

Formal Definition of a Nondeterministic Finite Automaton

A comment first

- The formal definition of an NFA is similar to that of a DFA. Both have states, an alphabet, transition function, one start state, and a collection of accept states

A comment first

- The formal definition of an NFA is similar to that of a DFA. Both have states, an alphabet, transition function, one start state, and a collection of accept states
- Differ in one essential way: the transition functions:

A comment first

- The formal definition of an NFA is similar to that of a DFA. Both have states, an alphabet, transition function, one start state, and a collection of accept states
- Differ in one essential way: the transition functions:
 - In a DFA transition function is $\delta : Q \times \Sigma \rightarrow Q$;

A comment first

- The formal definition of an NFA is similar to that of a DFA. Both have states, an alphabet, transition function, one start state, and a collection of accept states
- Differ in one essential way: the transition functions:
 - In a DFA transition function is $\delta : Q \times \Sigma \rightarrow Q$;
 - In an NFA it is $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$

A comment first

- The formal definition of an NFA is similar to that of a DFA. Both have states, an alphabet, transition function, one start state, and a collection of accept states
- Differ in one essential way: the transition functions:
 - In a DFA transition function is $\delta : Q \times \Sigma \rightarrow Q$;
 - In an NFA it is $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$
 - where $\mathcal{P}(Q)$ is the collection of subsets of Q called the power set of Q

Notation

For any alphabet Σ we write Σ_ϵ for $\Sigma \cup \{\epsilon\}$

NFA formal definition (Def. 1.37)

A nondeterministic finite automaton (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

NFA formal definition (Def. 1.37)

A nondeterministic finite automaton (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of **states**

NFA formal definition (Def. 1.37)

A nondeterministic finite automaton (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of **states**
- Σ is a finite **alphabet**

NFA formal definition (Def. 1.37)

A nondeterministic finite automaton (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of **states**
- Σ is a finite **alphabet**
- $\delta : Q \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q)$ is the **transition function**

NFA formal definition (Def. 1.37)

A nondeterministic finite automaton (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of **states**
- Σ is a finite **alphabet**
- $\delta : Q \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q)$ is the **transition function**
- $q_0 \in Q$ is the **start state**

NFA formal definition (Def. 1.37)

A nondeterministic finite automaton (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of **states**
- Σ is a finite **alphabet**
- $\delta : Q \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q)$ is the **transition function**
- $q_0 \in Q$ is the **start state**
- $F \subseteq Q$ is the set of **accept states**

Example NFA (Example 1.38)

Recall the NFA N_1 , Figure 1:

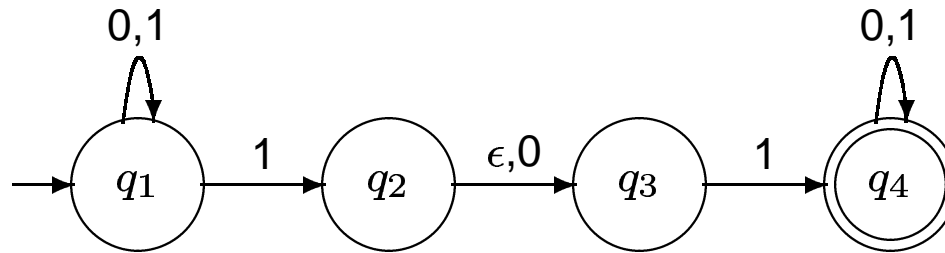


Figure 1: Nondeterministic Finite Automaton N_1

The formal definition of N_1

$N_1 = (Q, \Sigma, \delta, q_1, F)$ where:

1. $Q = \{q_1, q_2, q_3, q_4\}$
2. $\Sigma = \{0, 1\}$
3. δ is given in the table:

δ	0	1	ϵ
q_1	$\{q_1\}$	$\{q_1, q_2\}$	\emptyset
q_2	$\{q_3\}$	\emptyset	$\{q_3\}$
q_3	\emptyset	$\{q_4\}$	\emptyset
q_4	$\{q_4\}$	$\{q_4\}$	\emptyset

4. q_1 is the start set;
5. $F = \{q_4\}$

Computation performed by an NFA

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and w a string over Σ .
We say that N accepts w if:

Computation performed by an NFA

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and w a string over Σ .

We say that N accepts w if:

- We can write w as $w = y_1 y_2 \dots y_m$, $y_i \in \Sigma_\epsilon$, $1 \leq i \leq m$

Computation performed by an NFA

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and w a string over Σ .

We say that N accepts w if:

- We can write w as $w = y_1 y_2 \dots y_m$, $y_i \in \Sigma_\epsilon$, $1 \leq i \leq m$
- A sequence of states r_0, r_1, \dots, r_m exists in Q s.t:

Computation performed by an NFA

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and w a string over Σ .

We say that N accepts w if:

- We can write w as $w = y_1 y_2 \dots y_m$, $y_i \in \Sigma_\epsilon$, $1 \leq i \leq m$
- A sequence of states r_0, r_1, \dots, r_m exists in Q s.t:
 1. $r_0 = q_0$

Computation performed by an NFA

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and w a string over Σ .

We say that N accepts w if:

- We can write w as $w = y_1 y_2 \dots y_m$, $y_i \in \Sigma_\epsilon$, $1 \leq i \leq m$
- A sequence of states r_0, r_1, \dots, r_m exists in Q s.t:
 1. $r_0 = q_0$
 2. $r_{i+1} \in \delta(r_i, y_{i+1})$, for $i = 0, 1, \dots, m - 1$

Computation performed by an NFA

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and w a string over Σ .

We say that N accepts w if:

- We can write w as $w = y_1 y_2 \dots y_m$, $y_i \in \Sigma_\epsilon$, $1 \leq i \leq m$
- A sequence of states r_0, r_1, \dots, r_m exists in Q s.t:
 1. $r_0 = q_0$
 2. $r_{i+1} \in \delta(r_i, y_{i+1})$, for $i = 0, 1, \dots, m - 1$
 3. $r_m \in F$

Interpretation

- **Condition 1** says that the machine **starts** its computation **in the start state**

Interpretation

- **Condition 1** says that the machine **starts** its computation **in the start state**
- **Condition 2** says that **state r_{i+1} is one of the allowable new states** when N is in state r_i and reads y_{i+1} . Note that $\delta(r_i, y_{i+1})$ is a set

Interpretation

- **Condition 1** says that the machine **starts** its computation **in the start state**
- **Condition 2** says that **state r_{i+1} is one of the allowable new states** when N is in state r_i and reads y_{i+1} . Note that $\delta(r_i, y_{i+1})$ is a set
- **Condition 3** says that the machine accepts the input if the **last state is in the accept state set.**

An important property

Any NFA N can be converted to an equivalent NFA N' that has a single accept state

An important property

Any NFA N can be converted to an equivalent NFA N' that has a single accept state

Proof: by construction. Let $N = (Q, \Sigma, \delta, q_0, F)$ be any NFA. Then $N' = (Q', \Sigma', \delta', q'_0, F')$ where:

An important property

Any NFA N can be converted to an equivalent NFA N' that has a single accept state

Proof: by construction. Let $N = (Q, \Sigma, \delta, q_0, F)$ be any NFA. Then $N' = (Q', \Sigma', \delta', q'_0, F')$ where:

1. $Q' = Q \cup \{q_a\}$ where q_a is a new state

An important property

Any NFA N can be converted to an equivalent NFA N' that has a single accept state

Proof: by construction. Let $N = (Q, \Sigma, \delta, q_0, F)$ be any NFA. Then $N' = (Q', \Sigma', \delta', q'_0, F')$ where:

1. $Q' = Q \cup \{q_a\}$ where q_a is a new state
2. $\Sigma' = \Sigma$

An important property

Any NFA N can be converted to an equivalent NFA N' that has a single accept state

Proof: by construction. Let $N = (Q, \Sigma, \delta, q_0, F)$ be any NFA. Then $N' = (Q', \Sigma', \delta', q'_0, F')$ where:

1. $Q' = Q \cup \{q_a\}$ where q_a is a new state
2. $\Sigma' = \Sigma$
3. $q'_0 = q_0$

An important property

Any NFA N can be converted to an equivalent NFA N' that has a single accept state

Proof: by construction. Let $N = (Q, \Sigma, \delta, q_0, F)$ be any NFA. Then $N' = (Q', \Sigma', \delta', q'_0, F')$ where:

1. $Q' = Q \cup \{q_a\}$ where q_a is a new state
2. $\Sigma' = \Sigma$
3. $q'_0 = q_0$
- 4.

$$\delta'(q, a) = \begin{cases} \delta(q, a), & \text{if } a \neq \epsilon \text{ or } q \notin F; \\ \delta(q, a) \cup \{q_a\}, & \text{if } a = \epsilon \text{ and } q \in F. \end{cases}$$

An important property

Any NFA N can be converted to an equivalent NFA N' that has a single accept state

Proof: by construction. Let $N = (Q, \Sigma, \delta, q_0, F)$ be any NFA. Then $N' = (Q', \Sigma', \delta', q'_0, F')$ where:

1. $Q' = Q \cup \{q_a\}$ where q_a is a new state

2. $\Sigma' = \Sigma$

3. $q'_0 = q_0$

4.

$$\delta'(q, a) = \begin{cases} \delta(q, a), & \text{if } a \neq \epsilon \text{ or } q \notin F; \\ \delta(q, a) \cup \{q_a\}, & \text{if } a = \epsilon \text{ and } q \in F. \end{cases}$$

5. $F' = \{q_a\}$

Equivalence of NFA and DFA

- DFAs and NFAs recognize the same class of languages

Equivalence of NFA and DFA

- DFAs and NFAs recognize the same class of languages
- This equivalence is both surprising and useful

Equivalence of NFA and DFA

- DFAs and NFAs recognize the same class of languages
- This equivalence is both surprising and useful
 1. It is surprising because NFAs appears to have more power than DFA, so we might expect that NFA recognizes more languages

Equivalence of NFA and DFA

- DFAs and NFAs recognize the same class of languages
- This equivalence is both surprising and useful
 1. It is surprising because NFAs appears to have more power than DFA, so we might expect that NFA recognizes more languages
 2. It is useful because describing an NFA for a given language sometimes is much easier than describing a DFA

Theorem 1.39

Every nondeterministic finite automaton (NFA) has an equivalent deterministic finite automaton (DFA).

Proof idea: Convert NFA recognizing a language into a DFA that recognizes the same language by simulating the NFA

Theorem 1.39

Every nondeterministic finite automaton (NFA) has an equivalent deterministic finite automaton (DFA).

Proof idea: Convert NFA recognizing a language into a DFA that recognizes the same language by simulating the NFA

More on proof idea

The approach is to pretend to be an DFA simulating an NFA. Questions to be answered:

More on proof idea

The approach is to pretend to be an DFA simulating an NFA. Questions to be answered:

- How would you simulate an NFA if you were pretending to be a DFA?

More on proof idea

The approach is to pretend to be an DFA simulating an NFA. Questions to be answered:

- How would you simulate an NFA if you were pretending to be a DFA?
- What does one need to keep track of as the input is processed?

Note

- In the examples of NFA we kept track of various branches of the computation by placing tokens on each state that could be activated at given points in the input

Note

- In the examples of NFA we kept track of various branches of the computation by placing tokens on each state that could be activated at given points in the input
- Tokens were updated by moving, adding, and removing them according to the way NFA operates

Note

- In the examples of NFA we kept track of various branches of the computation by placing tokens on each state that could be activated at given points in the input
- Tokens were updated by moving, adding, and removing them according to the way NFA operates
- Hence, all we needed to keep track of was the set of states with tokens on them.

Conclusion

- An NFA with k states has 2^k subsets of states

Conclusion

- An NFA with k states has 2^k subsets of states
- Each subset corresponds to one of the possibilities that DFA must remember. So, the DFA simulating an NFA with k states has 2^k states

Conclusion

- An NFA with k states has 2^k subsets of states
- Each subset corresponds to one of the possibilities that DFA must remember. So, the DFA simulating an NFA with k states has 2^k states
- The only thing we need now is to figure out which are the start state, the accept states, and transition function

The formal proof

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing the language A . We construct the DFA M recognizing A . Before doing the full construction, consider first the easier case when N has no ϵ transitions.

Constructing $M = (Q', \Sigma, \delta', q'_0, F')$

1. $Q' = \mathcal{P}(Q)$; every state of M is a set of states of N

Constructing $M = (Q', \Sigma, \delta', q'_0, F')$

1. $Q' = \mathcal{P}(Q)$; every state of M is a set of states of N
2. For $R \in Q'$ and $a \in \Sigma$ let
$$\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$$

Constructing $M = (Q', \Sigma, \delta', q'_0, F')$

1. $Q' = \mathcal{P}(Q)$; every state of M is a set of states of N

2. For $R \in Q'$ and $a \in \Sigma$ let

$$\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$$

- **Note:** $\delta'(R, a) = \cup_{r \in R} \delta(r, a)$

Constructing $M = (Q', \Sigma, \delta', q'_0, F')$

1. $Q' = \mathcal{P}(Q)$; every state of M is a set of states of N
2. For $R \in Q'$ and $a \in \Sigma$ let
$$\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$$

- **Note:** $\delta'(R, a) = \cup_{r \in R} \delta(r, a)$
3. $q'_0 = \{q_0\}$; M starts computing in the same state as N

Constructing $M = (Q', \Sigma, \delta', q'_0, F')$

1. $Q' = \mathcal{P}(Q)$; every state of M is a set of states of N
2. For $R \in Q'$ and $a \in \Sigma$ let
$$\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$$

- **Note:** $\delta'(R, a) = \cup_{r \in R} \delta(r, a)$
3. $q'_0 = \{q_0\}$; M starts computing in the same state as N
4. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

Constructing $M = (Q', \Sigma, \delta', q'_0, F')$

1. $Q' = \mathcal{P}(Q)$; every state of M is a set of states of N

2. For $R \in Q'$ and $a \in \Sigma$ let

$$\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$$

- **Note:** $\delta'(R, a) = \cup_{r \in R} \delta(r, a)$

3. $q'_0 = \{q_0\}$; M starts computing in the same state as N

4. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

- that is, the machine M accepts if one of the possible states that N could be in at this point is an accept state

Consider the ϵ transitions

Notation: for any $R \in Q'$ define $E(R)$ to be the collection of states that can be reached from R by going only along ϵ transitions, including the members of R themselves.

Formally:

Consider the ϵ transitions

Notation: for any $R \in Q'$ define $E(R)$ to be the collection of states that can be reached from R by going only along ϵ transitions, including the members of R themselves.

Formally:

$$E(R) = R \cup \{q \in Q \mid \exists r \in R \wedge \delta(r, \epsilon) = q\}$$

Consider the ϵ transitions

Notation: for any $R \in Q'$ define $E(R)$ to be the collection of states that can be reached from R by going only along ϵ transitions, including the members of R themselves.

Formally:

$$E(R) = R \cup \{q \in Q \mid \exists r \in R \wedge \delta(r, \epsilon) = q\}$$

Note: any $q \in E(R)$ can be reached from R by traveling along 0 or more ϵ arrows.

Modifications

- **Modify δ' to place additional tokens on all states that can be reached by going along ϵ arrows after every step. This effect is achieved by replacing $\delta(r, a)$ by $E(\delta(r, a))$ in definition of δ' . That is**

Modifications

- **Modify δ' to place additional tokens on all states that can be reached by going along ϵ arrows after every step. This effect is achieved by replacing $\delta(r, a)$ by $E(\delta(r, a))$ in definition of δ' . That is**

$$\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some } r \in R\}$$

Modifications

- **Modify δ'** to place additional tokens on all states that can be reached by going along ϵ arrows after every step. This effect is achieved by replacing $\delta(r, a)$ by $E(\delta(r, a))$ in definition of δ' . That is

$$\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some } r \in R\}$$

- **Modify the start state** of M to put tokens initially on all states that can be reached from q_0 along ϵ transitions, i.e., $q'_0 = E(\{q_0\})$

Note

- M thus constructed obviously simulates N

Note

- M thus constructed obviously simulates N
- At every step in the computation of M it clearly enters a state that corresponds to the subset of states that N could be in at that point

Conclusion

Since every NFA can be converted into a DFA by the procedure discussed above, NFA provides an alternative way of characterizing regular sets

Corollary 1.40

A language is regular iff some NFA recognizes it

Corollary 1.40

A language is regular iff some NFA recognizes it

Proof:

Corollary 1.40

A language is regular iff some NFA recognizes it

Proof:

- If a language A is recognized by an NFA then A is recognized by the DFA equivalent, hence, A is regular

Corollary 1.40

A language is regular iff some NFA recognizes it

Proof:

- If a language A is recognized by an NFA then A is recognized by the DFA equivalent, hence, A is regular
- If a language A is regular, it means that it is recognized by a DFA. But any DFA is also an NFA hence, the language is recognized by an NFA

Example conversion NFA to DFA

Consider the NFA N_4 described in Figure 2:

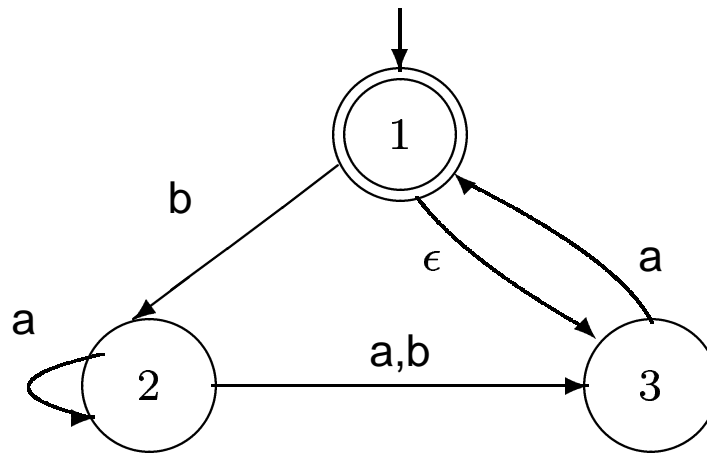


Figure 2: The NFA N_4

Formal description of N_4

$N_4 = (\{1, 2, 3\}, \{a, b\}, \delta, 1, \{1\})$ where δ is given in the table:

δ	a	b	ϵ
1	\emptyset	{2}	{3}
2	{2, 3}	{3}	\emptyset
3	{1}	\emptyset	\emptyset

Construction of $D_4 = (Q', \{a, b\}, \delta', q'_0, F')$

- Since N_4 's set of states is $Q = \{1, 2, 3\}$
 $Q' = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$

Construction of $D_4 = (Q', \{a, b\}, \delta', q'_0, F')$

- Since N_4 's **set of states** is $Q = \{1, 2, 3\}$
 $Q' = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$
- The **start state** of Q' : the set of states reachable from 1 (the start state of N_4) traveling by ϵ : $E(\{1\}) = \{1, 3\}$

Construction of $D_4 = (Q', \{a, b\}, \delta', q'_0, F')$

- Since N_4 's **set of states** is $Q = \{1, 2, 3\}$
 $Q' = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$
- The **start state** of Q' : the set of states reachable from 1 (the start state of N_4) traveling by ϵ : $E(\{1\}) = \{1, 3\}$
- The **accept state** of Q' : are those states of Q' that contain the accept states of Q : i.e.,
 $F' = \{\{1\}, \{1, 2\}, \{1, 3\}, \{1, 2, 3\}\}$

Transition function

$\delta' : \mathcal{P}(Q) \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q)$, where for each $q' \in \mathcal{P}(Q)$ and $x \in \Sigma$
we have $\delta'(q', x) \in \mathcal{P}(Q)$

Constructing D_4 's transitions

- **State \emptyset :** goes to \emptyset on both a, b , i.e., $\delta'(\emptyset, a) = \delta'(\emptyset, b) = \emptyset$

Constructing D_4 's transitions

- **State \emptyset** : goes to \emptyset on both a, b , i.e., $\delta'(\emptyset, a) = \delta'(\emptyset, b) = \emptyset$
- **State $\{1\}$** : goes to \emptyset on a and to $\{2\}$ on b

Constructing D_4 's transitions

- **State \emptyset :** goes to \emptyset on both a, b , i.e., $\delta'(\emptyset, a) = \delta'(\emptyset, b) = \emptyset$
- **State $\{1\}$:** goes to \emptyset on a and to $\{2\}$ on b
- **State $\{2\}$:** goes to $\{2, 3\}$ on a and to $\{3\}$ on b

Constructing D_4 's transitions

- **State \emptyset :** goes to \emptyset on both a, b , i.e., $\delta'(\emptyset, a) = \delta'(\emptyset, b) = \emptyset$
- **State $\{1\}$:** goes to \emptyset on a and to $\{2\}$ on b
- **State $\{2\}$:** goes to $\{2, 3\}$ on a and to $\{3\}$ on b
- **State $\{1, 2\}$:** goes to $\{2, 3\}$ on a and to $\{2, 3\}$ on b

Constructing D_4 's transitions

- **State \emptyset :** goes to \emptyset on both a, b , i.e., $\delta'(\emptyset, a) = \delta'(\emptyset, b) = \emptyset$
- **State $\{1\}$:** goes to \emptyset on a and to $\{2\}$ on b
- **State $\{2\}$:** goes to $\{2, 3\}$ on a and to $\{3\}$ on b
- **State $\{1, 2\}$:** goes to $\{2, 3\}$ on a and to $\{2, 3\}$ on b
- **And so on,** for each state

Putting all together

Figure 3 shows the resulting DFA D_4 equivalent to NFA N_4

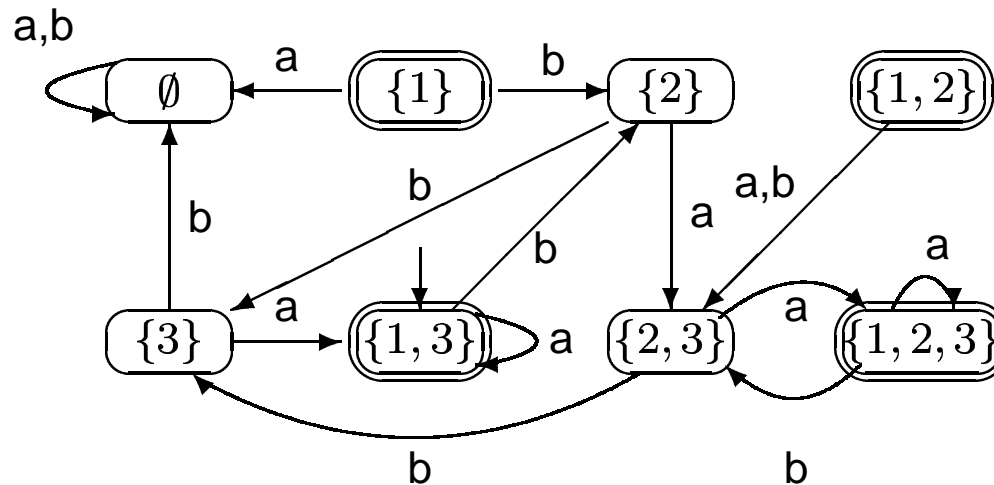


Figure 3: DFA D_4 equivalent to NFA N_4

Simplifying D_4

The automaton D_4 may be simplified by observing that no arrows point at the states $\{1\}$ and $\{1, 2\}$. Removing these states we obtain the automaton in Figure 4

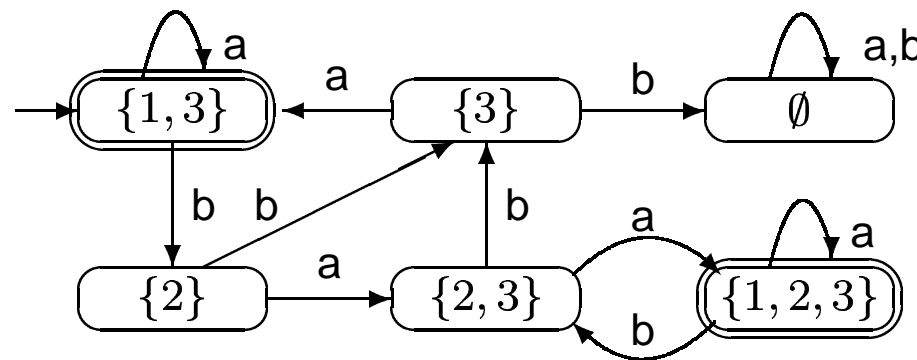


Figure 4: The simplified version of D_4