CMPT 413
Computational Linguistics

Anoop Sarkar

http://www.cs.sfu.ca/~anoop

Formal Languages: Recap

- Symbols: a, b, c
- Alphabet : finite set of symbols $\Sigma = \{a, b\}$
- String: sequence of symbols bab
- Empty string: ε Define: $\Sigma^{\varepsilon} = \Sigma \cup \{\varepsilon\}$
- Set of all strings: Σ^* cf. *The Library of Babel*, Jorge Luis Borges
- (Formal) Language: a set of strings

  $\{ a^n b^n : n > 0 \}$

Regular Languages

- The set of regular languages: each element is a regular language
- Each regular language is an example of a (formal) language, i.e. a set of strings

  e.g. { a^m b^n : m, n are +ve integers }

Regular Languages

- Defining the set of all regular languages:
  - The empty set and $\{a\}$ for all a in Σ^{ε} are regular languages
  - If L_1 and L_2 and L are regular languages, then:

    $L_1 \cdot L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$ (concatenation)

    $L_1 \cup L_2$ (union)

    $L^* = \cup_{i=0}^{\infty} L^i$ (Kleene closure)

    are also regular languages
  - There are no other regular languages

Formal Grammars

- A formal grammar is a concise description of a formal language
- A formal grammar uses a specialized syntax
- For example, a **regular expression** is a concise description of a regular language

  $(a|b)^*abb$: is the set of all strings over the alphabet $\{a, b\}$ which end in *abb*

Regular Expressions: Definition

- Every symbol of $\Sigma \cup \{ \varepsilon \}$ is a regular expression
- If r_1 and r_2 are regular expressions, so are
  - Concatenation: $r_1 r_2$
  - Alternation: $r_1|r_2$
  - Repetition: r_1*
- Nothing else is.
  - Grouping re's: e.g. aa|bc vs. ((aa)|b)c

Regular Expressions: Examples

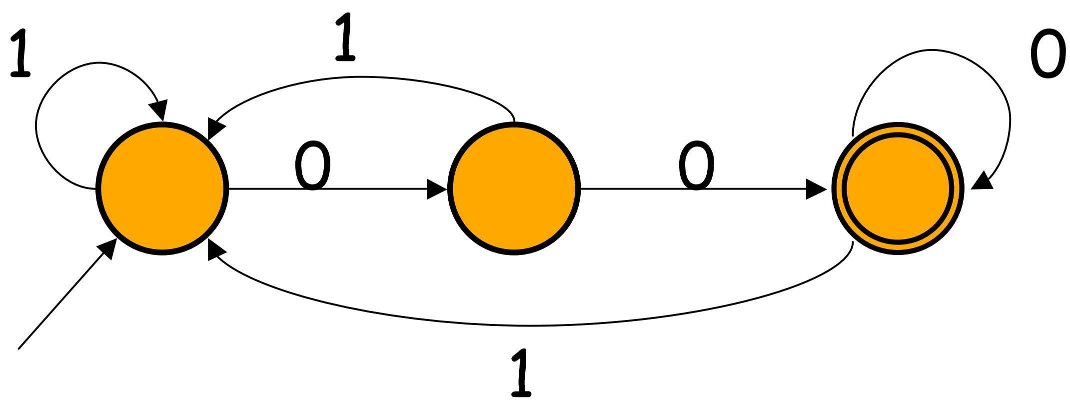
- Alphabet { V, C } V: vowel C: consonant
- A set of consonant-vowel sequences (CV|CCV)*
- All strings that do not contain "VC" as a substring C*V*
- Need a decision procedure: does a particular regular expression (regexp) accept an input string
- Provided by: Finite State Automata

Finite Automata: Recap

- A set of states S
  - One start state q_0, zero or more final states F
- An alphabet $\sum$ of input symbols
- A transition function:
  - $\delta: S \times \Sigma \Rightarrow S$
- Example: $\delta(1, a) = 2$

Finite Automata: Example

- What regular expression does this automaton accept?



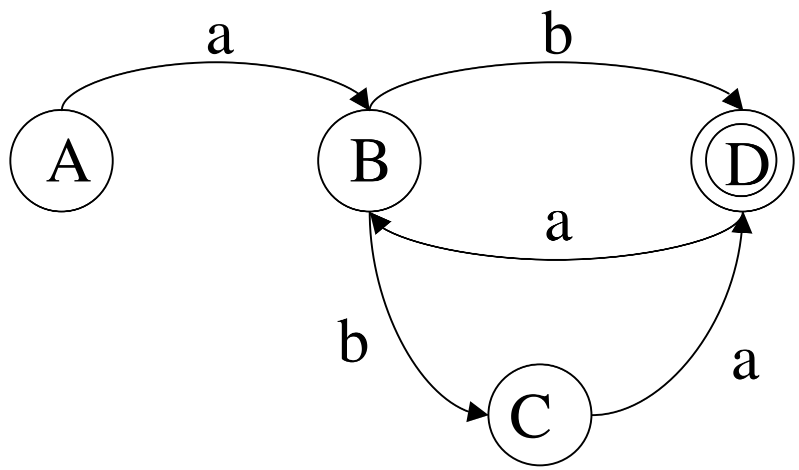
Answer: (0|1)*00

NFAs

- NFA: like a DFA, except
  - A transition can lead to more than one state, that is, $\delta: S \times \Sigma \Rightarrow 2^S$
  - One state is chosen non-deterministically
  - Transitions can be labeled with ε, meaning states can be reached without reading any input, that is,

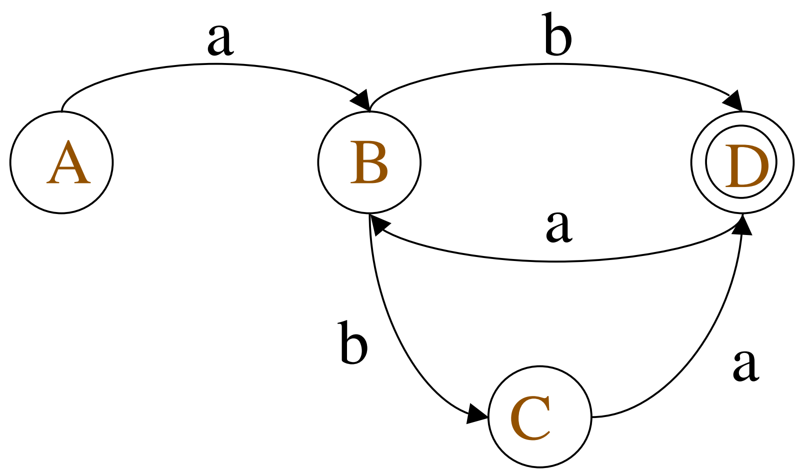
$$\delta: S \times \Sigma \cup \{ \varepsilon \} \Rightarrow 2^S$$

Recognition of strings (NFAs)



- Input string: aba#
- Recognition problem: Is input string in the language generated by the NFA?
- Recognition (without conversion to DFA) is also called *simulation* of NFA

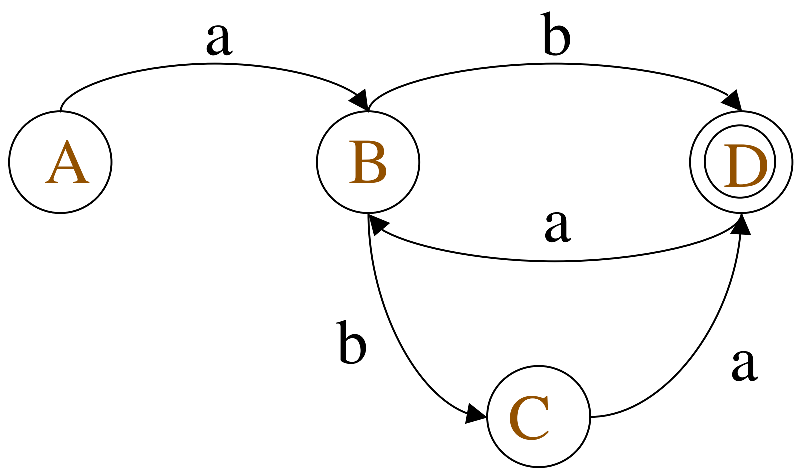
Recognition of strings (NFAs)



q is the transition function for the NFA

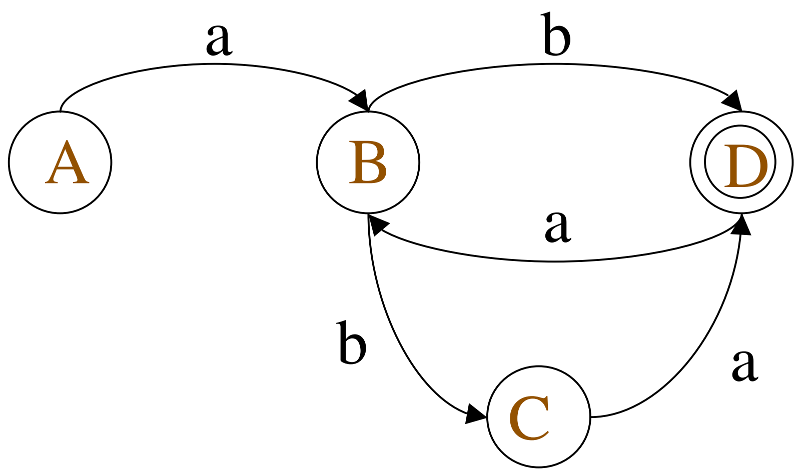
- Input tape: $_0$ a $_1$ b $_2$ a $_3$ # $_4$
- Start State: A Agenda: { (A, 0) }
- Pop (A, 0) from Agenda
- q(A, a) = B, Agenda: { (B, 1) }
- Pop (B, 1) from Agenda
- q(B, b) = { D, C } Agenda: { (D, 2), (C, 2) }

Recognition of strings (NFAs)



- Input tape: $_0$ a $_1$ b $_2$ a $_3$ # $_4$
- Pop (D, 2) from Agenda
- q(D, a) = { B }   Agenda: { (B, 3), (C, 2) }
- Pop (B, 3) from Agenda: B is not a final state
- Pop (C, 2) from Agenda: if Agenda empty, reject
- q(C, a) = { D }   Agenda: { (D, 3) }

Recognition of strings (NFAs)



- Input tape: $_0$ a $_1$ b $_2$ a $_3$ # $_4$
- Pop (D, 3) from Agenda
- Is (D, 3) an accept item?
- Yes: D is a final state **and** 3 is index of the end-of-string marker #
- Return accept

Recognition of strings (NFAs)

```
function NDRecognize (tape[], q):
    Agenda = { (start-state, 0) }
    Current = (state, index) = pop(Agenda)
    while (true) {
        if (Current is an accept item) return accept
        else Agenda = Agenda ∪ GenStates(q, state, tape[index])
        if (Agenda is empty) return reject
        else Current = (state, index) = pop(Agenda)
    }
function GenStates (q, state, index):
    return { (q', index) : for all q' = q(state, ε) } ∪
           { (q', index+1) : for all q' = q(state, tape[index+1]) }
```

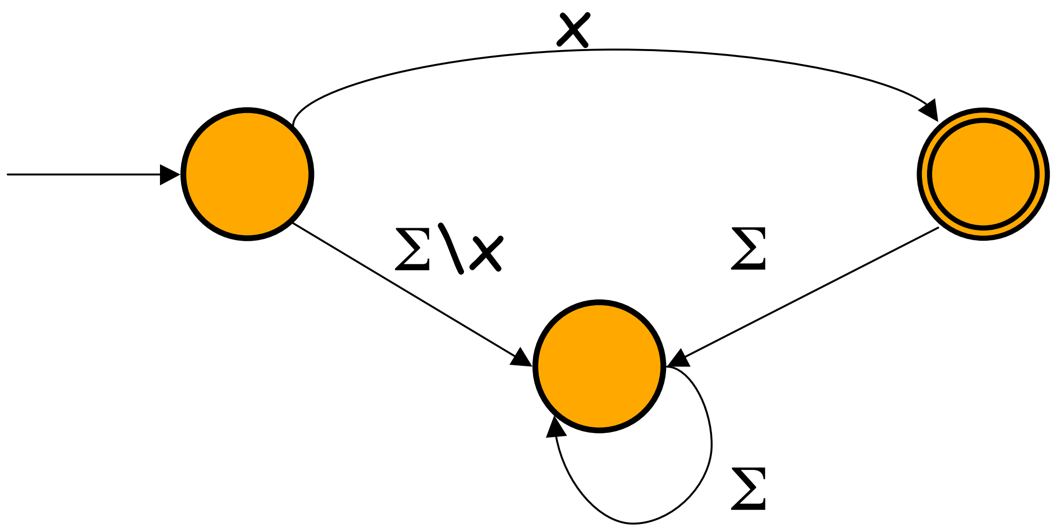
what if the input to this algorithm is a DFA?

Thompson's construction

- Converts regexps to NFA
- Five simple rules
  - Symbols
  - Empty String
  - Alternation (r_1 or r_2)
  - Concatenation (r_1 followed by r_2)
  - Repetition (r_1*)

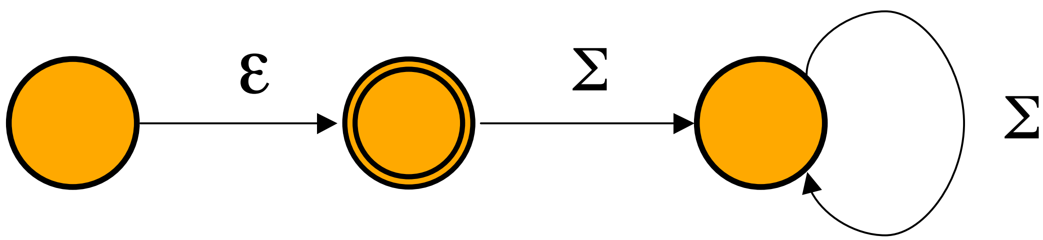
Thompson Rule 1

- For each symbol *x* of the alphabet, there is a NFA that accepts it (include a *sinkhole* state)



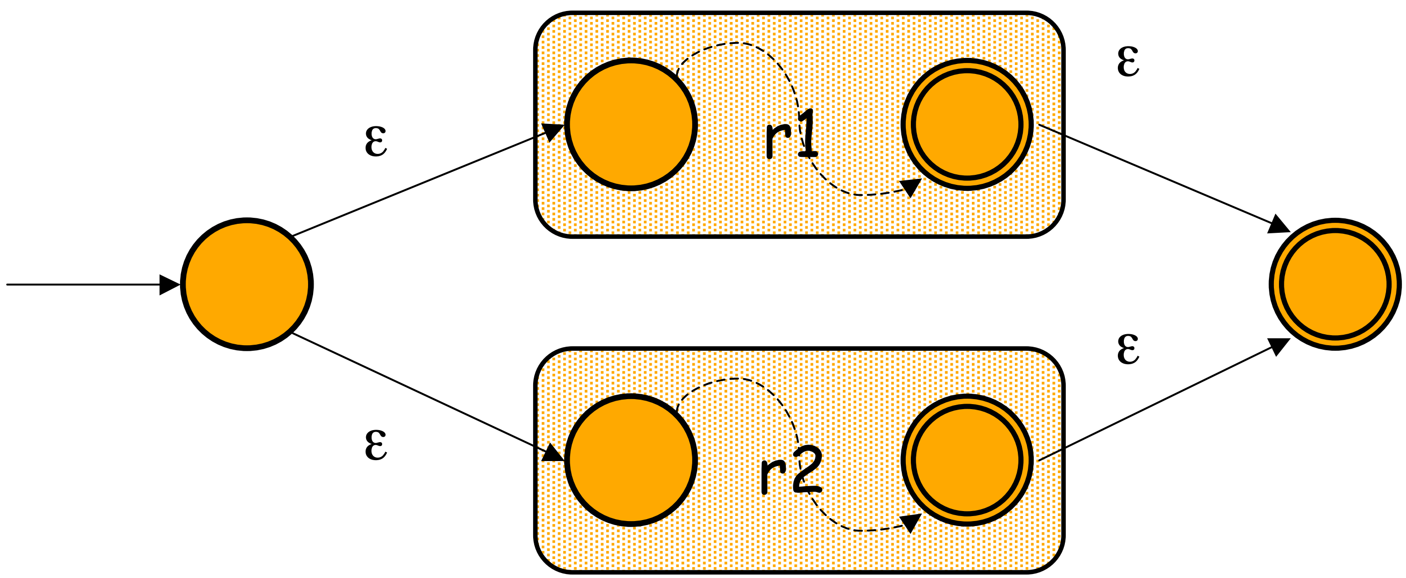
Thompson Rule 2

- There is an NFA that accepts only ε



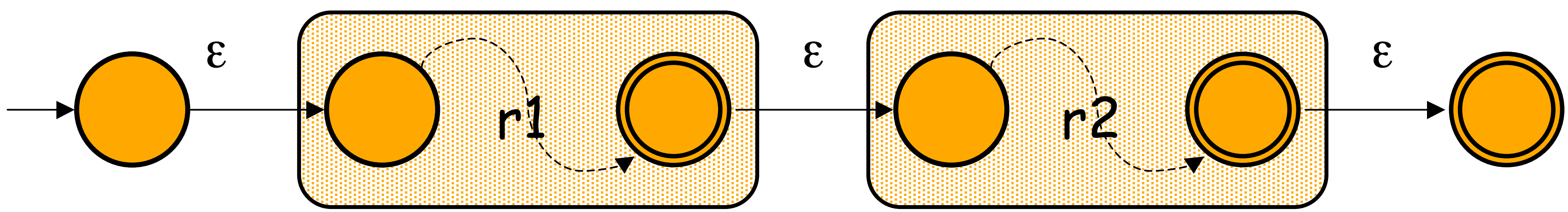
Thompson Rule 3

- Given two NFAs for r_1, r_2, there is a NFA that accepts $r_1|r_2$



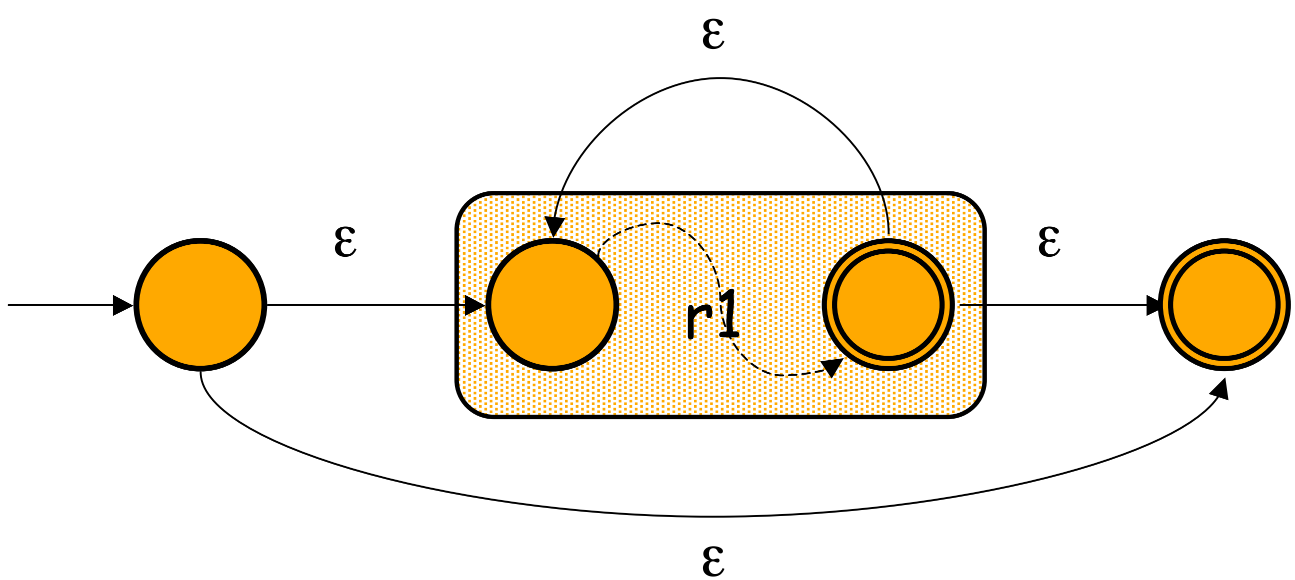
Thompson Rule 4

- Given two NFAs for r_1, r_2, there is a NFA that accepts $r_1 r_2$



Thompson Rule 5

- Given a NFA for r_1, there is an NFA that accepts $r_1{}^*$

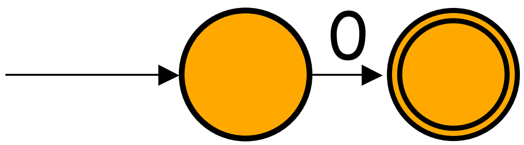


Example

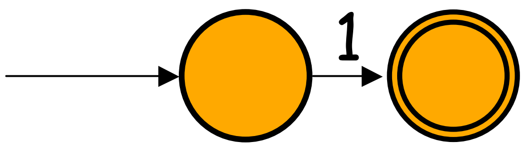
- Set of all binary strings that are divisible by four (include 0 in this set)
- Defined by the regexp: ((0|1)*00) | 0
- Apply Thompson's Rules to create an NFA

Basic Blocks 0 and 1

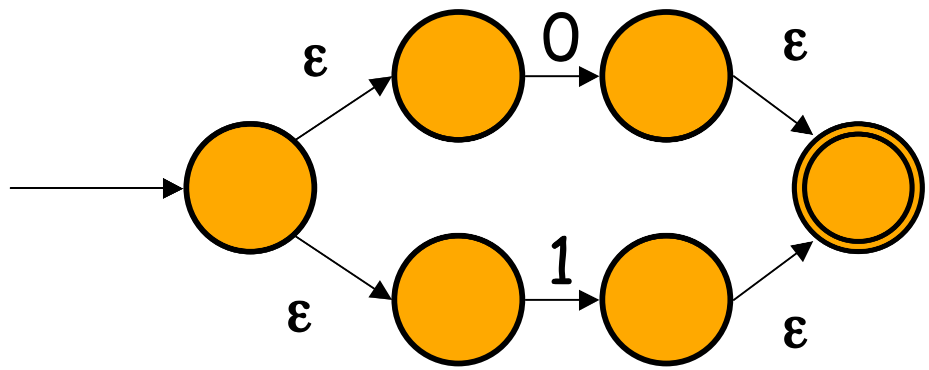
- 0



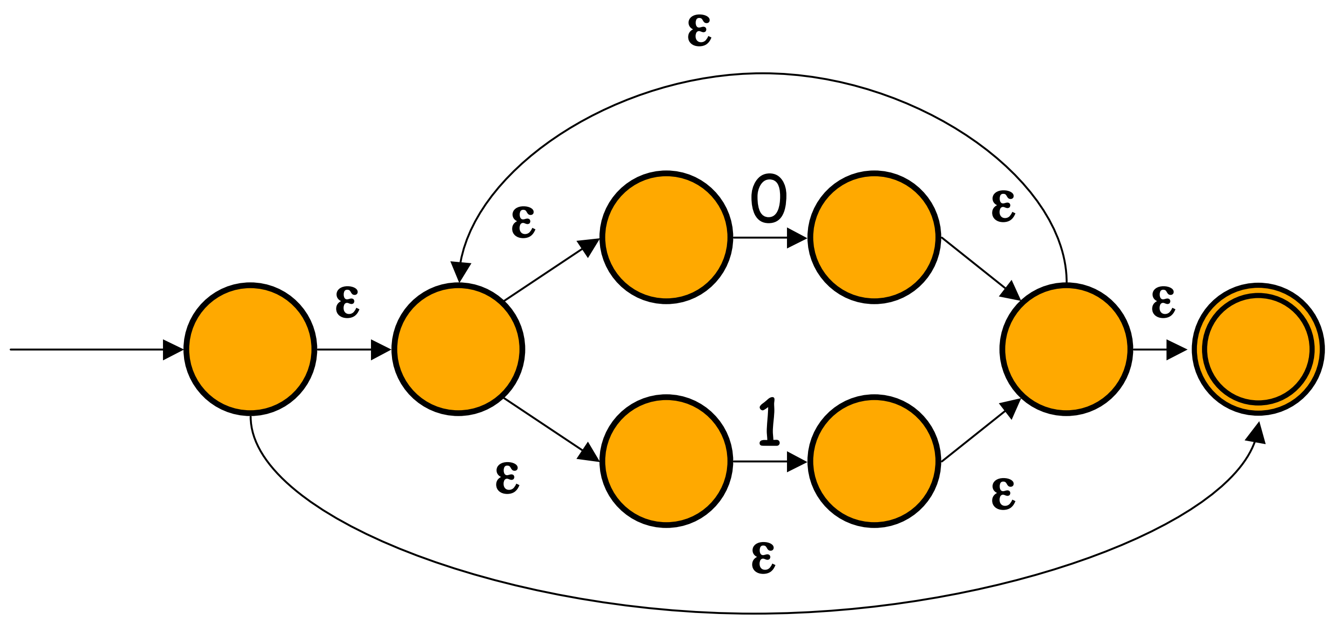
- 1



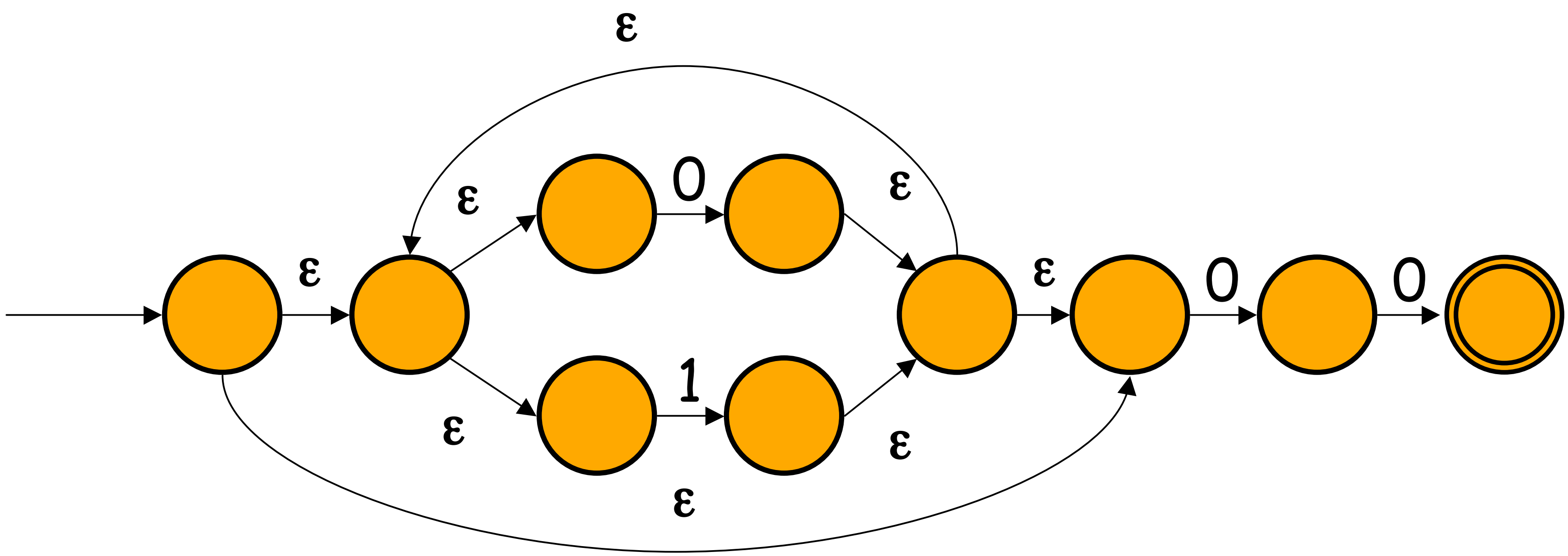
(this version does not report errors: no *sinkholes*)



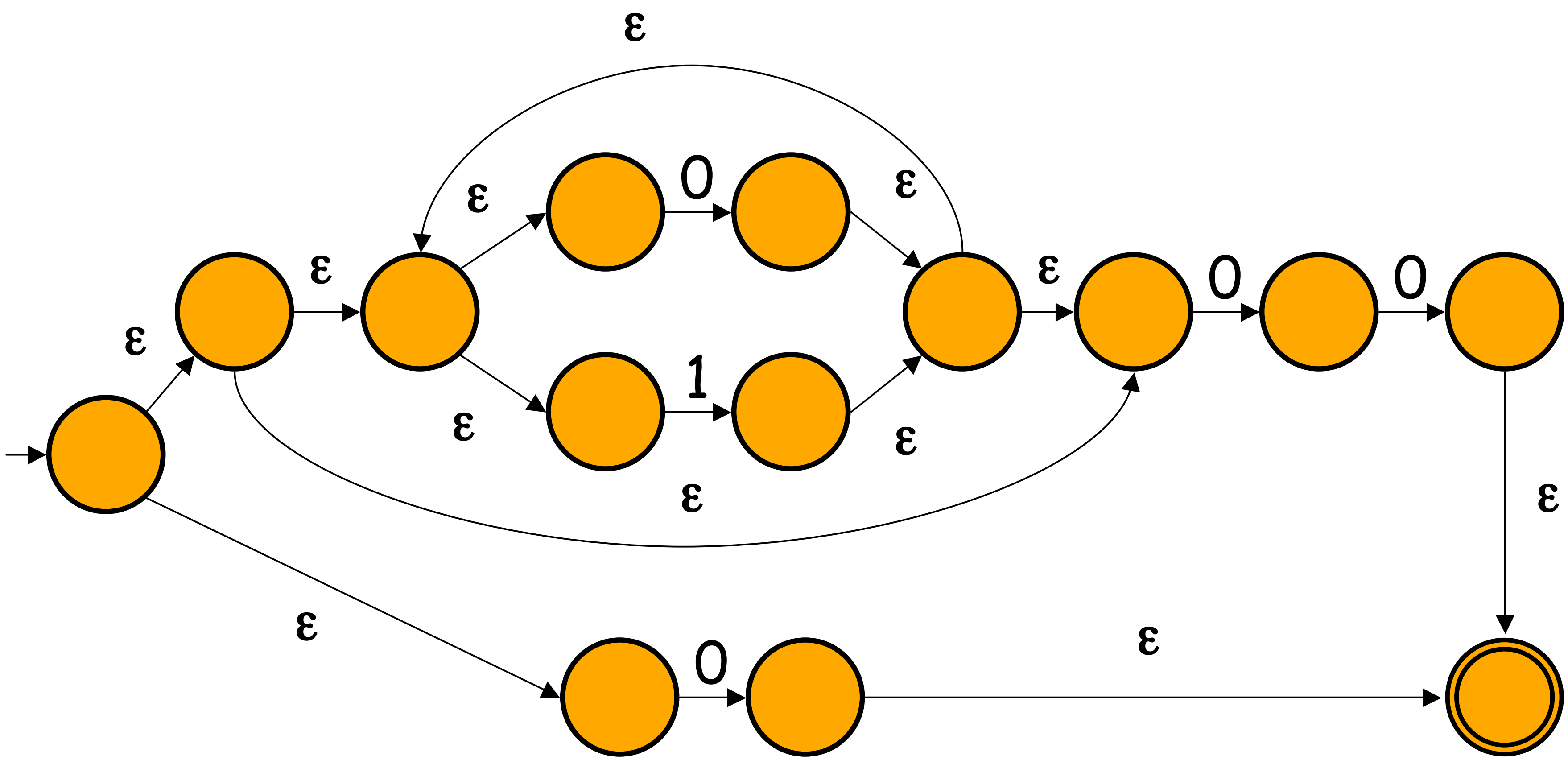
0|1



(0|1)*



(0|1)*00



((0|1)*00)|0

NFA to DFA Conversion

- Subset construction
- Idea: subsets of set of all NFA states are *equivalent* and become one DFA state
- Algorithm simulates movement through NFA
- Key problem: how to treat ε-transitions?

ε-Closure

- Start state: q_0
- ε-closure(S): S is a set of states

$$\begin{aligned}
&\textbf{initialize: } S \leftarrow \{q_0\} \\
&T \leftarrow S \\
&\textbf{repeat } T' \leftarrow T \\
&\quad T \leftarrow T' \cup [\cup_{s \in T'} \textbf{move}(s, \epsilon)] \\
&\textbf{until } T = T'
\end{aligned}$$

ε-Closure (T: set of states)

```
push all states in T onto stack
initialize ε-closure(T) to T
while stack is not empty do begin
    pop t off stack
    for each state u with u ∈ move(t, ε) do
        if u ∉ ε-closure(T) do begin
            add u to ε-closure(T)
            push u onto stack
        end
end
```

Conversion from NFA to DFA

- Conversion method closely follows the NFA recognition algorithm
- Instead of simulating, we can collect those NFA states that behave identically on the same input
- Group this set of states to form one state in the DFA

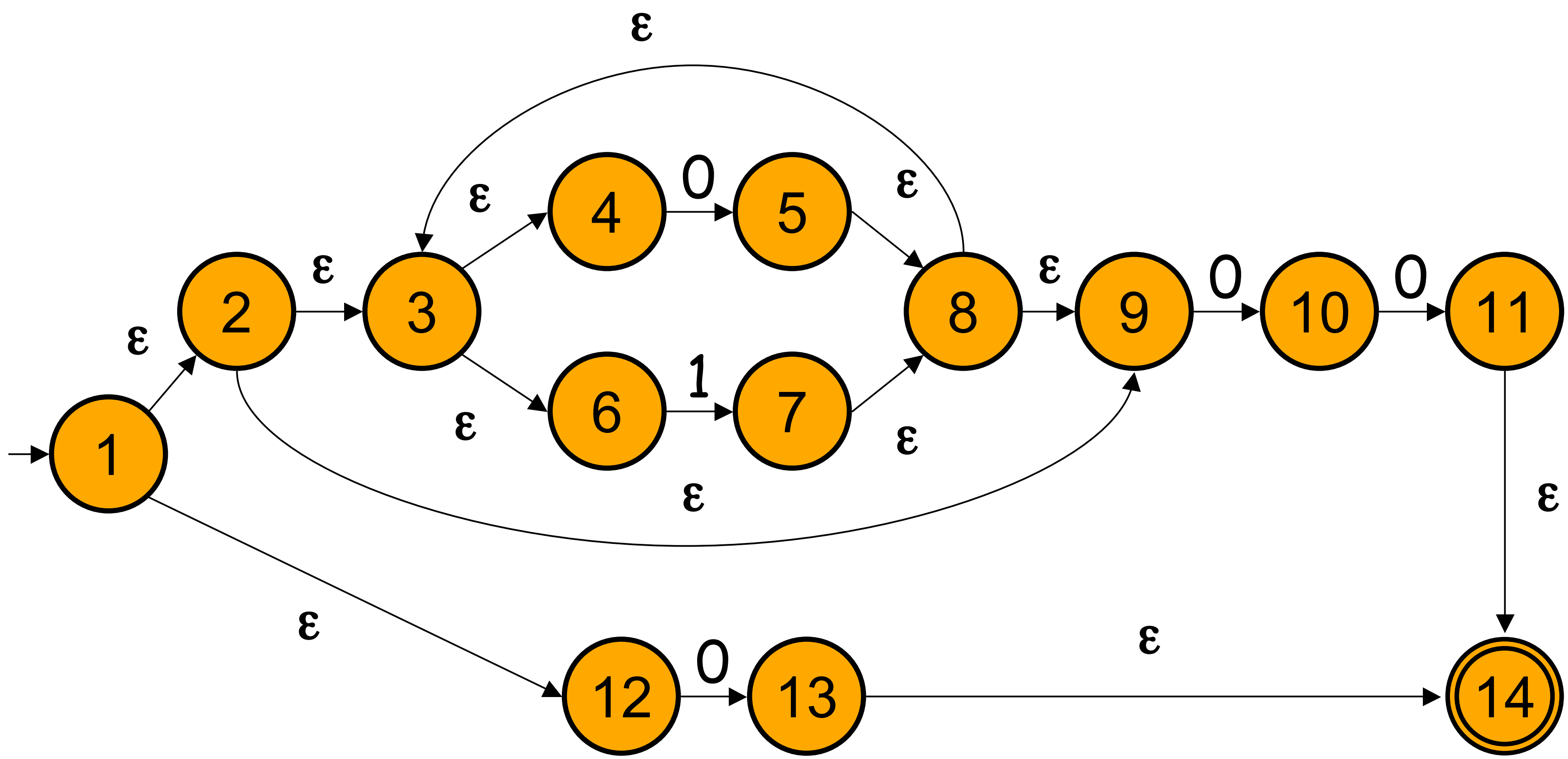
Subset Construction

```
add ε-closure(q0) to Dstates unmarked
while ∃ unmarked T ∈ Dstates do begin
    mark T;
    for each symbol c do begin
        U := ε-closure(move(T, c));
        if U ∉ Dstates then
            add U to Dstates unmarked
        Dtrans[T, c] := U;
    end
end
```

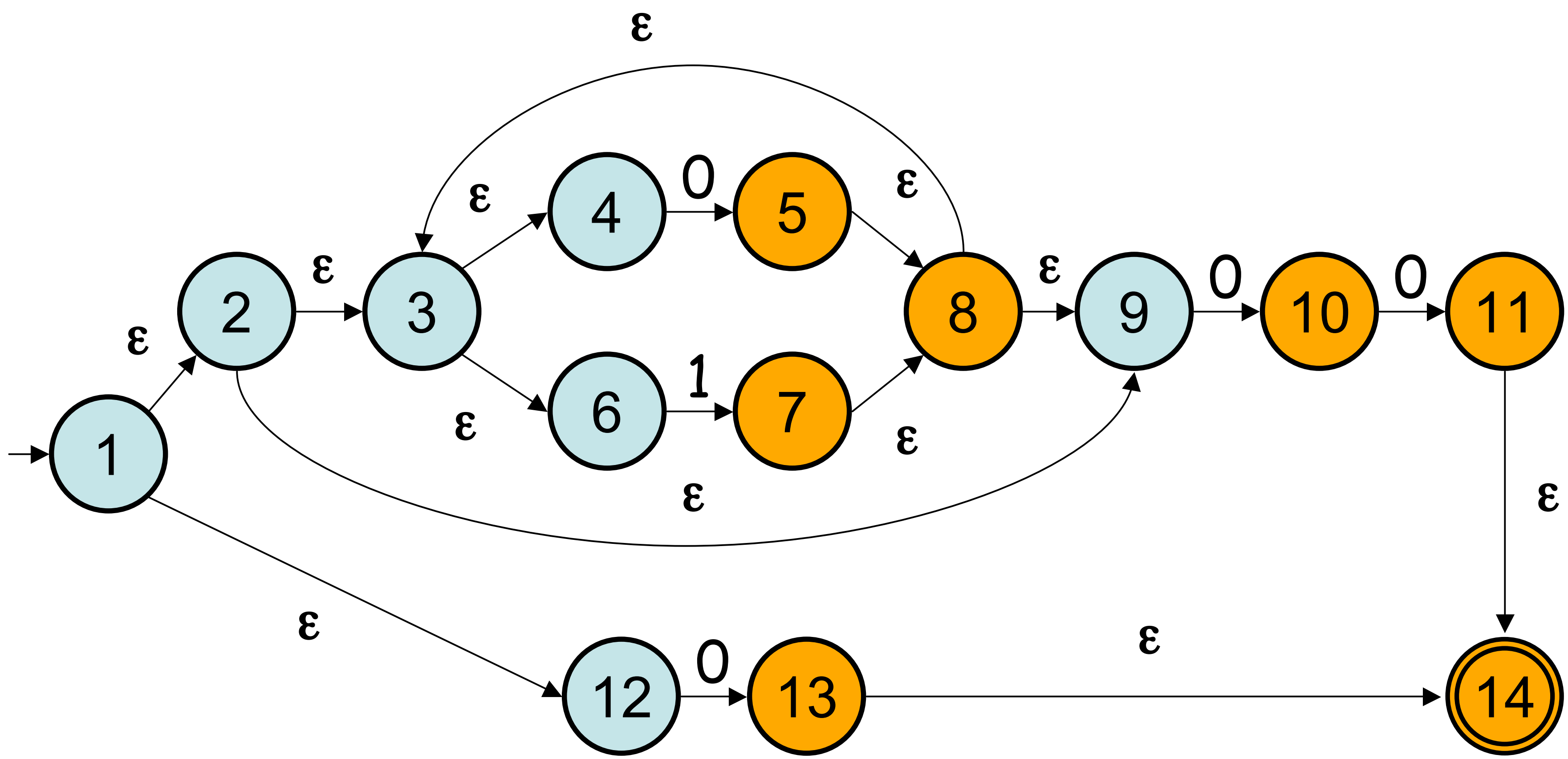
Subset Construction

```
states[0] = ε-closure({q0})
p = j = 0
while j ≤ p do begin
        for each symbol c do begin
                e = DFAedge(states[j], c)
                if e = states[i] for some i ≤ p
                then    Dtrans[j, c] = i
                else    p = p+1
                        states[p] = e
                        Dtrans[j, c] = p
        j = j + 1
        end
end
```

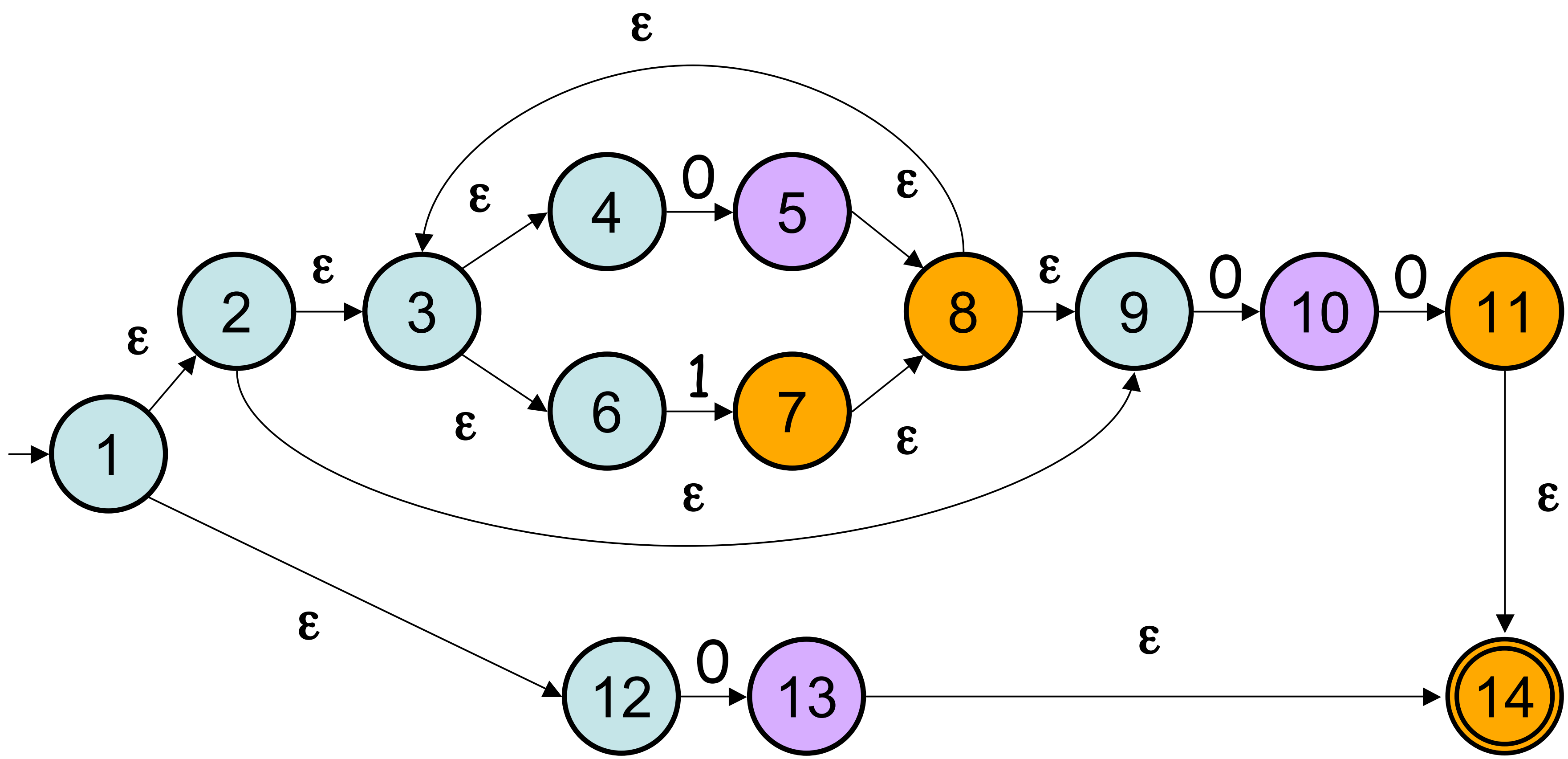
Example: subset construction



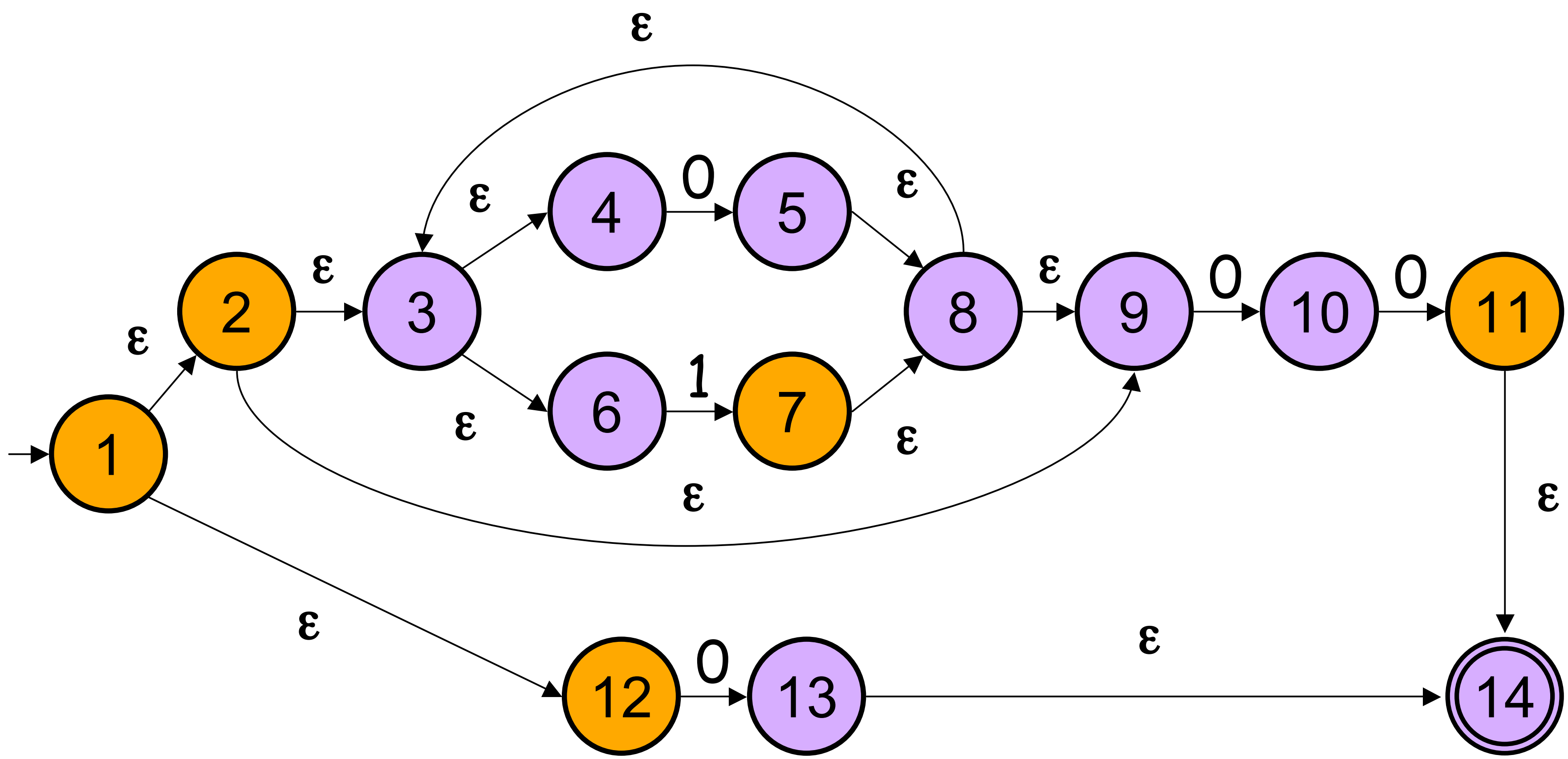
ε-*closure*(q_0)



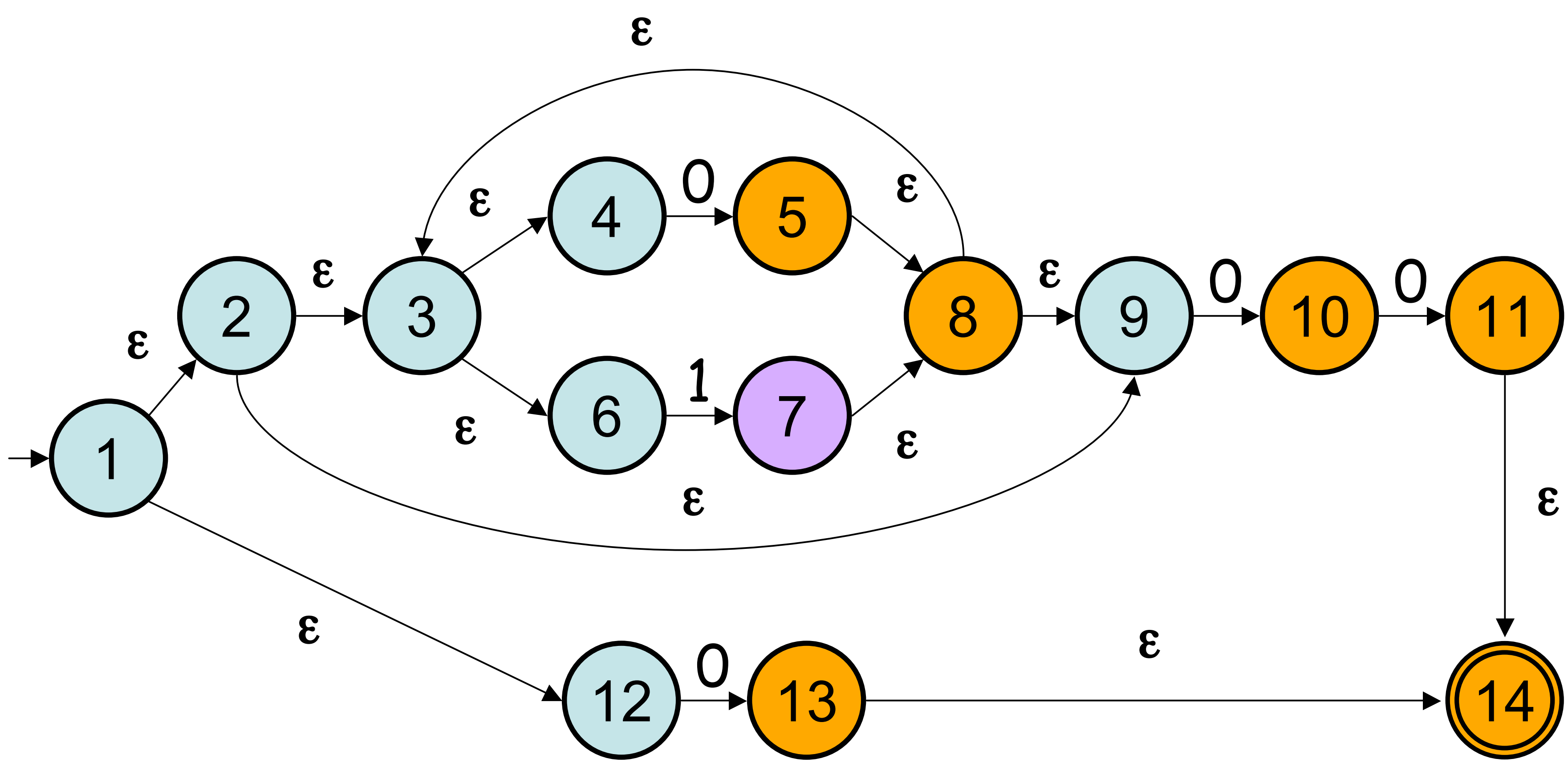
move(ε-*closure*(q_0), 0)



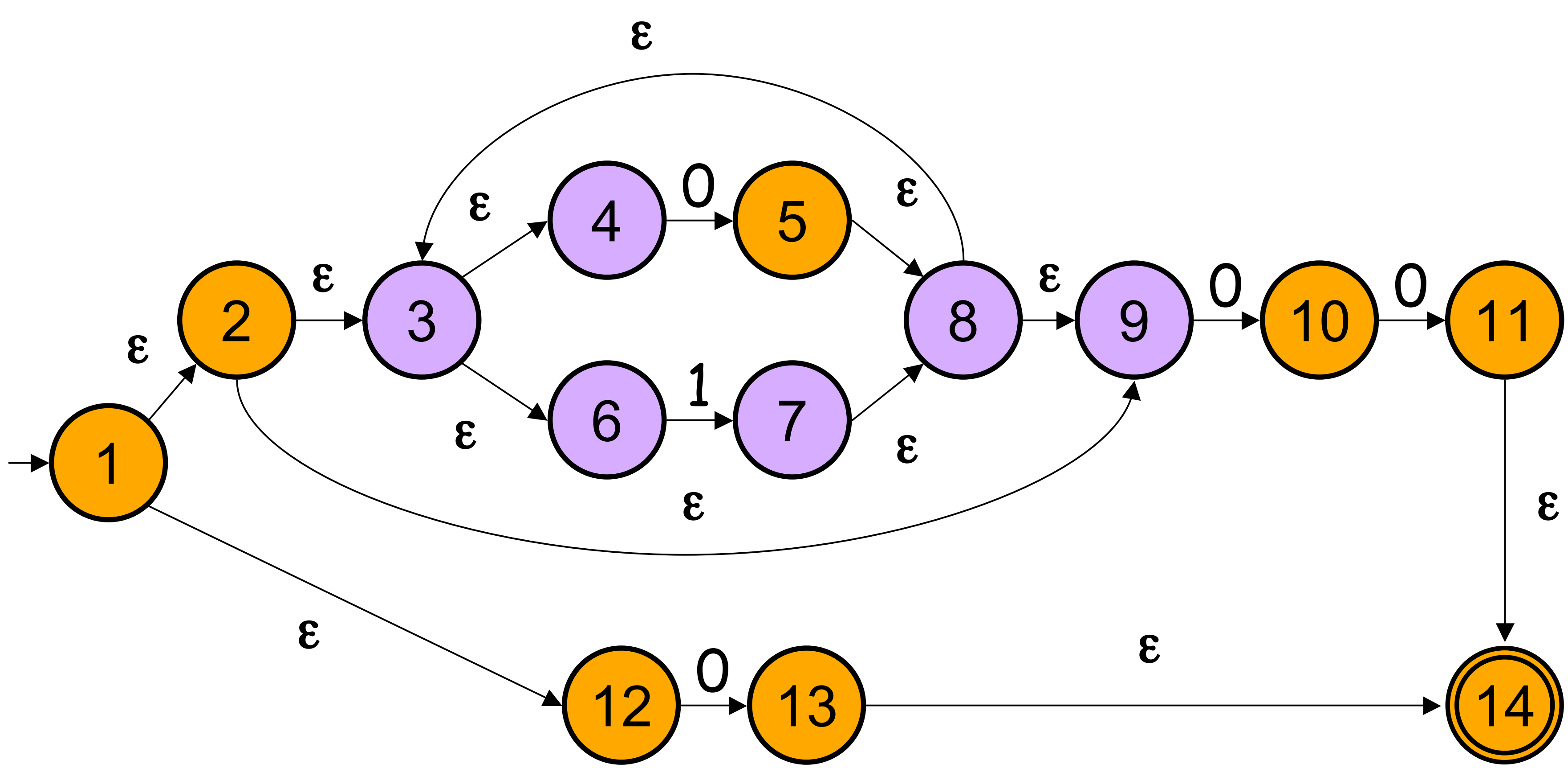
ε-*closure*(move(ε-*closure*(q_0), 0))



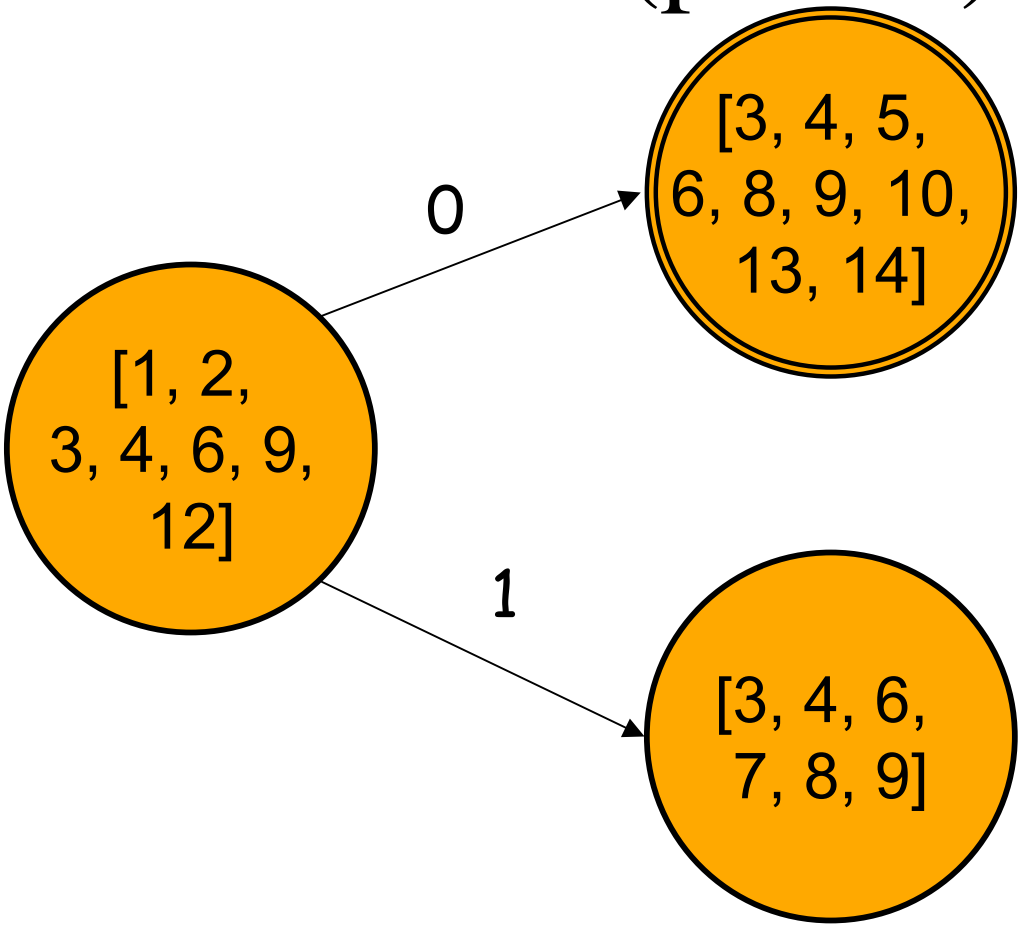
move(ε-*closure*(q_0), 1)



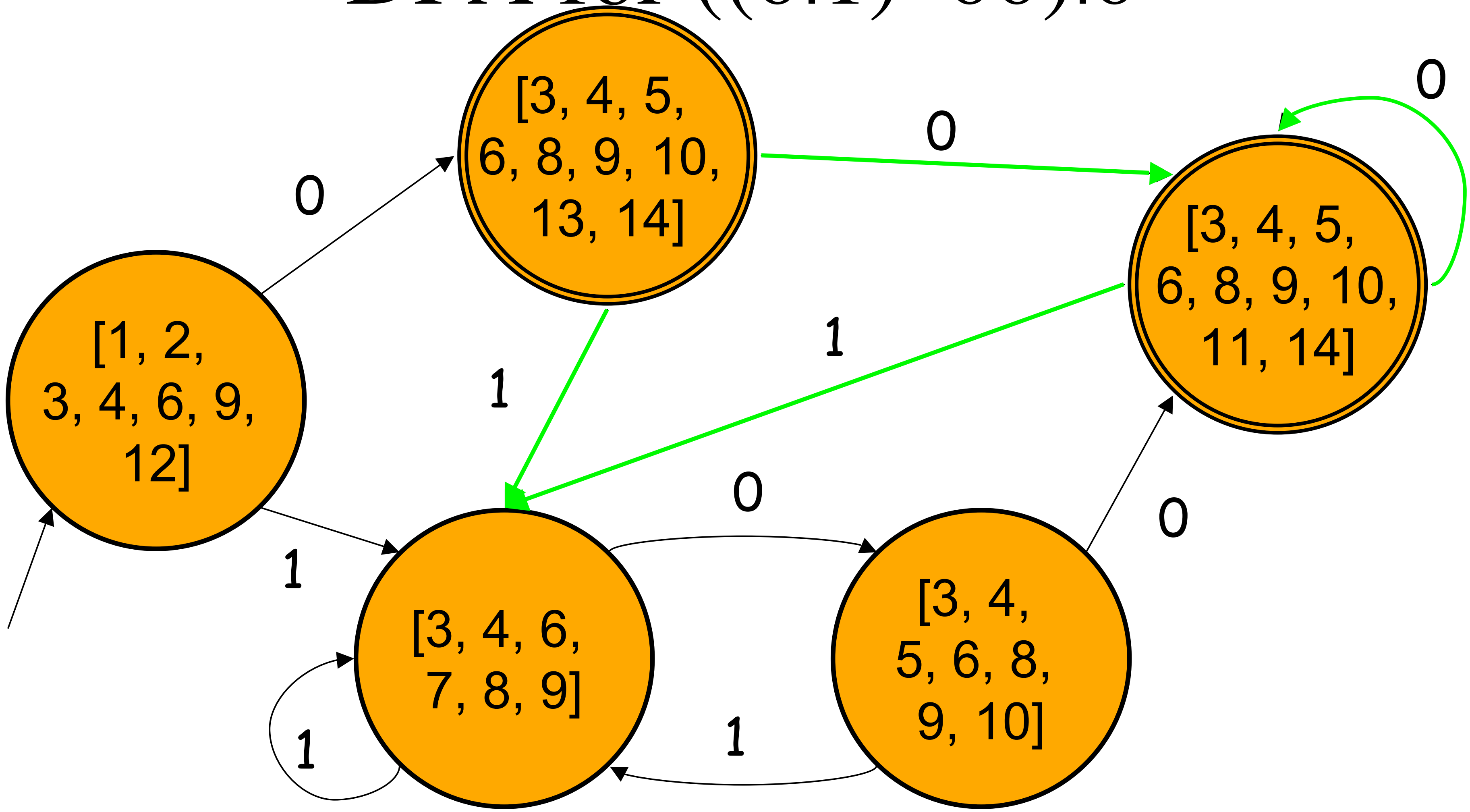
$\varepsilon\text{-}closure(\text{move}(\varepsilon\text{-}closure(q_0), 1))$



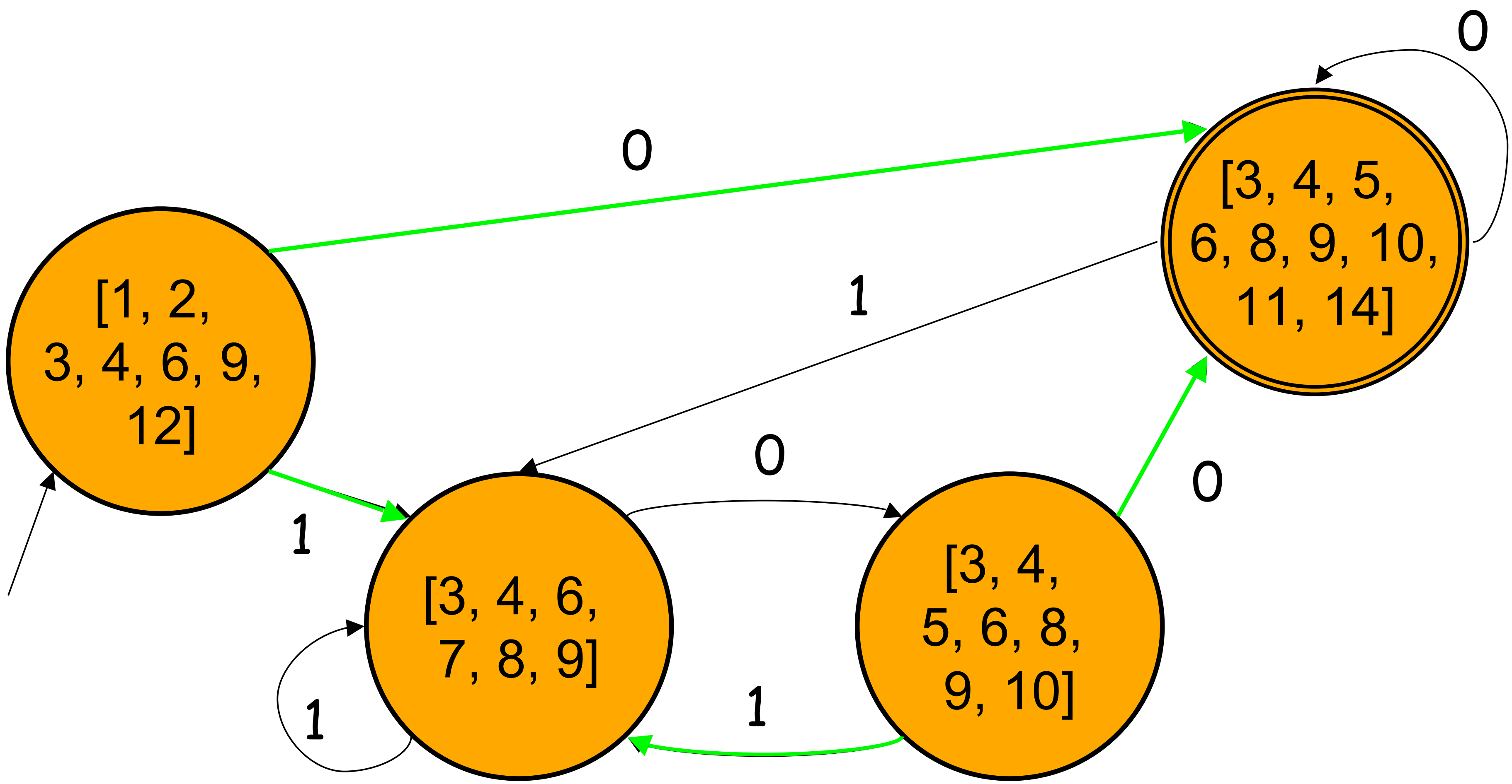
DFA (partial)



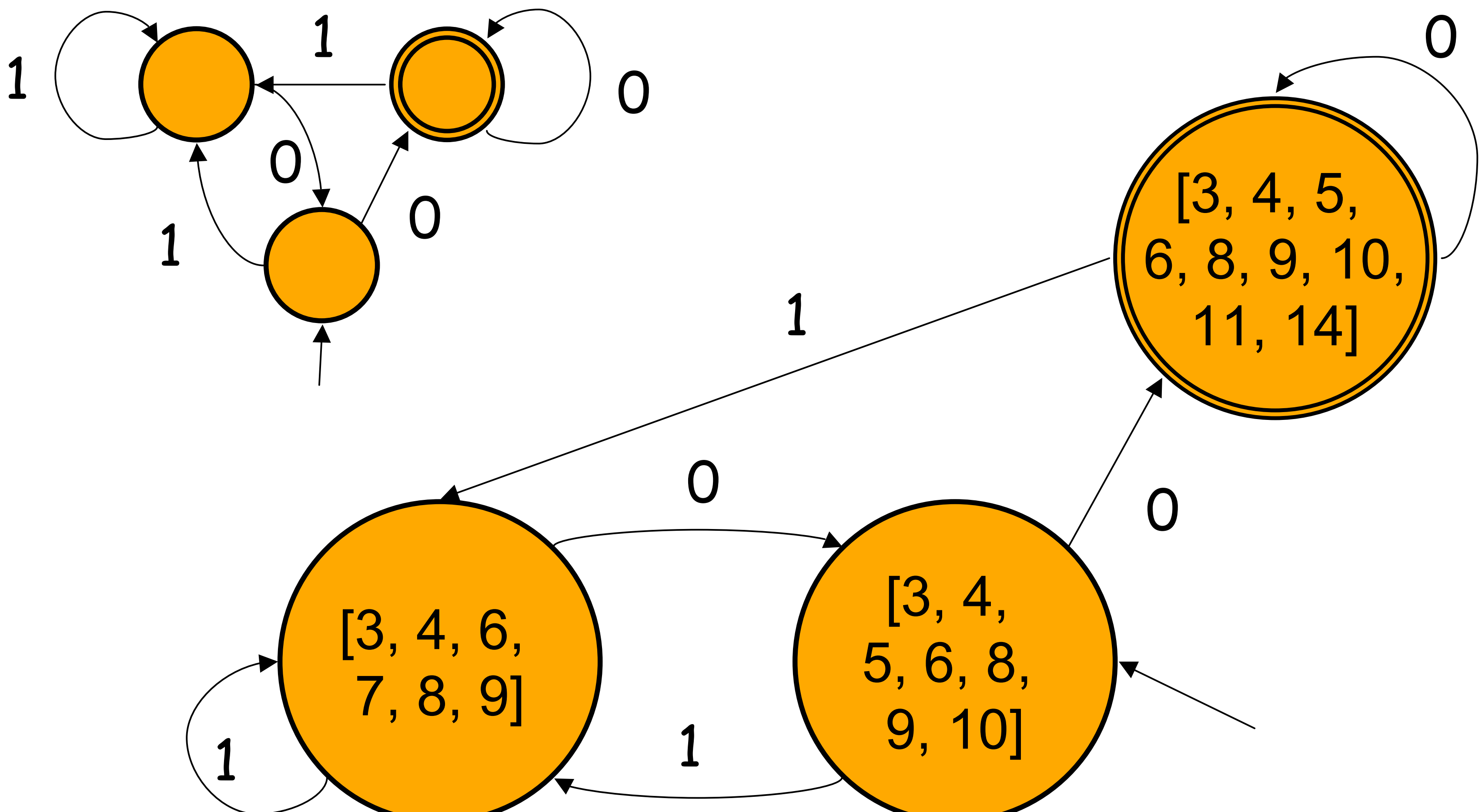
DFA for ((0|1)*00)|0



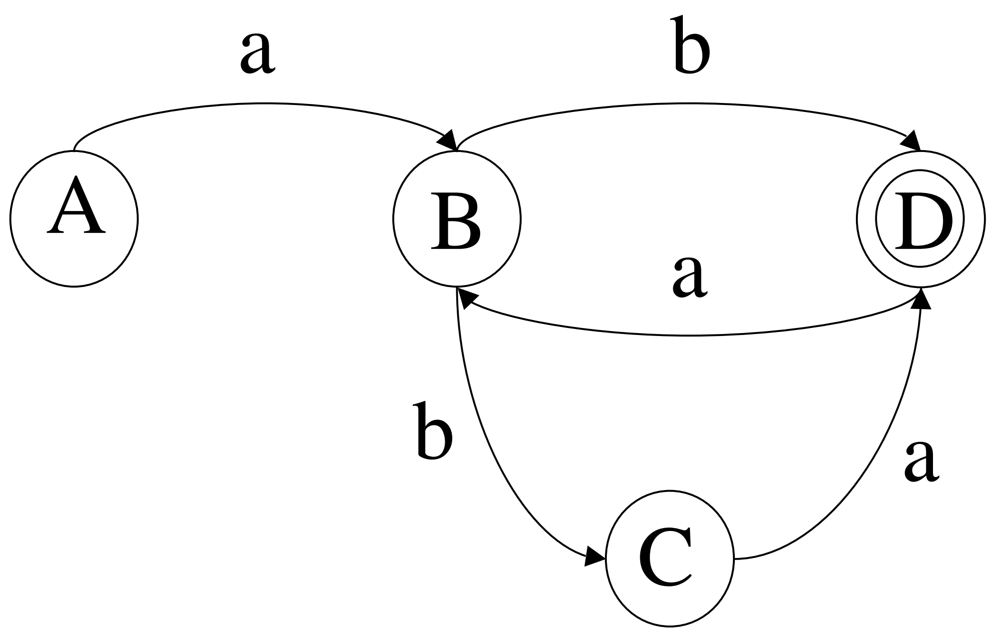
Minimization (I)



Minimization (II)

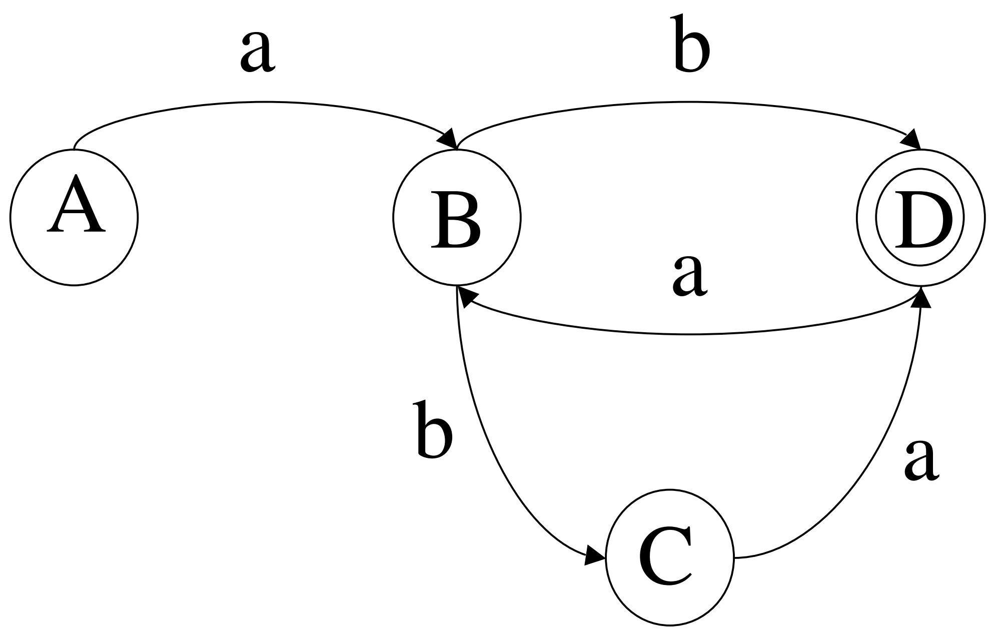


FSM to RegExp



What is the regular expression for this FSM?

NFA to RegExp



- $A = a\,B$
- $B = b\,D \mid b\,C$
- $D = a\,B \mid \varepsilon$
- $C = a\,D$

NFA to RegExp

- Three steps in the algorithm (apply in any order):

1. Substitution: for B = X pick every A = B | T and replace to get A = X | T
2. Factoring: (R S) | (R T) = R (S | T) and (R T) | (S T) = (R | S) T
3. Arden's Rule: For any set of strings S and T, the equation X = (S X) | T has X = (S*) T as a solution.

NFA to RegExp

- $A = a\ B$

  $B = b\ D \mid b\ C$

  $D = a\ B \mid \varepsilon$

  $C = a\ D$
- Substitute:

  $A = a\ B$

  $B = b\ D \mid b\ a\ D$

  $D = a\ B \mid \varepsilon$
- Factor:

  $A = a\ B$

  $B = (\ b \mid b\ a\ )\ D$

  $D = a\ B \mid \varepsilon$
- Substitute:

  $A = a\ (\ b \mid b\ a\ )\ D$

  $D = a\ (b \mid b\ a)\ D \mid \varepsilon$

NFA to RegExp

$A = a\ (\ b \mid b\ a\ )\ D$

$D = a\ (b \mid b\ a)\ D \mid \varepsilon$

- Factor:

$A = (a\ b \mid a\ b\ a\ )\ D$

$D = (a\ b \mid a\ b\ a)\ D \mid \varepsilon$

- Arden:

$A = (a\ b \mid a\ b\ a\ )\ D$

$D = (a\ b \mid a\ b\ a)^*\ \varepsilon$

- Remove epsilon:

$A = (a\ b \mid a\ b\ a\ )\ D$

$D = (a\ b \mid a\ b\ a)^*$

- Substitute:

$A = (a\ b \mid a\ b\ a\ )$
$\quad (a\ b \mid a\ b\ a)^*$

- Simplify:

$A = (a\ b \mid a\ b\ a)+$

Algorithms for FSMs
(finite-state machines)

- Recognition of a string in a regular language: is a string accepted by an NFA?
- Conversion of regular expressions to NFAs
- Determinization: converting NFA to DFA
- Converting an NFA into a regular expression
- Other useful *closure* properties: union, concatenation, Kleene closure, intersection