Books are the quietest and most constant friends; they are the most accessible and wisest counsellors, and the most patient teacher.

----Benjamin Franklin

年轻的时候以为不读书不足以了解人生，直到后来才发现如果不了解人生，是读不懂书的。读书的意义大概就是用生活所感去读书，用读书所得去生活吧。

----杨绛

Information Science and Technology College of
Northeast Normal University

Compiling and Running of Program

Dr. Zheng Xiaojuan
Professor

September. 2019

Summary for last lecture

- **Formal definition of *DFA* （∑， SS, S0, Φ, TS）**
- **Two way of representations of DFA (table; graph)**
- **What are**
  - **String acceptable by a DFA;**
  - **Set of strings (language) accepted by a DFA;**
- **How to implement a DFA?**
  - **Meaning;**
  - **Table based; -- transforming table & driver**
  - **Graph based; -- case statements**
- **Utilizing DFA to define**
  - **a set of strings, lexical structure of programming languages;**

Minimum requirement

Outline

2.1 Overview

- **2.1.1 General Function of a Scanner**
- **2.1.2 Some Issues about Scanning**

2.2 Finite Automata

- **2.2.1 Definition and Implementation of DFA**
- **2.2.2 Non-Determinate Finite Automata**
- **2.2.3 Transforming NFA into DFA**
- **2.2.4 Minimizing DFA**

2.3 Regular Expressions

- **2.3.1 Definition of Regular Expressions**
- **2.3.2 Regular Definition**
- **2.3.4 From Regular Expression to DFA**

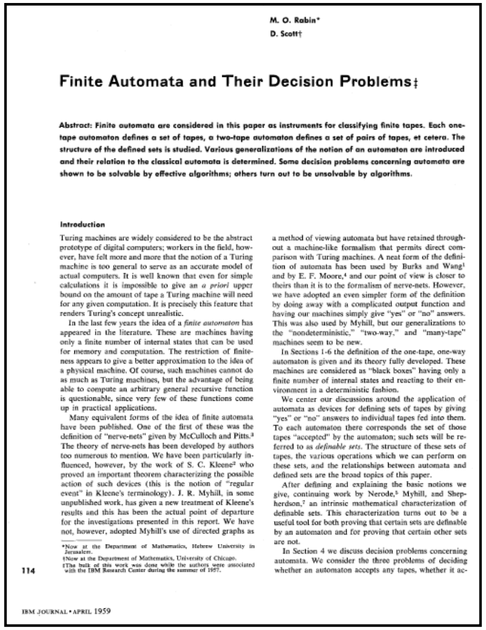
2.4 Design and Implementation of a Scanner

- **2.4.1 Developing a Scanner from DFA**
- **2.4.2 A Scanner Generator – Lex**

Definition of NFA

Non-Determinate Finite Automata

- 1976 年图灵奖： For their joint paper "Finite Automata and Their Decision Problem," which introduced the idea of nondeterministic machines, which has proved to be an enormously valuable concept. Their (Scott & Rabin) classic paper has been a continuous source of inspiration for subsequent work in this field.



M. O. Rabin*
D. Scott†

Finite Automata and Their Decision Problems‡

Abstract: Finite automata are considered in this paper as instruments for classifying finite tapes. Each one-tape automaton defines a set of tapes, a two-tape automaton defines a set of pairs of tapes, et cetera. The structure of the defined sets is studied. Various generalizations of the notion of an automaton are introduced and their relation to the classical automata is determined. Some decision problems concerning automata are shown to be solvable by effective algorithms; others turn out to be unsolvable by algorithms.

Introduction

Turing machines are widely considered to be the abstract prototype of digital computers; workers in the field, however, have felt more and more that the notion of a Turing machine is too general to serve as an accurate model of actual computers. It is well known that even for simple calculations it is impossible to give an *a priori* upper bound on the amount of tape a Turing machine will need for any given computation. It is precisely this feature that renders Turing's concept unrealistic.

In the last few years the idea of a *finite automaton* has appeared in the literature. These are machines having only a finite number of internal states that can be used for memory and computation. The restriction of finiteness appears to give a better approximation to the idea of a physical machine. Of course, such machines cannot do as much as Turing machines, but the advantage of being able to compute an arbitrary general recursive function is questionable, since very few of these functions come up in practical applications.

Many equivalent forms of the idea of finite automata have been published. One of the first of these was the definition of "nerve-nets" given by McCulloch and Pitts.[3] The theory of nerve-nets has been developed by authors too numerous to mention. We have been particularly influenced, however, by the work of S. C. Kleene[2] who proved an important theorem characterizing the possible action of such devices (this is the notion of "regular event" in Kleene's terminology). J. R. Myhill, in some unpublished work, has given a new treatment of Kleene's results and this has been the actual point of departure for the investigations presented in this report. We have not, however, adopted Myhill's use of directed graphs as a method of viewing automata but have retained throughout a machine-like formalism that permits direct comparison with Turing machines. A neat form of the definition of automata has been used by Burks and Wang[1] and by E. F. Moore,[4] and our point of view is closer to theirs than it is to the formalism of nerve-nets. However, we have adopted an even simpler form of the definition by doing away with a complicated output function and having our machines simply give "yes" or "no" answers. This was also used by Myhill, but our generalizations to the "nondeterministic," "two-way," and "many-tape" machines seem to be new.

In Sections 1-6 the definition of the one-tape, one-way automaton is given and its theory fully developed. These machines are considered as "black boxes" having only a finite number of internal states and reacting to their environment in a deterministic fashion.

We center our discussions around the application of automata as devices for defining sets of tapes by giving "yes" or "no" answers to individual tapes fed into them. To each automaton there corresponds the set of those tapes "accepted" by the automaton; such sets will be referred to as *definable sets*. The structure of these sets of tapes, the various operations which we can perform on these sets, and the relationships between automata and defined sets are the broad topics of this paper.

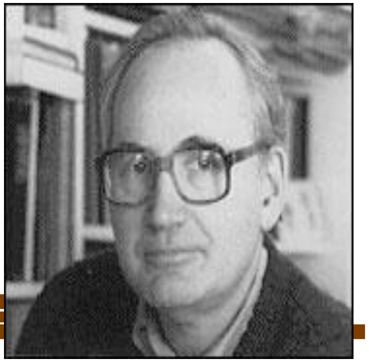
After defining and explaining the basic notions we give, continuing work by Nerode,[5] Myhill, and Shepherdson,[7] an intrinsic mathematical characterization of definable sets. This characterization turns out to be a useful tool for both proving that certain sets are definable by an automaton and for proving that certain other sets are not.

In Section 4 we discuss decision problems concerning automata. We consider the three problems of deciding whether an automaton accepts any tapes, whether it ac-

*Now at the Department of Mathematics, Hebrew University in Jerusalem.
†Now at the Department of Mathematics, University of Chicago.
‡The bulk of this work was done while the authors were associated with the IBM Research Center during the summer of 1957.

114

IBM JOURNAL • APRIL 1959



Michael O. Rabin　　Dana S. Scott

Information Science and Technology College of
Northeast Normal University

Formal Definition

- （∑，SS, SS0, Φ, TS）
- ∑（alphabet）Sigma，set of allowed characters, each character can be called as input symbol;
- SS = {S0, S1, S2, ……}，a finite set, each element is called state;
- $SS^0 \subseteq SS$, set of start states（起始状态可以从不同状态出发）
- Φ： $SS \times \Sigma \rightarrow 2^{SS}$（power set of SS）$\cup \{\perp\}$, transforming function
- $TS \subseteq SS$, set of terminal (accept) states
- Note：Φ is a function which accepts a state and a symbol and returns a set of states or ⊥(no definition)；

Differences between DFA & NFA

	DFA	NFA
Start state	**One start state**	**Set of start states**
ε	×	√
T (S, a)	**S' or ⊥**	**{S1, …, Sn} or ⊥**
implementation	**easy**	**Non-deterministic**

Example of NFA

$\sum$: {a, b, c, d}

SS: {S^0, $S1^0$, S2, S*}

Set of Start state: {S^0 , $S1^0$}

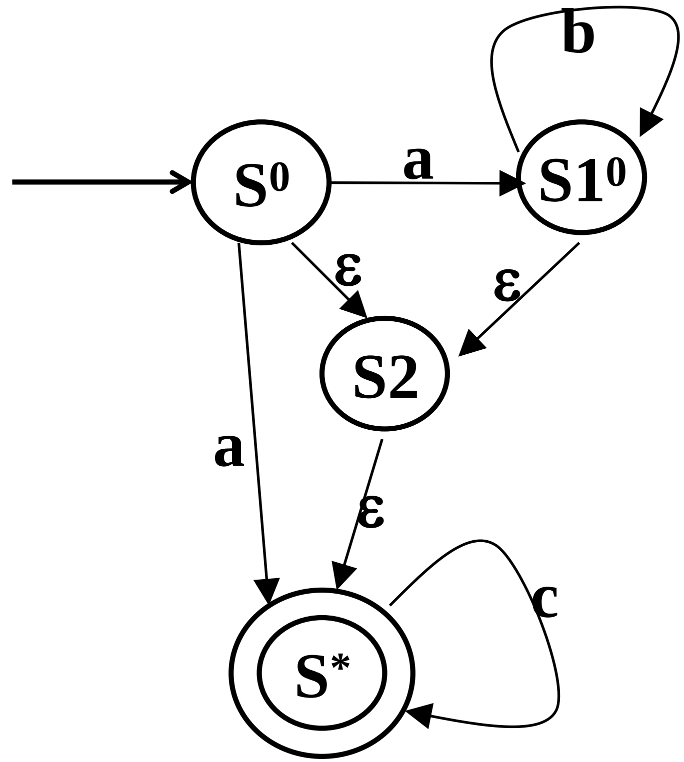
Set of terminal states: {S^*}

Φ: {(S^0,a)$\rightarrow$ {$S1^0$, S*},(S^0,ε)$\rightarrow$ {S2},

($S1^0$,b)$\rightarrow${$S1^0$}, ($S1^0$, ε)$\rightarrow${S2},

(S2, ε)$\rightarrow${S^*},

(S^*, c)$\rightarrow${S^*}



From NFA to DFA

- 定义：对于任何两个有限自动机M和M’，如果L(M)=L(M’)，则称M与M’等价
- 自动机理论中一个重要的结论：判定两个自动机等价性的算法是存在的
- 对于每个NFA M存在一个DFA M’，使得L(M)=L(M’)
- DFA与NFA识别能力相同!

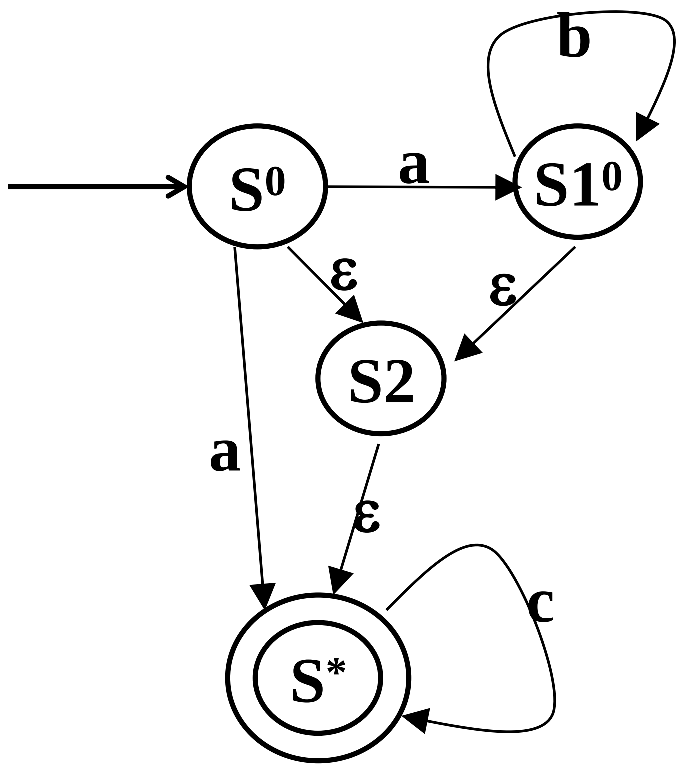
Main Idea

- **Solve two problems**
  - **ε edge**
    - **ε-closure (SS)** ε闭包
  - **Merging those edges with the same symbol**
    - **NextStates(SS, a)**
- **Conversion of NFA to DFA**
  - **Using a set of states in NFA as one state in DFA**
  - **Assuring accepting the same set of strings**

The process to calculate ε-closure (ε闭包)

- **For a given NFA *A*, and a set of states *SS*,**
  - **ε-closure(*SS*) = *SS*;**
  - **If there exists a state *s* in *SS*, which has a ε-edge referring to a state *s’* and *s’*∉ε-closure(*SS*), add to *s’* to ε-closure(*SS*) ;**
  - **Repeat until there is no state having ε-edge to states that is not in ε-closure(*SS*) ;**

ε-closure (ε闭包) -- Example



ε-closure($\{S^0, S1^0\}$) =

① $\{S^0, S1^0\}$

② $\{S^0, S1^0, S2\}$

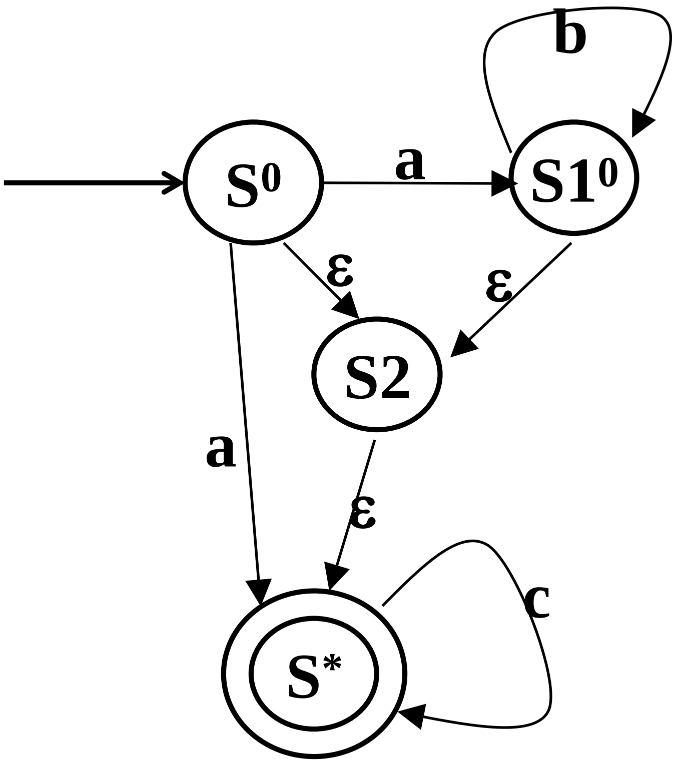
③ $\{S^0, S1^0, S2\}$

④ $\{S^0, S1^0, S2, S^*\}$

Moving States（转向状态）

- **For a given set of states *SS* and a symbol *a* in a NFA *A*,**
  - **NextStates(*SS*, *a*) = {s | if there is a state s1$\in$*SS*, and a edge s1 $\xrightarrow{a}$ s in *A* }**

Moving States



NextStates($\{S^0, S1^0\}$, a)
= $\{S1^0, S^*\}$

NextStates($\{S^0, S1^0\}$, b)
= $\{S1^0\}$

Algorithm：判定两个自动机等价性的算法（子集法）

- **Given a NFA *A* = {∑, SS, SS^0, Φ, TS}**
- **Generating an equivalent DFA *A'* = {∑, SS',S^0, Φ', TS'}**
- **Steps**

  (1) S^0 = ε-closure(SS^0), add S^0 to SS';（确定初始状态）

  (2) select one state S from SS', for any symbol a∈∑,

  let S' = ε-closure(NextStates(S,a)) ,（先转向再空闭包）

  add (S, a) → S' to Φ',（生成转换函数）

  if S'∉SS', add S' to SS';(产生新状态)

  （状态是有限的，n个元素最多2^n个状态）

  (3) repeat (2) until all states are handled;（产生所有新状态)

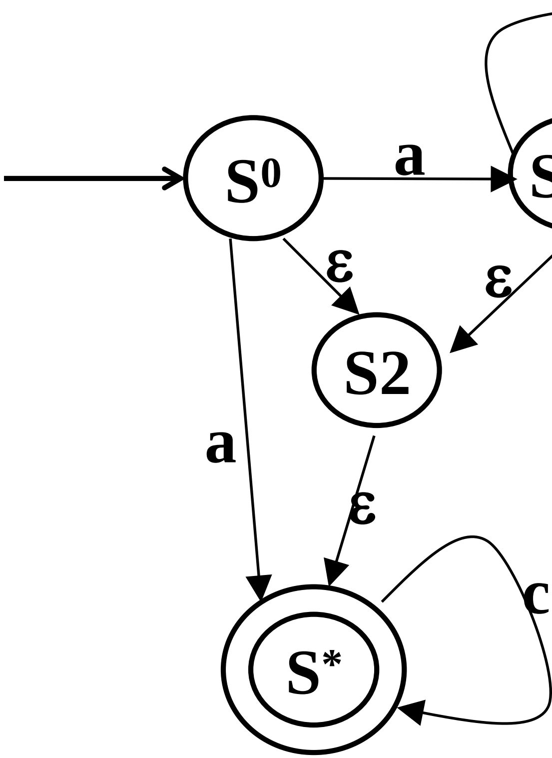
  (4) for a state S in SS', S = {S1, .., Sn}, if there exists Si∈TS, then S is an accept state in *A'*, add S to TS';（确定终止状态）

Example

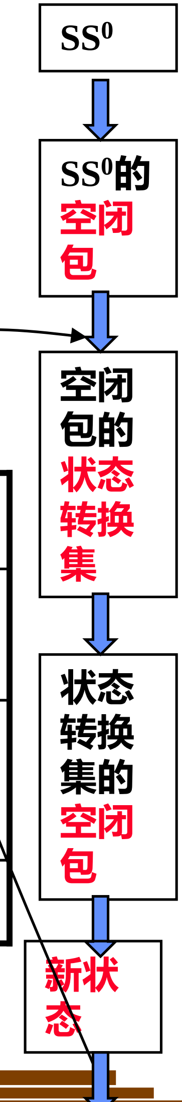
$\sum = \{a, b, c\}$,

$S0 = \varepsilon\text{-closure}(\{S^0, S1^0\})$

$= \{S^0, S1^0, S2, S^*\}$,



	a	b	c
$\{S^0, S1^0, S2, S^*\}$	$\{S1^0, S^*, S2\}$	$\{S1^0, S^*, S2\}$	$\{S^*\}$
$\{S1^0, S^*, S2\}$		$\{S1^0, S^*, S2\}$	$\{S^*\}$
$\{S^*\}$			$\{S^*\}$



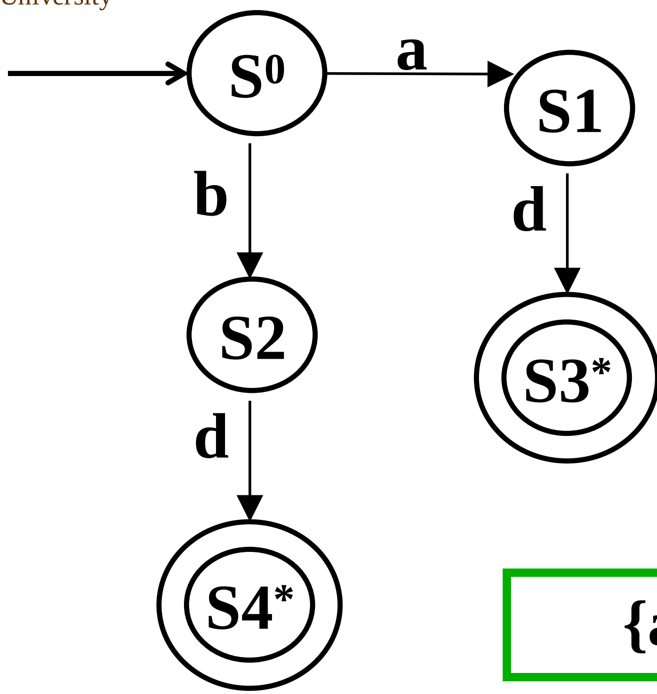
Minimizing DFA

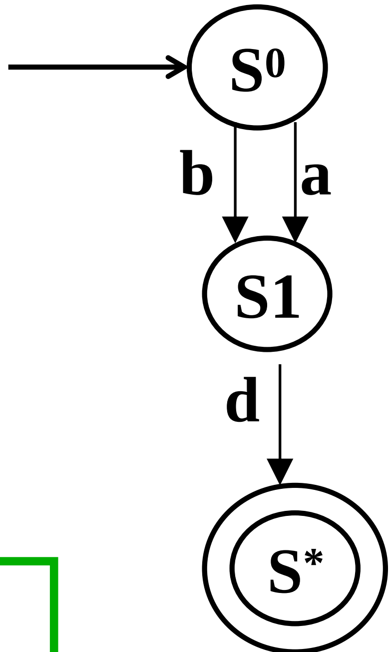
Problem

- **Equivalent of two DFAs**
  - **If the set of strings accepted by two DFAs are the same;**
- **Among those DFAs that accept the same set of strings, the *minimal DFA* refers to the one that has *minimal number of states*;**

How this happens?

Equivalent DFAs





{ad, bd}

There are states that accepting the same set of strings!

Main Idea

- **Equivalent states(等价状态)**
  - **For two states s1 and s2 in a DFA, if treat s1 and s2 as start states and they accept the same set of strings, s1 and s2 will be called *equivalent states*;**
- **Two ways to minimizing DFA**
  - **Merging equivalent states; (状态合并)**
  - ***Splitting non-equivalent states*;（状态分离）**

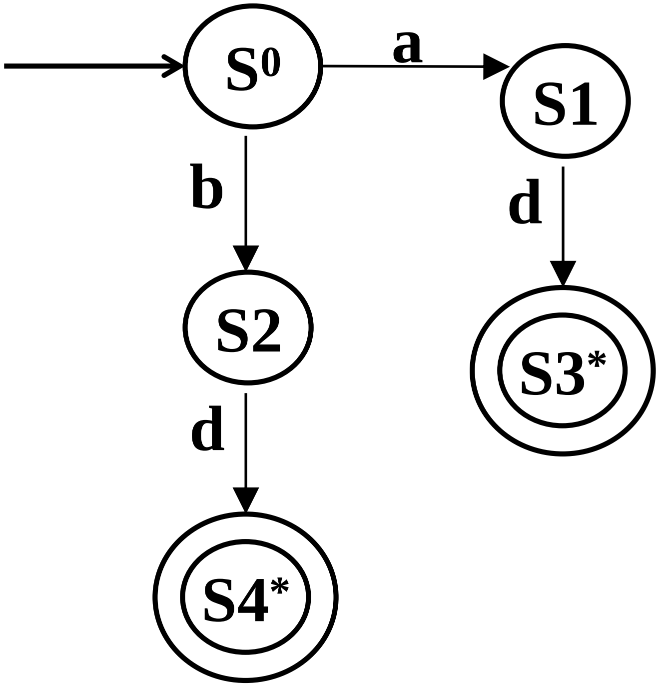
Algorithm

- **Given a DFA $A = \{\Sigma, SS, S^0, \Phi, TS\}$**
- **Generating an equivalent DFA $A' = \{\Sigma, SS', S^{0\prime}, \Phi', TS'\}$**
- **Splitting Steps**
  - **(1) two groups {non-terminal states}, {terminal states};**
  - **(2) select one group of states $SS_i = \{Si1, \ldots, Sin\}$,**
    replace SS_i with *split*(SS_i);
  - **(3) repeat (2) until all groups are handled;（不能再分）**
  - **(4) SS' = set of groups;**
  - **(5) $S^{0\prime}$ is the group consisting of S^0;**
  - **(6) if the group consisting of terminal states of *A*, it is terminal state of *A*';**
  - **(7) Φ': $SS_i \xrightarrow{a} SS_j$, if there is $Si \xrightarrow{a} Sj$ in A, $Si \in SS_i$, $Sj \in SS_j$**

Splitting a Set of States

- **Given**
  - **a DFA *A* = {∑, SS, S^0, Φ, TS};**
  - **Groups of states {SS1, …, SSm}, SS1∪ ……∪SSm = SS;**
  - **SS_i = {Si1,…, Sin},**
- ***split*(SS_i) is to split SS_i into two group G1 and G2,**
  - **For j = 1 to n**
    - **for any a∈∑,**
    - **If (Si1,a) → Sk ∧ (Sij, a) →Sl ∧ Sk and Sl belong to the same group SSp , add Sij to G1;**
    - **Otherwise, add Sij to G2;**

Simple Example

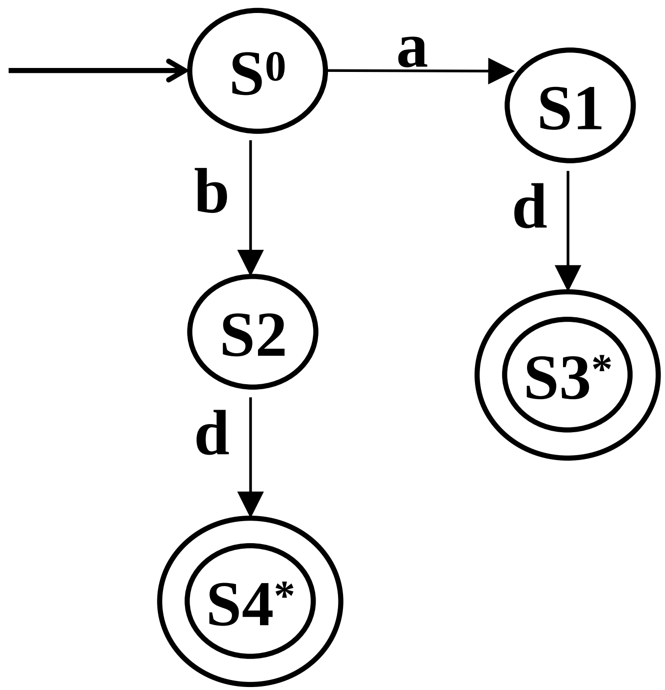


$\{S^0, S1, S2\}, \{S3^*, S4^*\}$

$\{S^0\}$, $\{S1, S2\}$,

$\{S3^*, S4^*\}$

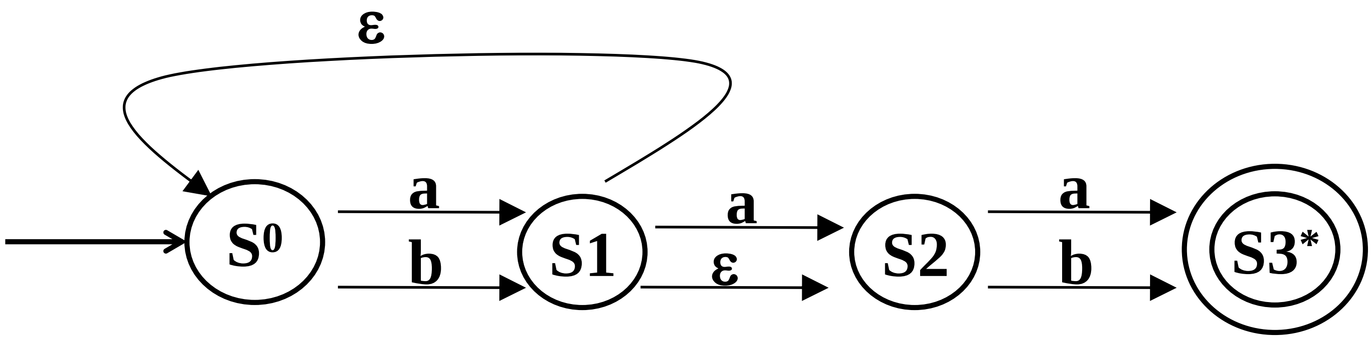
Simple Example

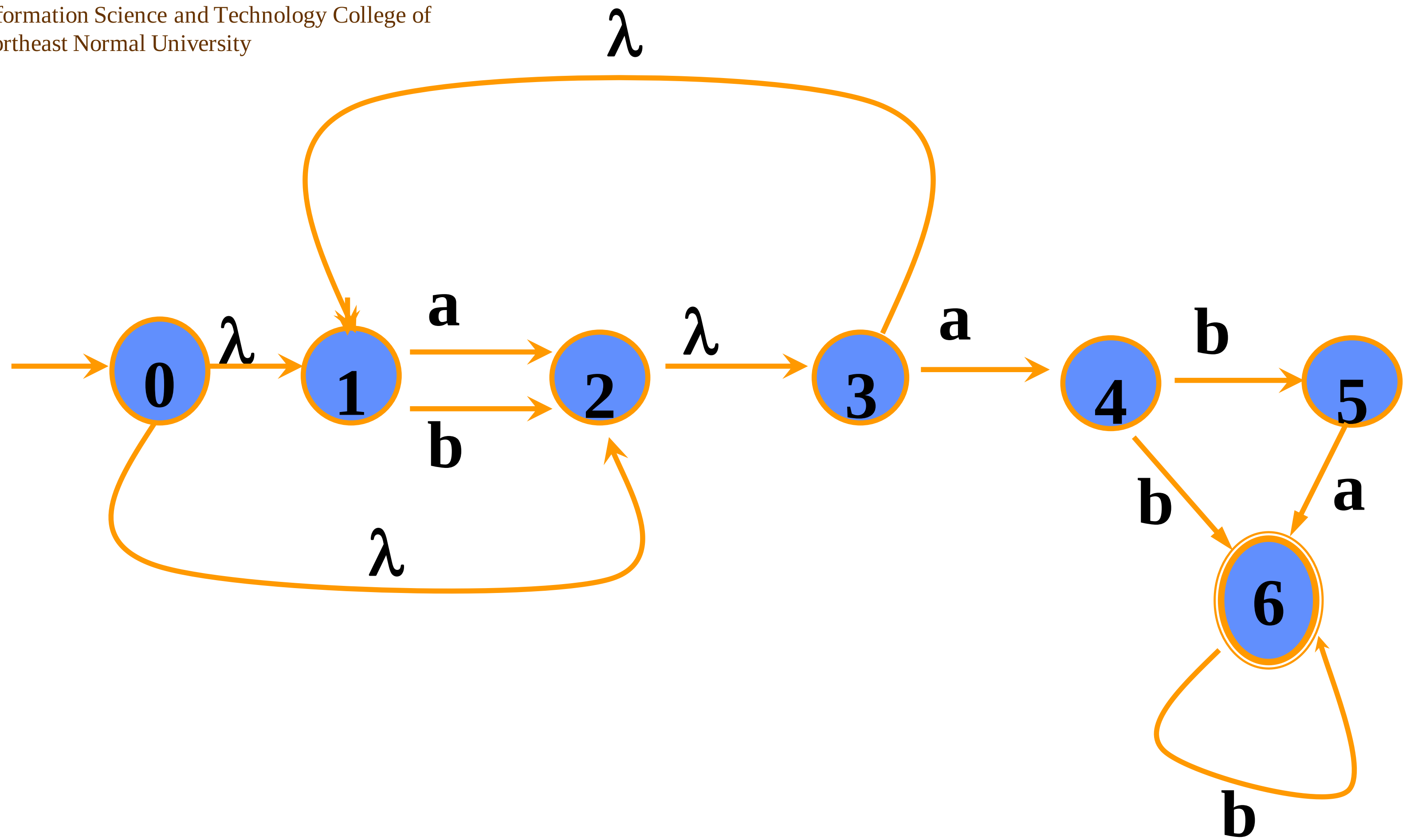


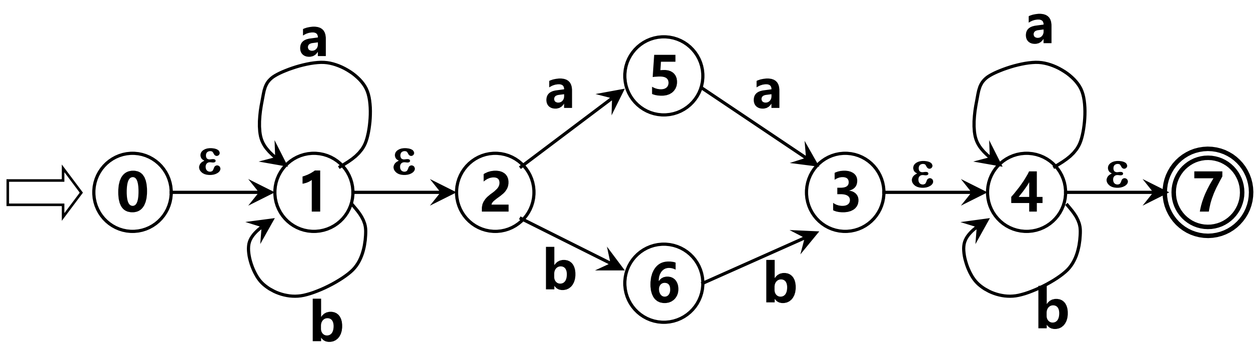
	a	b	d
S^0	S1	S2	⊥
S1	⊥	⊥	S3*
S2	⊥	⊥	S4*
S3*	⊥	⊥	⊥
S4*	⊥	⊥	⊥

Assignment

- **From NFA to DFA, and minimize it;**




λ
0
λ
1
a
b
2
λ
3
a
4
b
5
b
a
λ
6
b


0
ε
1
a
b
ε
2
a
5
a
b
6
b
3
ε
4
a
b
ε
7