بسم اللّه الرّحمن الرّحیم

دانشگاه صنعتی اصفهان ـ دانشکدهٔ مهندسی برق و کامپیوتر
(نیم‌سال تحصیلی ۴۰۲۲)

کامپایلر

حسین فلسفین

Conversion of an NFA to a DFA

The general idea behind the subset construction is that each state of the constructed DFA corresponds to a set of NFA states. After reading input $a_1 a_2 \cdots a_n$, the DFA is in that state which corresponds to the set of states that the NFA can reach, from its start state, following paths labeled $a_1 a_2 \cdots a_n$.

It is possible that the number of DFA states is exponential in the number of NFA states, which could lead to difficulties when we try to implement this DFA. However, part of the power of the automaton-based approach to lexical analysis is that for real languages, the NFA and DFA have approximately the same number of states, and the exponential behavior is not seen.

The subset construction of a DFA from an NFA.

Algorithm 3.20: The *subset construction* of a DFA from an NFA.

INPUT: An NFA N.

OUTPUT: A DFA D accepting the same language as N.

برای توصیف فرایند تبدیل از کتاب سیپسر استفاده کرده‌ام

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

PROOF IDEA If a language is recognized by an NFA, then we must show the existence of a DFA that also recognizes it. The idea is to convert the NFA into an equivalent DFA that simulates the NFA.

Recall the "reader as automaton" strategy for designing finite automata. How would you simulate the NFA if you were pretending to be a DFA? What do you need to keep track of as the input string is processed? In the examples of NFAs, you kept track of the various branches of the computation by placing a finger on each state that could be active at given points in the input. You updated the simulation by moving, adding, and removing fingers according to the way the NFA operates. All you needed to keep track of was the set of states having fingers on them.

If k is the number of states of the NFA, it has 2^k subsets of states. Each subset corresponds to one of the possibilities that the DFA must remember, so the DFA simulating the NFA will have 2^k states. Now we need to figure out which will be the start state and accept states of the DFA, and what will be its transition function. We can discuss this more easily after setting up some formal notation.

PROOF Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A. We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A. Before doing the full construction, let's first consider the easier case wherein N has no ε arrows. Later we take the ε arrows into account.

1. $Q' = \mathcal{P}(Q)$.
   Every state of M is a set of states of N. Recall that $\mathcal{P}(Q)$ is the set of subsets of Q.
2. For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R, a) = \{q \in Q | \ q \in \delta(r, a) \text{ for some } r \in R\}$.
   If R is a state of M, it is also a set of states of N. When M reads a symbol a in state R, it shows where a takes each state in R. Because each state may go to a set of states, we take the union of all these sets. Another way to write this expression is
   $$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a).$$
3. $q_0' = \{q_0\}$.
   M starts in the state corresponding to the collection containing just the start state of N.
4. $F' = \{R \in Q' | \ R \text{ contains an accept state of } N\}$.
   The machine M accepts if one of the possible states that N could be in at this point is an accept state.

Now we need to consider the ε arrows. To do so, we set up an extra bit of notation. For any state R of M, we define $E(R)$ to be the collection of states that can be reached from members of R by going only along ε arrows, including the members of R themselves. Formally, for $R \subseteq Q$ let

$$E(R) = \{q |\ q \text{ can be reached from } R \text{ by traveling along 0 or more } \varepsilon \text{ arrows}\}.$$

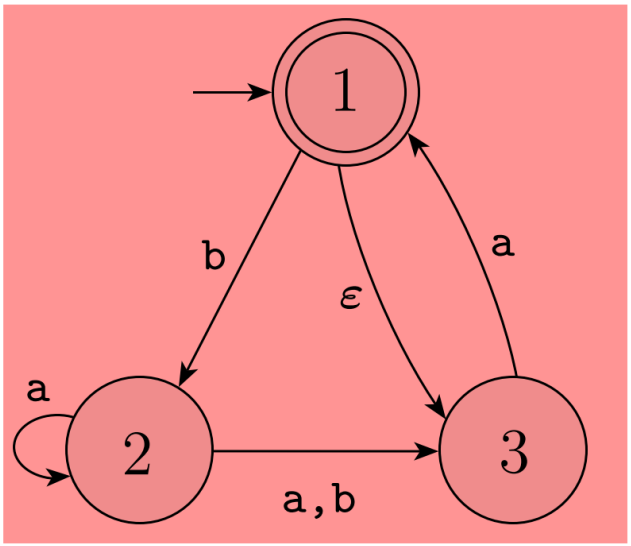
Then we modify the transition function of M to place additional fingers on all states that can be reached by going along ε arrows after every step. Replacing $\delta(r, a)$ by $E(\delta(r, a))$ achieves this effect. Thus

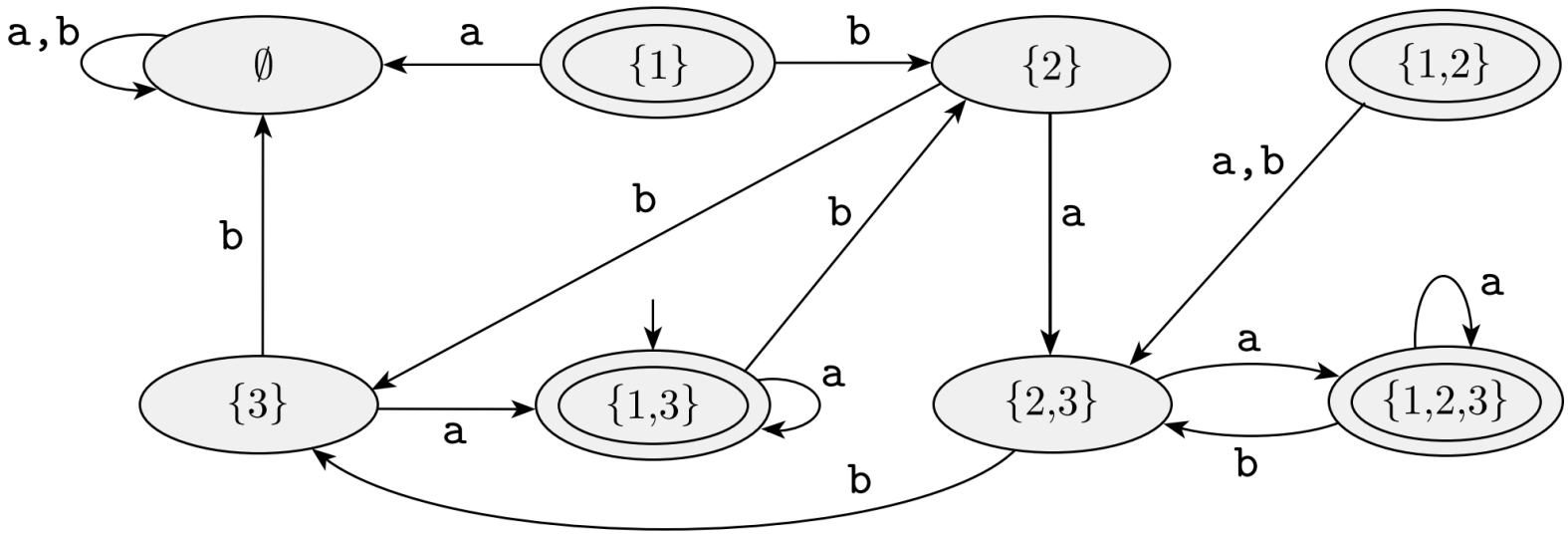
$$\delta'(R, a) = \{q \in Q |\ q \in E(\delta(r, a)) \text{ for some } r \in R\}.$$

Additionally, we need to modify the start state of M to move the fingers initially to all possible states that can be reached from the start state of N along the ε arrows. Changing $q_0{}'$ to be $E(\{q_0\})$ achieves this effect. We have now completed the construction of the DFA M that simulates the NFA N.

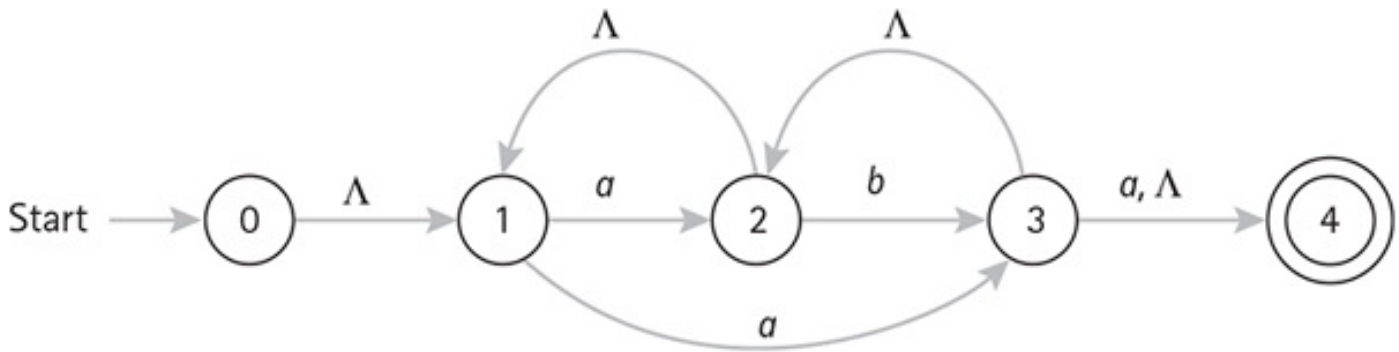
The construction of M obviously works correctly. At every step in the computation of M on an input, it clearly enters a state that corresponds to the subset of states that N could be in at that point. Thus our proof is complete.

An example:





An example:



	T_N	a	b	Λ
start	0	$\varnothing$	$\varnothing$	$\{1\}$
	1	$\{2,3\}$	$\varnothing$	$\varnothing$
	2	$\varnothing$	$\{3\}$	$\{1\}$
	3	$\{4\}$	$\varnothing$	$\{2,4\}$
final	4	$\varnothing$	$\varnothing$	$\varnothing$

	T_D	a	b
start	$\{0,1\}$	$\{1,2,3,4\}$	$\varnothing$
final	$\{1,2,3,4\}$	$\{1,2,3,4\}$	$\{1,2,3,4\}$
	$\varnothing$	$\varnothing$	$\varnothing$

یک مثال که کل فرایند را در بر دارد (کتاب اژدها)

a	{ action A_1 for pattern p_1 }
abb	{ action A_2 for pattern p_2 }
$\mathbf{a}^*\mathbf{b}^+$	{ action A_3 for pattern p_3 }

Note that these three patterns present some conflicts of the type discussed in Section 3.5.3. In particular, string *abb* matches both the second and third patterns, but we shall consider it a lexeme for pattern p_2, since that pattern is listed first in the above `Lex` program. Then, input strings such as $aabbb\cdots$ have many prefixes that match the third pattern. The `Lex` rule is to take the longest, so we continue reading b's, until another a is met, whereupon we report the lexeme to be the initial a's followed by as many b's as there are.

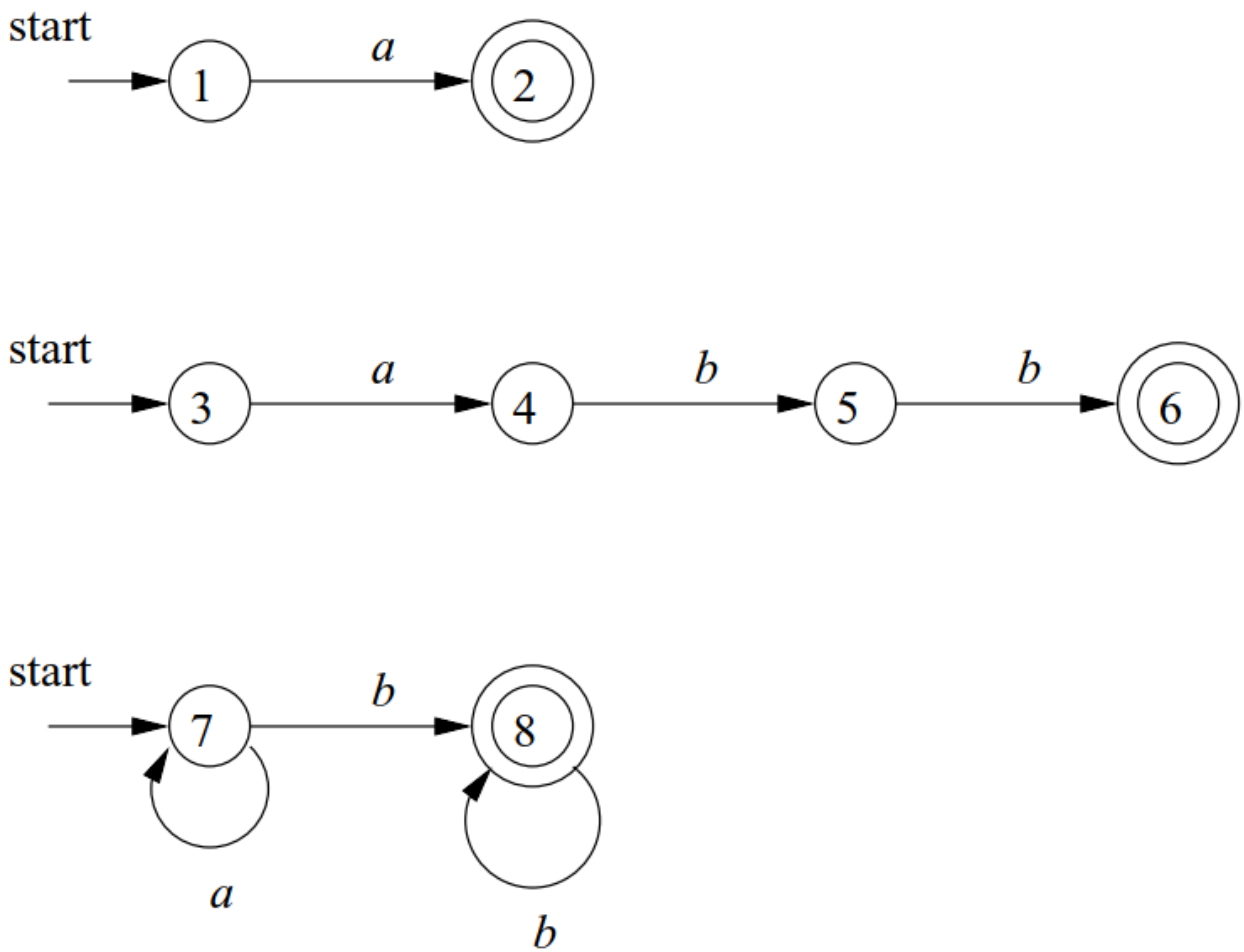


Figure 3.51: NFA's for **a**, **abb**, and $\mathbf{a}^*\mathbf{b}^+$

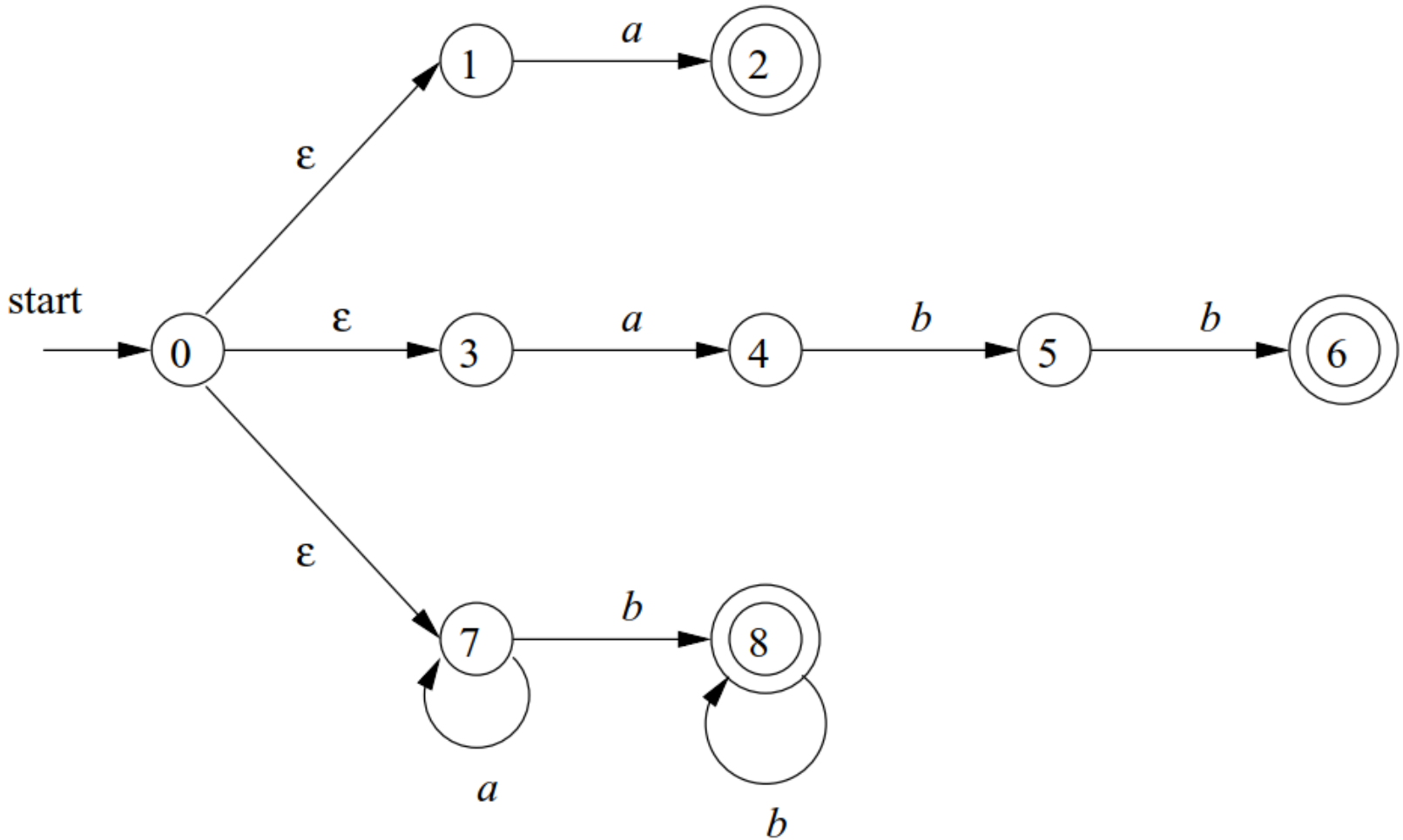


Figure 3.52: Combined NFA

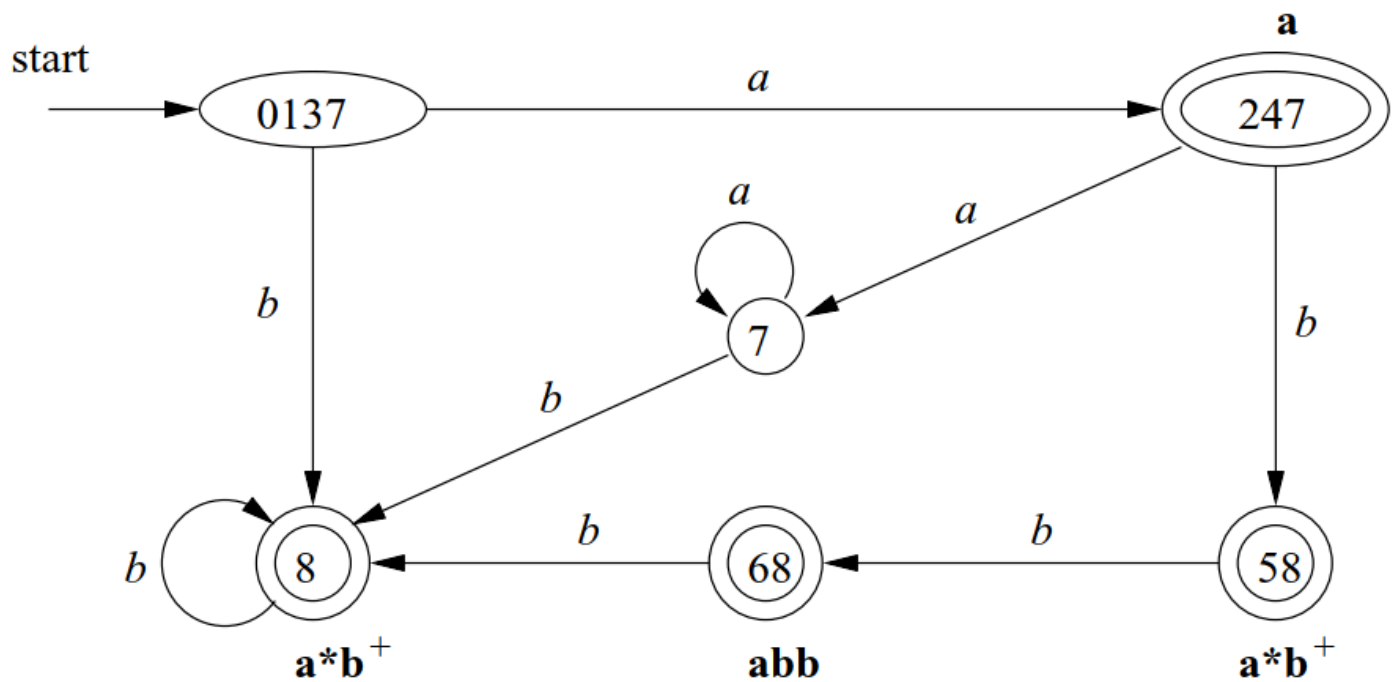


Figure 3.54: Transition graph for DFA handling the patterns **a**, **abb**, and **a*****b**$^+$

Example 3.29: Suppose the DFA of Fig. 3.54 is given input *abba*. The sequence of states entered is 0137, 247, 58, 68, and at the final *a* there is no transition out of state 68. Thus, we consider the sequence from the end, and in this case, 68 itself is an accepting state that reports pattern $p_2 =$ **abb**. □

Within each DFA state, if there are one or more accepting NFA states, determine the first pattern whose accepting state is represented, and make that pattern the output of the DFA state.

We simulate the DFA until at some point there is no next state (or strictly speaking, the next state is ∅, the dead state corresponding to the empty set of NFA states). At that point, we back up through the sequence of states we entered and, as soon as we meet an accepting DFA state, we perform the action associated with the pattern for that state.

3.9.6 Minimizing the Number of States of a DFA

Minimization of Finite Automata. For every DFA there is a minimum-state DFA accepting the same language. Moreover, the minimum-state DFA for a given language is unique except for the names given to the various states.

There can be many DFA's that recognize the same language. If we implement a lexical analyzer as a DFA, we would generally prefer a DFA with as few states as possible, since each state requires entries in the table that describes the lexical analyzer.

Any DFA defines a unique language, but the converse is not true. For a given language, there are many DFA's that accept it. There may be a considerable difference in the number of states of such equivalent automata. Representation of an automaton for the purpose of computation requires space proportional to the number of states. For storage efficiency, it is desirable to reduce the number of states as far as possible.

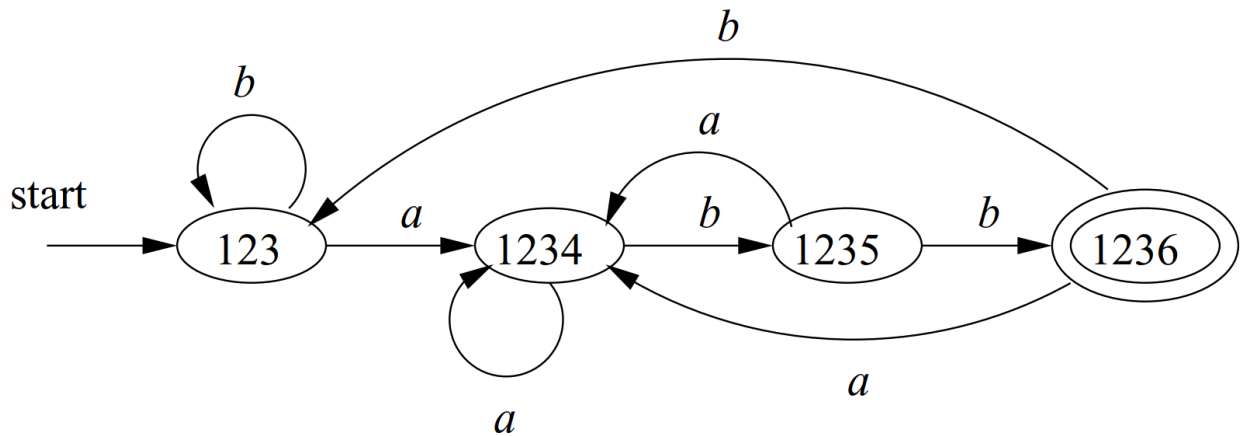
کمینه یکتاست DFA

The matter of the names of states is minor. We shall say that two automata are the same up to state names if one can be transformed into the other by doing nothing more than changing the names of states. It turns out that there is always a unique (up to state names) minimum state DFA for any regular language. Moreover, this minimum-state DFA can be constructed from any DFA for the same language by grouping sets of equivalent states.

Distinguishability

In order to understand the algorithm for creating the partition of states that converts any DFA into its minimum-state equivalent DFA, we need to see how input strings distinguish states from one another. We say that string x distinguishes state s from state t if exactly one of the states reached from s and t by following the path with label x is an accepting state. State s is distinguishable from state t if there is some string that distinguishes them.

Example 3.38 : The empty string distinguishes any accepting state from any nonaccepting state. In Fig. 3.36, the string bb distinguishes state A from state B, since bb takes A to a nonaccepting state C, but takes B to accepting state E. □



نگاهی کلی به چگونگی عملکرد الگوریتم کمینه‌سازی تعداد استیت‌ها

The state-minimization algorithm works by partitioning the states of a DFA into groups of states that cannot be distinguished. Each group of states is then merged into a single state of the minimum-state DFA. The algorithm works by maintaining a partition, whose groups are sets of states that have not yet been distinguished, while any two states from different groups are known to be distinguishable. When the partition cannot be refined further by breaking any group into smaller groups, we have the minimum-state DFA.

Initially, the partition consists of two groups: the accepting states and the nonaccepting states. The fundamental step is to take some group of the current partition, say $A = \{s_1, s_2, \ldots, s_k\}$, and some input symbol a, and see whether a can be used to distinguish between any states in group A. We examine the transitions from each of $s_1, s_2, \ldots, s_k$ on input a, and if the states reached fall into two or more groups of the current partition, we split A into a collection of groups, so that s_i and s_j are in the same group if and only if they go to the same group on input a. We repeat this process of splitting groups, until for no group, and for no input symbol, can the group be split further. The idea is formalized in the next algorithm.

Algorithm 3.39: Minimizing the number of states of a DFA.

INPUT: A DFA D with set of states S, input alphabet Σ, start state s_0, and set of accepting states F.

OUTPUT: A DFA D' accepting the same language as D and having as few states as possible.

METHOD:

1. Start with an initial partition Π with two groups, F and $S - F$, the accepting and nonaccepting states of D.

2. Apply the procedure of Fig. 3.64 to construct a new partition Π_{new}.

```
initially, let Π_new = Π;
for ( each group G of Π ) {
        partition G into subgroups such that two states s and t
                are in the same subgroup if and only if for all
                input symbols a, states s and t have transitions on a
                to states in the same group of Π;
        /* at worst, a state will be in a subgroup by itself */
        replace G in Π_new by the set of all subgroups formed;
}
```

Figure 3.64: Construction of Π_{new}

3. If $\Pi_{\text{new}} = \Pi$, let $\Pi_{\text{final}} = \Pi$ and continue with step (4). Otherwise, repeat step (2) with Π_{new} in place of Π.

4. Choose one state in each group of Π_{final} as the *representative* for that group. The representatives will be the states of the minimum-state DFA D'. The other components of D' are constructed as follows:

یک بیان دیگر ا(ز کتاب لینز)

procedure: mark

1. Remove all inaccessible states. This can be done by enumerating all simple paths of the graph of the dfa starting at the initial state. Any state not part of some path is inaccessible.

2. Consider all pairs of states (p, q). If $p \in F$ and $q \notin F$ or vice versa, mark the pair (p, q) as distinguishable.

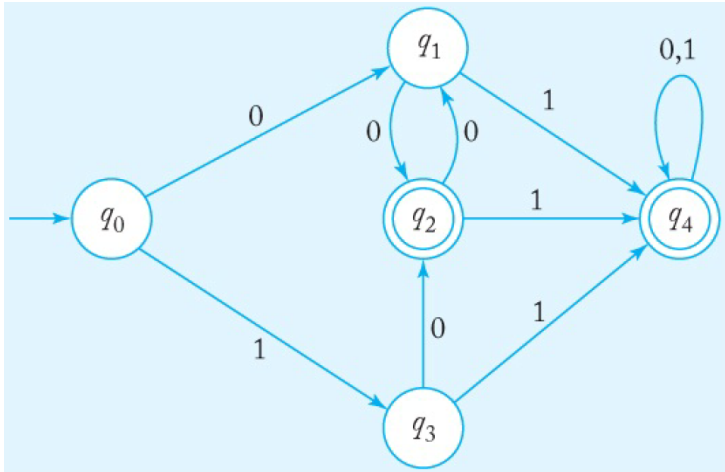
3. Repeat the following step until no previously unmarked pairs are marked. For all pairs (p, q) and all $a \in \Sigma$, compute $\delta(p, a) = p_a$ and $\delta(q, a) = q_a$. If the pair (p_a, q_a) is marked as distinguishable, mark (p, q) as distinguishable.

(a) The start state of D' is the representative of the group containing the start state of D.

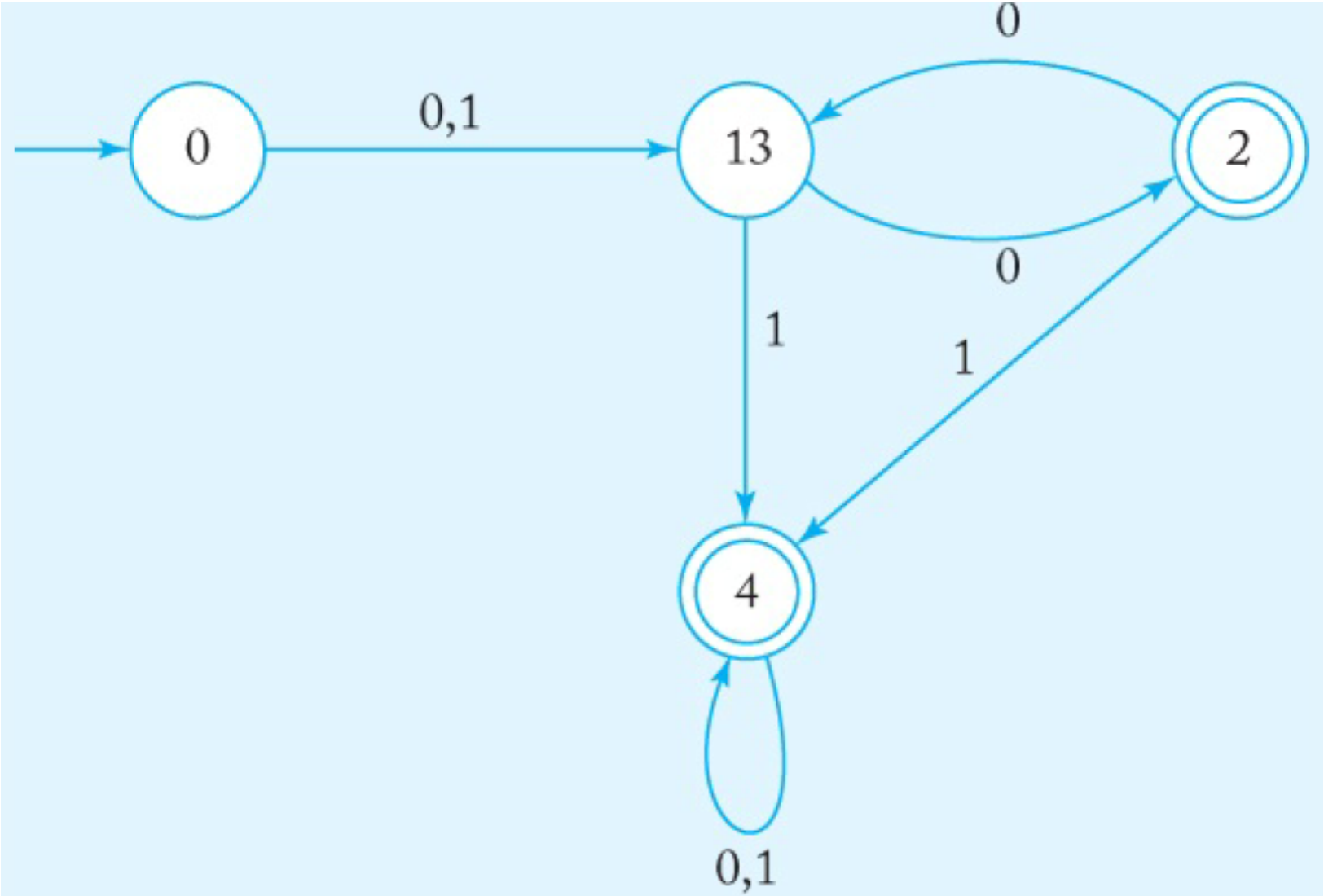
(b) The accepting states of D' are the representatives of those groups that contain an accepting state of D. Note that each group contains either only accepting states, or only nonaccepting states, because we started by separating those two classes of states, and the procedure of Fig. 3.64 always forms new groups that are subgroups of previously constructed groups.

(c) Let s be the representative of some group G of Π_{final}, and let the transition of D from s on input a be to state t. Let r be the representative of t's group H. Then in D', there is a transition from s to r on input a. Note that in D, every state in group G must go to some state of group H on input a, or else, group G would have been split according to Fig. 3.64.

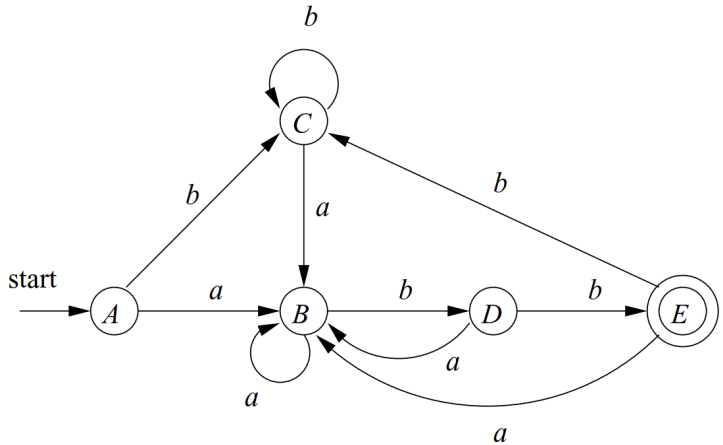
Example



In the second step of procedure mark we partition the state set into final and nonfinal states to get two equivalence classes $\{q_0, q_1, q_3\}$ *and* $\{q_2, q_4\}$. *In the next step, when we compute* $\delta(q_0, 0) = q_1$ *and* $\delta(q_1, 0) = q_2$, *we recognize that* q_0 *and* q_1 *are distinguishable, so we put them into different sets. So* $\{q_0, q_1, q_3\}$ *is split into* $\{q_0\}$ *and* $\{q_1, q_3\}$. *Also, since* $\delta(q_2, 0) = q_3$ *and* $\delta(q_4, 0) = q_4$, *the class* $\{q_2, q_4\}$ *is split into* $\{q_2\}$ *and* $\{q_4\}$. *The rest of the computations show that no further splitting is needed.*


0
0,1
13
2
0
0
1
1
4
0,1

Example



Example 3.40: Let us reconsider the DFA of Fig. 3.36. The initial partition consists of the two groups $\{A, B, C, D\}\{E\}$, which are respectively the nonaccepting states and the accepting states. To construct Π_{new}, the procedure of Fig. 3.64 considers both groups and inputs a and b. The group $\{E\}$ cannot be split, because it has only one state, so $\{E\}$ will remain intact in Π_{new}.

The other group $\{A, B, C, D\}$ can be split, so we must consider the effect of each input symbol. On input a, each of these states goes to state B, so there is no way to distinguish these states using strings that begin with a. On input b, states A, B, and C go to members of group $\{A, B, C, D\}$, while state D goes to E, a member of another group. Thus, in Π_{new}, group $\{A, B, C, D\}$ is split into $\{A, B, C\}\{D\}$, and Π_{new} for this round is $\{A, B, C\}\{D\}\{E\}$.

In the next round, we can split $\{A, B, C\}$ into $\{A, C\}\{B\}$, since A and C each go to a member of $\{A, B, C\}$ on input b, while B goes to a member of another group, $\{D\}$. Thus, after the second round, $\Pi_{\text{new}} = \{A, C\}\{B\}\{D\}\{E\}$. For the third round, we cannot split the one remaining group with more than one state, since A and C each go to the same state (and therefore to the same group) on each input. We conclude that $\Pi_{\text{final}} = \{A, C\}\{B\}\{D\}\{E\}$.

Now, we shall construct the minimum-state DFA. It has four states, corresponding to the four groups of Π_{final}, and let us pick A, B, D, and E as the representatives of these groups. The initial state is A, and the only accepting state is E. Figure 3.65 shows the transition function for the DFA. For instance, the transition from state E on input b is to A, since in the original DFA, E goes to C on input b, and A is the representative of C's group. For the same reason, the transition on b from state A is to A itself, while all other transitions are as in Fig. 3.36. □

STATE	a	b
A	B	A
B	B	D
D	B	E
E	B	A

Figure 3.65: Transition table of minimum-state DFA