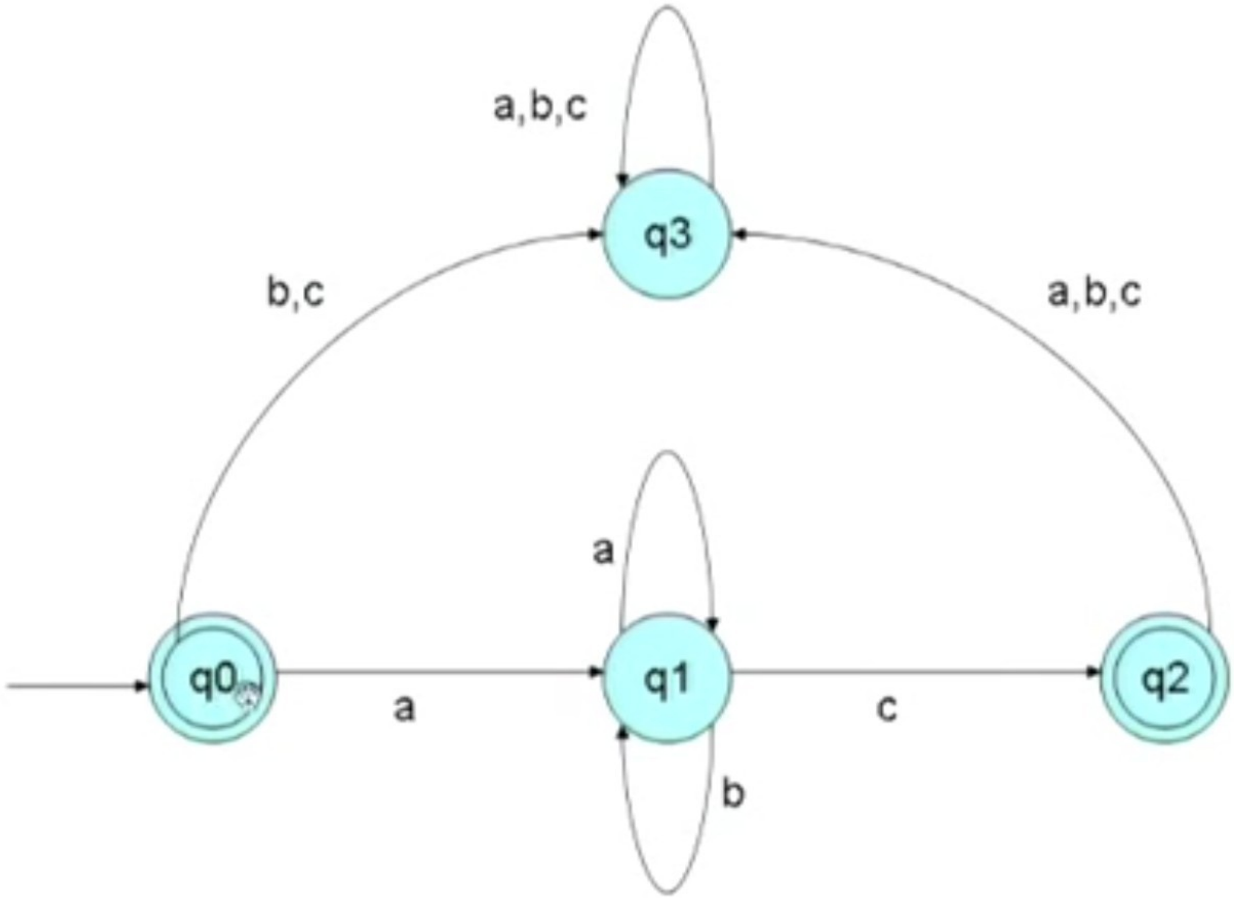
Lexical Analysis
(Part 2)

CSE 415: Compiler Construction

Phases of a Compiler

- Recognition of tokens - finite automata and transition diagrams
- Specification of tokens - regular expressions and regular definitions

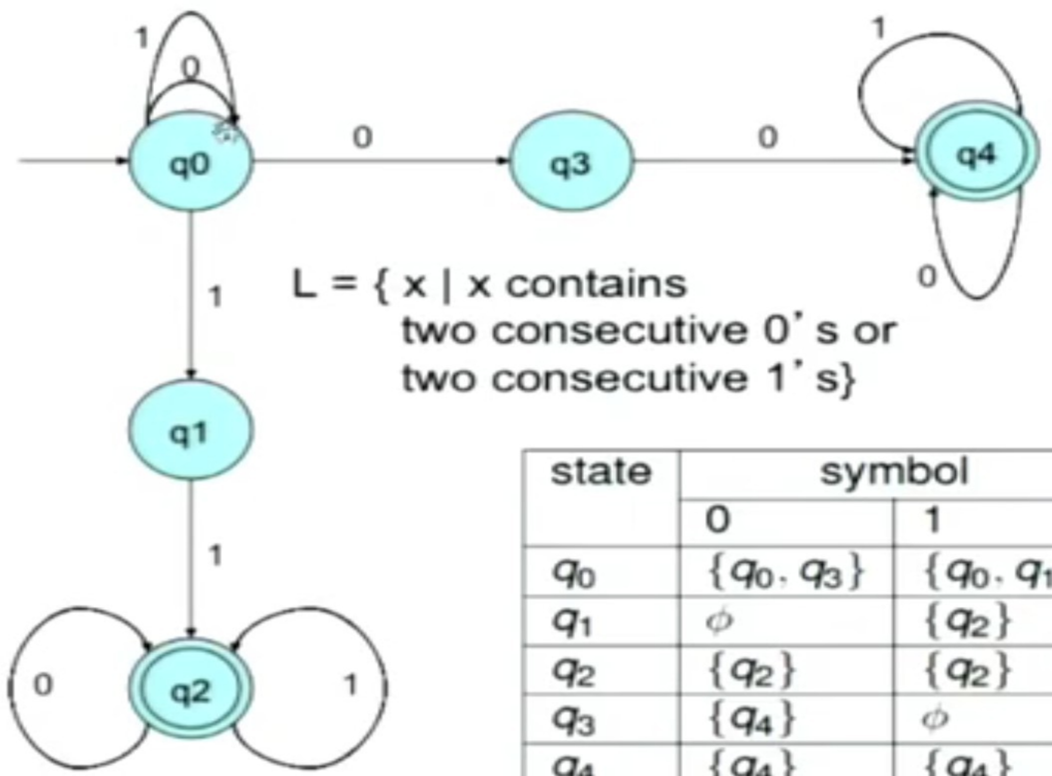
FSA Example



Non-deterministic FSA

- NFAs are FSA which allow 0, 1, or more transitions from a state on a given input symbol
- An NFA is a 5-tuple as before, but the transition function δ is different
- $\delta(q, a)$ = the set of all states p, such that there is a transition labelled a from q to p
- $\delta : Q \times \Sigma \rightarrow 2^Q$
- A string is accepted by an NFA if there *exists* a sequence of transitions corresponding to the string, that leads from the start state to some final state
- Every NFA can be converted to an equivalent DFA that accepts the same language

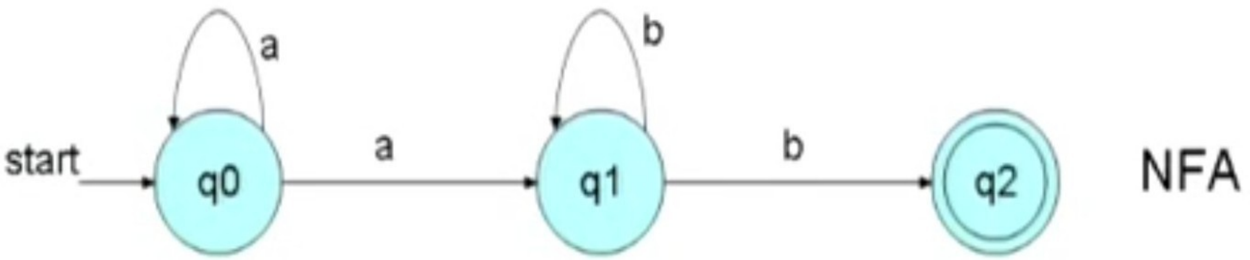
Non-deterministic FSA

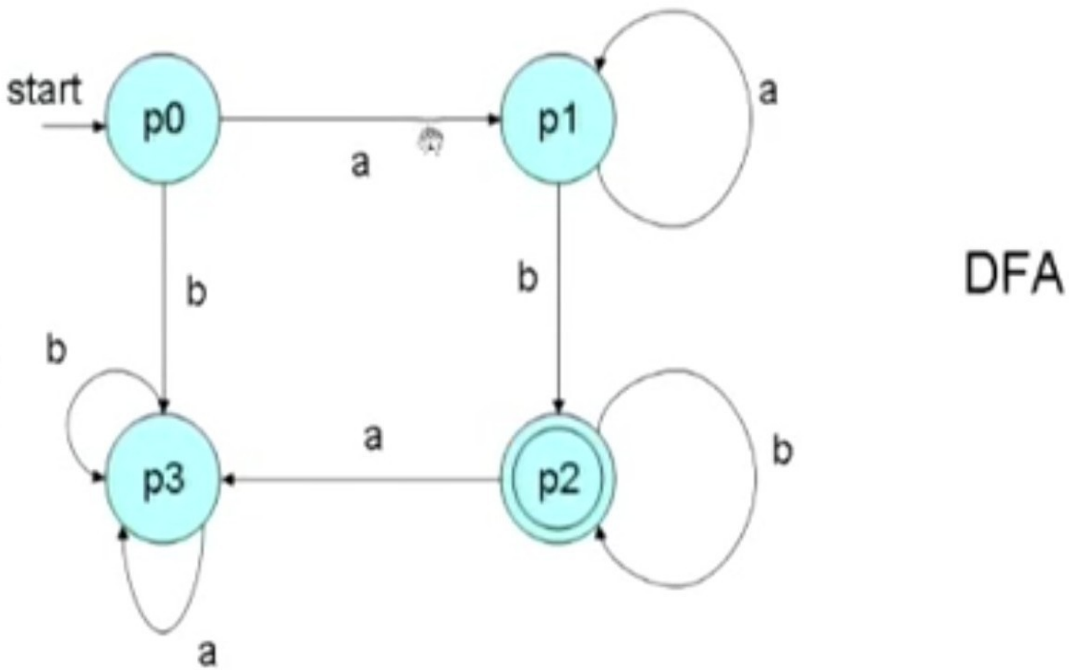


L = { x | x contains
two consecutive 0's or
two consecutive 1's}

state	symbol	
	0	1
q_0	$\{q_0, q_3\}$	$\{q_0, q_1\}$
q_1	ϕ	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$
q_3	$\{q_4\}$	ϕ
q_4	$\{q_4\}$	$\{q_4\}$

Equivalence of NFA and DFA



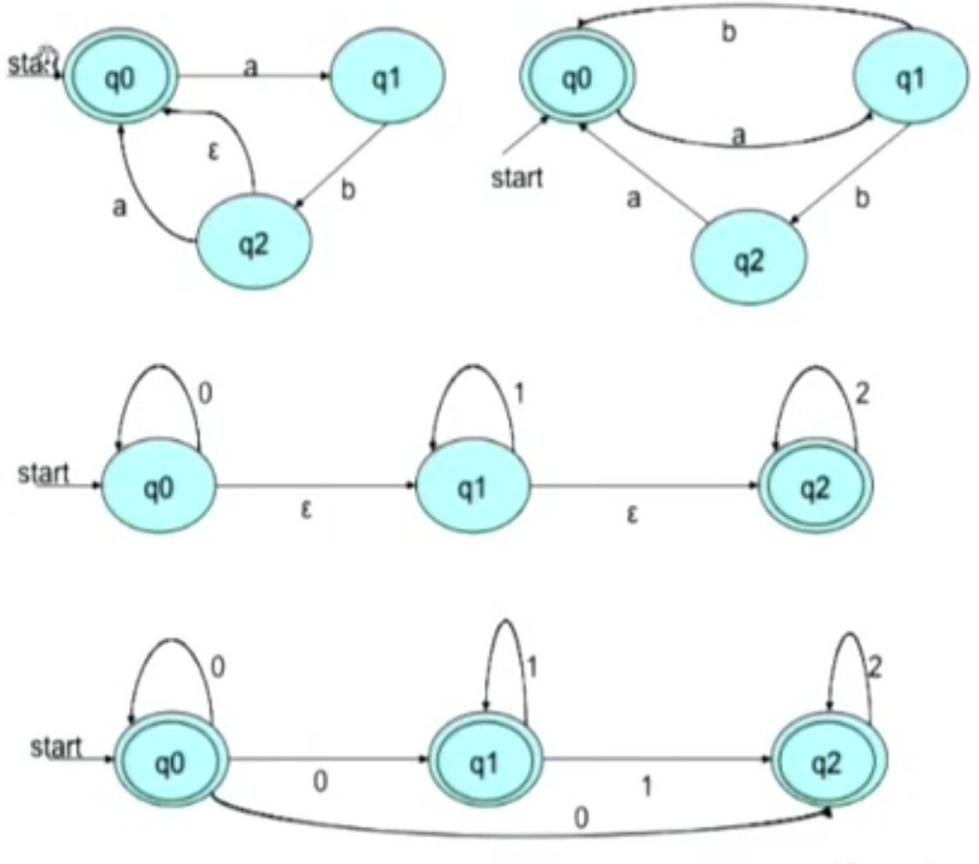


p0=[q0]
p1=[q0,q1]
p2=[q1,q2]
p3=Φ

Example of NFA to DFA Conversion

- The start state of the DFA would correspond to the set $\{q_0\}$ and will be represented by $[q_0]$
- Starting from $\delta([q_0], a)$, the new states of the DFA are constructed on *demand*
- Each subset of NFA states is a *possible* DFA state
- All the states of the DFA containing some final state as a member would be final states of the DFA
- For the NFA presented before (whose equivalent DFA was also presented)
  - $\delta[q_0], a) = [q_0, q_1]$, $\delta([q_0], b) = \phi$
  - $\delta([q_0, q_1], a) = [q_0, q_1]$, $\delta([q_0, q_1], b) = [q_1, q_2]$
  - $\delta(\phi, a) = \phi$, $\delta(\phi, b) = \phi$
  - $\delta([q_1, q_2], a) = \phi$, $\delta([q_1, q_2], b) = [q_1, q_2]$
  - $[q_1, q_2]$ is the final state
- In the worst case, the converted DFA may have 2^n states, where n is the no. of states of the NFA

NFA with ε-Move



Regular Expressions

Let Σ be an alphabet. The REs over Σ and the languages they denote (or generate) are defined as below

1. ϕ is an RE. $L(\phi) = \phi$
2. ϵ is an RE. $L(\epsilon) = \{\epsilon\}$
3. For each $a \in \Sigma$, a is an RE. $L(a) = \{a\}$
4. If r and s are REs denoting the languages R and S, respectively
    - (rs) is an RE, $L(rs) = R.S = \{xy \mid x \in R \wedge y \in S\}$
    - $(r + s)$ is an RE, $L(r + s) = R \cup S$
    - (r^*) is an RE, $L(r^*) = R^* = \bigcup_{i=0}^{\infty} R^i$

    (L^* is called the *Kleene closure* or *closure* of L)

Example of Regular Expressions

6. $r = c^*(a + bc^*)^*$
   L = set of all strings over {a,b,c} that do not have the substring *ac*
7. $L = \{w \mid w \in \{a.b\}^* \wedge w \text{ ends with } a\}$
   $r = (a + b)^* a$
8. L = {if, then, else, while, do, begin, end}
   $r = if + then + else + while + do + begin + end$

Example of Regular Definitions

A *regular definition* is a sequence of "equations" of the form $d_1 = r_1;\ d_2 = r_2;\ \ldots;\ d_n = r_n$, where each d_i is a distinct name, and each r_i is a regular expression over the symbols $\Sigma \cup \{d_1, d_2, \ldots, d_{i-1}\}$

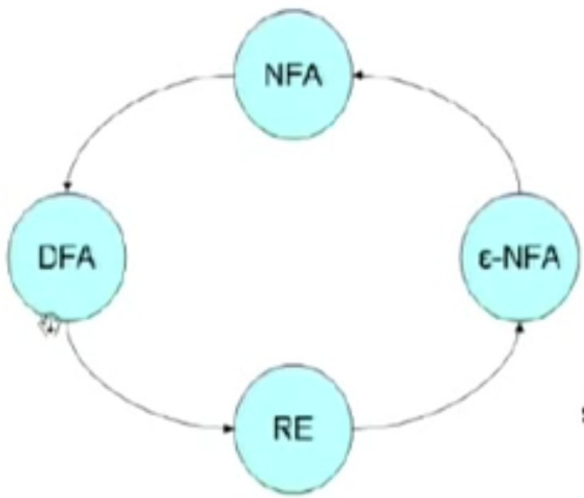
1. identifiers and integers
   $letter = a + b + c + d + e;\ \ digit = 0 + 1 + 2 + 3 + 4;$
   $identifier = letter(letter + digit)^*;\ \ number = digit\ digit^*$

2. unsigned numbers
   $digit = 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9;$
   $digits = digit\ digit^*;$
   $optional\_fraction = \dot{d}igits + \epsilon;$
   $optional\_exponent = (E(+|-|\epsilon)digits) + \epsilon$
   $unsigned\_number =$
   $digits\ optional\_fraction\ optional\_exponent$

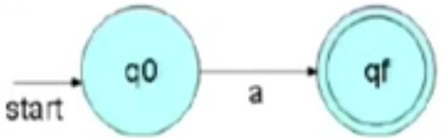
Equivalence of Regular Expressions and FSA

- Let r be an RE. Then there exists an NFA with ϵ-transitions that accepts $L(r)$. The proof is by construction.
- If L is accepted by a DFA, then L is generated by an RE. The proof is tedious.

Construction of FSA from RE

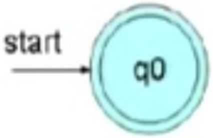


r=a

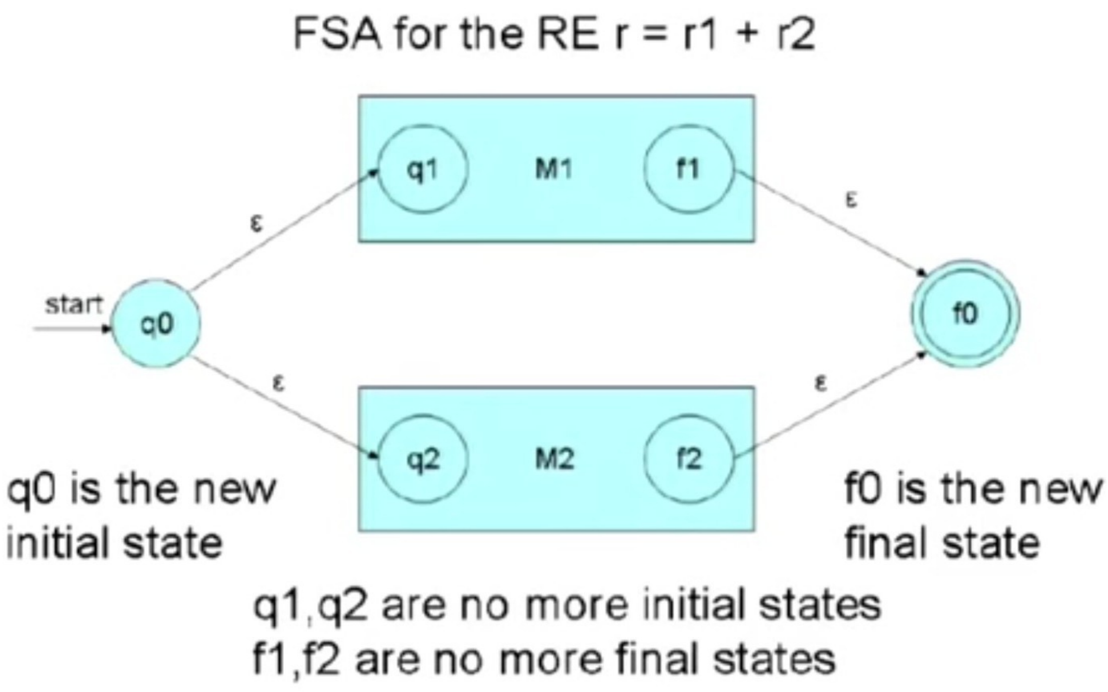


r=Φ

r=ε



Construction of FSA from RE



Construction of FSA from RE

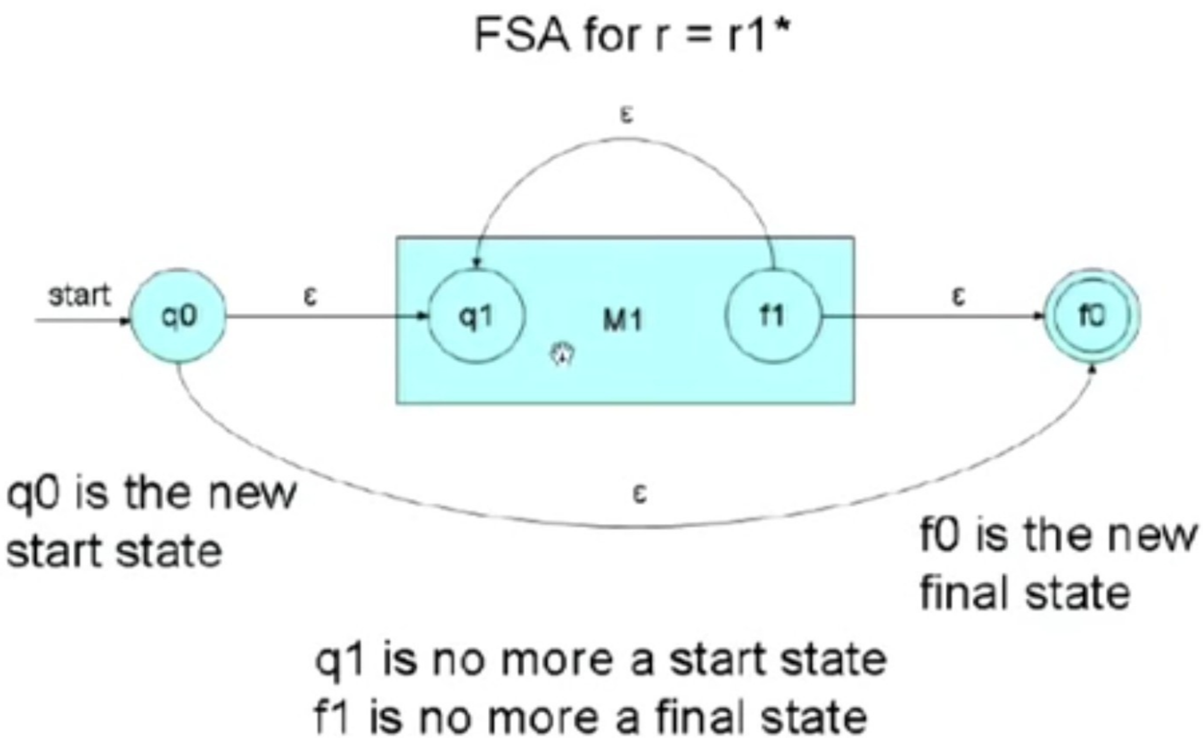
FSA for RE r = r1 r2

q1 is the new start state

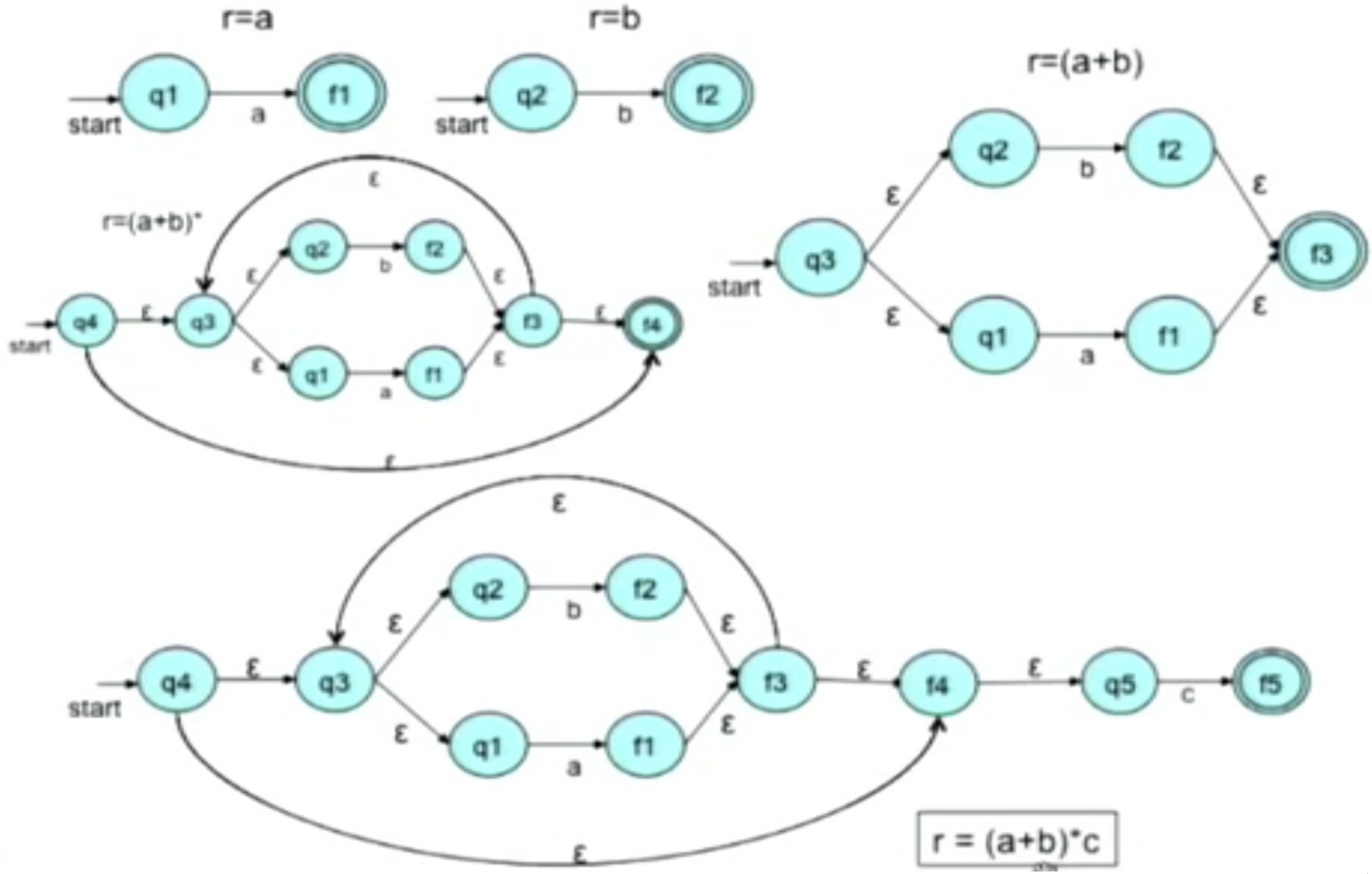
f2 is the new final state

f1 is no more a final state
q2 is no more a start state

Example of Regular Expressions



Example of Regular Expressions

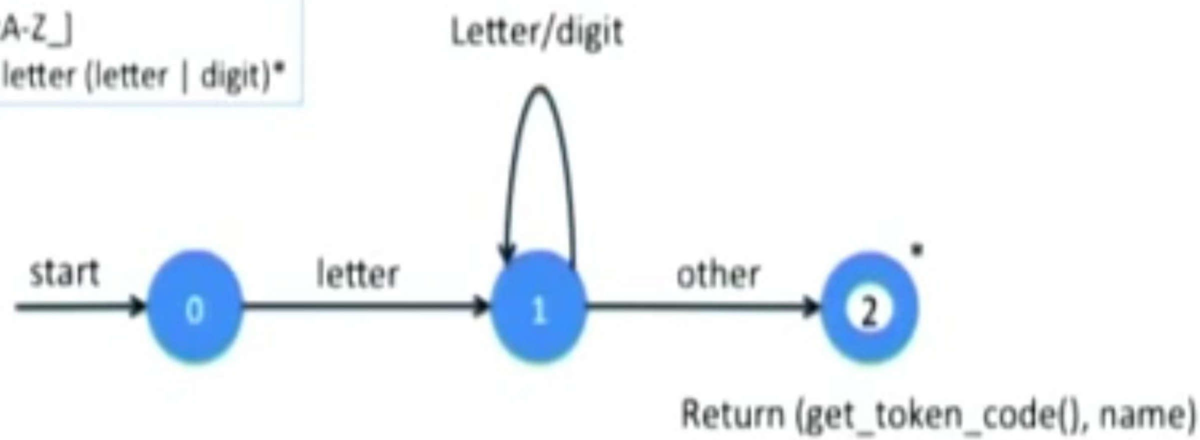


Transition Diagrams

- Transition diagrams are generalized DFAs with the following differences
  - Edges may be labelled by a symbol, a set of symbols, or a regular definition
  - Some accepting states may be indicated as *retracting states*, indicating that the lexeme does not include the symbol that brought us to the accepting state
  - Each accepting state has an action attached to it, which is executed when that state is reached. Typically, such an action returns a token and its attribute value
- Transition diagrams are not meant for machine translation but only for manual translation

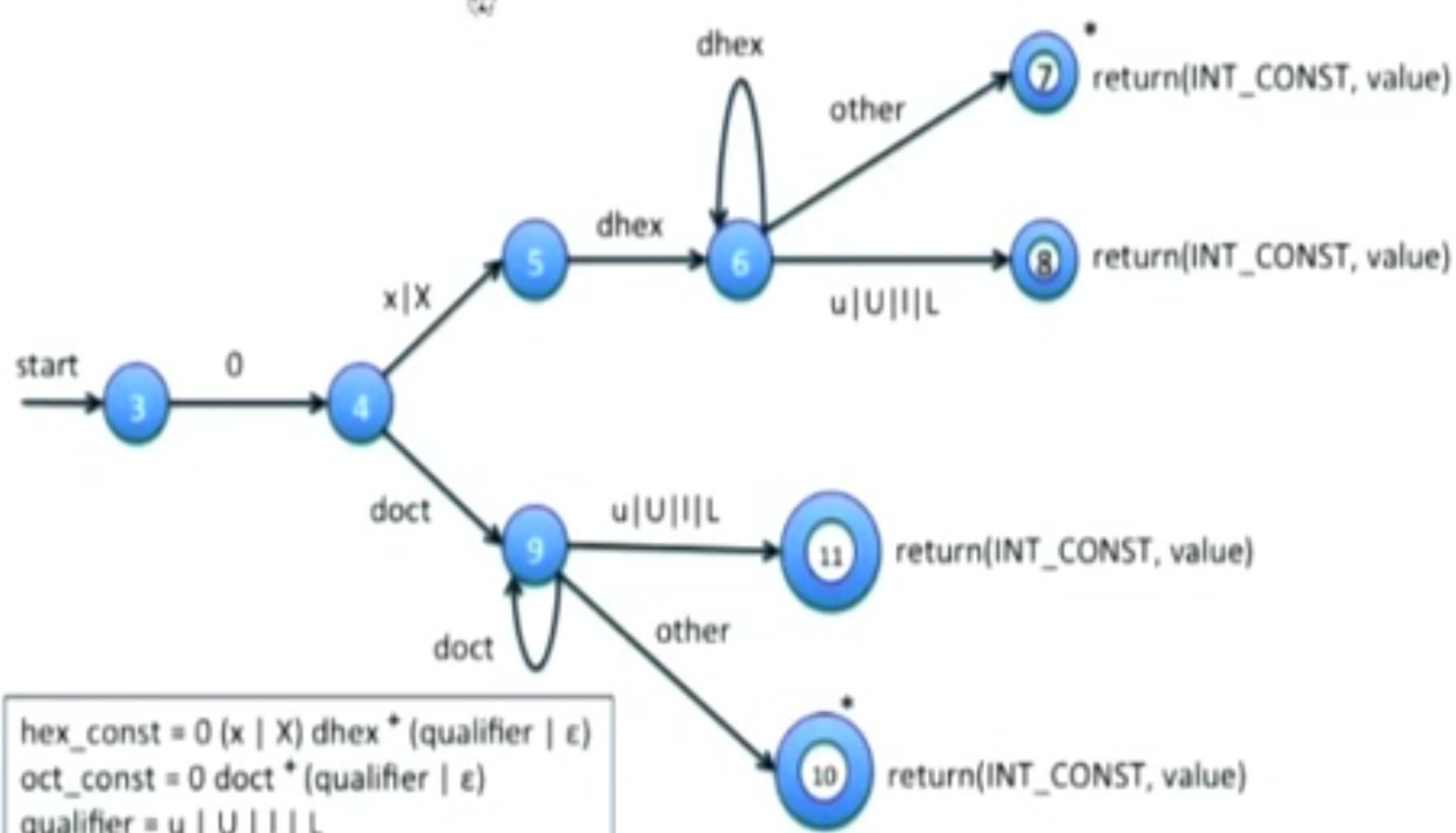
Transition Diagram for Identifiers and Reserve Words

letter = [a-zA-Z_]
Identifier = letter (letter | digit)*



- '*' indicates retraction state
- get_token_code() searches a table to check if the name is a reserved word and returns its integer code, if so
- Otherwise, it returns the integer code of IDENTIFIER token, with name containing the string of characters forming the token (name is not relevant for reserved words)

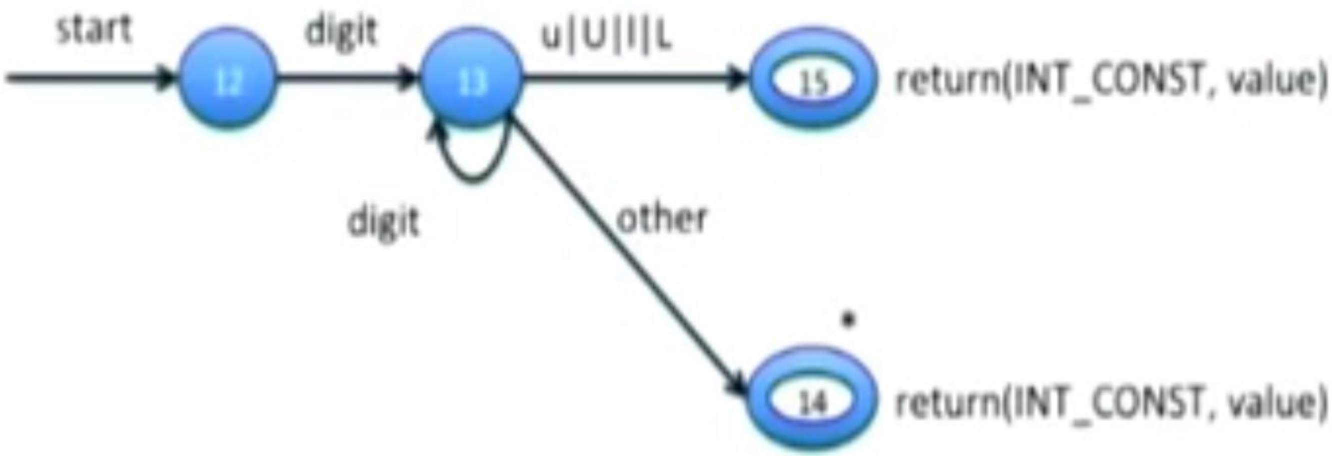
Transition Diagram for Hex and Oct Constants



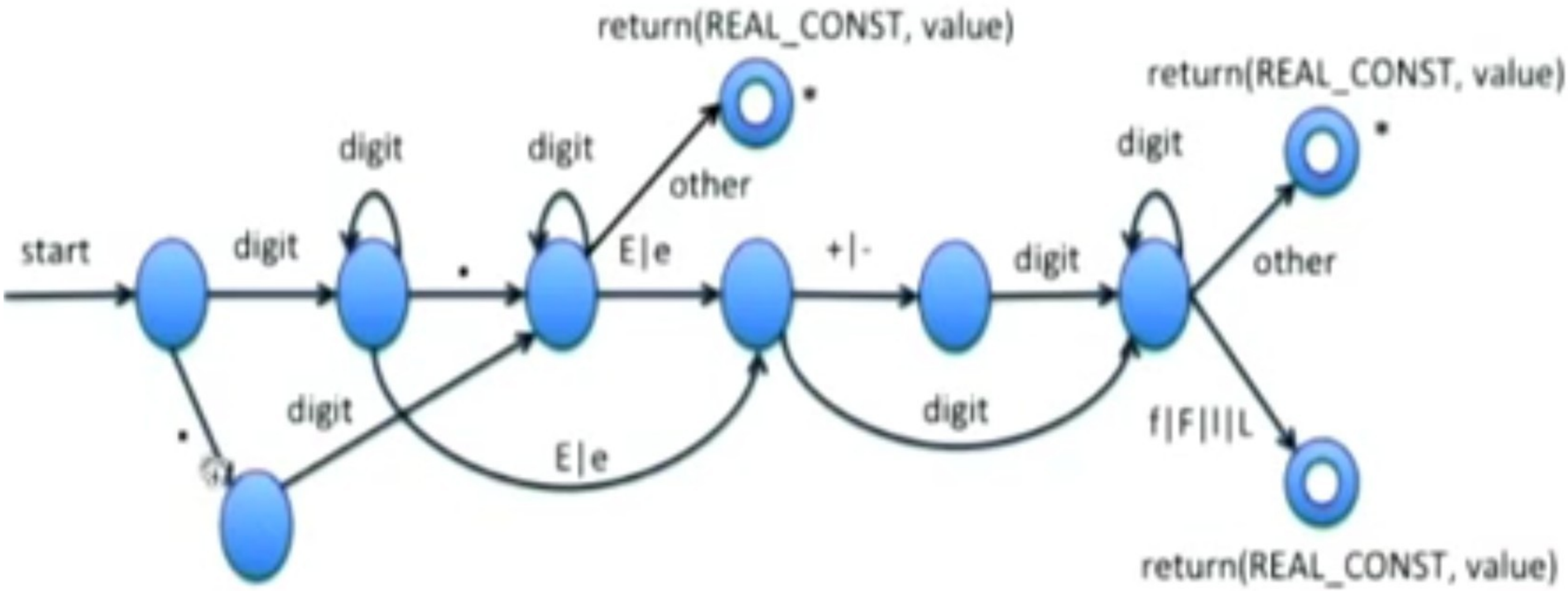
hex_const = 0 (x | X) dhex * (qualifier | ε)
oct_const = 0 doct * (qualifier | ε)
qualifier = u | U | l | L
dhex = [0-9A-F]
doct = [0-7]

Transition Diagram for Integer Constants

int_const = digit $^{+}$ (qualifier | ε)
qualifier = u | U | l | L
digit = [0-9]



Transition Diagram for Foat Constants



real_const = (digit$^+$ exponent (qualifier | ε)) |
(digit* "." digit$^+$ (exponent | ε) (qualifier | ε)) |
(digit$^+$ "." digit* (exponent | ε) (qualifier | ε))
exponent = (E|e)(+|-|ε) digit$^+$
qualifier = f | F | l | L
digit = [0-9]

Transition Diagrams for few Operators