COMP3173 Compiler Construction
Finite Automata

Dr. Zhiyuan Li

Outline

- DFA
- NFA and the Equivalence
- From Regular Expression to Automata
- DFA Minimization
- Implementation

- Read Dragon book Chapter 3.6 and 3.7

Purpose

- In the last lecture, we have introduced the regular expression/definition, which can be used to produce some strings in a regular language.
- But a lexer needs to do something vice versa. It takes an input and justifies whether it is a string (lexeme) in a regular language (token).
- Thus, we need to construct machines to recognize regular languages.
- The machines are called (deterministic/nondeterministic) finite automata.

Note: we will introduce some algorithms in this course, but we won’t prove the correctness of them. The proofs are sometimes difficult. Please accept the algorithms and apply them in the applications.

Deterministic Finite Automata

- A ***deterministic finite automata (DFA)*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where
  - Q is a finite set called ***states***,
  - Σ is a finite set called ***alphabet***,
  - $\delta: Q \times \Sigma \to Q$ is the ***transition function***,
  - $q_0 \in Q$ is the ***start state*** (also called the ***initial state***), and
  - $F \subseteq Q$ is the set of ***accept states*** (***final states***).
- For example, we define a DFA M by
  - $Q = \{q_0, q_1\}$,
  - $\Sigma = \{0,1\}$,
  - the transition function represented by a table,
  - q_0 is the start state, and
  - $\{q_1\}$ is the set of accept states.

δ	0	1
q_0	q_0	q_1
q_1	q_0	q_1

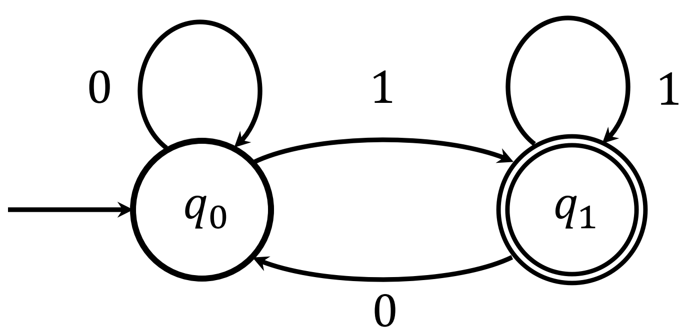
DFA

- A string $s = s_0 s_1 \cdots s_n$ is ***accepted*** by a DFA M if there is a sequence of states $q'_0, q'_1, \cdots, q'_{n+1}$ such that
  - each q'_i is a state of M,
  - q'_0 is the initial state q_0 of M,
  - q'_{n+1} is one of the final states of M, and
  - $\delta(q'_i, s_i) = q'_{i+1}$ for $0 \leq i \leq n$.
- The example on the previous page is an automata accepting the string 01011 because
  - assuming $s_0 = 0,\ s_1 = 1,\ s_2 = 0,\ s_3 = 1,\ s_4 = 1,$
  - $\delta(q_0, s_0) = \delta(q_0, 0) = q_0$
  - $\delta(q_0, s_1) = \delta(q_0, 1) = q_1$
  - $\delta(q_1, s_2) = \delta(q_1, 0) = q_0$
  - $\delta(q_0, s_3) = \delta(q_0, 1) = q_1$
  - $\delta(q_1, s_4) = \delta(q_1, 1) = q_1$
  - and q_1 is a final state.

δ	0	1
q_0	q_0	q_1
q_1	q_0	q_1

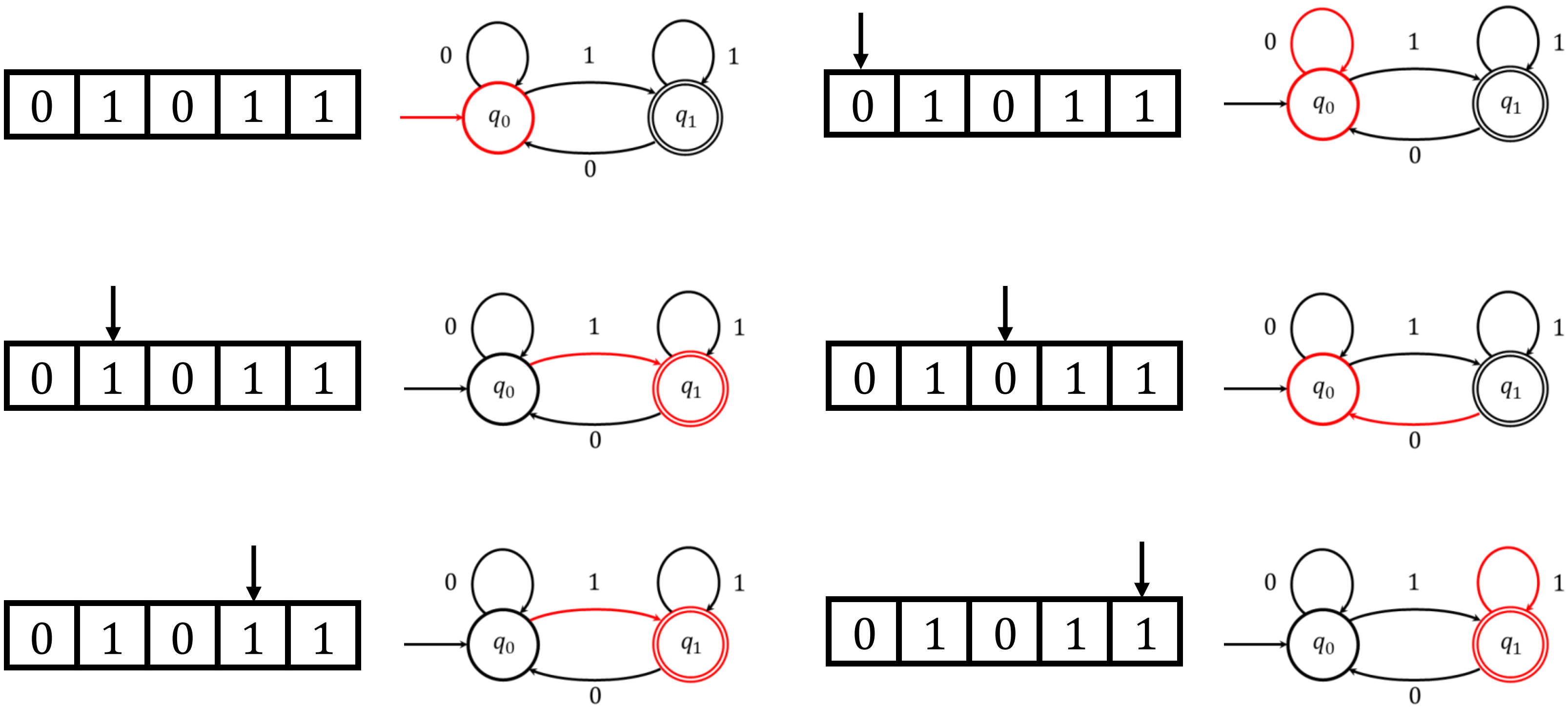
DFA

- The above definition and example seem to be abstract. We can present a DFA by a "weighted" directed graph $G = (V, E)$, called ***transition graph***, where
  - each state q is a vertex in the vertex set V;
  - if $\delta(q_i, a) = q_j$, then there is a directed edge from the vertex q_i to the vertex q_j with a as the "weight", meaning that "if the automata is at the state q_i and current input symbol is a, the automata moves to the state q_j";
  - the vertex for the start state q_0 is given an arrow pointing to it;
  - the vertex for a final state is double circled.
- The above example can be



DFA

- To verify that the automata accepts 01011.



DFA

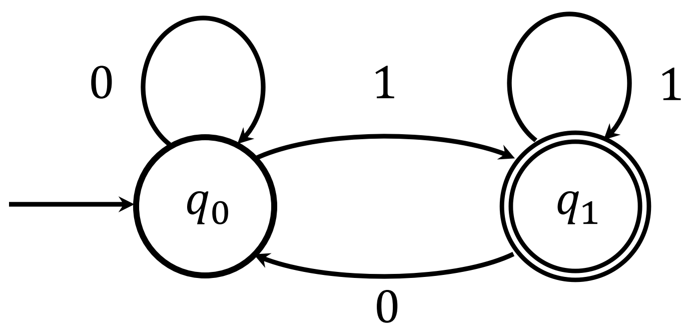
A string is not accepted by a DFA if:

- After reading all symbols in the string, the DFA ends at a non-final state.
  - For example, if we feed 0110 into the DFA above, it will stop at q_0 which is not a final state.
- Or at some point, the DFA cannot make any transition by reading the next symbol.
  - This can be caused by that the string containing a symbol which is not in the alphabet, for example $01a1$.

- If a string s is not accepted by a DFA M, we say M ***rejects*** s.

DFA

- A DFA M ***recognizes*** a language L if
  - M accepts all strings s in L, **and**
  - M rejects all strings s' not in L.
- We also say M is the ***recognizer*** of L.
- For example, the DFA defined above

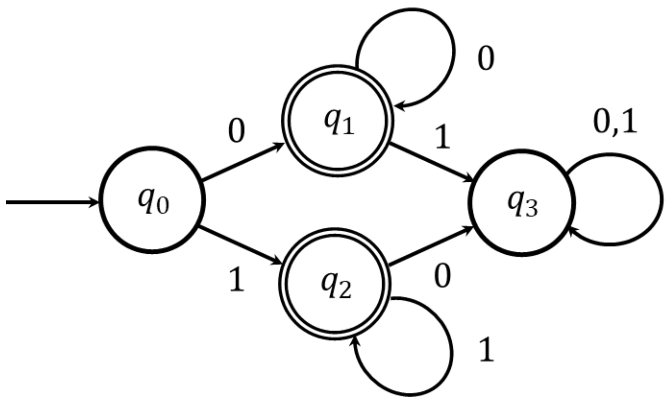


recognizes the language $L = \{w|w \text{ ends by } 1\}$.

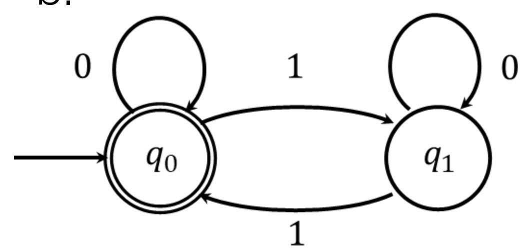
Exercise

- Describe the languages recognized by the following DFAs in English.

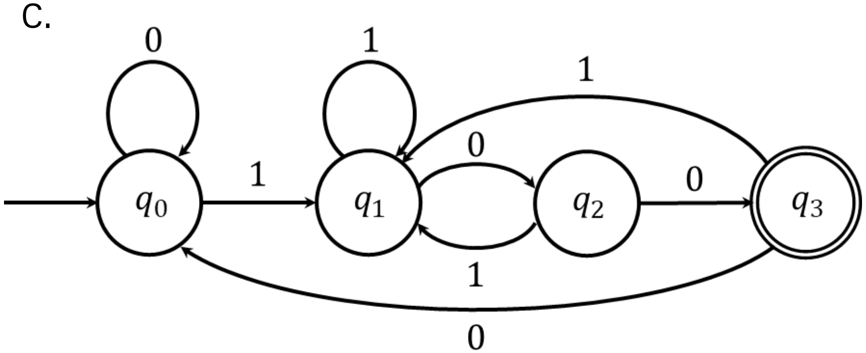
a.



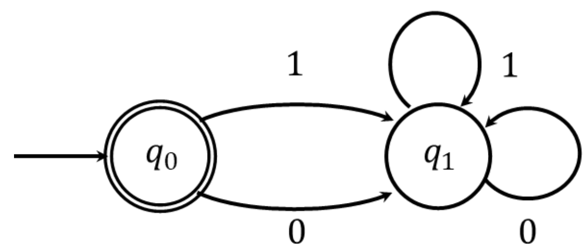
b.



c.

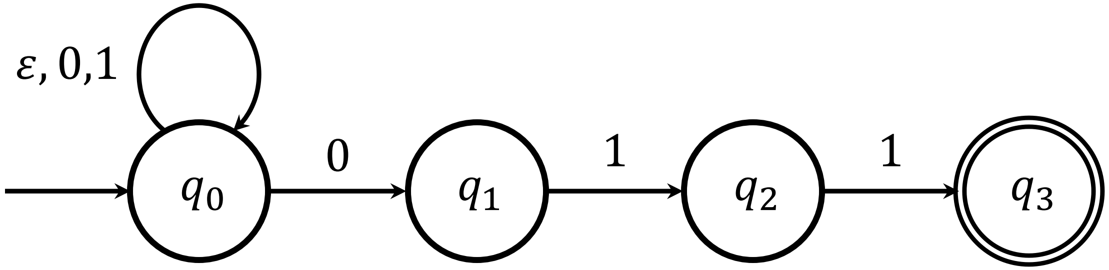


d.



NFA

- A ***nondeterministic*** finite automata is similar to a DFA but has no restrictions on the transitions. The transition is not a function, but a partial multi-valued relation, meaning that
  - the automata can move from one state to ***zero, one, or multiple*** states by taking one symbol from the input;
  - and the automata can make transitions without reading any symbol from the input. ε is considered as a symbol in the alphabet. This kind of transitions are called ***ε-transitions*** or ***ε-moves***.
  - Formally, the transition function for an NFA is $\delta: Q \times \Sigma \rightarrow \mathcal{P}(Q)$, where $\mathcal{P}(Q)$ is the powerset of Q.
- For example,



Equivalence

- Intuitively, NFAs are generalizations of DFAs because the transitions do not have restrictions. In other words, every DFA is an NFA.
- But are NFAs “more powerful” than DFAs?
- “more powerful” means that there are some languages can be recognized by NFAs but not DFA.
- Actually, **NFAs and DFAs are equivalent**. We prove this by presenting an algorithm to convert every NFA to a DFA which recognizes the same language.

Conversion

- Before giving the algorithm, we need some more definitions.
- $\boldsymbol{\varepsilon}$***-closure***$(\boldsymbol{q})$ is a set of NFA states which contains q itself and all reachable states from q on zero, one, or multiple ε-transitions alone. Formally,

$$\varepsilon\text{-closure}(q) = \{q\} \cup \delta(q, \varepsilon) \cup \delta(\delta(q, \varepsilon), \varepsilon) \cup \cdots$$

- $\boldsymbol{\varepsilon}$***-closure***$(\boldsymbol{T})$ is a set of NFA states which are reachable from the set of NFA states T on ε-transitions alone. Formally,

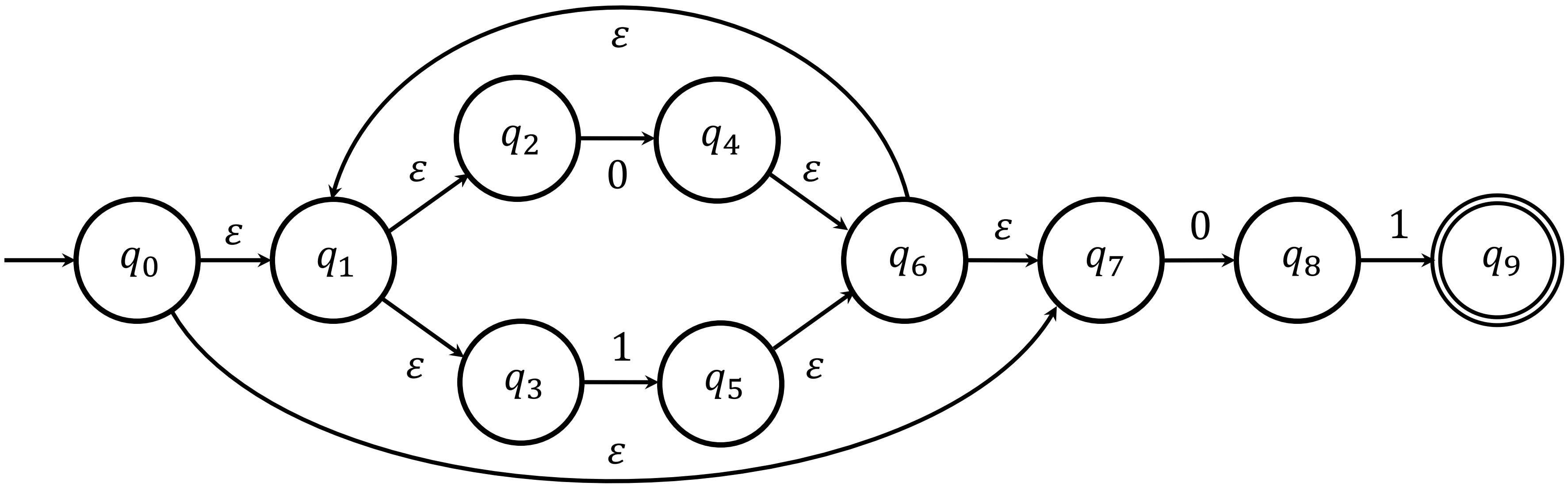
$$\varepsilon\text{-closure}(T) = \cup_{q \in T}\ \varepsilon\text{-closure}(q)$$

- $\boldsymbol{move(T, a)}$ is a set of NFA states to which there is a transition on input symbol a from some state q in T. Formally,

$$move(T, a) = \cup_{q \in T}\ \delta(q, a)$$

Conversion

- For example, given the following NFA and please find



- ε-closure(q_4)
- ε-closure(T) if $T = \{q_0, q_8\}$
- $move(\varepsilon\text{-closure}(q_4), 0)$

Conversion

Algorithm: Convert an NFA to a DFA

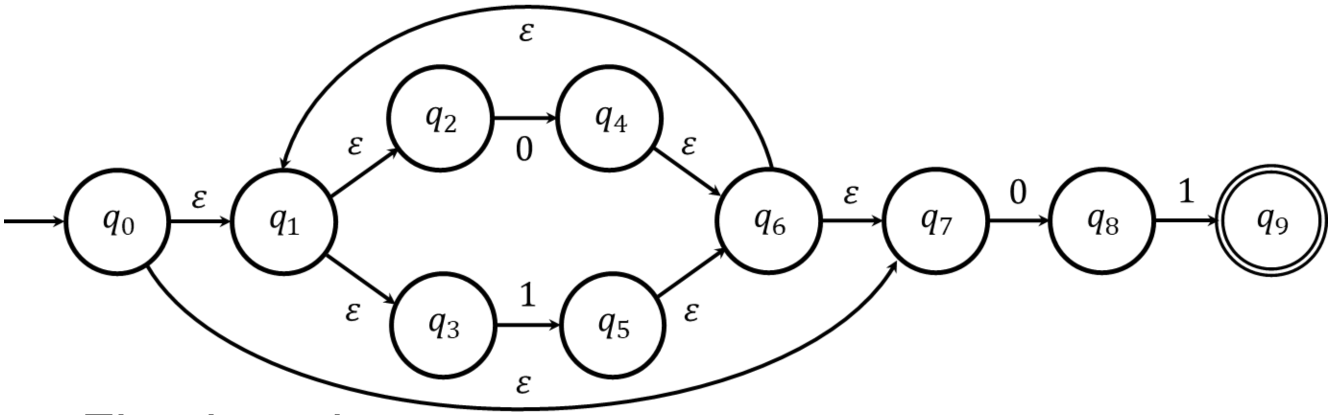
Input: an NFA $N = (Q, \Sigma, \delta, q_0, F)$

Output: a DFA $D = (Q', \Sigma, \delta', q_0', F')$

Initially, Q' has only one state ε-closure(q_0), and it is unmarked. Also, δ' is empty (consider the transition function as relation).

1. **for each** unmarked state T in Q'
2. Mark T
3. **for each** symbol $a \in \Sigma$
4. Let the state $U = \varepsilon$-closure$(move(T, a))$
5. **if** U is not in Q'
6. add U to Q'
7. **end if**
8. add $\delta'(T, a) = U$ to δ'
9. $q_0' = \varepsilon$-closure(q_0)
10. $F' = \{U | U$ has a final state in $F\}$

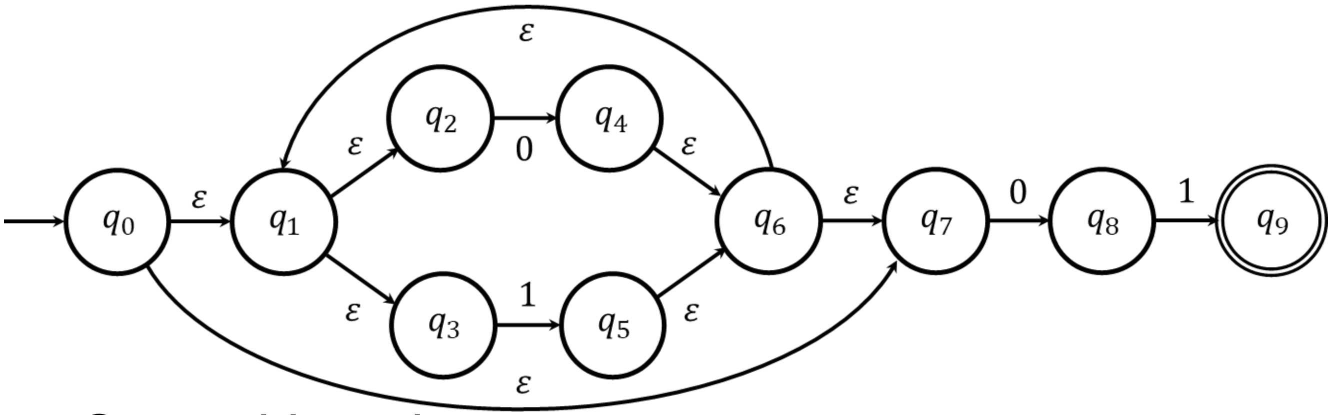
Conversion



- First iteration
  - $U_0 = \varepsilon\text{-closure}(q_0) = \{q_0, q_1, q_2, q_3, q_7\}$
  - $move(U_0, 0) = \{q_4, q_8\}$
  - $\varepsilon\text{-closure}\big(move(U_0, 0)\big)$
    $= \{q_4, q_6, q_1, q_2, q_3, q_7, q_8\} = U_1$
  - $move(U_0, 1) = \{q_5\}$
  - $\varepsilon\text{-closure}\big(move(U_0, 1)\big)$
    $= \{q_5, q_6, q_1, q_2, q_3, q_7\} = U_2$

Q	Q'	0	1
$\{q_0, q_1, q_2, q_3, q_7\}$	U_0	U_1	U_2
$\{q_4, q_6, q_1, q_2, q_3, q_7, q_8\}$	U_1		
$\{q_5, q_6, q_1, q_2, q_3, q_7\}$	U_2		

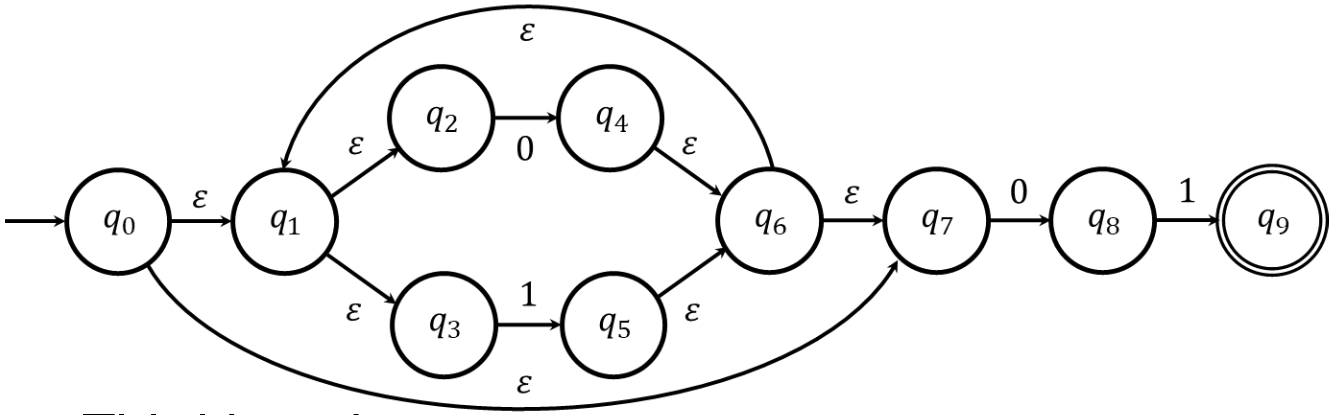
Conversion



- Second iteration
  - $move(U_1, 0) = \{q_4, q_8\}$
  - $\varepsilon\text{-closure}\big(move(U_1, 0)\big)$
    $= \{q_4, q_6, q_1, q_2, q_3, q_7, q_8\} = U_1$
  - $move(U_1, 1) = \{q_5, q_9\}$
  - $\varepsilon\text{-closure}\big(move(U_1, 1)\big)$
    $= \{q_5, q_6, q_1, q_2, q_3, q_7, q_9\} = U_3$

Q	Q'	0	1
$\{q_0, q_1, q_2, q_3, q_7\}$	U_0	U_1	U_2
$\{q_4, q_6, q_1, q_2, q_3, q_7, q_8\}$	U_1	U_1	U_3
$\{q_5, q_6, q_1, q_2, q_3, q_7\}$	U_2		
$\{q_5, q_6, q_1, q_2, q_3, q_7, q_9\}$	U_3		

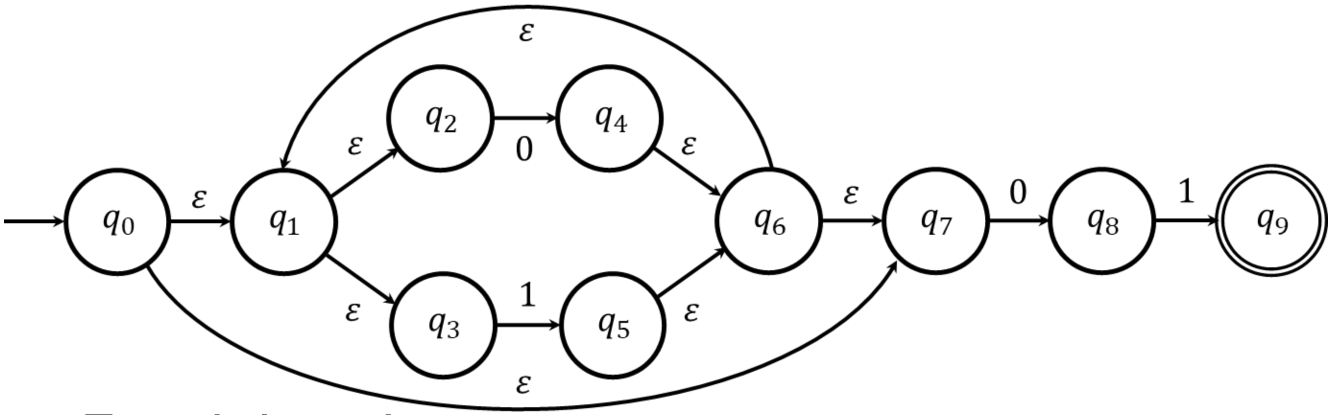
Conversion



- Third iteration
  - $move(U_2, 0) = \{q_4, q_8\}$
  - $\varepsilon\text{-closure}\big(move(U_2, 0)\big)$
    $= \{q_4, q_6, q_1, q_2, q_3, q_7, q_8\} = U_1$
  - $move(U_2, 1) = \{q_5\}$
  - $\varepsilon\text{-closure}\big(move(U_2, 1)\big)$
    $= \{q_5, q_6, q_1, q_2, q_3, q_7\} = U_2$

Q	Q'	0	1
$\{q_0, q_1, q_2, q_3, q_7\}$	U_0	U_1	U_2
$\{q_4, q_6, q_1, q_2, q_3, q_7, q_8\}$	U_1	U_1	U_3
$\{q_5, q_6, q_1, q_2, q_3, q_7\}$	U_2	U_1	U_2
$\{q_5, q_6, q_1, q_2, q_3, q_7, q_9\}$	U_3		

Conversion



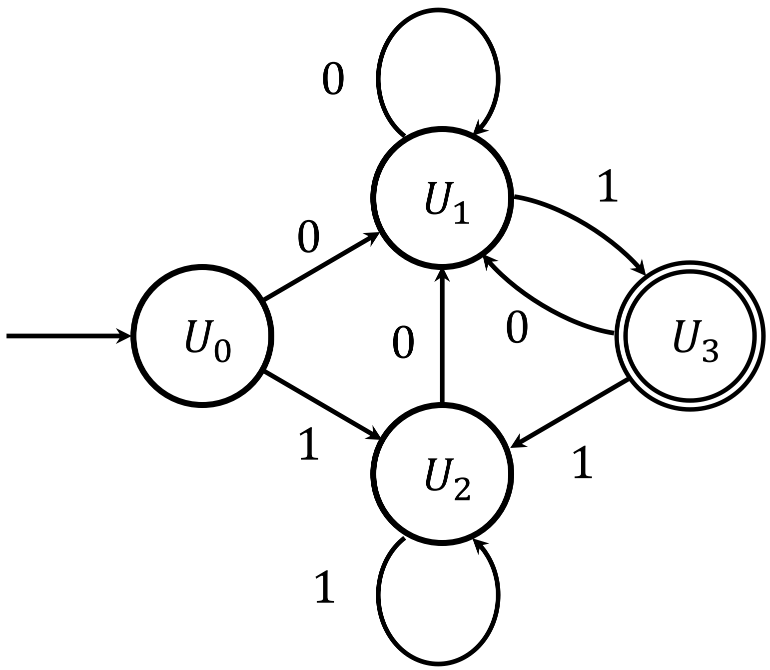
- Fourth iteration
  - $move(U_3, 0) = \{q_4, q_8\}$
  - $\varepsilon\text{-closure}\big(move(U_3, 0)\big)$
    $= \{q_4, q_6, q_1, q_2, q_3, q_7, q_8\} = U_1$
  - $move(U_3, 1) = \{q_5\}$
  - $\varepsilon\text{-closure}\big(move(U_3, 1)\big)$
    $= \{q_5, q_6, q_1, q_2, q_3, q_7\} = U_2$

Q	Q'	0	1
$\{q_0, q_1, q_2, q_3, q_7\}$	U_0	U_1	U_2
$\{q_4, q_6, q_1, q_2, q_3, q_7, q_8\}$	U_1	U_1	U_3
$\{q_5, q_6, q_1, q_2, q_3, q_7\}$	U_2	U_1	U_2
$\{q_5, q_6, q_1, q_2, q_3, q_7, q_9\}$	U_3	U_1	U_2

- There is no unmarked states in Q'. The algorithm halts.
- U_0 is the start state because it contains q_0. U_3 is the final state because it contains q_9.

Conversion

- Finally, we can draw the transition graph.



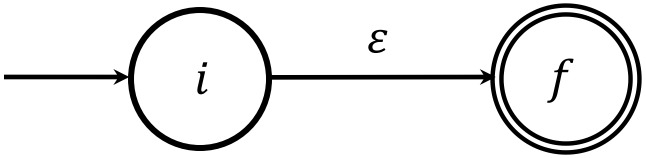
Q	Q'	0	1
$\{q_0, q_1, q_2, q_3, q_7\}$	U_0	U_1	U_2
$\{q_4, q_6, q_1, q_2, q_3, q_7, q_8\}$	U_1	U_1	U_3
$\{q_5, q_6, q_1, q_2, q_3, q_7\}$	U_2	U_1	U_2
$\{q_5, q_6, q_1, q_2, q_3, q_7, q_9\}$	U_3	U_1	U_2

From Regular Expression to NFA

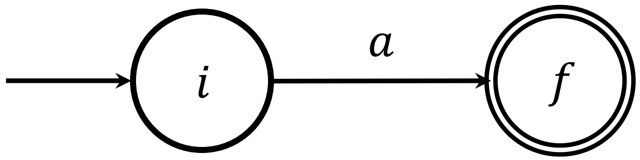
- Recall that languages, grammars, and machines are all equivalent.
- So, for every regular expression r, we can have an algorithm taking r as input and returning an NFA to recognize the language $L(r)$.
- The algorithm is done recursively.

From Regular Expression to NFA

- Base case:
  1. For expression ε construct the NFA



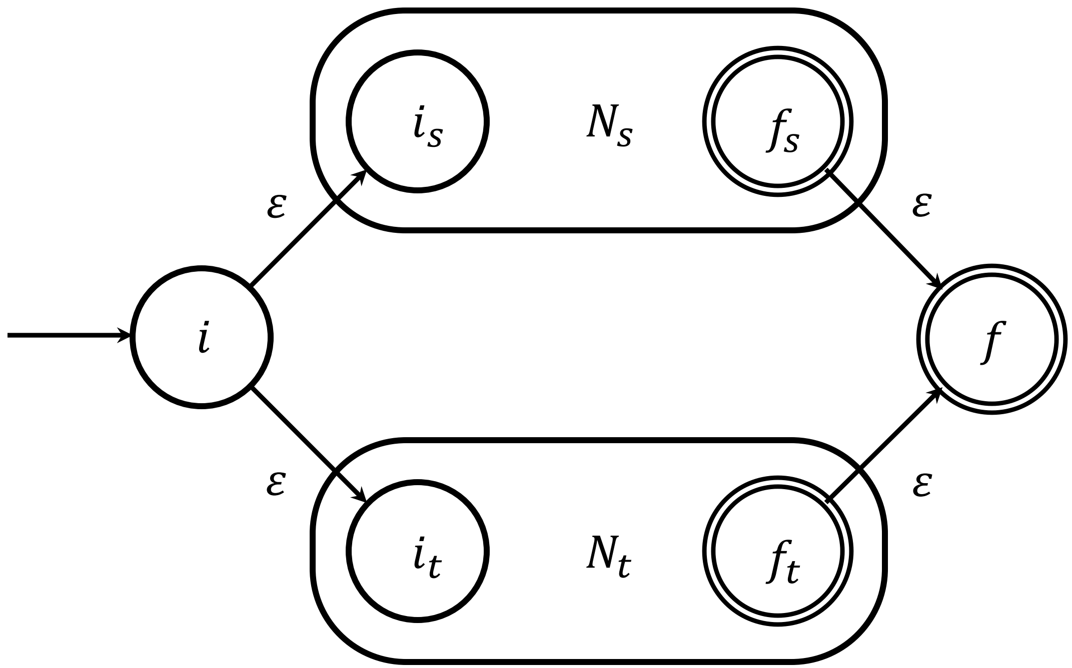
  2. For any subexpression a in Σ, construct the NFA



  - Here i is a new state, the start state of this NFA, and f is another new state, the accepting state for the NFA.
- These are the base cases because if you want to accept one symbol, the NFA only needs one transition.

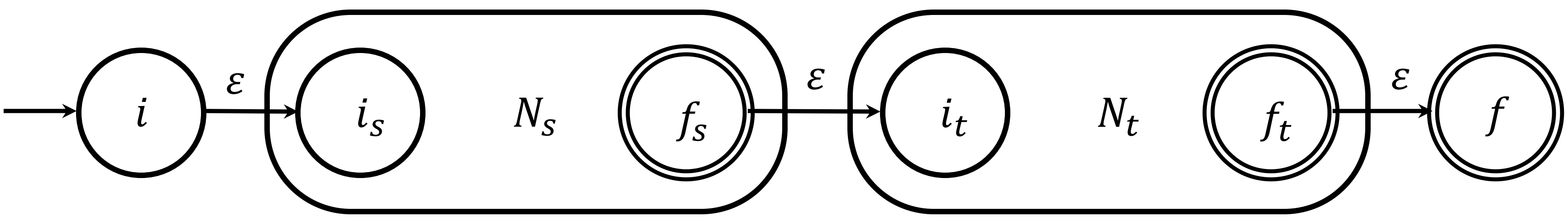
From Regular Expression to NFA

- Recursion: let N_s and N_t be NFA's for regular expressions s and t respectively.
  1. If $r = s|t$, the NFA for r is



From Regular Expression to NFA

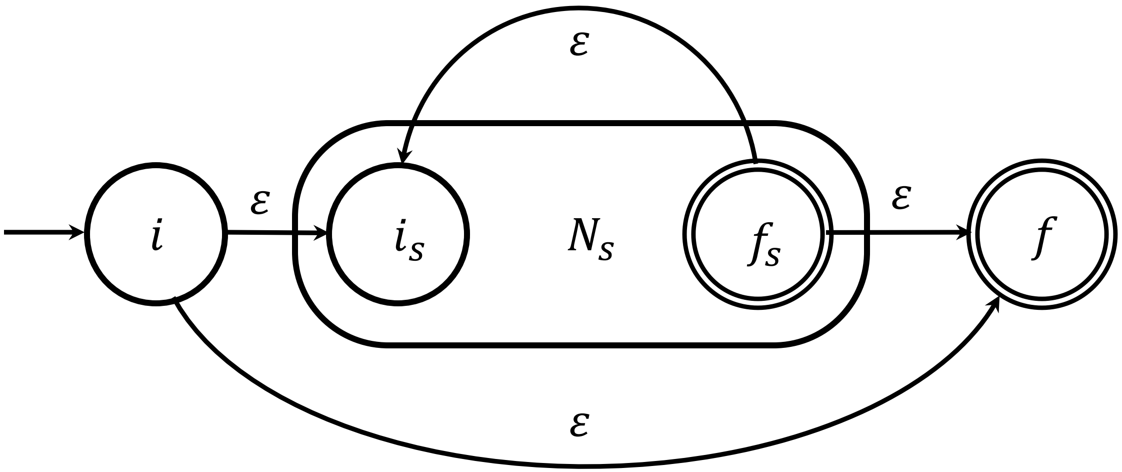
- Recursion: let N_s and N_t be NFA's for regular expressions s and t respectively.
  2. If $r = st$, the NFA for r is



From Regular Expression to NFA

- Recursion: let N_s be the NFA for the regular expressions s.

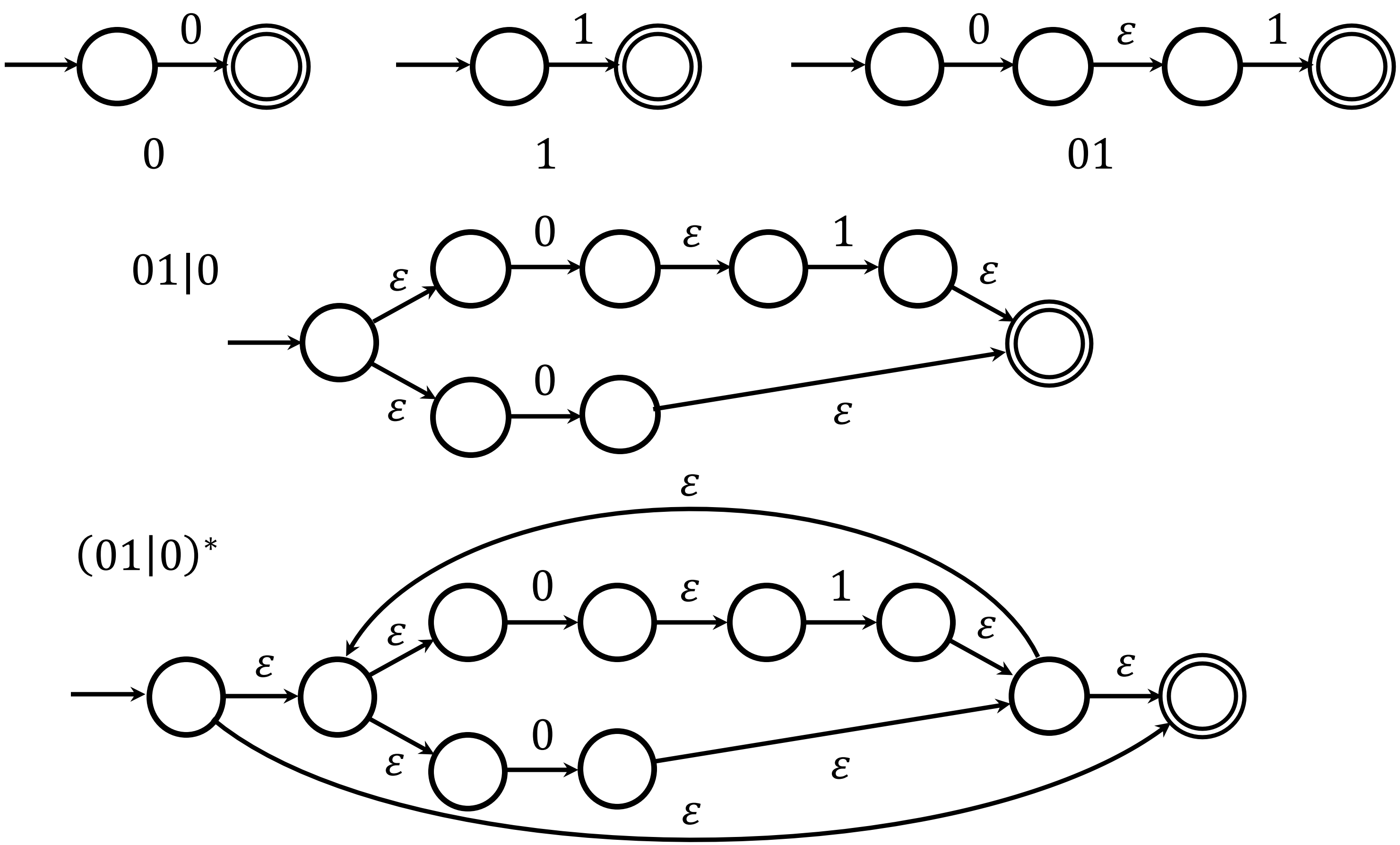
3. If $r = (s)^*$, the NFA for r is



4. If $r = (s)$, then N_s is the NFA for r.

From Regular Expression to NFA

- Example: construct an NFA for $(01|0)^*$



Exercises

- Construct an NFA for each of the regular expression. Then, convert NFAs to DFAs
  - $(0|1)^*010$
  - $1(01|10)^*1$
  - $(0^*|10)^*1$

DFA Minimization

- From the above examples/exercises, sometimes the construction results a complicated DFA. For application purposes, we need to find the simplest DFAs.
- The ***minimum state DFA*** D_m for the DFA D recognizes the same language but use the minimum number of states.
- In other words, among all DFAs recognize the same language as D, there is no DFA uses smaller number of states than D_m.

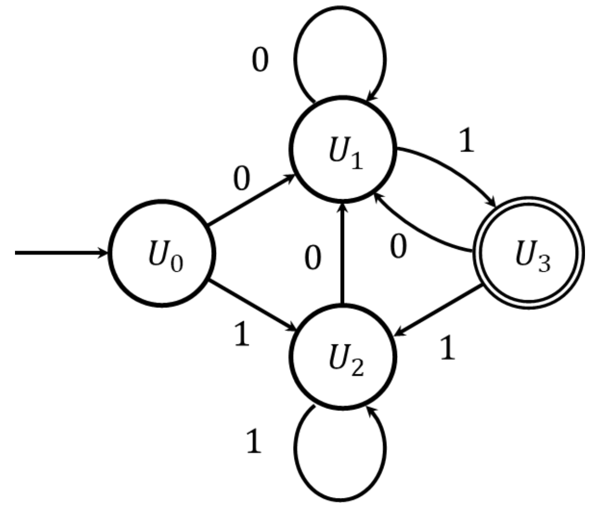
DFA Minimization

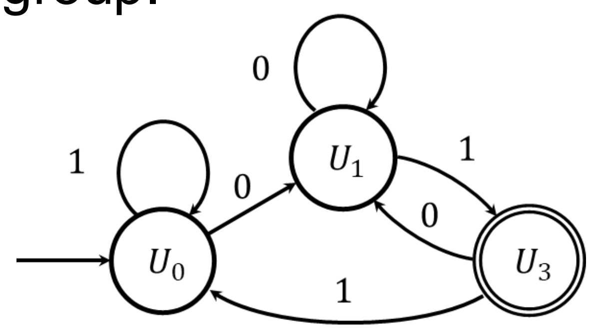
The sketch of the algorithm is as follow.

1. Initially, let $P = P' = \{F, Q \setminus F\}$ be a partition of the states.
2. For each group of states $Y \in P'$ (which is an equivalent class), split Y into two subsets under the condition:
3. Two states $q, q' \in Y$ are in the same subset if and only if for every symbol $c \in \Sigma$, states q and q' have transition on to the states in the same group of P.
4. Assign P' to P.
5. Repeat 2 and 3 until P is not changed.
6. Each group in P is a state for D_m.
7. The group which has the initial state is the initial state.
8. The groups which have a final state is a final state.

Example

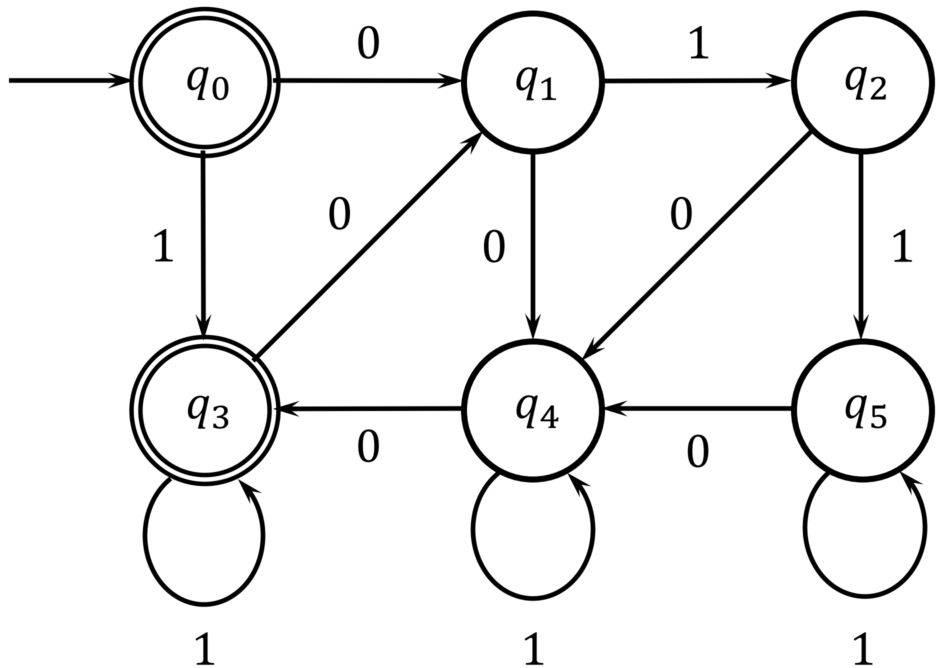
- Initially, $Q = \{\{U_3\}, \{U_0, U_1, U_2\}\}$
- We can try to split $\{U_0, U_1, U_2\}$.
- $\delta(U_1, 0) = U_1$, $\delta(U_1, 1) = U_3$
- $\delta(U_0, 0) = U_1$, $\delta(U_0, 1) = U_2$
  $\delta(U_2, 0) = U_1$, $\delta(U_2, 1) = U_2$
- U_0 and U_2 act in the same way
- Different from U_3
- Thus, U_0 and U_2 should be in the same group.
- After minimization,





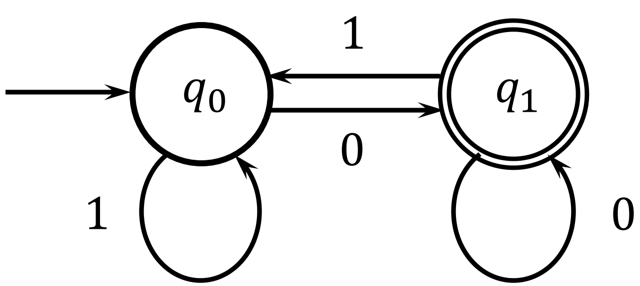
Exercise

- Minimize the following DFA



Implementation

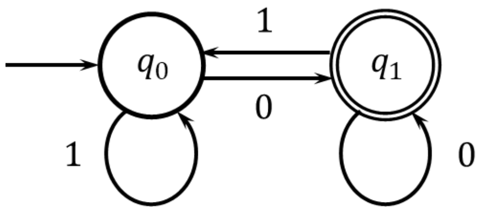
- The implementation of a lexer using a DFA is trivial.
- Using “`while`”, “`switch`”, and “`if`” is enough. Also remember to return the tokens and error messages.
- Consider the strings ended with “0”.



Implementation

- Initialize the initial state and a character buffer.

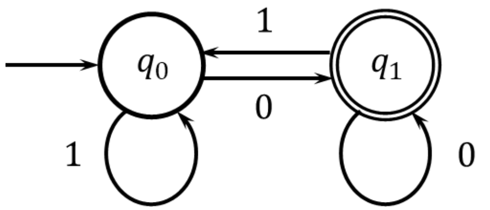
```
int state = 0;
char c;
```



Implementation

- Create a loop to read one character per round.

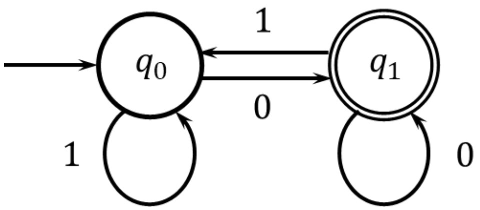
```
int state = 0;
char c;
while(1){
    c = nextchar();








}
```



Implementation

- Use switch to check the current state.

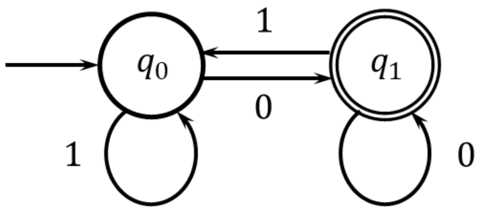
```
int state = 0;
char c;
while(1){
    c = nextchar();
    switch(state){








    }
}
```



Implementation

- Make transition by the current state and the input character.

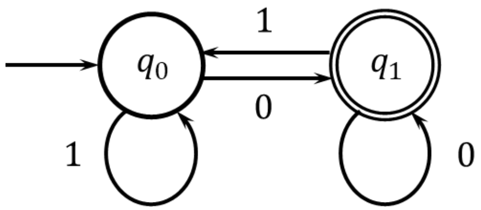
```
int state = 0;
char c;
while(1){
    c = nextchar();
    switch(state){
        case 0:
            if(c=='0')      state = 1;
            else if(c=='1') state = 0;

            break;
        case 1:
            if(c=='0')      state = 1;
            else if(c=='1') state = 0;




            break;
    }
}
```



Implementation

- Report error, if the next symbol is neither “0” nor “1” at q_0.

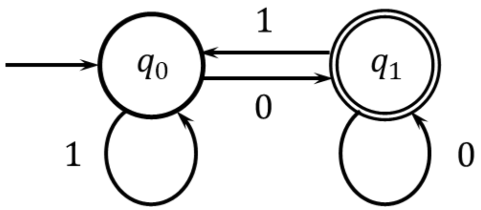
```
int state = 0;
char c;
while(1){
    c = nextchar();
    switch(state){
        case 0:
            if(c=='0')      state = 1;
            else if(c=='1') state = 0;
            else            error_recovery();
            break;
        case 1:
            if(c=='0')      state = 1;
            else if(c=='1') state = 0;




            break;
    }
}
```



Implementation

- Return the token, if the next symbol is neither "0" nor "1" at q_1.

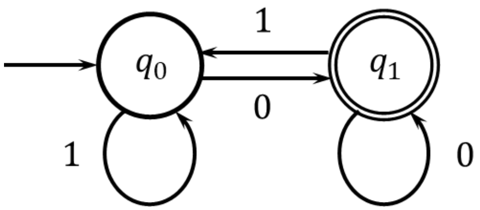
```
int state = 0;
char c;
while(1){
    c = nextchar();
    switch(state){
        case 0:
            if(c=='0')      state = 1;
            else if(c=='1') state = 0;
            else            error_recovery();
            break;
        case 1:
            if(c=='0')      state = 1;
            else if(c=='1') state = 0;
            else{

                            return some_token;
            }
            break;
    }
}
```



Implementation

- Resume the symbol because the language may have more characters (other than "0" or "1") and more tokens.

```
int state = 0;
char c;
while(1){
    c = nextchar();
    switch(state){
        case 0:
            if(c=='0')       state = 1;
            else if(c=='1') state = 0;
            else             error_recovery();
            break;
        case 1:
            if(c=='0')       state = 1;
            else if(c=='1') state = 0;
            else{
                    resume_last_input(c);
                    return some_token;
            }
            break;
    }
}
```



The End of Lecture 3