CS310

Regular Expressions

Sections:1.3 page 63

September 17, 2008

Regular Expressions

Use regular operations (Union, Concat, Kleene Star) and languages to create a regular expression R whose *value* is a language L(R)

not unique in general

order of operations: *, concat, U

$$R = 0^*10^*, \; L(R)=\{w \mid w \text{ has exactly one } 1\}$$

Regular Expressions

R = 0*10*, L(R)={w | w has exactly one 1}

Many programming languages contain a Regular Expression library

```
str =~ /0*10*/ # Perl anyone?
```

Σ is used to represent one symbol from the language

Exercise

{w | (w starts with 0 and has odd length) or (w starts with 1 and has even length) }

NFA?

How do we write this as a RE?

Definition

An expression R is Regular if:

- $R = a,\ a \in \Sigma$
- $R = \varepsilon$
- $R = \emptyset$
- $R = R_1 \cup R_2$, R_1 , R_2 are regular
- $R = R_1R_2$, R_1 , R_2 are regular
- $R = R_1^*$, R_1 is regular

Theorem: A language is regular if and only if some regular expression describes it

Can be represented by an NFA

Proof

Lemma (1.55): If L is described by a regular expression R, then there exists an NFA that accepts it

Proof: For each type of regular expression, develop an NFA that accepts it.

- $R = a, a \in \Sigma$
- $R = \varepsilon$
- $R = \emptyset$
- $R = R_1 \cup R_2$, R_1 , R_2 are regular
- $R = R_1R_2$, R_1 , R_2 are regular
- $R = R_1^*$, R_1 is regular

Example

aa* U aba*b*

Exercise

{w | every odd position of w is 1 }

NFA?

How do we write this?

Exercise

{w | w does not contain 110 }

NFA?

How do we write this?

Exercise

{w| w contains even # 0s or exactly two 1s}

NFA?

How do we write this?

Proof

Lemma: If a language is regular, it is described by a regular expression

Proof Idea: If a language is regular, there exists a DFA that accepts it. We need to convert a DFA to a regular expression.

Steps:

Convert DFA to GNFA

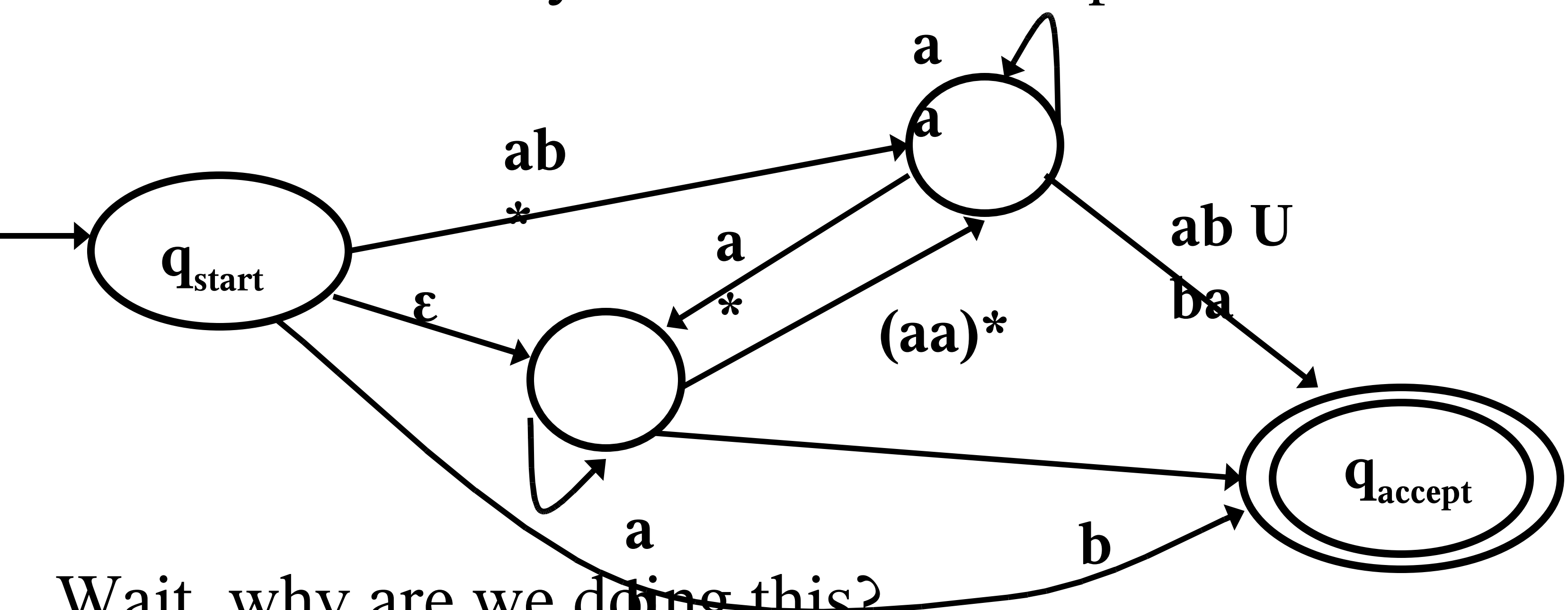
Convert GNFA to Regular Expression

GNFA?!

Generalized NFA

NFA where the transitions may have regular expressions as labels rather than just ∑ or ε

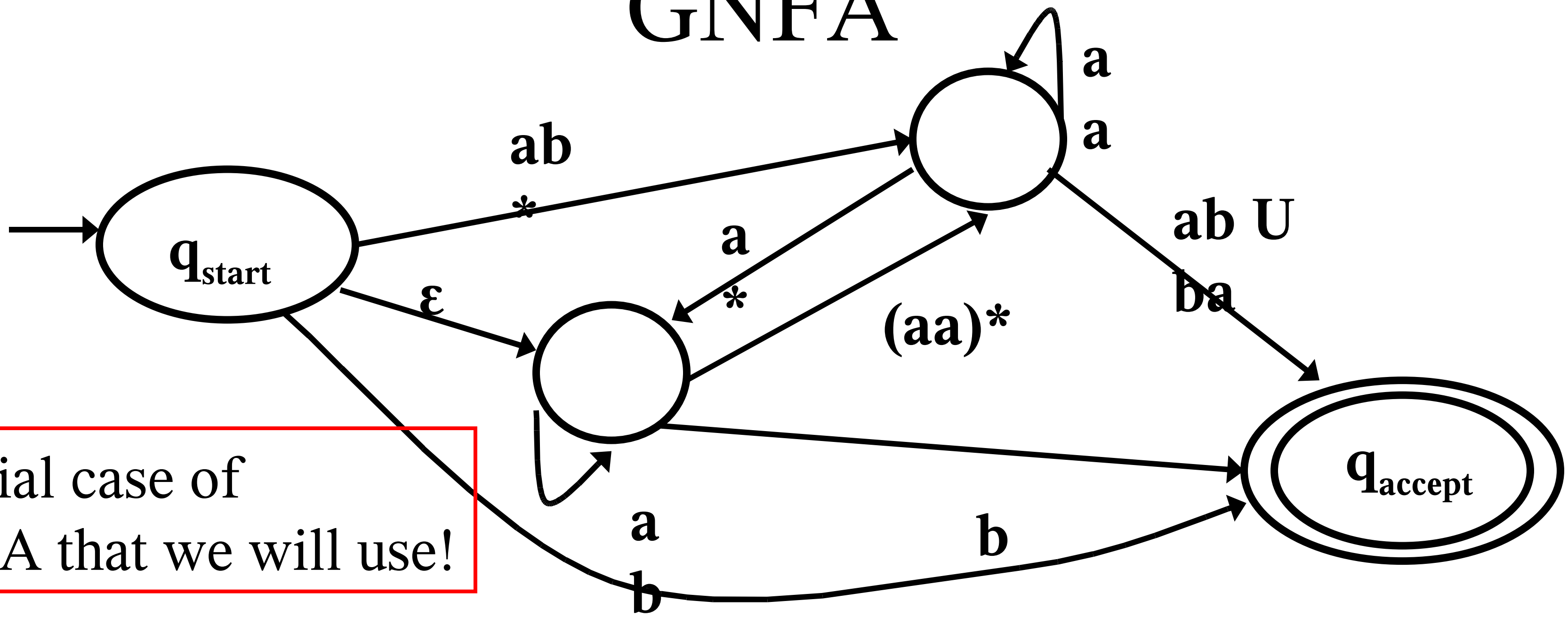
Reads *blocks* of symbols from the input



Wait, why are we doing this?

to build up the regular expression slowly from the DFA

GNFA



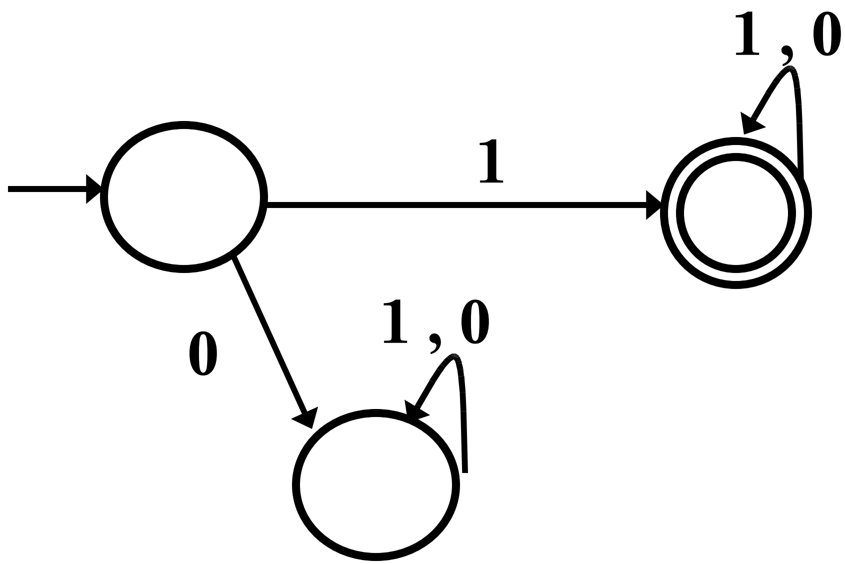
Special case of GNFA that we will use!

Start state transitions to every other state, no transitions to start state

Single accept state, transition to it from every other state, no way out, Start state != accept state

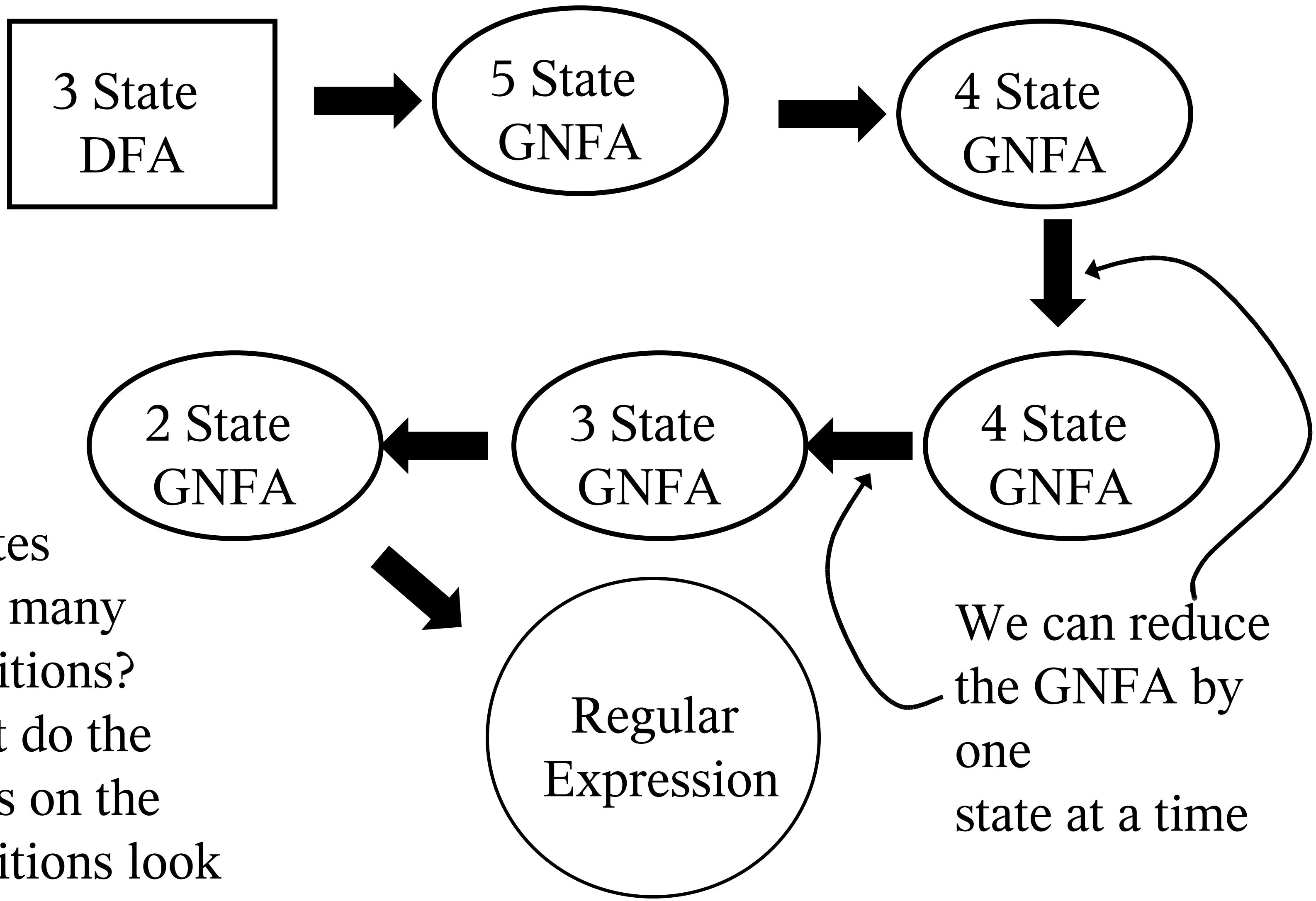
Except for the start and accept states, one arrow goes from every state to every other state (except the start state) and also from every state to itself.

DFA to GNFA



- start state with ε- transitions to old start state and Ø to every other state
- eans you never take the transition
- multiple transitions in same direction with Union
- nsition exists between states, add transitions with Ø labels (just as placehold

DFA to Regular Expression



2 states
How many transitions?
What do the labels on the transitions look like?

We can reduce the GNFA by one
state at a time

GNFA to Regular Expression

Each GNFA has at least 2 states (start and accept)

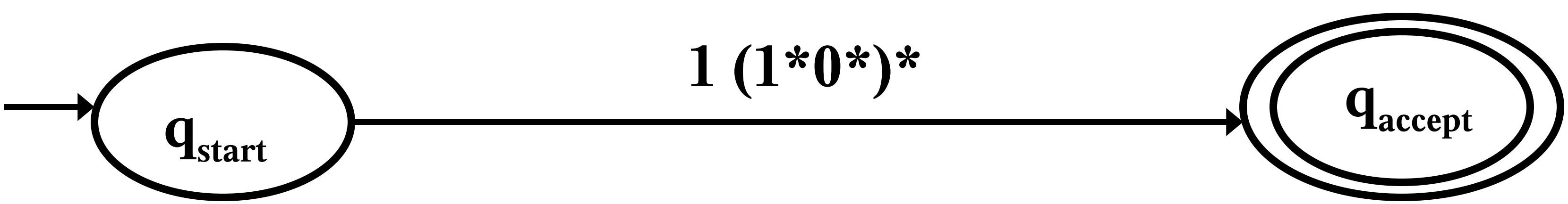
To convert GNFA to Regular Expression:

GNFA has k states, k >= 2

if k > 2 then

Produce a GNFA with k-1 states

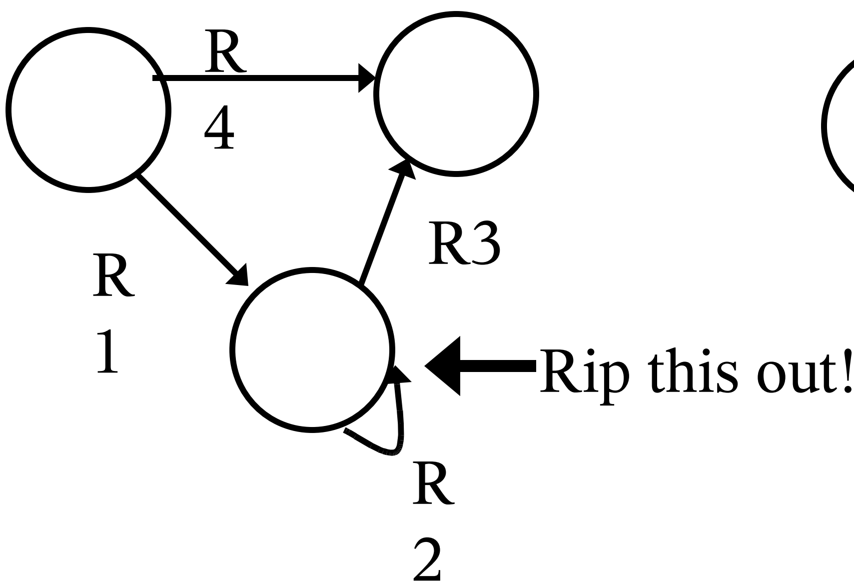
repeat

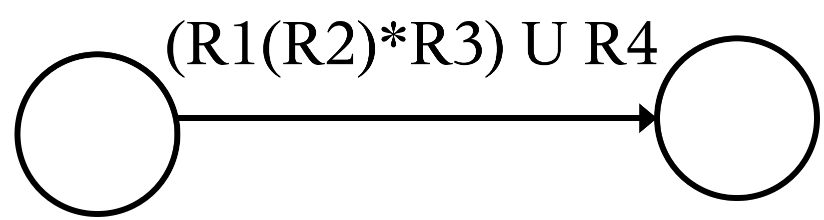


GNFA to k-1 States

Pick any state in the machine that is not the start or accept state and remove it

Fix up the transitions so the language remains the same





This change needs to be made for every pair of states connected through the removed state

Example, NFA to Regular Expression