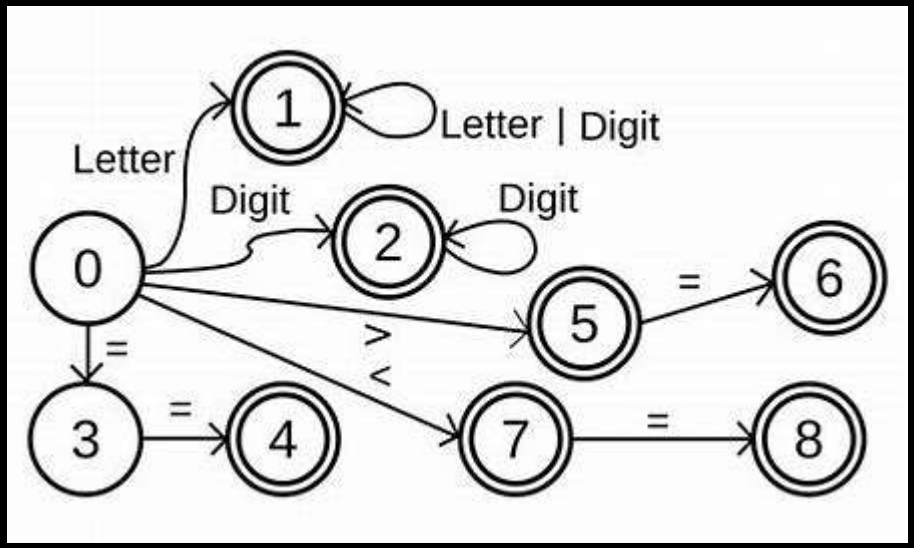


02
有穷自动机引论
Introduction to Finite Automata

入门例子

确定型有穷自动机

非确定型有穷自动机

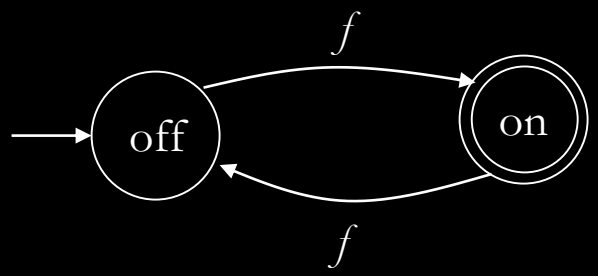
空转移有穷自动机

等价性证明

非形式化的解释

- 有限自动机 Finite Automata (FA)
- FA是状态的有限集合，这些状态具有从一种状态转移到另一种状态的转换规则。
- 最初的应用是顺序开关电路，其中“状态”是内部位的设置。
- 如今，FA可以对数种软件进行建模。

一个FA例子

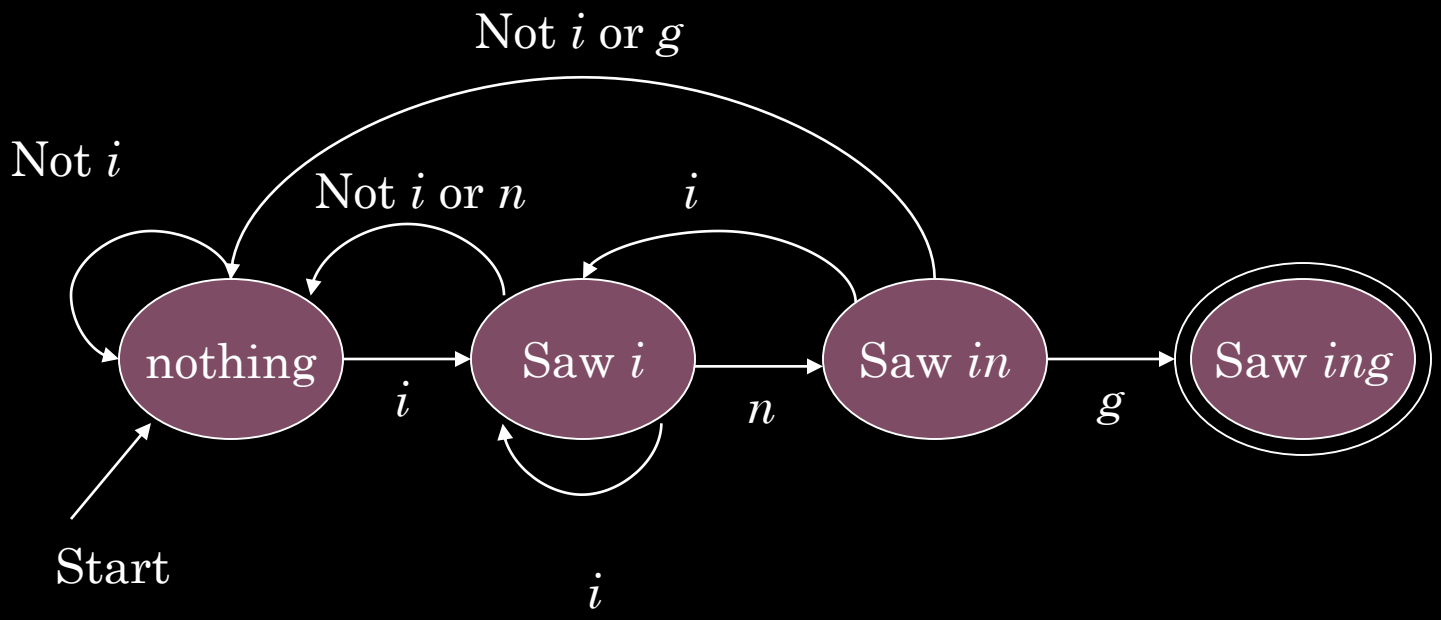


- 自动机可处于off和on状态，初始状态处于off状态，并尝试达到on这一“终结状态”，*f*是发生状态转移时接受的符号
- 怎样的*f*序列可达到终结状态?
- 答案: {*f*, *fff*, *fffff*, …} = {f^n: n is odd}
- 这是基于字母表{*f*}的确定型FA的示例

FA的表示

- 最简单的表示方法是图 graph表示
- 节点 Nodes = 状态 States.
- 弧Arcs 指示状态转移 transitions.
- 弧上的标签Labels 说明是什么符号 symbol导致了转移.

示例：识别以“ing”结尾的符号串



代码实现自动机

- 在C/C++，可为每一个状态写一小段代码：
  1. 读取下一个输入，
  2. 决定下一个状态，
  3. 跳转到下一状态的对应代码起始处。

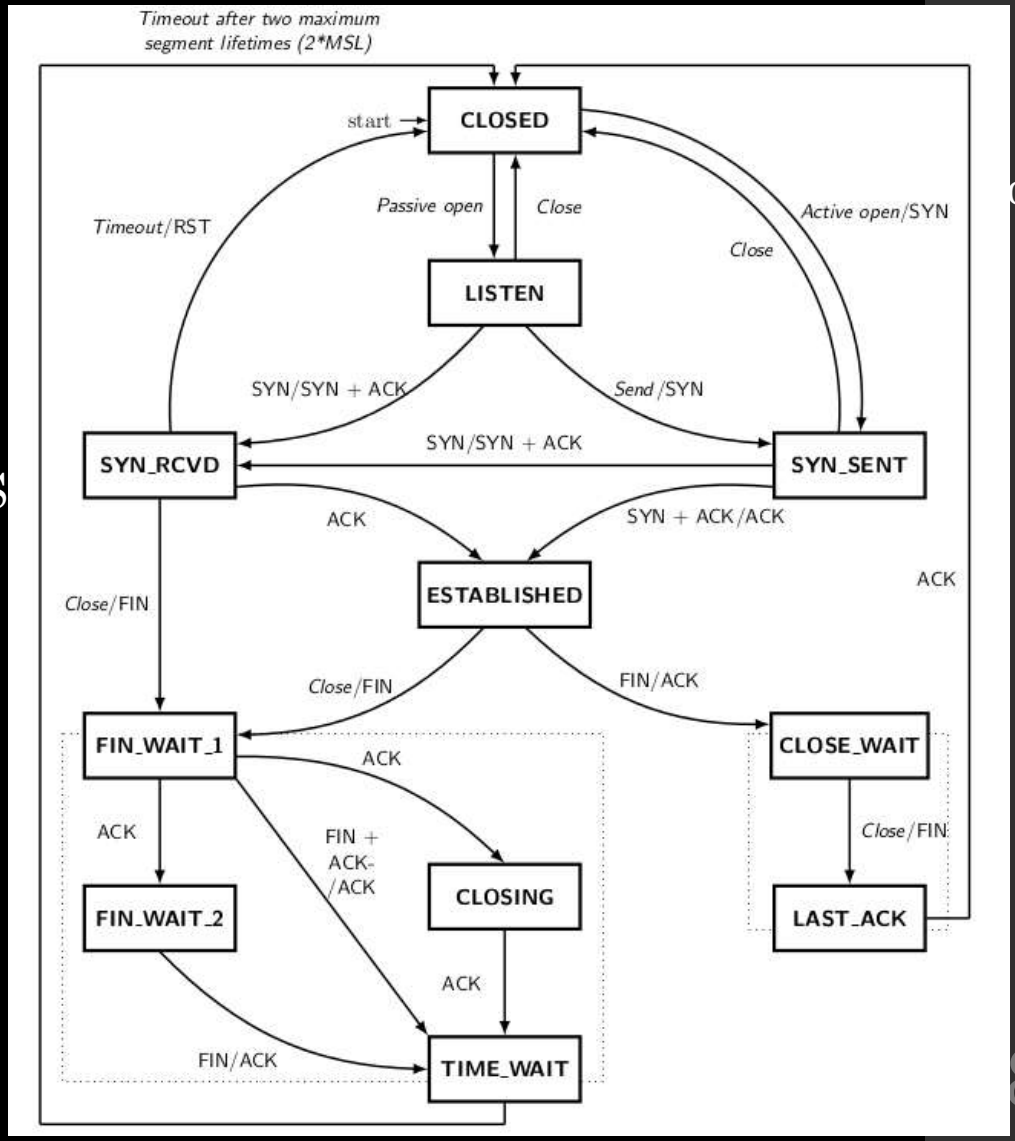
示例：代码实现自动机

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

自动机到代码的几点思考

- 在没有goto的Java中，将如何做到这一点？
- 你不愿也不需要写这些代码
- 更常见的做法是采用代码生成器接受“正则表达式”来描述要寻找的模式。
  - 例如，UNIX下可直接用grep命令查找 *ing
  - Word和现代编辑器都支持正则表达式搜索

示例：一个数据发送协议



这个协议是否正确？
它有可能失败吗？
能保证终止吗？

一个完整的例子——生命星球

- 在一个遥远的行星上有三个种群a、b、c。
- 任意两个不同种群的个体可交配繁殖，结果：
  1. 这两个个体死亡。
  2. 第三种群的两个新个体出生。

生命星球

- 注意：个体数量是不变的。
- 如果在某个时候所有个体都属于同一物种，那么星球将失败fail。
  - 因为这样就不能再繁殖了。
- 状态 = 整数三元组 – 种群a、b、c的个体数量

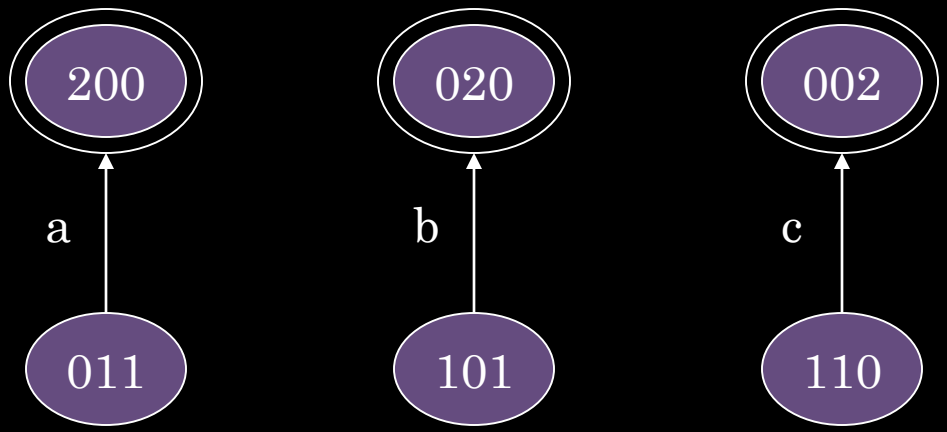
问题

- 在给定的状态下，行星是否有可能失败？比如选择了错误的繁殖方法。
- 在给定的状态下，星球最终一定会失败吗？
- 这些问题反映了有关协议的真实问题。
  - “星球可能失败吗？”就像在询问协议是否会进入某些不希望的状态或错误状态。
  - “星球必然失败吗？”就像在询问是否能保证协议终止。
    - 在这里，“失败”其实是我们想要的好结局。

定义转移

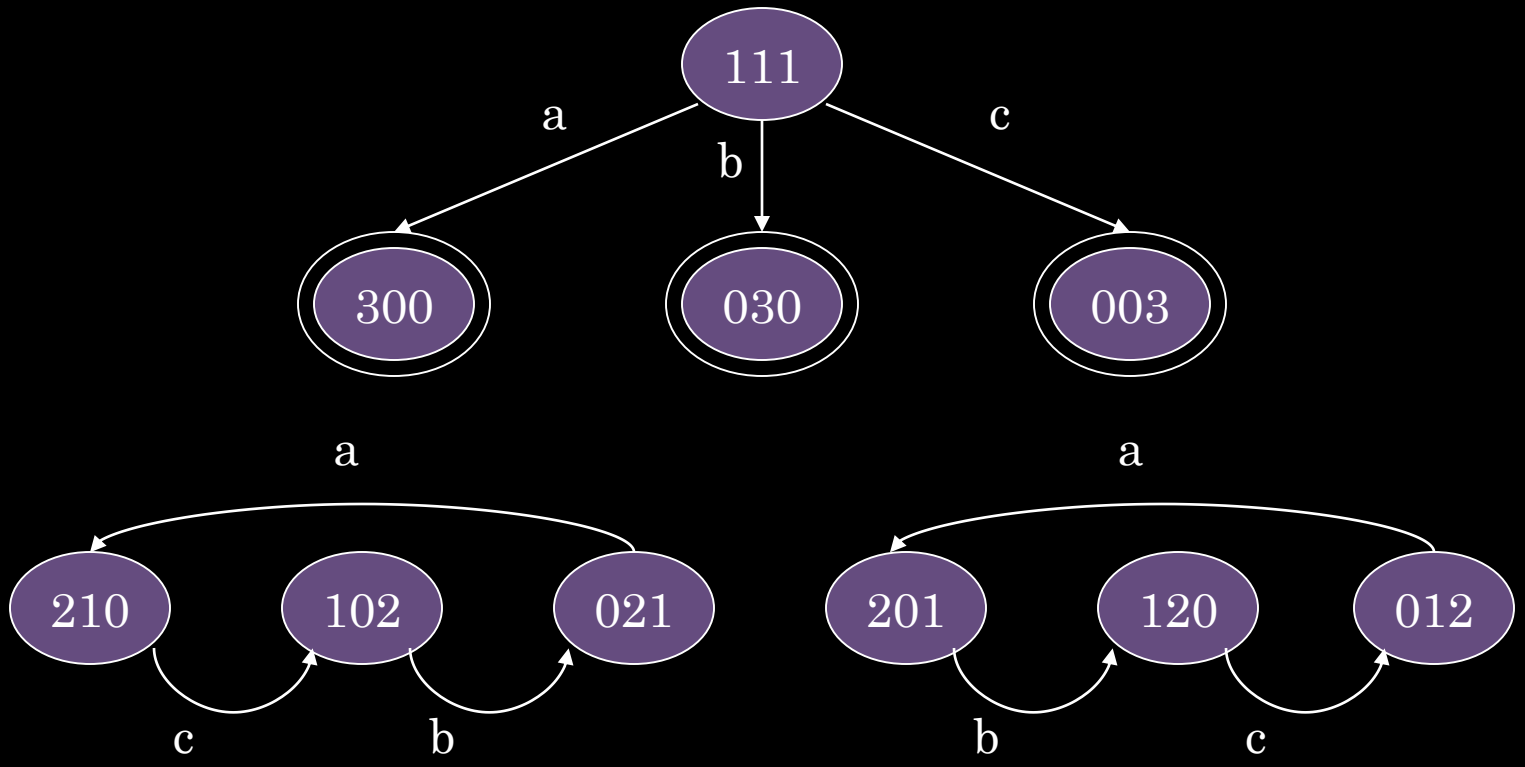
- 当分别来自种群b、c的两个个体繁殖并被替换为种群a的两个个体时，定义发生α-事件。
- 类似的可定义b-事件和 c-事件。
- 这三个事件分别用a、b、c三个符号表示。

有2个个体的生命星球



注意：所有状态都必然失败。

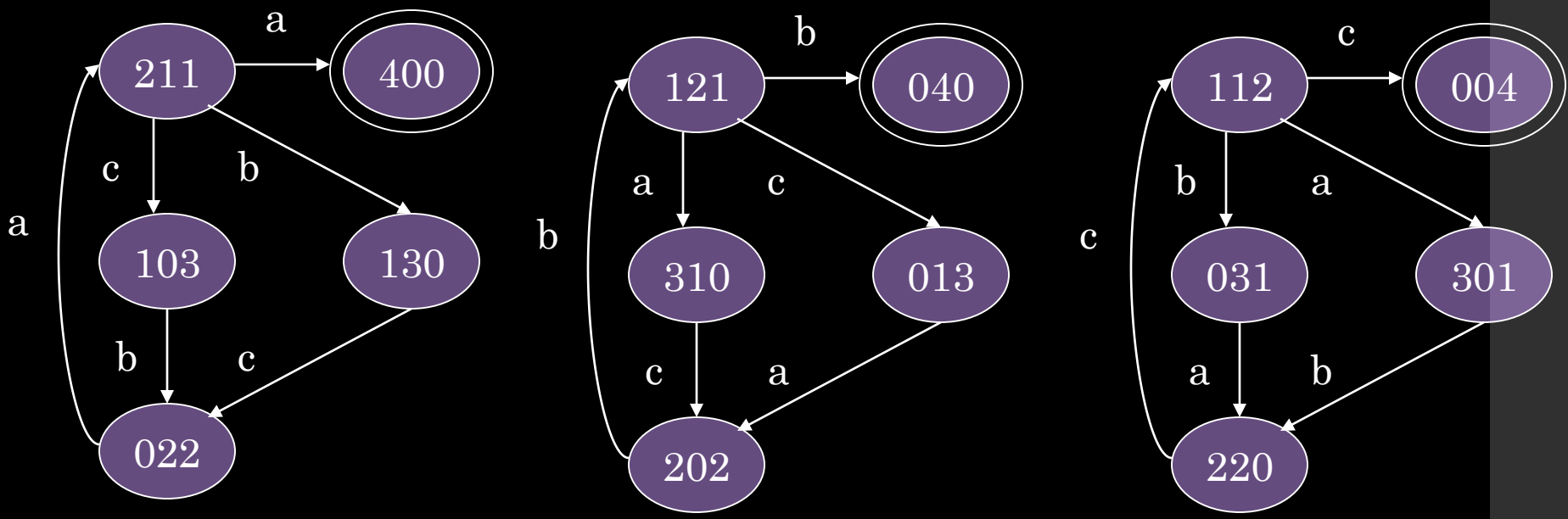
有3个个体的生命星球



注意：4个必然失败的状态，
其它都是不会失败的状态。

状态111有多个可能的转移。

有4个个体的生命星球

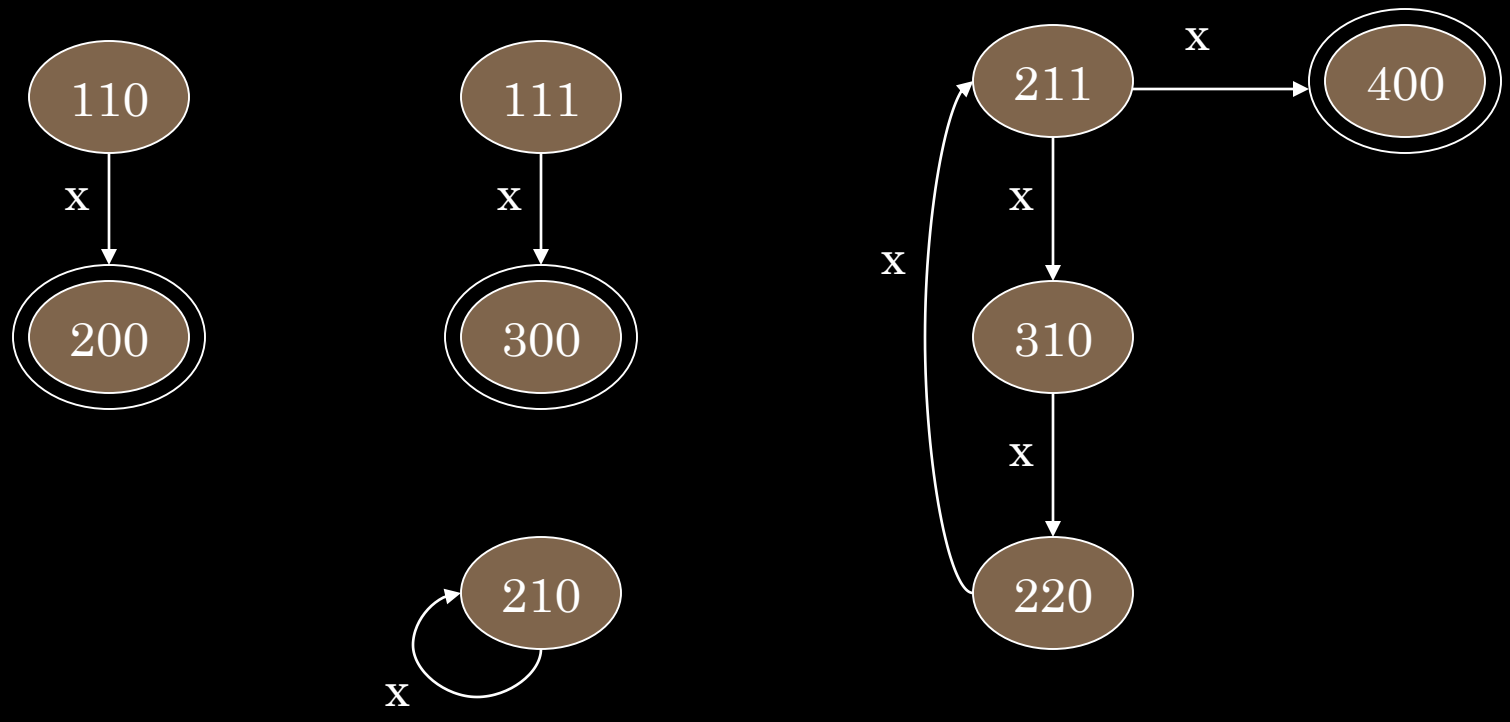


注意： 状态400、040、004必然失败。
其它状态都可能失败。

考虑对称性

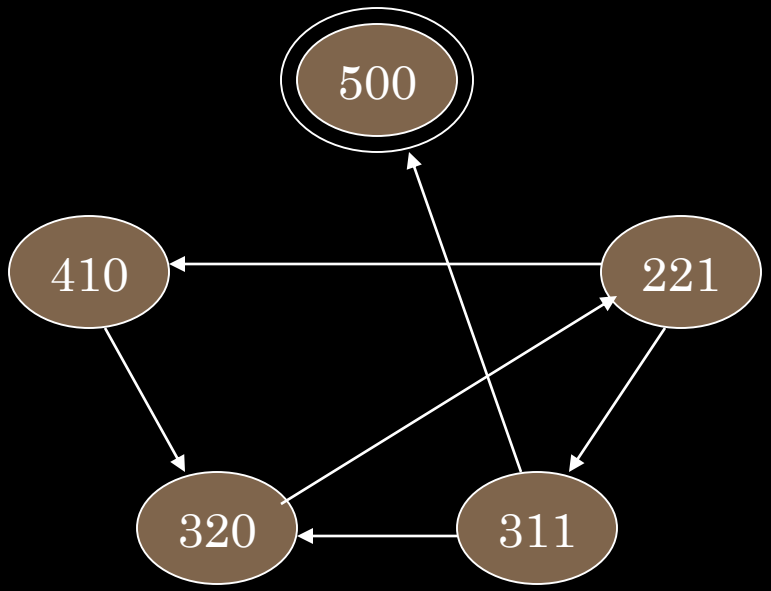
- 失败的可能性仅取决于三个种群的数量，而不取决于哪个物种具有哪个数量。
- 因此，用计数列表来表示状态，并按从大到小顺序排序。
- 这样仅需一个转移符号x。

情形2, 3, 4



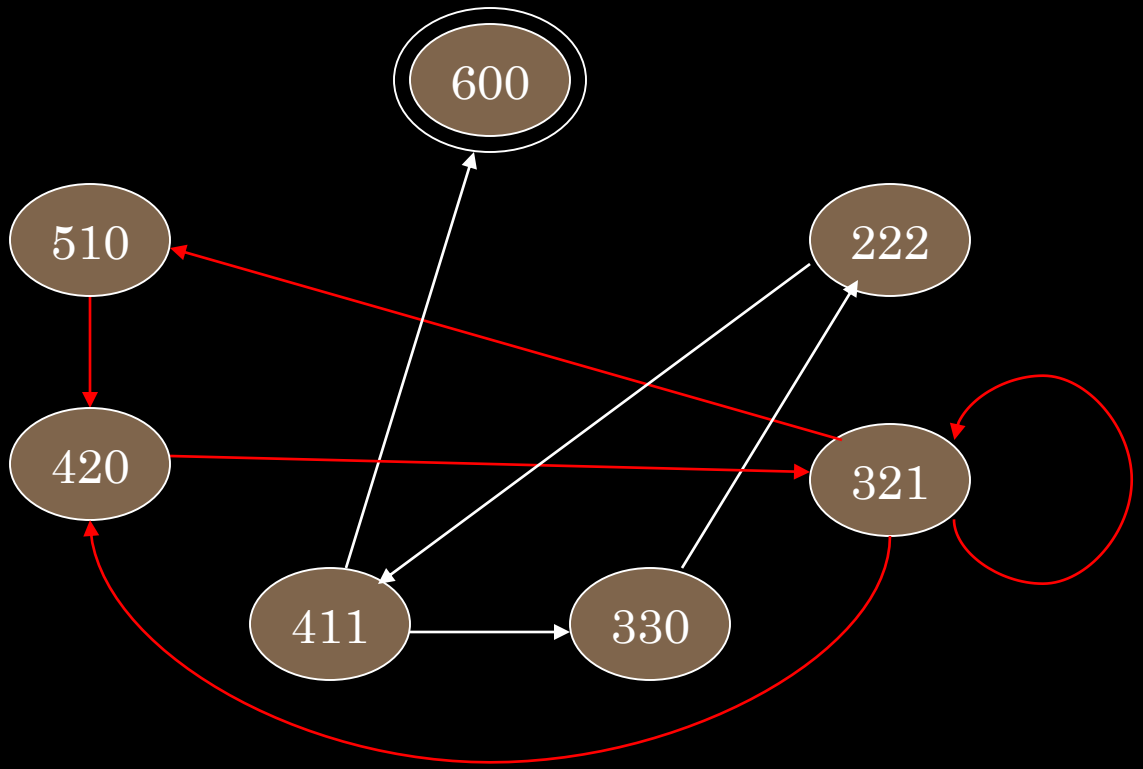
注意：当n = 4时，产生了不确定性*nondeterminism*，状态211在相同的输入下可能到达不同的下一状态。

5个体



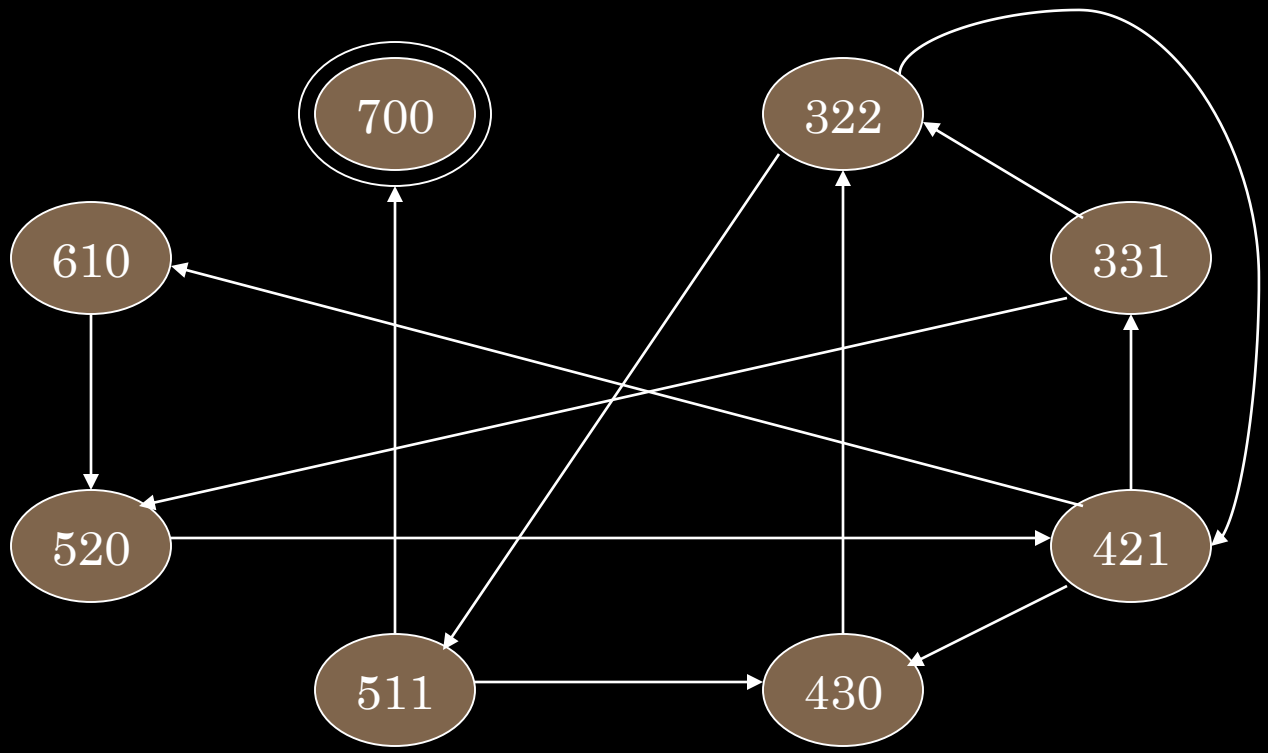
注意： 500必然失败，其它都可能失败。

6个体



注意： 600必然失败，510、420、321不可能失败，411、330、222可能失败。

7个体



Notice: 700必然失败，其它状态都可能失败。

思考题

（课后自行完成，不需要提交）

1. 不使用对称性，n个个体有多少个状态？
2. 如果使用对称性呢？
3. 对于n个个体，如何判断一个状态是“必然失败”，“可能失败”还是“不会失败”？

确定型有穷自动机

- 一个确定型有穷自动机，可形式化定义为一个五元组$\{Q, \Sigma, \delta, q_0, F\}$，包含：
  1. 状态：A finite set of *states* (Q, typically).
  2. 字母表：An *input alphabet* (Σ, typically).
  3. 转移函数：A *transition function* (δ, typically).
  4. 初始状态：A *start state* (q_0, in Q, typically).
  5. 终结状态：A set of *final states* ($F \subseteq Q$, typically).

     这里，终结 “Final”和接受 “accepting”是同义词

转移函数

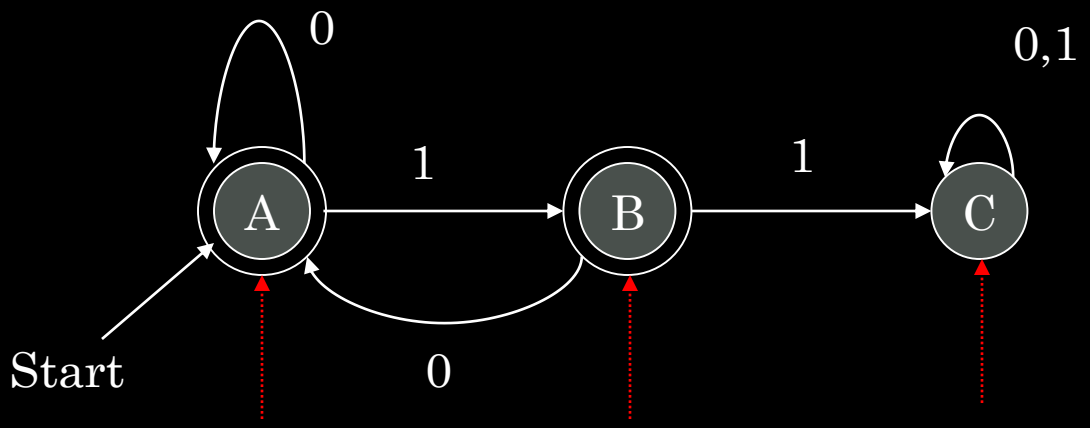
- 接受两个参数：当前状态和输入符号
- δ(q, a) = DFA 在状态q输入 a后到达的下一状态

DFA的图表示

- 节点 = 状态
- 弧表示转移函数
  - 从状态p到状态q的弧，由所有从p到q转移的输入符号组成标签。
- 起始状态由Start标签的箭头指示
- 终结状态用双圆圈表示

示例: 一个DFA图

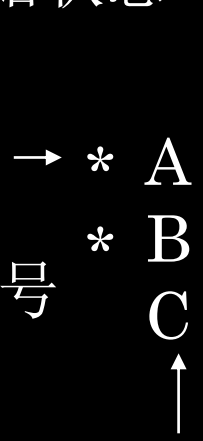
接受所有不带两个连续1的符号串。
Accepts all strings without two consecutive 1's.



另一种表示法：转移表
Transition Table

箭头指向起始状态
Arrow for
start state

终结状态加星号
Final states
starred

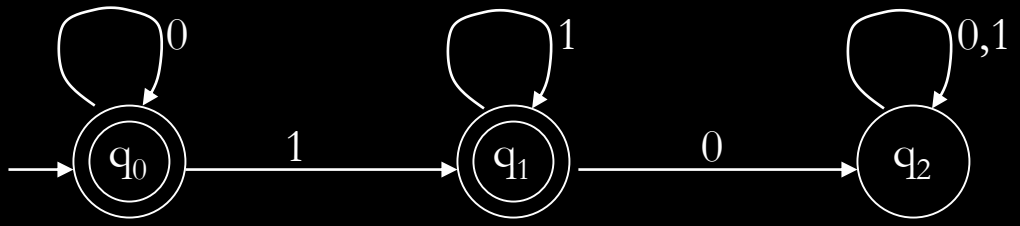


行=状态
Rows = states

0	1
A	B
A	C
C	C

列=输入符号
Columns =
input symbols

例子



alphabet $\Sigma = \{0, 1\}$
start state $Q = \{q_0, q_1, q_2\}$
initial state q_0
accepting states $F = \{q_0, q_1\}$

transition function δ:

states \ inputs	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_2	q_2

扩展转移函数
Extended Transition Function

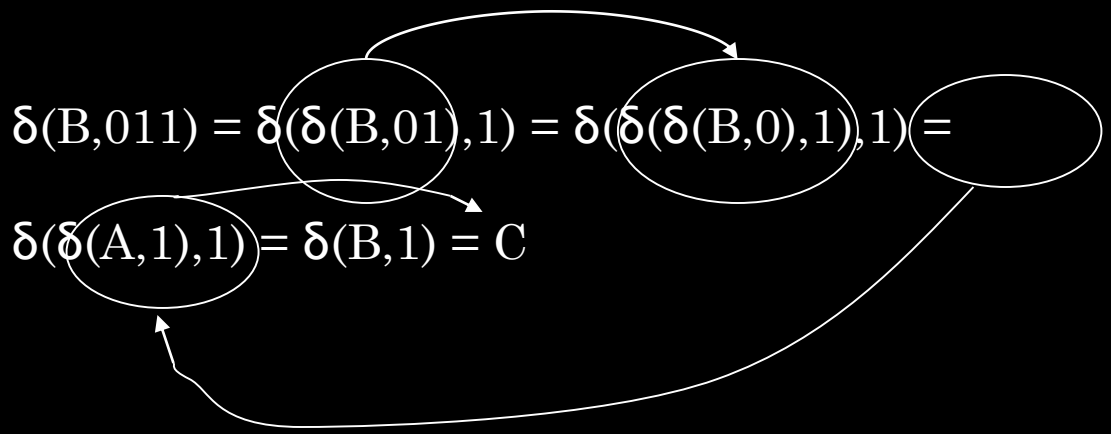
- 通过将δ扩展到状态和符号串来描述输入符号串对DFA的影响。
- 通过符号串长度进行归纳：
- Induction on length of string.
- Basis: $\delta(q, \epsilon) = q$
- Induction: $\delta(q,wa) = \delta(\delta(q,w),a)$
  - w is a string; a is an input symbol.

扩展δ: 直观解释

- 约定:
  - … w, x, y, x为符号串
  - a, b, c,…为单个符号
- 针对状态q和输入$a_1a_2…a_n$计算扩展δ，方法是遵循转移图中的路径，依次选择带有标签a_1, a_2,…,a_n的弧

示例: 扩展Delta函数

	0	1
A	A	B
B	A	C
C	C	C



对Delta-hat标记的说明

- 扩展δ在书中戴上了“帽子”以与δ本身区别开
- 事实上这并不必要，因为当符号串包含单个符号时，两者的结果是一致的
- $\hat{\delta}(q, a) = \delta(\hat{\delta}(q, \epsilon), a) = \delta(q, a)$

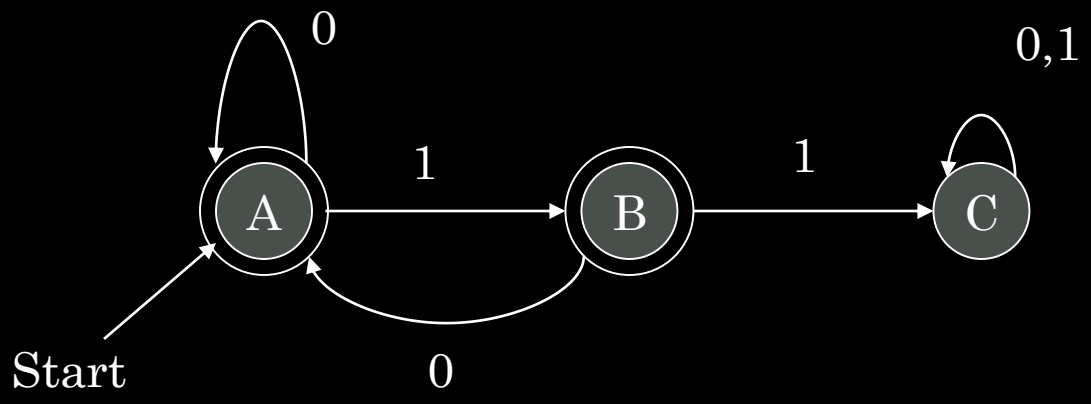
Extended deltas

DFA的语言

- 所有种类的自动机都定义了语言
- 如果A是自动机，则L(A)是它的语言
- 对于DFA A， L(A)是从起始状态到终结状态的路径上标记符号串的集合。
- 形式化: L(A) = 满足$\delta(q_0, w)$属于F的符号串w的集合

示例: 语言中的符号串

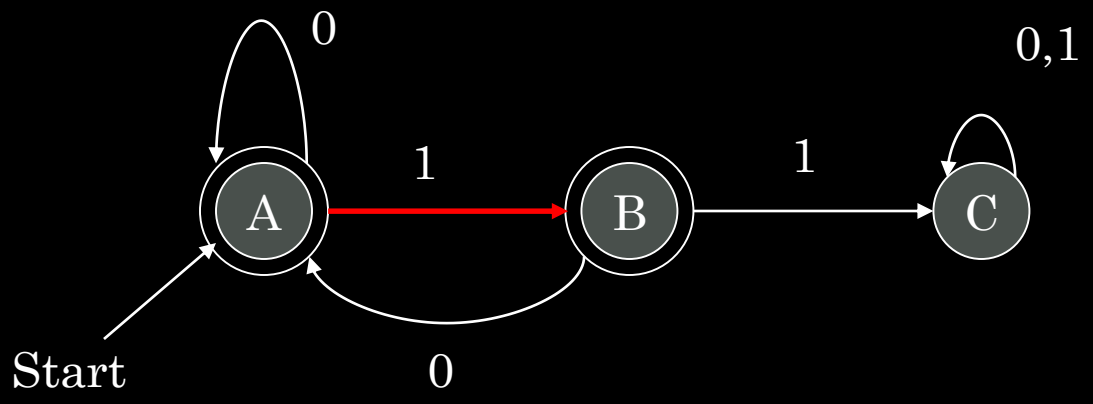
符号串101在以下DFA的语言中
从A开始



示例: 语言中的符号串

符号串101在以下DFA的语言中。

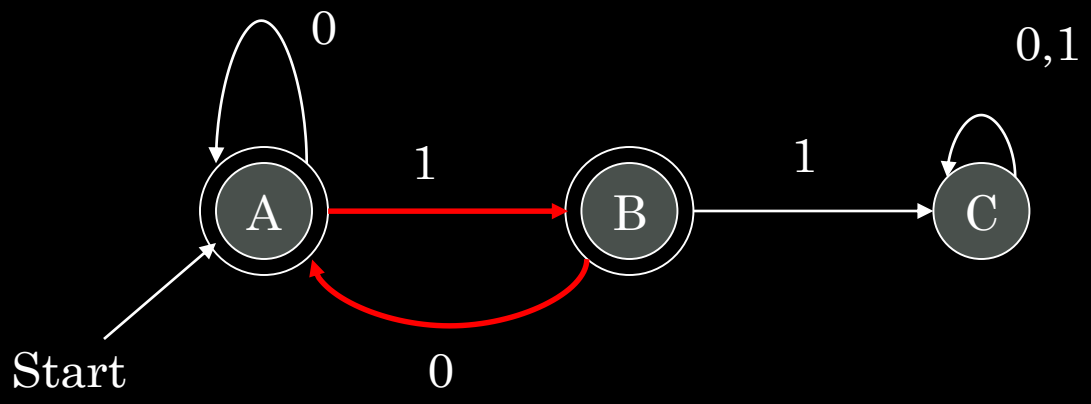
跟随标签为1的弧，到达B



示例: 语言中的符号串

符号串101在以下DFA的语言中。

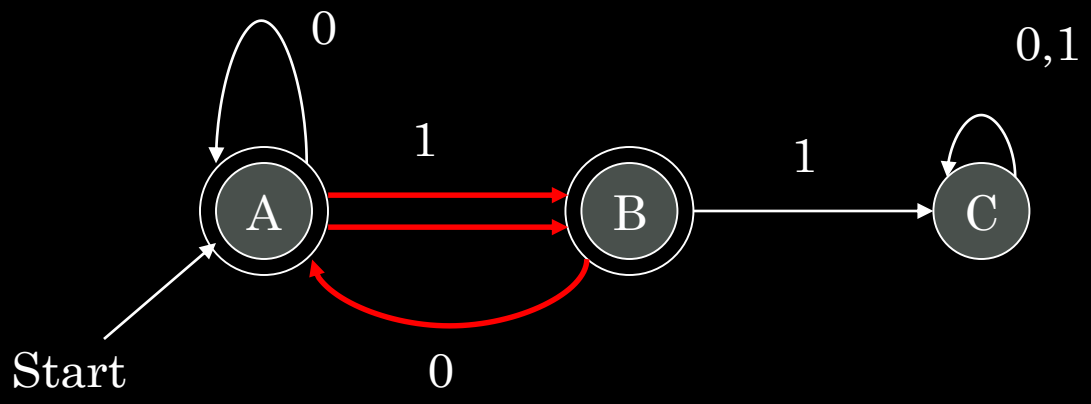
接着是从B出发的0弧，回到A



示例: 语言中的符号串

符号串101在以下DFA的语言中。

最后顺着从A出发的1弧又回到B
B是一个终结/接受状态，所以101在语言中

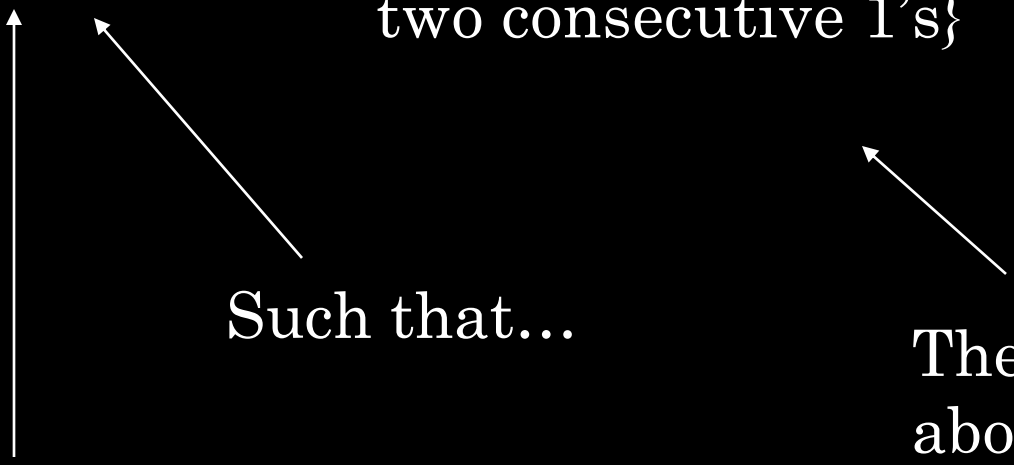


示例 – 结论

- 例子中DFA的语言为不包含连续两个1的01串

{w | w is in {0,1}* and w does not have

two consecutive 1's}



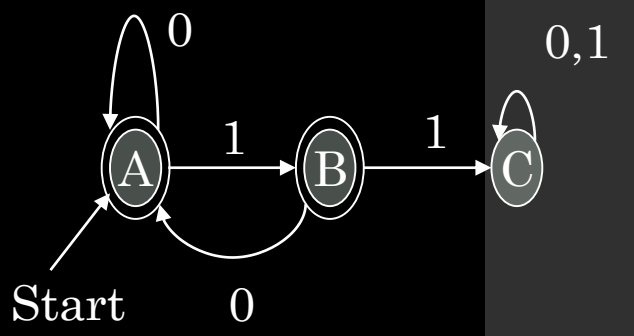
集合等价性证明
Set Equivalence

- 接下来，需要证明集合的两个描述实际上是同一集合。
- 这里，一个集合是“此DFA的语言”，另一个集合是“不包含连续两个1的01串”

证明 – (2)

- 一般来说，要证明S=T，需要分别证明两个部分：S ⊆ T和T ⊆ S，即对任意w：
  1. 如果w在S中，则w在T中
  2. 如果w在T中，则w在S中
- 在此例中，令
  - S =此DFA的语言
  - T = “不包含连续两个1的01串”

Part 1: S ⊆ T



- 要证: 如果w被接受，则w不含连续的1
- 证明是对w的长度的归纳
- 技巧: 把归纳假设（inductive hypothesis） 扩展到比你需要的更详细。

归纳假设
The Inductive Hypothesis

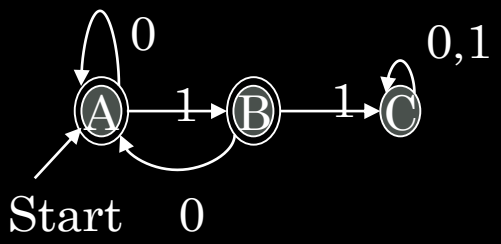
1. 如果δ(A, w) = A，则w不含连续的1且不以1结尾
2. 如果δ(A, w) = B，则w不含连续的1且以单个1结尾

- Basis: |w| = 0; i.e., w = ϵ.
  - (1) 满足，因为ϵ不含1
  - (2) 满足，因为δ(A, ϵ)不为B

“length of”

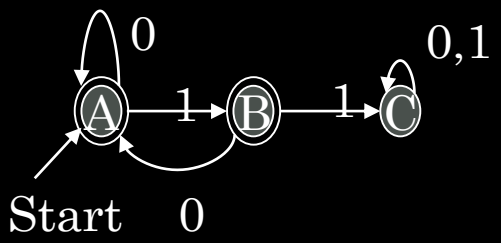
注意:
当“如果”的前提为假时，
整个句子是真的

归纳步骤
Inductive Step



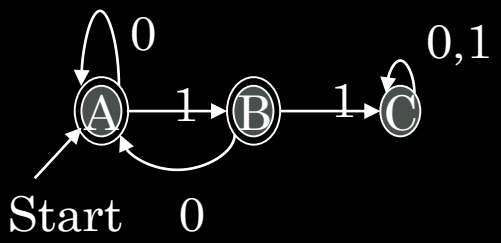
- 假设 (1)、(2) 对于所有比w短的符号串都成立，|w|至少为1
- 因为w非空，令w = xa，其中a为w结尾符号，x为之前符号串
- 对x来说，归纳假设为真

归纳步骤
Inductive Step – (2)

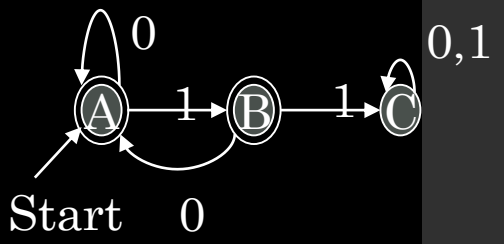


- 下面证明 (1) 、(2) 对w = xa也成立
- 对于(1)：如果 δ(A, w) = A，则w不含连续的1且不以1结尾。
- 因为δ(A, w) = A，δ(A, x)必须为A或B，且a为0 (见DFA图)
- 根据归纳假设，x不含11
- 因此，w也不含11且不以1结尾

Inductive Step – (3)



- 类似的，对于(2)：如果δ(A, w) = B，则w不含11且以1结尾
- 因为δ(A, w) = B，δ(A, x)必为A，且a为1 (由DFA图可知).
- 根据归纳假设，x不含11且不以1结尾
- 因此，w不含11但以1结尾



Part 2: T ⊆ S

X

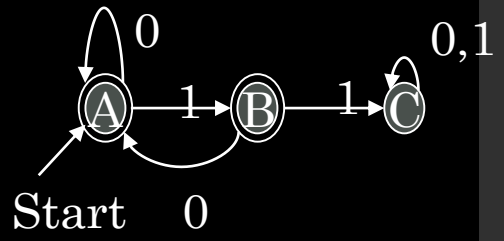
- 现在要证明：如果w不含11，则w可被DFA接受

Y

- *逆否命题*：如果w不被DFA接受，则w包含11

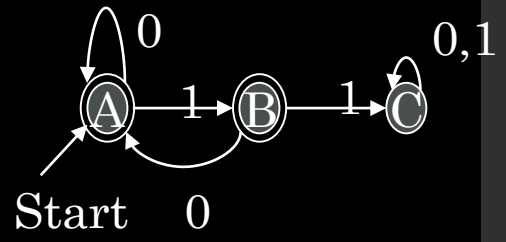
思路: “if X then Y”的逆否命题“if not Y then not X.”与原命题等价

利用逆否命题来证明



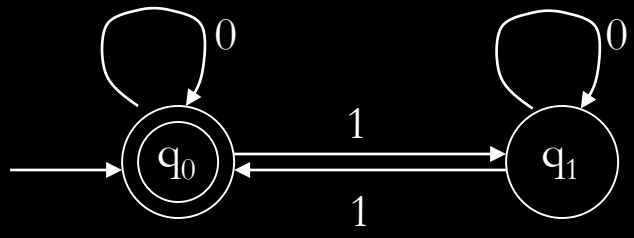
- 每个w都使得DFA停在一个唯一的状态
  - 可以简单通过归纳法证明
- 每个状态都只有1和0两个转换
- 只有停在C状态时，w不被接受

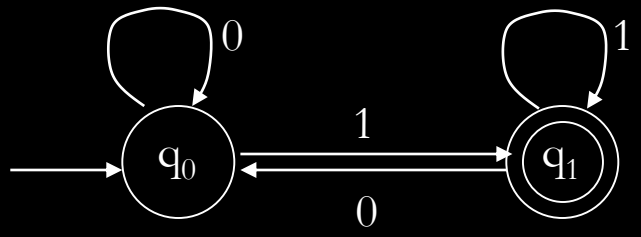
利用逆否命题来证明– (2)

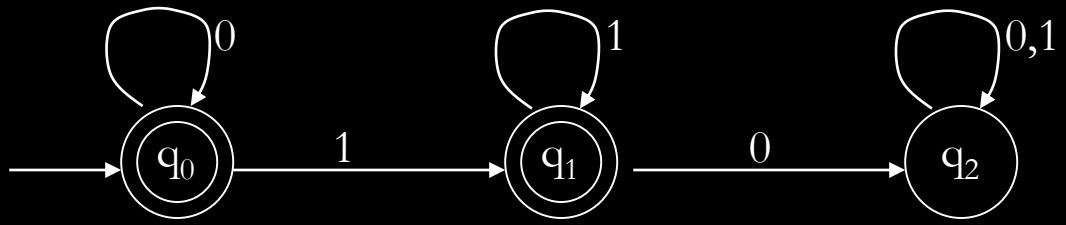


- 到达状态C [形式化：δ(A,w) = C]意味着w = x1y，x可达B，且y为w第一次到达C后的尾部
- 如果δ(A,x) = B，则显然有某个z使得x = z1
- 因此，w = z11y且包含11

示例：DFA到语言







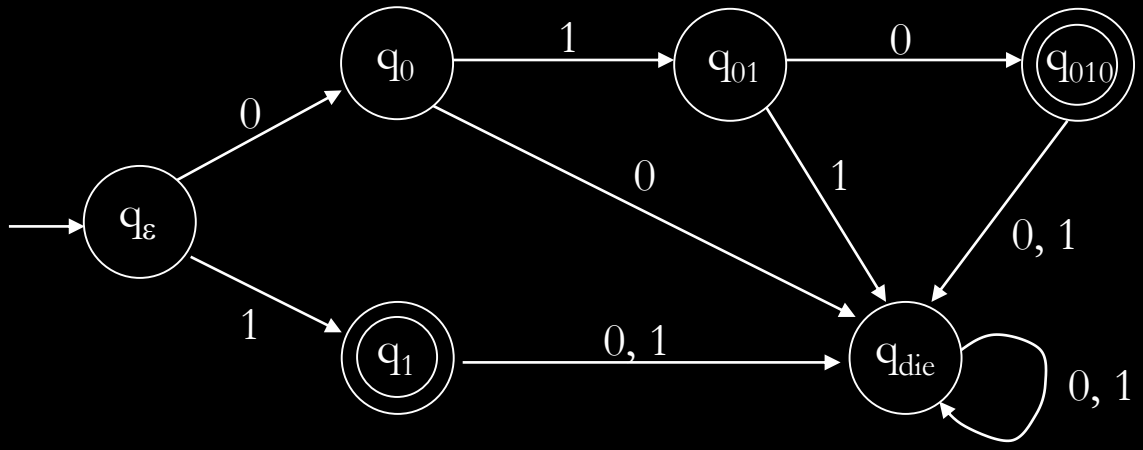
这些DFA接受的语言是？
（作业I中回答，不用证明）

示例：语言到DFA

- 为下面的语言构造（construct）一个DFA

$L = \{010, 1\}$ （$\Sigma = \{0, 1\}$）

- DFA



另一个例子

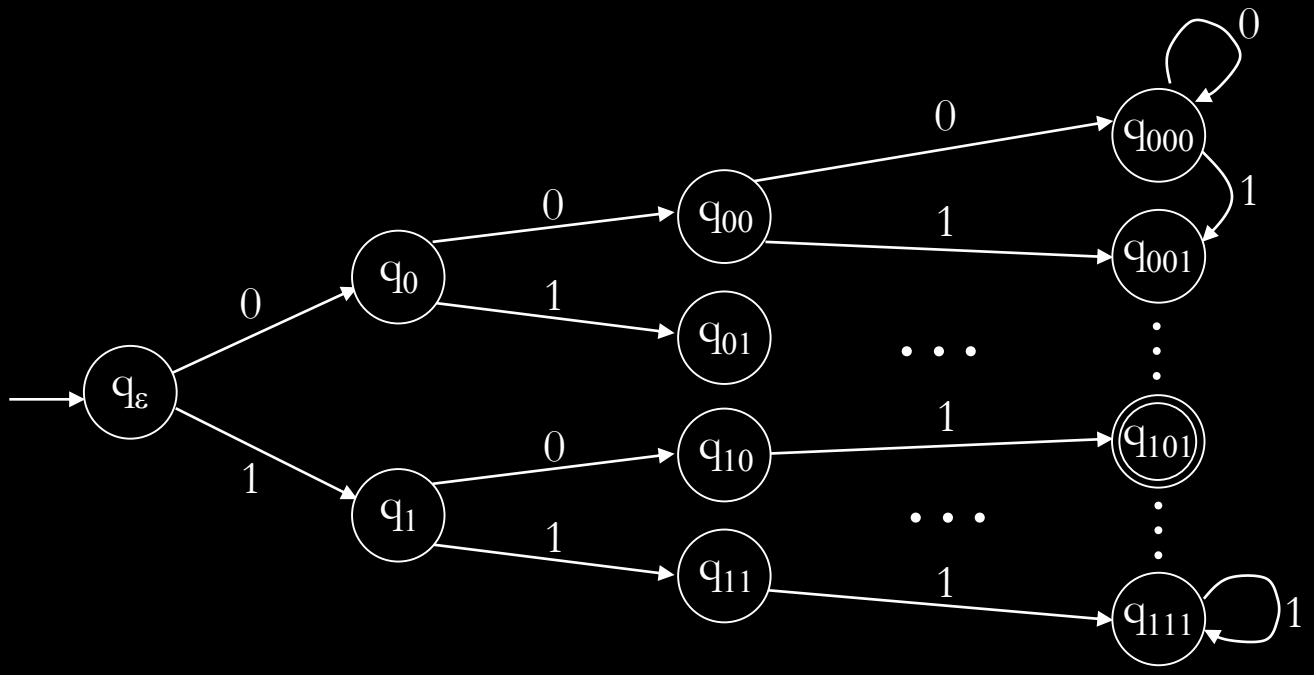
- 构造{0, 1}上的DFA接受所有以101结尾的符号串

- 提示: 此DFA可以“记得” 读到的最后3位符号

- 怎样记住3个符号？

答案

- 构造{0, 1}上的DFA接受所有以101结尾的符号串



思考一下，一共有多少个状态？可以少一些吗？

正则语言
Regular Languages

- 一个语言L能被某个DFA接受，则称它是正则的
  - 注意: 此DFA必须只接受L中的符号串，不接受任何其它符号串
- 有些语言不是正则的
  - 本质上，正则语言“无法数出”任意大的整数

示例：一个非正则语言

$L_1 = \{0^n 1^n \mid n \geq 1\}$

- 注意: a^i是对连续i个a的缩写
  - 例如，$0^4 = 0000$,
- 读作: “The set of strings consisting of n 0’s followed by n 1’s, such that n is at least 1.”
- 可知，$L_1 = \{01, 0011, 000111, \ldots\}$
- 为什么不是正则语言？

另一个示例

L_2 = {w | w in {(,)}* and w is *balanced* }

- 注意: 字母表包含括号符号(parenthesis symbols): '(' 和 ')'
- 匹配的括号(balanced parens)是算术表达式中的要求
  - 例如: (), (()), (()), (()()),...
- 为什么不是正则语言?

正则的语言也很多

- 正则语言可用多种形式描述，例如正则表达式
- 在很多不同上下文中有很多有用的性质
- 示例: 程序语言中表示浮点数的符号串就是正则语言

示例: 一个正则语言

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

- DFA:
  - 23个状态： 0, 1,…,22
  - 对应一个整数除以23的23种余数
  - 起始和唯一的终结状态都是0

L_3之DFA的转移函数

- 如果w代表整数i，则假设δ(0, w) = i%23.
- 可知w0代表2i，由此δ(i%23, 0) = (2i)%23.
- 类似w1代表2i+1，由此δ(i%23, 1) = (2i+1)%23.
- 示例: δ(15,0) = 30%23 = 7; δ(11,1) = 23%23 = 0.

思路: 设计一个DFA使得其中每一个状态都能记住过去的结果

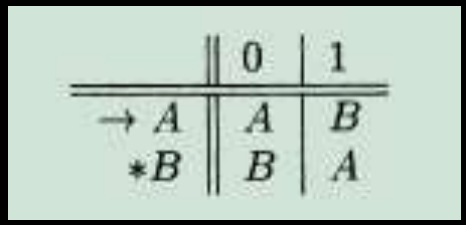
另一个例子

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

- 示例: 01110100属于L_4，因为其逆置，00101110为46的二进制表示
- 此DFA比较难以构造
- 但后面可以根据定理得出正则语言的逆置仍然是正则的（见第5单元，或课本第4章）

作业2

- 1. 说明下列DFA对应的语言并用归纳法证明结论。说明前文中几个DFA对应的语言。

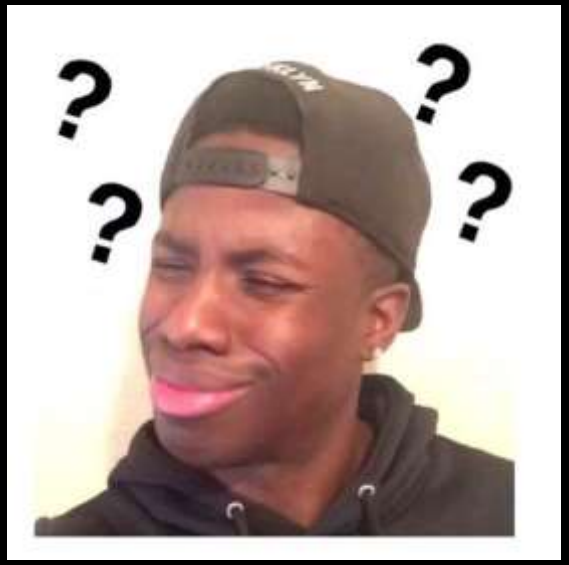


	0	1
→ A	A	B
*B	B	A

- 2. 构造下列语言的DFA
  - 以01开头的01串
  - 以01结尾的01串
  - 包含“01”的01串
  - 不包含“01”的01串
- 3. 思考题（不用提交）
  - 构造一个DFA识别包含“10110”的01串

什么是非确定？

- 非确定型有穷自动机 *nondeterministic finite automaton （NFA）* 具有在同一时刻处于多个状态的能力
- 从一个状态出发，通过一个输入符号可以转移到一个状态集合



什么是非确定？– (2)

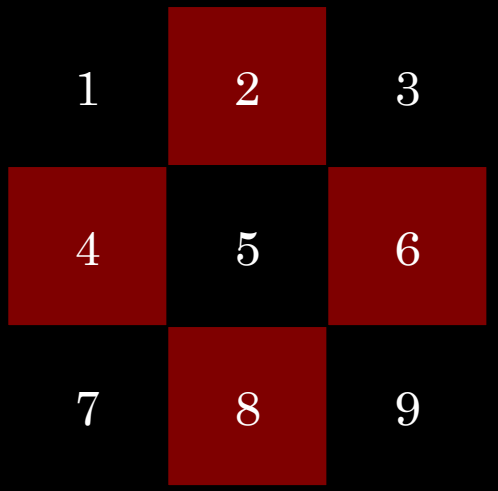
- 假设存在一个自动机，从一个状态出发，通过一个输入符号，可能存在多个可能的下一状态
- 每一步运行都必须做一次选择，选择其中一个作为下一状态
- NFA会遍历所有可能的选择，当有n种选择时，可以认为会复制为n个

什么是非确定？– (3)

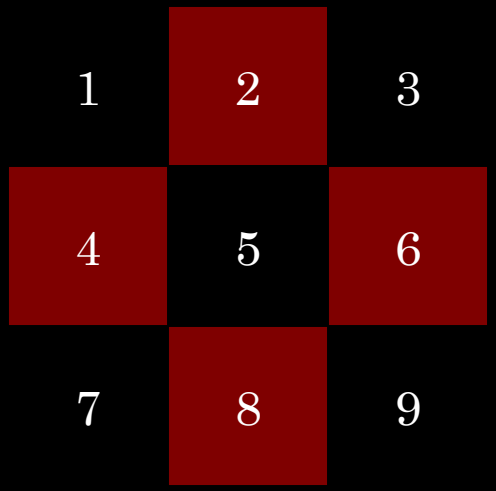
- NFA从一个初始状态出发
- 当存在一系列选择能到达终结状态时接受
- 直觉上说: NFA总是“guesses right.”

示例: 在棋盘上移动

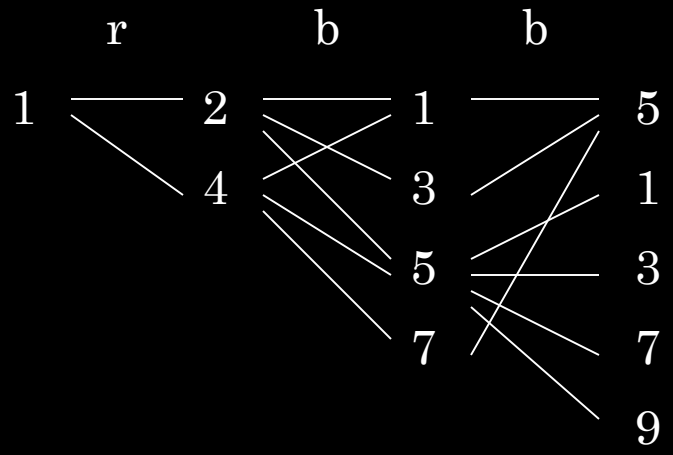
- States = 格子1-9
- Inputs = r (移动到相邻红格子)， b (移动到相邻黑格子).
- Start state, final state 对角



示例: 棋盘 – (2)



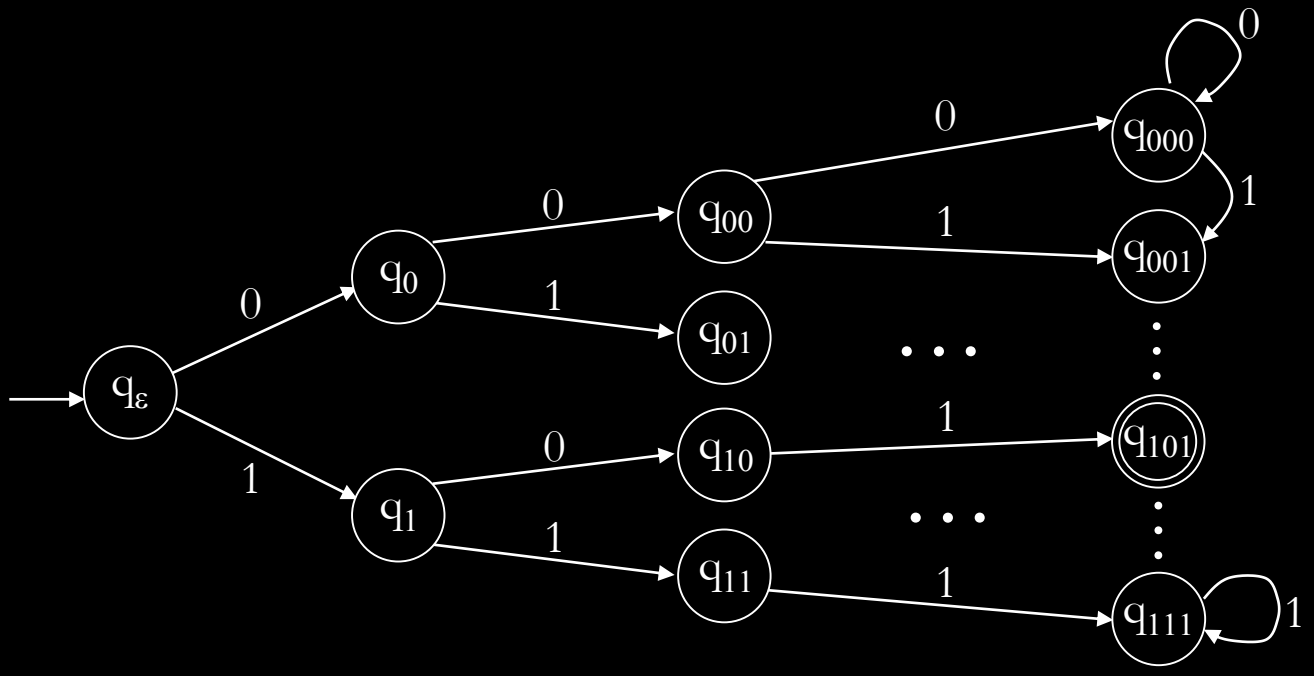
		r	b
→	1	2,4	5
	2	4,6	1,3,5
	3	2,6	5
	4	2,8	1,5,7
	5	2,4,6,8	1,3,7,9
	6	2,8	3,5,9
	7	4,8	5
	8	4,6	5,7,9
*	9	6,8	5



← Accept, since final state reached

还记得吗？上个单元的例子

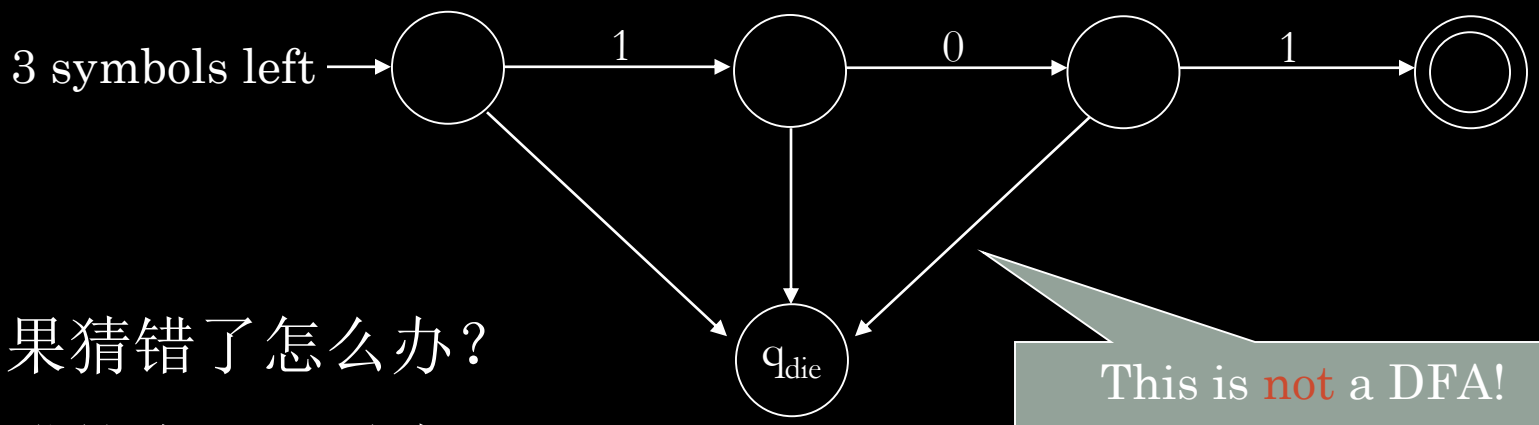
- DFA接受所有以101结尾的{0, 1}上符号串



引入非确定性，可以简化为……

- 假设我们能猜到何时符号串只剩下3个符号了
- 则直接查看是否为序列101，是的话就接受

- 因为我们可以复制出无数自动机，当符号串长度不小于3时，只要有也总有一个能猜对



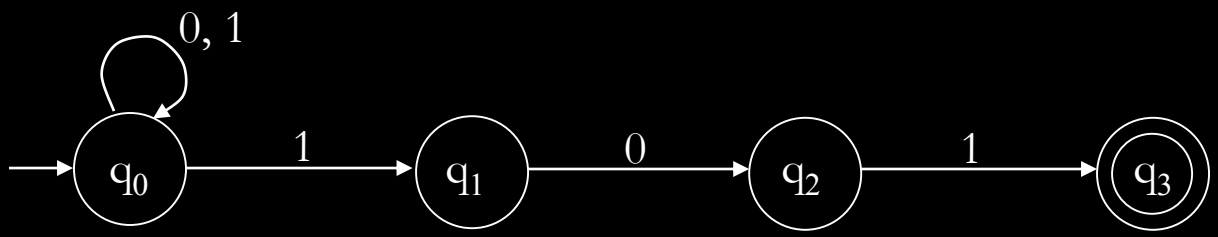
- 如果猜错了怎么办？
- 猜错就去死，重来！

非确定性
Nondeterminism

- 非确定性就是作猜测的能力，可以下面方法验证
- 识别以010结尾符号串语言的非形式化非确定性算法：
  1. 猜测输入串是否还剩3个符号，猜测是对是错？
  2. 如果是yes，则查看是否为101，是的话就接受
  3. 如果是no,，则再多读一个符号，回到步骤1

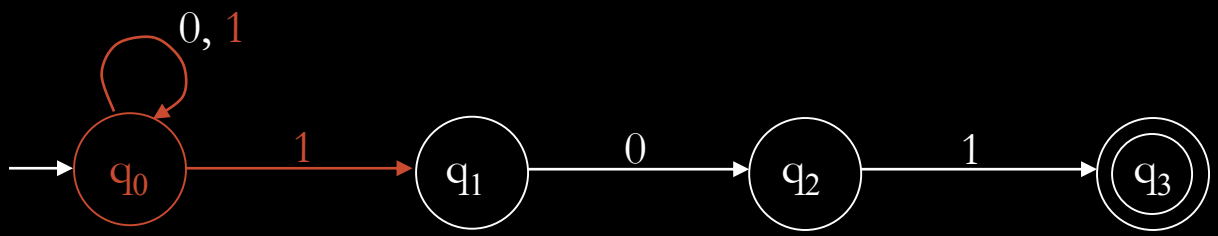
非确定型有穷自动机

- 这种自动机可以用来实现这样的猜测



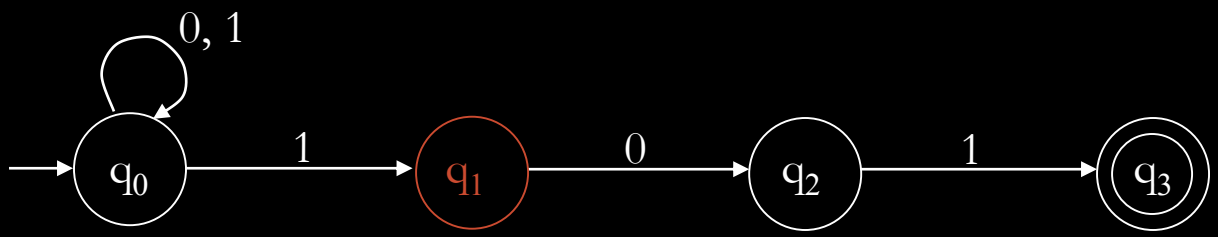
- 每个状态对同一个符号可以有0个、1个或多个转移

猜测的语义



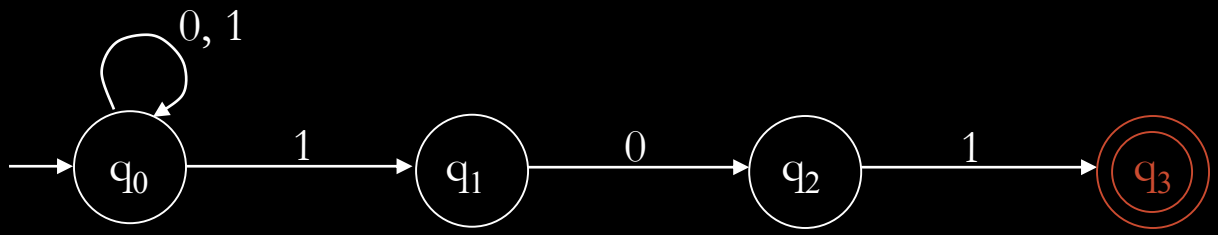
- 状态q_0有两个标签为1的转移
- 在读到1时，我们可以在停留于q_0 或移动到q_1之间进行选择

猜测的语义



- 状态q_1没有标签为1的转移，但有1个标签为0的转移
- 在q_1读到1时，我们失败了（死亡）；如果读到0，继续到q_2

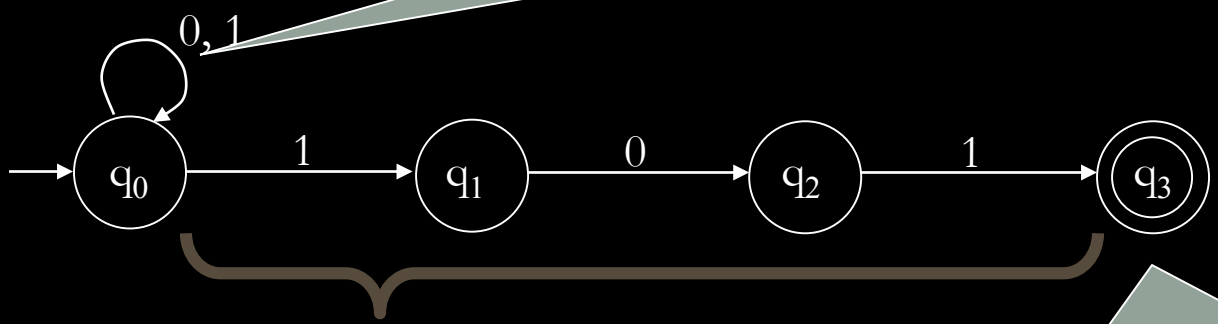
猜测的语义



- 状态q_3没有任何转移
- 在q_3读到任何符号，都将导致死亡的结局

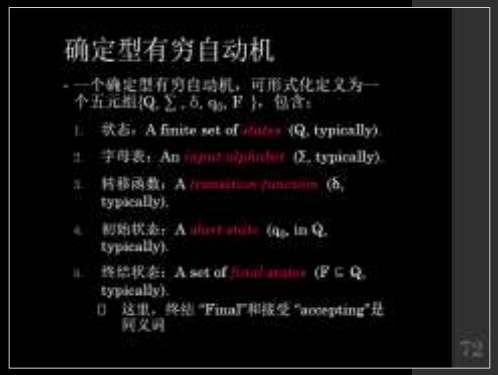
自动机的含义

猜测距输入末尾还有3个符号



如果确实如此，
猜测会看到101

猜猜看是不是到了输入末尾



形式化的NFA

NFA定义为五元组：$\{Q, \Sigma, \delta, q_0, F\}$

- 状态：A finite set of states, typically Q.
- 字母表：An input alphabet, typically Σ.
- 转移函数：A transition function, typically δ.
- 初始状态：A start state in Q, typically q_0.
- 终结状态：A set of final states $F \subseteq Q$.

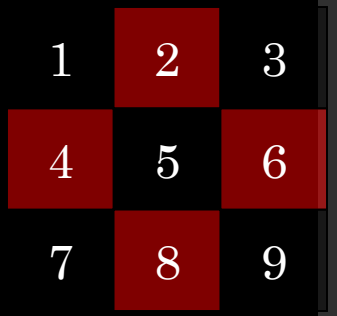
NFA转移函数

- δ(q, a)的结果是状态的一个集合
- 可扩展到符号串：
- Basis: δ(q, ϵ) = {q}
- Induction: δ(q, wa) = the union over all states p in δ(q, w) of δ(p, a)

NFA识别的语言

- NFA接收符号串w仅当$\delta(q_0, w)$包含至少一个终结状态
- NFA识别的语言是其接受符号串的集合

示例: NFA识别的语言

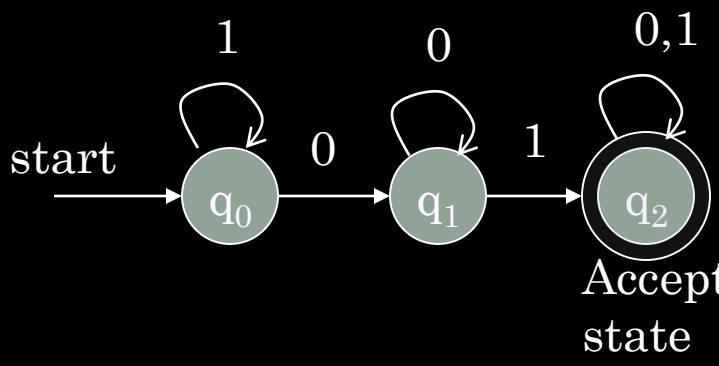


- 在棋盘例子中，NFA 接受rbb
- 如输入仅包含b，状态集合{5}和{1,3,7,9}交替可达，因此只有偶数长度的非空b串会被接受
- 如输入包含至少一个 r 又如何呢?

Regular expression: (0+1)*01(0+1)*

识别含01符号串的DFA

- 为什么此自动机是确定的?



- 如果要求语言中包括空串怎么办?
- 含10110的符号串又如何呢?

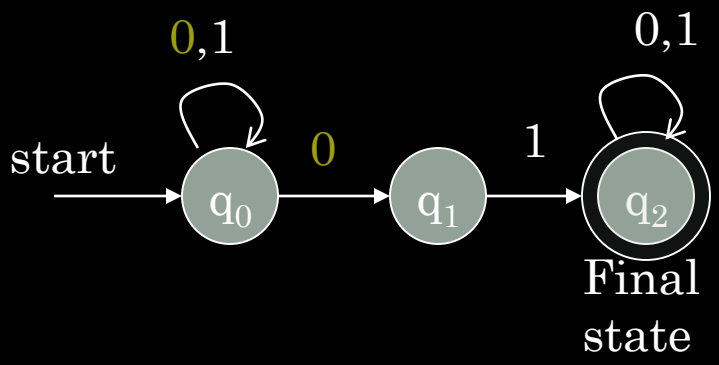
- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0, 1\}$
- start state = q_0
- $F = \{q_2\}$
- Transition table

	symbols	
states: δ	0	1
→ q_0	q_1	q_0
q_1	q_1	q_2
*q_2	q_2	q_2

Regular expression: (0+1)*01(0+1)*

识别含01符号串的NFA

为什么它是不确定的？



当在 q_1 处接受0会发生什么？

含10110的符号串又如何呢？

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0,1\}$
- start state = q_0
- $F = \{q_2\}$
- Transition table

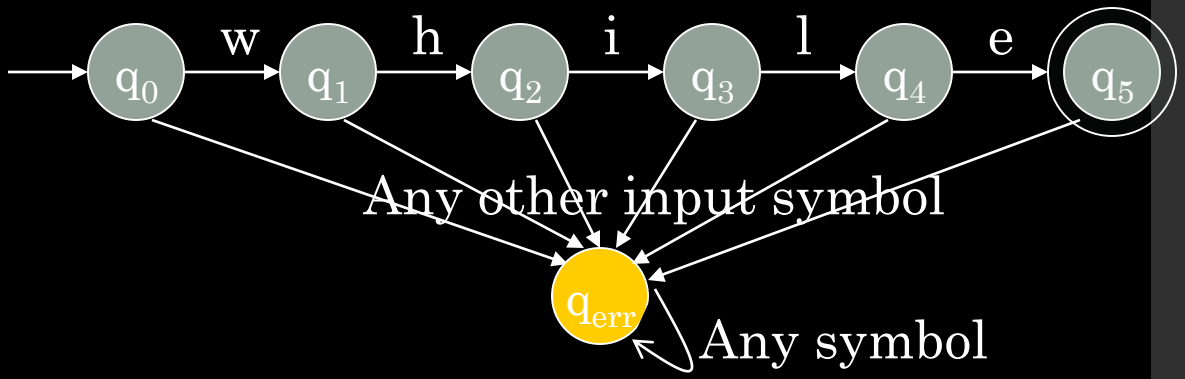
states	δ	0	1
→	q_0	$\{q_0, q_1\}$	$\{q_0\}$
	q_1	Φ	$\{q_2\}$
	$*q_2$	$\{q_2\}$	$\{q_2\}$

symbols

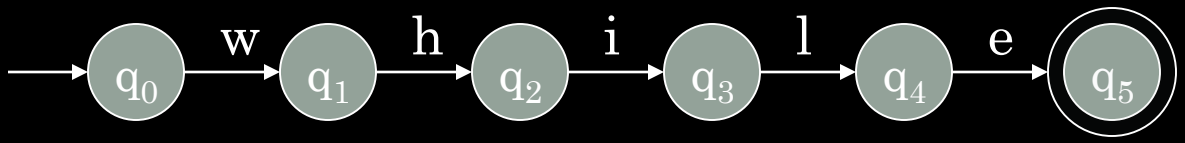
注意：省略显式地显示错误状态只是为了设计上的方便
（NFA通常遵循这一点），这个特性不应该与非确定性的概念混淆

什么是“错误状态(error state)”?

- 识别关键词“*while*”的DFA



- 同样功能的NFA



Transitions into a dead state are implicit

示例

- 为下列语言构造NFA
- L = { w | w ends in 01}

- 关键词识别：Keyword recognizer (e.g., if, then, else, while, for, include, etc.)

- 搜索：第一个符号在稍后某个位置出现至少一次的符号串

NFA的优势和注意事项 Advantages & Caveats

- 非常适合建模特定结构的符号串（正则表达式）
  - 字符串处理：grep
  - 词法分析：lex
- 但是一个不确定的状态机能否在现实中实现？
  - 概率模型可以看作是非确定性状态机的扩展（例如掷硬币、掷骰子）
    - 但与他们不一样
  - 并行计算机可以同时存在于多个“状态”中。量子计算机？

NFA的硬件技术

- Micron's Automata Processor (introduced in 2013)
- 2D array of MISD (multiple instruction single data) fabric w/ thousands to millions of processing elements.
- 1 input symbol = fed to all states (i.e., cores)
- Non-determinism using circuits
- http://www.micronautomata.com/

尽管如此，DFA与NFA在接受语言的能力上等价！

比较： DFA vs. NFA

- **DFA**

1. 所有转换都是确定的
   - 每个转换都指向恰好一个状态
2. 对每一个状态，所有可能的输入符号都必须定义对应的转换
3. 如果最终一个状态在F中，则接受输入
4. 有时比较难以构造，因为状态数太多
5. 现实中是可以直接实现的

- **NFA**

1. 有些转换可以是非确定的
   - 一个转换可能指向一个状态的集合
2. 并非所有输入符号的对应转换都要显式定义”
3. 最终状态集合中只要有一个在F中就接受输入
4. 通常比DFA更易于构造
5. 现实中的实现能力受限

DFA与NFA等价

- 先证DFA → NFA

- DFA可以转化接受相同语言的NFA
- 如$\delta_D(q, a) = p$，则令NFA有$\delta_N(q, a) = \{p\}$
- 这个NFA中转换指向的状态集合永远只包含一个状态，就是DFA 在接受同样输入符号时到达的状态

等价性 – (2)

- 再证NFA→DFA
- 对任意NFA，都存在一个DFA接受相同语言
- 通过子集构造法 *subset construction* 证明
- 该DFA的状态数可能NFA状态数的指数级
- 因此，NFA也恰好接受正则语言

子集构造法
Subset Construction

- 给定NFA有状态集合Q，输入符号集Σ，转换函数δ_N，初始状态q_0，以及终结状态集F，构造等价的DFA：
  - 状态集 2^Q (Q所有子集的集合)
  - 输入符号集 Σ
  - 初始状态$\{q_0\}$
  - 终结状态集 = 所有包含F中任意元素的状态

关键点

- DFA状态的名字是 NFA 状态的集合
- 对于DFA 状态，表达式{p,q} 是一个单独的名字，而不是一个集合
- 类比: 一个类的对象值是另一个类对象的集合

子集构造法 – (2)

- 转移函数δ_D 定义为:

$\delta_D(\{q_1,\ldots,q_k\}, a)$ 是对所有$i = 1,\ldots,k$的 $\delta_N(q_i, a)$结果的并集

- 示例: 构造DFA等价于“棋盘” NFA

示例: 子集构造法

		r	b
→	1	2,4	5
	2	4,6	1,3,5
	3	2,6	5
	4	2,8	1,5,7
	5	2,4,6,8	1,3,7,9
	6	2,8	3,5,9
	7	4,8	5
	8	4,6	5,7,9
*	9	6,8	5

	r	b
→ {1}	{2,4}	{5}
{2,4}		
{5}		

注意:这里是DFA构造的惰性（*lazy*）形式，
即只有当必要的时候才构造一个新的状态
这样通常可以减少最终的状态数

示例:子集构造法

		r	b
→	1	2,4	5
	2	4,6	1,3,5
	3	2,6	5
	4	2,8	1,5,7
	5	2,4,6,8	1,3,7,9
	6	2,8	3,5,9
	7	4,8	5
	8	4,6	5,7,9
*	9	6,8	5

	r	b
→ {1}	{2,4}	{5}
{2,4}	{2,4,6,8}	{1,3,5,7}
{5}		
{2,4,6,8}		
{1,3,5,7}		

示例:子集构造法

		r	b
→	1	2,4	5
	2	4,6	1,3,5
	3	2,6	5
	4	2,8	1,5,7
	5	2,4,6,8	1,3,7,9
	6	2,8	3,5,9
	7	4,8	5
	8	4,6	5,7,9
*	9	6,8	5

		r	b
→	{1}	{2,4}	{5}
	{2,4}	{2,4,6,8}	{1,3,5,7}
	{5}	{2,4,6,8}	{1,3,7,9}
	{2,4,6,8}		
	{1,3,5,7}		
*	{1,3,7,9}		

示例:子集构造法

		r	b
→	1	2,4	5
	2	4,6	1,3,5
	3	2,6	5
	4	2,8	1,5,7
	5	2,4,6,8	1,3,7,9
	6	2,8	3,5,9
	7	4,8	5
	8	4,6	5,7,9
*	9	6,8	5

		r	b
→	{1}	{2,4}	{5}
	{2,4}	{2,4,6,8}	{1,3,5,7}
	{5}	{2,4,6,8}	{1,3,7,9}
	{2,4,6,8}	{2,4,6,8}	{1,3,5,7,9}
	{1,3,5,7}		
*	{1,3,7,9}		
*	{1,3,5,7,9}		

示例:子集构造法

		r	b
→	1	2,4	5
	2	4,6	1,3,5
	3	2,6	5
	4	2,8	1,5,7
	5	2,4,6,8	1,3,7,9
	6	2,8	3,5,9
	7	4,8	5
	8	4,6	5,7,9
*	9	6,8	5

		r	b
→	{1}	{2,4}	{5}
	{2,4}	{2,4,6,8}	{1,3,5,7}
	{5}	{2,4,6,8}	{1,3,7,9}
	{2,4,6,8}	{2,4,6,8}	{1,3,5,7,9}
	{1,3,5,7}	{2,4,6,8}	{1,3,5,7,9}
*	{1,3,7,9}		
*	{1,3,5,7,9}		

示例:子集构造法

		r	b
→	1	2,4	5
	2	4,6	1,3,5
	3	2,6	5
	4	2,8	1,5,7
	5	2,4,6,8	1,3,7,9
	6	2,8	3,5,9
	7	4,8	5
	8	4,6	5,7,9
*	9	6,8	5

		r	b
→	{1}	{2,4}	{5}
	{2,4}	{2,4,6,8}	{1,3,5,7}
	{5}	{2,4,6,8}	{1,3,7,9}
	{2,4,6,8}	{2,4,6,8}	{1,3,5,7,9}
	{1,3,5,7}	{2,4,6,8}	{1,3,5,7,9}
*	{1,3,7,9}	{2,4,6,8}	{5}
*	{1,3,5,7,9}		

示例:子集构造法

		r	b
→	1	2,4	5
	2	4,6	1,3,5
	3	2,6	5
	4	2,8	1,5,7
	5	2,4,6,8	1,3,7,9
	6	2,8	3,5,9
	7	4,8	5
	8	4,6	5,7,9
*	9	6,8	5

		r	b
→	{1}	{2,4}	{5}
	{2,4}	{2,4,6,8}	{1,3,5,7}
	{5}	{2,4,6,8}	{1,3,7,9}
	{2,4,6,8}	{2,4,6,8}	{1,3,5,7,9}
	{1,3,5,7}	{2,4,6,8}	{1,3,5,7,9}
*	{1,3,7,9}	{2,4,6,8}	{5}
*	{1,3,5,7,9}	{2,4,6,8}	{1,3,5,7,9}

等价性证明: 子集构造法

- 归纳法证明
- 基于 |w| 进行归纳，归纳假设（IH）：

$$\delta_N(q_0, w) = \delta_D(\{q_0\}, w)$$

- Basis: $w = \epsilon$:
- $$\delta_N(q_0, \epsilon) = \delta_D(\{q_0\}, \epsilon) = \{q_0\}.$$

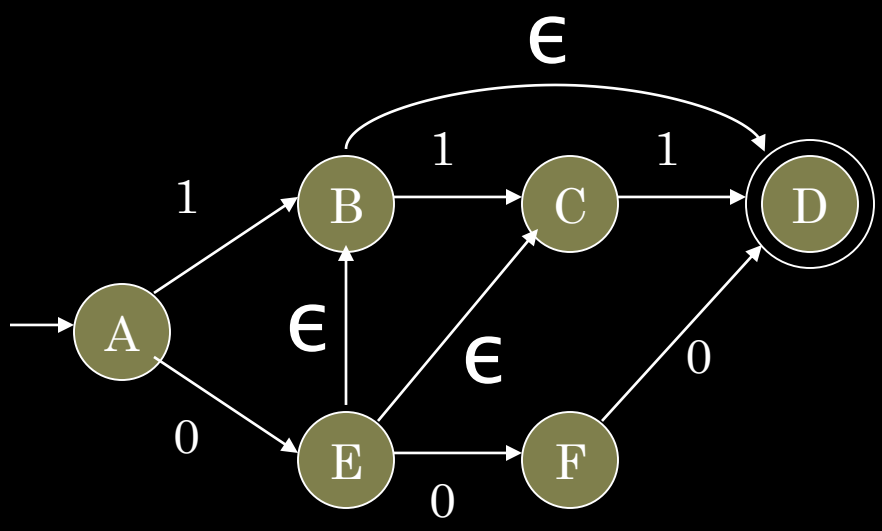
归纳步骤

- 假设IH对比w短的符号串成立
- 令w = xa; IH对x成立
- 令$\delta_N(q_0, x) = \delta_D(\{q_0\}, x) = S$
- 令T为S中所有状态p对应$\delta_N(p, a)$的并集
- 则$\delta_N(q_0, w) = \delta_D(\{q_0\}, w) = T$
  - 对NFA: 由δ_N定义可证
  - 对DFA: 由δ_D定义加上对δ_D的扩展
    - 先有$\delta_D(S, a) = T$，再扩展δ_D到输入w = xa.

带空转移的NFA
NFA's With ϵ-Transitions

- 允许ϵ输入（即没有输入）时发生状态转移
- 这些转换是自动完成的，不需要查看输入符号串
- 这样有时会很方便构造自动机
- 但仍然只接受正则语言

示例: ϵ-NFA

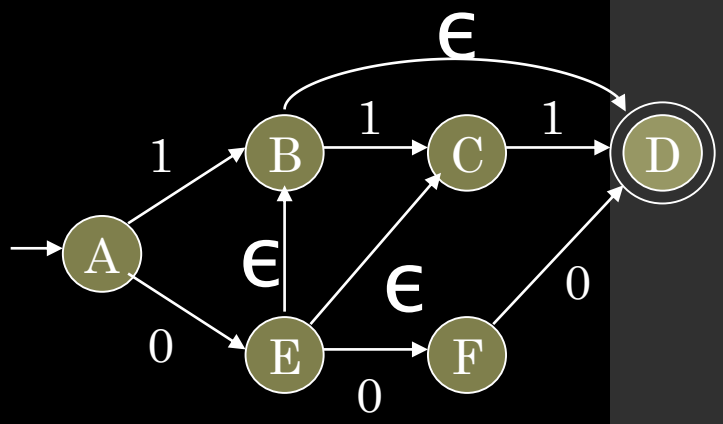


		0	1	ϵ
→	A	{E}	{B}	∅
	B	∅	{C}	{D}
	C	∅	{D}	∅
*	D	∅	∅	∅
	E	{F}	∅	{B, C}
	F	{D}	∅	∅

状态的闭包
Closure of States

- CL(q) = 所有从状态q出发可以通过标签为ϵ的弧可以到达的状态集合
- 示例: CL(A) = {A};

  CL(E) = {B, C, D, E}.



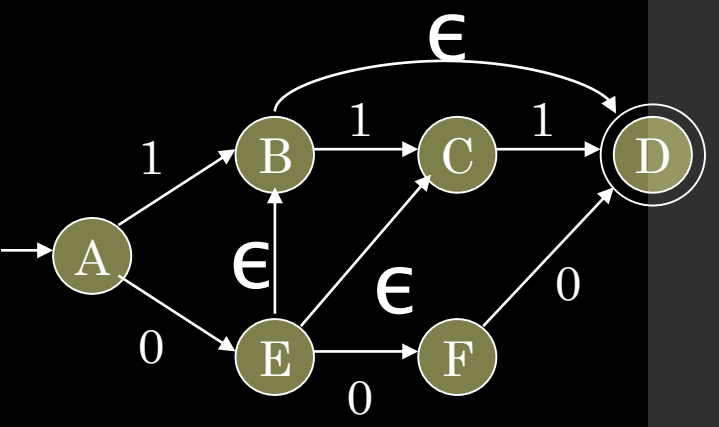
- 状态集合的闭包 = 集合中每个状态闭包的并集

扩展转移函数

- Basis: $\hat{\delta}(q, \epsilon) = CL(q)$.
- Induction: $\hat{\delta}(q, xa)$ 计算如下:
  1. 初始$\hat{\delta}(q, x) = S$.
  2. 取所有S中p的$CL(\hat{\delta}(p, a))$ 的并集
- Intuition: $\hat{\delta}(q, w)$ 是从q出发通过标签为w的路径能够到达的状态集合

注意当a是单个符号时，$\hat{\delta}(q, a)$ 不是状态的集合

示例: 扩展转移函数



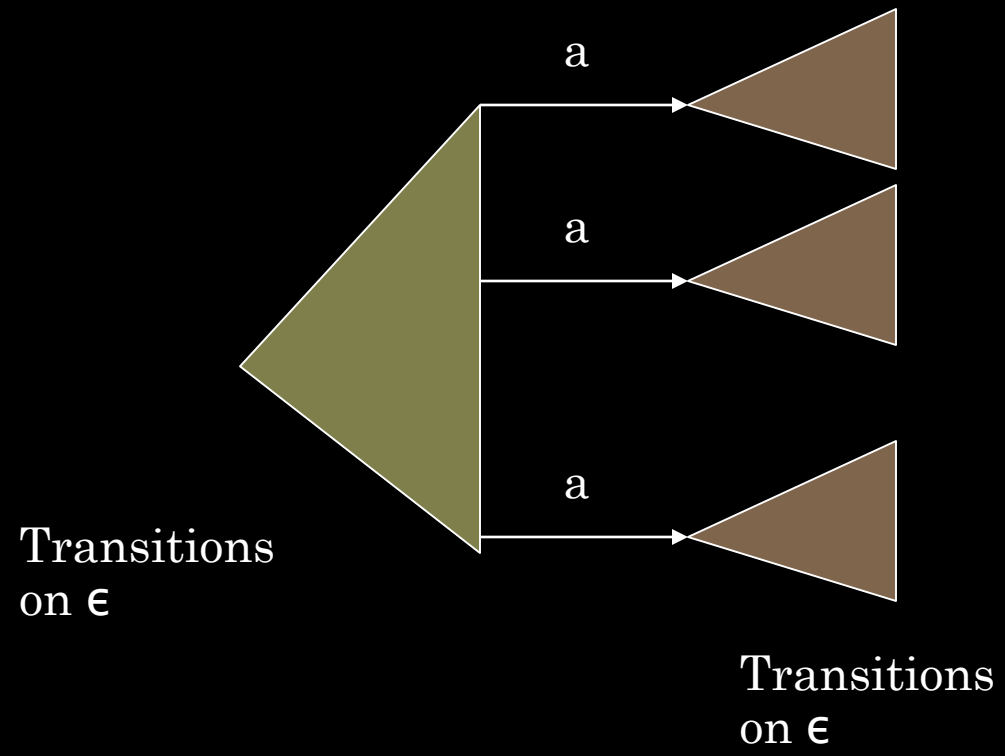
- $\hat{\delta}(A, \epsilon) = CL(A) = \{A\}$.
- $\hat{\delta}(A, 0) = CL(\{E\}) = \{B, C, D, E\}$.
- $\hat{\delta}(A, 01) = CL(\{C, D\}) = \{C, D\}$.
- ϵ-NFA的*语言* 是形如$\hat{\delta}(q_0, w)$包含终结状态的符号串w的集合

NFA, ϵ-NFA的等价性

- 每一个NFA都是 ϵ-NFA
  - 仅仅是包含0个ϵ转移
- 相对的， 要求对任意ϵ-NFA能够构造出一个NFA 接受相同的语言
- 思路是将空转移与接下来的有真实输入的转移结合在一起

注意:此处证明与课本方法稍有不同

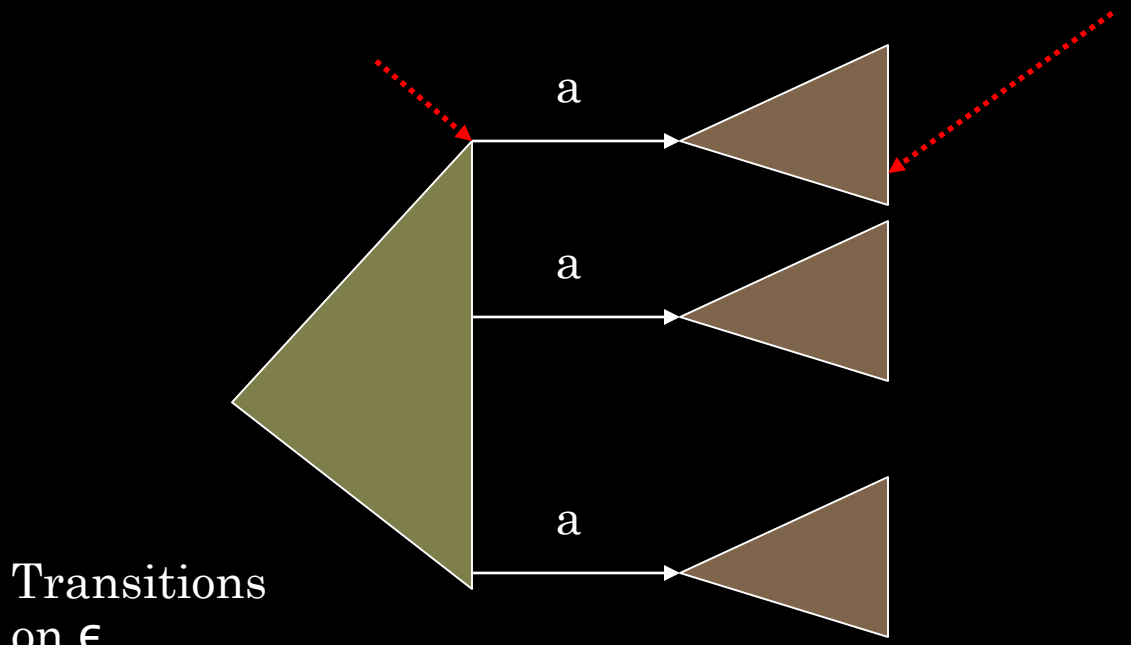
图解空转移消除



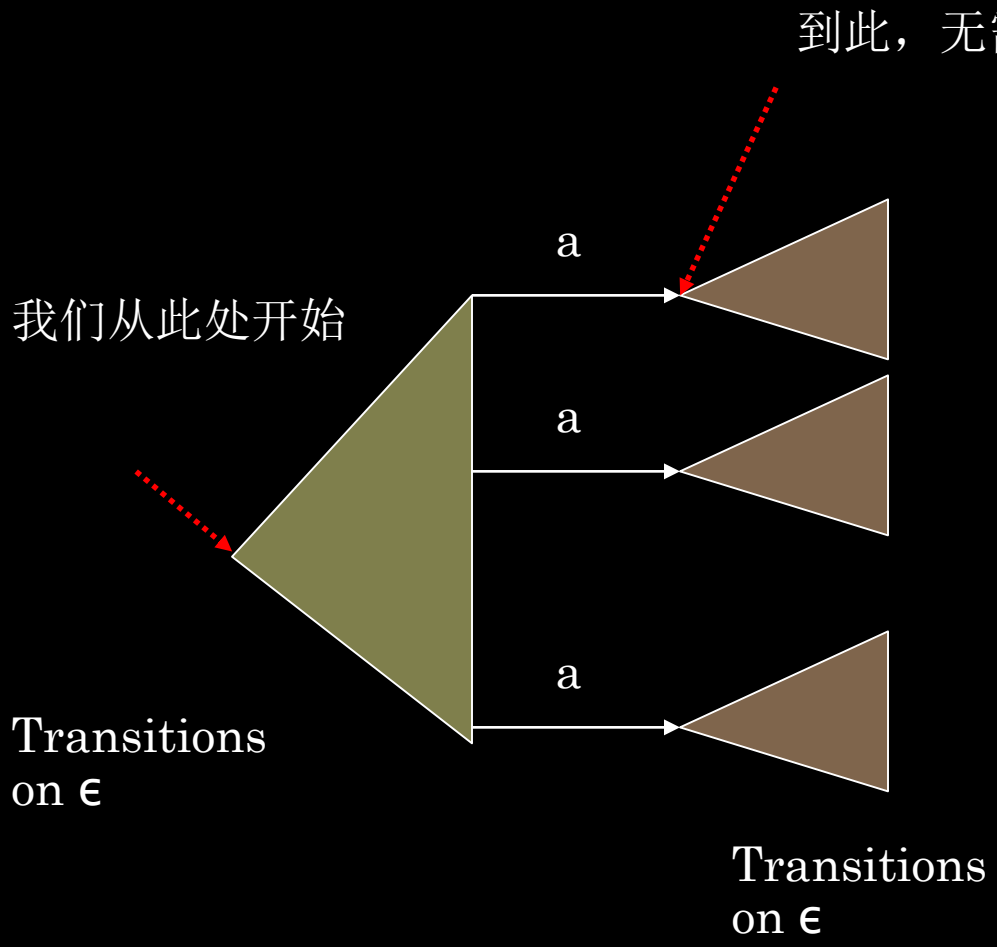
图解空转移消除

课文证明从此开始

到此处进行子集构造



图解空转移消除



Equivalence – (2)

- Start with an ϵ-NFA with states Q, inputs Σ, start state q_0, final states F, and transition function δ_E.
- Construct an "ordinary" NFA with states Q, inputs Σ, start state q_0, final states F', and transition function δ_N.

Equivalence – (3)

- Compute $\delta_N(q, a)$ as follows:
  1. Let S = CL(q).
  2. $\delta_N(q, a)$ is the union over all p in S of $\delta_E(p, a)$.
- F' = the set of states q such that CL(q) contains a state of F.
- Intuition: δ_N incorporates ϵ–transitions before using a but not after.

等价性证明

- 令ϵ-NFA = $(Q, \Sigma, q_0, F, \delta_E)$
- 构造NFA = $(Q, \Sigma, q_0, F', \delta_N)$

等价性证明– (3)

- 通过对 |w| 归纳可证

$$CL(\delta_N(q_0, w)) = \hat{\delta}_E(q_0, w).$$

- 因此，ϵ-NFA 接受w 当且仅当NFA 也接受

闭包: CL(B)
= {B,D}; CL(E)
= {B,C,D,E}

示例: ϵ-NFA-to-NFA

		0	1	ϵ
→	A	{E}	{B}	∅
	B	∅	{C}	{D}
	C	∅	{D}	∅
*	D	∅	∅	∅
	E	{F}	∅	{B, C}
	F	{D}	∅	∅

ϵ-NFA

		0	1
→	A	{E}	{B}
*	B	∅	{C}
	C	∅	{D}
*	D	∅	∅
*	E	{F}	{C, D}
	F	{D}	∅

因为E的闭包包含 B 和 C，可通过输入1转换到 C 和 D.

因为B 和 E 的闭包包含终结状态D.

小结

- DFA、NFA、ϵ–NFA都接受同一种语言：正则语言
- NFA通常更易于设计，且状态数可能比对应的DFA指数级减少
- 但只有DFA能在现实中实现！

作业3

- 1. 构造NFA
  - 构造一个NFA识别包含“10110”的01串
  - 构造一个NFA识别不包含“10110”的01串
- 2.NFA → DFA

	0	1
$\rightarrow q_0$	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\varnothing$	$\{q_1, q_2\}$
$*q_2$	$\varnothing$	$\varnothing$