Matching regular expressions

Recall the *string searching* problem:

- Given a (long) text T (|T| = n) and a (short) string S (|S| = m), find an occurrence of S as a substring of T
  - Solvable by Brute Force – O(mn), or by KMP / BM / Suffix trees – O(m+n)

Often we want to search for strings that are not completely specified. Here we consider *regular expressions*.

A regular expression `R` defines a set of strings or a language L_R over an alphabet Σ.

Examples: Σ `= {a,b}`, R_1 `= a|b`, L_{R_1} `= {a,b}`

R_2 `= a*b`, L_{R_2} `= {b,ab,aab,aaab,aaaab,...}`

A string s matches `R` if and only if $s \in L_R$

Regular Expressions

Let Σ be the designated alphabet

- `R = ε` is a regular expression; $L_R = \{\varepsilon\}$
- `R = σ` is a regular expression, for any $\sigma \in \Sigma$; $L_R = \{\sigma\}$

If `R` and `S` are regular expressions so are

- `RS` (denotes *concatenation*); $L_{RS} = \{YZ : Y \in L_R \wedge Z \in L_S\}$

- `R|S` (denotes *choice* between `R` *or* `S`); $L_{R|S} = L_R \cup L_S$

If `R` is a regular expression, so is

- `R*` (denotes *0 or more copies* of `R`, called *closure*)

$$L_{R^*} = \{\varepsilon\} \cup \{YZ : Y \in L_R \wedge Z \in L_{R^*}\}$$

- `(R)` $L_{(R)} = L_R$

Order of precedence (highest first)

- closure (*), then concatenation, then choice (|)
- Parentheses can be used to override this rule

Example:

Σ = {a,b,c,d}, R = (a*b|ac)d

L_R = {acd,bd,abd,aabd,aaabd,aaaabd,...}

Other operations

- *complement* **-x**
  - equivalent to the 'or' of all characters in Σ except **x**
- *any single character* **?**
  - equivalent to the 'or' of all characters
- *etc.*

Problem: Given a text T and a regular expression `R`, find an occurrence, if any, of a substring of T that matches `R`.

The unix egrep command. Consider the file `simple.txt`:

```
1: baccba
2: bd
3: aaaaacdcccc
4: acaabadcbaccdb
5: aaaaaabdbbcbb
```

```
$ egrep -o -n '(a*b|ac)d' < simple.txt
```

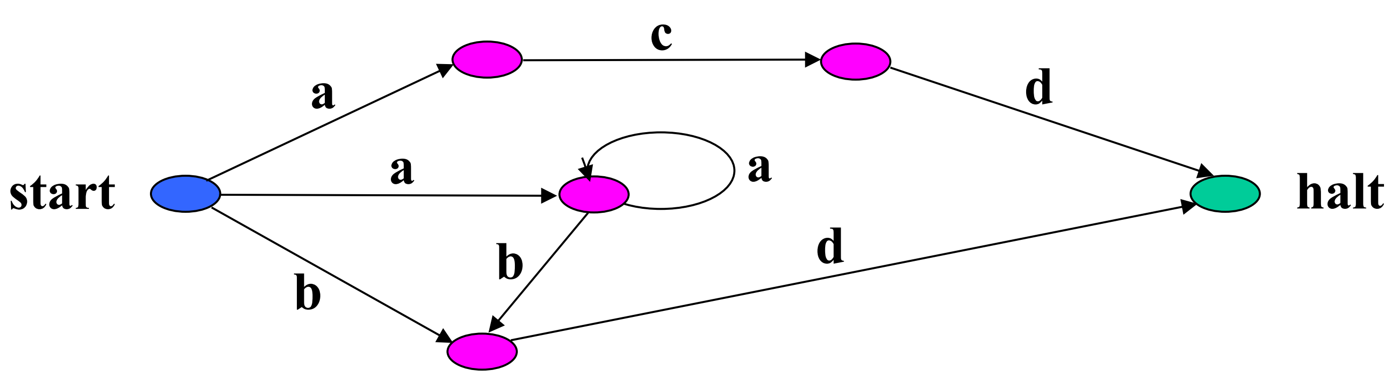
outputs the text in red

Pattern matching algorithm (as in *egrep*)

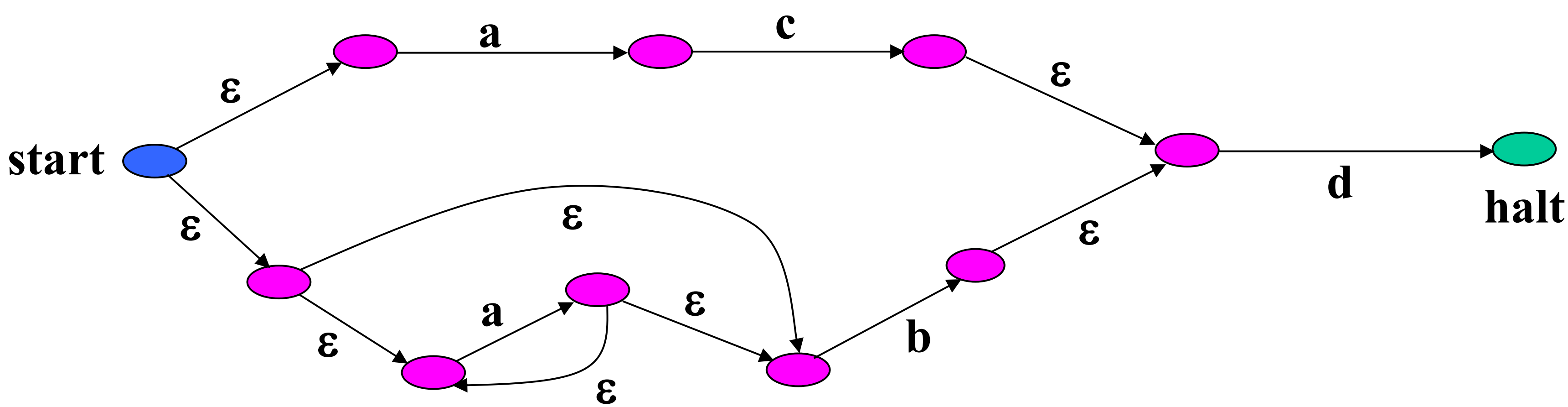
- generalisation of Brute Force algorithm
- consider each position j of T, where j runs from 1 to |T|
- keep track of which prefixes of `R` are matched by substrings of T ending at position j

Non-deterministic finite automata (NDFAs)

- An NDFA is a directed graph
  - the vertices are the *states*
  - there is a single *start* state and a single *accepting* or *halt* state
  - the edges are *transitions*, each labelled by a single character or by ε (the empty string)
- An NDFA *accepts* a string S if there is a path from the start state to the halt state, whose edge labels spell out S.
- Example: NDFA for `(a*b|ac)d`



An alternative NDFA for `(a*b|ac)d`

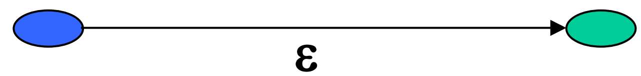


Objective: for any regular expression `R`, construct an NDFA **A** for `R` such that **A** accepts a string `s` if and only if `s` matches `R`

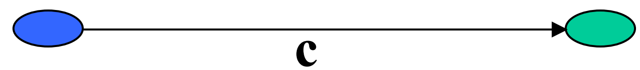
NDFA construction (for a regular expression R)

- **Build partial automata for sub-expressions of R, and define ways of combining them**

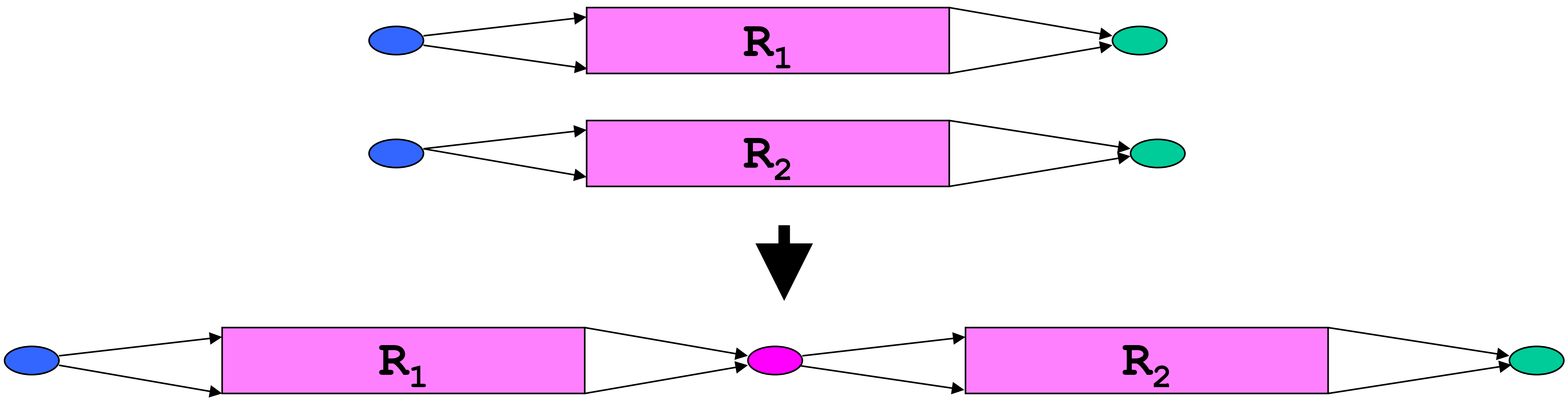
1. $R = \varepsilon$ – empty string:



2. $R = c$ – single character:

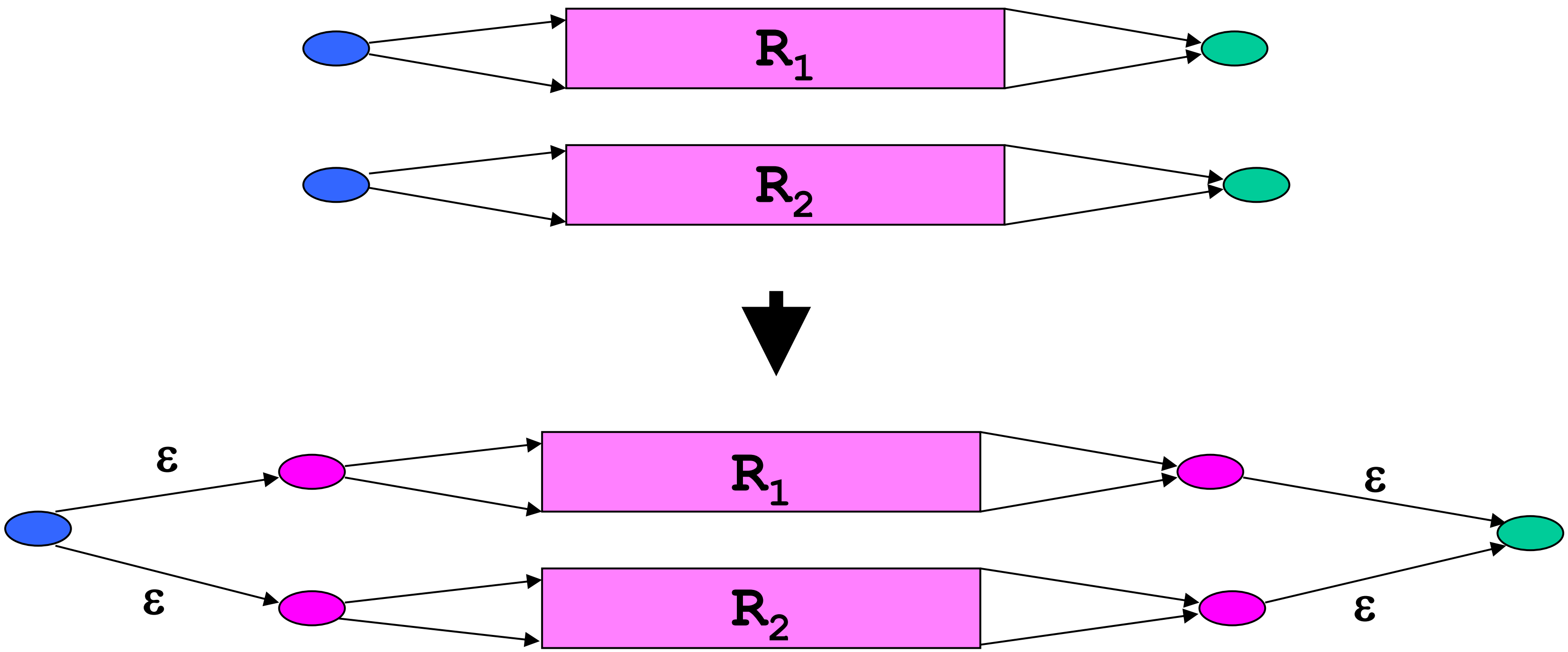


3. $R = R_1R_2$ – concatenation:



NDFA Construction (continued)

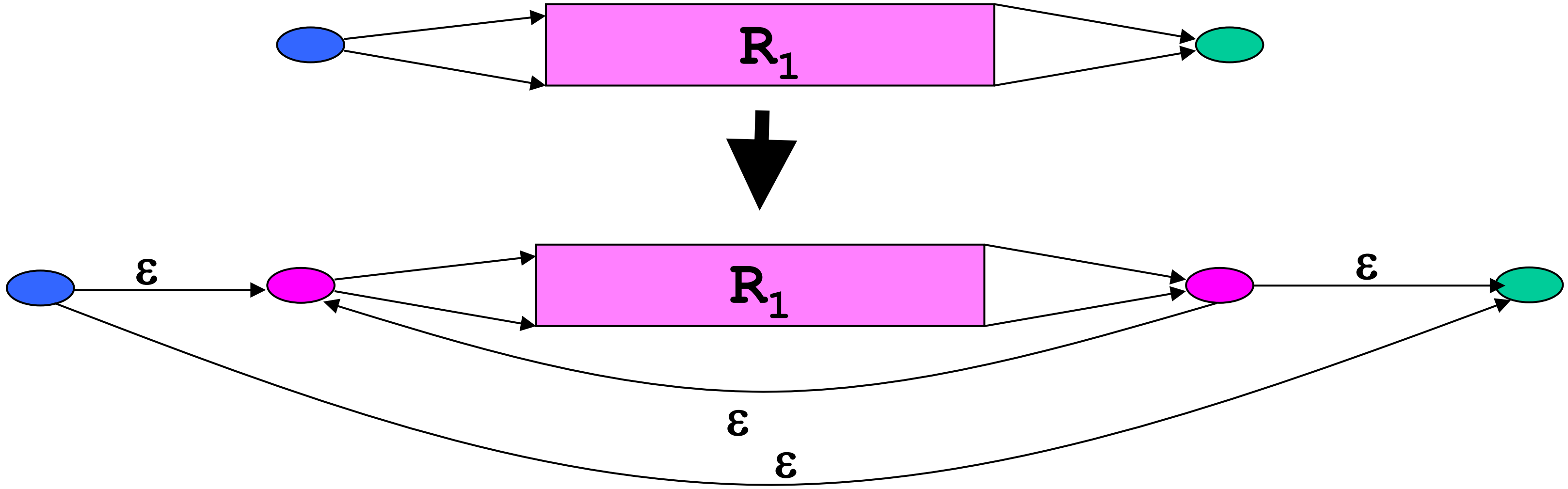
4. $R = R_1 | R_2$ – choice:



5. For (R) use the NDFA for R

NDFA Construction (continued)

6. $R = R_1^*$ – closure:



- Length of a regular expression R, denoted $|R|$ is the number of symbols (characters in Σ, operations and brackets) in R
- In the constructed NDFA:
  - there is one start state and one halt state
  - there are at most $2r$ states, where r is the length of R (≤ 2 new states per step)
  - each state has either 1 outgoing edge labelled by a character or ≤ 2 outgoing edges labelled by ε

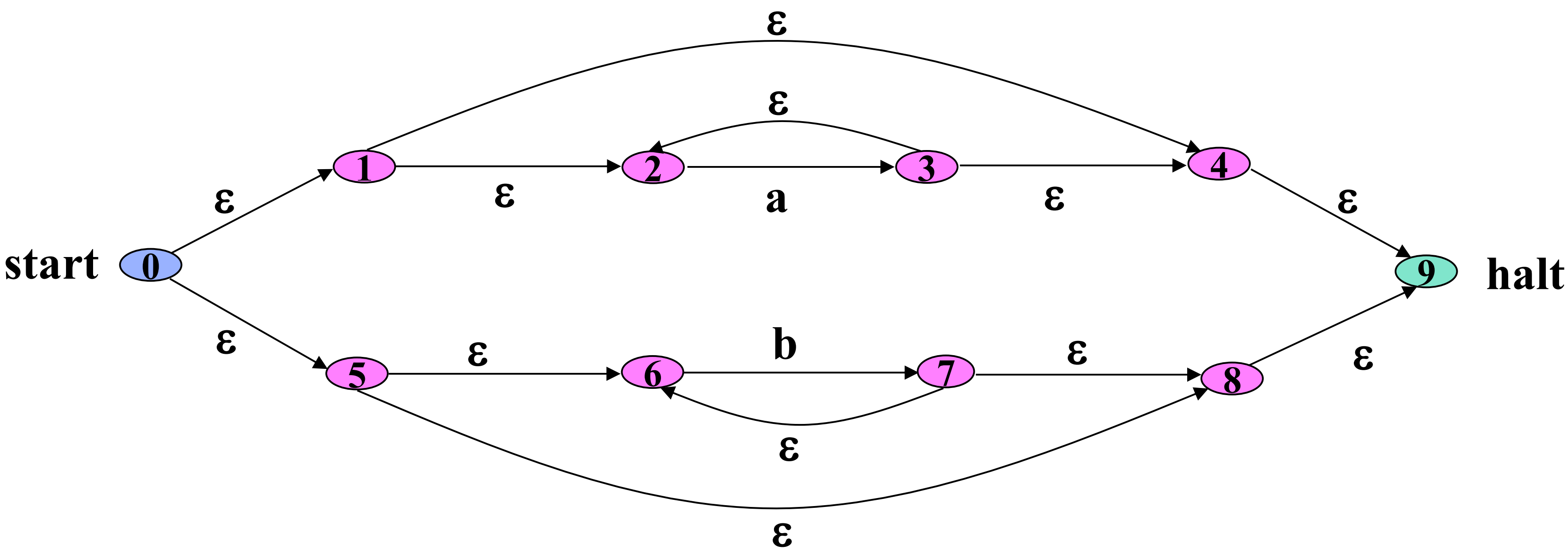
Simulating an NDFA

```
public class SetOfStates {
    ...
}


/** Returns the union of ss and the states reachable
  * from a state in ss by following only ε-edges */
public SetOfStates extendByEpsilon (SetOfStates ss)


/** Returns the set of states reachable from a state in
  * ss by following an edge labelled with character c */
public SetOfStates extendByChar (SetOfStates ss, char c)
```

Examples of `extendByEpsilon`, `extendByChar`



NDFA for `a*|b*`

```
extendByEpsilon({0,3}) = {0,1,2,3,4,5,6,8,9}

extendByChar({0,2,6},'a') = {3}
```

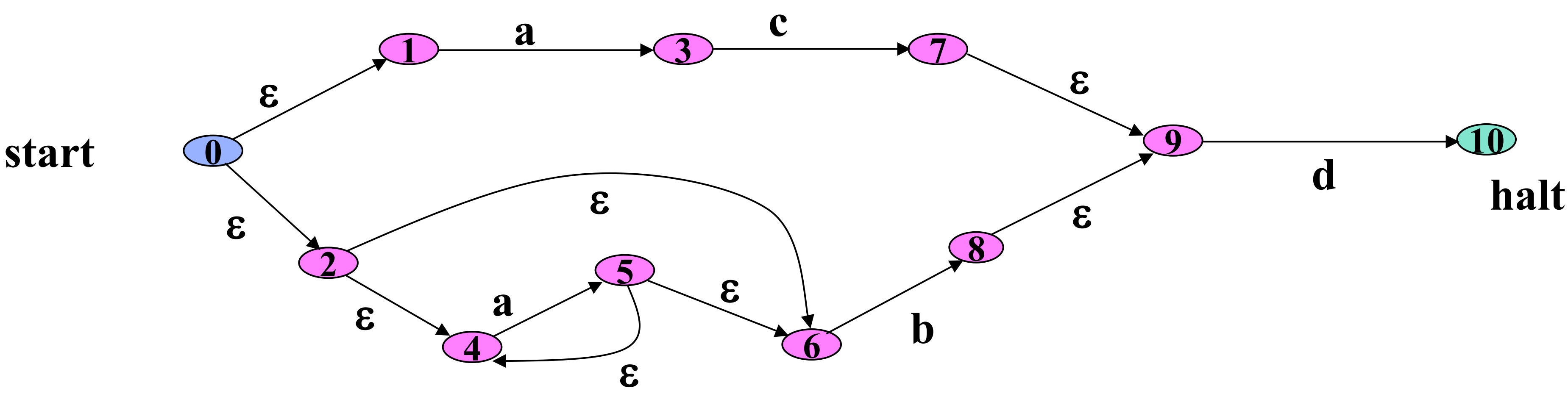
Algorithm to simulate an NDFA

```
/** Input:  a string T, and an NDFA for R
  * Output: true if the NDFA accepts a substring of T;
  * false otherwise; Assume chars of a string of length
  * r are indexed by 1..r */

public boolean simulateNDFA(State startState, String T)
{
  SetOfStates x, y;
  x = extendByEpsilon({startState});
  if (x.contains(haltState))
    return true; // ε matches R

  for (int j = 1; j <= T.length(); j++) {
    y = extendByChar(x, T.charAt(j)) ∪ {startState};
    x = extendByEpsilon(y);
    if (x.contains(haltState))
      return true;
  }
  return false;
}
```

NDFA for (a*b|ac)d



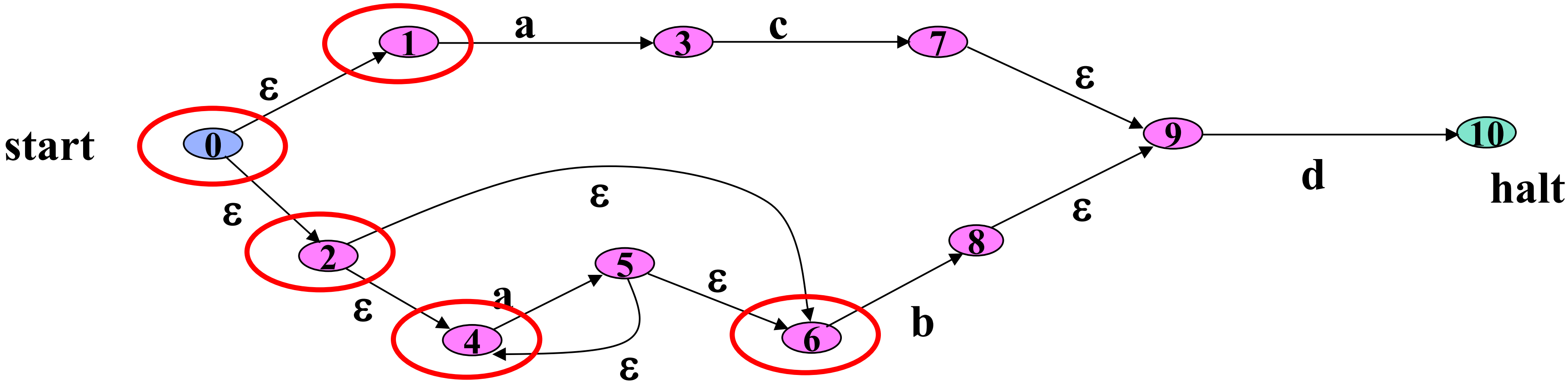
Consider the simulation of this NDFA when searching the string

```
T = c a a b c a c a b d a c d
    1 2 3 4 5 6 7 8 9 0 1 2 3
```

Note: `startState = 0`

`extendByEpsilon({startState}) = {0,1,2,4,6}`

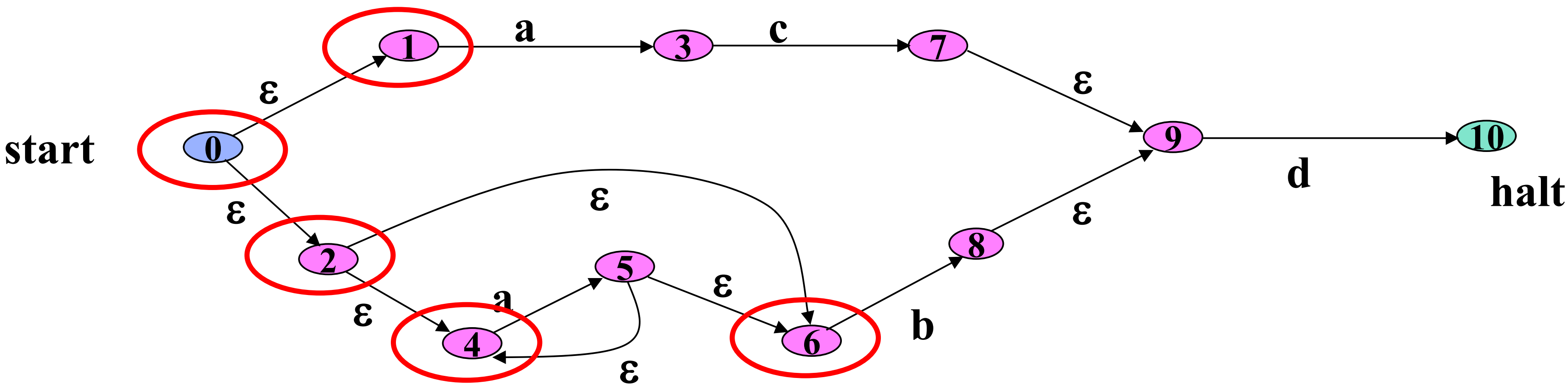
NDFA for `(a*b|ac)d`




```
T = c a a b c a c a b d a c d
    1 2 3 4 5 6 7 8 9 0 1 2 3
```

```
X:                    {0,1,2,4,6}
```

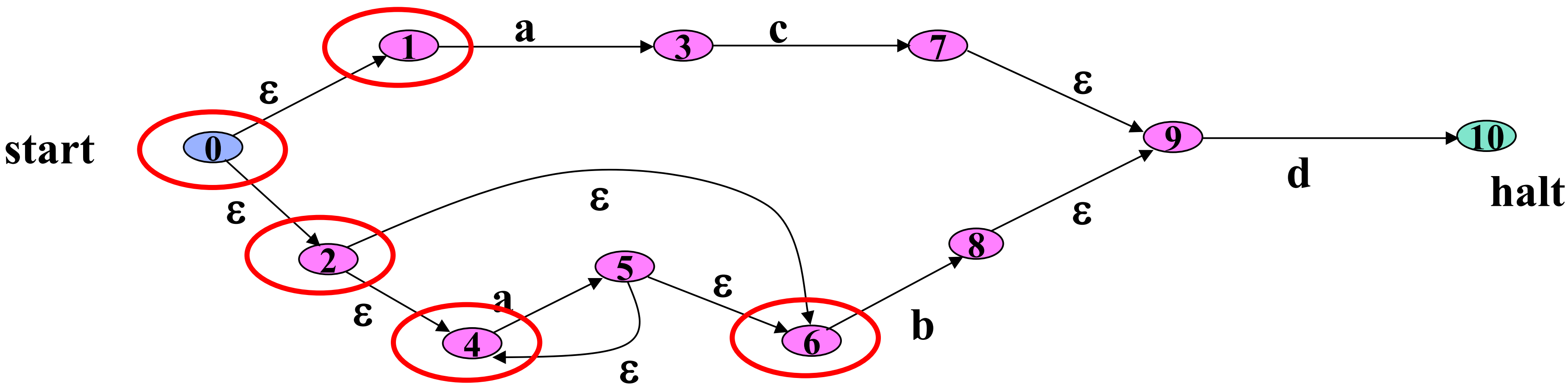
NDFA for `(a*b|ac)d`




```
T = c a a b c a c a b d a c d
    1 2 3 4 5 6 7 8 9 0 1 2 3
```

```
J:                   1
T(J):                c
Y:                   {0}
X:    {0,1,2,4,6}
```

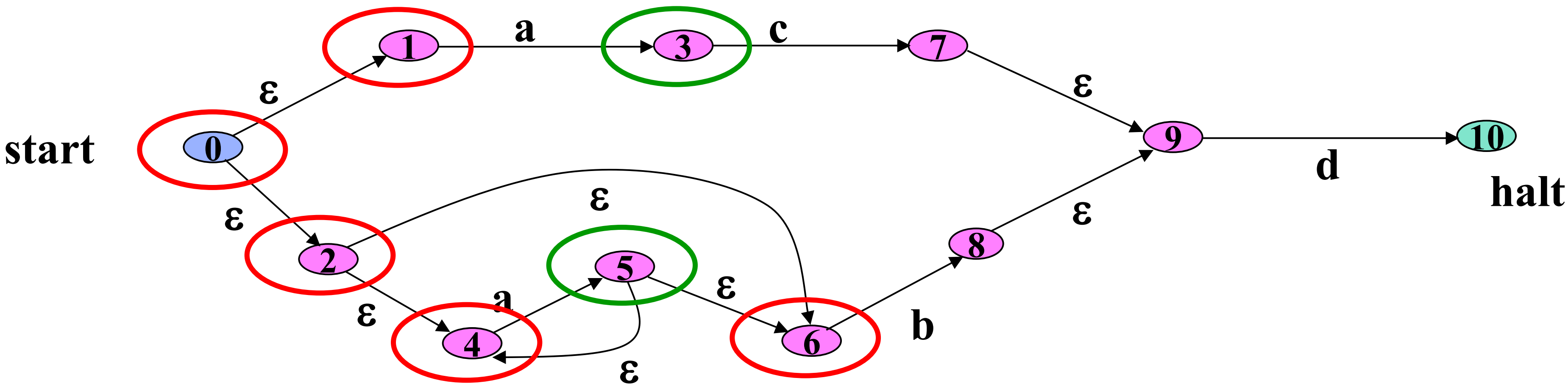
NDFA for (a*b|ac)d




```
T = c a a b c a c a b d a c d
    1 2 3 4 5 6 7 8 9 0 1 2 3

J:                       1
T(J):                    c
Y:                       {0}
X:     {0,1,2,4,6}  →   {0,1,2,4,6}
```

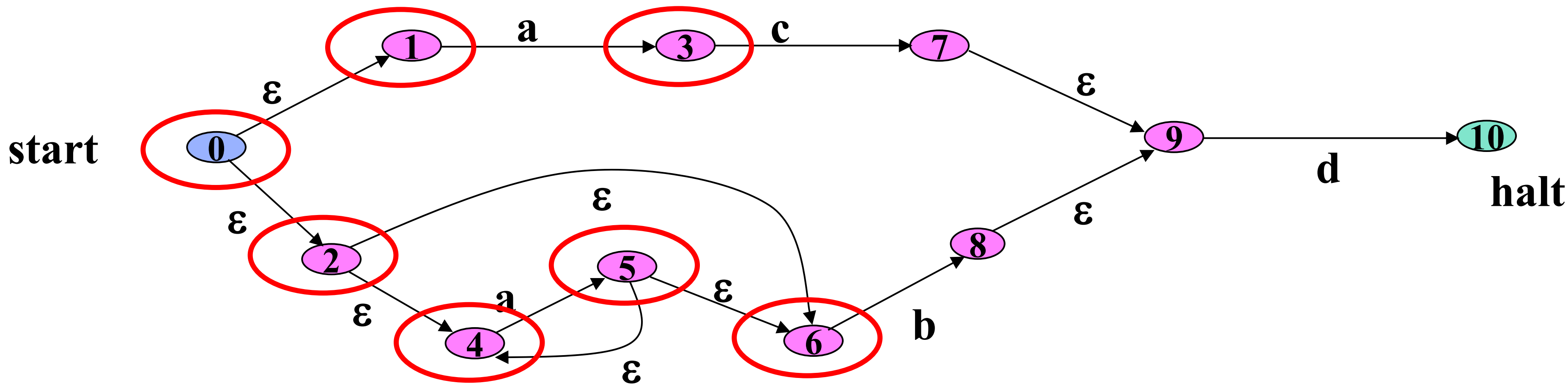
NDFA for (a*b|ac)d




```
T = c a a b c a c a b d a c d
    1 2 3 4 5 6 7 8 9 0 1 2 3

J:                   2
T(J):                a
Y:                   {0,3,5}
X:     {0,1,2,4,6}
```

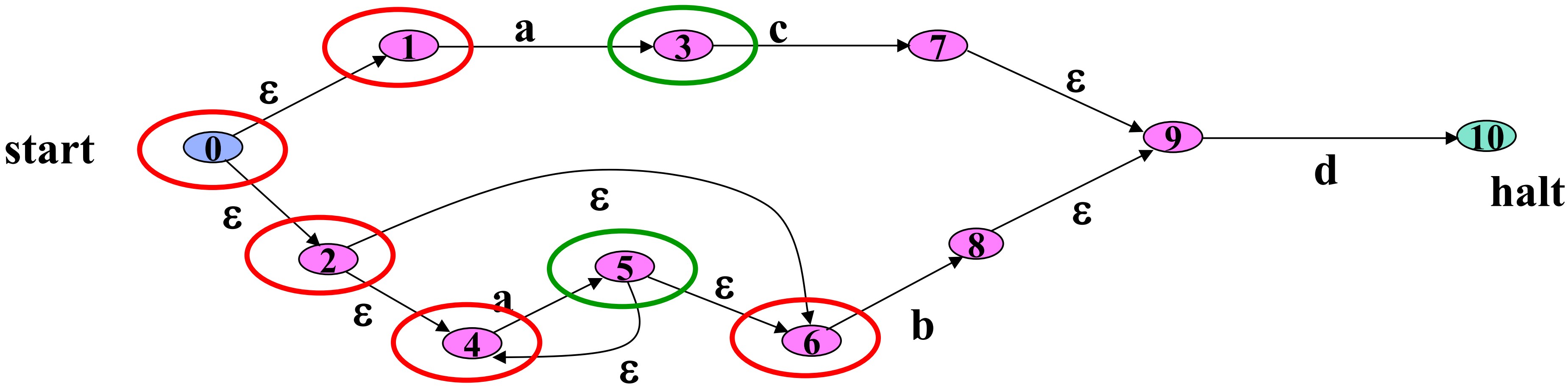
NDFA for `(a*b|ac)d`




```
T = c a a b c a c a b d a c d
    1 2 3 4 5 6 7 8 9 0 1 2 3
```

```
J:                       2
T(J):                    a
Y:                       {0,3,5}
X:    {0,1,2,4,6}  →  {0,1,2,3,4,5,6}
```

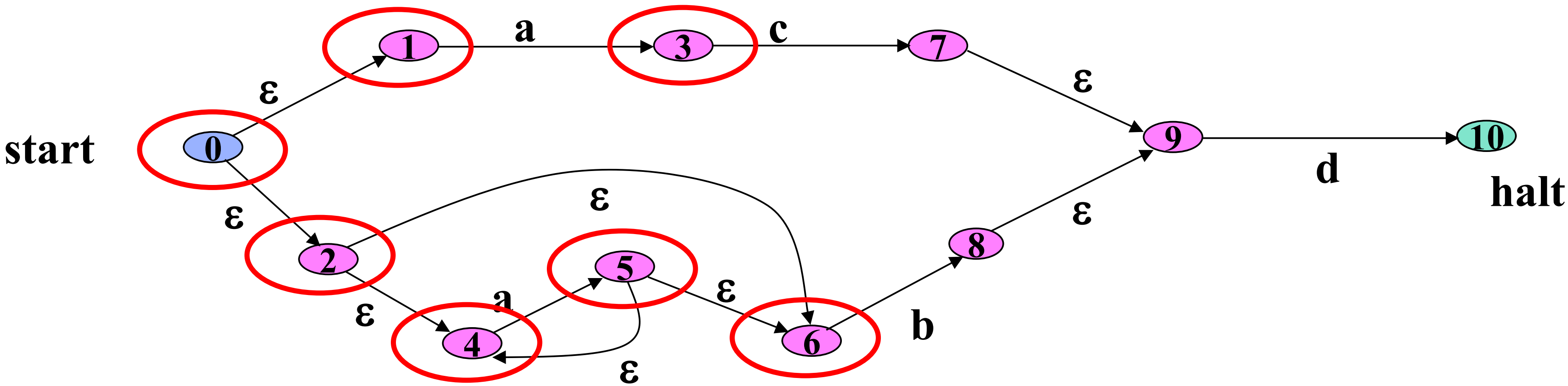
NDFA for (a*b|ac)d




```
T = c a a b c a c a b d a c d
    1 2 3 4 5 6 7 8 9 0 1 2 3
```

```
J:                    3
T(J):                 a
Y:                    {0,3,5}
X: {0,1,2,3,4,5,6}
```

NDFA for (a*b|ac)d



T = c a a b c a c a b d a c d
1 2 3 4 5 6 7 8 9 0 1 2 3

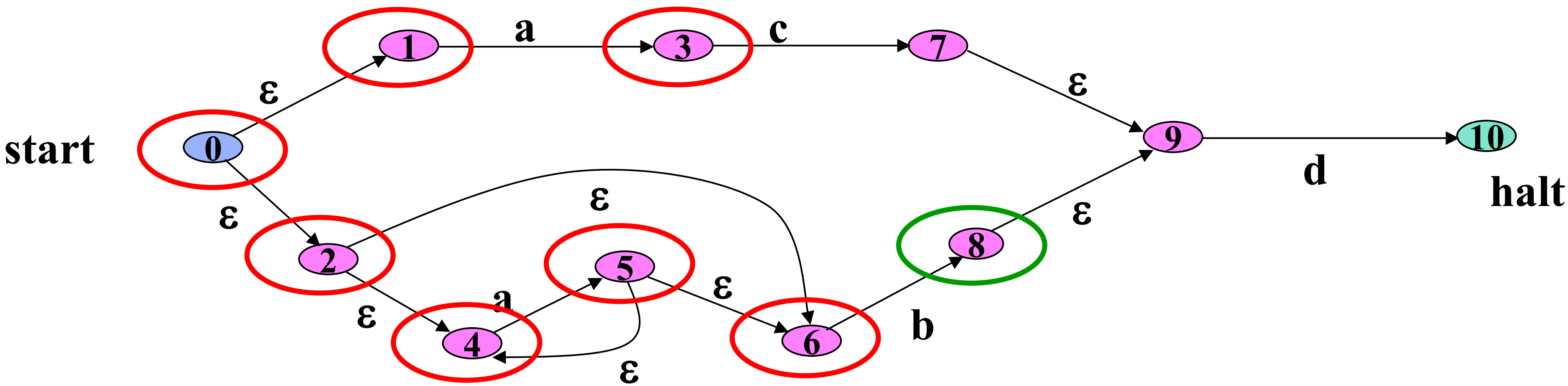
J: 3

T(J): a

Y: {0,3,5}

X: {0,1,2,3,4,5,6} → {0,1,2,3,4,5,6}

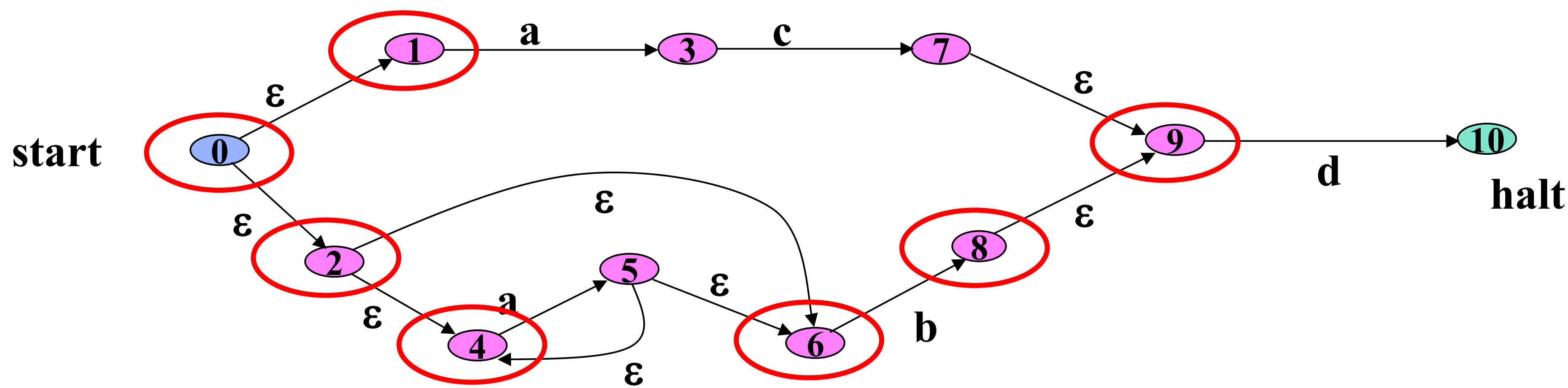
NDFA for (a*b|ac)d




```
T = c a a b c a c a b d a c d
    1 2 3 4 5 6 7 8 9 0 1 2 3
```

```
J:                    4
T(J):                 b
Y:                    {0,8}
X: {0,1,2,3,4,5,6}
```

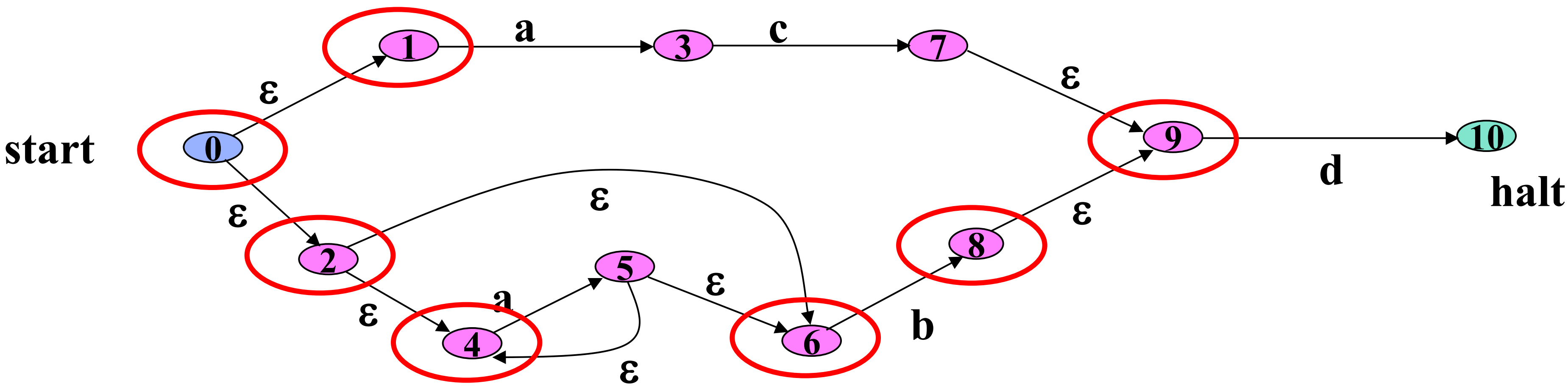
NDFA for (a*b|ac)d




```
T = c a a b c a c a b d a c d
    1 2 3 4 5 6 7 8 9 0 1 2 3
```

```
J:                     4
T(J):                  b
Y:                     {0,8}
X: {0,1,2,3,4,5,6}→{0,1,2,4,6,8,9}
```

NDFA for (a*b|ac)d



T = c a a b c a c a b d a c d
    1 2 3 4 5 6 7 8 9 0 1 2 3

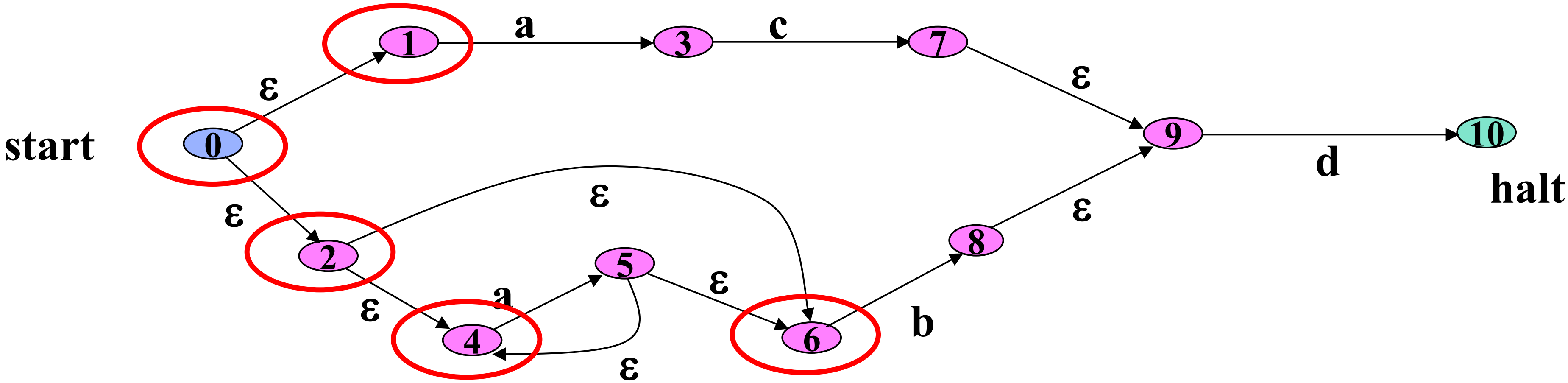
J: 5

T(J): c

Y: {0}

X: {0,1,2,4,6,8,9}

NDFA for (a*b|ac)d



T = c a a b c a c a b d a c d

1 2 3 4 5 6 7 8 9 0 1 2 3

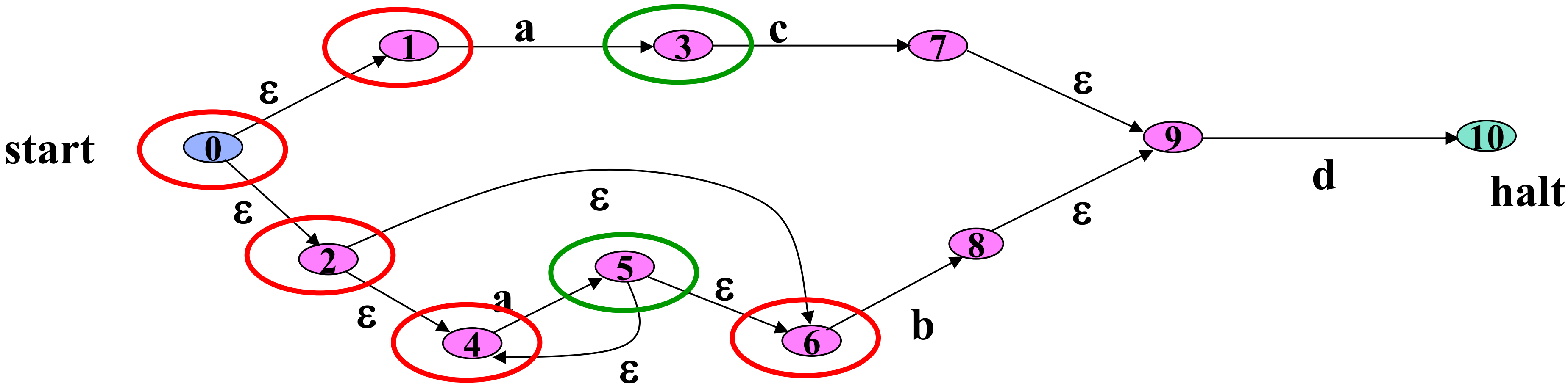
J: 5

T(J): c

Y: {0}

X: {0,1,2,4,6,8,9}→{0,1,2,4,6}

NDFA for (a*b|ac)d



T = c a a b c a c a b d a c d
1 2 3 4 5 6 7 8 9 0 1 2 3

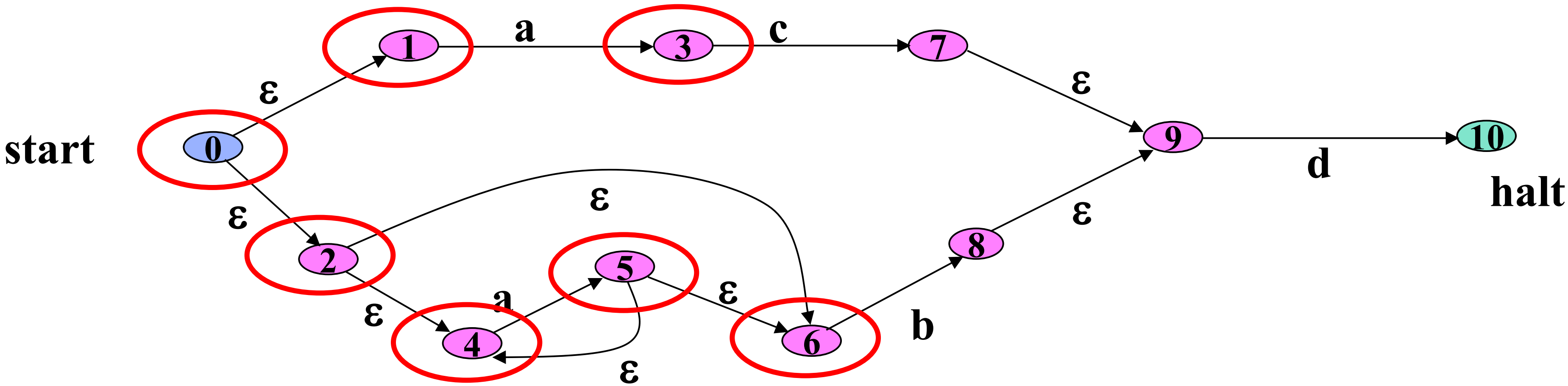
J: 6

T(J): a

Y: {0,3,5}

X: {0,1,2,4,6}

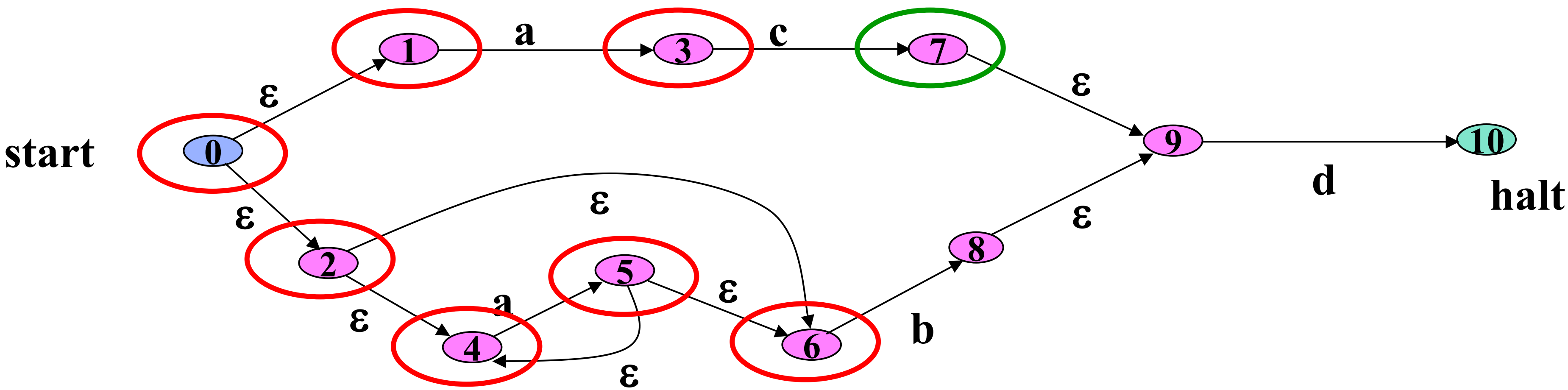
NDFA for `(a*b|ac)d`




```
T = c a a b c a c a b d a c d
    1 2 3 4 5 6 7 8 9 0 1 2 3

J:                      6
T(J):                   a
Y:                      {0,3,5}
X:    {0,1,2,4,6} →     {0,1,2,3,4,5,6}
```

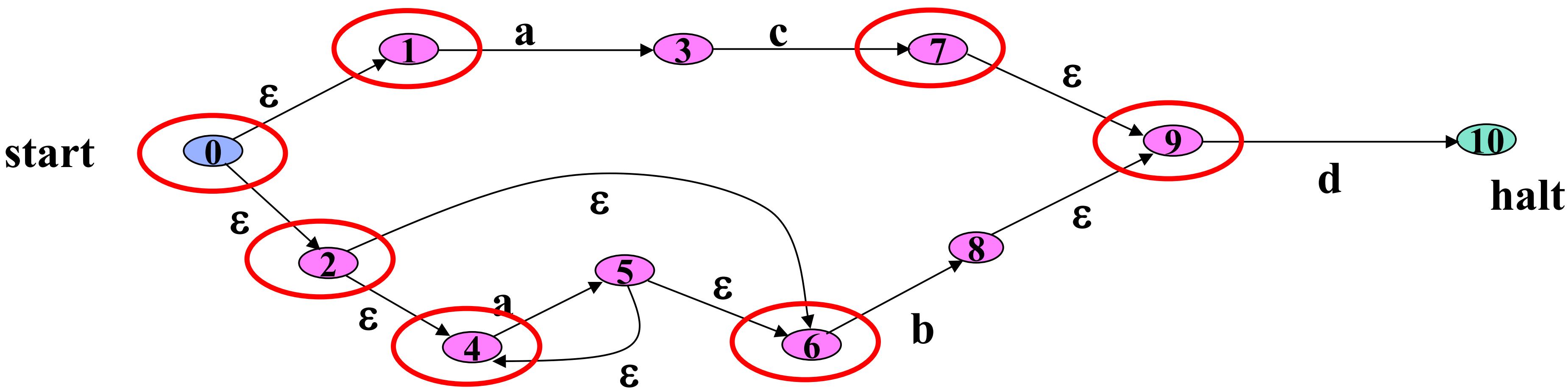
NDFA for (a*b|ac)d



T = c a a b c a c a b d a c d
1 2 3 4 5 6 7 8 9 0 1 2 3

J: 7
T(J): c
Y: {0,7}
X: {0,1,2,3,4,5,6}

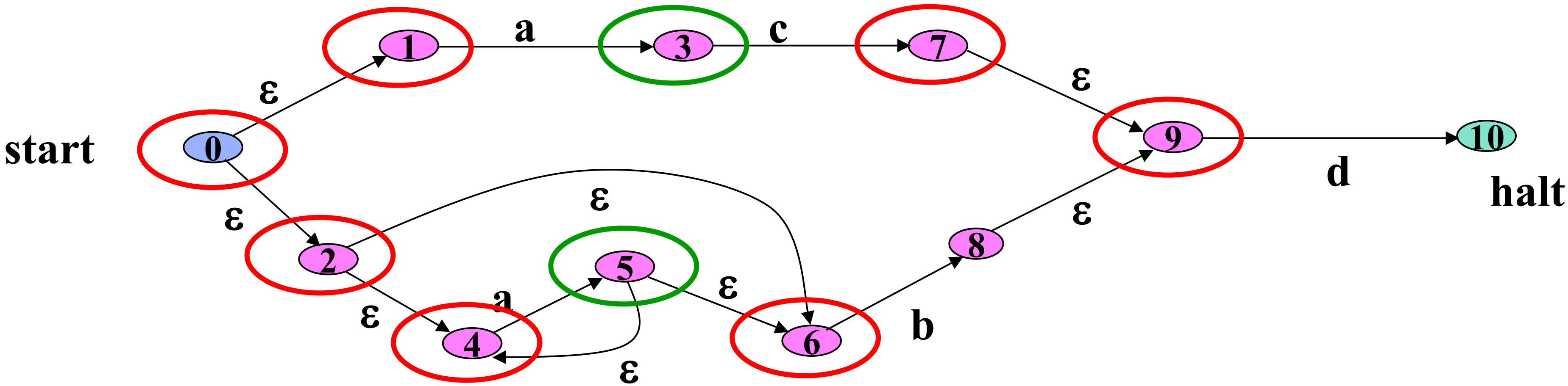
NDFA for (a*b|ac)d




```
T = c a a b c a c a b d a c d
    1 2 3 4 5 6 7 8 9 0 1 2 3
```

```
J:                      7
T(J):                   c
Y:                      {0,7}
X: {0,1,2,3,4,5,6}→     {0,1,2,4,6,7,9}
```

NDFA for (a*b|ac)d



T = c a a b c a c *a* b d a c d

1 2 3 4 5 6 7 8 9 0 1 2 3

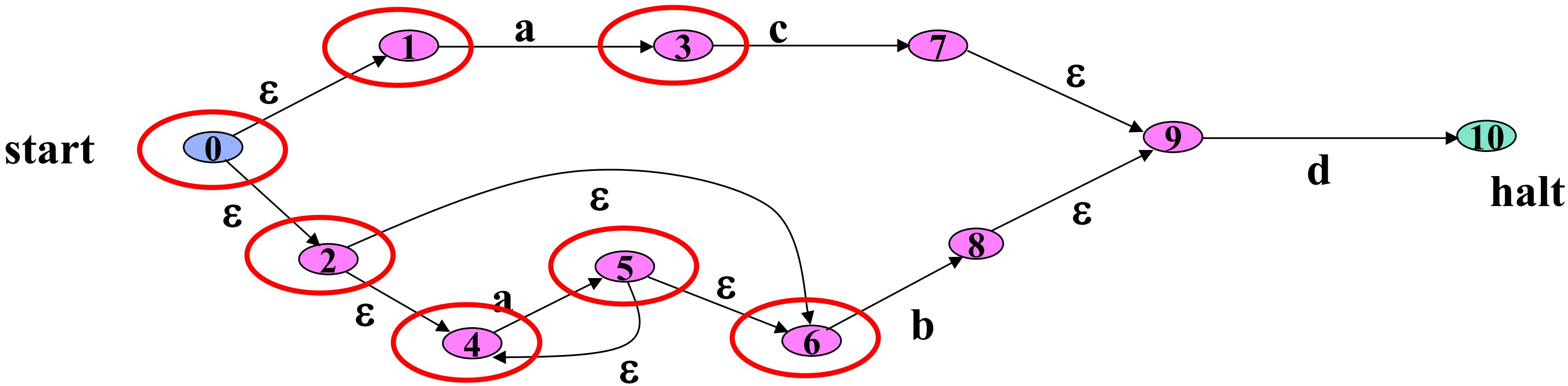
J: 8

T(J): a

Y: {0,3,5}

X: {0,1,2,4,6,7,9}

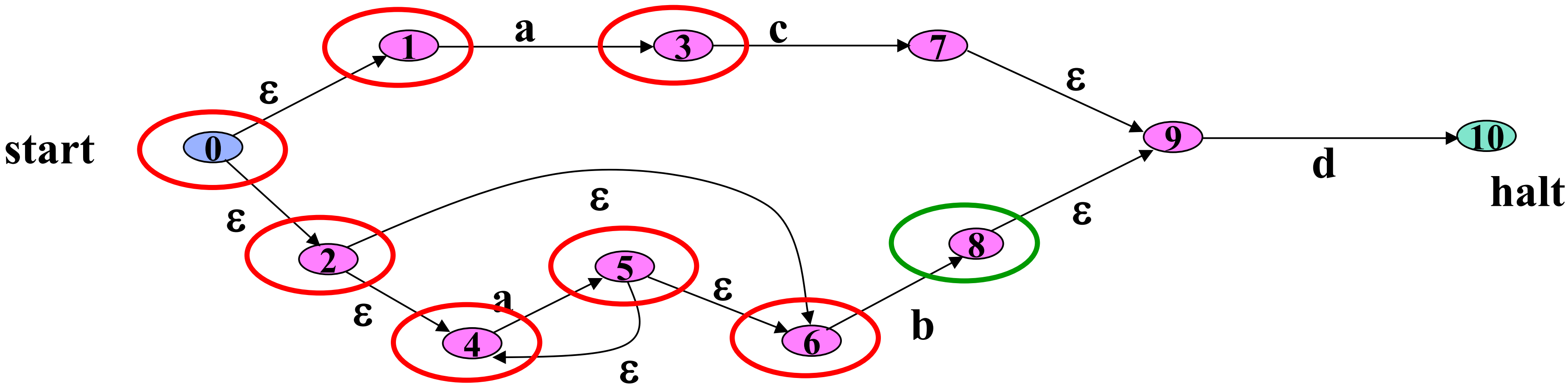
NDFA for (a*b|ac)d




```
T = c a a b c a c a b d a c d
    1 2 3 4 5 6 7 8 9 0 1 2 3
```

```
J:                    8
T(J):                 a
Y:                    {0,3,5}
X: {0,1,2,4,6,7,9}→{0,1,2,3,4,5,6}
```

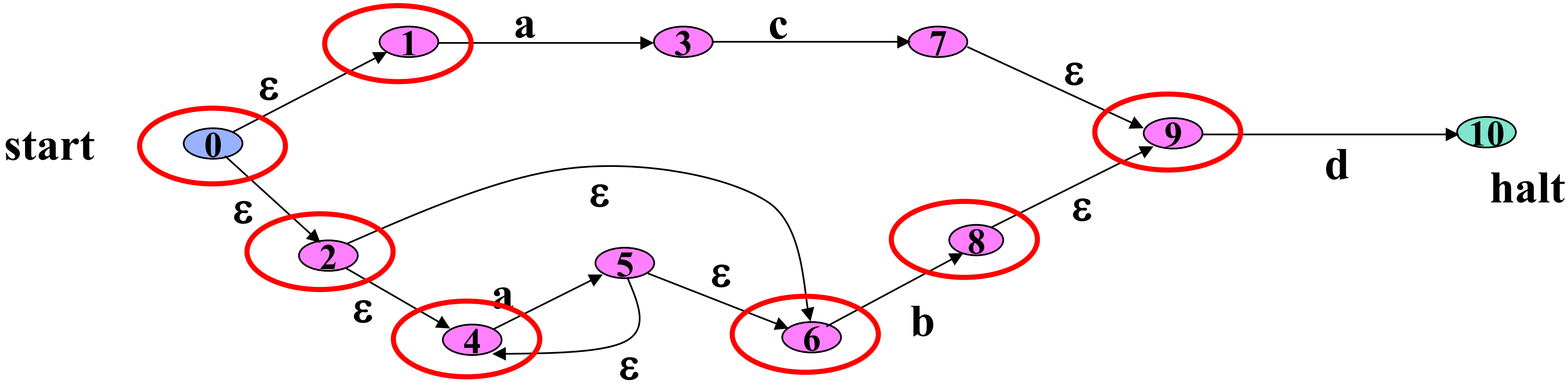
NDFA for (a*b|ac)d




```
T = c a a b c a c a b d a c d
    1 2 3 4 5 6 7 8 9 0 1 2 3

J:                    9
T(J):                 b
Y:                    {0,8}
X: {0,1,2,3,4,5,6}
```

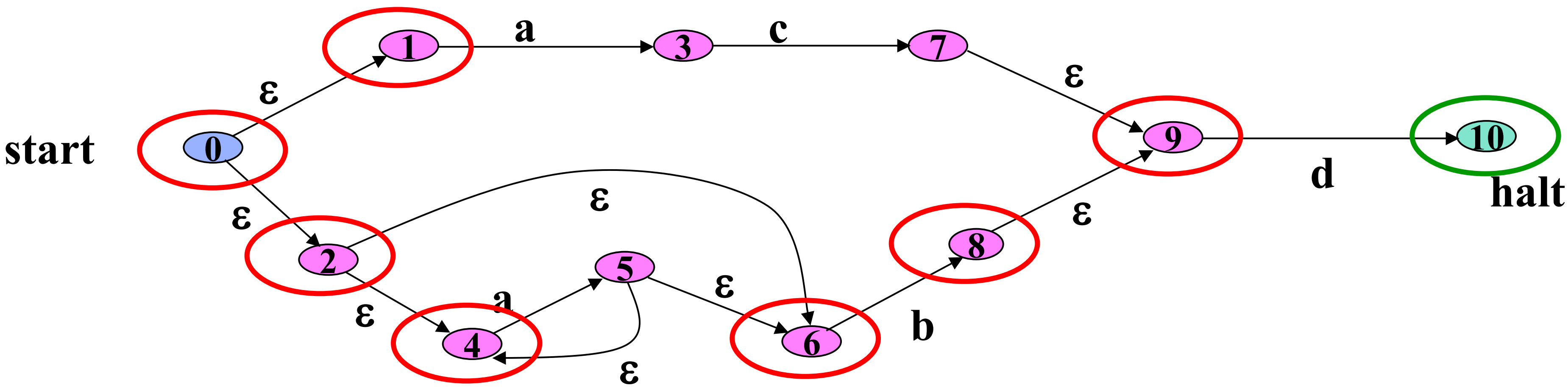
NDFA for `(a*b|ac)d`




```
T = c a a b c a c a b d a c d
    1 2 3 4 5 6 7 8 9 0 1 2 3

J:                     9
T(J):                  b
Y:                     {0,8}
X: {0,1,2,3,4,5,6}→{0,1,2,4,6,8,9}
```

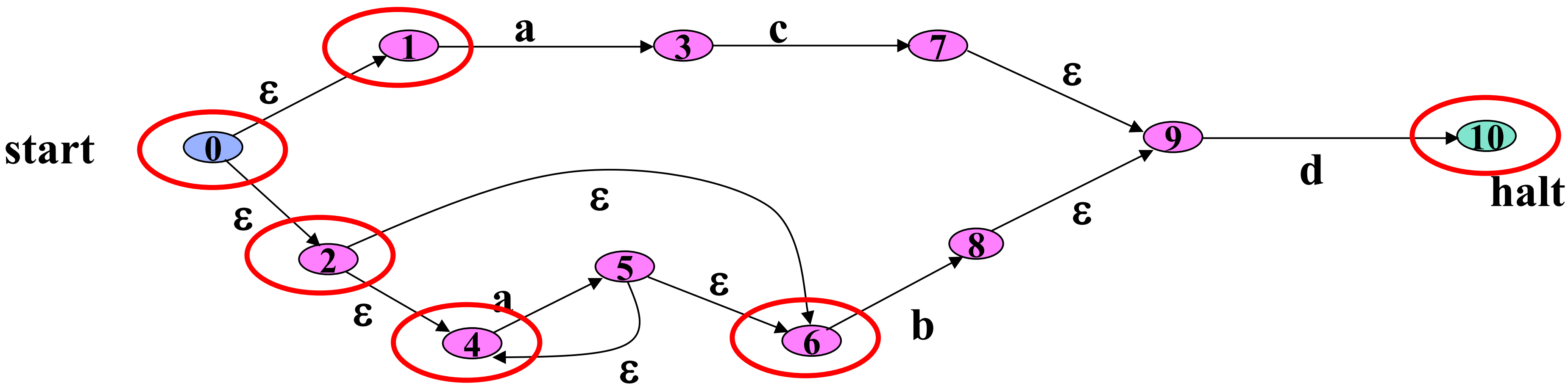
NDFA for `(a*b|ac)d`




```
T = c a a b c a c a b d a c d
    1 2 3 4 5 6 7 8 9 0 1 2 3
```

```
J:                      10
T(J):                   d
Y:                      {0,10}
X: {0,1,2,4,6,8,9}
```

NDFA for `(a*b|ac)d`




```
T = c a a b c a c a b d a c d
    1 2 3 4 5 6 7 8 9 0 1 2 3
```

```
J:                     10
T(J):                  d
Y:                     {0,10}
X: {0,1,2,4,6,8,9}→{0,1,2,4,6,10}
```

Analysis of NDFA simulation

- Let r = |`R`| and t = |T|
- Number of loop iterations is $\leq$ t
- With a suitable choice for type `SetOfStates`
  - `extendByChar` requires O(n) time, where n is the number of states
  - `extendByEpsilon` requires O(n+m) time by, say, depth first search, where m is the number of edges
  - By earlier observations, $n \leq 2r$ and $m \leq 2n \leq 4r$, so $O(m+n) = O(r)$

∴ Overall complexity is O(rt)