Compilers

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

Today's Lecture

- Lexical Analysis
- Finite Automata
- Regular Expressions

Lexical Analysis

- The process of converting a sequence of characters (such as in a computer program or web page) into a sequence of lexical tokens (strings with an assigned and thus identified meaning).
- Taken from:

https://en.wikipedia.org/wiki/Lexical_analysis

- Lexical analysis is based in formal language theory.
- A formal language tool called a finite automata is used to help with recognizing string patterns (sequences of characters).

Formal Language

- **Alphabet** – Non-empty set of symbols. ∑ is used to represent the alphabet. For example, ∑ = {a, b} represents an alphabet with the symbols a and b in it.
- **Strings** – Sequence of symbols.
- **Formal Language** – Set of strings. For example: L = {aa, ab, ba, bb}.
- The strings aa, ab, ba, bb makeup the language L
- Basically, any set of strings can be thought of as a language.
- Taken from:

https://en.wikipedia.org/wiki/Alphabet_(formal_languages)
https://en.wikipedia.org/wiki/Formal_language

Kleene Star (Σ^*)

- **Σ^*** – Represents the set of all possible strings for a given alphabet. Any language over Σ will be a subset of Σ^*.

- For example:

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

$\Sigma^* = \{a, b, aa, ab, ba, bb, aaa, aab, abb, \ldots\}$

Keeps going forever. It should generate every possible combination of a and b. The set is infinite.

- For example:

$\Sigma = \{a\}$

$\Sigma^* = \{a, aa, aaa, aaaa, \ldots\}$

All strings that only contain the letter a

Example - Strings from Alphabet

- Which of the following strings are valid for the alphabet $\Sigma = \{a, b, +\}$

String	Is Valid?
abc	
a+b	
ab++	
a-b	
aabb	
//+ab	
+++++	

Example - Strings from Alphabet

- Which of the following strings are valid for the alphabet $\Sigma = \{a, b, +\}$

ANSWER

String	Is Valid?	
abc	No	← c is not in alphabet
a+b	Yes	
ab++	Yes	
a-b	No	← - is not in alphabet
aabb	Yes	
//+ab	No	← / is not in alphabet
+++++	Yes	

Example - Strings from Alphabet

- Which of the following strings are valid for the alphabet $\Sigma = \{0, 1\}$

String	Is Valid?
0	
000	
1	
010101	
012	
1-0	
01+	

Example - Strings from Alphabet

- Which of the following strings are valid for the alphabet $\Sigma = \{0, 1\}$

ANSWER

String	Is Valid?	
0	Yes	
000	Yes	
1	Yes	
010101	Yes	
012	No	← 2 is not in alphabet
1-0	No	← - is not in alphabet
01+	No	← + is not in alphabet

Example - Languages

- Which strings are members of the language L for the given alphabet?

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

$L = \{a, b, aa, bb, ab\}$

String	Is Member of Language?
a	
ab	
ba	
bb	
bbb	
bba	
abc	

- Which strings are members of the language L for the given alphabet?

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

$L = \{a, b, aa, bb, ab\}$ ← This defines the valid strings in L

String	Is Member of Language?	
a	Yes	
ab	Yes	
ba	No	← ba is not in L
bb	Yes	
bbb	No	← bbb is not in L
bba	No	← bba is not in L
abc	No	← abc is not in L or ∑* (c is not even a character in the alphabet)

Example – Languages

Example – Languages

- Which strings are members of the language L for the given alphabet?

$\Sigma = \{ ; \}$

$L = \{; , ;; , ;;;\}$

String	Is Member of Language?
;	
a	
;;;	
;;;;	
;b	
;;;;;	

Example - Languages

- Which strings are members of the language L for the given alphabet?

$\Sigma = \{ ; \}$

$L = \{; , ;; , ;;;\}$ ← This defines the valid strings in L

String	Is Member of Language?	
;	Yes	
a	No	← a is not in L and a not in alphabet
;;;	Yes	
;;;;	No	← Too many ;
;b	No	← ;b is not in L and b not in alphabet
;;;;;	No	← Too many ;

- Now on to finite automata…

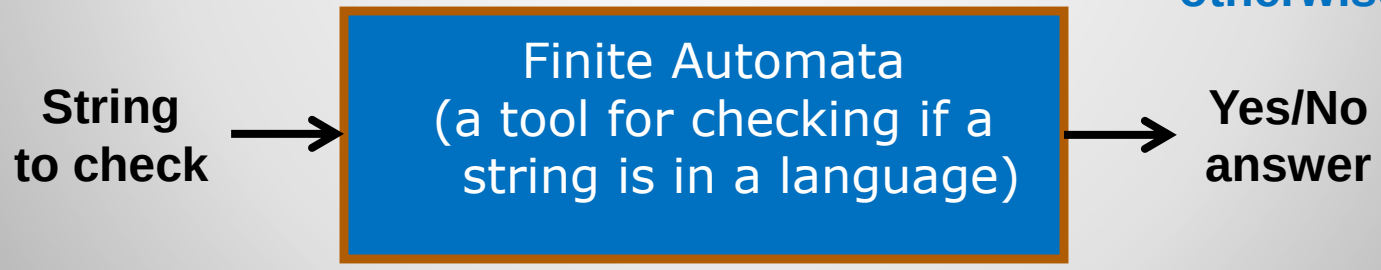
Finite Automata

- **Finite Automata** - A tool for recognizing a language.
- A finite automata is given a string and tells whether or not that string is a member of the language it defines.

- **Regular Languages** – Can be recognized by finite automata and regular expressions.

Input
Give a string as input to the finite automata

Output
Finite automata returns yes if the input string is in the language and no otherwise

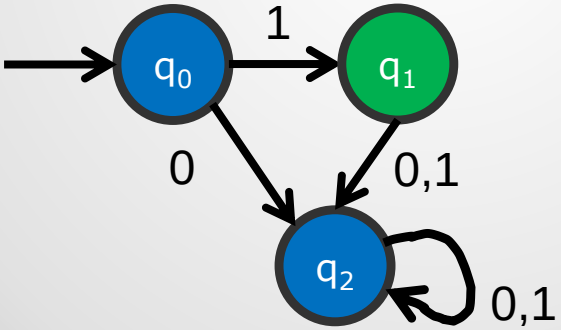


Finite Automata

Deterministic Finite Automata (DFA)

- Circles represent states. Each state is identified by the letter q and a numeric subscript. For example, q_0.
- Arrows represent transitions from one state to another.
- Starting state. One arrow from nowhere that points at a specific state (q_0 in the finite automata below).
- A green circle is an accepting state.
- If all input characters are consumed and it is in an accepting state, then the string is accepted.
- DFA. Each state must have one transition leaving it for each symbol in the alphabet.

q_0 is the staring state



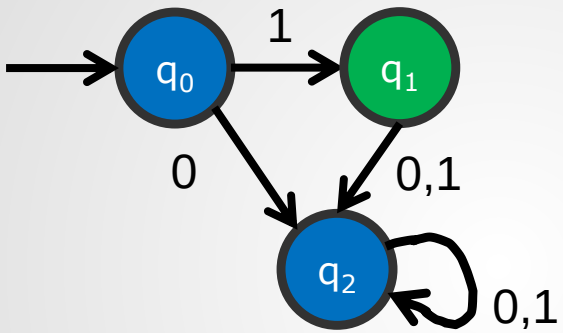
What string(s) will be accepted by this finite automata?

Deterministic Finite Automata

Deterministic Finite Automata (DFA)

- Only the string 1 will be accepted.
- L = { 1 }

q_0 is the staring state



If 1 is encountered when in state q_0 it transitions to state q_1.

If 0 is encountered when in state q_0 it transitions to q_2. Once it is in q_2 it cannot leave that state.

If it is in state q_1 and it gets any input, it will go to state q_2 and be stuck there.

Deterministic Finite Automata

This finite automata accepts strings that start with 1.

A string is accepted if there is a path from the start state to the accepting state.

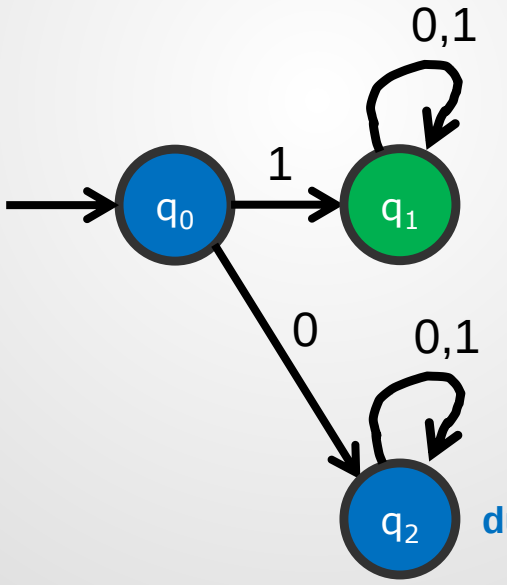
Alphabet: $\Sigma = \{0, 1\}$

$L = \{1, 11, 10, 111, 110, 101, 100\ldots\}$

States. Each circle is a state. The states in this finite automata are q_0, q_1, and q_2.

Transitions. A transition is an arrow going from one state to another. Each transition is labeled with a symbol from the alphabet.

Start State. The start state has an arrow coming into it. q_0 is the start state.



Accepting State. An accepting state is colored green (also called a final state)

Dead State. q_2 is a dead state or dummy state since the accepting state (q_1) cannot be reached.

Deterministic Finite Automata

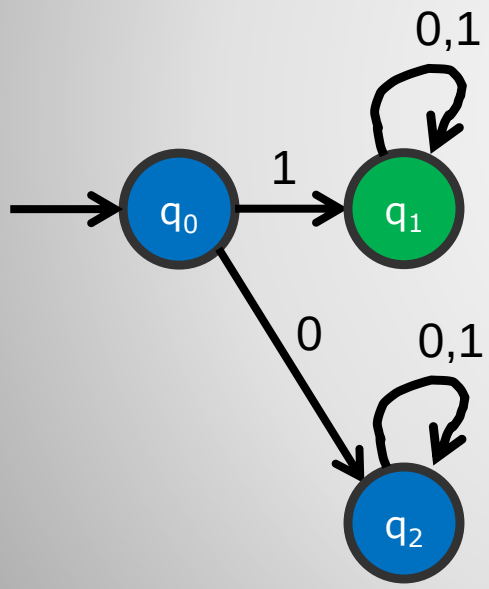
Transition Table

A transition table shows the state changes given an input.

→ identifies the start state (q_0 below)

* identifies an accepting state (q_2 below)

Alphabet: $\Sigma = \{0, 1\}$



Current State	0	1
→q_0	q_2	q_1
*q_1	q_1	q_1
q_2	q_2	q_2

DFA Transition Table

Draw the finite automata for the given alphabet and transition table.

→ Signifies starting state

* Signifies accepting state

Alphabet: $\Sigma = \{0, 1\}$

Current State	0	1
$\rightarrow q_0$	q_1	q_2
q_1	q_1	q_3
q_2	q_2	q_2
$*q_3$	q_3	q_3

Draw Finite Automata from Transition Table – Problem 1

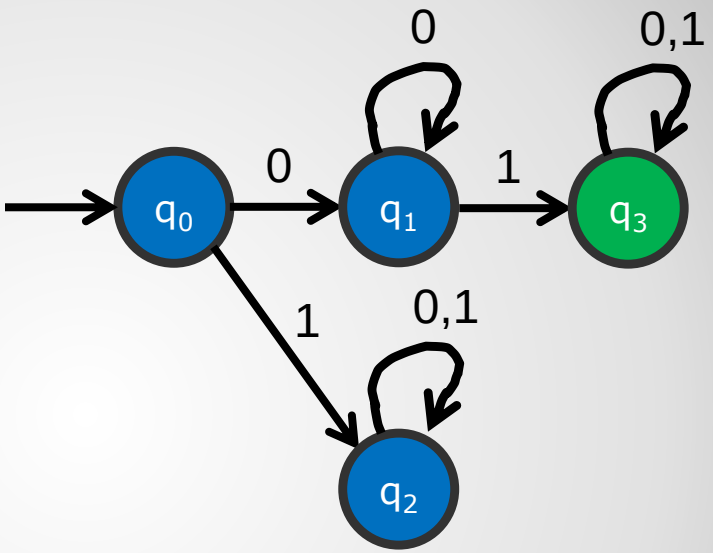
Draw the finite automata for the given alphabet and transition table.

→ Signifies starting state

* Signifies accepting state

Alphabet: Σ = {0, 1}

Current State	0	1
→q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_2	q_2
*q_3	q_3	q_3



Answer

Draw Finite Automata from Transition Table – Problem 1

Draw the finite automata for the given alphabet and transition table.

→ Signifies starting state

* Signifies accepting state

Alphabet: $\Sigma = \{0, 1\}$

Current State	0	1
$\rightarrow q_0$	q_2	q_1
q_1	q_0	q_3
q_2	q_2	q_2
$*q_3$	q_1	q_3

Draw Finite Automata from Transition Table – Problem 2

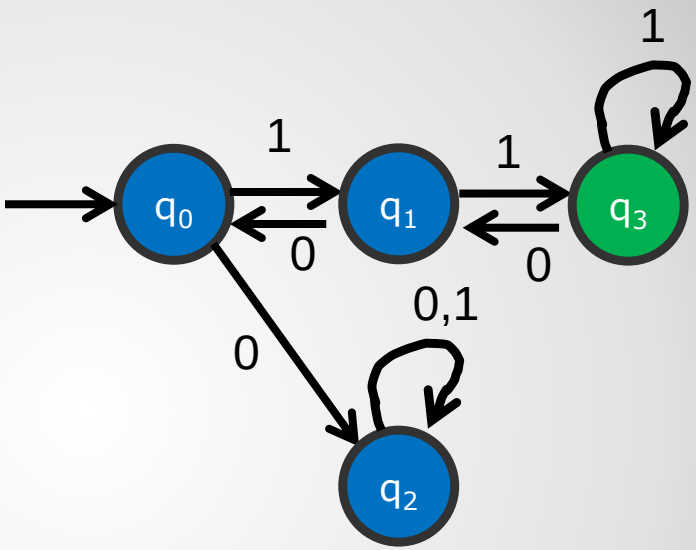
Draw the finite automata for the given alphabet and transition table.

→ Signifies starting state

* Signifies accepting state

Alphabet: Σ = {0, 1}

Current State	0	1
→q_0	q_2	q_1
q_1	q_0	q_3
q_2	q_2	q_2
*q_3	q_1	q_3



Answer

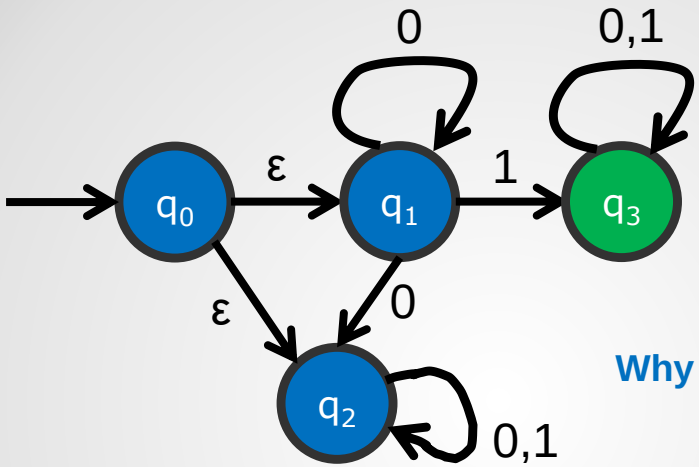
Draw Finite Automata from Transition Table – Problem 2

Types of Finite Automata

- **Deterministic Finite Automata (DFA)**
  - Every state must have a transition for each symbol in the alphabet.
  - Cannot have more than one edge with the same symbol leaving a state.
- **Non-deterministic Finite Automata (NFA)**
  - Allows more than one transition with the same label from a state.
  - Allows transitions for an empty string (ε).
- All DFAs are NFAs.
- Not every NFA is an DFA.
  - If an NFA has a ε transition, then it is not a DFA.
  - If an NFA has a state with multiple transitions with the same label leaving that state it is not a DFA.
- Note: ε is epsilon.

Nondeterministic Finite Automata (NFA)

q_0 is the staring state

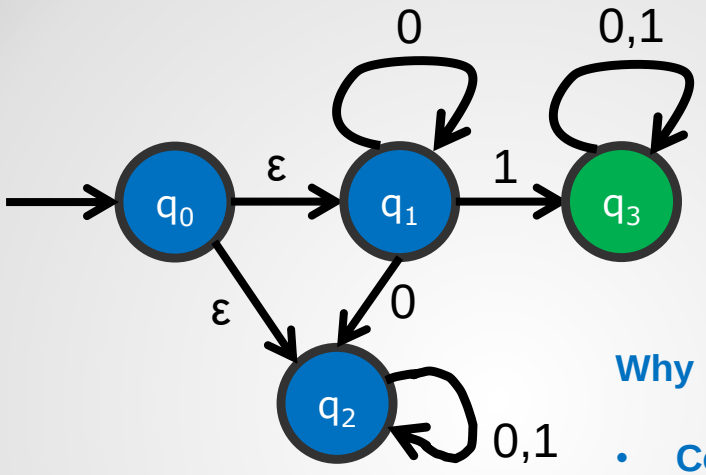


Why is this an NFA and not a DFA?

Nondeterministic Finite Automata

Nondeterministic Finite Automata (NFA)

q_0 is the staring state



Why is this an NFA and not a DFA?

- **Contains empty string transitions (ε).**
- **Contains multiple transitions coming from the same state that have the same label.**

Nondeterministic Finite Automata

- Now on to regular expressions…

Regular Expressions

Regular Expressions

- Regular Expression - More human-readable way to define a language (compared to finite automata).
- Regular expressions (RE) are equivalent to finite automata (FA) with respect to the languages they define.
- The language defined by a regular expression is called a regular language.
- A regular expression contains the symbols from the alphabet and the following three operations:
  - Union – Use the | symbol for union. For example, a|b means either a or b.
  - Concatenation – Place the characters one after another. For example, ab means an a followed by a b.
  - Closure – Use the * symbol for closure (the star means 0 or more). For example, a* means 0 or more of a.
- Square Brackets [ ]. Use the [ ] to select one character from a range. For example, [0-9] means it will match any character in the range 0 to 9. The same as (0|1|2|3|4|5|6|7|8|9).
- Parenthesis. Used to show precedence.

- Regular expression: [1-9][0-9]*

This means one character in the range 1-9 followed by 0 or more characters in the range 0-9. Some strings that follow this pattern are: 1, 111, 10, 1999.

- Regular expression: a | b

This means one character that is either a or b. The only strings that follow this pattern are: a, b.

- Regular expression: [a-c] | xy

This means one character in the range a-c or x. The only strings that follow this pattern are: a, b, c, xy

Regular Expression Examples

Regular Expression Examples

- Regular expression: a*b*

This means 0 or more a characters followed by 0 or more b characters. Some strings that follow this pattern are: ab, aaab, abbb, aaa, bbb

- Regular expression: (ab)*

This means 0 or more ab strings (not the same as a*b*). Some strings that follow this pattern are: ab, abab, ababab, abababab

- Regular expression:

c*d*(cd)(ef)*

- Which of the following strings are recognized by the above regular expression?

String	Is Recognized
cd	
ef	
cde	
cdef	
dcdef	
dcef	
cdcdcdef	
cccdcd	

Regular Expression Example 1

- Regular expression:

c*d*(cd)(ef)*

- Which of the following strings are recognized by the above regular expression?

String	Is Recognized
cd	Yes
ef	No
cde	No
cdef	Yes
dcdef	Yes
dcef	No
cdcdcdef	No
cccdcd	Yes

Regular Expression Example 1

- Regular expression:

(p|s)*((gh)|i)*

- Which of the following strings are recognized by the above regular expression?

String	Is Recognized
s	
ppp	
sghi	
ppgg	
ssghghi	
pghp	
pig	
pigh	

Regular Expression Example 2

- Regular expression:

(p|s)*((gh)|i)*

- Which of the following strings are recognized by the above regular express?

String	Is Recognized
s	Yes
ppp	Yes
sghi	Yes
ppgg	No
ssghghi	Yes
pghp	No
pig	No
pigh	Yes

Regular Expression Example 2

- Regular expression:

([a-c][d-f])*z

- Which of the following strings are recognized by the above regular express?

String	Is Recognized
a	
dz	
abcdef	
z	
aaazz	
zf	
eeez	
bbbeeez	

Regular Expression Example 3

- Regular expression:

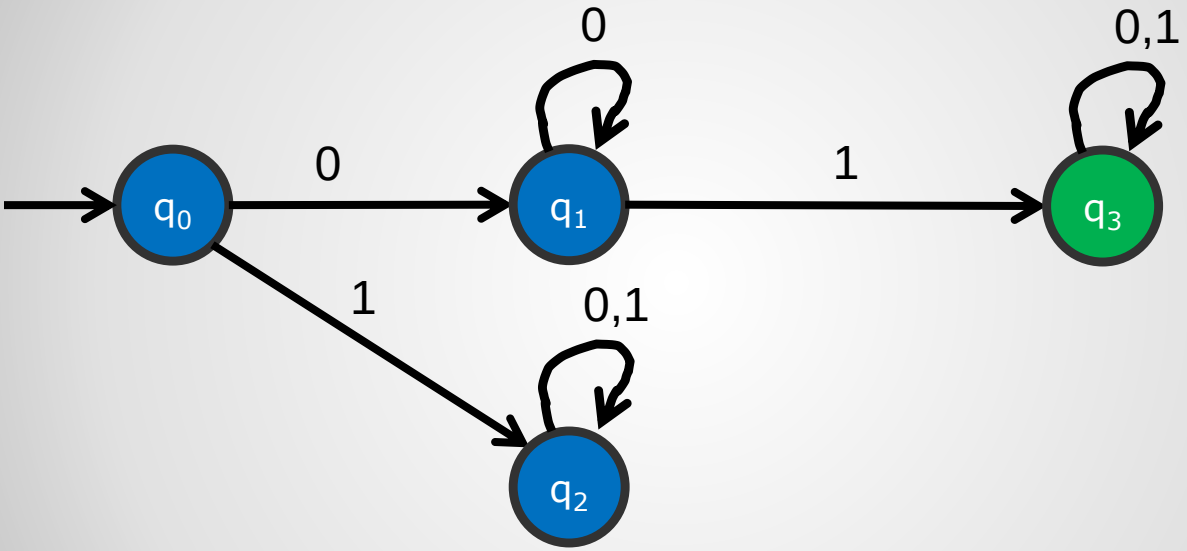
([a-c][d-f])*z

- Which of the following strings are recognized by the above regular express?

String	Is Recognized
a	No
dz	Yes
abcdef	No
z	Yes
aaazz	No
zf	No
eeez	Yes
bbbeeez	Yes

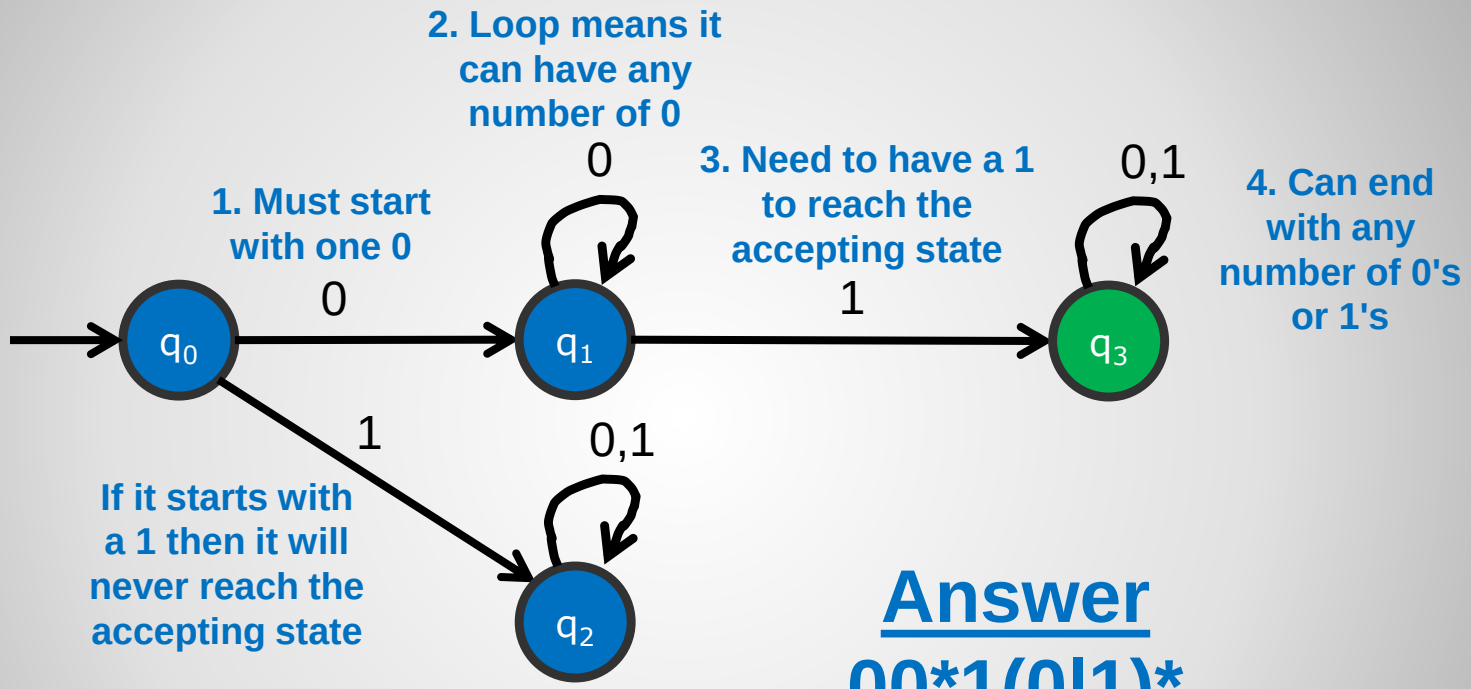
Regular Expression Example 3

What regular expression is equivalent to the following deterministic finite automata?



Regular Expression for a DFA

What regular expression is equivalent to the following deterministic finite automata?

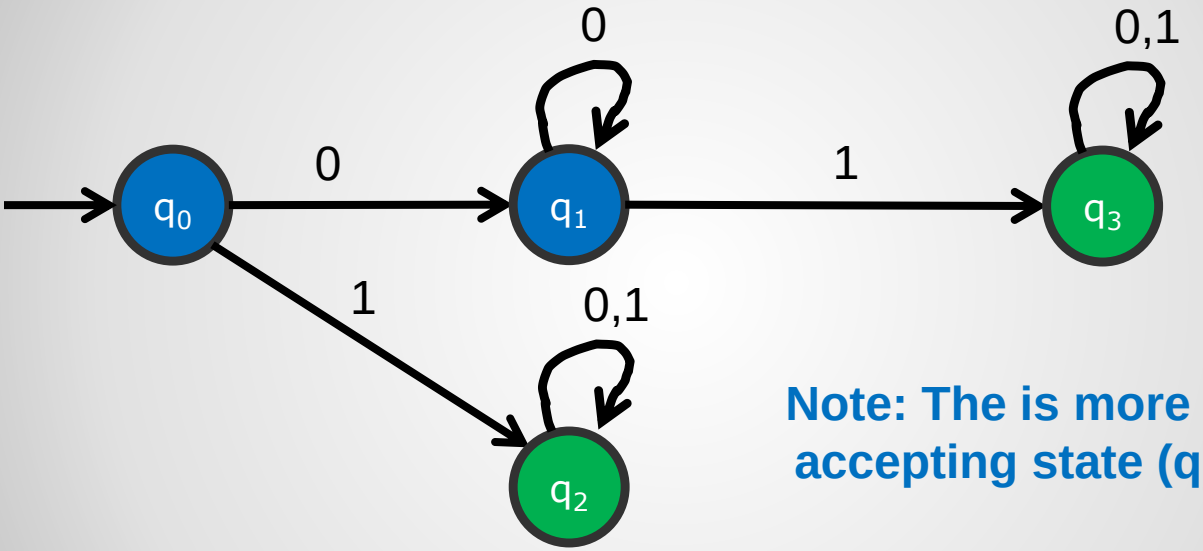


Answer

00*1(0|1)*

Regular Expression for a DFA

What regular expression is equivalent to the following deterministic finite automata?

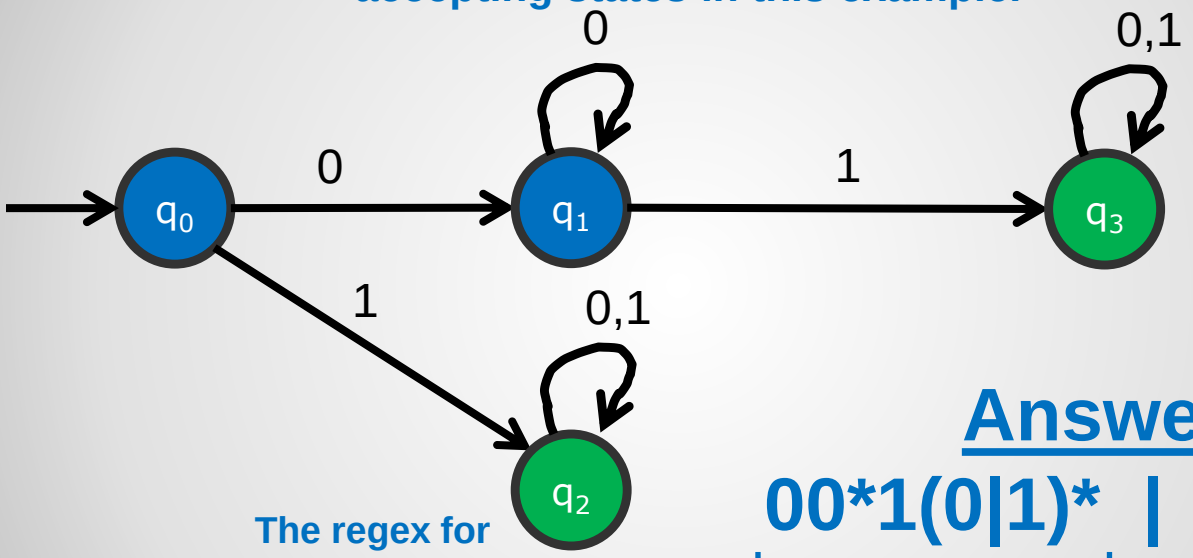


Note: The is more than one accepting state (q_2 and q_3)

Regular Expression for a DFA

What regular expression is equivalent to the following deterministic finite automata?

Use a union for each path to an accepting state. q_2 and q_3 are accepting states in this example.



The regex for this part is: 00*1(0|1)*

The regex for this part is: 1(0|1)*

Answer

00*1(0|1)* | 1(0|1)*

Path to q_3 | **Union** | **Path to q_2**

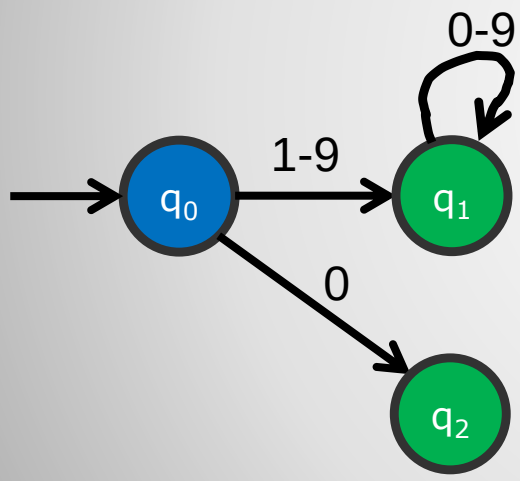
Regular Expression for a DFA

Regular Expressions vs Finite Automata

- Regular Expressions and finite automata both recognize regular languages.
- They have exactly the same power with respect to recognizing languages (Kleene's Theorem).
- You can convert any finite automata to an equivalent regular expression (FA → Reg exp).
- You can convert any regular expression to an equivalent finite automata (Reg exp → FA).
- Note: FA can be either a DFA or an NFA.

Unsigned Integer RE and FA

- Regular Expression
  0 | [1-9][0-9]*
- Finite Automata



This regular expression is either a 0 or [1-9] followed by any number of [0-9]

[0-9] loop transition can be done any number of times. This is equivalent to the [0-9]* in the regular expression.

The 0 transition is the same as the 0 in the regular expression (left side of regular expression)

Unsigned Integer RE and FA

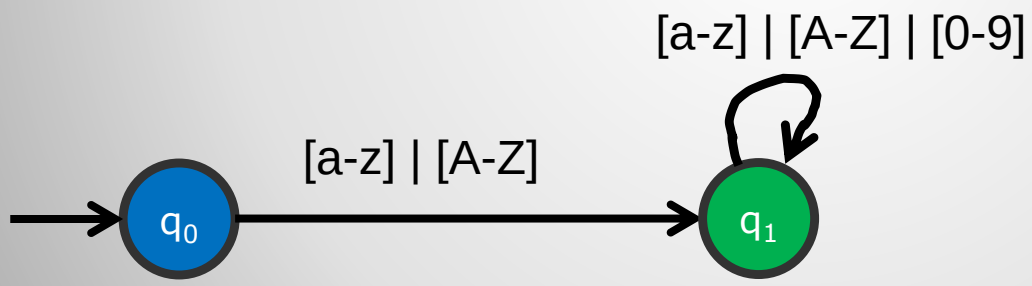
Identifier RE and FA

- Assume we define an identifier as being any character followed by any number of alphanumeric characters.

- Regular Expression
  ([a-z] | [A-Z])([a-z] | [A-Z] | [0-9])*

Concatenation is being done between the two sets of parenthesis. One character (from first parenthesis) is concatenated with any number of alphanumeric characters (second parenthesis).

- Finite Automata



The loop transition also allows numeric characters [0-9]

Identifier RE and FA

Reserved Word RE and FA

- Assume we need to recognize the "if" keyword in a language. If is the character i followed by the character f.
- Regular Expression (recognizes the word if)
  if

Concatenation is being done. The character i is followed by the character f.

- Finite Automata (recognizes the word if)

No loop transition. State q_1 is not an accepting state.

Reserved Word RE and FA

Convert Regular Expression to an NFA

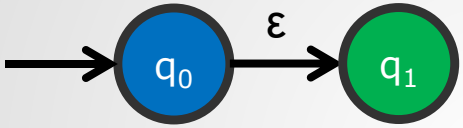
- Use Thompson's construction algorithm to convert a regular expression to an NFA.

https://en.wikipedia.org/wiki/Thompson%27s_construction

- Thompson's construction algorithm focuses on the following three operations:
  - Union
  - Concatenation
  - * operations.

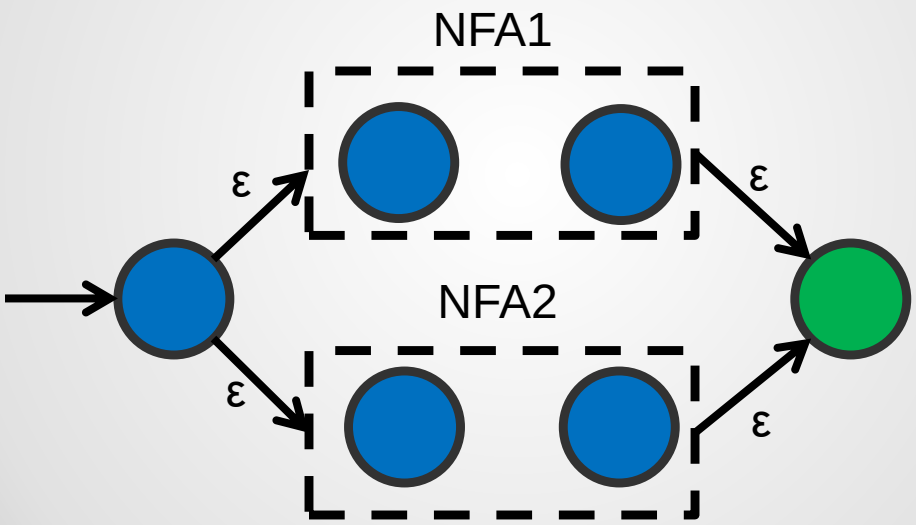
- There should be one starting state and one accepting state in the constructed NFA.

- Empty expression
- This goes directly from the start state to the accept state.



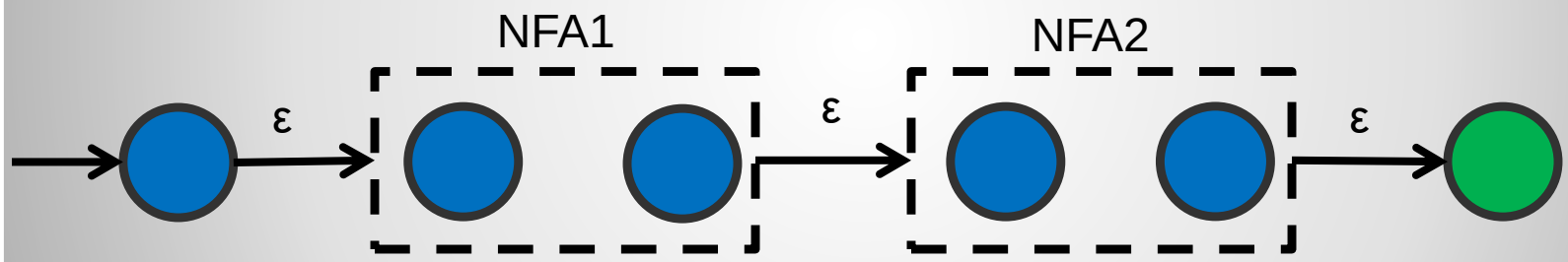
Convert Regular Expression to an NFA – Empty Expression

- Union expression.
- NFA1 and NFA2 are NFAs.
- The two paths through the whole NFA allow strings that are accepted by either NFA1 or NFA2.
- This is the union of both NFAs.



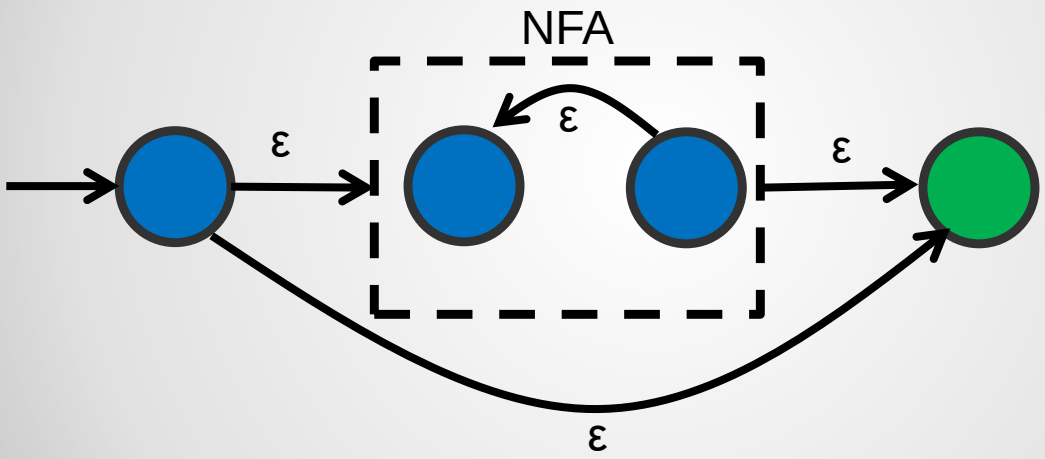
Convert Regular Expression to an NFA – Union

- Concatenation expression.
- NFA1 is be followed by NFA2.
- Only strings that match NFA1 followed by NFA2 will be accepted.
- This is the concatenation of both NFAs.



Convert Regular Expression to an NFA – Concatenation

- * expression.
- The NFA can be applied 0 or more times to the string.



Convert Regular Expression to an NFA – *

- **End of Slides**

End of Slides