

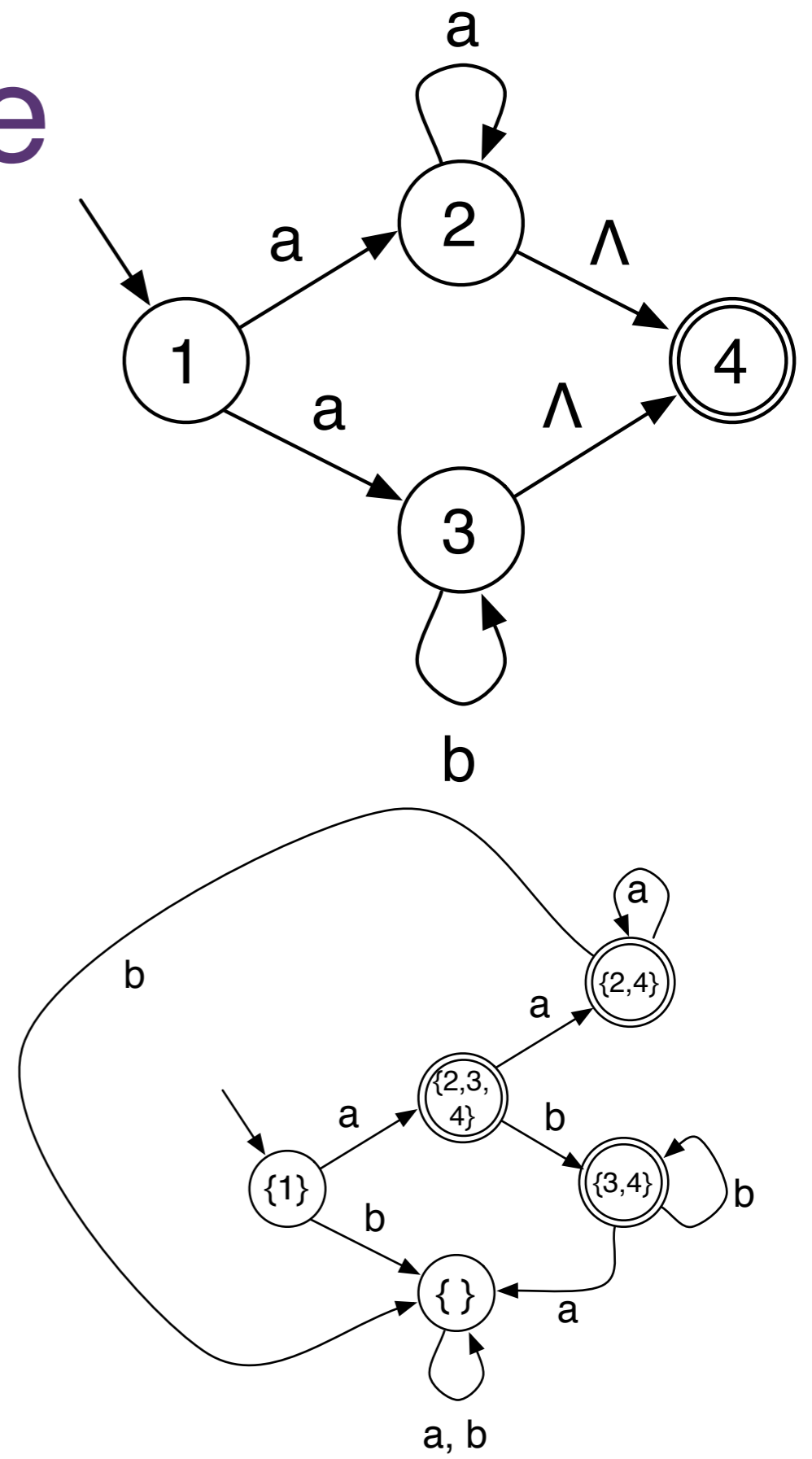
Subset Construction Revisited

Turning an NFA into a DFA

An Important Result

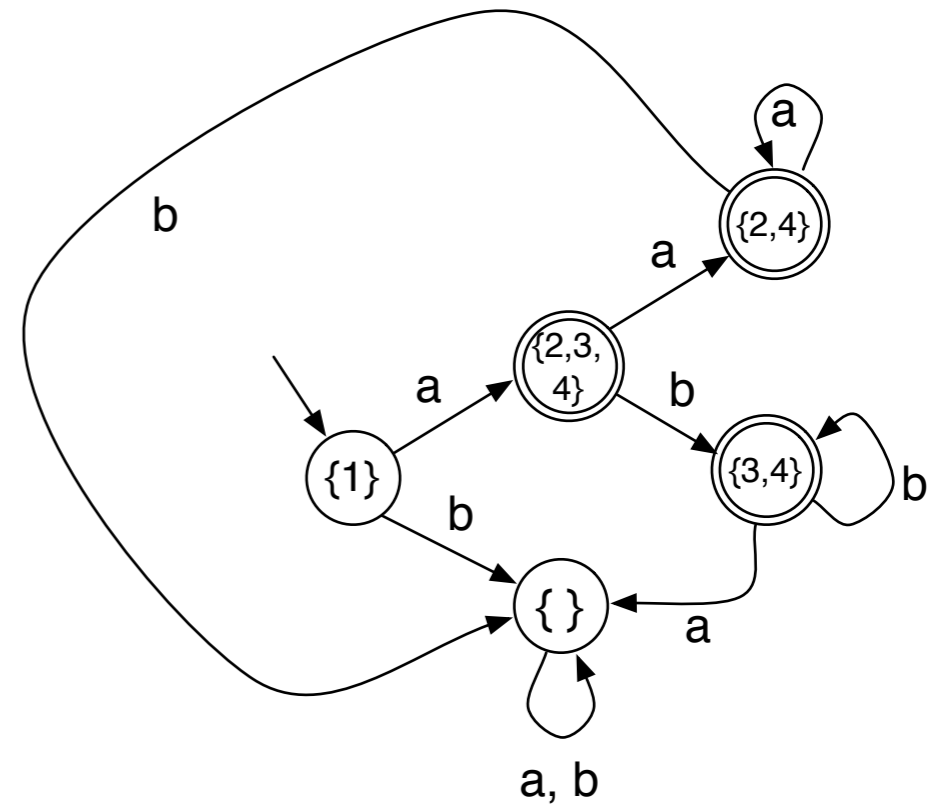
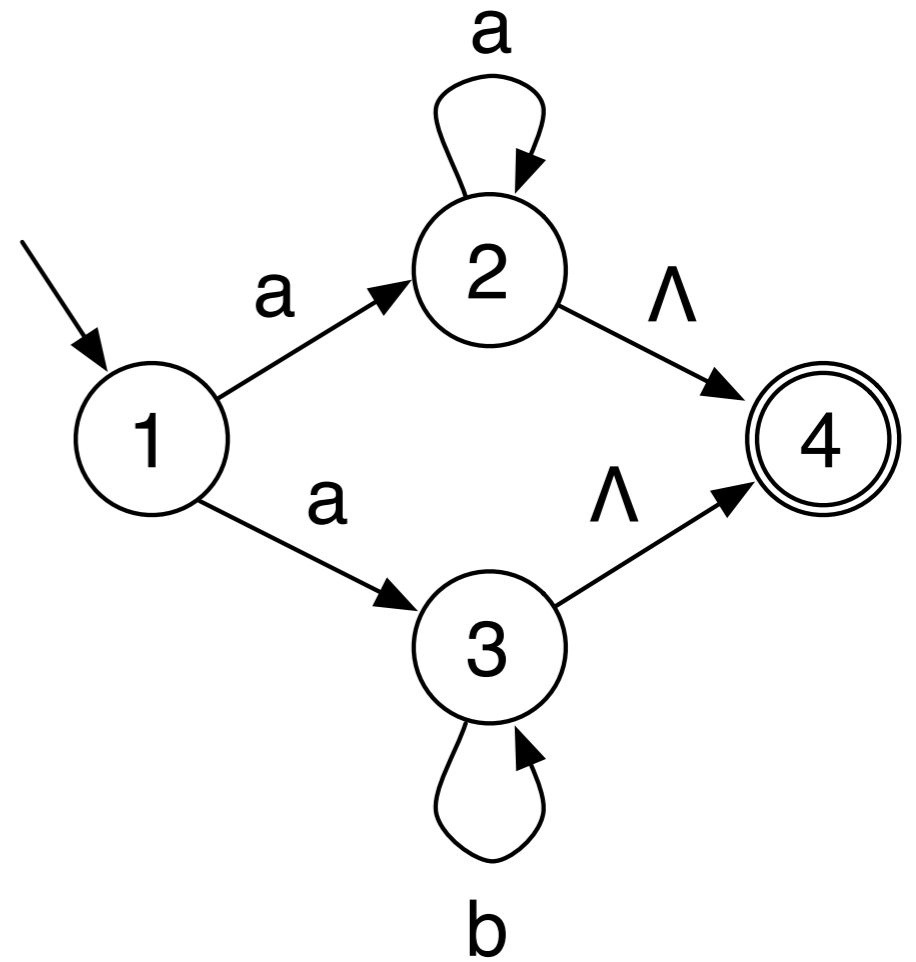
- Why such emphasis on turning NFAs into DFAs?
 - ▶ because we can!
- As a consequence, ***any*** regular language can be recognized by a ***deterministic*** mechanism

Example



Example

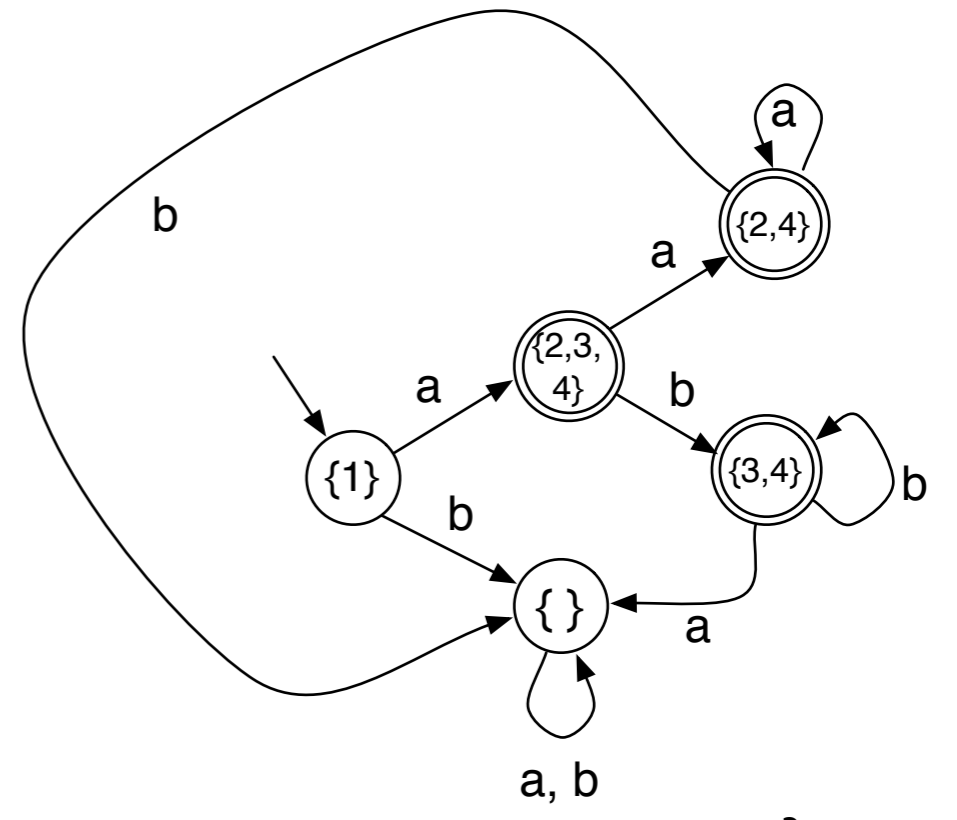
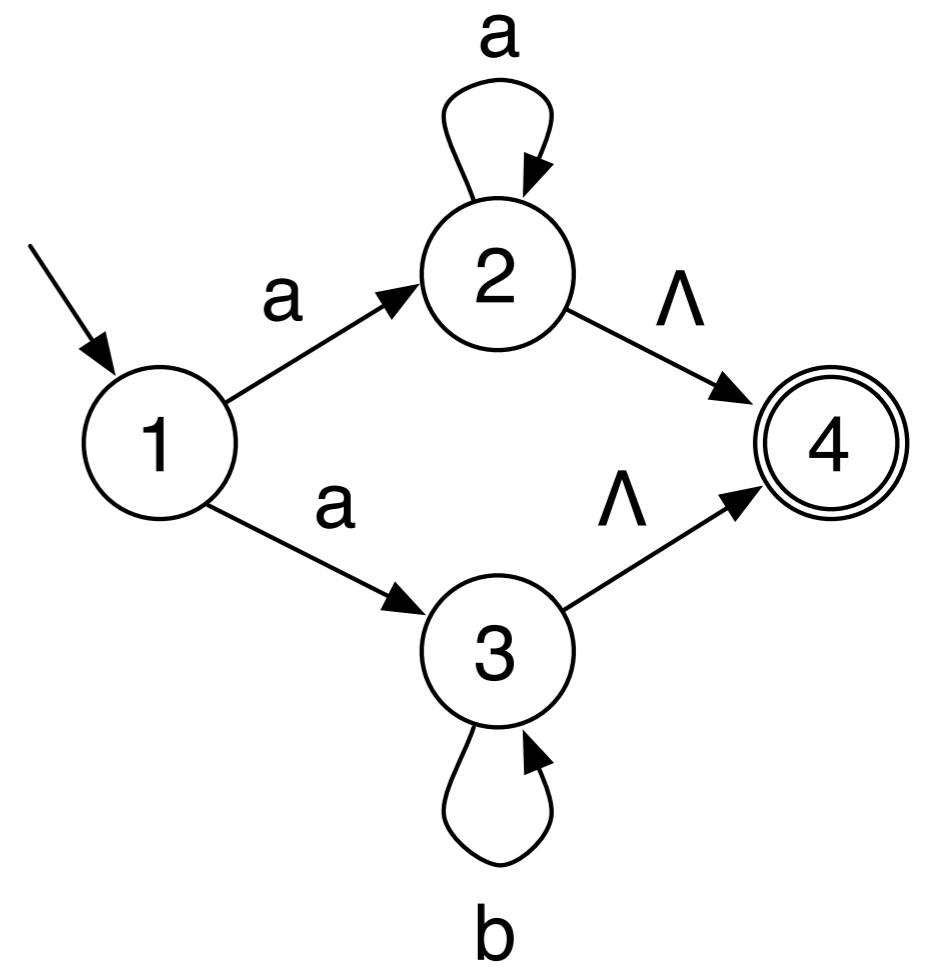
1. Start: $\lambda\{1\} = \{1\}$



Example

1. Start: $\lambda\{1\} = \{1\}$

2. read a: $\lambda\{2, 3\} = \{2, 3, 4\}$



Example

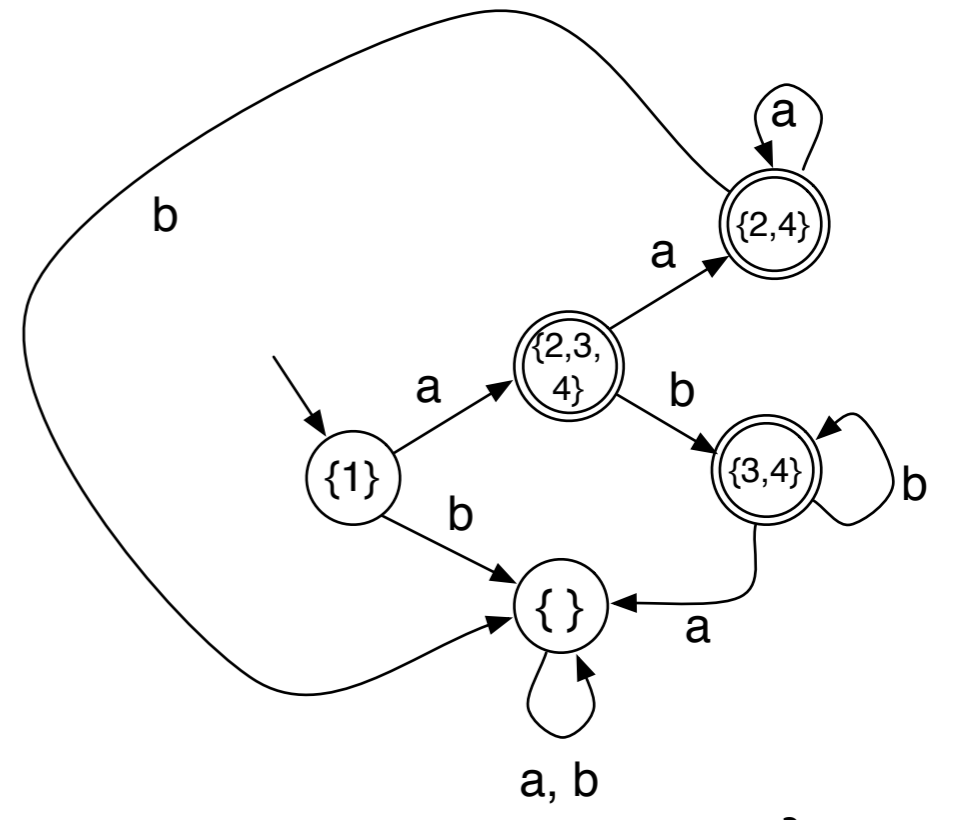
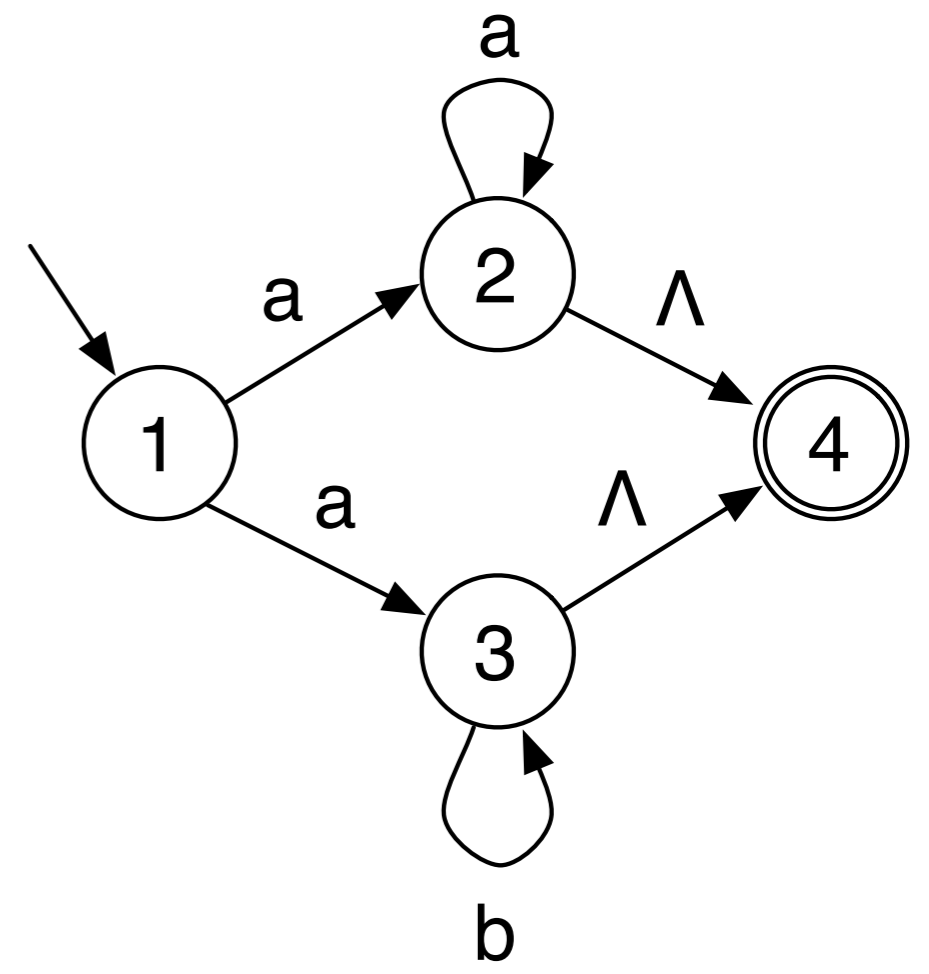
1. Start: $\lambda\{1\} = \{1\}$

2. read a: $\lambda\{2, 3\} = \{2, 3, 4\}$

$$\delta(a, 2) = \{2\}$$

$$\delta(a, 3) = \{\}$$

$$\delta(a, 4) = \{\}$$



Example

1. Start: $\lambda\{1\} = \{1\}$

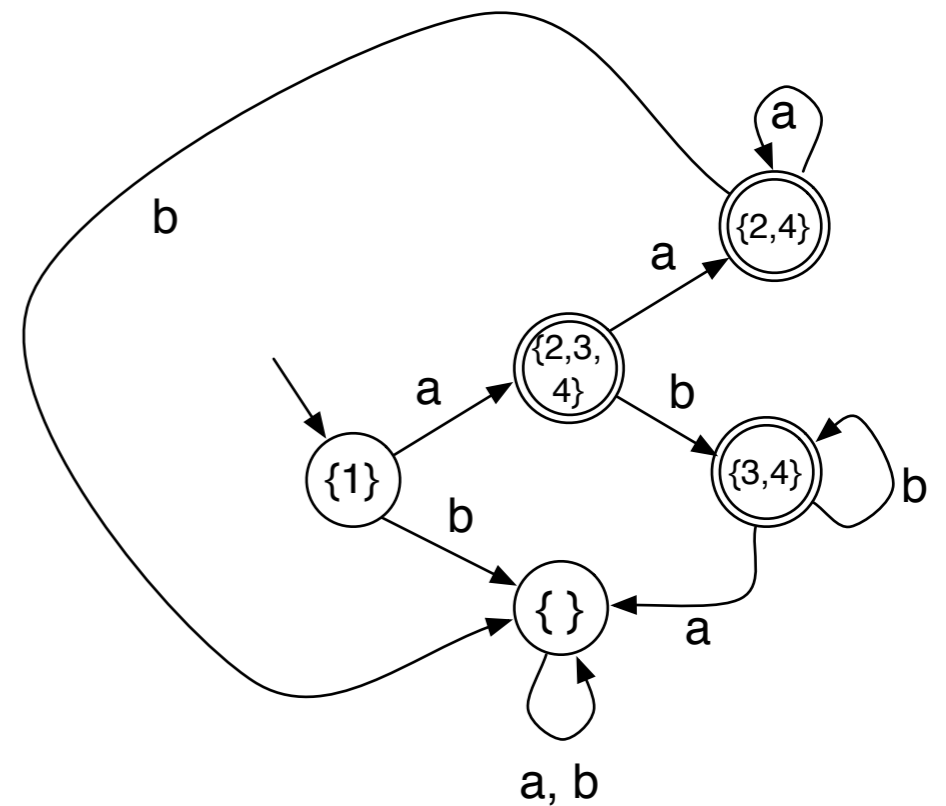
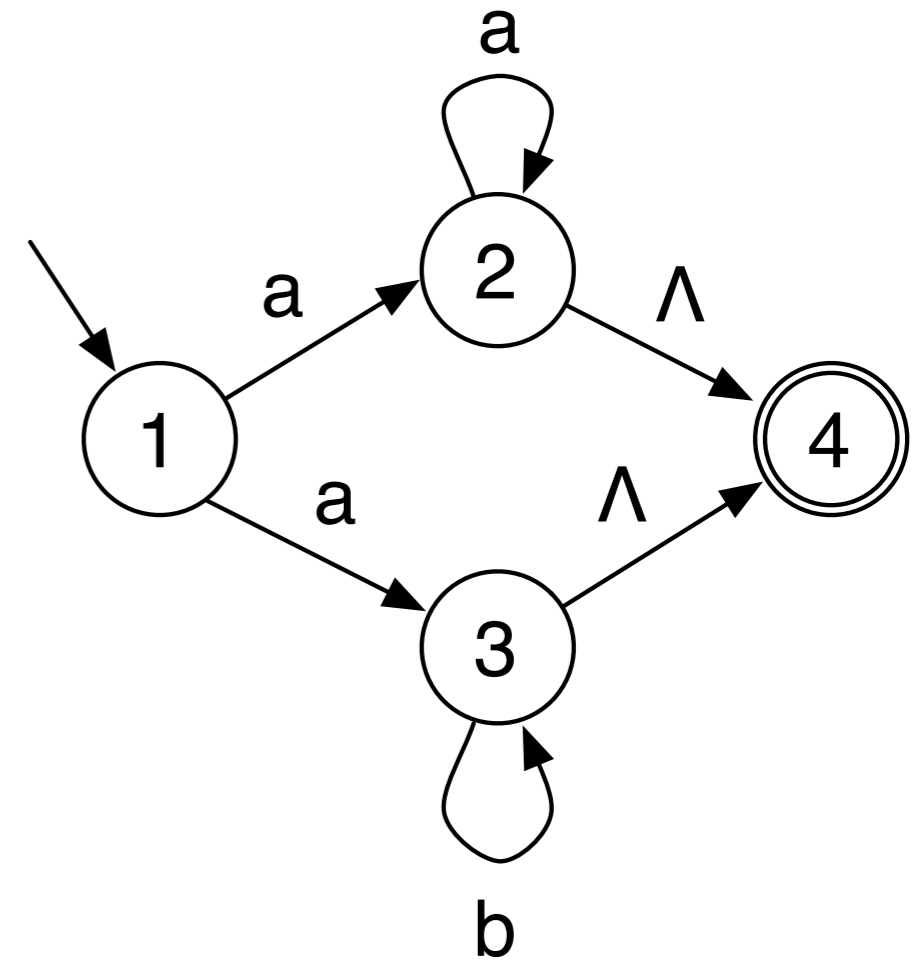
2. read a: $\lambda\{2, 3\} = \{2, 3, 4\}$

$$\delta(a, 2) = \{2\}$$

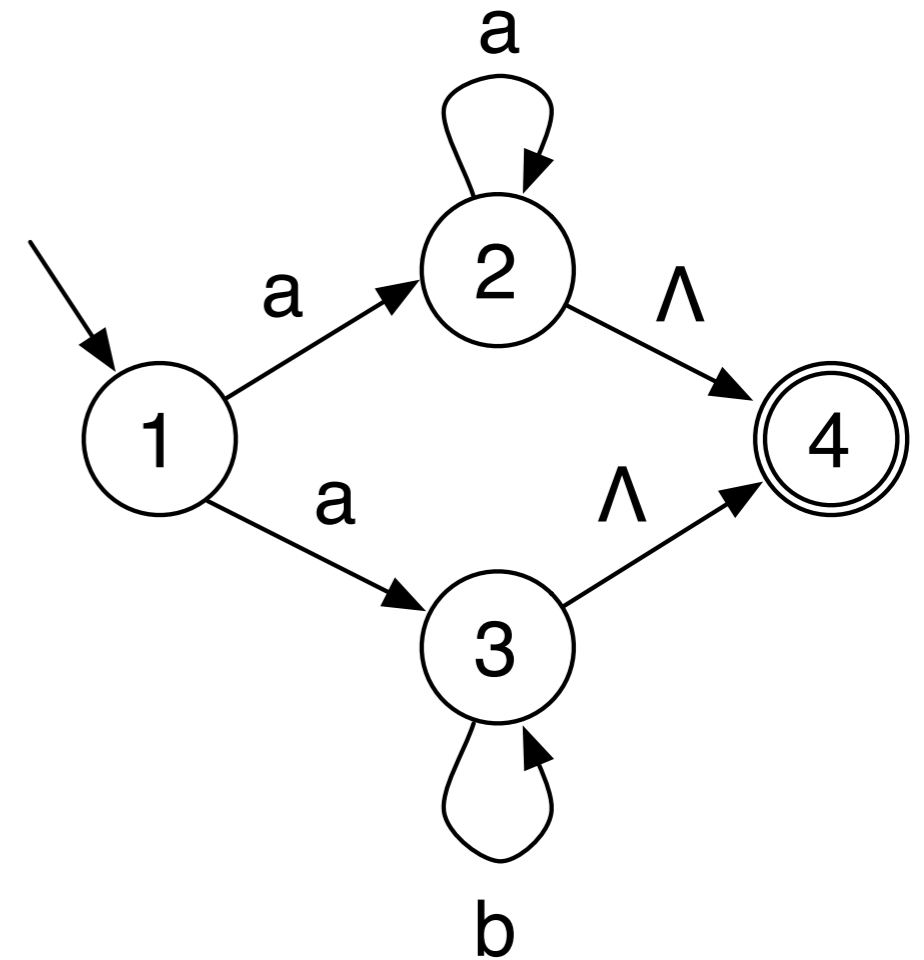
$$\delta(a, 3) = \{\}$$

$$\delta(a, 4) = \{\}$$

3. read aa: $\lambda\{2\} = \{2, 4\}$



Example



1. Start: $\lambda\{1\} = \{1\}$

2. read a: $\lambda\{2, 3\} = \{2, 3, 4\}$

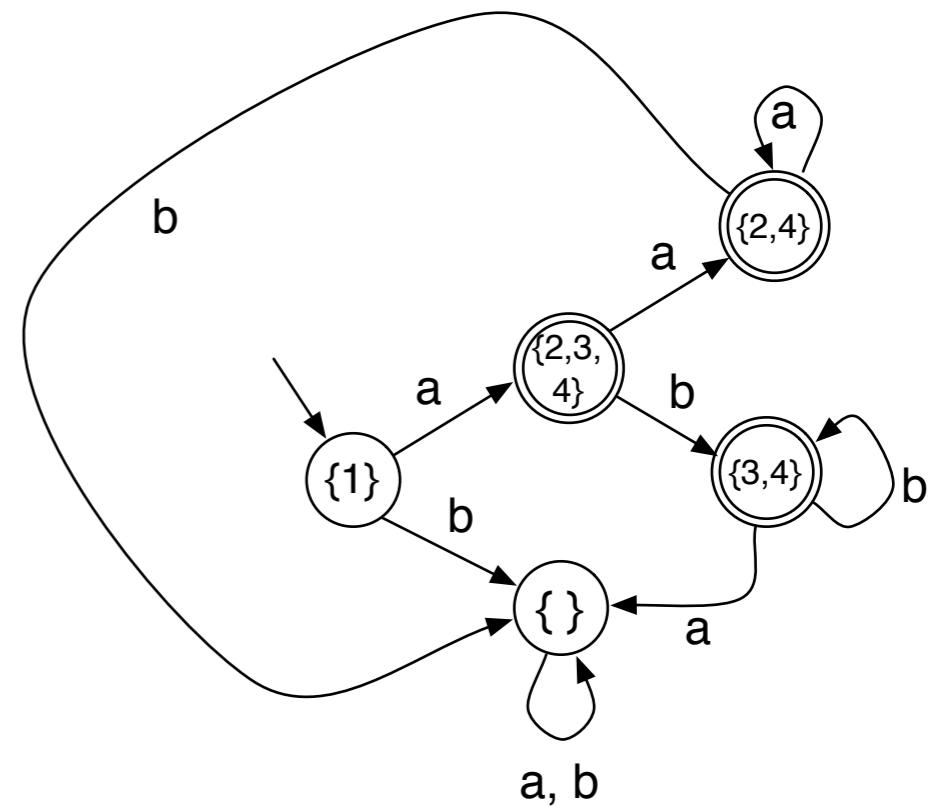
$$\delta(a, 2) = \{2\}$$

$$\delta(a, 3) = \{\}$$

$$\delta(a, 4) = \{\}$$

3. read aa: $\lambda\{2\} = \{2, 4\}$

4. read aaa: $\lambda\{2\} = \{2, 4\}$



What's the DFA?

- A DFS is a 5-tuple:
 1. alphabet = alphabet nfa
 2. allStates = subset of \mathcal{P} (allStates nfa)
 3. finalStates = $\{s \in \text{allStates} \mid s = \{n_0, n_1, \dots, n_k\} \text{ and } n_i \in \text{finalStates nfa}\}$
 4. startState =
 5. transitions =

How to build the transition function

NFA to DFA Algorithm

(11.8)

Input: An NFA over alphabet A with transition function T_N .

Output: A DFA over A with transition function T_D that accepts the same language as the NFA. The states of the DFA are represented as certain subsets of NFA states.

1. The DFA start state is $\lambda(s)$, where s is the NFA start state. Now perform Step 2 for this DFA start state.
2. If $\{s_1, \dots, s_n\}$ is a DFA state and $a \in A$, then construct the following DFA state as a DFA table entry in either of two ways:

$$\begin{aligned} T_D(\{s_1, \dots, s_n\}, a) &= \lambda(T_N(s_1, a) \cup \dots \cup T_N(s_n, a)) && \text{(closure of union)} \\ &= \lambda(T_N(s_1, a)) \cup \dots \cup \lambda(T_N(s_n, a)) && \text{(union of closure).} \end{aligned}$$

Repeat Step 2 for each new DFA state constructed in this way.

3. A DFA state is final if one of its elements is an NFA final state.

States are not Sets of States

- In the book, a single state in the DFA is a circle containing a label like $\{n_0, n_1, \dots, n_k\}$
- The text goes freely from the single DFAstate $\{n_0, n_1, \dots, n_k\}$ to the individual NFAstates n_0, n_1, \dots, n_k
- In ML, we have to be explicit about this:
 - ▶ Use the functions
 - DFAState: NFAstate list \rightarrow DFAstate
 - NFAState: DFAstate \rightarrow NFAstate list

In ML

```
> val DFASState = fn : State/1 list -> State/1
    val NFASState = fn : State/1 -> State/1 list
val n0 = newState();
val n1 = newState();
val n2 = newState();
val d0 = DFASState [];
val d1 = DFASState [n0, n1];
val d2 = DFASState [n0, n2];
val d3 = DFASState [n1, n2];
val d4 = DFASState [n1, n0, n2];

> val n0 = State "q157" : State/1
> val n1 = State "q158" : State/1

> val d4 = State "q164" : State/1
```

```
val testDFASState1 = assert (DFASState[] = d0);
val testDFASState2 = assert (DFASState[n0, n1] = d1);
val testDFASState3 = assert (DFASState[n1, n0] = d1);
val testDFASState4 = assert (DFASState[n2, n1] = d3);
val testDFASState5 = assert (DFASState[n2, n0] = d2);
val testDFASState6 = assert (DFASState[n0, n1, n2] = d4);
val testDFASState7 = assert (DFASState[n0, n1, n1, n2] = d4);
```

```
val testNFASState1 = assert (NFASState d0 = []);
val testNFASState2 = assert (NFASState d1 = [n0, n1]);
val testNFASState3 = assert (NFASState d3 = [n1, n2]);
val testNFASState4 = assert (NFASState d2 = [n0, n2]);
val testNFASState5 = assert (NFASState d4 = [n0, n1, n2]);
```

In conventional notation

- **DFASState** $\{n_0, n_1, \dots, n_k\}$ = the single state $\{n_0, n_1, \dots, n_k\}_d$
- **NFASState** $\{n_0, n_1, \dots, n_k\}_d$ = the set $\{n_0, n_1, \dots, n_k\}$