1 - Formal Languages

Alphabet	$\Sigma = \{ symbols \}$
Word	Sequence of symbols from Σ
Language	$\{words\}$ • $\Sigma * \quad = \{ all\ words,\ including\ empty\}$ • L = ∅ or {} (empty language) • L = {ε} (language w' empty string)

Finite Languages

recognition algorithm for finite-languages: **state diagrams**

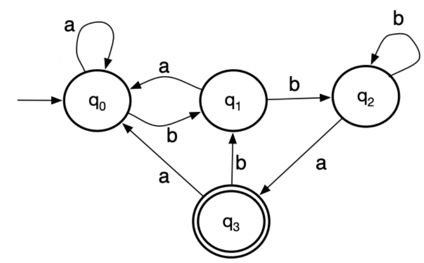
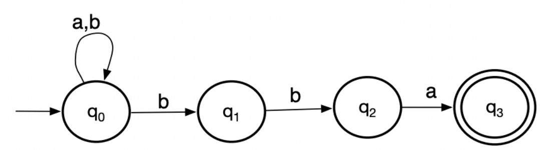
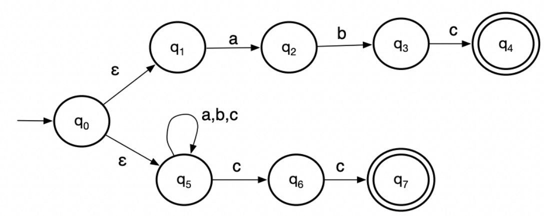
Regular Language

Regular Language
L is **Regular** if it's: 1. {} or {ε} 2. $\{a\}$ for some $a\epsilon\Sigma$ 3. Built using: union(|), concatenation(x) or Kleene-Star(*) of 2 RL's

Representing Regular Languages

They can all be converted between each other:

DFA	**Deterministic:** no choice; for each state, only 1 transition on a given symbol	Ends with bba

		
NFA	**Non-deterministic:** allows multiple transitions from a state on the same symbol	Ends with bba: 
ε-NFA	permit ε-**transitions**: state transitions without consuming a symbol	$L = \{abc\} \cup \{w : w \text{ ends with } cc\}$ 

Σ	Alphabet
Q	set of states
q0	start state (∈Q)
A	Set of Accept states(∈Q)
δ	Depends on which one

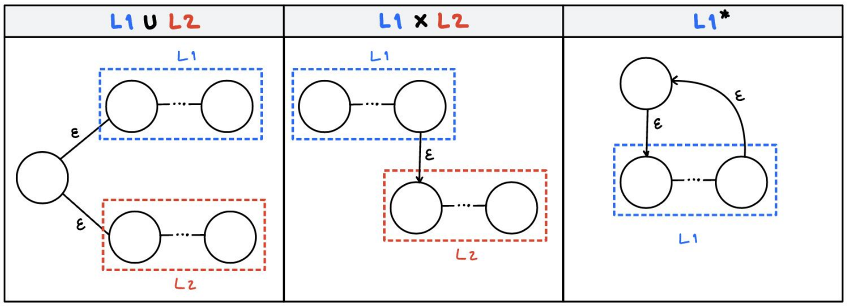
*set for NFA/ε-NFA

Main Difference is the transition function:

DFA	**NFA**	**ε-NFA**
state + symbol => state	{states} + symbol => {states}	({states} + symbol)|ε => {states}

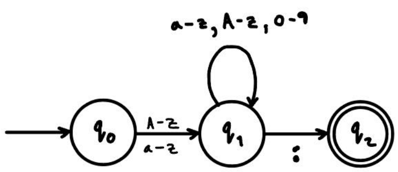
Epsilon closure, E(S)
S = {states} E(S) = {states: reachable from S in 0+ ε-transitions

TYPE	**Accepts a string**	**Algorithm (input: w = $a_1a_2a_3$...)**
DFA	accept state	s = q0 **for** (every a_i) Transition not defined: **Reject** Otherwise: **transition** **End of input:** **if** (in accept state) **Accept** **else** **Reject**
NDA	1 state is accepting	S = {q0} **for** (every a_i) Transition not defined: **Reject** Otherwise: **transition** **End of input:** **if** (a state is accepting) **Accept** **else** **Reject**
ε-NFA	1 state is accepting	s = E({q0}) **for** (every a_i) Transition not defined: **Reject** Otherwise: 1- **transition** on symbol 2- **transition** on epsilon **End of input:** **if** (a state is accepting) **Accept** **else** **Reject**



EX: DFA recognition algorithm

example



$\Sigma = \{a\text{-}z, A\text{-}Z, 0\text{-}9, :\}$

$q_0 = q_0$

$Q = \{q_0, q_1, q_2\}$

$A = \{q_2\}$

δ : $\delta(q_0, A\text{-}Z \text{ or } a\text{-}z) = q_1$

$\delta(q_1, a\text{-}z \text{ or } A\text{-}Z \text{ or } 0\text{-}9) = q_1$

$\delta(q_1, :) = q_2$

Input: loop2:

Seen	remaining	S
ε	loop2:	q_0
l	oop2:	q_1
lo	op2:	q_1
loo	p2:	q_1
loop	2:	q_1
loop2	:	q_2
loop2:	ε	q_2

accept state

accept

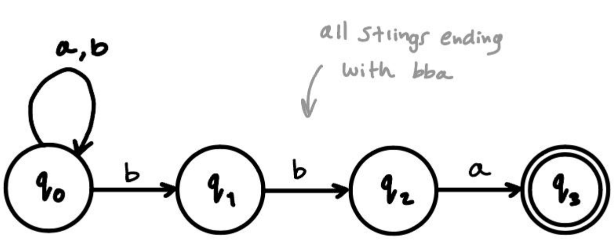
Input: end$

Seen	remaining	S
ε	end$	q_0
e	nd$	q_1
en	d$	q_1
end	$	q_1

$\delta(q_1, \$)$ not defined

EX: NFA recognition Examples

Example



Input: abbba

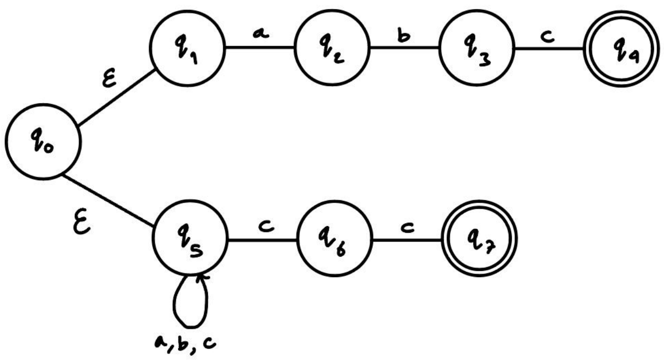
Seen	remaining	S
ε	abbba	$\{q_0\}$
a	bbba	$\{q_0\}$
ab	bba	$\{q_0, q_1\}$
abb	ba	$\{q_0, q_1, q_2\}$
abbb	a	$\{q_0, q_3\}$
abbba	ε	$\{q_0, q_3\}$

accept

EX: ε-NFA Recognition

Example

$L = \{abc\} \cup \{\text{ends with } cc\}$



Input: abcaccc

Seen	remaining	S
ε	abcaccc	$\{q_0, q_1, q_5\}$
a	bcaccc	$\{q_2, q_5\}$
ab	caccc	$\{q_3, q_5\}$
abc	accc	$\{q_4, q_5, q_6\}$
abca	ccc	$\{q_5\}$
abcac	cc	$\{q_5, q_6\}$
abcacc	c	$\{q_5, q_6, q_7\}$
abcaccc	ε	Accept

0+ ε closures

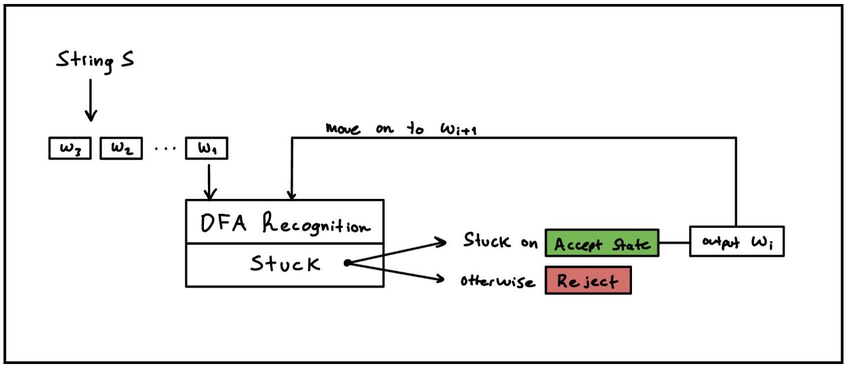
MIPS Assembler

Scanning

Input: code

Output: tokens

Scanning	**Recognition**
Breaking a string into words that are in the language	determines whether a word in the language
Output: 1. sequence of tokens 2. scanning error	**Output:** 1. accept 2. reject
Scanning leverages recognition (runs on a DFA)	

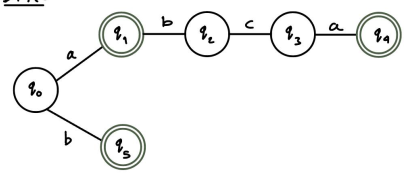


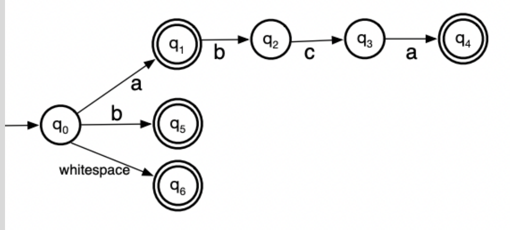
MAX-MUNCH	SIMPLIFIED
GREEDY	
Not guaranteed to find a tokenization if one exists	
Works best when tokens seperated by whitespace	

DFA	MAX	SIMPLIFIED
Consumer Each Letter		
Accept State?	1. **flag** as last accept-state 2. **Be Greedy** (keep going)	**Be Greedy** (keep going)
EOF or Stuck?		
Accept	1. **Backtrack** to last seen accept-state 2. **Un-Consume** input that was greedily consumed 3. **Output** what was consume after the accept-state 4. **Reset** to start state	**reject**
Not	1. **Output** what was consumed 2. **Reset** to start state	3. **Output** what was consumed 4. **Reset** to start state

Σ = {a,b,c} , L = {a, b, abca}

DFA:





Input

ababca

a b abca

Maximal

	Unseen	Consumed	State	
1	ababca	ε	q_0	
2	babca	a	q_1	→ Flag
3	abca	a b (greedy)	q_2	
4	bca	a	error	
5	babca	ε	q_0	
6	abca	b	q_5	→ Flag
7	bca	a	error	
8	abca	ε	q_0	
9	bca	a	q_1	→ Flag
10	ca	b	q_2	
11	a	c	q_3	
	ε	a	q_4	→ Flag

1. Backtrack to q_1
2. Unconsume b
3. Output a
4. State = q_0

1. Backtrack to q_5
2. ~~Unconsume~~
3. Output b
4. State = q_0

end of input : in accept State

output : abca

Simplified

	Unseen	Consumed	State	
1	ababca	ε	q_0	
2	babca	a	q_1	→ output a
3	abca	b	q_2	
4	bca	a	error	→ current State not accepting REJECT

With Whitespace:

	Unseen	Consumed	State	
1	a_b_abca	ε	q_0	
2	_b_abca	a	q_1	
3	_b_abca	_	error	→ in accept State output : a
4	_b_abca	ε	q_0	
5	b_abca	_	q_6	
6	b_abca	b	error	→ in accept State output : _
7	b_abca	ε	q_0	
8	_abca	b	q_5	
9	_abca	_	error	→ in accept State output : b
10	abca	_	q_6	
11	abca	a	error	→ in accept State output : _
12	bca	a	q_1	
13	ca	b	q_2	
14	a	c	q_3	
15	ε	a	q_4	→ end : in accept State output abca

baba

	Unseen	Consumed	State	
1	baba	ε	q_0	
2	aba	b	q_5	→ Flag
3	ba	a	error	→ output: b
4	aba	ε	q_0	
5	ba	a	q_1	→ Flag
6	a	b	q_2	
7	ε	a	error	
8	ba	ε	q_0	
9	a	b	q_5	→ Flag
10	ε	a	error	→ output: b
11	a	ε	q_0	
12	ε	a	q_1	→ Flag

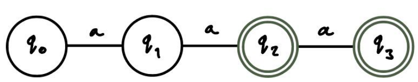
end of output : in accept state
output : ab

	Unseen	Consumed	State	
1	baba	ε	q_0	
2	aba	b	q_5	
3	aba	a	error	→ in accept state output : b
4	aba	ε	q_0	
5	ba	a	q_1	
6	a	b	q_2	
7	a	a	error	→ not in accept state REJECT

With Whitespace:

Ex

$L = \{aa, aaa\}$



Input / Maximal / Simplified

aaaaa

Maximal

Ex 1: w = aaaaa

	Unseen	Consumed	State	
1	aaaa	a	q_1	
2	aaa	a	q_2	Flag
3	aa	a	q_3	Flag
4	a	a	error	→ output: aaa
5	aa	ε	q_0	
6	a	a	q_1	
7	ε	a	q_2	Flag

end of output : in accept state
output : aa

Simplified

	Unseen	Consumed	State	
1	aaaaa	ε	q_0	
2	aaaa	a	q_1	
3	aaa	a	q_2	
4	aa	a	q_3	
5	a	a	error	→ in accept state output: aaa
6	aa	ε	q_0	
7	a	a	q_1	
	ε	a	q_2	→ end : in accept output: aa

aaaa

Maximal

Ex 2: w = aaaa

	Unseen	Consumed	State	
1	aaaa	ε	q_0	
2	aaa	a	q_1	
3	aa	a	q_2	Flag
4	a	a	q_3	Flag
5	ε	a	error	→ output: aaa
6	a	ε	q_0	
7	ε	a	q_1	

end of output : not in accept state
- CAN'T BACKTRACK Anymore
- Reject input

Simplified

	Unseen	Consumed	State	
1	aaaa	ε	q_0	
2	aaa	a	q_1	
3	aa	a	q_2	
4	a	a	q_3	
5	a	a	error	→ in accept state output: aaa
6	a	ε	q_0	
7	ε	a	q_1	→ end : not in accept REJECT

Ex: L = {++, +, i, j}		
Ex	Maximal	SMM
	Tokens	Tokens
i+++j	i, ++, +, j	i, ++, +, j
i+++++j	i, ++, ++, +, j	i, ++, ++, +, j
i++ + ++j	i, ++, +, ++, j	i, ++, +, ++, j
i+++ ++j	i, ++, +, ++, j	i, ++, +, ++, j