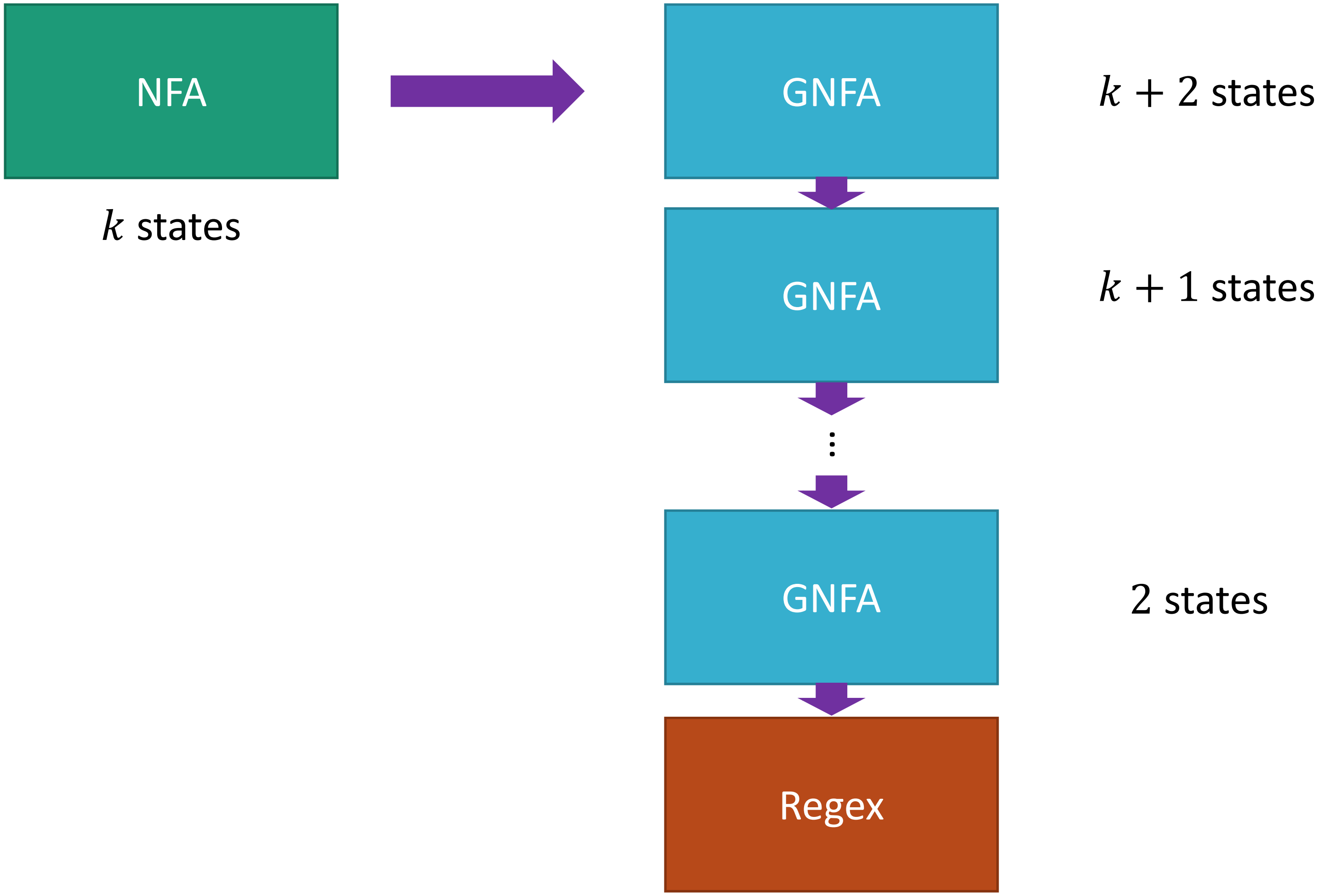


$$R(q_s, q) = a^* b$$

$$R(q_a, q) = \emptyset$$

$$R(q, \underset{q_s}{\cancel{qs}}) = \emptyset$$

NFA -> Regular expression



NFA

k states

GNFA

$k + 2$ states

GNFA

$k + 1$ states

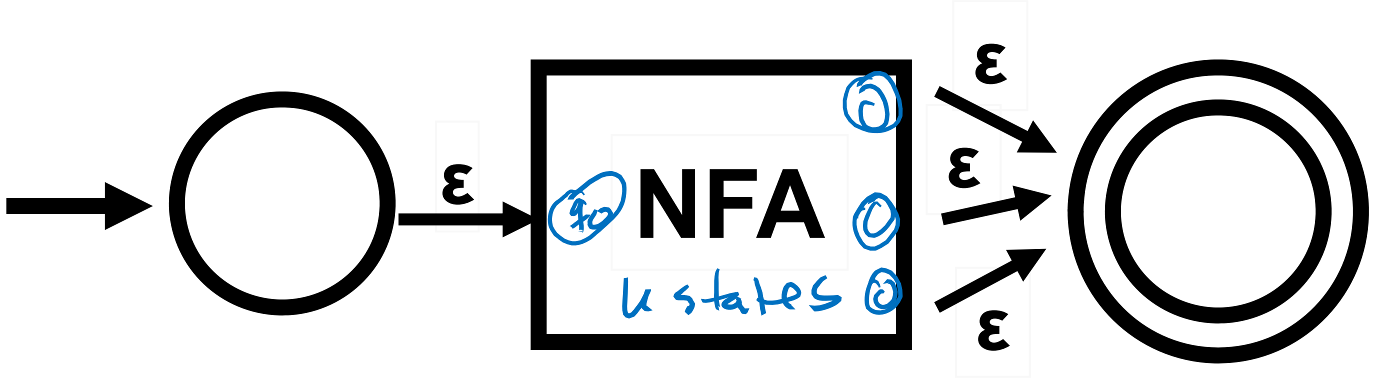
⋮

GNFA

2 states

Regex

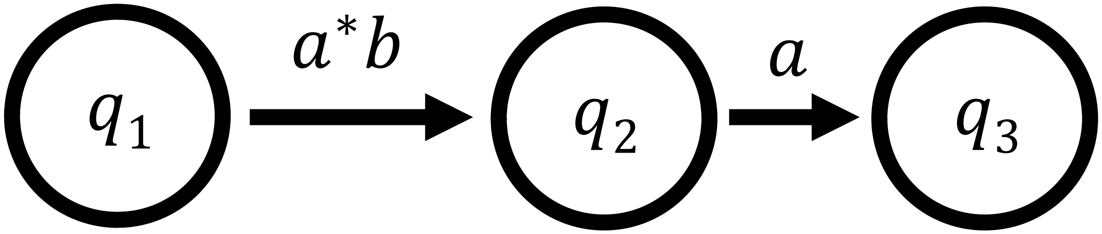
NFA -> GNFA

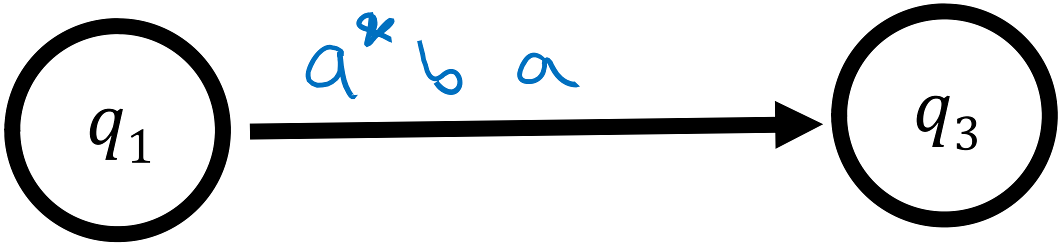


- Add a new start state with no incoming arrows.
- Make a unique accept state with no outgoing arrows.

GNFA -> Regular expression

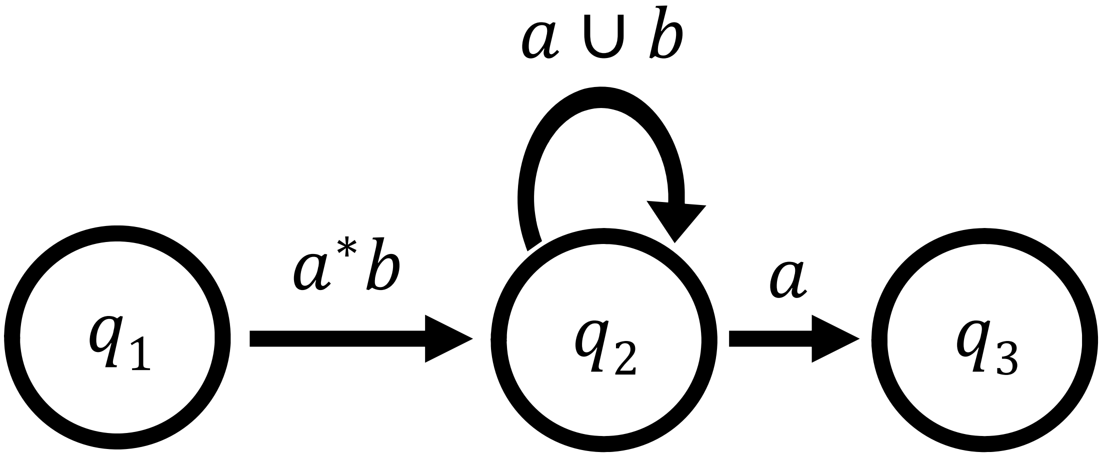
Idea: While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state

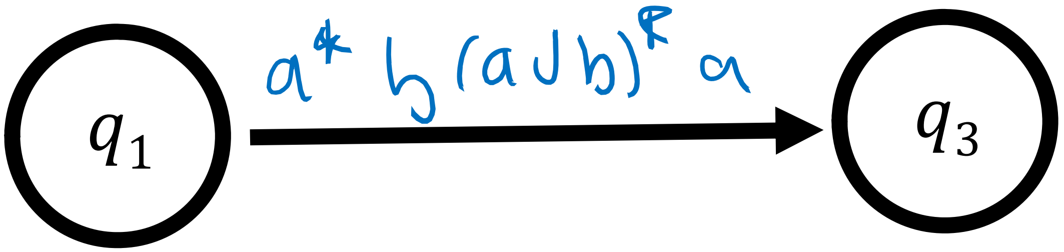




GNFA -> Regular expression

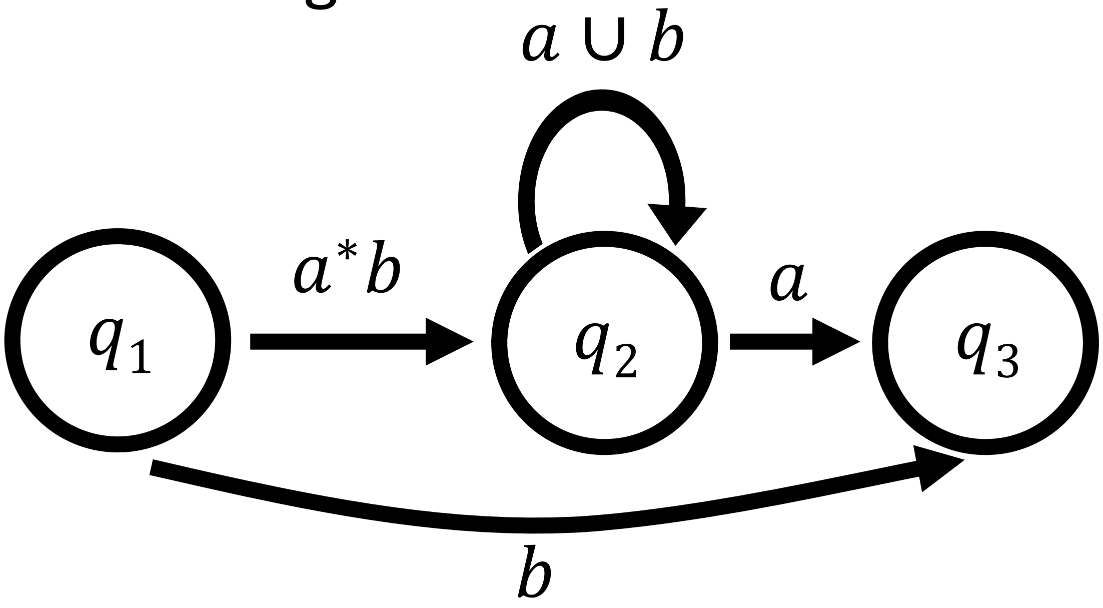
Idea: While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state

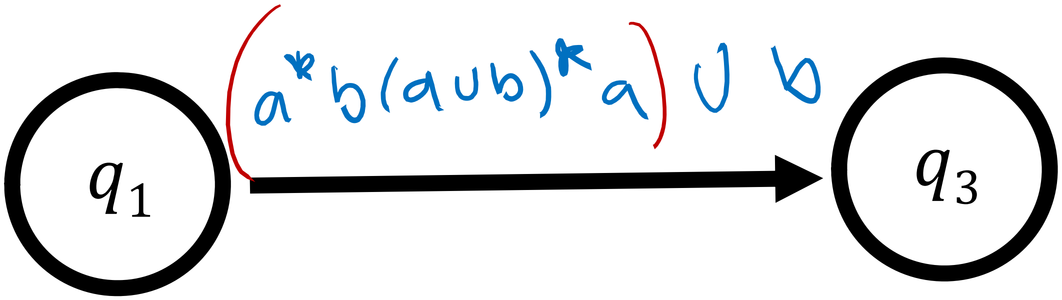




GNFA -> Regular expression

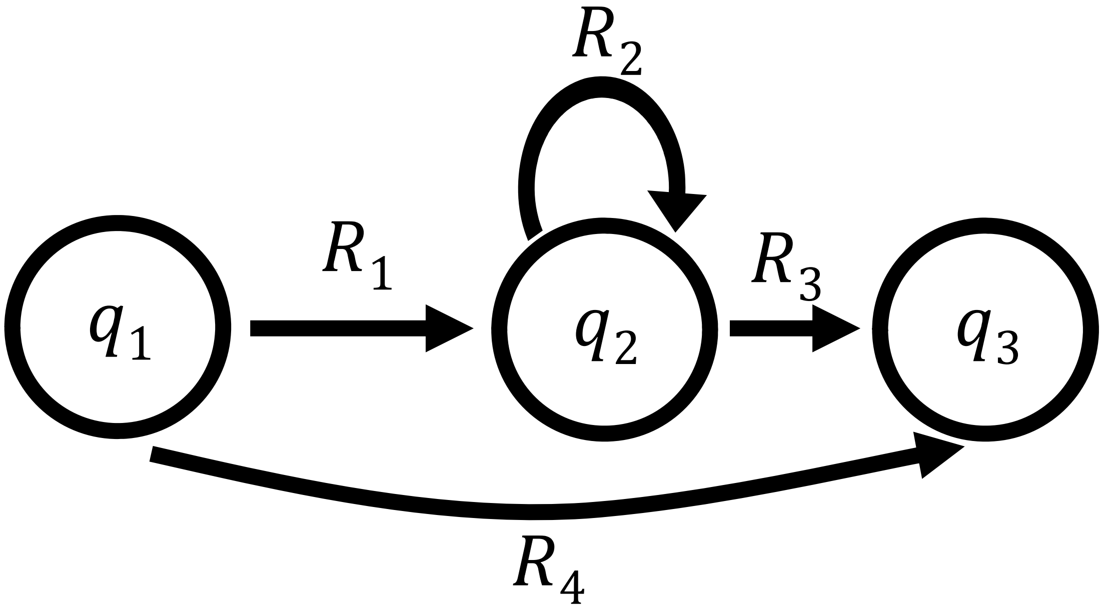
Idea: While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state

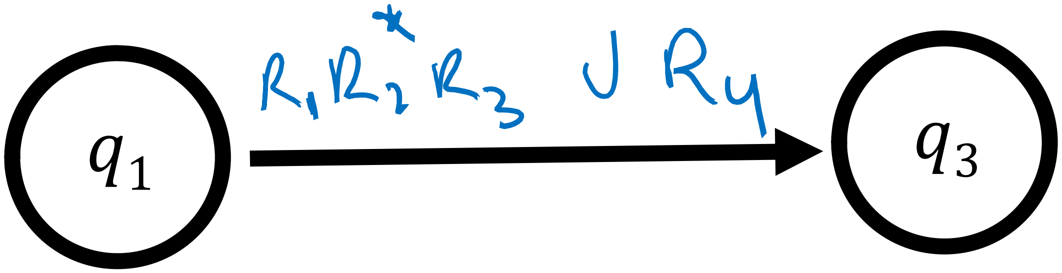




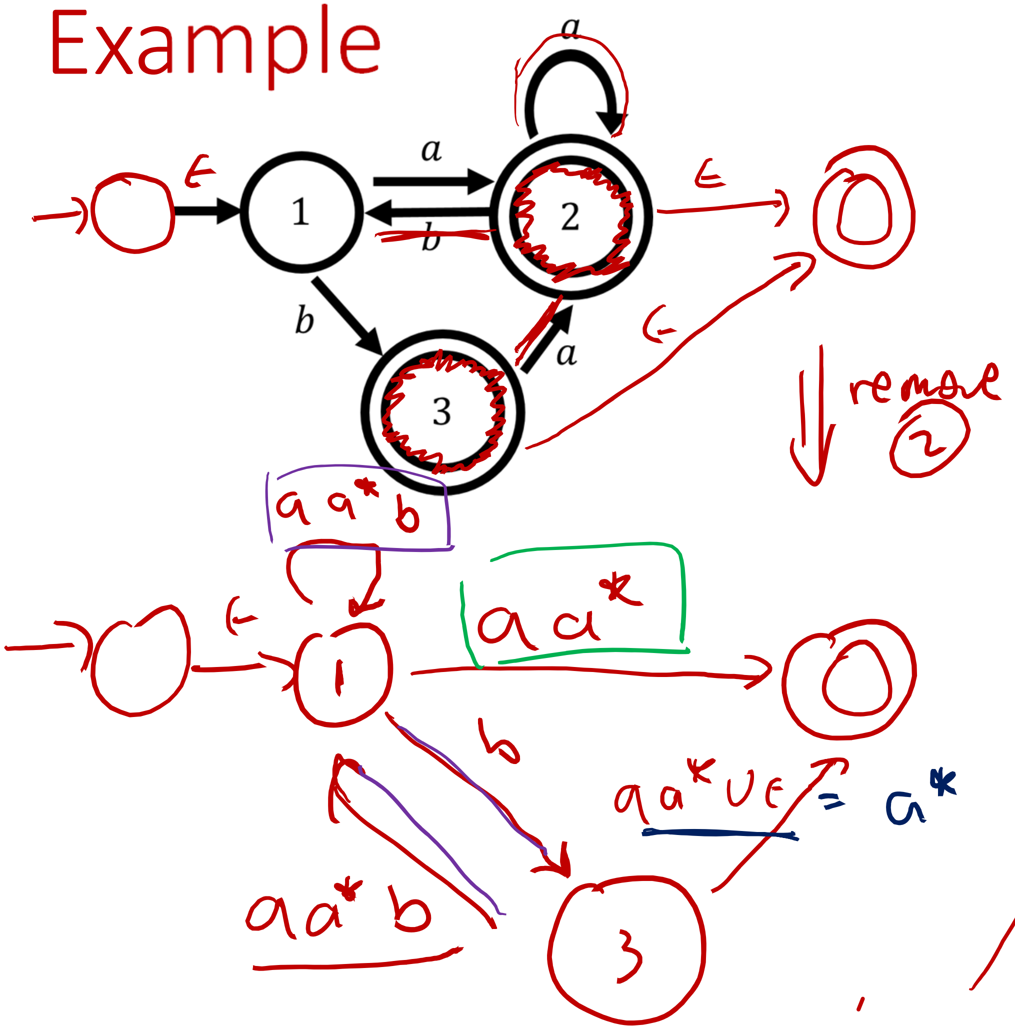
GNFA -> Regular expression

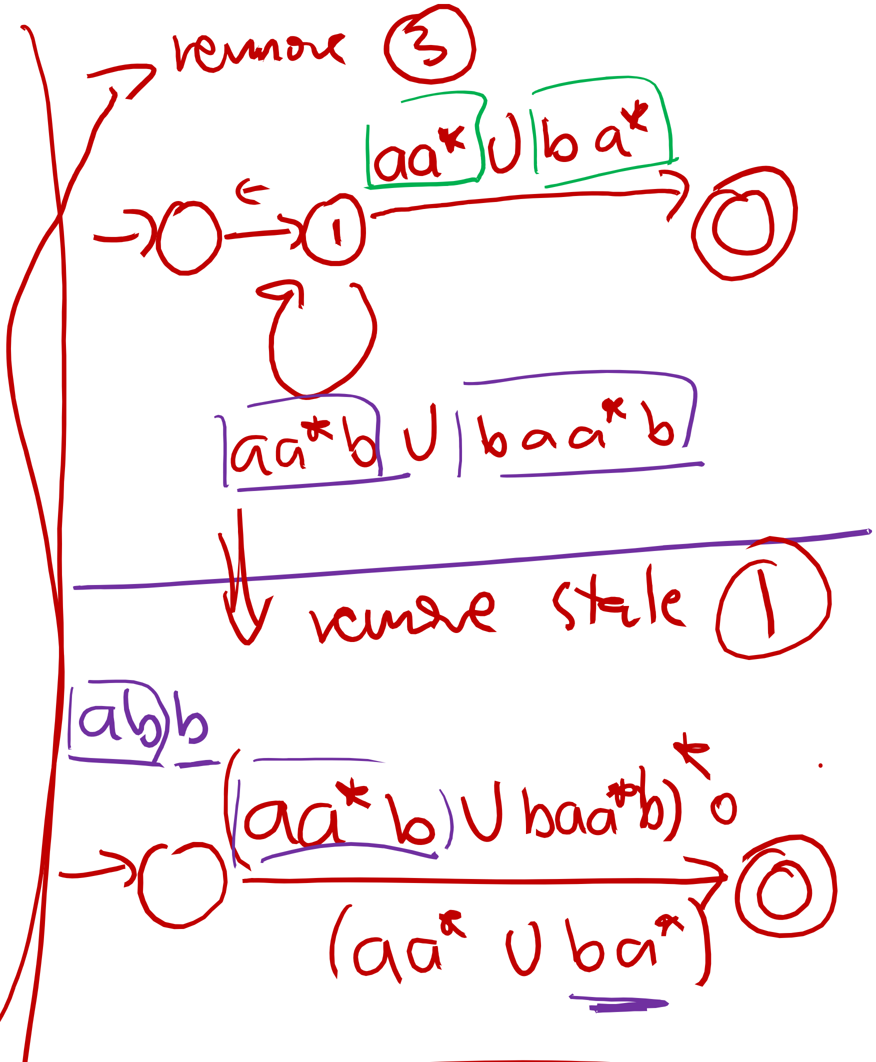
Idea: While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state





Example





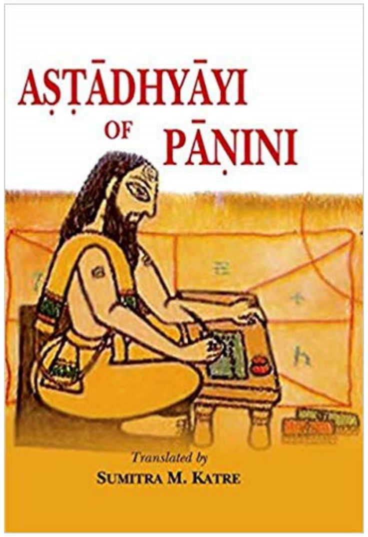
$(aa^*b \cup baa^*b)^*(aa^* \cup ba^*)$

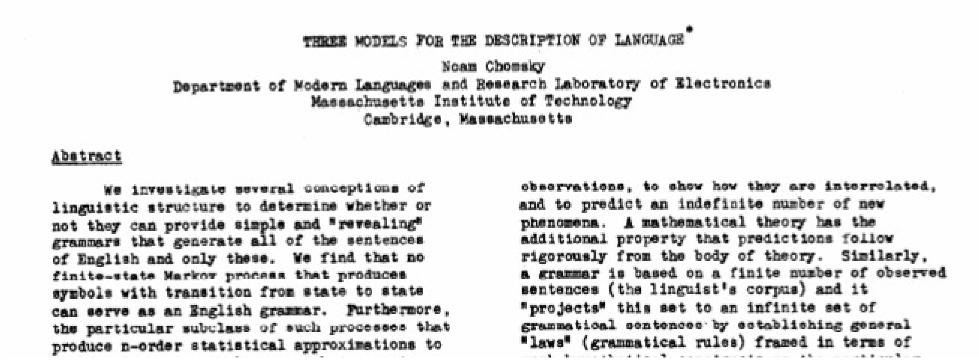
Context-Free Grammars

Some History

An abstract model for two distinct problems

Rules for parsing natural languages





THREE MODELS FOR THE DESCRIPTION OF LANGUAGE*

Noam Chomsky
Department of Modern Languages and Research Laboratory of Electronics
Massachusetts Institute of Technology
Cambridge, Massachusetts

Abstract

We investigate several conceptions of linguistic structure to determine whether or not they can provide simple and "revealing" grammars that generate all of the sentences of English and only these. We find that no finite-state Markov process that produces symbols with transition from state to state can serve as an English grammar. Furthermore, the particular subclass of such processes that produce n-order statistical approximations to

observations, to show how they are interrelated, and to predict an indefinite number of new phenomena. A mathematical theory has the additional property that predictions follow rigorously from the body of theory. Similarly, a grammar is based on a finite number of observed sentences (the linguist's corpus) and it "projects" this set to an infinite set of grammatical sentences by establishing general "laws" (grammatical rules) framed in terms of

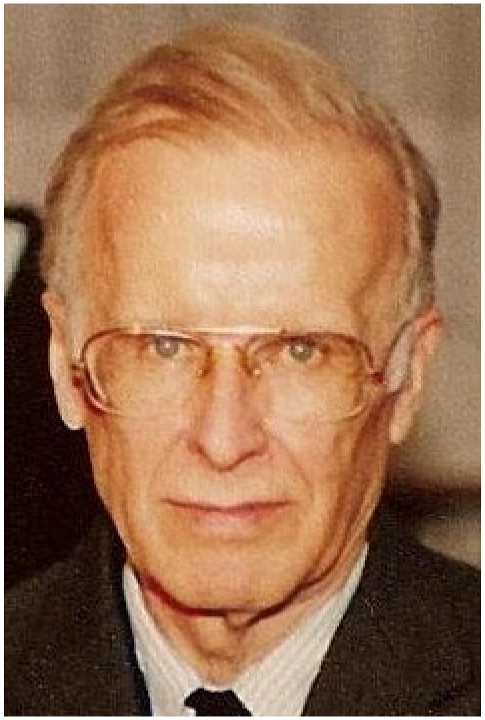
Some History

An abstract model for two distinct problems

Specification of syntax and compilation for programming languages

1977 ACM Turing Award citation
(John Backus)

For profound, influential, and lasting contributions to the design of practical high-level programming systems, notably through his work on FORTRAN, and for seminal publication of formal procedures for the specification of programming languages.



Context-Free Grammar (Informal)

Example Grammar G

Variables: A, B

$A \rightarrow 0A1$
$A \rightarrow B$
$B \rightarrow \#$

→ Rules

Terminals: 0, 1, #

Derivation

$A \Rightarrow 0A1 \Rightarrow 0\underbrace{0A1}_{A}1 \Rightarrow 00B11 \Rightarrow 00\#11$

$L(G) = \{0^n \# 1^n \mid n \geq 0\}$

Context-Free Grammar (Informal)

$E \Rightarrow E + T$
$\Rightarrow E + T + T$
$\Rightarrow E + T + T + T + T$
$\vdots$

Example Grammar G

$$
\begin{aligned}
E &\to E + T \\
E &\to T \\
T &\to T \times F \\
T &\to F \\
F &\to (E) \\
F &\to a \\
F &\to b
\end{aligned}
$$

Derivation

$E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T \Rightarrow a + F \Rightarrow a + (E)$
$\Rightarrow a + (T) \Rightarrow a + (T \times F) \Rightarrow a + (F \times F) \Rightarrow a + (a \times F)$
$\Rightarrow \underline{a + (a \times b)}$

$L(G) =$ well-formed arithmetic expressions
using $a, b, +, \times, (,)$

Socially Awkward Professor Grammar

<PHRASE> → <FILLER><PHRASE>

<PHRASE> → <START><END>

<FILLER> → LIKE

<FILLER> → UMM

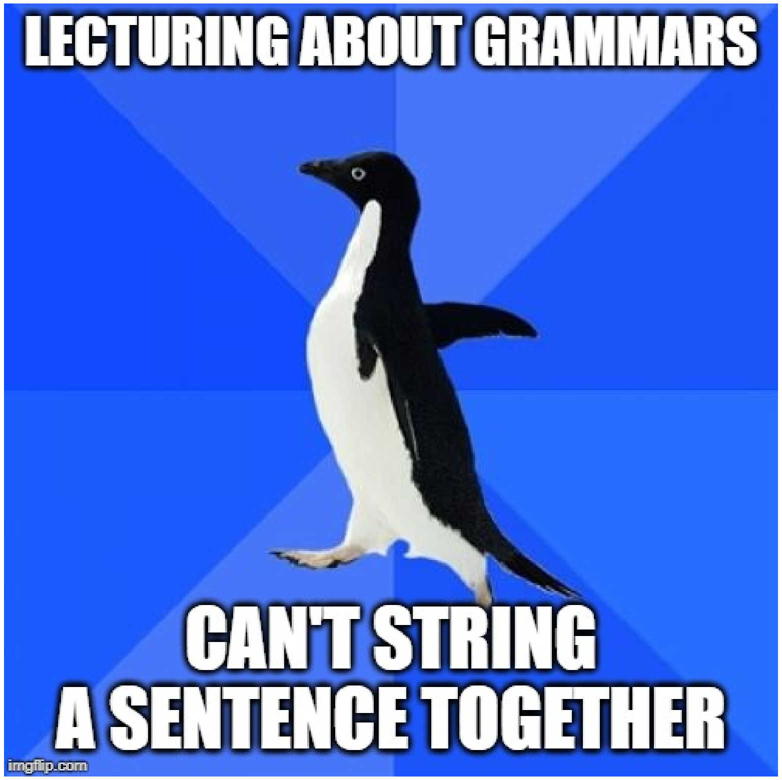
<START> → YOU KNOW

<START> → ε

<END> → WHOOPS

<END> → SORRY

<END> → $#@!



Socially Awkward Professor Grammar

Backus-Naur form

<PHRASE> → <FILLER><PHRASE> | <START><END>

<FILLER> → LIKE | UMM

<START> → YOU KNOW | ε

<END> → WHOOPS | SORRY | $#@!

Context-Free Grammar (Formal)

A CFG is a 4-tuple $G = (V, \Sigma, R, S)$

- V is a finite set of variables
- Σ is a finite set of terminal symbols (disjoint from V)
- R is a finite set of production rules of the form $A \rightarrow w$, where $A \in V$ and $\underline{w \in (V \cup \Sigma)^*}$
- $S \in V$ is the start symbol

Example: $G = (\{S\}, \Sigma, R, S)$ where $R = \{S \rightarrow aSb, S \rightarrow \varepsilon\}$

$S \rightarrow aSb \mid \varepsilon$

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

Context-Free Grammar (Formal)

A CFG is a 4-tuple $G = (V, \Sigma, R, S)$

V = variables Σ = terminals R = rules S = start

- We say $uAv \Rightarrow uwv$ ("uAv yields uwv") if $A \rightarrow w$ is a rule of the grammar
- We say $u \overset{*}{\Rightarrow} v$ ("u derives v") if $u = v$ or there exists a sequence such that $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow v$
- Language of the grammar: $L(G) = \{w \in \Sigma^* | S \overset{*}{\Rightarrow} w\}$

Example: $G = (\{S\}, \Sigma, R, S)$ where $R = \{S \rightarrow aSb, S \rightarrow \varepsilon\}$

$$L(G) = \{a^n b^n | n \geq 0\}$$

CFG Examples

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

Give context-free grammars for the following languages

1. The empty language

$S \to \epsilon$

$V=\{S\}$ $R=\emptyset$

$\Sigma=\{0,1\}$ S (start variable)

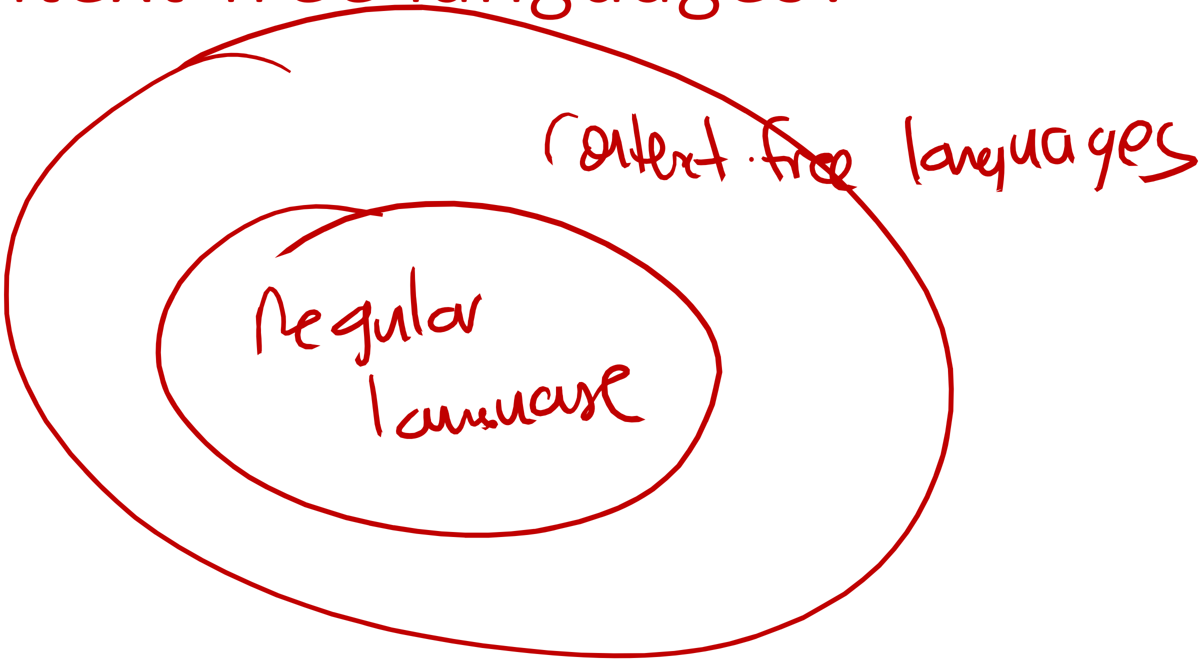
2. Strings of properly nested parentheses

$S \to (S) \mid SS \mid \epsilon$

3. Strings with equal # of a's and b's

Pumping Lemma II: Pump Harder

Non context-free languages?



- Could it be the case that every language is context-free?

Pumping Lemma for regular languages

Let L be a regular language.

Then there exists a “pumping length” p such that

For every $w \in L$ where $|w| \geq p$,
w can be split into three parts $w = xyz$ where:

1. $|y| > 0$
2. $|xy| \leq p$
3. $xy^iz \in L$ for all $i \geq 0$

Pumping Lemma for context-free languages

Let L be a context-free language.

Then there exists a “pumping length” p such that

For every $w \in L$ where $|w| \geq p$,

w can be split into five parts $w = u\underline{v}x\underline{y}z$ where:

1. $|vy| > 0$ Either $v \neq \epsilon$, or $y \neq \epsilon$
2. $|vxy| \leq p$
3. $uv^ixy^iz \in L$ for all $i \geq 0$

Example:

$L = \{w \in \{0,1\}^* \mid w = w^R\}$

$w = 0$

Pumping Lemma for context-free languages

Let L be a context-free language.

Then there exists a “pumping length” p such that

For every $w \in L$ where $|w| \geq p$,

w can be split into five parts $w = uvxyz$ where:

1. $|vy| > 0$
2. $|vxy| \leq p$
3. $uv^ixy^iz \in L$ for all $i \geq 0$

Example: $p = 3$

$L = \{w \in \{0,1\}^* | w = w^R\}$

$w = 010$

$u = \epsilon$
$v = 0$
$x = 1$
$y = 0$
$z = \epsilon$

$uv^ixy^iz =$

$0^i10^i \in L$

Pumping Lemma as a game

1. YOU pick the language L to be proved non context-free.
2. ADVERSARY picks a possible pumping length p.
3. YOU pick w of length at least p.
4. ADVERSARY divides w into u, v, x, y, z, obeying rules of the Pumping Lemma: $|vy| > 0$ and $|vxy| \leq p$.
5. YOU win by finding $i \geq 0$, for which uv^ixy^iz is not in L.

If *regardless* of how the ADVERSARY plays this game, you can always win, then L is non context-free

Pumping Lemma example

Claim: $L = \{a^n b^n c^n | n \geq 0\}$ is not ~~regular~~ context-free

Proof: Assume L is ~~regular~~ context-free with pumping length p

1. Find $w \in L$ with $|w| \geq p$ $w = a^p b^p c^p$
2. Show that w cannot be pumped

If $w = uvxyz$ with $|vy| > 0, |vxy| \leq p$, then...

1) v and y only consist of one kind of character

$uv^2xy^2z \notin L$ (# of characters is wrong)

2) Either v or y has two kinds of characters

$uv^2xy^2z \notin L$ (order is wrong)