Finite Automata

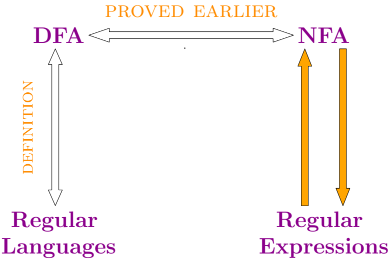
- Deterministic Finite Automata
- Languages decided by a DFA – Regular Languages
- Closure Properties of regular languages
- Non-Deterministic Finite Automata, DFA= NFA
- Regular Expression: Computation as Description
- DFA=NFA=RegExp, Generalized NFA
- Non-Regular Languages, The Pumping Lemma
- Minimizing DFA

Imdad ullah Khan

Equivalence of DFA, NFA, and RegExp

DFA $\equiv$ NFA $\equiv$ RegExp

L is regular $\iff$ L can be represented by a regex



If L can be represented by a regexp, then L can be recognized by an NFA

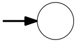
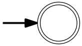
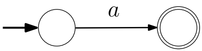
If L can be recognized by an NFA, then L can be represented by a regexp

Making NFA from Regexp

If L can be represented by a regexp, then L can be recognized by an NFA

Proof by induction on the length of regexp!

Base cases:

Regexp	**Language**	**NFA, N**	$L(N)$
$R = \emptyset$	$L(R) = \emptyset$		$\emptyset$
$R = \epsilon$	$L(R) = \{\epsilon\}$		$\{\epsilon\}$
$R = a$	$L(R) = \{a\}$		$\{a\}$

If L can be represented by a regexp, then L can be recognized by an NFA

Proof by induction on the length of regexp!

Inductive Hypothesis: Assume the language of every regexp of length $< k$ is recognized by an NFA

Inductive Step: Let R a regexp of length k

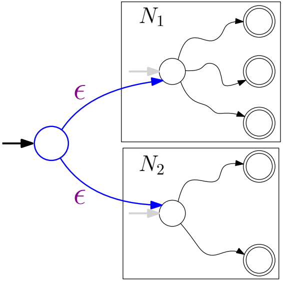
Case 1: $R = R_1 + R_2 \quad L(R) = L(R_1) \cup L(R_2)$

R_1 and R_2 have lengths $< k$,

By IH, there exists N_1 and N_2

with $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$

By closure under union $\exists\ N$ with $L(N) = L(R)$



Making NFA from Regexp

If L can be represented by a regexp, then L can be recognized by an NFA

Proof by induction on the length of regexp!

Inductive Hypothesis: Assume the language of every regexp of length $< k$ is recognized by an NFA

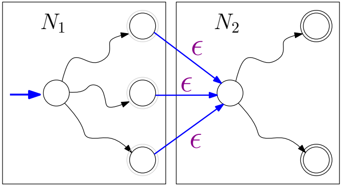
Inductive Step: Let R a regexp of length k

Case 2: $R = R_1 \circ R_2 \quad L(R) = L(R_1) \circ L(R_2)$

R_1 and R_2 have lengths $< k$,

By IH, there exists N_1 and N_2

with $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$



By closure under concatenation $\exists$ N with $L(N) = L(R)$

Making NFA from Regexp

If L can be represented by a regexp, then L can be recognized by an NFA

Proof by induction on the length of regexp!

Inductive Hypothesis: Assume the language of every regexp of length $< k$ is recognized by an NFA

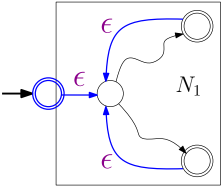
Inductive Step: Let R a regexp of length k

Case 3: $R = (R_1)^*$ $\qquad L(R) = L(R_1)^*$

R_1 has length $< k$,

By IH, there exists N_1 with $L(N_1) = L(R_1)$

By closure under start $\exists\ N$ with $L(N) = L(R)$



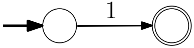
Making NFA from Regexp

If L can be represented by a regexp, then L can be recognized by an NFA

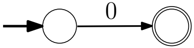
Convert $(1(1+0))^*$ to NFA

Step 1:

1 :

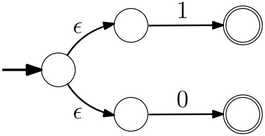


0 :



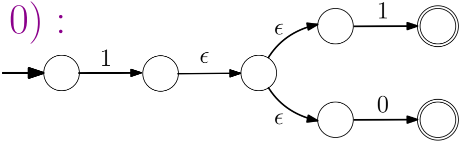
Step 2:

$1+0$:



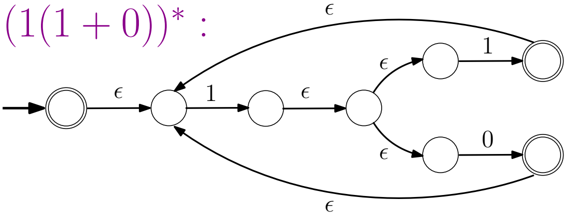
Step 3:

$1(1+0)$:



Step 4:

$(1(1+0))^*$:



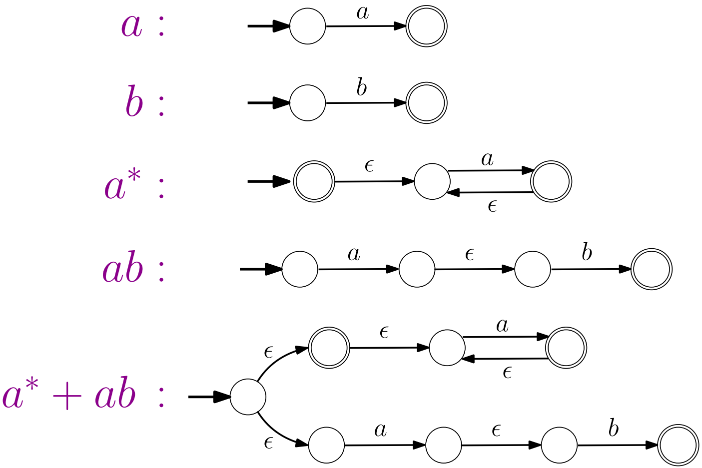
Making NFA from Regexp

If L can be represented by a regexp, then L can be recognized by an NFA

Convert

$a^* + ab$

to NFA



Making Regexp from NFA

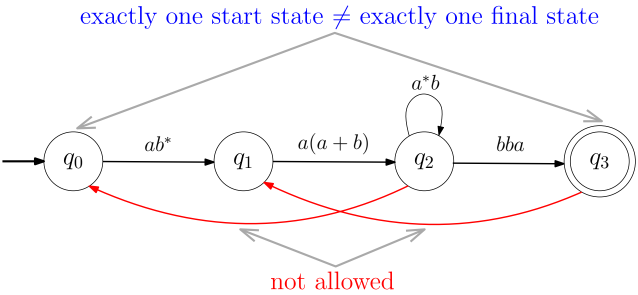
If L can be recognized by an NFA, then L can be represented by a regexp

Constructive Proof: Simplify NFA by removing states one at a time and replacing transition labels with regexps

We get generalized NFA

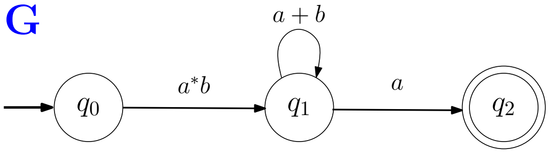
An NFA with following restriction and generalization

- Only one start state with no incoming transitions
- Only one final state with no outgoing transitions
- Start and final states are distinct
- Transitions are labeled with (general) regexps



Language of Generalized NFA (GNFA)

A GNFA accepts a string w, iff there is a walk from start state to final state with (concatenated) regexp $R_1 R_2 \cdots R_k$ such that w matches $R_1 R_2 \cdots R_k$



- G does not accept *aaaa*
- G accepts *baa*
- G accepts *bba*
- G does not accept *aabaab*
- G accepts *aabba*

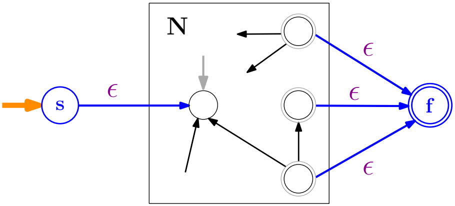
Converting NFA to GNFA

Every NFA can be converted to a GNFA

An NFA with following restriction and generalization

- Only one start state with no incoming transitions
- Only one final state with no outgoing transitions
- Start and final states are distinct
- Transitions are labeled with (general) regexps

- If needed add a new start node with no incoming transition
- If needed add a unique final state with no outgoing transition
- Existing transitions are already labeled with (simple) regexps

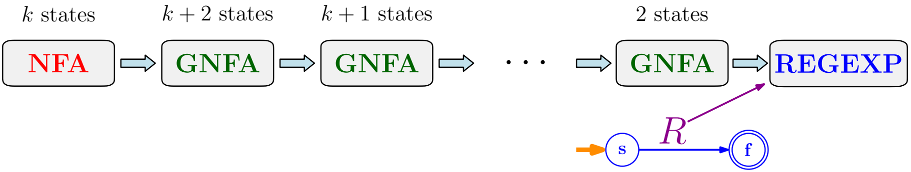


Making Regexp from NFA

If L can be recognized by an NFA, then L can be represented by a regexp

Constructive Proof: Let N be the NFA such that $L(N) = L$

- Convert the NFA N to a GNFA
- Reduce states in GNFA by removing states one at a time and replacing transition labels with regexps to account for removed state
- When only two states and one transition remains, the label of the one transition R is the required one, i.e. $L(R) = L$

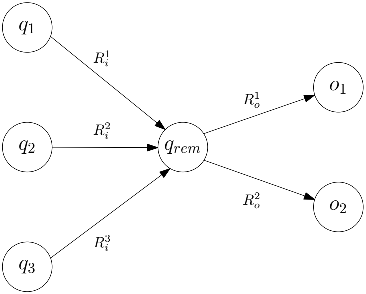


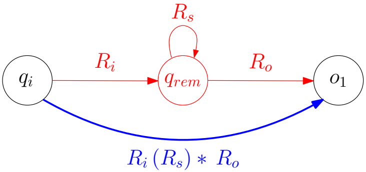
Making Regexp from NFA

If L can be recognized by an NFA, then L can be represented by a regexp

Constructive Proof: Let N be the NFA such that $L(N) = L$

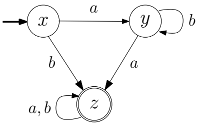
Reduce states in GNFA by removing states one at a time and replacing transition labels with regexps to account for removed state



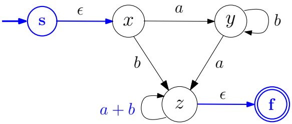


Making Regexp from NFA

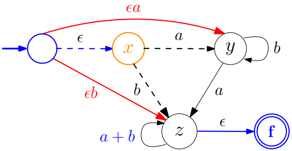
1. Input NFA



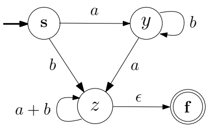
2. Initial GNFA



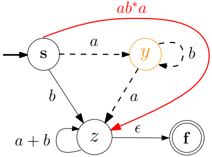
3. Removing State x



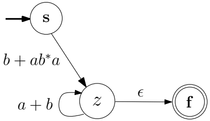
4. Redrawn GNFA



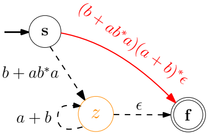
5. Removing State y



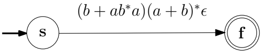
6. Redrawn GNFA



7. Removing State z



8. Final GNFA



9. Output Regexp

$$(b+ab^*a)(a+b)^*\epsilon$$

Making Regexp from NFA