Introduction to the Theory of Computation

XU Ming

School of Software Engineering, East China Normal University

March 4, 2025

OUTLINE

- The equivalence of DFA and NFA
- An Application of Finite Automata: Text Search

Equivalence of DFA and NFA

Equivalence of DFA and NFA

There are many languages for which an NFA is easier to be constructed than a DFA. That means NFA's are usually easier to "program" in.

However, surprisingly, for any NFA N there is a DFA D such that $L(D) = L(N)$, and vise verse.

The proof that DFA's can do whatever NFA's can do involves an important "construction" called the subset construction.

The subset construction starts from an NFA $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Its goal is the description of a DFA $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that

$$L(D) = L(N).$$

Here is the detail of the construction.

- $Q_D = \{S \mid S \subseteq Q_N\}$;
- $F_D = \{S \mid S \subseteq Q_N, S \cap F_N \neq \emptyset\}$;
- For every $S \subseteq Q_N$ and $a \in \Sigma$,

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

Note that $|Q_D| = 2^{|Q_N|}$, although most states in Q_D are likely to be garbage.

Example

Let's construct δ_D from NFA $A_2 = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$ where δ is the transition function

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
$\bigstar q_2$	$\emptyset$	$\emptyset$

Since the state set of N is $\{q_0, q_1, q_2\}$, the subset construction produces a DFA with $2^3 = 8$ states, corresponding to all the subsets of these three states. Next slide shows the transition table for these eight states.

	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	$\emptyset$	$\{q_2\}$
$\bigstar\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\bigstar\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\bigstar\{q_1, q_2\}$	$\emptyset$	$\{q_2\}$
$\bigstar\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

To make the point clearer, we can invent new names for these states, e.g., A for $\emptyset$, B for $\{q_0\}$, and so on.

	0	1
A	A	A
$\rightarrow B$	E	B
C	A	D
$\bigstar D$	A	A
E	E	F
$\bigstar F$	E	B
$\bigstar G$	A	D
$\bigstar H$	E	F

Starting in the start state B, we can only reach states B, E, and F. The other five states are inaccessible from the start state and may as well not be there.

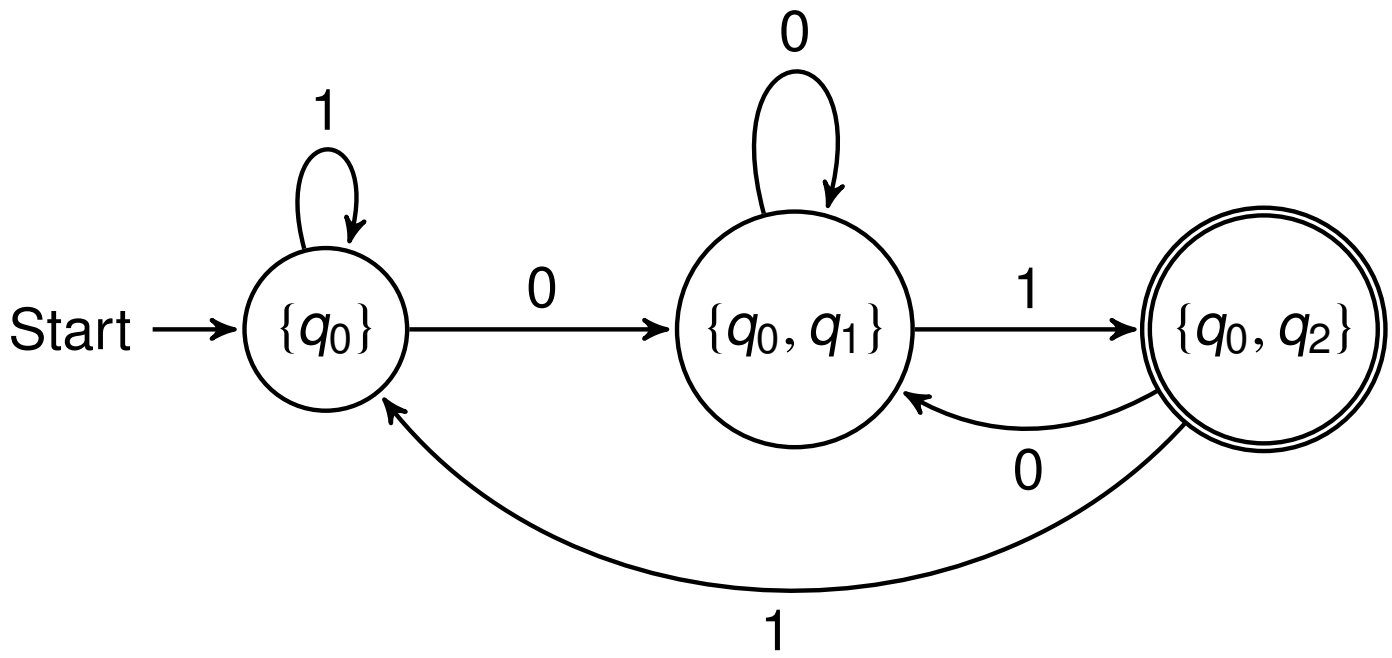
We can often avoid the exponential blow-up by constructing the transition table for DFA D only for accessible states S as follows (lazy evaluation):

Basis step: $S = \{q_0\}$ is accessible in DFA D.

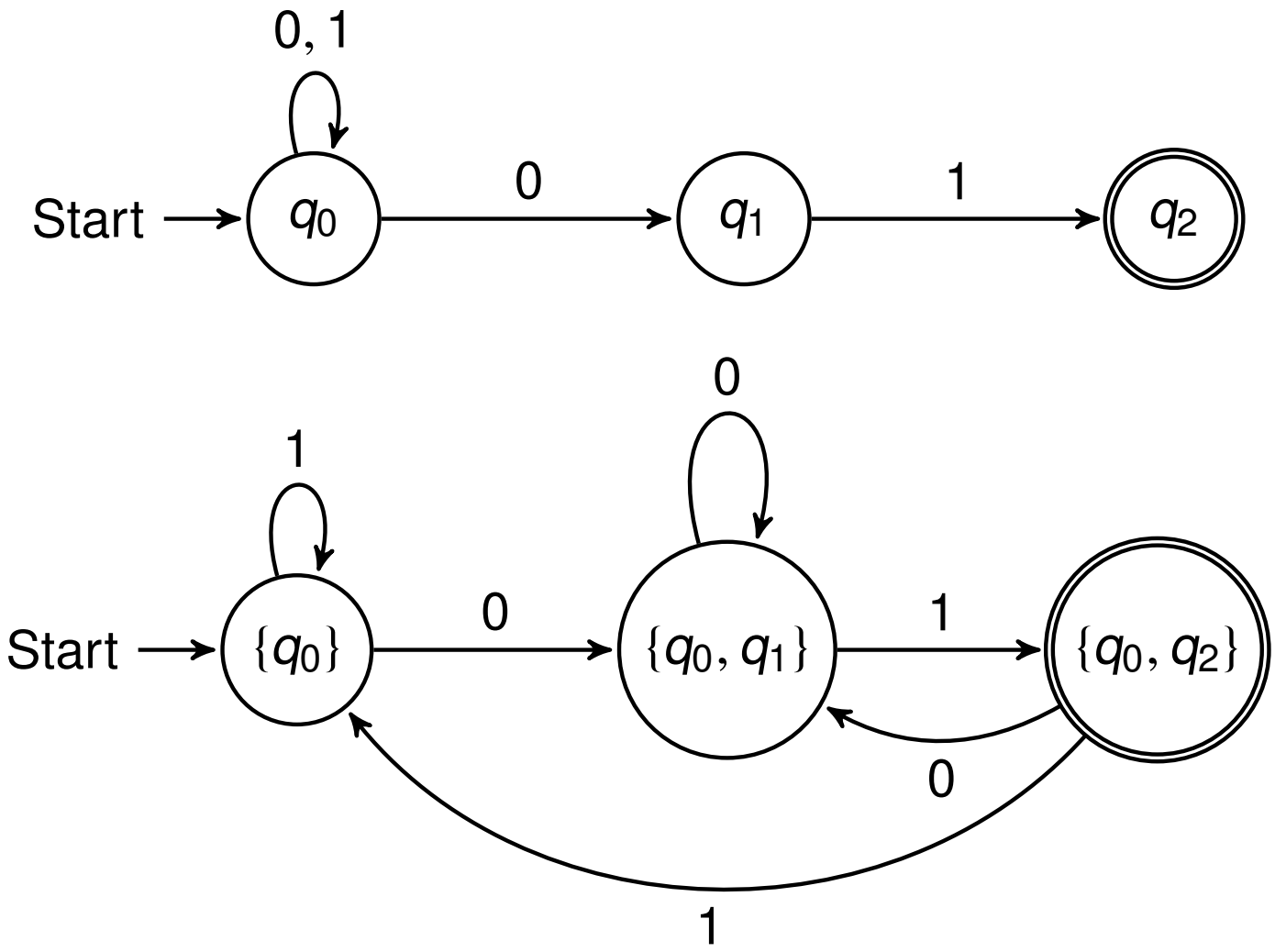
Inductive step: If S is accessible, so are the states $\delta_D(S, a)$ for any $a \in \Sigma$.

In this way, we obtain the "subset" DFA with accessible states only.

	0	1
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\bigstar\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$



Comparing the DFA and NFA, we find that they have same number of states, but different number of transitions.



Although the intuition was suggested by the examples, we need to show formally that the subset construction works!

Theorem 2.1

Let $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ be the DFA constructed from NFA $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ by the subset construction, then $L(D) = L(N)$.

Proof What we actually prove, by induction on $|w|$, that is

$$\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w).$$

Basis step: Let $|w| = 0$, i.e. $w = \epsilon$. It follows from the definition of $\hat{\delta}$.

Inductive step: Let $|w| = n + 1$, and assume the statement is true for length n. Split w as $w = xa$,

$$\begin{aligned}
\hat{\delta}_D(\{q_0\}, xa) &= \delta_D(\hat{\delta}_D(\{q_0\}, x), a) && \text{(definition of } \hat{\delta}_D\text{)} \\
&= \delta_D(\hat{\delta}_N(q_0, x), a) && \text{(induction hypothesis)} \\
&= \bigcup_{p \in \hat{\delta}_N(q_0, x)} \delta_N(p, a) && \text{(definition of } \delta_D\text{)} \\
&= \hat{\delta}_N(q_0, xa) && \text{(definition of } \hat{\delta}_N\text{)}
\end{aligned}$$

□

Theorem 2.2

A language L is accepted by some DFA if and only if L is accepted by some NFA.

Proof (if) This part is Theorem 2.1.
(only-if) Note that any DFA can be converted to an equivalent NFA by modifying the δ_D to δ_N by the rule

- If $\delta_D(q, a) = p$, then $\delta_N(q, a) = \{p\}$.

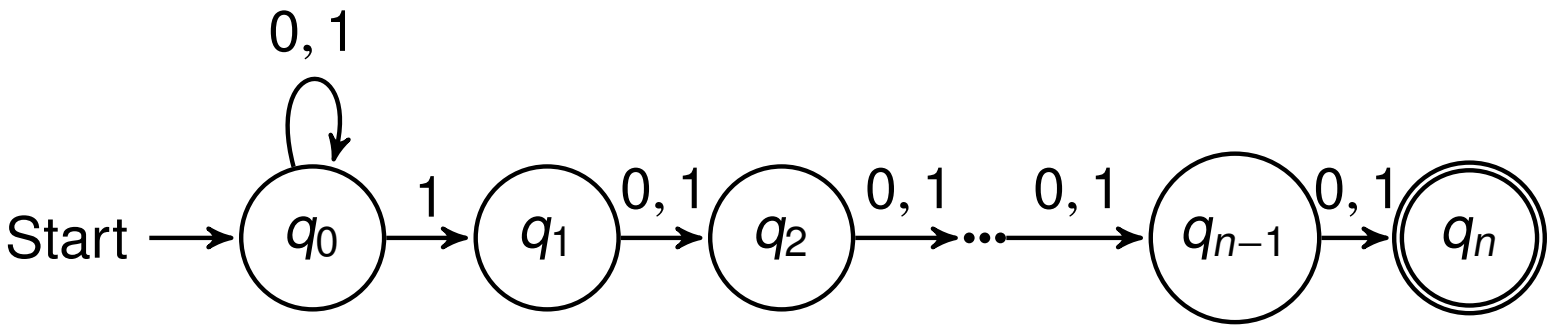
By induction on $|w|$, it will be shown in the tutorial that if $\hat{\delta}_D(q_0, w) = p$, then $\hat{\delta}_N(q_0, w) = \{p\}$. The claim of the theorem follows. □

Bad Case for the Subset Construction

For DFA constructed by NFA, exponential growth in the number of states is possible.

Example

There is an NFA N with $n + 1$ states that has no equivalent DFA with fewer than 2^n states.



Clearly, $L(N) = \{x1c_2c_3 \cdots c_n \mid x \in \{0,1\}^*, c_i \in \{0,1\}\}$. Intuitively, a DFA D that accepts $L(N)$ must remember the last n symbols it has read.

Since there are 2^n bit sequences $a_1 a_2 \cdots a_n$, if D with fewer than 2^n states exists, then there would be some state q such that D can be in state q after reading two different sequences of n bits, say $a_1 a_2 \cdots a_n$ and $b_1 b_2 \cdots b_n$.

Since the sequences are different, there is an i such that $a_i \neq b_i$. Suppose (by symmetry) that $a_i = 1$ and $b_i = 0$.

Two cases will be consider.

1. If $i = 1$, i.e., the sequences are $1a_2 \cdots a_n$ and $0b_2 \cdots b_n$, then q must be both an accepting state and a nonaccepting state.
2. If $i > 1$, i.e., the sequences are $a_1 \cdots a_{i-1}1a_{i+1} \cdots a_n$ and $a_1 \cdots a_{i-1}0b_{i+1} \cdots b_n$, then consider the state p that D enters after reading $a_1 \cdots a_{i-1}1a_{i+1} \cdots a_n c_1 \cdots c_{i-1}$ and $a_1 \cdots a_{i-1}0b_{i+1} \cdots b_n c_1 \cdots c_{i-1}$. Then p must be both accepting and nonaccepting.

The claim follows by contradiction.

An Application of Finite Automata: Text Search

NFA for Text Search

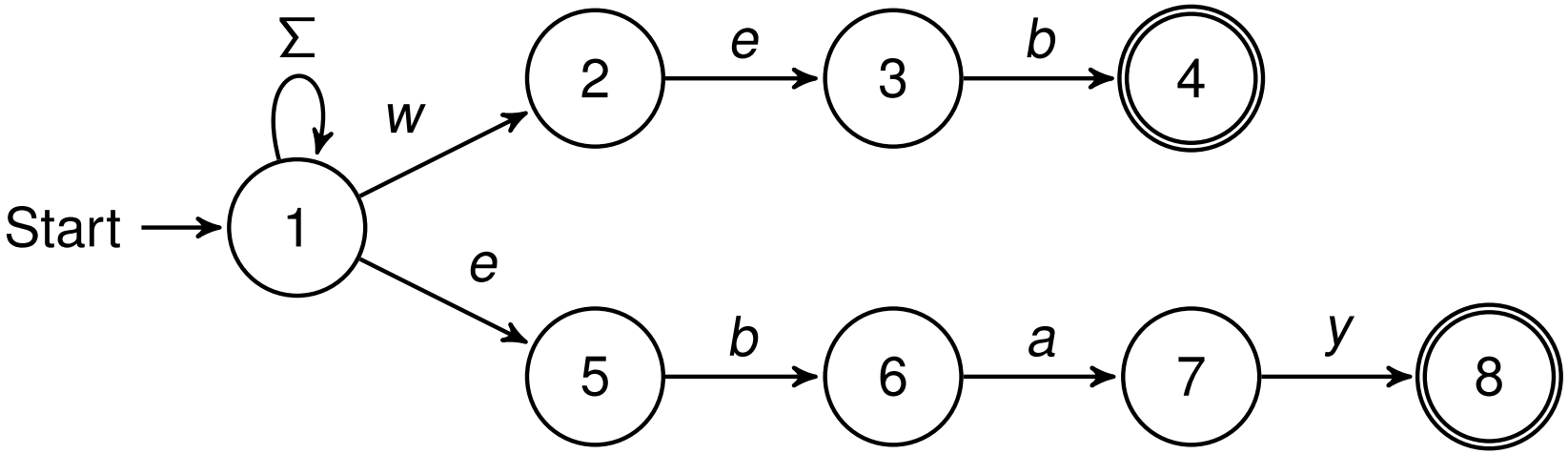
We can design an NFA to recognize a set of keywords in the text.

1. There is a start state with a transition to itself on every input symbol.

2. For each keyword $a_1 a_2 \cdots a_k$, there are k states, say $q_1, q_2, \ldots, q_k$. 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 $a_1 a_2 \cdots a_k$ 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.

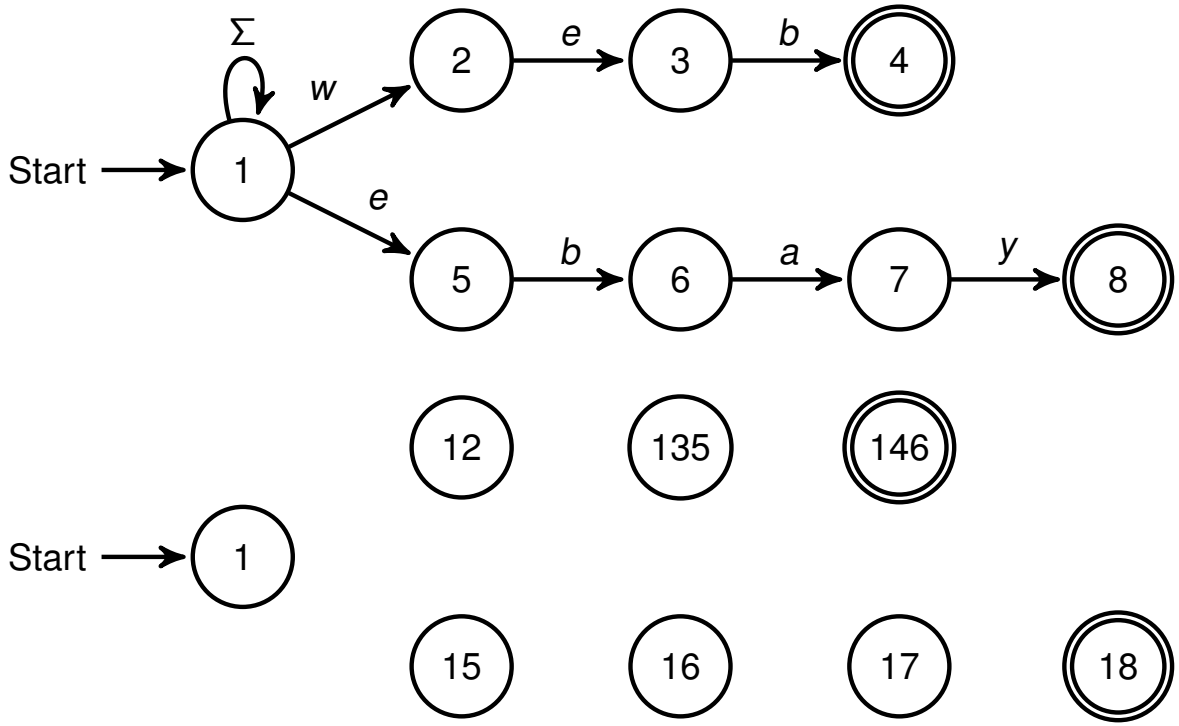
As an illustrating example, we can convert the NFA for text search to an equivalent DFA using subset construction.

More precise rules for constructing the set of DFA states would make the process efficiently. They are

1. If q_0 is the start state of the NFA, then $\{q_0\}$ is one of the DFA's states.
2. Suppose p is one of the NFA's states, and it is reached from q_0 along a path whose symbols are $a_1 a_2 \cdots a_m$. Then one of the DFA's states is the set of NFA's states consisting of:
   - q_0, p, and every other NFA's state that is reachable from q_0 by following a path whose labels are a suffix of $a_1 a_2 \cdots a_m$, that is, any sequence of symbols of the form $a_j a_{j+1} \cdots a_m$.

Example

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



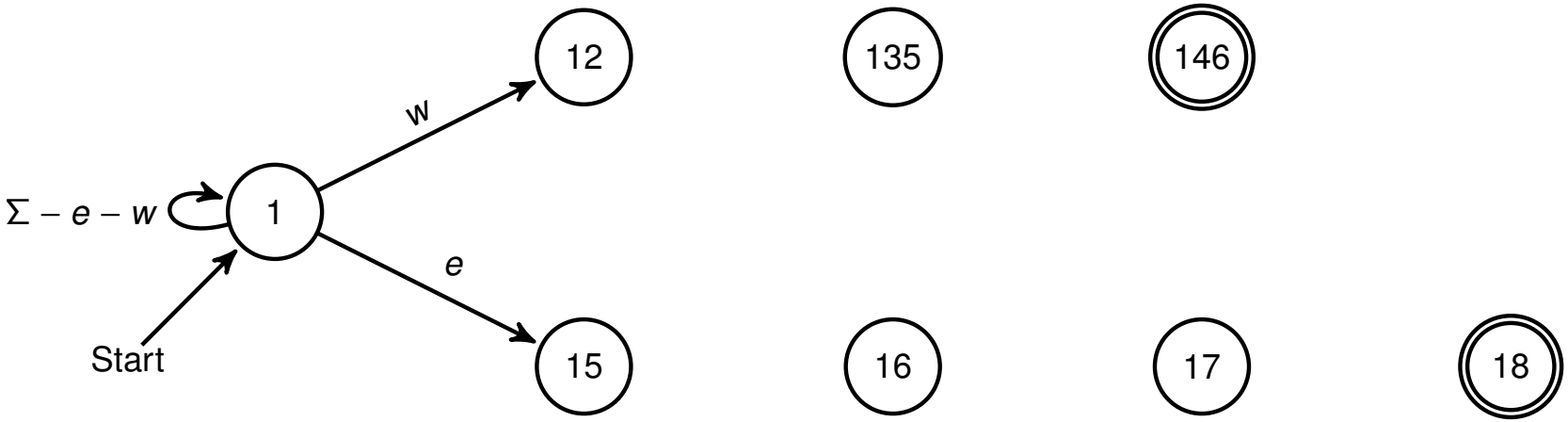
Here 1 is shorthand for $\{1\}$, 12 for $\{1, 2\}$, and 135 for $\{1, 3, 5\}$, and so on.

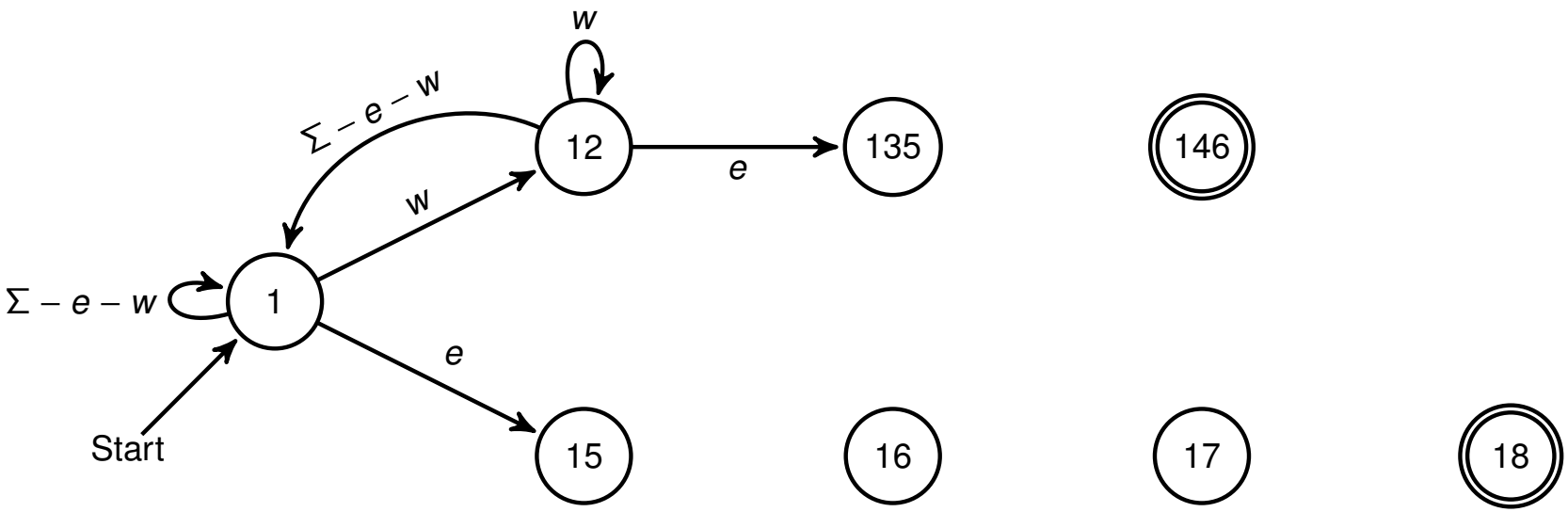
The transition for each of the DFA states may be calculated according to the formula

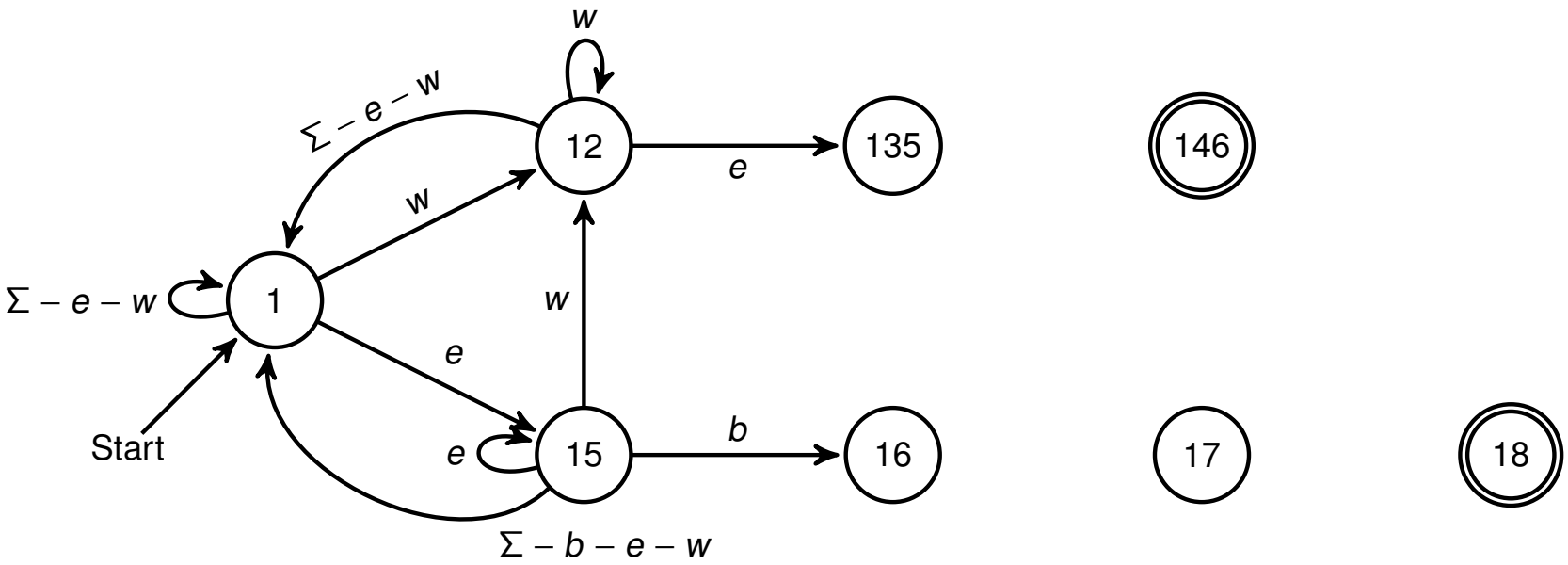
$$\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a).$$

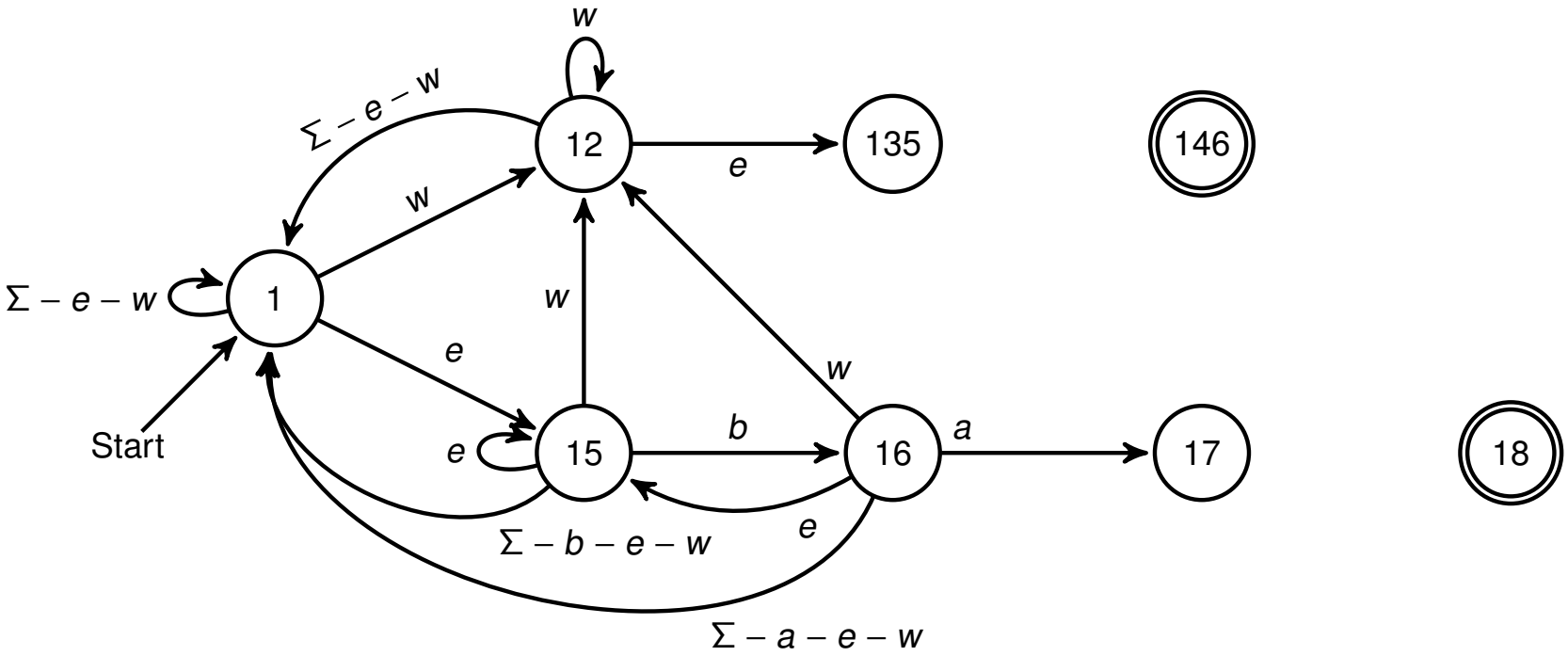
Question

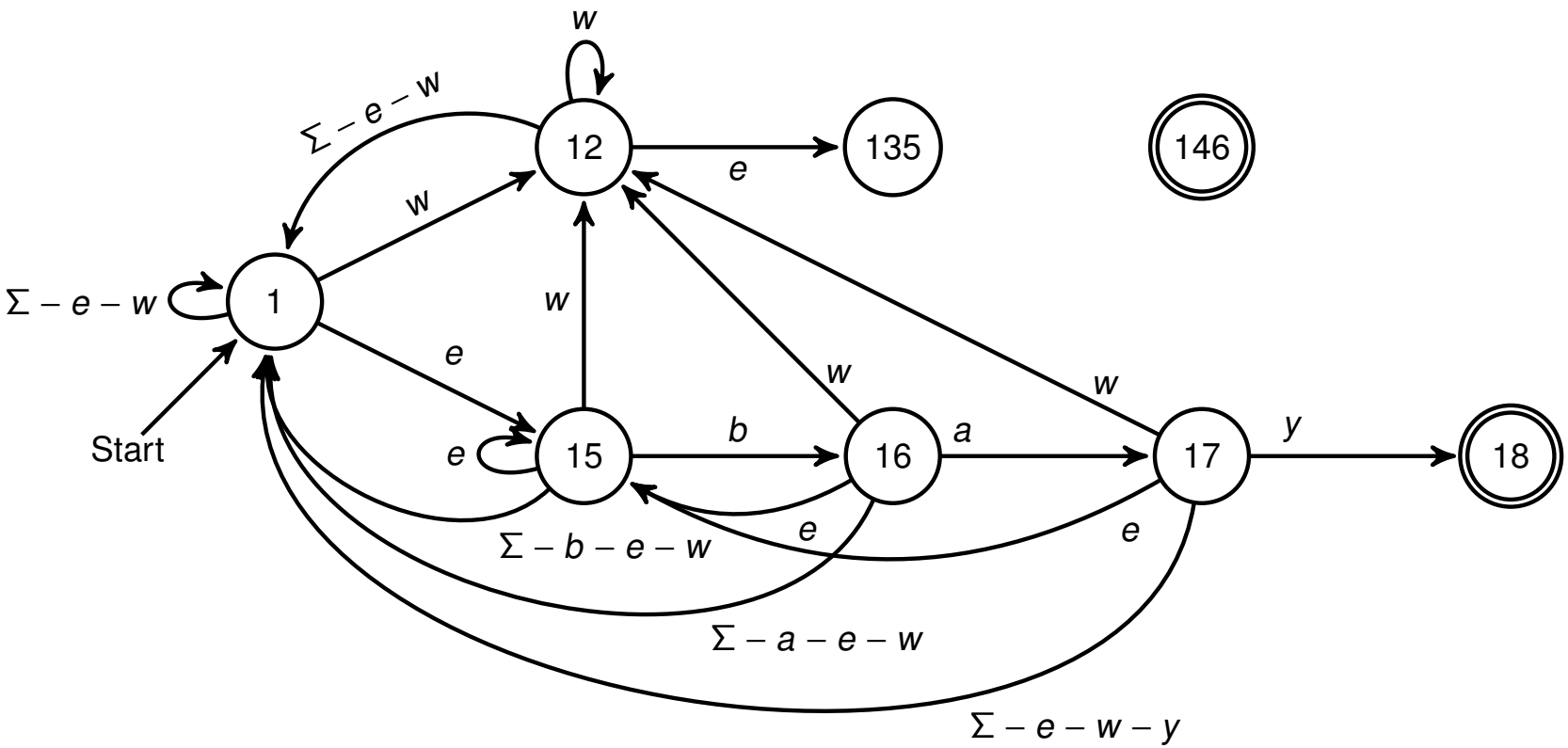
1. For automaton recognizing keywords in text, when will it happen that the number of states of "subset" DFA is less than that of NFA?
2. Does the "rules for constructing DFA states" follow from the subset construction and the strategy of "lazy evaluation"?

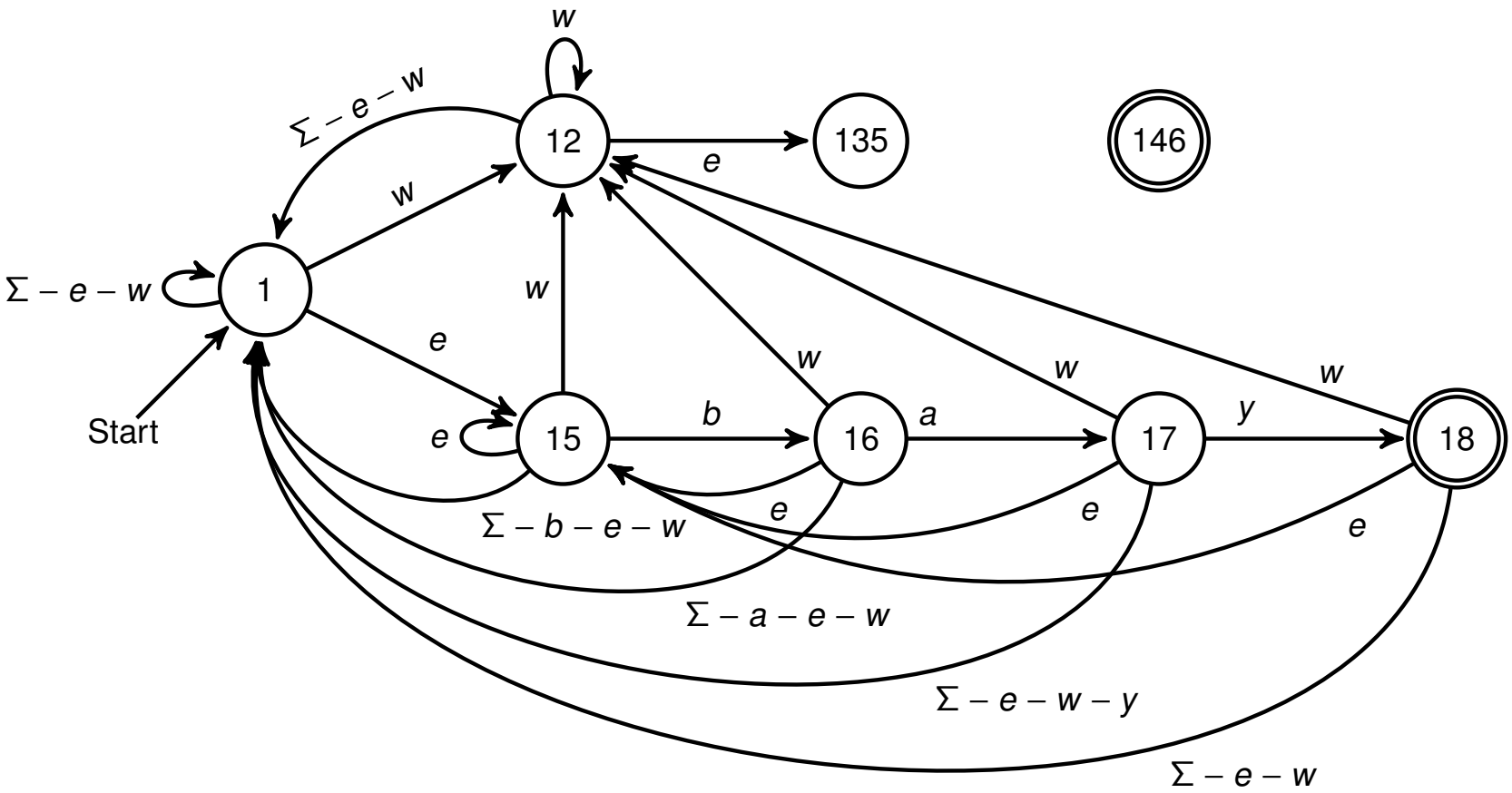

12
135
146
w
Σ – e – w
1
e
Start
15
16
17
18

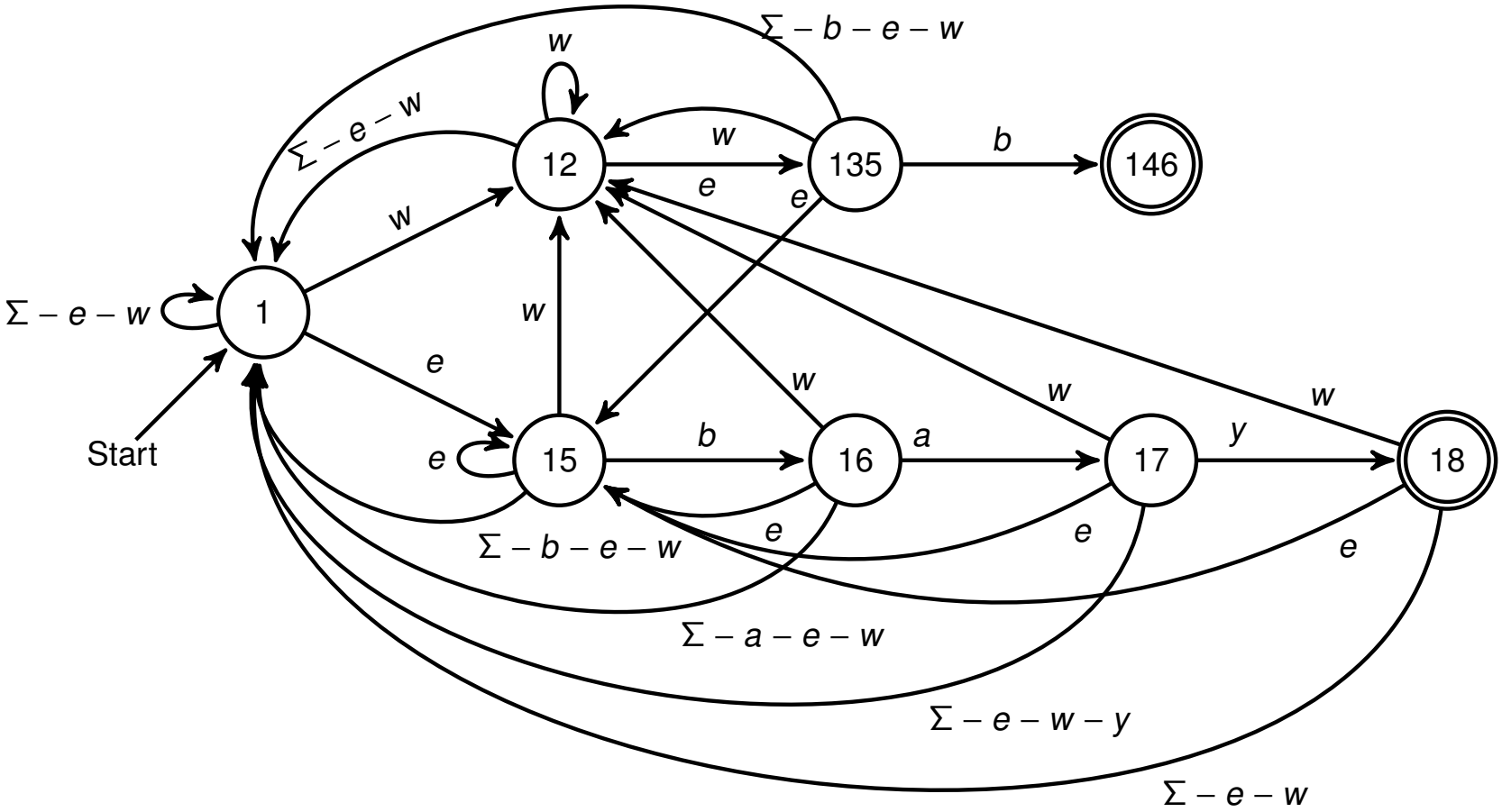

w
Σ – e – w
12
135
146
e
w
Σ – e – w
1
e
Start
15
16
17
18

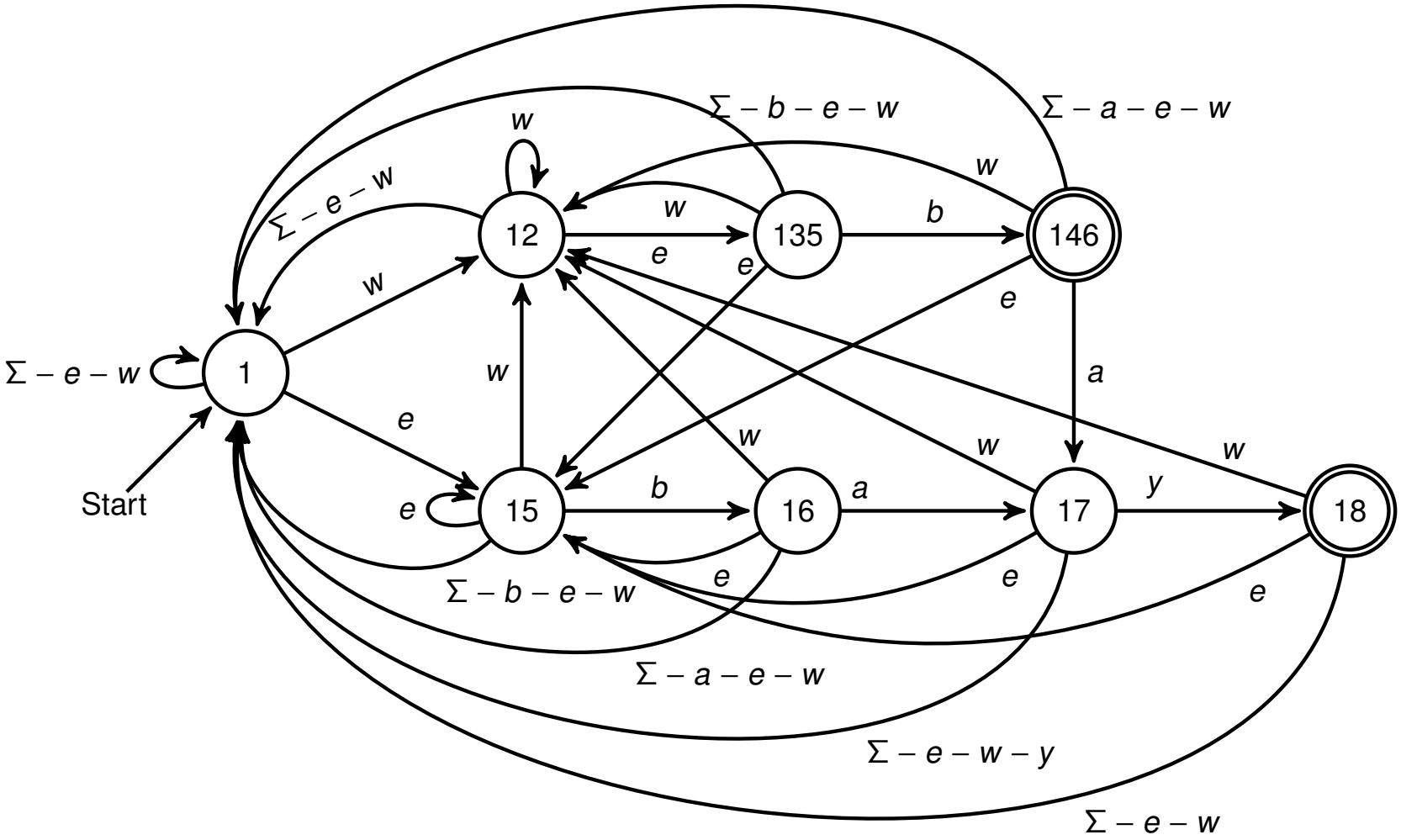

w
Σ – e – w
12
135
146
e
w
Σ – e – w
1
w
e
Start
e
15
b
16
17
18
Σ – b – e – w


w
Σ – e – w
12
135
146
e
w
Σ – e – w
1
w
e
w
Start
e
15
b
16
a
17
18
Σ – b – e – w
e
Σ – a – e – w


w
Σ – e – w
12
e
135
146
w
Σ – e – w
1
w
e
w
w
w
Start
e
15
b
16
a
17
y
18
e
e
Σ – b – e – w
Σ – a – e – w
Σ – e – w – y


w
Σ – e – w
12
135
146
e
w
Σ – e – w
1
w
e
w
w
w
Start
e
15
b
16
a
17
y
18
e
e
e
Σ – b – e – w
Σ – a – e – w
Σ – e – w – y
Σ – e – w


Σ – b – e – w
w
Σ – e – w
12
w
135
b
146
e
e
w
Σ – e – w
1
w
e
w
w
w
w
Start
e
15
b
16
a
17
y
18
e
Σ – b – e – w
e
e
e
Σ – a – e – w
Σ – e – w – y
Σ – e – w


Start
1
12
135
146
15
16
17
18
Σ – e – w
w
Σ – b – e – w
Σ – a – e – w
e
b
a
y
Σ – e – w – y

Homework

Exercises 2.2.9 & 2.4.1

Languages defined by other condition

One may ask what if the acceptance condition

$$\{w \mid \hat{\delta}(q_0, w) \in F\}$$

is changed to:

$$\{w \mid w = uv \wedge \hat{\delta}(q_0, u) \in F\},$$

which means we accept this string w whenever some intermediate state is in F.

We can rephrase it as making all states in F absorbing, since the successive states are irrevalent to accept or reject in the setting.

Languages defined by other condition

DFAs with final states being absorbing are still DFAs, which has the restriction

$$\delta(q, a) = q \quad \text{for each } q \in F \text{ and } a \in \Sigma,$$

so the new languages in the consideration are still in the scope of regular languages.

A more interesting question is whether every regular language over alphabet Σ can be accepted by such a special DFA.

Unfortuniately, the answer is negative.

Languages defined by other condition

But we could partially fill the the gap by: Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$, there is a DFA $B = (Q_B, \Sigma_B, \delta_B, q_0, F_B)$ where

- $Q_B = Q \cup \{good, bad\}$,
- $\Sigma_B = \Sigma \cup \{\#\}$,
- $$\delta_B(q, a) = \begin{cases} \delta(q, a) & \text{if } q \in Q \text{ and } a \in \Sigma, \\ good & \text{if } q \in F \text{ and } a = \#, \\ bad & \text{if } q \in Q \setminus F \text{ and } a = \#, \\ good/bad & \text{if } q = good/bad. \end{cases}$$
- $F = \{good\}$,

so that

$$L(B) = \{u\#v \mid u \in L(A) \wedge v \in \Sigma_B^*\}.$$