Session 3

- An Application of Finite Automate: Text Search
- Finite Automate's with ϵ-Transition
- Regular Expressions

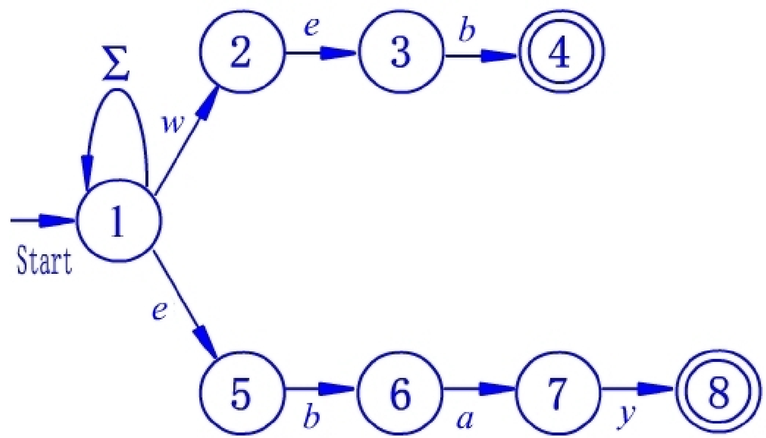
An Application of Finite Automate: Text Search

NFA for Text Search

We can design an NFA to recognizes a set of keywords in the text.

1. There is a start state with a transition to itself on every input symbol.
2. For each keyword $a_1a_2\cdots a_k$, there are k states, say $q_1, q_2, \cdots q_k$. There is a transition from the start state to q_1 on symbol a_1, a transition from q_1 to q_2 on symbol a_2, and so on. The state q_k is an accepting state and indicates that keyword $a_1a_2\cdots a_k$ has been found.

Example An NFA N recognizing occurrences of the words `web` and `ebay`.



Convert to an Equivalent DFA

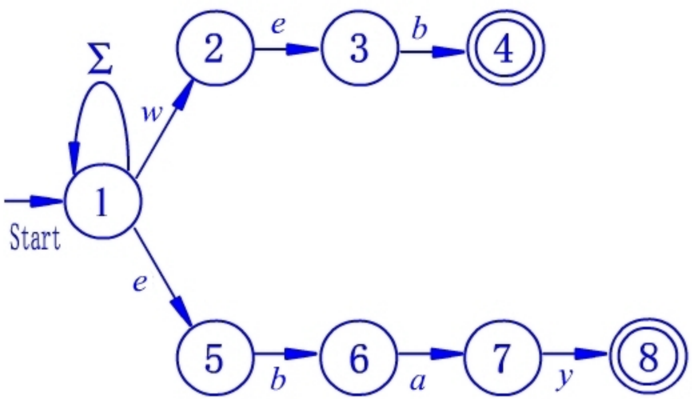
It's quite *common* in practice for the DFA to have roughly the same number of states as the NFA from which it is constructed.

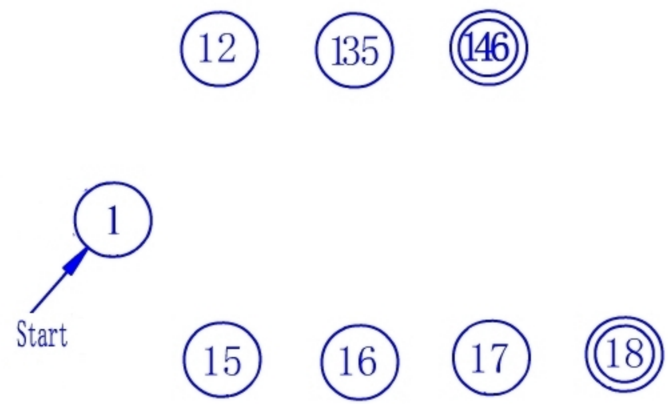
As an illustrating example, we can convert the NFA for text search to an equivalent DFA using subset construction.

The rules for constructing the set of DFA states:

1. If q_0 is the start state of the NFA, then $\{q_0\}$ is one of the states of the DFA.
2. Suppose p is one of the NFA states, and it is reached from q_0 along a path whose symbols are $a_1a_2\cdots a_m$. Then one of the DFA states is the set of NFA states consisting of:
   - q_0, p, and every other state of the NFA that is reachable from q_0 by following a path whose labels are a suffix of $a_1a_2\cdots a_m$, that is, any sequence of symbols of the form $a_ja_{j+1}\cdots a_m$.

Example The states of DFA D constructed by N using above rules.





Here 1 is shorthand for $\{1\}$, 12 for $\{1, 2\}$, and 135 for $\{1, 3, 5\}$, and so on.

The transition for each of the DFA states may be calculated by the formula:

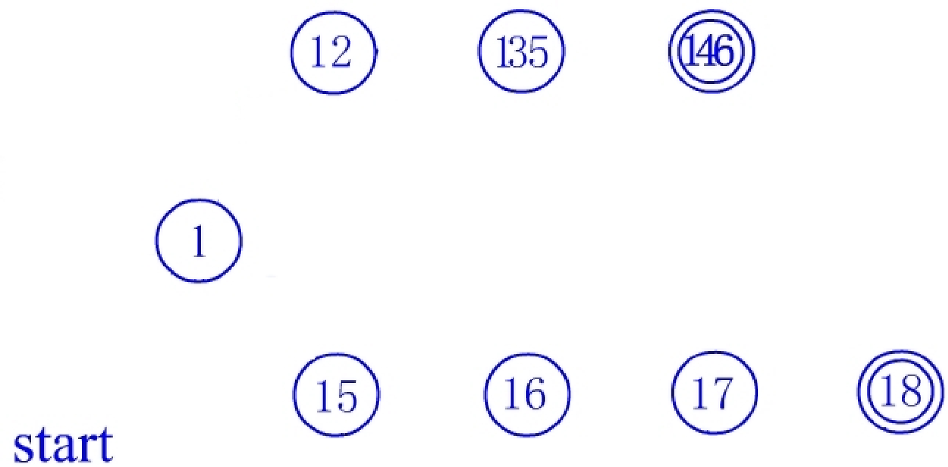
$$\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a).$$

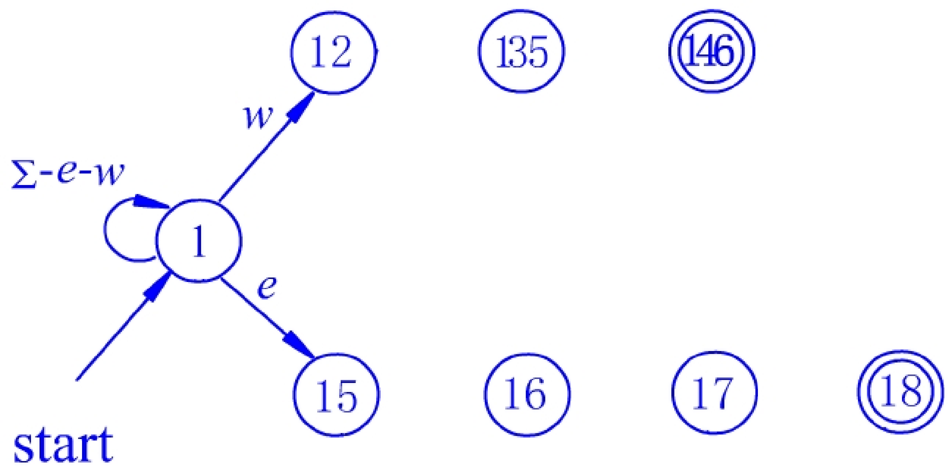
For example,

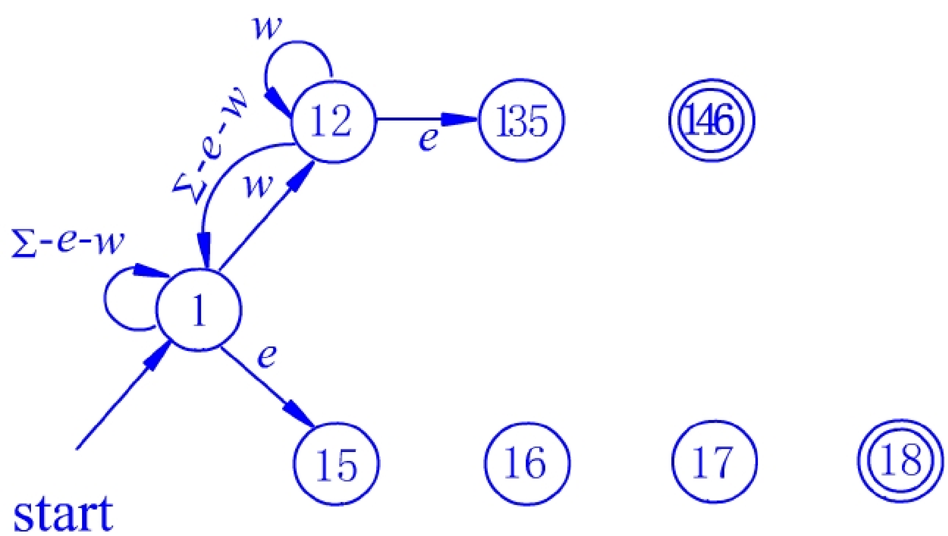
$$\delta_D(\{1\}, x) = \delta_N(1, x) = \{1\}, \quad (x \neq e,\ x \neq w)$$
$$\delta_D(\{1\}, w) = \delta_N(1, w) = \{1, 2\}, \quad \delta_D(\{1\}, e) = \delta_N(1, e) = \{1, 5\}.$$

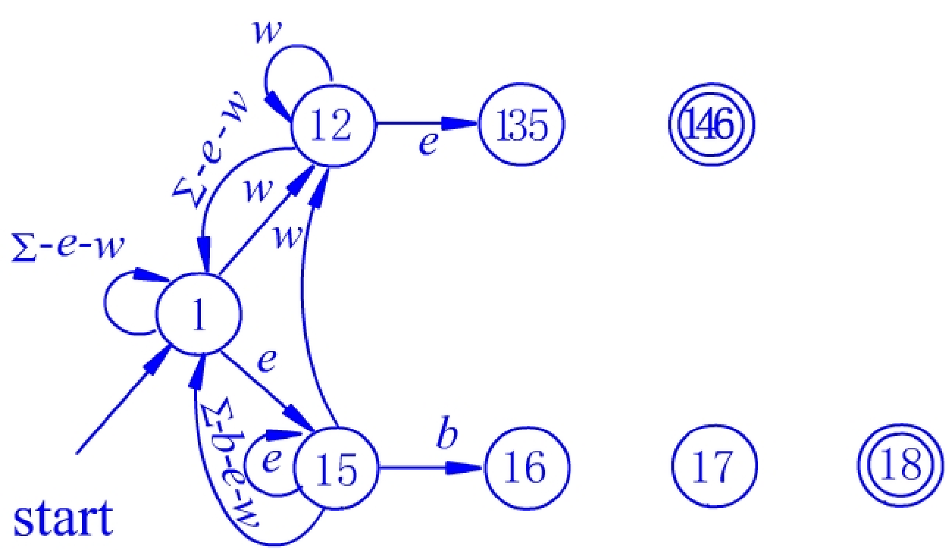
and

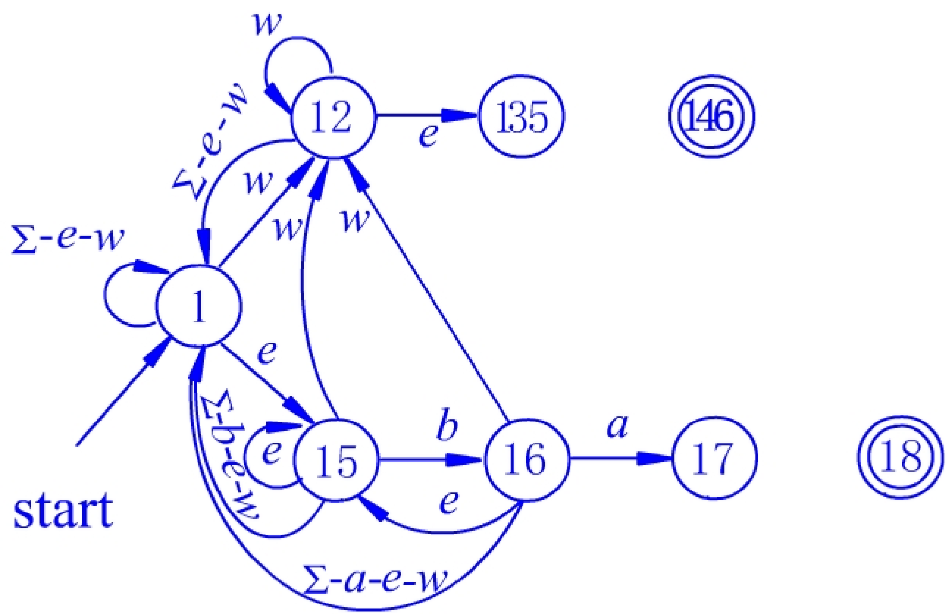
$$\delta_D(\{1, 2\}, x) = \delta_N(1, x) \cup \delta_N(2, x) = \{1\}, \quad (x \neq e,\ x \neq w)$$
$$\delta_D(\{1, 2\}, w) = \delta_N(1, w) \cup \delta_N(2, w) = \{1, 2\},$$
$$\delta_D(\{1, 2\}, e) = \delta_N(1, e) \cup \delta_N(2, e) = \{1, 5\} \cup \{3\} = \{1, 3, 5\}.$$

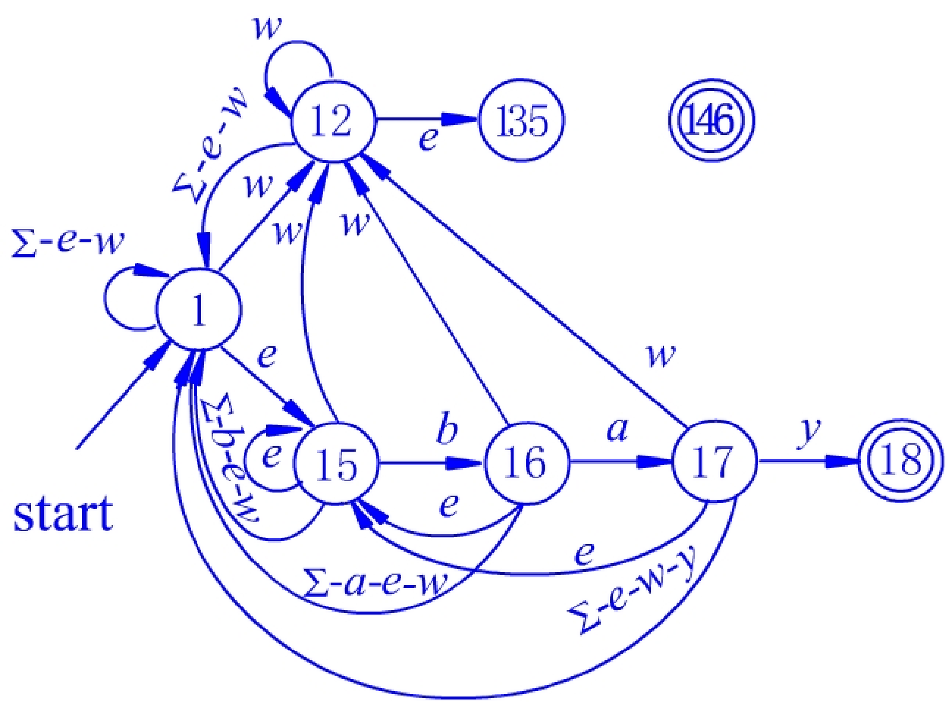

12
135
146
1
15
16
17
18
start


12
135
146
w
Σ-e-w
1
e
15
16
17
18
start

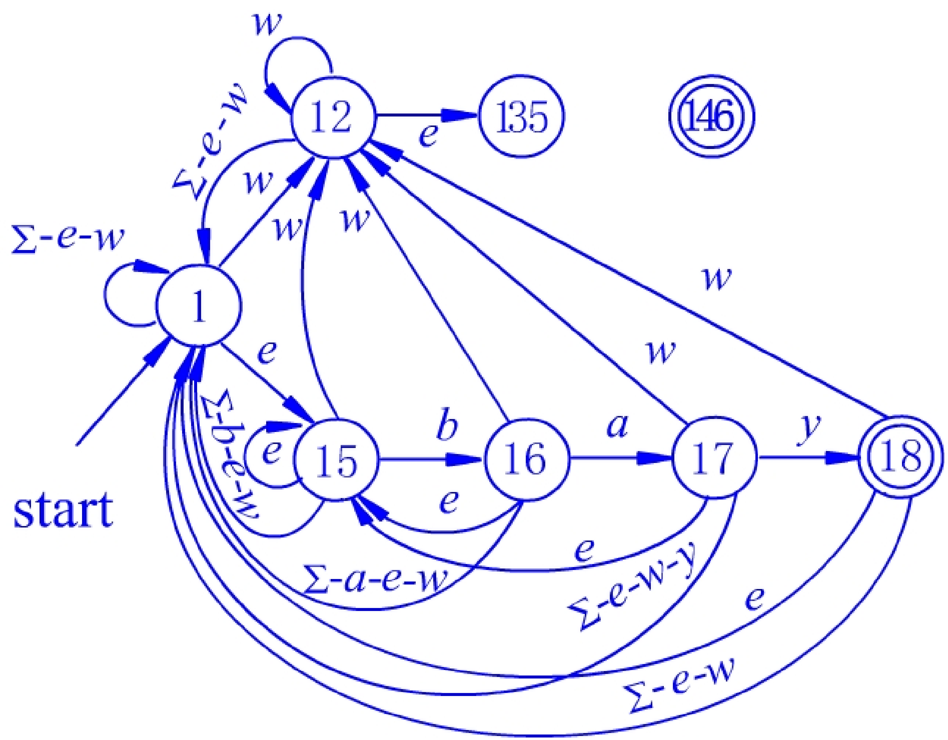

w
Σ-e-w
12
e
135
146
w
Σ-e-w
1
e
start
15
16
17
18

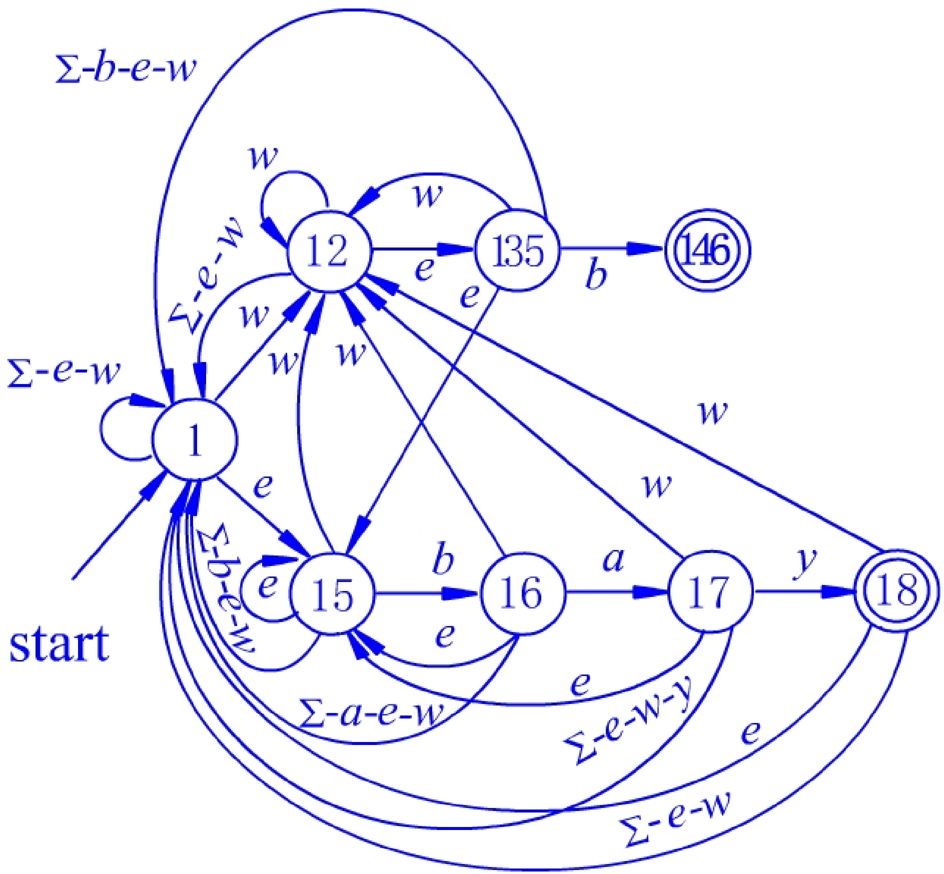

w
12
135
146
e
Σ-e-w
w
Σ-e-w
w
1
e
Σ-b-e-w
e
15
b
16
17
18
start

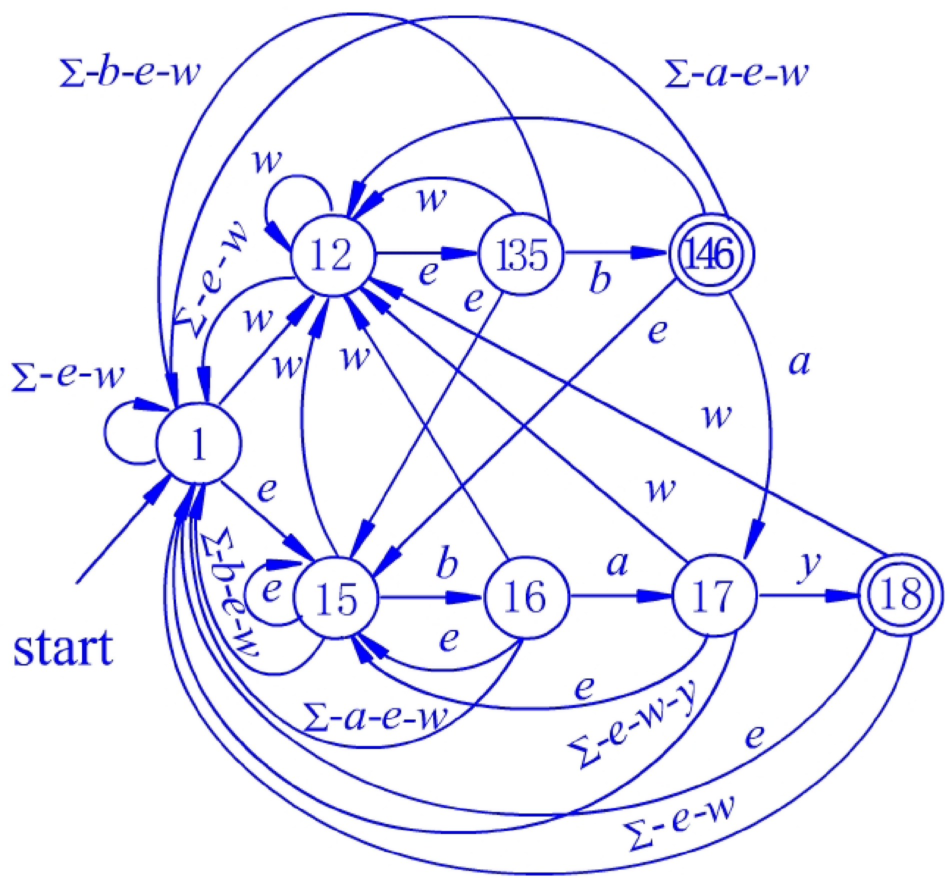

w
12
135
146
e
Σ-e-w
w
w
w
Σ-e-w
1
e
Σ-b-e-w
e
15
b
16
a
17
18
e
start
Σ-a-e-w



w
Σ-e-w
12
e
135
146
w
w
w
Σ-e-w
1
e
Σ-b-e-w
e
15
b
16
a
17
y
18
w
start
e
e
Σ-a-e-w
Σ-e-w-y


start
1
12
135
146
15
16
17
18
w
Σ-e-w
e
b
a
y
Σ-b-e-w
Σ-a-e-w
Σ-e-w-y

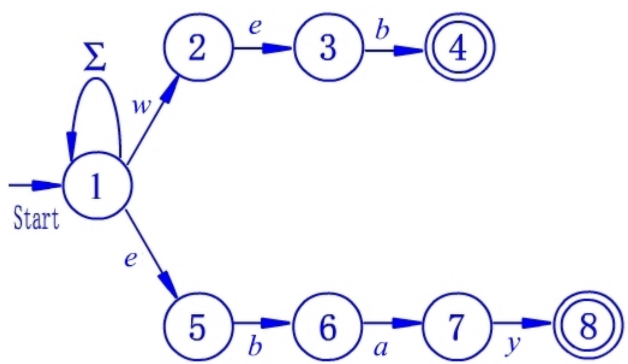

Σ-b-e-w
w
w
Σ-e-w
12
135
146
e
e
b
w
w
w
w
Σ-e-w
1
w
w
e
Σ-b-e-w
b
a
y
e
15
16
17
18
start
e
e
Σ-a-e-w
Σ-e-w-y
e
Σ-e-w

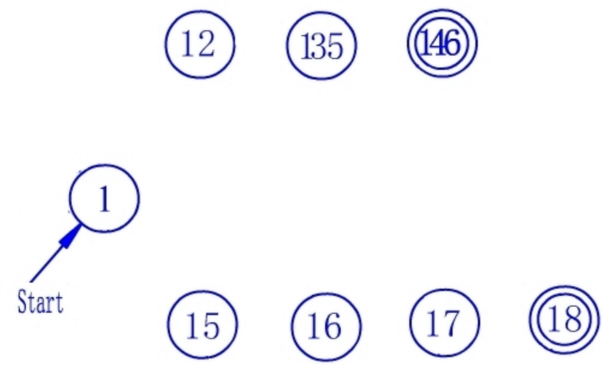


Σ-b-e-w
Σ-a-e-w
w
w
12
135
146
e
b
e
e
a
Σ-e-w
w
w
w
w
w
w
Σ-e-w
1
e
Σ-b-e-w
b
a
y
e
15
16
17
18
start
e
e
Σ-a-e-w
Σ-e-w-y
e
Σ-e-w

Question

1. For automaton recognizing keywords in text, when it will happen that the number of states of “subset” DFA is less than that of NFA?





2. Why the “rules for constructing DFA states” are follows from the subset construction and the strategy of “lazy evaluation”?

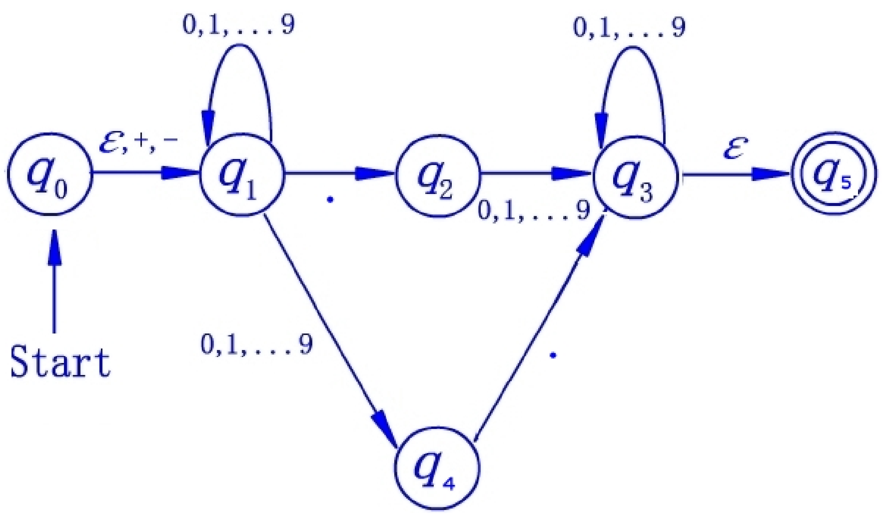
Finite Automate's with ϵ-Transition

Use of ϵ-Transition

In order to add “programming convenience”, we will extend NFA to ϵ-NFA. In transition diagrams of such NFA ϵ (the empty string) is allowed as a label.

However, this new capability does not expand the class of language that can be accepted by finite automata.

Example Design a automaton A_3 accepting decimal numbers consisting of (1) an optional + or – sign, (2) a sting of digits, (3) a decimal point, and (4) another strings of digits. One of the strings (2) and (4) are optional.



Definition of ϵ-NFA

A nondeterministic finite automate with ϵ-transition is a quintuple $(Q, \Sigma, \delta, q_0, F)$.

- Q is a finite set of *states*
- Σ is a finite set of *input symbols*, or a alphabet
- δ is a *transition function* from $Q \times \Sigma \cup \{\epsilon\}$ to the powerset of Q
- q_0 is a *start state*, a member of Q
- F is a set of *final* or *accepting* states, a subset of Q

Example The NFA A_3 is an ϵ-NFA, which can be represented as

$$A_3 = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{., +, -, 0, \cdots, 9\}, \delta, q_0, \{q_5\})$$

where the transition table for δ is

	ϵ	$+,-$	$.$	$1,\cdots,9$
$\rightarrow q_0$	$\{q_1\}$	$\{q_1\}$	$\emptyset$	$\emptyset$
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$	$\{q_1, q_4\}$
q_2	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_3\}$
q_3	$\{q_5\}$	$\emptyset$	$\emptyset$	$\{q_3\}$
q_4	$\emptyset$	$\emptyset$	$\{q_3\}$	$\emptyset$
$\star q_5$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Epsilon-Closures

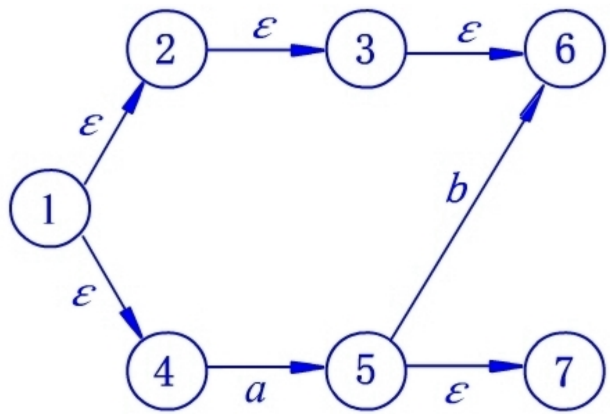
Informally, the ϵ-closures of a state q is a set of the state q and other states by following all transitions out of q that are labeled ϵ.

The recursive definition of the ϵ-closures ECLOSE(q)

Basis step: $q \in \text{ECLOSE}(q)$.

Inductive step: If $p \in \text{ECLOSE}(q)$, then $\delta(p, \epsilon) \in \text{ECLOSE}(q)$.

Example For the collection of states, which may be part of some ϵ-NFA, construct ECLOSE(1) and ECLOSE(5).



Solution ECLOSE(1) = {1, 2, 3, 4, 6}, ECLOSE(5) = {5, 7}

Extended Transition Function

The transition function δ of an ϵ-NFA can be extended to $\hat{\delta}$:

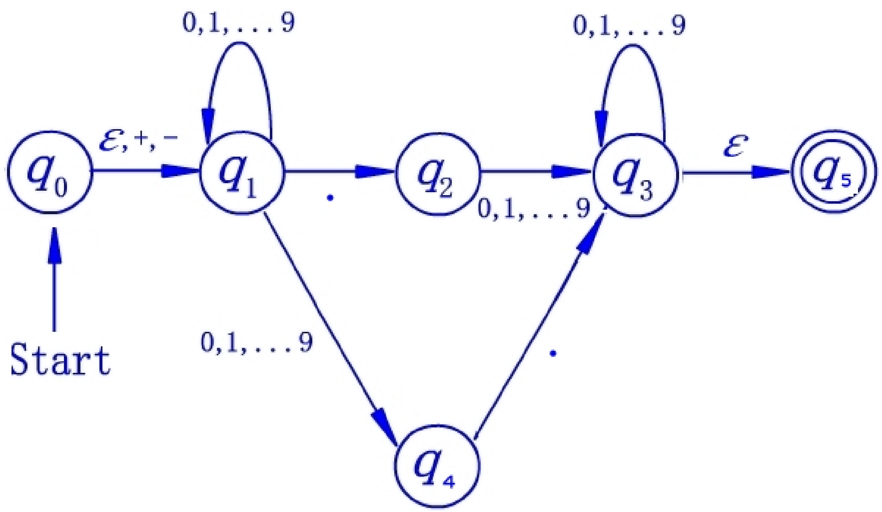
Basis step: $\hat{\delta}(q, \epsilon) = \text{ECLOSE}(q)$.

Inductive step: Suppose $w = xa$, then

$$\hat{\delta}(q, w) = \bigcup_{r \in \delta(\hat{\delta}(q,x),a)} \text{ECLOSE}(r)$$

where $\delta(S, a) = \bigcup_{p \in S} \delta(p, a)$.

Example Compute $\hat{\delta}(q_0, 5.6)$ for the ϵ-NFA A_3



- $\hat{\delta}(q_0, \epsilon) = \text{ECLOSE}(q_0) = \{q_0, q_1\}$.
- Since $\delta(q_0, 5) \cup \delta(q_1, 5) = \{q_1, q_4\}$, $\hat{\delta}(q_0, 5) = \text{ECLOSE}(q_1) \cup \text{ECLOSE}(q_4) = \{q_1, q_4\}$.
- Since $\delta(q_1, .) \cup \delta(q_4, .) = \{q_2, q_3\}$, $\hat{\delta}(q_0, 5.) = \text{ECLOSE}(q_2) \cup \text{ECLOSE}(q_3) = \{q_2, q_3, q_5\}$.
- Since $\delta(q_2, 6) \cup \delta(q_3, 6) \cup \delta(q_5, 6) = \{q_3\}$, $\hat{\delta}(q_0, 5.6) = \text{ECLOSE}(q_3) = \{q_3, q_5\}$.

◀

Language of ϵ-NFA

The language of an ϵ-NFA $A = (Q, \Sigma, \delta, q_0, F)$ is defined by

$$L(A) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

Example For $A_3 = (\{q_0, q_1, \cdots, q_5\}, \{., +, -, 0, \cdots, 9\}, \delta, q_0, \{q_5\})$, since $\hat{\delta}(q_0, 5.6)$ contains the accepting state $\{q_5\}$, so the string 5.6 is in the language of that ϵ-NFA.

Equivalence of DFA and ϵ-NFA

Given any ϵ-NFA E, we can find a DFA D that accepts the same language as E. The construction is very close to the subset construction, as the states of D are subsets of the states of E.

Let ϵ-NFA $E = (Q_E, \Sigma, \delta_E, q_0, F_E)$, we will construct an equivalent DFA $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$.

Here is the detail of the construction.

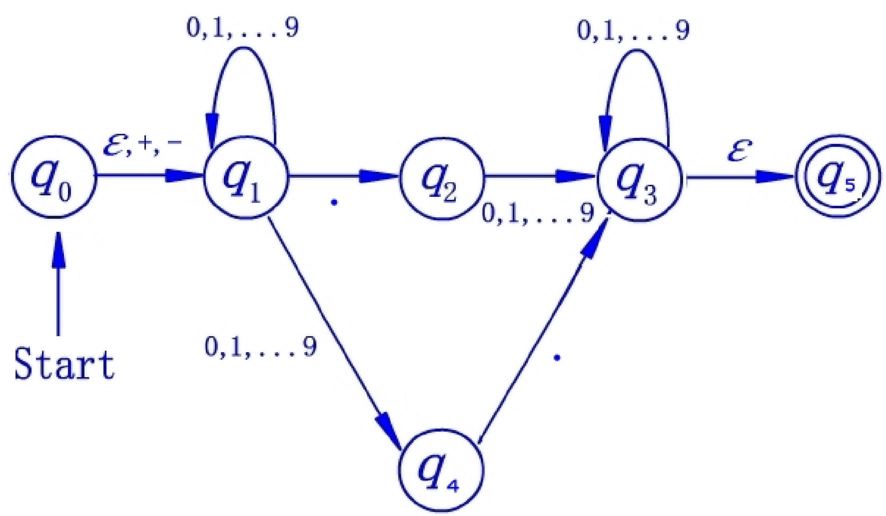
- $Q_D = \{S \mid S \subseteq Q_E \text{ and } S = \text{ECLOSE}(S)\}$ (accessible states)
- $q_D = \text{ECLOSE}(q_0)$
- $F_D = \{S \mid S \in Q_D, S \cap F_E \neq \emptyset\}$
- For every $S \in Q_D$ and $a \in \Sigma$,

$$\delta_D(S, a) = \bigcup_{r \in \delta_E(S,a)} \text{ECLOSE}(r)$$

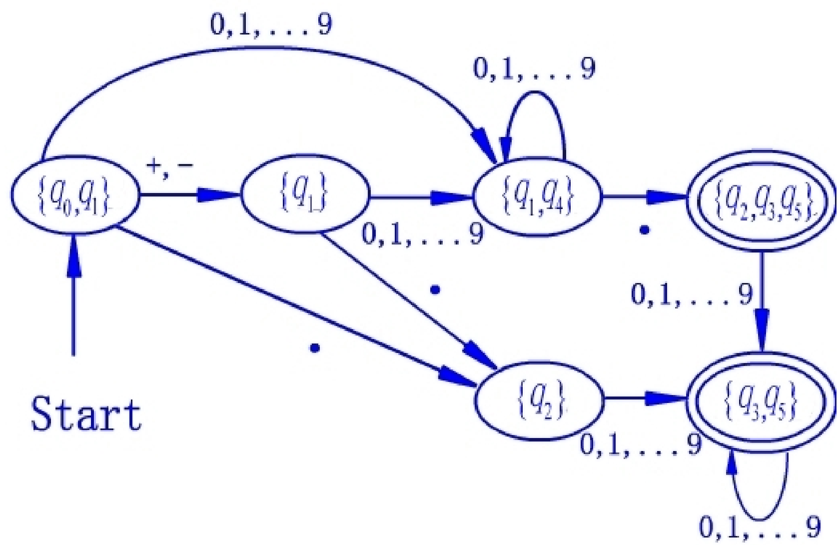
where $\delta_E(S, a) = \bigcup_{p \in S} \delta_E(p, a)$.

☞ In general, $|Q_D| \neq 2^{|Q_E|}$.

Example Eliminate ϵ-transition and construct an DFA from the ϵ-NFA A_3



Solution Omitted the dead state $\emptyset$ and all transitions to the dead state.



Theorem 2.3 *A language L is accepted by some ϵ-NFA if and only if L is accepted by some DFA.*

Proof (if) This direction is easy. Suppose $L = L(D)$ for some DFA. Turn D into an ϵ-NFA E by adding transition $\delta(q, \epsilon) = \emptyset$ for all states q of D.

(only-if) Let $E = (Q_E, \Sigma, \delta_E, q_0, F_E)$ is an ϵ-NFA. Apply the modified subset construction described above to produce the DFA $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$. We show

$$\hat{\delta}_E(q_0, w) = \hat{\delta}_D(q_D, w)$$

by induction on $|w|$.

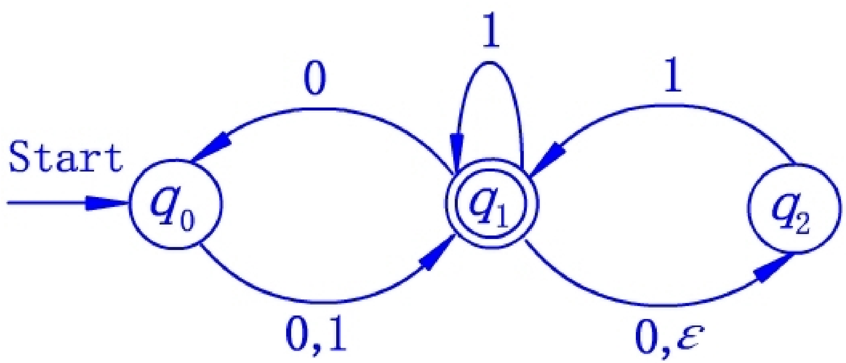
Basis step: $\hat{\delta}_E(q_0, \epsilon) = \text{ECLOSE}(q_0) = q_D = \hat{\delta}_D(q_D, \epsilon)$.

Inductive step:

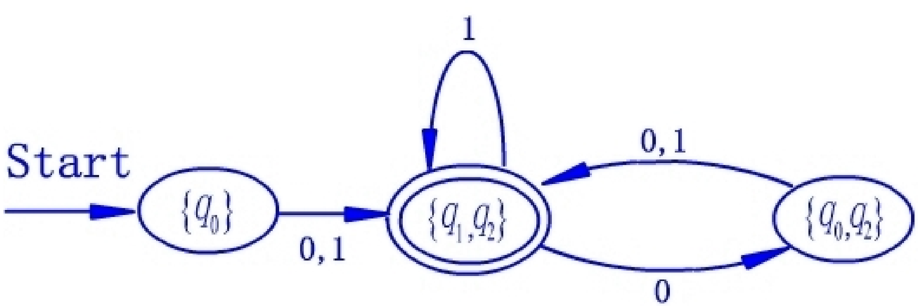
$$\begin{aligned}
\hat{\delta}_E(q_0, xa) &= \bigcup_{p \in \delta_E(\hat{\delta}_E(q_0,x),a)} \text{ECLOSE}(p) && \text{(definition of } \hat{\delta}_E\text{)} \\
&= \bigcup_{p \in \delta_E(\hat{\delta}_D(q_D,x),a)} \text{ECLOSE}(p) && \text{(induction hypothesis)} \\
&= \delta_D(\hat{\delta}_D(q_D, x), a) && \text{(modified subset construction)} \\
&= \hat{\delta}_D(q_D, xa) && \text{(definition of } \hat{\delta}_D\text{)}
\end{aligned}$$

◀

Example Convert the following ϵ-NFA into an equivalent DFA.



Solution   Apply the modified subset construction, we have




Break for 15 minutes

Regular Expressions

Regular expressions denote languages, e.g.

$$\mathbf{01}^* + \mathbf{10}^*$$

- A regular expression is a "user-friendly", declarative way of describing a regular language.
- A FA (DFN and NFA) is a "blueprint" for constructing a machine recognizing a regular language.

Building Regular Expressions

Inductive definition of regular expression (RE) and its language.

Basis step:

- ϵ and $\emptyset$ are regular expression.

$$L(\epsilon) = \{\epsilon\}, \quad L(\emptyset) = \emptyset$$

- If $a \in \Sigma$, then $\mathbf{a}$ is a regular expression.

$$L(\mathbf{a}) = \{a\}$$

Inductive step:

- If E is a regular expression, then (E) is a regular expression.

$$L((E)) = L(E)$$

- If E and F are regular expression, then $E + F$ is a regular expression.

$$L(E + F) = L(E) \cup L(F)$$

- If E and F are regular expression, then $E.F$ (or EF) is a regular expression.

$$L(E.F) = L(E).L(F)$$

- If E is a regular expression, then E^* is a regular expression.

$$L(E^*) = (L(E))^*$$

Precedence of Regular Expression Operators

1. The star operator is of highest precedence.
2. Next in precedence comes the concatenation or dot operator.
3. Finally, all unions or plus operators are grouped with their operands.

Example $\mathbf{01}^* + \mathbf{1}$ is grouped $(\mathbf{0}(\mathbf{1})^*) + \mathbf{1}$.

Languages Associated with Regular Expressions

Example The expression $R = (\mathbf{aa})^*(\mathbf{bb})^*\mathbf{b}$ denoted the set of all strings with an even number of a's followed by an odd number of b's; that is

$$L(R) = \{a^{2n}b^{2m+1} \mid n \geq 0, m \geq 0\}.$$

Example Exhibit the language $L(\mathbf{a}^* \cdot (\mathbf{a} + \mathbf{b}))$ in set notation.

Solution Since $L(\mathbf{a}^* \cdot (\mathbf{a} + \mathbf{b})) = L(\mathbf{a}^*)(L(\mathbf{a} + \mathbf{b})) = (L(\mathbf{a}))^*(L(\mathbf{a}) \cup L(\mathbf{b}))$. So the answer is $\{a, aa, aaa, \cdots, b, ab, aab, \cdots\}$.

Going from an informal description or set notation to a regular expression tends to be a little harder.

Example Write a regular expression for the set of strings that consist of alternating 0's and 1's.

Solution
$$(\mathbf{01})^* + (\mathbf{10})^* + \mathbf{1}(\mathbf{01})^* + \mathbf{0}(\mathbf{10})^*,$$

or equivalently
$$(\epsilon + \mathbf{1})(\mathbf{01})^*(\epsilon + \mathbf{0}).$$

Example Find a regular expression for the language $L = \{x \in \{0,1\}^* \mid w \text{ has no pair of consecutive zeros}\}$.

Solution One observation is that whenever a 0 occurs, it must be followed immediately by a 1. This suggests $(\mathbf{1}^*\mathbf{011}^*)^*$. Right? No.

The strings ending in 0 or consisting of all 1's are unaccounted for. So the correct answer is

$$(\mathbf{1}^*\mathbf{011}^*)^*(\mathbf{0}+\epsilon)+\mathbf{1}^*(\mathbf{0}+\epsilon).$$

Another shorter answer is $(\mathbf{1}+\mathbf{01})^*(\mathbf{0}+\epsilon)$.

Example Find a regular expression that denotes all bit strings whose value, when interpreted as binary integer, is great than or equal to 40.

Solution The bit string must be at least 6 bits long. If it is longer than 6 bits, its value is at least 64, so anything will do. If it is exactly 6 bits, then either the second bit from the left (16) or the third bit from the left (8) must be 1. So the solution is

$$(\mathbf{111} + \mathbf{110} + \mathbf{101})(\mathbf{0} + \mathbf{1})(\mathbf{0} + \mathbf{1})(\mathbf{0} + \mathbf{1}) + \mathbf{1}(\mathbf{0} + \mathbf{1})(\mathbf{0} + \mathbf{1})(\mathbf{0} + \mathbf{1})(\mathbf{0} + \mathbf{1})(\mathbf{0} + \mathbf{1})(\mathbf{0} + \mathbf{1})(\mathbf{0} + \mathbf{1})^*$$

Regular Expressions in UNIX

We introduce the UNIX notation for extended regular expressions. This notation gives us a number of additional capabilities.

UNIX regular expressions allow one to write character classes to present ASCII characters as succinctly as possible. The rules for character classes are:

- The symbol $\cdot$ (dot) stands for any character.

- The sequence $[a_1 a_2 \cdots a_k]$ stands for the regular expression $\mathbf{a_1} + \mathbf{a_2} + \cdots + \mathbf{a_k}$.

- Between the square brace one can put a range of the form $x - y$ to mean all the characters from x to y in the ASCII sequence. e.g., the digits can be expressed [0-9], the upper-case letters [A-Z].

- There are special notations for several of most common classes of characters. [:digit:] stands [0-9]; [:alpha:] stands [A-Za-z]; [:alnum:] stands [A-Za-z0-9].

There are several operators that are used in UNIX regular expressions.

- The operator | is used in place of + to denote union.
- The operator ? means "zero or one of". Thus, $R?$ in UNIX is the same as $\epsilon + R$.
- The operator + means "one or more of". Thus, $R+$ in UNIX is shorthand for RR^*.
- The operator $\{n\}$ means "n copies of". Thus, $R\{5\}$ in UNIX is shorthand for $RRRRR$.

Finding Patterns in Text

The general problem for which regular-expression technology has been found useful is the description of a vaguely defined class of patterns in text.

Suppose that we want to scan a very large number of Web pages and *detect* addresses. We have to develop expression for street addresses. If we use UNIX-style notation, we have:

```
[0-9]+[A-Z]?_[A-Z][a-z]*(_[A-Z][a-z]*)*_(Street|St\.|Avenue|Ave\.|Road|Rd\.)
```

```
[0-9]+[A-Z]?_[A-Z][a-z]*(_[A-Z][a-z]*)*_(Street|St\.|Avenue|Ave\.|Road|Rd\.)
```

For example,

```
56 Rhode Island Avenue
123A Main St.
3663 Zhongshan North Road
```

If we work with this expression, we shall do fairly well. However, we shall eventually discover that we are missing:

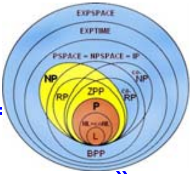
1. Streets that are called something other than a street, avenue, or road. For example, we shall miss “Place,” and “Way.”

2. Street names that are numbers, or partially numbers, like “42nd Street.”

3. Street names that don’t end in anything like “Street”. For example, “El Camino Real.”

Question Modify the expression developed to include all the mentioned options.

Thank you!


EXPSPACE
EXPTIME
PSPACE = NPSPACE = IP
NP
co-NP
RP
ZPP
co-RP
P
L
BPP