. What are the first several strings generated by a regular expression

1) ab(ab)*(a+b)
aba, abb, ababa, ababb, abababa

2) (a*)(b*)
λ, a, b, aa, ab, bb, aaa, aab, abb, bbb

3) (a*)+(b*)
λ, a, b, aa, bb, aaa, bbb

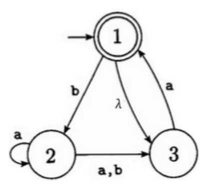
4) (ab)*
λ, ab, abab, ababab, abababab

5) a(a+b)*
a, aa, ab, aaa, aba, abb, aaaa, abbb

6) (a+b)*
λ, a, b, aa, ab, bb, aaa, bbb

7) ((a+b)(a+b))*
λ, aa, ab, ba, bb, aaaa, abab, baba, bbbb

Convert NFA to DFA



Step 1 - NFA State Transition Table
From the local state's perspective what state does the value go to / what does it send out

accepting state
➔ Starting State
∅ empty Set

	a	b
→1	{∅}	{2}
2	{2,3}	{3}
3	{1}	{∅}

Step 2 - Convert to DFA Transition Table

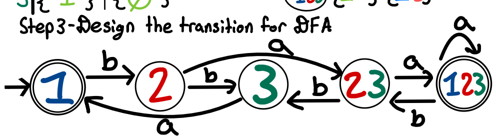
*Union of states from NFA Transition Table

	a	b
→1	{∅}	{2}
2	{2,3}	{3}
3	{1}	{∅}

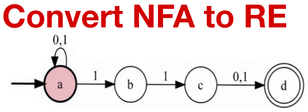
NFA Table to DFA Table ➔

	a	b
→1	{∅}	{2}
2	{23}	{3}
3	{1}	{∅}
23	{123}	{3}
123	{123}	{23}

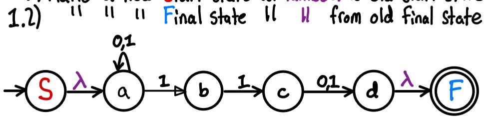
Step3 - Design the transition for DFA



Convert NFA to RE



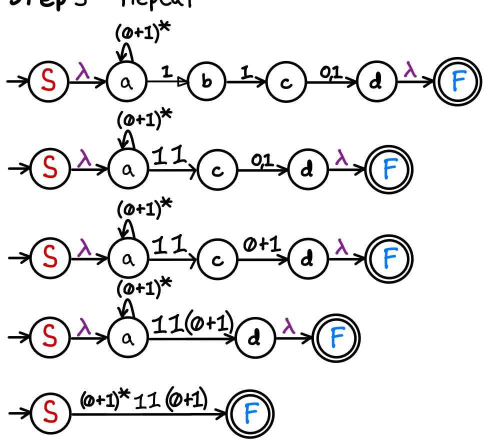
Step 1 - "Fix the NFA"
1.1) Make a new Start State w/ lambda to old start state
1.2) " " " Final state " " from old final state



Step 2 - Pick states w/ few transitions

- Loops do not count for they become stars
- Parallel edges you'd combine them with a plus +

Step 3 - Repeat

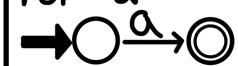


Therefore (0+1)*11(0+1)

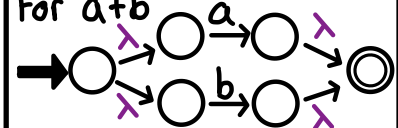
Convert RE to NFA

For lambda (λ)

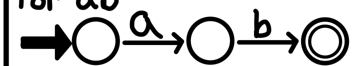
For 'a'



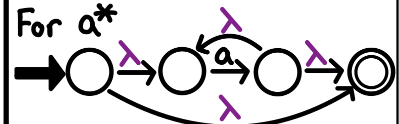
For a+b



For ab



For a*



Convert CFG to PDA

S → A Ɛ
A → a A | a
Ɛ → a Ɛ b | λ

→

S → A Ɛ
A → a A
A → a
Ɛ → a Ɛ b
Ɛ → λ

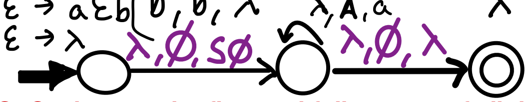
one triple (read, pop, push) for each production
one triple for each terminal character on self-loop

S → A Ɛ
A → a A
A → a
Ɛ → a Ɛ b
Ɛ → λ

See-a-terminal remove and add nothing
a, a, λ
b, b, λ

Left most derv A →
λ, S, AƐ
λ, A, aA
λ, A, a

Right most derv Ɛ →
λ, Ɛ, aƐb
λ, Ɛ, λ,



Given an ambiguous CFG show that it is ambiguous via parse trees and/or leftmost derivations.

Ambiguous - because it goes to the same string set production

S → BC | λ
B → bbB | C | λ
C → cC | c

Shows ambiguity - Not able to parse
S → BC → CC → cC → cc
S → BC → C → cC → cc

Proven by the left-most diveration from left-to-right first non-terminal character

Given a lexical specification and a string, what tokens are generated

Given a non-regular language pick a string that would be good for a pumping lemma proof

$L = \{a^i b^j c^k \mid i+k=j\}$ quality

easy argue
- $a^p b^p$
- $b^p c^p$
- $a^p b^{2p} c^p$

- $a^p b^p$
- $b^p c^p$
- $a^{p/2} b^p c^{p/2}$
- $a^p b^{2p} c^p$

Given a CFG what are the first and follow sets of all the non-terminals

Production	Indicates that
T→R	1st(R) ⊆ 1st(T)
T→aTc	a ∈ 1st(T)
R→bR	b ∈ 1st(R)
R→λ	nothing about any 1st set since λ len = ∅

Pattern	Meaning
X → … Yz …	z ∈ follow(Y)
X → … YZ …	1st(Z) ⊆ f(Y)
X → … Y	f(X) ⊆ f(Y)

1. A → aA
2. A → λ
3. B → bB
4. B → λ
5. C → cC
6. C → λ
7. S → ABCd

Step1: First constraints - Left most non-Terminal

S→ABCd: first(A) ⊆ first(S), first(B) ⊆ first(S), first(C) ⊆ first(S), d ∈ first(S)
A→aA : a ∈ first(A)
B→bB : b ∈ first(B)
C→cC : c ∈ first(C)
A → λ: —
B → λ: —
C → λ: —

Step2: Build out sets of first w/ elements of

First(S): abcd
First(A): a
First(B): b
First(C): c

Step3: Follow our left devr. Keep in mind of nullable

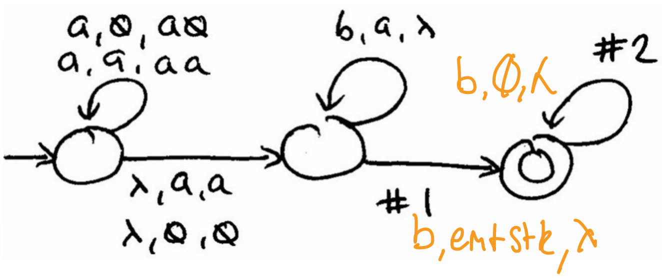
S→ABCd : first(B) ⊆ follow(A), first(C) ⊆ follow(B), first(C) ⊆ follow(A), d ∈ follow(C), d ∈ follow(C), d ∈ follow(A), d ∈ follow(B)

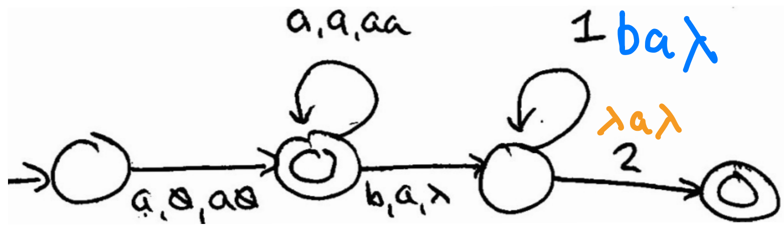
Step4: Build follow set

follow(A) : b, c, d
follow(B) : d, c
follow(C) : d

Step 5 - Build predictor table

	first() RHS	follow	Union
S → ABCd	abcd		abcd
A → aA	a		a
B → bB	b		b
C → cC	c		c
A → λ	—	b, c, d	b, c, d
B → λ	—	d, c	d, c
C → λ	—	d,	d,





S'→S$
S→BA
A→+BA|-BA|λ
B→DC
C→*DC|/DC|λ
D→a|(S)

1(S) a,(f(S):),$

1(A) +,-   f(A):),$

1(B) a,(f(B) +,-,),$

1(C) *,/   f(C) +,-,),$

1(D) a,(f(D) +,-,*,/,),$