Finite Automata
(有限自动机 或 有穷自动机)

Motivation

An Example

Slides Courtesy of Stanford CS154 and etc

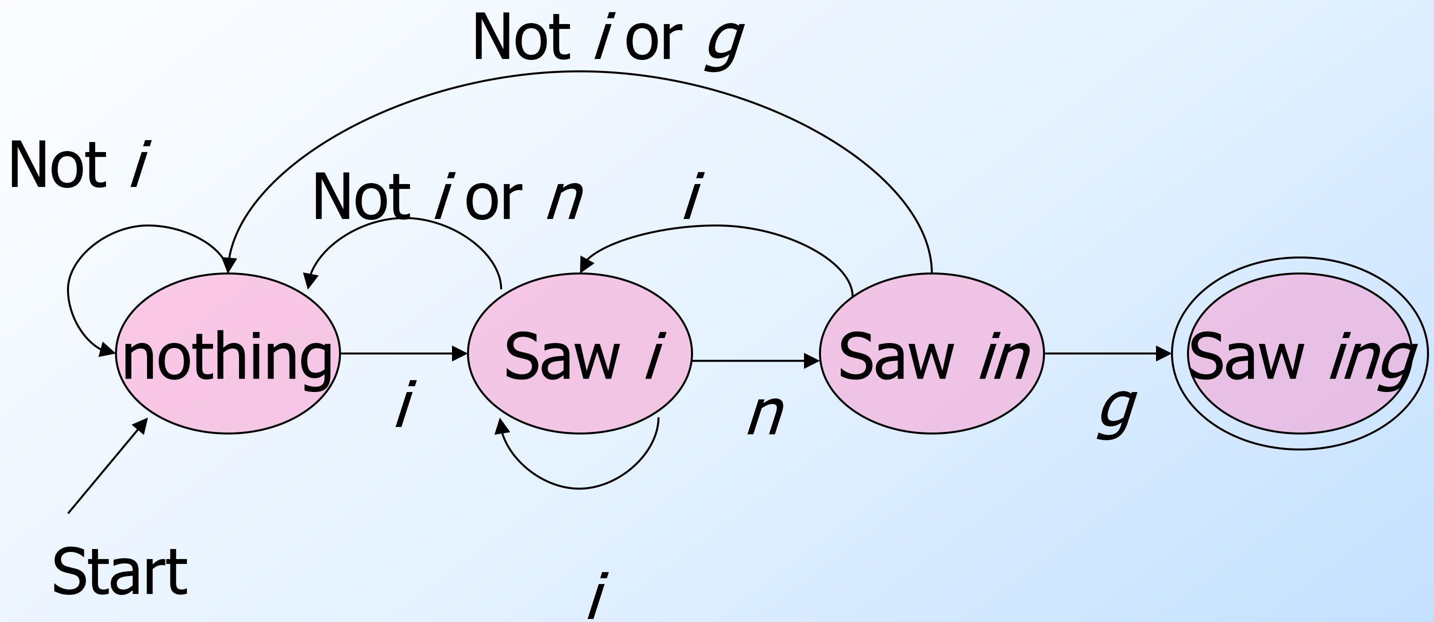
Informal Explanation

- Finite automata are finite collections of states (状态) with transition rules (转移规则) that take you from one state to another.
- Original application was sequential switching circuits, where the "state" was the settings of internal bits.
- Today, several kinds of software can be modeled by FA.

Representing FA (有限自动机的图示法)

- Simplest representation is often a graph.
  - Nodes(节点） = states.
  - Arcs (边) indicate state transitions.
  - Labels (标签) on arcs tell what causes the transition.

Example: Recognizing Strings Ending in “ing”



Automata to Code

- In C/C++, make a piece of code for each state. This code:
  1. Reads the next input.
  2. Decides on the next state.
  3. Jumps to the beginning of the code for that state.

Example: Automata to Code

```
2: /* i seen */
  c = getNextInput();
  if (c == ’n’) goto 3;
  else if (c == ’i’) goto 2;
  else goto 1;
3: /* ”in” seen */
  . . .
```

Automata to Code – Thoughts

- How would you do this in Python, which formally has no goto?
- You don't really write code like this. Rather, a code generator takes a "regular expression"（正则表达式） describing the pattern(s) you are looking for.
  - Example: `.*ing` works in grep （检索形如*ing的字符串）.
- 自动机与代码之间的关系是什么?
  - 为什么不用程序做计算模型?

Example in Java: without using goto

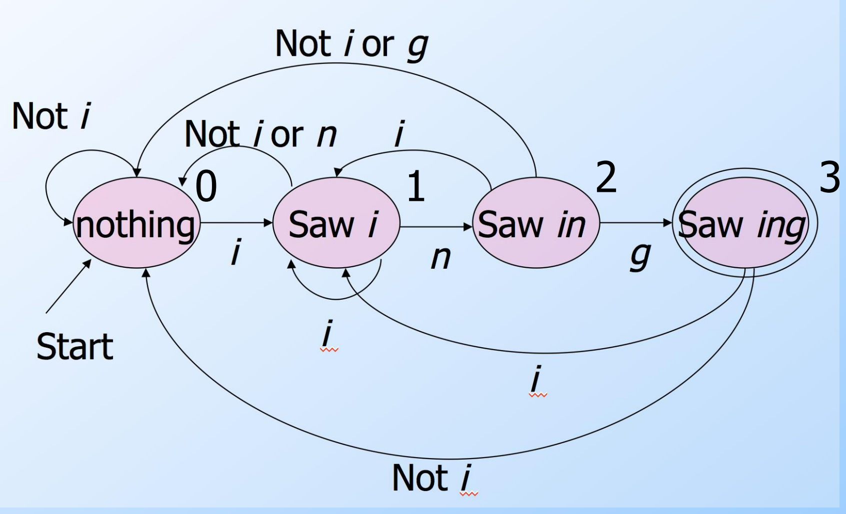
```
Scanner scan = new Scanner(System.in);
String s = scan.next();
int q = 0;
for (char c : s.toCharArray()) {
  switch (q) {
    case 0: q = (c=='i')? 1 : 0; break;
    case 1: q = (c=='n')? 2 : ((c=='i')? 1 : 0); break;
    case 2: q = (c=='g')? 3 : ((c=='i')? 1 : 0); break;
    case 3: q = (c=='i')? 1 : 0;
    }
}
if (q==3)
  System.out.println("accept.");
else
  System.out.println("reject.");
```

Start state

Loop through string s

Transitions

Final state



Formal Introduction to Finite Automata

Languages

Deterministic Finite Automata

(DFA, 确定性有限自动机)

Representations of Automata

"Finite Automata and Their Decision Problems"

- By Scott Dana and Rabin Michael (1959).
- Doctoral advisor of Both: Alonzo Church
  - 两位都是图灵的师弟
- Turing Award (1976)
- Automata theory is closely related to formal language theory.
  - In this context, automata are used as finite representations of formal languages that may be infinite.

Alphabets (字母表)

- An *alphabet* is any finite set of symbols.
  - A symbol is some unit of information representation
- Examples:
  - ASCII, Unicode, {0,1} (*binary alphabet*), {a,b,c}.

Strings(字符串)

- The set of *strings* over an alphabet Σ is the set of lists, each element of which is a member of Σ.
  - Strings shown with no commas, e.g., abc.
- Σ* denotes this set of strings.
- ε stands for the *empty string*
  - (string of length 0，空串).

Example: Strings

- $\{0,1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$
- Subtlety: 0 as a string, 0 as a symbol look the same.
  - Context（上下文） determines the type.

Languages （语言）

- A *language* is a subset of Σ^* for some alphabet Σ. (定义在某个字母表上的一组字符串)
- Example: The set of strings of 0's and 1's with no two consecutive 1's.
  - L = {ε, 0, 1, 00, 01, 10, 000, 001, 010, 100, 101, 0000, 0001, 0010, 0100, 0101, 1000, 1001, 1010, . . . }

Hmm… 1 of length 0, 2 of length 1, 3, of length 2, 5 of length 3, 8 of length 4. I wonder how many of length 5?

Deterministic Finite Automata
(确定性有限自动机)

- A formalism for defining languages, consisting of:
  1. A finite set of *states* (Q, typically，状态集).
  2. An *input alphabet* (Σ, typically, 输入字母表).
  3. A *transition function* (δ, typically, 转移函数).
  4. A *start state* (q_0, in Q, typically, 起始状态).
  5. A set of *final states* (F ⊆ Q, typically, 终止状态).
     - "Final" and "accepting（接受）" are synonyms. (终止与接受是同义词)

The Transition Function
（转移函数，面向一个字符定义）

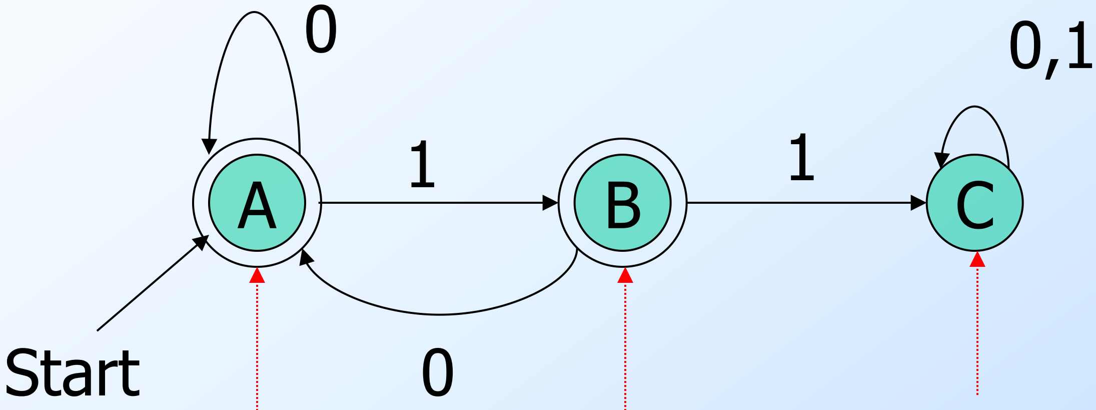
- Takes two arguments:
  - a state and an input symbol.
- $\delta(q, a)$ = the state that the DFA goes to when it is in state *q* and input *a* is received.
  - 当前处于状态*q*
  - 当前输入字符*a*
  - *下一个状态等于*$\delta(q, a)$

Graph Representation of DFA's (DFA的图表示)

- Nodes = states.
- Arcs represent transition function.
  - Arc from state p to state q labeled by all those input symbols that have transitions from p to q.
- Arrow labeled "Start" to the start state.
- Final states indicated by double circles.

Example: Graph of a DFA

Accepts all strings without two consecutive 1's.
(接受不含两个连续1的所有0/1串，如"0", "01","1","101")



Previous string OK, does not end in 1.

Previous String OK, ends in a single 1.

Consecutive 1's have been seen.

Alternative Representation: Transition Table （转移表）

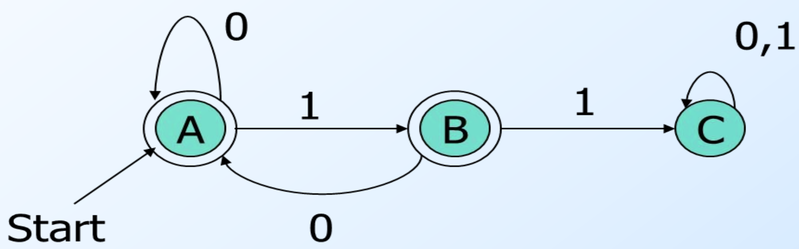
Final states starred

Columns = input symbols

	0	1
→ * A	A	B
* B	A	C
C	C	C

Arrow for start state

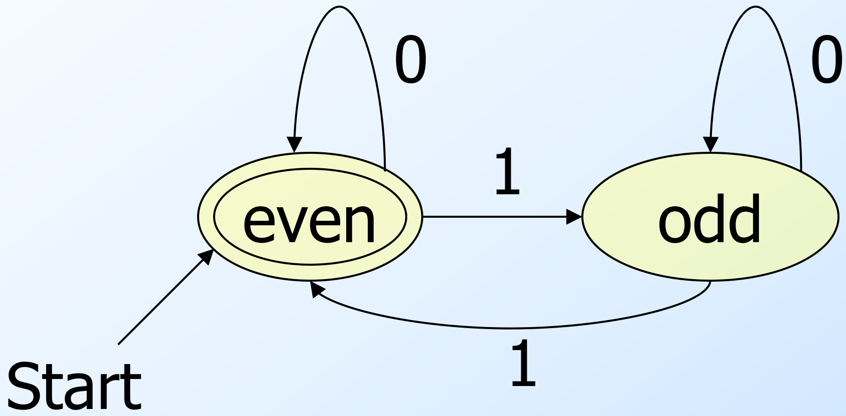
Rows = states



Python piece for coding the DFA

```
{
    'Q': {'A', 'B', 'C'},
     'SIGMA': {'0', '1'},
    'DELTA':
    {
                    ('A', '0'): 'A',
                    ('A', '1'): 'B'
                    ('B', '0'): 'A',
                    ('B', '1'): 'C'
                    ('C', '0'): 'C',
                    ('C', '1'): 'C'
          }
    'q0': 'A'
    'F': {'A', 'B'}
}
```

More Examples: (1)
An Even Number of 1's

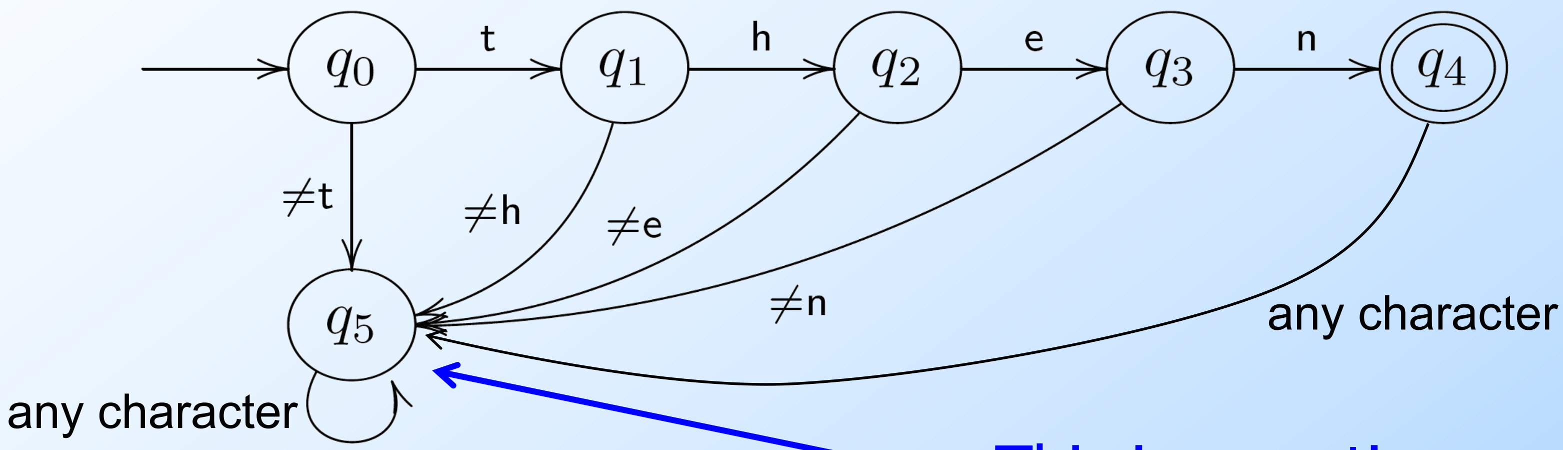


注意:

这个自动机是有限的，而包含偶数个1的字符串的个数是无限的

More Examples: (2)
Password/Keyword Example

It reads the word and accepts it if it stops in an accepting state

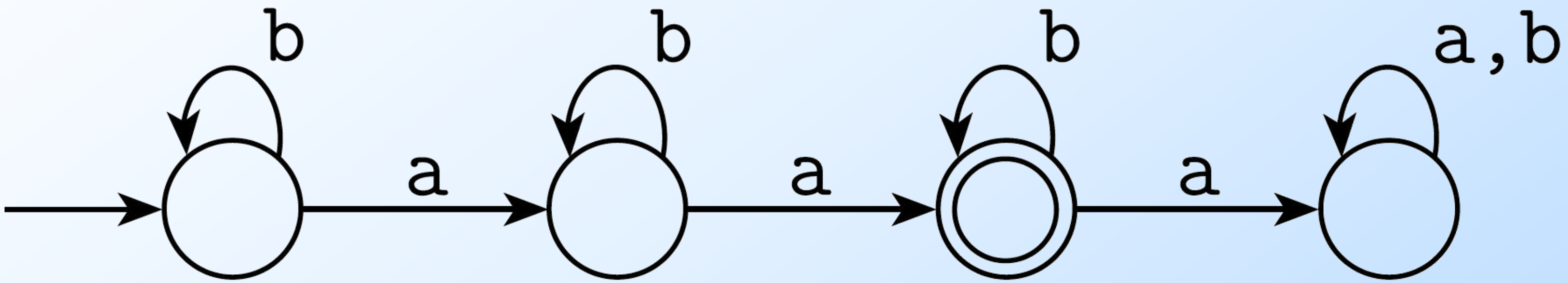


Only the word then is accepted

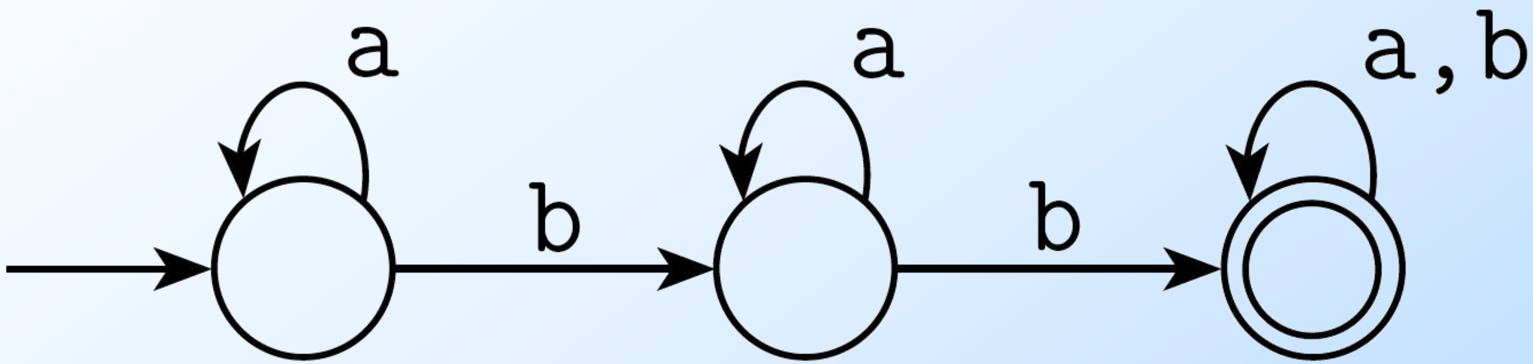
This is sometimes called a **dead** state.

BTW, there is a potential security risk on the password application if this finite automaton reports failure too quickly.

More Examples: (3)
Exactly Two a’s

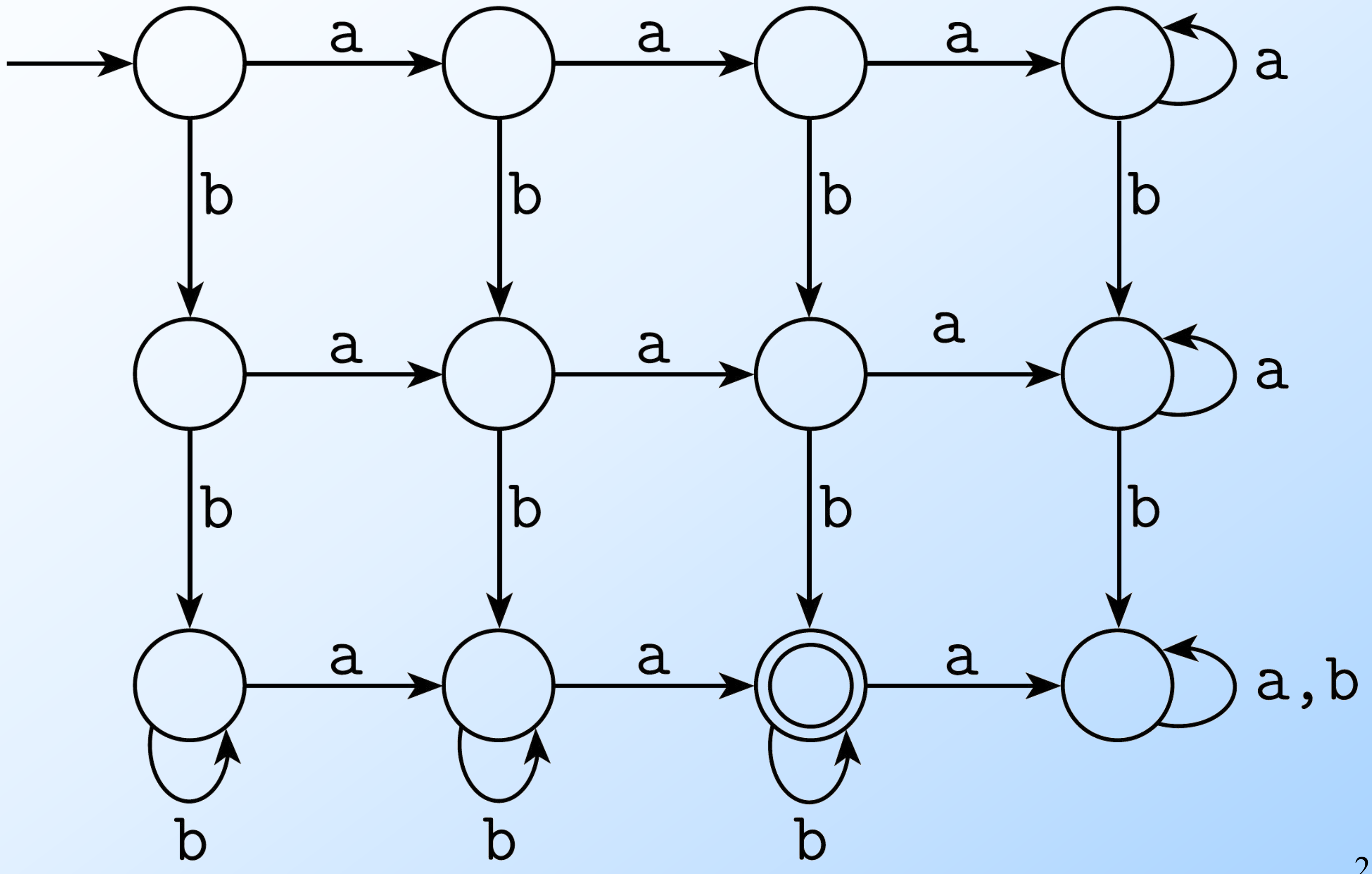


More Examples: (4)
At Least Two b's



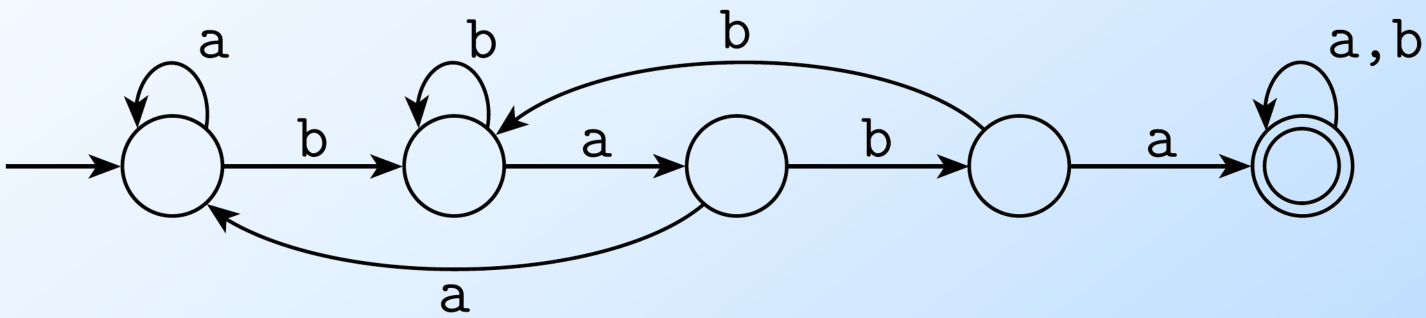
More Examples: (5)

Exactly two a’s and at least two b’s

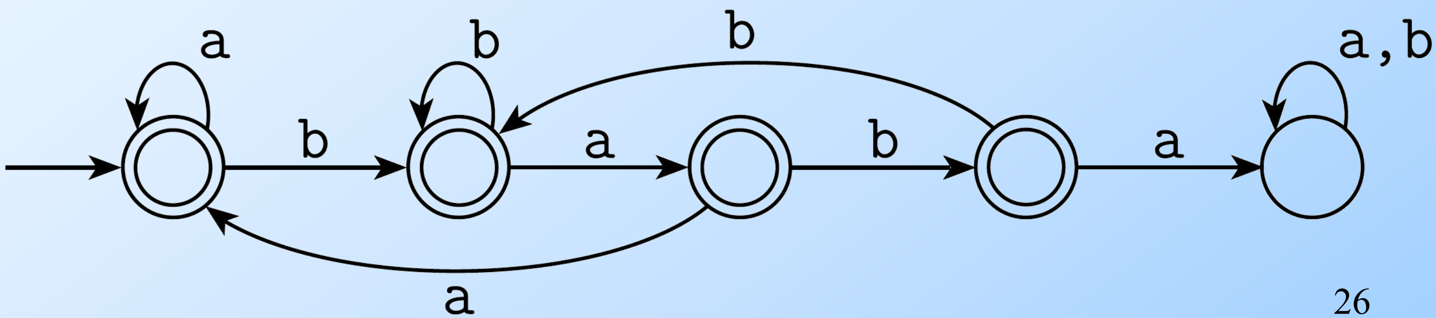


More Examples: (6)
Containing Substrings or Not

- Contains "baba":



- Does not contain "baba":



General Comments

- Some things are easy with finite automata:
  - Substrings (…abcabc…)
  - Subsequences (…a…b…c…b…a…)
  - Modular counting (odd number of 1's)
- Some things are impossible with finite automata (we will prove this later):
  - An equal number of a's and b's
  - More 0's than 1's
- But when they **can** be used, they are fast.

Extended Transition Function

(扩展转移函数，面向一个字符串定义：转移函数迭代使用的最终效果)

- We describe the effect of a string of input symbols on a DFA by extending δ to a state and a string.
  - 自动机从左至右逐个读输入串中的字符，并且相应地依次发生转移，直到整个输入串都读完。
- Induction on length of string.
- Basis: $\delta(q, \epsilon) = q$ （空串ε代表没有读入任何字符）
- Induction: $\delta(q,wa) = \delta(\delta(q,w),a)$
  - w is a string; a is an input symbol.

Extended δ: Intuition

- Convention:
  - … w, x, y, x are strings.
  - a, b, c,… are single symbols.

- Extended δ is computed for state q and inputs $a_1a_2…a_n$ by following a path in the transition graph, starting at q and selecting the arcs with labels a_1, a_2,…,a_n in turn.

Example: Extended Delta

	0	1
A	A	B
B	A	C
C	C	C

$\delta(B,011) = \delta(\delta(B,01),1) = \delta(\delta(\delta(B,0),1),1) =$

$\delta(\delta(A,1),1) = \delta(B,1) = C$

Delta-hat (带帽子的delta)

- In book, the extended δ has a "hat" to distinguish it from δ itself.
- Not needed, because both agree when the string is a single symbol.
- $\hat{\delta}(q, a) = \delta(\hat{\delta}(q, \epsilon), a) = \delta(q, a)$

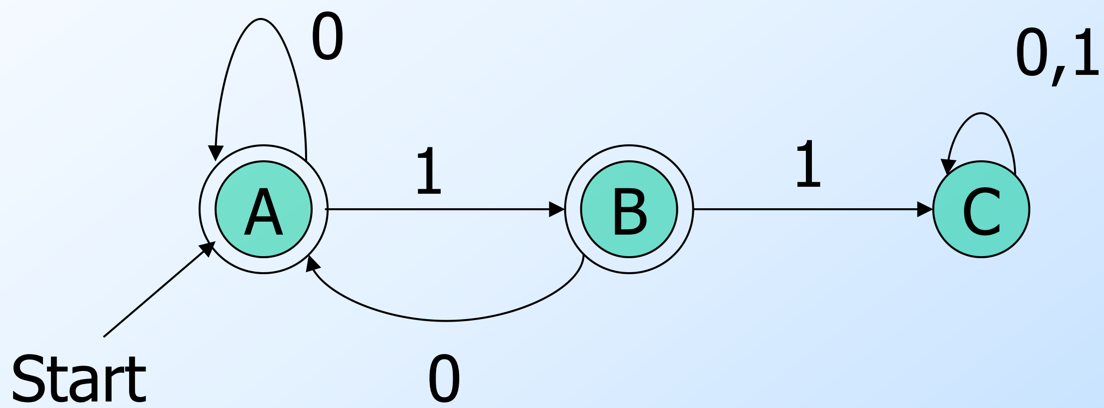
Extended deltas

Language of a DFA

- Automata of all kinds define languages.
- If A is an automaton, L(A) is its language.
- For a DFA A, L(A) is the set of strings labeling paths from the start state to a final state.
- Formally: L(A) = the set of strings w such that $\delta(q_0, w)$ is in F.

Example: String in a Language

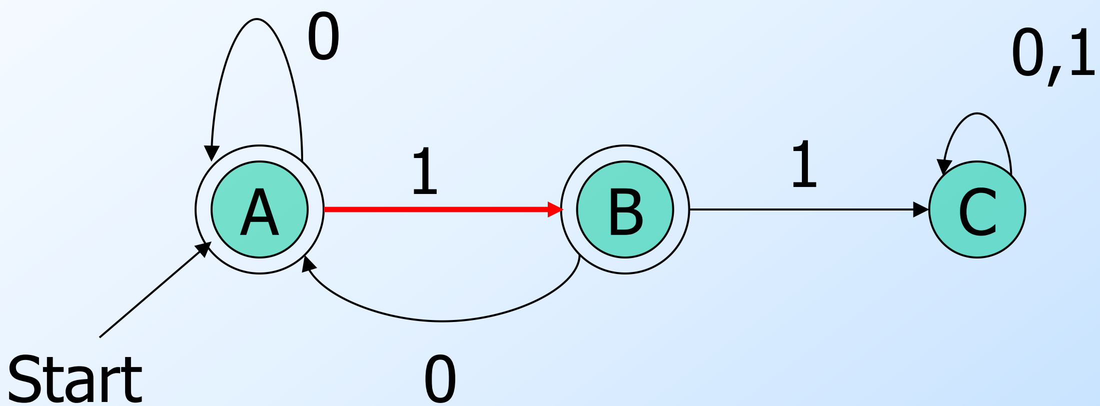
String 101 is in the language of the DFA below. Start at A.



Example: String in a Language

String 101 is in the language of the DFA below.

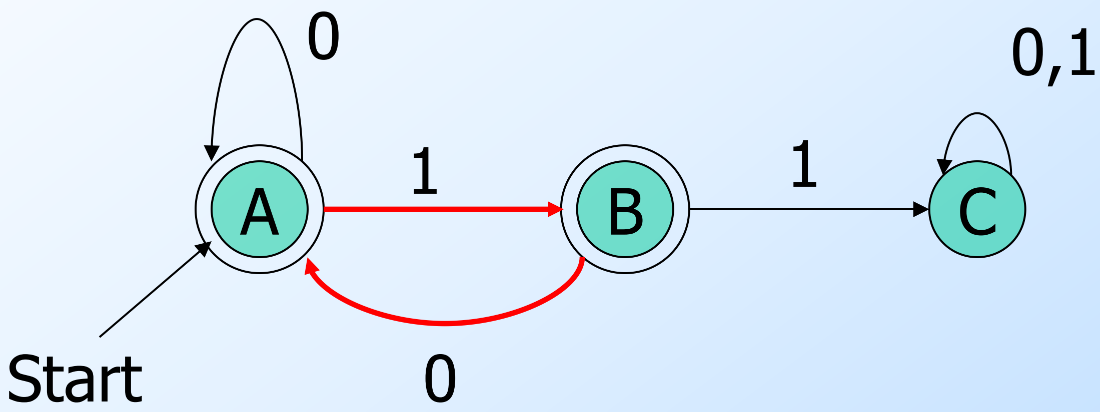
Follow arc labeled 1.



Example: String in a Language

String 101 is in the language of the DFA below.

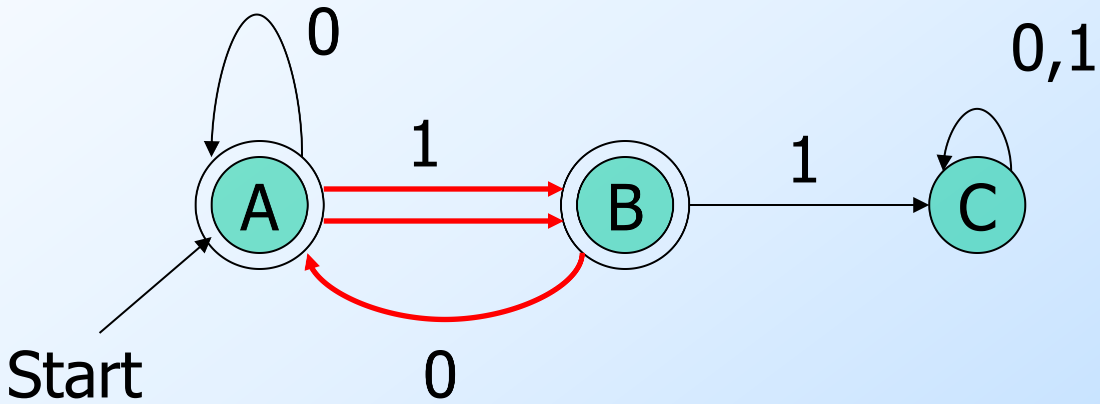
Then arc labeled 0 from current state B.



Example: String in a Language

String 101 is in the language of the DFA below.

Finally arc labeled 1 from current state A. Result is an accepting state, so 101 is in the language.



Example – Concluded

- The language of our example DFA is:

{w | w is in {0,1}* and w does not have two consecutive 1's}

Such that…

These conditions about w are true.

Read a *set former* as "The set of strings w…

Proofs of Set Equivalence

(集合等价，例如包含同一组字符串)

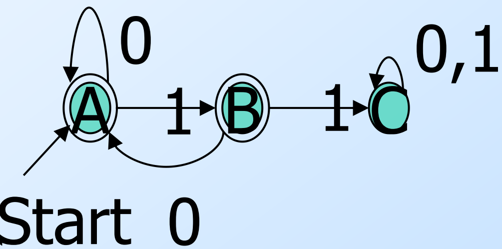
- Often, we need to prove that two descriptions of sets are in fact the same set.
  - 两种不同描述实际上描述的是同一集合
- Here,
  - one set is "the language of this DFA,"
  - and the other is "the set of strings of 0's and 1's with no consecutive 1's."

Proofs – (2)

- In general, to prove S=T, we need to prove two parts: $S \subseteq T$ and $T \subseteq S$. That is:
  1. If w is in S, then w is in T.
  2. If w is in T, then w is in S.
- As an example, let S = the language of our running DFA, and T = "no consecutive 1's."
  - S基于自动机描述，T基于自然语言描述

Part 1: $S \subseteq T$

(如果w作为一个输入串被自动机接受，那么w必定不包含两个连续的1)



- To prove: if w is accepted by then w has no consecutive 1's.
- Proof is an induction on length of w.
- Important trick: Expand the inductive hypothesis to be more detailed than you need.
  - 证明技巧是展开归纳假设，直到有足够多的细节可用

The Inductive Hypothesis (简称IH，归纳假设)

IH (1): If $\delta(A, w) = A$, then w has no consecutive 1's and does **not end in 1**.

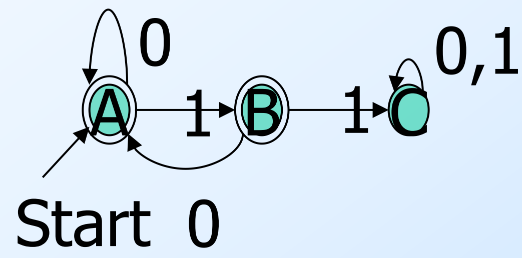
IH (2): If $\delta(A, w) = B$, then w has no consecutive 1's and ends in **a single 1**.

- Basis: $|w| = 0$; i.e., $w = \varepsilon$ （空串情形）.
  - IH (1) holds since $\delta(A, \varepsilon) = A$ and ε has no 1's at all.
  - IH (2) holds *vacuously*, since $\delta(A, \varepsilon) = A$ is not B.

"length of"

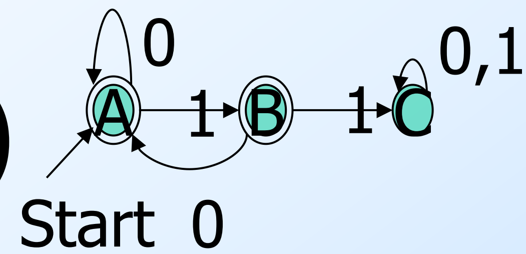
Important concept:
If the "if" part of "if..then" is false, the statement is true.

Inductive Step (归纳步)



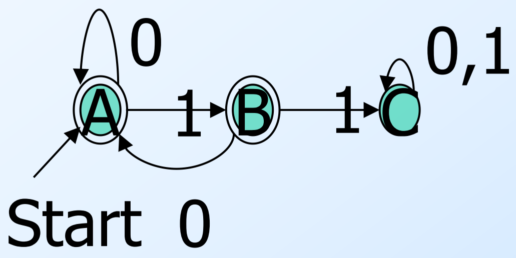
- Assume IH (1) and (2) are true for strings shorter than w, where |w| is at least 1.
- Because w is not empty, we can write w = xa, where *a* is the last symbol of w, and x is the string that precedes.
- IH （归纳假设） is true for x.

Inductive Step – (2)



- Need to prove IH (1) and IH (2) for w = xa.
- IH (1) for w is: If δ(A, w) = A, then w has no consecutive 1's and does not end in 1.
- Since δ(A, w) = A, δ(A, x) must be A or B, and *a* must be 0 (参见右上角的自动机).
- By the IH, x has no 11's.
- Thus, w has no 11's and does not end in 1.

Inductive Step – (3)



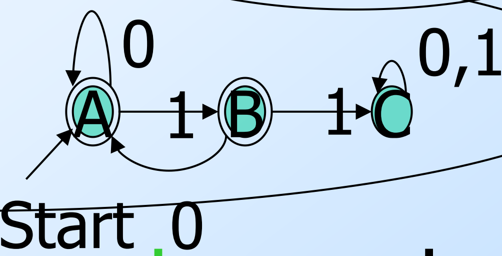
- Now, prove IH (2) for w = xa: If δ(A, w) = B, then w has no 11's and ends in a single 1.
- Since δ(A, w) = B, δ(A, x) must be A, and *a* must be 1 (参见右上角的自动机).
- By the IH, x has no 11's and does not end in 1.
- Thus, w has no 11's and ends in 1.

Part 2: T ⊆ S

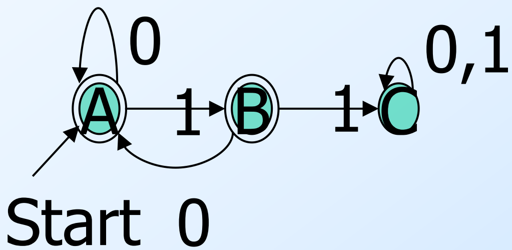
X

- Now, we must prove: if w has no 11's, then w is accepted by

Y



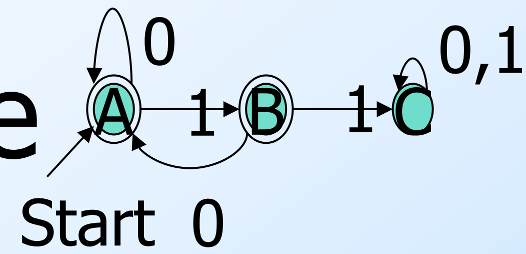
- *Contrapositive* : If w is not accepted by



then w has 11.

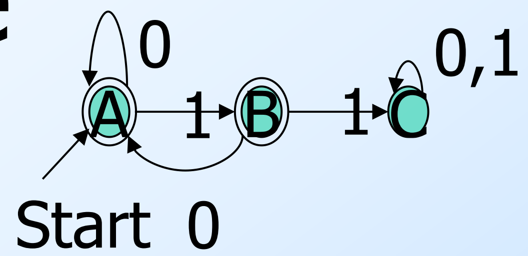
Key idea: contrapositive of "if X then Y" is the equivalent statement "if not Y then not X."

Using the Contrapositive



- Every w gets the DFA to exactly one state.
  - Simple inductive proof based on:
    - Every state has exactly one transition on 1, one transition on 0.
- The only way w is not accepted is if it gets to C.

Using the Contrapositive – (2)



- The only way to get to C [formally: $\delta(A,w) = C$] is if $w = x1y$, x gets to B, and y is the tail of w that follows what gets to C for the first time.
- If $\delta(A,x) = B$ then surely $x = z1$ for some z.
- Thus, $w = z11y$ and has 11.

Regular Languages
(正则语言，通过自动机定义版)

- A language L is *regular* if it is the language accepted by some DFA.
  - Note: the DFA must accept only the strings in L, no others.
- Some languages are not regular.
  - Intuitively, regular languages "cannot count" to arbitrarily high integers.

Example: A Nonregular Language

$L_1 = \{0^n1^n \mid n \geqslant 1\}$

- Note: a^i is conventional for i a's.
  - Thus, $0^4 = 0000$, e.g.
- Read: "The set of strings consisting of n 0's followed by n 1's, such that n is at least 1.
- Thus, $L_1 = \{01, 0011, 000111, \ldots\}$

Another Example

$L_2 = \{w \mid w \text{ in } \{(,)\}^* \text{ and } w \text{ is } \textit{balanced}\}$

- Note: alphabet consists of the parenthesis symbols '(' and ')'.
- Balanced parens are those that can appear in an arithmetic expression.
  - E.g.: (), ()(), (()), (()()),…

But Many Languages are Regular

- Regular Languages can be described in many ways, e.g., regular expressions.
- They appear in many contexts and have many useful properties.
- Example: the strings that represent floating point numbers in your favorite language is a regular language.

Example: A Regular Language

L_3 = { w | w in {0,1}* and w, viewed as a binary integer is divisible by 23}

- The DFA:
  - 23 states, named 0, 1,…,22.
  - Correspond to the 23 remainders （余数） of an integer divided by 23.
  - Start and only final state is 0.

Transitions of the DFA for L_3

- If string w represents integer i, then assume $\delta(0, w) = i\%23$.
- Then w0 represents integer 2i, so we want $\delta(i\%23, 0) = (2i)\%23$.
- Similarly: w1 represents 2i+1, so we want $\delta(i\%23, 1) = (2i+1)\%23$.
- Example: $\delta(15,0) = 30\%23 = 7$; $\delta(11,1) = 23\%23 = 0$.

Key idea: design a DFA by figuring out what each state needs to remember about the past.

Another Example

L_4 = { w | w in {0,1}* and w, viewed as the reverse of a binary integer is divisible by 23}

- Example: 01110100 is in L_4, because its reverse, 00101110 is 46 in binary.

- **Hard to construct the DFA.**
- But theorem says the reverse of a regular language is also regular.
  - 利用正则语言的性质，前提是证明出性质