词法分析

WANG Hanfei

September 17, 2015

©hfwang

1. 基本概念
   - 定义
   - 分析方法
   - 正则表达式
2. 确定的有限自动机
   - 基本概念
   - DFA 的设计
   - DFA 算法
3. 不确定的有限自动机
   - 基本概念
   - 正则表达式到 NFA 的转换
   - NFA 到 DFA 的转换
   - DFA 状态最小化
4. 相关问题

©hfwang

1 基本概念
- 定义
- 分析方法
- 正则表达式

2 确定的有限自动机
- 基本概念
- DFA 的设计
- DFA 算法

3 不确定的有限自动机
- 基本概念
- 正则表达式到 NFA 的转换
- NFA 到 DFA 的转换
- DFA 状态最小化

4 相关问题

©hfwang

概念

词法分析

- 词法分析 (Lexical Analysis)是编译器的第一阶段，完成该工作的程序称为词法分析器 (lexer, scanner).
- Lexer对字符源文件进行分词和重组产生编译器下一阶段语法分析所需要的单词序列.
- Lexer还需要过滤掉 white spaces 和注释.
- 单词是字符串的一个分类，或用数学语言讲是字符串集合的子集合.
- 本章要解决的问题：单词的形式化描述方法和识别方法.

Example: XL 语言

概念

词法分析

- 词法分析 (Lexical Analysis)是编译器的第一阶段，完成该工作的程序称为词法分析器 (lexer, scanner).
- Lexer对字符源文件进行分词和重组产生编译器下一阶段语法分析所需要的单词序列.
- Lexer还需要过滤掉 white spaces 和注释.
- 单词是字符串的一个分类，或用数学语言讲是字符串集合的子集合.
- 本章要解决的问题：单词的形式化描述方法和识别方法.

Example: XL 语言

概念

词法分析

- 词法分析 (Lexical Analysis)是编译器的第一阶段，完成该工作的程序称为词法分析器 (lexer, scanner).
- Lexer对字符源文件进行分词和重组产生编译器下一阶段语法分析所需要的单词序列.
- Lexer还需要过滤掉 white spaces 和注释.
- 单词是字符串的一个分类，或用数学语言讲是字符串集合的子集合.
- 本章要解决的问题：单词的形式化描述方法和识别方法.

Example: XL 语言

概念

词法分析

- 词法分析 (Lexical Analysis)是编译器的第一阶段，完成该工作的程序称为词法分析器 (lexer, scanner).
- Lexer对字符源文件进行分词和重组产生编译器下一阶段语法分析所需要的单词序列.
- Lexer还需要过滤掉 white spaces 和注释.
- 单词是字符串的一个分类，或用数学语言讲是字符串集合的子集合.
- 本章要解决的问题：单词的形式化描述方法和识别方法.

Example: XL 语言

概念

词法分析

- 词法分析 (Lexical Analysis)是编译器的第一阶段，完成该工作的程序称为词法分析器 (lexer, scanner).
- Lexer对字符源文件进行分词和重组产生编译器下一阶段语法分析所需要的单词序列.
- Lexer还需要过滤掉 white spaces 和注释.
- 单词是字符串的一个分类，或用数学语言讲是字符串集合的子集合.
- 本章要解决的问题：单词的形式化描述方法和识别方法.

Example: XL 语言

概念

词法分析

- 词法分析 (Lexical Analysis)是编译器的第一阶段，完成该工作的程序称为词法分析器 (lexer, scanner).
- Lexer对字符源文件进行分词和重组产生编译器下一阶段语法分析所需要的单词序列.
- Lexer还需要过滤掉 white spaces 和注释.
- 单词是字符串的一个分类，或用数学语言讲是字符串集合的子集合.
- 本章要解决的问题：单词的形式化描述方法和识别方法.

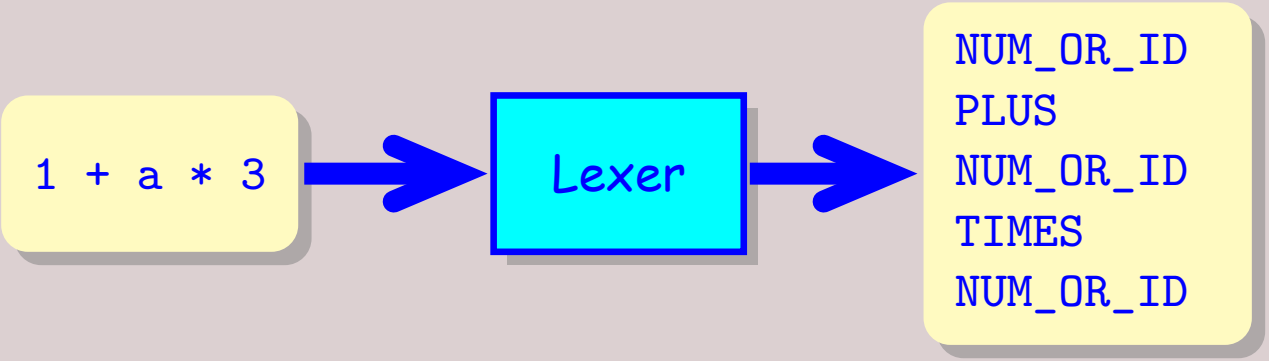
Example: XL 语言

概念

词法分析

- 词法分析 (Lexical Analysis)是编译器的第一阶段，完成该工作的程序称为词法分析器 (lexer, scanner).
- Lexer对字符源文件进行分词和重组产生编译器下一阶段语法分析所需要的单词序列.
- Lexer还需要过滤掉 white spaces 和注释.
- 单词是字符串的一个分类，或用数学语言讲是字符串集合的子集合.
- 本章要解决的问题：单词的形式化描述方法和识别方法.

Example: XL 语言



词法分析方法

分析方法分类

- 手工编程，如：
  - XL 语言：buffer 的处理，while(1) 多重循环，switch-case 语句对输入的每个字符进行判断，
  - lcc(Princeton 大学 C 编译器)，Fraser and Hanson，Compiler 第 2 章，见本章 C 编译器 — 词法分析与语法分析
- 自动方法：

词法分析方法

分析方法分类

- 手工编程，如：
  - XL 语言：buffer 的处理，while(1) 多重循环，switch-case 语句对输入的每个字符进行判断.
  - lcc(Princeton 大学 C 编译器), A retargetable C Compiler 第六章，见文件：C 编译器 ---设计与实现.pdf.
- 自动方法：

词法分析方法

分析方法分类

- 手工编程，如：
  - XL 语言：buffer 的处理，while(1) 多重循环，switch-case 语句对输入的每个字符进行判断.
  - lcc(Princeton 大学 C 编译器), A retargetable C Compiler 第六章，见文件：C 编译器 ---设计与实现.pdf.
- 自动方法：

词法分析方法

分析方法分类

- 手工编程，如：
  - XL 语言：buffer 的处理，while(1) 多重循环，switch-case 语句对输入的每个字符进行判断.
  - lcc(Princeton 大学 C 编译器), A retargetable C Compiler 第六章，见文件：C 编译器 ---设计与实现.pdf.
- 自动方法：

词法分析方法

分析方法分类

- 手工编程，如：
  - XL 语言：buffer 的处理，while(1) 多重循环，switch-case 语句对输入的每个字符进行判断.
  - lcc(Princeton 大学 C 编译器), A retargetable C Compiler 第六章，见文件：C 编译器 ---设计与实现.pdf.
- 自动方法：

字符串集合的数学定义.

用集合论语言从对字符串集合进行精确的描述！

Σ^*的递归定义

设 Σ 是一个有限字母表集合:

- $\varepsilon \in \Sigma^*$，称之为空串(归纳基础).
- 设 $a \in \Sigma$, $w \in \Sigma^*$, 则 $aw \in \Sigma^*$(归纳步骤)
- Σ^* 中的元素称为词(word), 习惯上用 w 为 $a_1a_2\ldots a_n(a_i \in \Sigma)$ 简记 n 元组 $(a_1, a_2, \ldots, a_n)$.

由上定义可得:

- 空集合 $\emptyset$ (集合 Σ 的全集).

Example

设 $\Sigma = \{a - z, 0 - 9, +, *, (,)\}$,
则 $abc \in \Sigma^*$, $(+ - a \notin \Sigma^*$, $0a + b \in \Sigma^*$, $a/b \notin \Sigma^*$.

字符串集合的数学定义

用集合论语言从对字符串集合进行精确的描述！

Σ^*的递归定义

设 Σ 是一个有限字母表集合：

- $\varepsilon \in \Sigma^*$，称之为空串(归纳基础).
- 设 $a \in \Sigma, w \in \Sigma^*$，则序偶 $\langle a, w\rangle \in \Sigma^*$(归纳条款).
- Σ^* 中的元素称为词 (word)，习惯上用序列 $a_1a_2\cdots a_n(a_i \in \Sigma)$ 简记 n 重组 $\langle a_1, a_2, \ldots a_n\rangle$.

由上定义可知：

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n \quad (\text{其中：} \Sigma^0 \triangleq \{\varepsilon\})$$

Example

设 $\Sigma = \{a - z, 0 - 9, +, *, (,)\}$，
则 $abc \in \Sigma^*$，$(+-a \notin \Sigma^*$，$0a+b \in \Sigma^*$，$a/b \notin \Sigma^*$.

字符串集合的数学定义

用集合论语言从对字符串集合进行精确的描述！

Σ^*的递归定义

设 Σ 是一个有限字母表集合：

- $\varepsilon \in \Sigma^*$，称之为空串(归纳基础).
- 设 $a \in \Sigma, w \in \Sigma^*$，则序偶 $\langle a, w\rangle \in \Sigma^*$(归纳条款).
- Σ^* 中的元素称为词 (word)，习惯上用序列 $a_1a_2\cdots a_n(a_i \in \Sigma)$ 简记 n 重组 $\langle a_1, a_2, \ldots a_n\rangle$.

由上定义可知：

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n \quad (\text{其中: } \Sigma^0 \triangleq \{\varepsilon\})$$

Example

设 $\Sigma = \{a - z, 0 - 9, +, *, (,)\}$，

则 $abc \in \Sigma^*$，$(+-a \notin \Sigma^*$，$0a+b \in \Sigma^*$，$a/b \notin \Sigma^*$.

字符串集合的数学定义

用集合论语言从对字符串集合进行精确的描述！

Σ^*的递归定义

设 Σ 是一个有限字母表集合：

- $\varepsilon \in \Sigma^*$，称之为空串(归纳基础).
- 设 $a \in \Sigma, w \in \Sigma^*$，则序偶 $\langle a, w\rangle \in \Sigma^*$(归纳条款).
- Σ^* 中的元素称为词 (word)，习惯上用序列 $a_1a_2\cdots a_n(a_i \in \Sigma)$ 简记 n 重组 $\langle a_1, a_2, \ldots a_n\rangle$.

由上定义可知：

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n \quad (\text{其中: } \Sigma^0 \triangleq \{\varepsilon\})$$

Example

设 $\Sigma = \{a - z, 0 - 9, +, *, (,)\}$，

则 $abc \in \Sigma^*$，$(+ - a \notin \Sigma^*$，$0a + b \in \Sigma^*$，$a/b \notin \Sigma^*$.

字符串集合的数学定义

用集合论语言从对字符串集合进行精确的描述！

Σ^*的递归定义

设 Σ 是一个有限字母表集合：

- $\varepsilon \in \Sigma^*$，称之为空串(归纳基础).
- 设 $a \in \Sigma, w \in \Sigma^*$，则序偶 $\langle a, w\rangle \in \Sigma^*$(归纳条款).
- Σ^* 中的元素称为词 (word)，习惯上用序列 $a_1a_2\cdots a_n(a_i \in \Sigma)$ 简记 n 重组 $\langle a_1, a_2, \ldots a_n\rangle$.

由上定义可知：

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n \quad (\text{其中：} \Sigma^0 \triangleq \{\varepsilon\})$$

Example

设 $\Sigma = \{a - z, 0 - 9, +, *, (,)\}$，

则 $abc \in \Sigma^*$，$(+-a \notin \Sigma^*$，$0a+b \in \Sigma^*$，$a/b \notin \Sigma^*$.

字符串集合的数学定义

用集合论语言从对字符串集合进行精确的描述！

Σ^*的递归定义

设 Σ 是一个有限字母表集合：

- $\varepsilon \in \Sigma^*$，称之为空串(归纳基础).
- 设 $a \in \Sigma, w \in \Sigma^*$，则序偶 $\langle a, w\rangle \in \Sigma^*$(归纳条款).
- Σ^* 中的元素称为词 (word)，习惯上用序列 $a_1a_2\cdots a_n(a_i \in \Sigma)$ 简记 n 重组 $\langle a_1, a_2, \ldots a_n\rangle$.

由上定义可知：

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n \quad (\text{其中：} \Sigma^0 \triangleq \{\varepsilon\})$$

Example

设 $\Sigma = \{a - z, 0 - 9, +, *, (,)\}$，

则 $abc \in \Sigma^*$，$(+-a \notin \Sigma^*$，$0a+b \in \Sigma^*$，$a/b \notin \Sigma^*$.

连接运算

设 Σ^* 为字符串集合，Σ^* 上的连接运算 (concatenation)``$\cdot$'' 递归定义如下：

- $\forall w \in \Sigma^*$, $\varepsilon \cdot w = w$ (归纳基础).
- 设 $u, v \in \Sigma^*$，且 $u \cdot v$ 已定义，则 $\forall a \in \Sigma$, $\langle a, u \rangle \cdot v = \langle a, u \cdot v \rangle$(归纳条款).

由离散数学得知，$\langle \Sigma^*, \cdot, \varepsilon \rangle$ 构成一个么半群. $u \cdot v$ 通常简记为 uv.

相关记号

连接运算

设 Σ^* 为字符串集合，Σ^* 上的连接运算 (concatenation)``$\cdot$'' 递归定义如下：

- $\forall w \in \Sigma^*$, $\varepsilon \cdot w = w$ (归纳基础).
- 设 $u, v \in \Sigma^*$，且 $u \cdot v$ 已定义，则 $\forall a \in \Sigma$, $\langle a, u\rangle \cdot v = \langle a, u \cdot v\rangle$(归纳条款).

由离散数学得知，$\langle \Sigma^*, \cdot, \varepsilon\rangle$ 构成一个么半群. $u \cdot v$ 通常简记为 uv.

相关记号

Σ^* 中任意一个 w, 称之为由 w 的一个字符串 (string).

连接运算

设 Σ^* 为字符串集合，Σ^* 上的连接运算 (concatenation)``$\cdot$'' 递归定义如下：

- $\forall w \in \Sigma^*$, $\varepsilon \cdot w = w$ (归纳基础).
- 设 $u, v \in \Sigma^*$，且 $u \cdot v$ 已定义，则 $\forall a \in \Sigma$, $\langle a, u\rangle \cdot v = \langle a, u \cdot v\rangle$(归纳条款).

由离散数学得知，$\langle \Sigma^*, \cdot, \varepsilon\rangle$ 构成一个么半群. $u \cdot v$ 通常简记为 uv.

相关记号

- $w^n \triangleq \underbrace{ww\cdots w}_{n\text{次}}$, 称之为词 w 的 n 次幂 (power).

连接运算

设 Σ^* 为字符串集合，Σ^* 上的连接运算 (concatenation)``$\cdot$'' 递归定义如下：

- $\forall w \in \Sigma^*$, $\varepsilon \cdot w = w$ (归纳基础).
- 设 $u, v \in \Sigma^*$，且 $u \cdot v$ 已定义，则 $\forall a \in \Sigma$, $\langle a, u\rangle \cdot v = \langle a, u \cdot v\rangle$(归纳条款).

由离散数学得知，$\langle \Sigma^*, \cdot, \varepsilon\rangle$ 构成一个么半群. $u \cdot v$ 通常简记为 uv.

相关记号

- $w^n \triangleq \underbrace{ww\cdots w}_{n\text{次}}$, 称之为词 w 的 n 次幂 (power).

连接运算

设 Σ^* 为字符串集合，Σ^* 上的连接运算 (concatenation)``$\cdot$'' 递归定义如下：

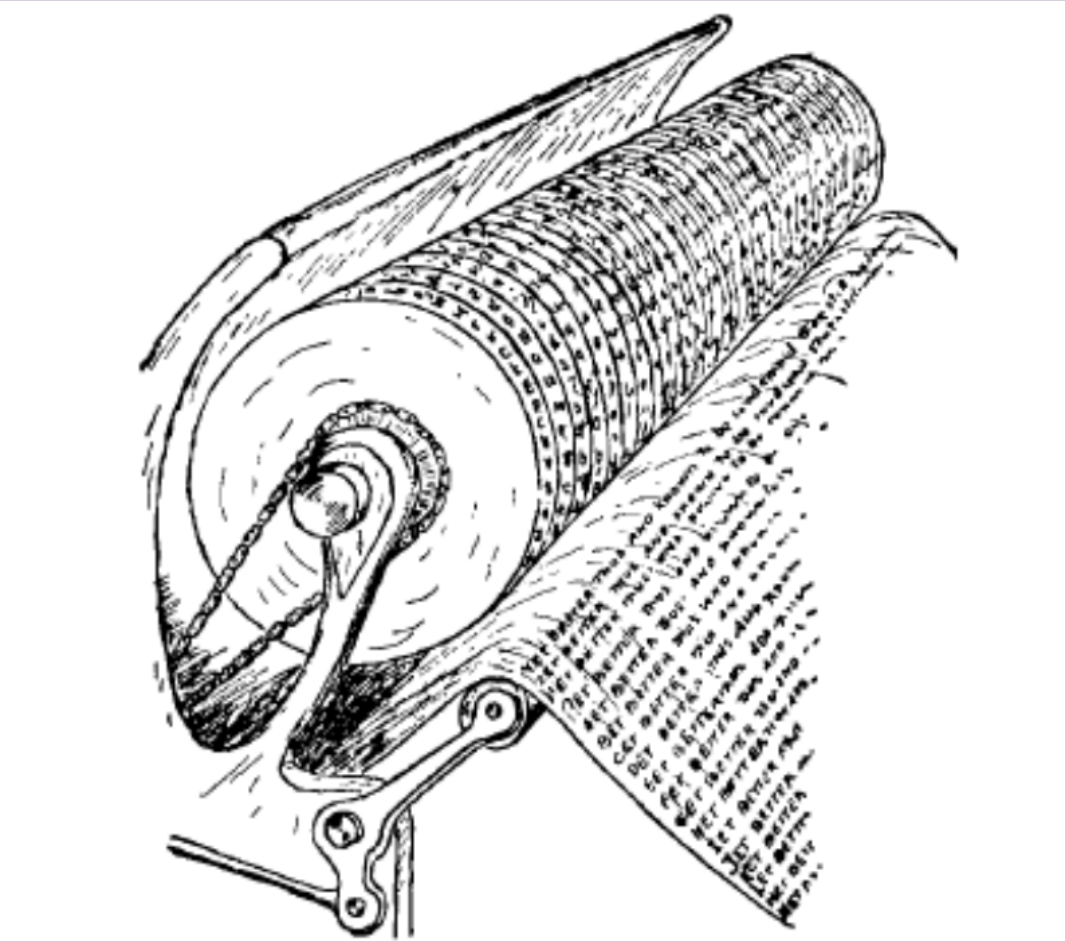
- $\forall w \in \Sigma^*$, $\varepsilon \cdot w = w$ (归纳基础).
- 设 $u, v \in \Sigma^*$，且 $u \cdot v$ 已定义，则 $\forall a \in \Sigma$, $\langle a, u\rangle \cdot v = \langle a, u \cdot v\rangle$(归纳条款).

由离散数学得知，$\langle \Sigma^*, \cdot, \varepsilon\rangle$ 构成一个么半群. $u \cdot v$ 通常简记为 uv.

相关记号

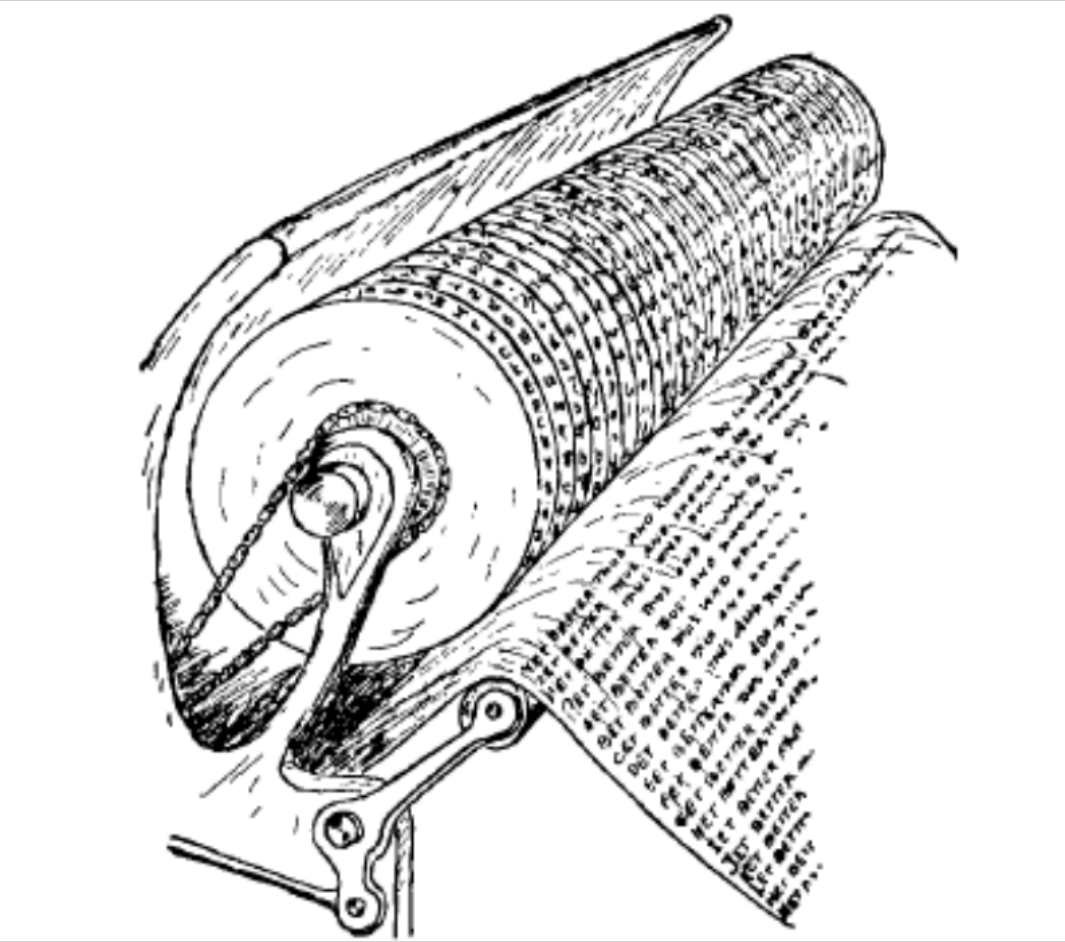
- $w^n \triangleq \underbrace{ww\cdots w}_{n\text{次}}$, 称之为词 w 的 n 次幂 (power).

Example: 《从一到无穷大》P.10.



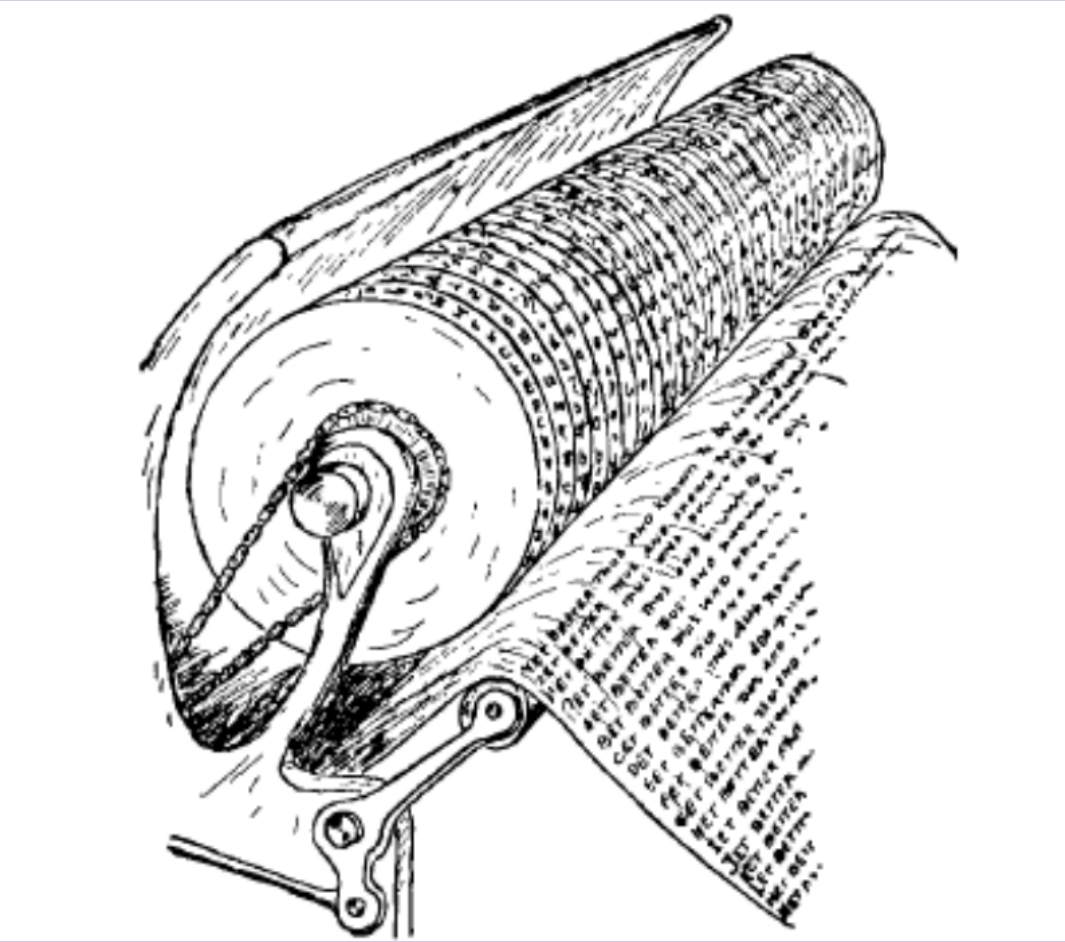
- horse has six legs and...
- i like apples cooked in terpentin...

Example: 《从一到无穷大》P.10.



- horse has six legs and...
- i like apples cooked in terpentin...

Example: 《从一到无穷大》P.10.



- horse has six legs and...
- i like apples cooked in terpentin...

©hfwang

Datatype of Σ^* in OCaml

```
# type 'a list  = (* construtors *)
  | Nil (* empty list *)
  | Cons of 'a * 'a list (* element type is parametric 'a*)
# let s = Cons ('a', Cons ('b', Cons('c', Nil)));;
val s : char list = Cons ('a', Cons ('b', Cons ('c', Nil)))
# let t = Cons ('d', Cons ('e', Nil));;
val s : char list = Cons ('d', Cons ('e', Nil))
# Cons (1, Cons (2, Cons(3, Nil)));;
- : int list = Cons (1, Cons (2, Cons (3, Nil)))
# let rec concat l l' = match l with
  | Nil -> l'
  | Cons (a, l1) -> Cons (a, concat l1 l');;
val concat : 'a list -> 'a list -> 'a list = <fun>
# concat s t;;
Cons ('a', Cons ('b', Cons ('c', Cons ('d', Cons ('e', Nil)))))
# (* construction by constructors *)
# (* destruction by constructors *)
```

©hfwang

Example: Natural numbers in Coq

```
$ coqtop
coq < Print nat.
 Inductive nat : Set :=  O : nat | S : nat -> nat
coq < Check S (S (S O)).
 3 : nat
coq < Unset Printing Notations.
coq < Check 3.
 S (S (S O)) : nat
coq <  Check 100.
 S (S (S (S (S (S (S (S (S (S (S (S (S (S (S (S (S
  (S (S (S (S (S (S (S (..)))))))))))))))))))))) : nat
```

©hfwang

相关定义

前缀 后缀 中缀

设 Σ^* 为字符串集合，$w = a_1a_2\cdots a_n \in \Sigma^*(a_i \in \Sigma)$，则：

- $\varepsilon, a_1, a_1a_2, \ldots, a_1a_2\ldots a_i, a_1a_2\ldots a_n$ 是词 w 的前缀 (prefix).
- $\varepsilon, a_n, a_{n-1}a_n, \ldots, a_ia_{i+1}\ldots a_n, a_1a_2\ldots wa_n$ 是词 w 的后缀 (suffix).
- $a_ia_{i+1}\ldots a_j$ $(1 \le i < j \le n)$ 是词 w 的中缀 (infix).

长度

设 $w \in \Sigma^*$，词 w 的长度 (Length)(记为 $|w|$) 递归定义如下：

相关定义

前缀 后缀 中缀

设 Σ^* 为字符串集合，$w = a_1a_2\cdots a_n \in \Sigma^*(a_i \in \Sigma)$，则：

- $\varepsilon, a_1, a_1a_2, \ldots, a_1a_2\ldots a_i, a_1a_2\ldots a_n$ 是词 w 的前缀 (prefix).
- $\varepsilon, a_n, a_{n-1}a_n, \ldots, a_ia_{i+1}\ldots a_n, a_1a_2\ldots wa_n$ 是词 w 的后缀 (suffix).
- $a_ia_{i+1}\ldots a_j$ $(1 \le i < j \le n)$ 是词 w 的中缀 (infix).

长度

设 $w \in \Sigma^*$，词 w 的长度 (Length)(记为 $|w|$) 递归定义如下：

相关定义

前缀 后缀 中缀

设 Σ^* 为字符串集合，$w = a_1a_2\cdots a_n \in \Sigma^*(a_i \in \Sigma)$，则：

- $\varepsilon, a_1, a_1a_2, \ldots, a_1a_2\ldots a_i, a_1a_2\ldots a_n$ 是词 w 的前缀 (prefix).
- $\varepsilon, a_n, a_{n-1}a_n, \ldots, a_ia_{i+1}\ldots a_n, a_1a_2\ldots wa_n$ 是词 w 的后缀 (suffix).
- $a_ia_{i+1}\ldots a_j$ $(1 \le i < j \le n)$ 是词 w 的中缀 (infix).

长度

设 $w \in \Sigma^*$，词 w 的长度 (Length)(记为 $|w|$) 递归定义如下：

$|\varepsilon| = 0$

如果 $|w|$ 已定义，则 $|wa| = |w| + 1$

自然有和谐集合的定义 $|w| = |w_1| + |w_2|$ 当 $w = w_1w_2$

相关定义

前缀 后缀 中缀

设 Σ^* 为字符串集合，$w = a_1a_2\cdots a_n \in \Sigma^*(a_i \in \Sigma)$，则：

- $\varepsilon, a_1, a_1a_2, \ldots, a_1a_2\ldots a_i, a_1a_2\ldots a_n$ 是词 w 的前缀 (prefix).
- $\varepsilon, a_n, a_{n-1}a_n, \ldots, a_ia_{i+1}\ldots a_n, a_1a_2\ldots wa_n$ 是词 w 的后缀 (suffix).
- $a_ia_{i+1}\ldots a_j$ $(1 \le i < j \le n)$ 是词 w 的中缀 (infix).

长度

设 $w \in \Sigma^*$，词 w 的长度 (Length)(记为 $|w|$) 递归定义如下：

1. $|\varepsilon| = 0$.
2. 如果 $|w|$ 已定义，则 $|wa| = |w| + 1$.
3. 除有有限步骤后成立的情况外，$|w| = |w_1| + |w_2|$.

相关定义

前缀 后缀 中缀

设 Σ^* 为字符串集合，$w = a_1a_2\cdots a_n \in \Sigma^*(a_i \in \Sigma)$，则：

- $\varepsilon, a_1, a_1a_2, \ldots, a_1a_2\ldots a_i, a_1a_2\ldots a_n$ 是词 w 的前缀 (prefix).
- $\varepsilon, a_n, a_{n-1}a_n, \ldots, a_ia_{i+1}\ldots a_n, a_1a_2\ldots wa_n$ 是词 w 的后缀 (suffix).
- $a_ia_{i+1}\ldots a_j$ $(1 \le i < j \le n)$ 是词 w 的中缀 (infix).

长度

设 $w \in \Sigma^*$，词 w 的长度 (Length)(记为 $|w|$) 递归定义如下：

1. $|\varepsilon| = 0$.
2. 如果 $|w|$ 已定义，则 $\forall a \in \Sigma, |aw| = |w| + 1$.
3. 由长度和链接运算的定义可知：$\forall u, v \in \Sigma^*, |uv| = |u| + |v|$.

©hfwang

相关定义

前缀 后缀 中缀

设 Σ^* 为字符串集合，$w = a_1a_2\cdots a_n \in \Sigma^*(a_i \in \Sigma)$，则：

- $\varepsilon, a_1, a_1a_2, \ldots, a_1a_2\ldots a_i, a_1a_2\ldots a_n$ 是词 w 的前缀 (prefix).
- $\varepsilon, a_n, a_{n-1}a_n, \ldots, a_ia_{i+1}\ldots a_n, a_1a_2\ldots wa_n$ 是词 w 的后缀 (suffix).
- $a_ia_{i+1}\ldots a_j$ $(1 \le i < j \le n)$ 是词 w 的中缀 (infix).

长度

设 $w \in \Sigma^*$，词 w 的长度 (Length)(记为 $|w|$) 递归定义如下：

1. $|\varepsilon| = 0$.
2. 如果 $|w|$ 已定义，则 $\forall a \in \Sigma, |aw| = |w| + 1$.
3. 由长度和链接运算的定义可知：$\forall u, v \in \Sigma^*, |uv| = |u| + |v|$.

相关定义

前缀 后缀 中缀

设 Σ^* 为字符串集合，$w = a_1a_2\cdots a_n \in \Sigma^*(a_i \in \Sigma)$，则：

- $\varepsilon, a_1, a_1a_2, \ldots, a_1a_2\ldots a_i, a_1a_2\ldots a_n$ 是词 w 的前缀 (prefix).
- $\varepsilon, a_n, a_{n-1}a_n, \ldots, a_ia_{i+1}\ldots a_n, a_1a_2\ldots wa_n$ 是词 w 的后缀 (suffix).
- $a_ia_{i+1}\ldots a_j$ $(1 \le i < j \le n)$ 是词 w 的中缀 (infix).

长度

设 $w \in \Sigma^*$，词 w 的长度 (Length)(记为 $|w|$) 递归定义如下：

1. $|\varepsilon| = 0$.
2. 如果 $|w|$ 已定义，则 $\forall a \in \Sigma, |aw| = |w| + 1$.
3. 由长度和链接运算的定义可知：$\forall u, v \in \Sigma^*, |uv| = |u| + |v|$.

©hfwang

相关定义

前缀 后缀 中缀

设 Σ^* 为字符串集合，$w = a_1a_2\cdots a_n \in \Sigma^*(a_i \in \Sigma)$，则：

- $\varepsilon, a_1, a_1a_2, \ldots, a_1a_2\ldots a_i, a_1a_2\ldots a_n$ 是词 w 的前缀 (prefix).
- $\varepsilon, a_n, a_{n-1}a_n, \ldots, a_ia_{i+1}\ldots a_n, a_1a_2\ldots wa_n$ 是词 w 的后缀 (suffix).
- $a_ia_{i+1}\ldots a_j$ $(1 \le i < j \le n)$ 是词 w 的中缀 (infix).

长度

设 $w \in \Sigma^*$，词 w 的长度 (Length)(记为 $|w|$) 递归定义如下：

1. $|\varepsilon| = 0$.
2. 如果 $|w|$ 已定义，则 $\forall a \in \Sigma, |aw| = |w| + 1$.
3. 由长度和链接运算的定义可知：$\forall u, v \in \Sigma^*, |uv| = |u| + |v|$.

©hfwang

语言

设 Σ^* 是字符串集合, $\forall L \subseteq \Sigma^*$ 称为 Σ^* 上的**语言**，在词法分析中称为**单词**.

Example

- NUM_OR_ID = {w | w 是以字母开始并以字母和数字组成的字符串}.
- PLUS = {+}

Token, Pattern and Lexeme 的数学定义

语言

设 Σ^* 是字符串集合, $\forall L \subseteq \Sigma^*$ 称为 Σ^* 上的语言，在词法分析中称为单词.

Example

- NUM_OR_ID = { w | w 是以字母开始并以字母和数字组成的字符串}.
- PLUS = {+}.

Token, Pattern and Lexeme 的数学定义

语言

设 Σ^* 是字符串集合, $\forall L \subseteq \Sigma^*$ 称为 Σ^* 上的语言，在词法分析中称为单词.

Example

- NUM_OR_ID = { w | w 是以字母开始并以字母和数字组成的字符串}.
- PLUS = {+}.

Token, Pattern and Lexeme 的数学定义

- 令 Σ 是字符集合
- 令 Δ 是对应于 token 的集合
- 令 Π 是模式集合
- 令 Λ 是字符串集合
- 令 $\mathrm{Pattern}: \Delta \to \Pi$
- 令 $\mathrm{Lexeme}: \Delta \to \Lambda$

语言

设 Σ^* 是字符串集合，$\forall L \subseteq \Sigma^*$ 称为 Σ^* 上的语言，在词法分析中称为单词.

Example

- NUM_OR_ID = { w | w 是以字母开始并以字母和数字组成的字符串}.
- PLUS = {+}.

Token, Pattern and Lexeme 的数学定义

- 单词：Σ^* 上的子集合.
- 模式：刻画子集合的描述 规则 [illegible]
- 单词：属于单词的一个字符串 [illegible]

语言

设 Σ^* 是字符串集合，$\forall L \subseteq \Sigma^*$ 称为 Σ^* 上的语言，在词法分析中称为单词.

Example

- NUM_OR_ID = { w | w 是以字母开始并以字母和数字组成的字符串}.
- PLUS = {+}.

Token, Pattern and Lexeme 的数学定义

- 单词：Σ^*上的子集合.
- 模式：对该子集合的描述：如，NUM_OR_ID.
- 词形：具体单词中一个元素，如：abc，如果单词中不止一个元素，词法分析器必须记录词形，以区别同种单词的不同词形.

语言

设 Σ^* 是字符串集合，$\forall L \subseteq \Sigma^*$ 称为 Σ^* 上的语言，在词法分析中称为单词.

Example

- NUM_OR_ID = { w | w 是以字母开始并以字母和数字组成的字符串}.
- PLUS = {+}.

Token, Pattern and Lexeme 的数学定义

- 单词：Σ^*上的子集合.
- 模式：对该子集合的描述：如，NUM_OR_ID.
- 词形：具体单词中一个元素，如：abc，如果单词中不止一个元素，词法分析器必须记录词形，以区别同种单词的不同词形.

语言

设 Σ^* 是字符串集合，$\forall L \subseteq \Sigma^*$ 称为 Σ^* 上的语言，在词法分析中称为单词.

Example

- NUM_OR_ID = { w | w 是以字母开始并以字母和数字组成的字符串}.
- PLUS = {+}.

Token, Pattern and Lexeme 的数学定义

- 单词：Σ^*上的子集合.
- 模式：对该子集合的描述：如，NUM_OR_ID.
- 词形：具体单词中一个元素，如：abc，如果单词中不止一个元素，词法分析器必须记录词形，以区别同种单词的不同词形.

语言

设 Σ^* 是字符串集合，$\forall L \subseteq \Sigma^*$ 称为 Σ^* 上的语言，在词法分析中称为单词.

Example

- NUM_OR_ID = { w | w 是以字母开始并以字母和数字组成的字符串}.
- PLUS = {+}.

Token, Pattern and Lexeme 的数学定义

- 单词：Σ^*上的子集合.
- 模式：对该子集合的描述：如，NUM_OR_ID.
- 词形：具体单词中一个元素，如：abc，如果单词中不止一个元素，词法分析器必须记录词形，以区别同种单词的不同词形.

问题

如何用形式化的方法定义单词

- 递归定义太广泛，不利于构造高效率的词法分析器. 词法规则比一般递归意义下的语言要简单很多.
- 自然语言：不利于机器自动识别词法规范.
- 寻求一个有足够表达能力，利于计算机识别，能够构造高效分析器的定义方法.

©hfwang

正则表达式的定义 (Regular Expression).

设 Σ^* 是字符串集合:

1. 归纳基础:
   1. $\forall a \in \Sigma$, a 是正则表达式.
   2. ε 是正则表达式.
   3. $\emptyset$ 是正则表达式.
2. 设 r 和 s 是正则表达式 (称为metavariable, 在构造中可以替代任何具体字符串), 则:
3. 由以上方法在有限步生成的字符串都是正则表达式.

正则表达式的定义 (Regular Expression).

设 Σ^* 是字符串集合:

1. 归纳基础:
   1. $\forall a \in \Sigma$, a 是正则表达式.
   2. ε 是正则表达式.
   3. $\emptyset$ 是正则表达式.
2. 设 r 和 s 是正则表达式 (称为metavariable, 在构造中可以替代任何具体字符串), 则:
3. 由以上方法在有限步生成的字符串都是正则表达式.

正则表达式的定义 (Regular Expression).

设 Σ^* 是字符串集合:

1. 归纳基础:
   1. $\forall a \in \Sigma$, a 是正则表达式.
   2. ε 是正则表达式.
   3. $\emptyset$ 是正则表达式.
2. 设 r 和 s 是正则表达式 (称为metavariable, 在构造中可以替代任何具体字符串), 则:
3. 由以上方法在有限步生成的字符串都是正则表达式.

正则表达式的定义 (Regular Expression).

设 Σ^* 是字符串集合:

1. 归纳基础:
   1. $\forall a \in \Sigma$, a 是正则表达式.
   2. ε 是正则表达式.
   3. $\emptyset$ 是正则表达式.
2. 设 r 和 s 是正则表达式 (称为metavariable, 在构造中可以替代任何具体字符串), 则:
   1. $(r|s)$ 是正则表达式 (或者 r 或 s)
   2. (rs) (简记为 rs) 是正则表达式 (先是 r 后是 s)
   3. $(r)^*$ 是正则表达式 (零个或多个 r 的连接)
3. 由以上方法在有限步生成的字符串都是正则表达式.

正则表达式的定义 (Regular Expression).

设 Σ^* 是字符串集合:

1. 归纳基础:
   1. $\forall a \in \Sigma$, a 是正则表达式.
   2. ε 是正则表达式.
   3. $\emptyset$ 是正则表达式.
2. 设 r 和 s 是正则表达式 (称为metavariable, 在构造中可以替代任何具体字符串), 则:
   1. $(r|s)$ 是正则表达式 (并运算, union).
   2. (rs) (简记为 rs) 是正则表达式 (连接运算, concatenation).
   3. $(r)^*$ 是正则表达式 (闭包运算, Kleene closure).
3. 由以上方法在有限步生成的字符串都是正则表达式.

正则表达式的定义 (Regular Expression).

设 Σ^* 是字符串集合:

1. 归纳基础:
   1. $\forall a \in \Sigma$, a 是正则表达式.
   2. ε 是正则表达式.
   3. $\emptyset$ 是正则表达式.
2. 设 r 和 s 是正则表达式 (称为metavariable, 在构造中可以替代任何具体字符串), 则:
   4. $(r \mid s)$ 是正则表达式 (并运算, union).
   5. $(r{\cdot}s)$ (简记 rs) 是正则表达式 (连接运算, concatenation).
   6. (r^*) 是正则表达式 (闭包运算, Kleene closure).
3. 由以上方法在有限步生成的字符串都是正则表达式.

©hfwang

正则表达式的定义 (Regular Expression).

设 Σ^* 是字符串集合:

1. 归纳基础:
   1. $\forall a \in \Sigma$, a 是正则表达式.
   2. ε 是正则表达式.
   3. $\emptyset$ 是正则表达式.
2. 设 r 和 s 是正则表达式 (称为metavariable, 在构造中可以替代任何具体字符串), 则:
   4. $(r \mid s)$ 是正则表达式 (并运算, union).
   5. $(r{\cdot}s)$ (简记 rs) 是正则表达式 (连接运算, concatenation).
   6. (r^*) 是正则表达式 (闭包运算, Kleene closure).
3. 由以上方法在有限步生成的字符串都是正则表达式.

正则表达式的定义 (Regular Expression).

设 Σ^* 是字符串集合:

1. 归纳基础:
   1. $\forall a \in \Sigma$, a 是正则表达式.
   2. ε 是正则表达式.
   3. $\emptyset$ 是正则表达式.
2. 设 r 和 s 是正则表达式 (称为metavariable, 在构造中可以替代任何具体字符串), 则:
   4. $(r \mid s)$ 是正则表达式 (并运算, union).
   5. $(r{\cdot}s)$ (简记 rs) 是正则表达式 (连接运算, concatenation).
   6. (r^*) 是正则表达式 (闭包运算, Kleene closure).
3. 由以上方法在有限步生成的字符串都是正则表达式.

正则表达式的定义 (Regular Expression).

设 Σ^* 是字符串集合:

1. 归纳基础:
   1. $\forall a \in \Sigma$, a 是正则表达式.
   2. ε 是正则表达式.
   3. $\emptyset$ 是正则表达式.
2. 设 r 和 s 是正则表达式 (称为metavariable, 在构造中可以替代任何具体字符串), 则:
   4. $(r \mid s)$ 是正则表达式 (并运算, union).
   5. $(r \cdot s)$ (简记 rs) 是正则表达式 (连接运算, concatenation).
   6. (r^*) 是正则表达式 (闭包运算, Kleene closure).
3. 由以上方法在有限步生成的字符串都是正则表达式.

正则表达式的定义 (Regular Expression).

设 Σ^* 是字符串集合:

1. 归纳基础:
   1. $\forall a \in \Sigma$, a 是正则表达式.
   2. ε 是正则表达式.
   3. $\emptyset$ 是正则表达式.
2. 设 r 和 s 是正则表达式 (称为metavariable, 在构造中可以替代任何具体字符串), 则:
   4. $(r \mid s)$ 是正则表达式 (并运算, union).
   5. $(r{\cdot}s)$ (简记 rs) 是正则表达式 (连接运算, concatenation).
   6. (r^*) 是正则表达式 (闭包运算, Kleene closure).
3. 由以上方法在有限步生成的字符串都是正则表达式.

Example

((a | (bb))*) 是正则表达式:

1. a, b 是正则表达式, by Def ①
2. (bb)是正则表达式, by Def ⑤ and ①,
   此时, 定义式中的元变量 r 和 s 均替换为具体正则表达式 b.
3. (a | (bb))正则表达式, by Def ④ and ②,
   此时, 定义式中的元变量 r 和 s 分别替换为具体正则表达式 a 和 (bb).
4. ((a | (bb))*) 正则表达式, by ⑥ and ③
   此时, 定义式中的元变量 r替换为具体正则表达式 (a | (bb)).

©hfwang

Example

((a | (bb))*) 是正则表达式:

1. a, b 是正则表达式, by Def ①
2. (bb)是正则表达式, by Def ⑤ and ①,
   此时, 定义式中的元变量 r 和 s 均替换为具体正则表达式 b.
3. (a | (bb))正则表达式, by Def ④ and ②,
   此时, 定义式中的元变量 r 和 s 分别替换为具体正则表达式 a 和 (bb).
4. ((a | (bb))*) 正则表达式, by ⑥ and ③
   此时, 定义式中的元变量 r替换为具体正则表达式 (a | (bb)).

©hfwang

Example

((a | (bb))*) 是正则表达式:

1. a, b 是正则表达式, by Def ①
2. (bb)是正则表达式, by Def ⑤ and ①,
   此时, 定义式中的元变量 r 和 s 均替换为具体正则表达式 b.
3. (a | (bb))正则表达式, by Def ④ and ②,
   此时, 定义式中的元变量 r 和 s 分别替换为具体正则表达式 a 和 (bb).
4. ((a | (bb))*) 正则表达式, by ⑥ and ③
   此时, 定义式中的元变量 r替换为具体正则表达式 (a | (bb)).

©hfwang

Example

((a | (bb))*) 是正则表达式:

1. a, b 是正则表达式, by Def ①
2. (bb)是正则表达式, by Def ⑤ and ①,
   此时, 定义式中的元变量 r 和 s 均替换为具体正则表达式 b.
3. (a | (bb))正则表达式, by Def ④ and ②,
   此时, 定义式中的元变量 r 和 s 分别替换为具体正则表达式 a 和 (bb).
4. ((a | (bb))*) 正则表达式, by ⑥ and ③
   此时, 定义式中的元变量 r替换为具体正则表达式 (a | (bb)).

Example

((a | (bb))*) 是正则表达式:

1. a, b 是正则表达式, by Def ①
2. (bb)是正则表达式, by Def ⑤ and ①,
   此时, 定义式中的元变量 r 和 s 均替换为具体正则表达式 b.
3. (a | (bb))正则表达式, by Def ④ and ②,
   此时, 定义式中的元变量 r 和 s 分别替换为具体正则表达式 a 和 (bb).
4. ((a | (bb))*) 正则表达式, by ⑥ and ③
   此时, 定义式中的元变量 r替换为具体正则表达式 (a | (bb)).

©hfwang

Datatype of Regular Expression in OCaml

```
# type  'a regexpr =
  | Epsilon
  | Symbol of 'a  (* 'a is parametric type of alphabet *)
  | Star of 'a regexpr
  | Or of 'a regexpr * 'a regexpr
  | Seq of 'a regexpr * 'a regexpr;;
# let r = Star (Or ((Symbol 'a'), Seq (Symbol 'b', Symbol 'b')));;
val r : char regexpr = Star (Or (Symbol 'a',
   Seq (Symbol 'b', Symbol 'b')))
# let rec reg2string r = match r with
  | Epsilon -> print_string "ep"
  | Symbol x -> print_char x
  | Star r1 -> print_char '('; reg2string r1; print_string "*)"
  | Or (r1, r2) -> print_char '('; reg2string r1;
      print_char '|'; reg2string r2; print_char ')'
  | Seq (r1, r2) ->  print_char '('; reg2string r1;
      reg2string r2;print_char ')';;
val reg2string : char regexpr -> unit = <fun>
# reg2string r;;
((a|(bb))*)- : unit = ()
```

©hfwang

正则表达式的语义

设 r 是 Σ^* 上的正则表达式，则 r 的语义，记为：$L(r)$ 递归定义如下：

1. 归纳基础：
   1. $\forall a \in \Sigma, L(a) = \{a\}$.
   2. $L(\varepsilon) = \{\varepsilon\}$
   3. $L(\emptyset) = \emptyset$
2. 设 r 和 s 是正则表达式，则：

L 可以看成 Σ 上的所有的正则表达式集合到 2^{Σ^*} 上的函数（字符串集合的幂集合）！

正则表达式的语义

设 r 是 Σ^* 上的正则表达式，则 r 的语义，记为：$L(r)$ 递归定义如下：

1. 归纳基础：
   1. $\forall a \in \Sigma, L(a) = \{a\}$.
   2. $L(\varepsilon) = \{\varepsilon\}$.
   3. $L(\emptyset) = \emptyset$.
2. 设 r 和 s 是正则表达式，则：

L 可以看成 Σ 上的所有的正则表达式集合到 2^{Σ^*} 上的函数（字符串集合的幂集合）！

正则表达式的语义

设 r 是 Σ^* 上的正则表达式，则 r 的语义，记为：$L(r)$ 递归定义如下：

1. 归纳基础：
   1. $\forall a \in \Sigma, L(a) = \{a\}$.
   2. $L(\varepsilon) = \{\varepsilon\}$.
   3. $L(\emptyset) = \emptyset$.
2. 设 r 和 s 是正则表达式，则：

L 可以看成 Σ 上的所有的正则表达式集合到 2^{Σ^*} 上的函数（字符串集合的幂集合）！

正则表达式的语义

设 r 是 Σ^* 上的正则表达式，则 r 的语义，记为：$L(r)$ 递归定义如下：

1. 归纳基础：
   1. $\forall a \in \Sigma, L(a) = \{a\}$.
   2. $L(\varepsilon) = \{\varepsilon\}$.
   3. $L(\emptyset) = \emptyset$.
2. 设 r 和 s 是正则表达式，则：
   1. $L(r \mid s) = L(r) \cup L(s)$
   2. $L(rs) = \{w_1w_2 \mid w_1 \in L(r) \wedge w_2 \in L(s)\} = L(r)L(s)$
   3. $L(r^*) = \bigcup_{n=0}^{\infty} L(r)^n$

   注：$L(r)^0 = \{\varepsilon\}$，$L(r)^n = L(r)^{n-1}L(r)$，$n \geq 1$

L 可以看成 Σ 上的所有的正则表达式集合到 2^{Σ^*} 上的函数（字符串集合的幂集合）！

正则表达式的语义

设 r 是 Σ^* 上的正则表达式，则 r 的语义，记为：$L(r)$ 递归定义如下：

1. 归纳基础：
   1. $\forall a \in \Sigma, L(a) = \{a\}$.
   2. $L(\varepsilon) = \{\varepsilon\}$.
   3. $L(\emptyset) = \emptyset$.
2. 设 r 和 s 是正则表达式，则：
   1. $L(r \mid s) = L(r) \cup L(s)$.
   2. $L(rs) = \{w_1 w_2 \mid w_1 \in L(r) \wedge w_2 \in L(s)\} = L(r)L(s)$.
   3. $L(r^*) = \bigcup_{n=0}^{\infty} L(r)^n$

   注：$L(r)^0 = \{\varepsilon\}$，$L(r)^n = L(r)^{n-1}L(r)$，$n \geq 1$.

L 可以看成 Σ 上的所有的正则表达式集合到 2^{Σ^*} 上的函数（字符串集合的幂集合）！

正则表达式的语义

设 r 是 Σ^* 上的正则表达式，则 r 的语义，记为：$L(r)$ 递归定义如下：

1. 归纳基础：
   1. $\forall a \in \Sigma, L(a) = \{a\}$.
   2. $L(\varepsilon) = \{\varepsilon\}$.
   3. $L(\emptyset) = \emptyset$.
2. 设 r 和 s 是正则表达式，则：
   4. $L((r \mid s)) = L(r) \bigcup L(s)$.
   5. $L((r \cdot s)) = \{w_r \cdot w_s \mid w_r \in L(r) \wedge w_s \in L(s)\} \triangleq L(r) \cdot L(s)$.
   6. $L((r^*)) = \bigcup_{n=0}^{\infty} (L(r))^n$.

   注：$L(r)^0 \triangleq \{\varepsilon\}$; $(L(r))^n \triangleq (L(r))^{n-1} \cdot L(r)$ $(n \geq 1)$.

L 可以看成 Σ 上的所有的正则表达式集合到 2^{Σ^*} 上的函数（字符串集合的幂集合）！

正则表达式的语义

设 r 是 Σ^* 上的正则表达式，则 r 的语义，记为：$L(r)$ 递归定义如下：

1. 归纳基础：
   1. $\forall a \in \Sigma, L(a) = \{a\}$.
   2. $L(\varepsilon) = \{\varepsilon\}$.
   3. $L(\emptyset) = \emptyset$.
2. 设 r 和 s 是正则表达式，则：
   4. $L((r \mid s)) = L(r) \bigcup L(s)$.
   5. $L((r \cdot s)) = \{w_r \cdot w_s \mid w_r \in L(r) \wedge w_s \in L(s)\} \triangleq L(r) \cdot L(s)$.
   6. $L((r^*)) = \bigcup_{n=0}^{\infty} (L(r))^n$.

   注：$L(r)^0 \triangleq \{\varepsilon\}$; $(L(r))^n \triangleq (L(r))^{n-1} \cdot L(r)$ $(n \geq 1)$.

L 可以看成 Σ 上的所有的正则表达式集合到 2^{Σ^*} 上的函数（字符串集合的幂集合）！

正则表达式的语义

设 r 是 Σ^* 上的正则表达式，则 r 的语义，记为：$L(r)$ 递归定义如下：

1. 归纳基础：
   1. $\forall a \in \Sigma, L(a) = \{a\}$.
   2. $L(\varepsilon) = \{\varepsilon\}$.
   3. $L(\emptyset) = \emptyset$.
2. 设 r 和 s 是正则表达式，则：
   4. $L((r \mid s)) = L(r) \bigcup L(s)$.
   5. $L((r \cdot s)) = \{w_r \cdot w_s \mid w_r \in L(r) \bigwedge w_s \in L(s)\} \triangleq L(r) \cdot L(s)$.
   6. $L((r^*)) = \bigcup_{n=0}^{\infty} (L(r))^n$.

   注：$L(r)^0 \triangleq \{\varepsilon\}$; $(L(r))^n \triangleq (L(r))^{n-1} \cdot L(r)$ $(n \geq 1)$.

L 可以看成 Σ 上的所有的正则表达式集合到 2^{Σ^*} 上的函数（字符串集合的幂集合）！

正则表达式的语义

设 r 是 Σ^* 上的正则表达式，则 r 的语义，记为：$L(r)$ 递归定义如下：

1. 归纳基础：
   1. $\forall a \in \Sigma, L(a) = \{a\}$.
   2. $L(\varepsilon) = \{\varepsilon\}$.
   3. $L(\emptyset) = \emptyset$.
2. 设 r 和 s 是正则表达式，则：
   4. $L((r \mid s)) = L(r) \bigcup L(s)$.
   5. $L((r \cdot s)) = \{w_r \cdot w_s \mid w_r \in L(r) \bigwedge w_s \in L(s)\} \triangleq L(r) \cdot L(s)$.
   6. $L((r^*)) = \bigcup_{n=0}^{\infty} (L(r))^n$.

      注：$L(r)^0 \triangleq \{\varepsilon\}$; $(L(r))^n \triangleq (L(r))^{n-1} \cdot L(r)$ $(n \geq 1)$.

L 可以看成 Σ 上的所有的正则表达式集合到 2^{Σ^*} 上的函数（字符串集合的幂集合）！

正则表达式的语义

设 r 是 Σ^* 上的正则表达式，则 r 的语义，记为：$L(r)$ 递归定义如下：

1. 归纳基础：
   1. $\forall a \in \Sigma, L(a) = \{a\}$.
   2. $L(\varepsilon) = \{\varepsilon\}$.
   3. $L(\emptyset) = \emptyset$.
2. 设 r 和 s 是正则表达式，则：
   4. $L((r \mid s)) = L(r) \bigcup L(s)$.
   5. $L((r \cdot s)) = \{w_r \cdot w_s \mid w_r \in L(r) \wedge w_s \in L(s)\} \triangleq L(r) \cdot L(s)$.
   6. $L((r^*)) = \bigcup_{n=0}^{\infty} (L(r))^n$.

      注：$L(r)^0 \triangleq \{\varepsilon\}$; $(L(r))^n \triangleq (L(r))^{n-1} \cdot L(r)$ $(n \geq 1)$.

L 可以看成 Σ 上的所有的正则表达式集合到 2^{Σ^*} 上的函数（字符串集合的幂集合）！

Example

$((a \mid (bb))^*)$ 表示的语言集合:

$$
\begin{aligned}
& L(((a \mid (bb))^*)) \\
= & \bigcup_{n=0}^{\infty} (L((a \mid (bb))))^n \\
= & \bigcup_{n=0}^{\infty} \{a, bb\}^n \\
= & \{\varepsilon, a, bb, abb, bba, aabb, \ldots\} \\
= & \{\text{character string composed by a and bb}\}
\end{aligned}
$$

©hfwang

约定

为了方便书写，特约定正则表达式的优先级别如下：

$$\text{high} \xrightarrow{a \quad * \quad \cdot \quad |} \text{low}$$

Example

$$((a \mid (bb))^*) \equiv (a \mid bb)^* \neq a \mid bb^* \equiv (a \mid (b(b^*)))$$

定义

设 r 和 s 是正则表达式，称 r 和 s 等价，记为：$r = s$ iff $L(r) = L(s)$.

Example

$$a \mid b = b \mid a,\ ab \neq ba,\ a^{**} = a^*.$$

约定

为了方便书写，特约定正则表达式的优先级别如下：

$$\text{high} \xrightarrow{a \quad * \quad \cdot \quad |} \text{low}$$

Example

$$((a \mid (bb))^*) \equiv (a \mid bb)^* \neq a \mid bb^* \equiv (a \mid (b(b^*)))$$

定义

设 r 和 s 是正则表达式，称 r 和 s 等价，记为：r = s iff L(r) = L(s).

Example

$$a \mid b = b \mid a,\ ab \neq ba,\ a^{**} = a^*.$$

约定

为了方便书写，特约定正则表达式的优先级别如下：

$$\text{high} \xrightarrow{a \quad * \quad \cdot \quad |} \text{low}$$

Example

$$((a \mid (bb))^*) \equiv (a \mid bb)^* \neq a \mid bb^* \equiv (a \mid (b(b^*)))$$

定义

设 r 和 s 是正则表达式，称 r 和 s 等价，记为：$r = s$ iff $L(r) = L(s)$.

Example

$$a \mid b = b \mid a,\ ab \neq ba,\ a^{**} = a^*.$$

约定

为了方便书写，特约定正则表达式的优先级别如下：

$$\text{high} \xrightarrow{a \quad * \quad \cdot \quad |} \text{low}$$

Example

$$((a \mid (bb))^*) \equiv (a \mid bb)^* \neq a \mid bb^* \equiv (a \mid (b(b^*)))$$

定义

设 r 和 s 是正则表达式，称 r 和 s 等价，记为：$r = s$ iff $L(r) = L(s)$.

Example

$$a \mid b = b \mid a,\ ab \neq ba,\ a^{**} = a^*.$$

©hfwang

正则表达式的设计

设计方法

- 有限集合：枚举法，如：

keyword = if | else | switch; L(keyword) = {if, else, switch}

- 无限集合：Kleene Closure 的使用，如：

L(ID) = {以字母和下划线开始，以字母、下划线和数字组成的字符串集合}

则

$$
\begin{aligned}
\text{digit} &= 0 \mid 1 \mid \cdots \mid 9 \\
\text{letter} &= a \mid b \mid \cdots \mid z \mid A \mid B \mid \cdots \mid Z \\
\text{ID} &= (\text{letter} \mid _)(\text{letter} \mid \text{digit} \mid _)^*
\end{aligned}
$$

©hfwang

正则表达式的设计

设计方法

- 有限集合：枚举法，如：

keyword = if | else | switch; L(keyword) = {if, else, switch}

- 无限集合：Kleene Closure 的使用，如：

L(ID) = {以字母和下划线开始，以字母、下划线和数字组成的字符串集合}

则

$$
\begin{aligned}
\text{digit} &= 0 \mid 1 \mid \cdots \mid 9 \\
\text{letter} &= a \mid b \mid \cdots \mid z \mid A \mid B \mid \cdots \mid Z \\
\text{ID} &= (\text{letter} \mid _)(\text{letter} \mid \text{digit} \mid _)^*
\end{aligned}
$$

正则表达式的设计

设计方法

- 有限集合：枚举法，如：

keyword = if | else | switch; L(keyword) = {if, else, switch}

- 无限集合：Kleene Closure 的使用，如：

L(ID) = {以字母和下划线开始，以字母、下划线和数字组成的字符串集合}

则

$$
\begin{aligned}
\text{digit} &= 0 \mid 1 \mid \cdots \mid 9 \\
\text{letter} &= a \mid b \mid \cdots \mid z \mid A \mid B \mid \cdots \mid Z \\
\text{ID} &= (\text{letter} \mid _)(\text{letter} \mid \text{digit} \mid _)^*
\end{aligned}
$$

©hfwang

Example 1/6

L(r) = {以字母 a, b 和 c 字符串，但是该串中 b 字母出现且仅出现一次}

解

$$r = (a \mid c)^* b (a \mid c)^*$$
$$L(r) = L$$

©hfwang

Example 1/6

L(r) = {以字母 a, b 和 c 字符串，但是该串中 b 字母出现且仅出现一次}

解

$$r = (a \mid c)^* b (a \mid c)^*$$

$$L(r) = L$$

Example 2/6 (解法 1)

$L(r)$ = {以字母 a, b 和 c 字符串，但是该串中任意两个 b 都不相邻}

解

分析串的结构：

$$\underbrace{\underbrace{y}_{y}\underbrace{\underbrace{x\cdots x}_{A}b\underbrace{x\cdots x}_{A}b\cdots\underbrace{x\cdots x}_{A}b}_{A^*}\underbrace{x\cdots x}_{B}}_{yA^*B}$$

其中：x 表示非 b 字母，y 或者是 ε 或者是 b：

$$x\cdots x = (a \mid c)(a \mid c)^* \triangleq (a \mid c)^+$$

$$A = (a \mid c)^+ b$$

$$r = (b \mid \varepsilon)((a \mid c)^+ b)^*(a \mid c)^*$$

Example 2/6 (解法 1)

$L(r)$ = {以字母 a, b 和 c 字符串，但是该串中任意两个 b 都不相邻}

解

分析串的结构：

$$\underbrace{\underbrace{\underbrace{y}_{y}}_{y}\ \underbrace{\underbrace{x\cdots x}_{A}\underbrace{b\,x\cdots x}_{A}\cdots\underbrace{x\cdots x}_{A}\underbrace{b}_{A}}_{A^*}\ \underbrace{\underbrace{x\cdots x}_{B}}_{B}}_{yA^*B}$$

其中：x 表示非 b 字母，y 或者是 ε 或者是 b：

$$x\cdots x = (a \mid c)(a \mid c)^* \triangleq (a \mid c)^+$$

$$A = (a \mid c)^+ b$$

$$r = (b \mid \varepsilon)((a \mid c)^+ b)^*(a \mid c)^*$$

Example 2/6 (解法 1)

$L(r)$ = {以字母 a, b 和 c 字符串，但是该串中任意两个 b 都不相邻}

解

分析串的结构：

$$\underbrace{\underbrace{\underbrace{y}_{y}}_{y}\ \underbrace{\underbrace{x\cdots x\,b}_{A}\ \underbrace{x\cdots x\,b}_{A}\cdots\underbrace{x\cdots x\,b}_{A}}_{A^*}\ \underbrace{\underbrace{x\cdots x}_{B}}_{B}}_{yA^*B}$$

其中：x 表示非 b 字母，y 或者是 ε 或者是 b:

$$x\cdots x = (a \mid c)(a \mid c)^* \triangleq (a \mid c)^+$$

$$A = (a \mid c)^+ b$$

$$r = (b \mid \varepsilon)((a \mid c)^+ b)^*(a \mid c)^*$$

Example 2/6 (解法 2)

L(r) = {以字母 a, b 和 c 字符串，但是该串中任意两个 b 都不相邻}

解

分析串的结构：

$$\underbrace{\underbrace{\underbrace{y}_{y}}_{y}\ \underbrace{\underbrace{x}_{x}\cdots\underbrace{x}_{x}\ \underbrace{xb}_{R}\cdots\underbrace{x}_{x}\cdots\underbrace{x}_{x}\ \underbrace{xb}_{R}\ \underbrace{x}_{x}\cdots\underbrace{x}_{x}}_{(x|R)^*}}$$

$x = (a \mid c)$

$R = (a \mid c)b = ab \mid cb$

$r = (b \mid \varepsilon)(a \mid c \mid ab \mid cb)^*$

Example 2/6 (解法 2)

$L(r)$ = {以字母 a, b 和 c 字符串，但是该串中任意两个 b 都不相邻}

解

分析串的结构：

$$\underbrace{\underbrace{y}_{y}}_{y}\ \underbrace{\underbrace{x}_{x}\cdots\underbrace{x}_{x}\ \underbrace{xb}_{R}\cdots\underbrace{x}_{x}\cdots\underbrace{x}_{x}\ \underbrace{xb}_{R}\ \underbrace{x}_{x}\cdots\underbrace{x}_{x}}_{(x|R)^*}$$

$x = (a \mid c)$

$R = (a \mid c)b = ab \mid cb$

$r = (b \mid \varepsilon)(a \mid c \mid ab \mid cb)^*$

Example 2/6 (解法 2)

L(r) = {以字母 a, b 和 c 字符串，但是该串中任意两个 b 都不相邻}

解

分析串的结构：

$$\underbrace{\underbrace{y}_{y}}_{y}\ \underbrace{\underbrace{x}_{x}\cdots\underbrace{x}_{x}\ \underbrace{xb}_{R}\cdots\underbrace{x}_{x}\cdots\underbrace{x}_{x}\ \underbrace{xb}_{R}\ \underbrace{x}_{x}\cdots\underbrace{x}_{x}}_{(x|R)^*}$$

$$x = (a \mid c)$$

$$R = (a \mid c)b = ab \mid cb$$

$$r = (b \mid \varepsilon)(a \mid c \mid ab \mid cb)^*$$

Example 3/6

$L(r)$ = {以字母 a, b 和 c 字符串, 无 ab 字串}

解

分析串的结构：

$$\underbrace{\underbrace{\underbrace{bb\cdots b}_{b^*}\underbrace{xx\cdots x}_{x^*}\underbrace{cbb\cdots b}_{y}}_{z}\underbrace{\cdots\underbrace{xx\cdots x}_{x^*}\underbrace{cbb\cdots b}_{y}\underbrace{xx\cdots x}_{x^*}}_{z}}_{(x|z)^*}$$

$$x = (a \mid c)$$

$$z = cb^*$$

$$r = b^*(a \mid c \mid cb^*)^*(a \mid c)^*$$

$$= b^*(a \mid c \mid cb^*)^*$$

Example 3/6

$L(r)$ = {以字母 a, b 和 c 字符串，无 ab 字串}

解

分析串的结构：

$$\underbrace{bb\cdots b}_{b^*}\underbrace{\underbrace{x\,x\cdots x}_{x^*}\,c\,\underbrace{\underbrace{bb\cdots b}_{y}}_{z}\cdots\underbrace{x\,x\cdots x}_{x^*}\,c\,\underbrace{\underbrace{bb\cdots b}_{y}}_{z}}_{(x|z)^*}\underbrace{x\,x\cdots x}_{x^*}$$

$$\begin{aligned} x &= (a \mid c) \\ z &= cb^* \\ r &= b^*(a \mid c \mid cb^*)^*(a \mid c)^* \\ &= b^*(a \mid c \mid cb^*)^* \end{aligned}$$

Example 3/6

$L(r)$ = {以字母 a, b 和 c 字符串，无 ab 字串}

解

分析串的结构：

$$\underbrace{bb\cdots b}_{b^*}\underbrace{\underbrace{x\,x\cdots x}_{x^*}\,c\,\underbrace{\underbrace{bb\cdots b}_{y}}_{z}\cdots\underbrace{x\,x\cdots x}_{x^*}\,c\,\underbrace{\underbrace{bb\cdots b}_{y}}_{z}}_{(x|z)^*}\underbrace{x\,x\cdots x}_{x^*}$$

$$\begin{aligned} x &= (a \mid c) \\ z &= cb^* \\ r &= b^*(a \mid c \mid cb^*)^*(a \mid c)^* \\ &= b^*(a \mid c \mid cb^*)^* \end{aligned}$$

Example 4/6

$L(r)$ = {以字母 a 和 b 字符串，其中 a 出现偶数次}

解

分析串的结构：

$$\underbrace{\underbrace{x\cdots xa}_{x^*a}\underbrace{x\cdots xa}_{x^*a}\cdots\underbrace{x\cdots xa}_{x^*a}\underbrace{x\cdots xa}_{x^*a}}_{((x^*a)(x^*a))^*x^*}\underbrace{x\cdots x}_{x^*}$$

$$x = (b \mid \varepsilon)$$

$$x^* = b^*$$

$$r = ((b^*a)(b^*a))^*b^*$$

Example 4/6

L(r) = {以字母 a 和 b 字符串，其中 a 出现偶数次}

解

分析串的结构：

$$\underbrace{\underbrace{x\cdots xa}_{x^*a}\underbrace{x\cdots xa}_{x^*a}\cdots\underbrace{x\cdots xa}_{x^*a}\underbrace{x\cdots xa}_{x^*a}\underbrace{x\cdots x}_{x^*}}_{((x^*a)(x^*a))^*x^*}$$

$$x = (b \mid \varepsilon)$$

$$x^* = b^*$$

$$r = ((b^*a)(b^*a))^*b^*$$

Example 4/6

L(r) = {以字母 a 和 b 字符串，其中 a 出现偶数次}

解

分析串的结构：

$$\underbrace{\underbrace{x\cdots xa}_{x^*a}\underbrace{x\cdots xa}_{x^*a}\cdots\underbrace{x\cdots xa}_{x^*a}\underbrace{x\cdots xa}_{x^*a}\underbrace{x\cdots x}_{x^*}}_{((x^*a)(x^*a))^*x^*}$$

$$x = (b \mid \varepsilon)$$

$$x^* = b^*$$

$$r = ((b^*a)(b^*a))^*b^*$$

Example 5/6

$\mathcal{L}(r) = \{s \mid s \in \{0,1\}^* \wedge s \text{ has even number of occurrences of } 011\}$

Solution

1. Suppose s is regular expression which has no substring 011, so the string of $\mathcal{L}(r)$ will be

$$\underbrace{\underbrace{s011s011\cdots\cdots s011s011}_{\text{even } 011}s}_{(s011s011)^*s}$$

2. like no ab's in $a, b \in \{0,1\}^*$
3. be careful that the concatenation of rr and rr will not generate the new substring 011

Example 5/6

$\mathcal{L}(r) = \{s \mid s \in \{0,1\}^* \wedge s \text{ has even number of occurrences of } 011\}$

Solution

1. Suppose s is regular expression which has no substring 011. so the string of $\mathcal{L}(r)$ will be
$$\underbrace{\underbrace{s011s011 \cdots\cdots s011s011}_{\text{even } 011}s}_{(s011s011)^*s}$$
2. like no ab's re, $s = 1^*(0 \mid 01)^*$.
3. be careful that the concatenation of rr and rsr will not generate the new substring 011.

Example 5/6

$\mathcal{L}(r) = \{s \mid s \in \{0,1\}^* \wedge s \text{ has even number of occurrences of } 011\}$

Solution

1. Suppose s is regular expression which has no substring 011. so the string of $\mathcal{L}(r)$ will be

$$\underbrace{s011s011 \cdots\cdots s011s011s}_{\underbrace{\text{even } 011}_{(s011s011)^*s}}$$

2. like no ab's re, $s = 1^*(0 \mid 01)^*$.
3. be careful that the concatenation of rr and rsr will not generate the new substring 011.

©hfwang

Example 5/6

$\mathcal{L}(r) = \{s \mid s \in \{0,1\}^* \wedge s \text{ has even number of occurrences of } 011\}$

Solution

1. Suppose s is regular expression which has no substring 011. so the string of $\mathcal{L}(r)$ will be
$$\underbrace{s011s011 \cdots\cdots s011s011s}_{\underbrace{\text{even } 011}_{(s011s011)^*s}}$$
2. like no ab's re, $s = 1^*(0 \mid 01)^*$.
3. be careful that the concatenation of rr and rsr will not generate the new substring 011.

Example 5/6

$\mathcal{L}(r) = \{s \mid s \in \{0,1\}^* \wedge s \text{ has even number of occurrences of } 011\}$

Solution

1. Suppose s is regular expression which has no substring 011. so the string of $\mathcal{L}(r)$ will be
$$\underbrace{s011s011 \cdots\cdots s011s011s}_{\underbrace{\text{even } 011}_{(s011s011)^*s}}$$
2. like no ab's re, $s = 1^*(0 \mid 01)^*$.
3. be careful that the concatenation of rr and rsr will not generate the new substring 011.

©hfwang

Example 6/6

Example

$$L(r) = \{a^n b^n \mid n \in \mathbf{N}\}$$

解

不能用正则表达式表示，由于 $L(r)$ 是无限集合，所以 r 中一定有 Kleene closure 运算，设：

$$r = r_1 s^* r_2$$

1. 设 $ab \in L(s)$，则：$\{ab, abab, \cdots\} \subseteq (L(s))^*$，而 $abab \notin L(r)$。
2. 设 $a \in L(s)$，则 $\forall y, z$ 使 $yaz \in L(r)$ 个数 y 与 z 中的 a 数相等，所以 $yaaz \notin L(r)$。

Example 6/6

Example

$$L(r) = \{a^n b^n \mid n \in \mathbf{N}\}$$

解

不能用正则表达式表示，由于 $L(r)$ 是无限集合，所以 r 中一定有 Kleene closure 运算，设：

$$r = r_1 s^* r_2$$

1. 设 $ab \in L(s)$，则：$\{ab, abab, \cdots\} \subseteq (L(s))^*$，而 $abab \notin L(r)$.
2. 设 $a \in L(s)$，则：$\forall w_1 = a^p \in L(r_1) \wedge w_2 = b^q \in L(r_2)$，使得：$\forall i \in \mathbf{N}, w_1 a^i w_2 = a^{p+i} b^q \notin L(r)$.

Example 6/6

Example

$$L(r) = \{a^n b^n \mid n \in \mathbf{N}\}$$

解

不能用正则表达式表示，由于 $L(r)$ 是无限集合，所以 r 中一定有 Kleene closure 运算，设：

$$r = r_1 s^* r_2$$

1. 设 $ab \in L(s)$，则：$\{ab, abab, \cdots\} \subseteq (L(s))^*$，而 $abab \notin L(r)$.
2. 设 $a \in L(s)$，则：$\forall w_1 = a^p \in L(r_1) \wedge w_2 = b^q \in L(r_2)$，使得：$\forall i \in \mathbf{N}, w_1 a^i w_2 = a^{p+i} b^q \notin L(r)$.

Example 6/6

Example

$$L(r) = \{a^n b^n \mid n \in \mathbf{N}\}$$

解

不能用正则表达式表示，由于 $L(r)$ 是无限集合，所以 r 中一定有 Kleene closure 运算，设：

$$r = r_1 s^* r_2$$

1. 设 $ab \in L(s)$，则：$\{ab, abab, \cdots\} \subseteq (L(s))^*$，而 $abab \notin L(r)$.
2. 设 $a \in L(s)$，则：$\forall w_1 = a^p \in L(r_1) \wedge w_2 = b^q \in L(r_2)$，使得：$\forall i \in \mathbf{N}, w_1 a^i w_2 = a^{p+i} b^q \notin L(r)$.

小节

- 仅能表达可递归定义的语言集合中的一部分.
- 能用正则表达式定义的 Σ^* 上的子集合称为正则语言.
- 从 Kleene Closure 的定义可看出，它使得正则语言中的某个子串能够独立于其他的部分重复 0 次或 n 多次后还属于该正则语言 (泵引理).
- 如果语言在递归构造中有两个部分相关联，则不能用正则表达式描述. 上例表明正则表达式不能描述在程序语言中嵌套的符号对，如：(和)，注释嵌套等.
- 正则表达式足以描述程序语言中的词法.
- 正则表达式在程序设计中的应用：请参考

小节

- 仅能表达可递归定义的语言集合中的一部分.
- 能用正则表达式定义的 Σ^* 上的子集合称为正则语言.
- 从 Kleene Closure 的定义可看出，它使得正则语言中的某个子串能够独立于其他的部分重复 0 次或 n 多次后还属于该正则语言 (泵引理).
- 如果语言在递归构造中有两个部分相关联，则不能用正则表达式描述. 上例表明正则表达式不能描述在程序语言中嵌套的符号对，如：(和)，注释嵌套等.
- 正则表达式足以描述程序语言中的词法.
- 正则表达式在程序设计中的应用：请参考

小节

- 仅能表达可递归定义的语言集合中的一部分.
- 能用正则表达式定义的 Σ^* 上的子集合称为正则语言.
- 从 Kleene Closure 的定义可看出，它使得正则语言中的某个子串能够独立于其他的部分重复 0 次或 n 多次后还属于该正则语言 (泵引理).
- 如果语言在递归构造中有两个部分相关联，则不能用正则表达式描述. 上例表明正则表达式不能描述在程序语言中嵌套的符号对，如：(和)，注释嵌套等.
- 正则表达式足以描述程序语言中的词法.
- 正则表达式在程序设计中的应用：请参考

小节

- 仅能表达可递归定义的语言集合中的一部分.
- 能用正则表达式定义的 Σ^* 上的子集合称为正则语言.
- 从 Kleene Closure 的定义可看出，它使得正则语言中的某个子串能够独立于其他的部分重复 0 次或 n 多次后还属于该正则语言 (泵引理).
- 如果语言在递归构造中有两个部分相关联，则不能用正则表达式描述. 上例表明正则表达式不能描述在程序语言中嵌套的符号对，如：(和)，注释嵌套等.
- 正则表达式足以描述程序语言中的词法.
- 正则表达式在程序设计中的应用：请参考

小节

- 仅能表达可递归定义的语言集合中的一部分.
- 能用正则表达式定义的 Σ^* 上的子集合称为正则语言.
- 从 Kleene Closure 的定义可看出，它使得正则语言中的某个子串能够独立于其他的部分重复 0 次或 n 多次后还属于该正则语言 (泵引理).
- 如果语言在递归构造中有两个部分相关联，则不能用正则表达式描述. 上例表明正则表达式不能描述在程序语言中嵌套的符号对，如：(和)，注释嵌套等.
- 正则表达式足以描述程序语言中的词法.
- 正则表达式在程序设计中的应用：请参考
  - Mastering Regular Expressions, Third Edition, O'Reilly 2006.
  - Regular Expression Cookbook, Second Edition, O'Reilly 2009 (Perl, Python, Ruby, .NET, Java, JavaScript, PHP, and Regular Expressions in Unix).

小节

- 仅能表达可递归定义的语言集合中的一部分.
- 能用正则表达式定义的 Σ^* 上的子集合称为正则语言.
- 从 Kleene Closure 的定义可看出，它使得正则语言中的某个子串能够独立于其他的部分重复 0 次或 n 多次后还属于该正则语言 (泵引理).
- 如果语言在递归构造中有两个部分相关联，则不能用正则表达式描述. 上例表明正则表达式不能描述在程序语言中嵌套的符号对，如：(和)，注释嵌套等.
- 正则表达式足以描述程序语言中的词法.
- 正则表达式在程序设计中的应用：请参考
  - Mastering Regular Expressions, third Ed., Friedl, J. O'Reilly, 2006.
  - Regular Expression Cookbook, Goyvaerts, J. and Levithan, S. O'Reilly, 2009 (Perl, Ruby, Python, Java, C#, VB, PHP, JavaScript, C++, Tcl, Emacs, Vim, grep, MySQL, Oracle, SQL Server, PostgreSQL, DB2, MySQL, Informix, Sybase, Access, SQLite, ...)

小节

- 仅能表达可递归定义的语言集合中的一部分.
- 能用正则表达式定义的 Σ^* 上的子集合称为正则语言.
- 从 Kleene Closure 的定义可看出，它使得正则语言中的某个子串能够独立于其他的部分重复 0 次或 n 多次后还属于该正则语言 (泵引理).
- 如果语言在递归构造中有两个部分相关联，则不能用正则表达式描述. 上例表明正则表达式不能描述在程序语言中嵌套的符号对，如：(和)，注释嵌套等.
- 正则表达式足以描述程序语言中的词法.
- 正则表达式在程序设计中的应用：请参考
  - Mastering Regular Expressions, third Ed., Friedl, J. O'Reilly, 2006.
  - Regular Expressions Cookbook, Goyvaerts, J. et al. O'Reilly, 2009. (RegexBuddy).
  - 我爱正则表达式 (http://iregex.org/).

小节

- 仅能表达可递归定义的语言集合中的一部分.
- 能用正则表达式定义的 Σ^* 上的子集合称为正则语言.
- 从 Kleene Closure 的定义可看出，它使得正则语言中的某个子串能够独立于其他的部分重复 0 次或 n 多次后还属于该正则语言 (泵引理).
- 如果语言在递归构造中有两个部分相关联，则不能用正则表达式描述. 上例表明正则表达式不能描述在程序语言中嵌套的符号对，如：(和)，注释嵌套等.
- 正则表达式足以描述程序语言中的词法.
- 正则表达式在程序设计中的应用：请参考
  - Mastering Regular Expressions, third Ed., Friedl, J. O'Reilly, 2006.
  - Regular Expressions Cookbook, Goyvaerts, J. et al. O'Reilly, 2009. (RegexBuddy).
  - 我爱正则表达式 (http://iregex.org/).

小节

- 仅能表达可递归定义的语言集合中的一部分.
- 能用正则表达式定义的 Σ^* 上的子集合称为正则语言.
- 从 Kleene Closure 的定义可看出，它使得正则语言中的某个子串能够独立于其他的部分重复 0 次或 n 多次后还属于该正则语言 (泵引理).
- 如果语言在递归构造中有两个部分相关联，则不能用正则表达式描述. 上例表明正则表达式不能描述在程序语言中嵌套的符号对，如：(和)，注释嵌套等.
- 正则表达式足以描述程序语言中的词法.
- 正则表达式在程序设计中的应用：请参考
  - Mastering Regular Expressions, third Ed., Friedl, J. O'Reilly, 2006.
  - Regular Expressions Cookbook, Goyvaerts, J. et al. O'Reilly, 2009. (RegexBuddy).
  - 我爱正则表达式 (http://iregex.org/).

小节

- 仅能表达可递归定义的语言集合中的一部分.
- 能用正则表达式定义的 Σ^* 上的子集合称为正则语言.
- 从 Kleene Closure 的定义可看出，它使得正则语言中的某个子串能够独立于其他的部分重复 0 次或 n 多次后还属于该正则语言 (泵引理).
- 如果语言在递归构造中有两个部分相关联，则不能用正则表达式描述. 上例表明正则表达式不能描述在程序语言中嵌套的符号对，如：(和)，注释嵌套等.
- 正则表达式足以描述程序语言中的词法.
- 正则表达式在程序设计中的应用：请参考
  - Mastering Regular Expressions, third Ed., Friedl, J. O'Reilly, 2006.
  - Regular Expressions Cookbook, Goyvaerts, J. et al. O'Reilly, 2009. (RegexBuddy).
  - 我爱正则表达式 (http://iregex.org/).

1 基本概念
- 定义
- 分析方法
- 正则表达式

2 确定的有限自动机
- 基本概念
- DFA 的设计
- DFA 算法

3 不确定的有限自动机
- 基本概念
- 正则表达式到 NFA 的转换
- NFA 到 DFA 的转换
- DFA 状态最小化

4 相关问题

©hfwang

基本概念

正则表达式给出了单词的形式化描述，确定的有限自动机解决如何识别 正则表达式所描述的语言，即词法分析器的程序设计.

Example

ID s= (_|A|B|···|Z)(_|A|B|···|Z)*

程序设计中的几个状态：

准备接受 ID 的第一个字符

已接受至少一个 ID 的字符

其他字符，退出 ID 分析

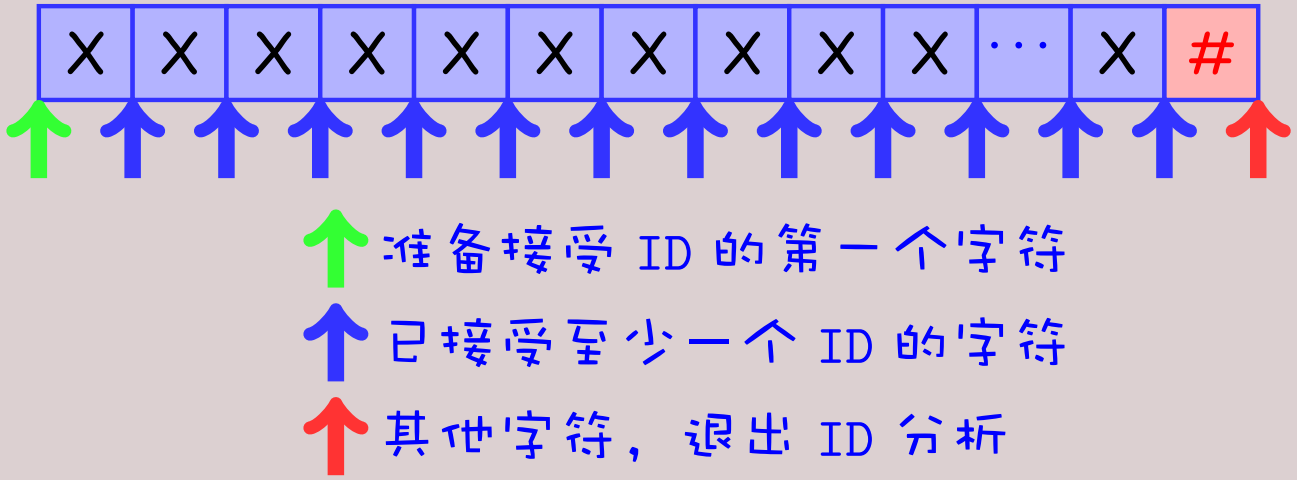
基本概念

正则表达式给出了单词的形式化描述，确定的有限自动机解决如何识别 正则表达式所描述的语言，即词法分析器的程序设计.

Example

ID s= (_|A|B|···|Z)(_|A|B|···|Z)*

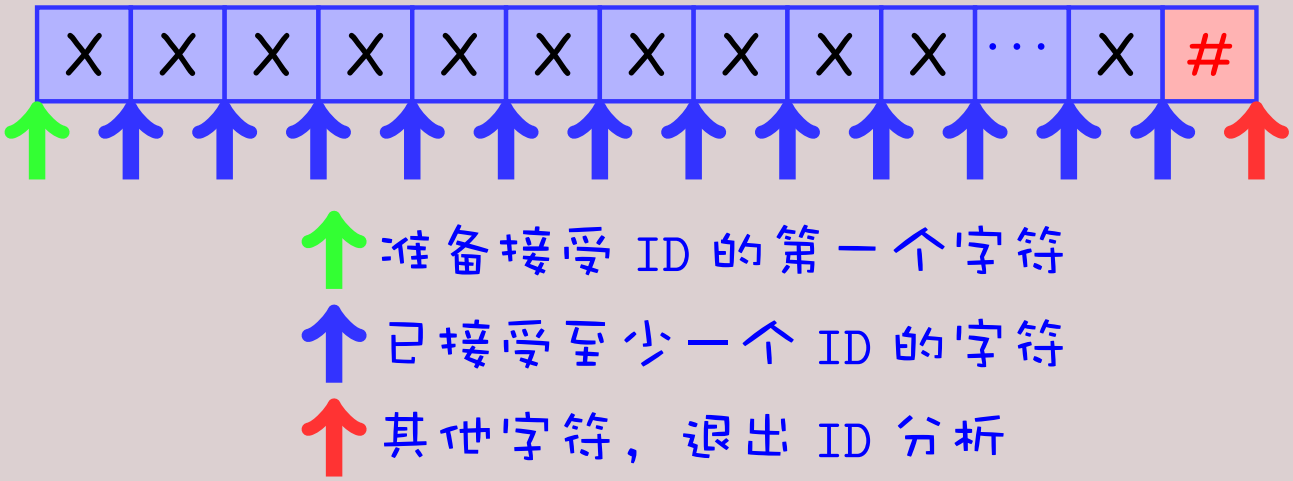
程序设计中的几个状态：



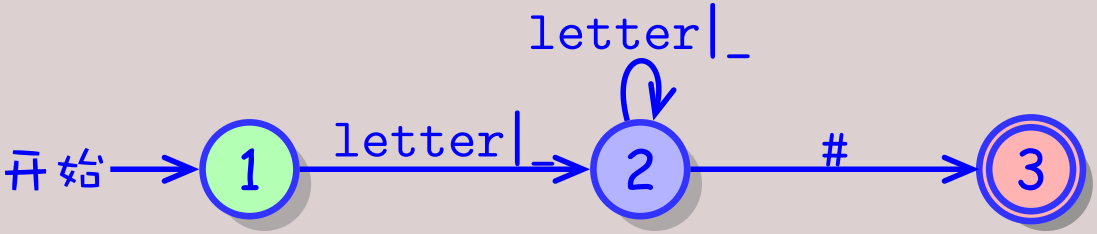
Example

ID = (_|A|B|···|Z)(_|A|B|···|Z)*

程序设计中的几个状态：



状态转换图：



确定的有限自动机 (Deterministic Finite Automata).

DFA 定义

$M = \langle \Sigma, S, T, s_0, F \rangle$, 其中:

- Σ 是字母表.
- S 是有限状态集合.
- T 是 $S \times \Sigma \to S, \langle s, a \rangle \mapsto t$ 的部分函数, 称之为状态转换函数(状态转换图).
- $s_0 \in S$ 是开始状态.
- $F \subseteq S$ 称之为终止 (接受) 状态集合.

Example

$\Sigma = \{_, A, B, \cdots, Z\}$, $S = \{1, 2\}$

$s_0 = 1$, $F = \{2\}$

T:

	letter	_
1	2	2
2	2	2

状态转换图

确定的有限自动机 (Deterministic Finite Automata).

DFA 定义

$M = \langle \Sigma, S, T, s_0, F \rangle$, 其中:

- Σ 是字母表.
- S 是有限状态集合.
- T 是 $S \times \Sigma \to S, \langle s, a \rangle \mapsto t$ 的部分函数, 称之为状态转换函数(状态转换图).
- $s_0 \in S$ 是开始状态.
- $F \subseteq S$ 称之为终止 (接受) 状态集合.

Example

$\Sigma = \{_, A, B, \cdots, Z\}$, $S = \{1, 2\}$

$s_0 = 1$, $F = \{2\}$

T:

	letter	_
1	2	2
2	2	2

状态转换图

确定的有限自动机 (Deterministic Finite Automata).

DFA 定义

$M = \langle \Sigma, S, T, s_0, F \rangle$, 其中:

- Σ 是字母表.
- S 是有限状态集合.
- T 是 $S \times \Sigma \to S, \langle s, a \rangle \mapsto t$ 的部分函数, 称之为状态转换函数(状态转换图).
- $s_0 \in S$ 是开始状态.
- $F \subseteq S$ 称之为终止 (接受) 状态集合.

Example

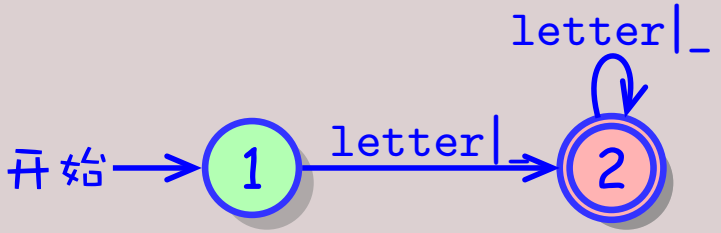
$\Sigma = \{_, A, B, \cdots, Z\}$, $S = \{1, 2\}$

$s_0 = 1$, $F = \{2\}$

T:

	letter	_
1	2	2
2	2	2

状态转换图



DFA 所识别的语言

定义

设 M 是一个 DFA，M 所识别的语言：

$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_n, (s_i \in S \wedge s_n \in F) \wedge s_0 \xrightarrow{x_1} s_1 \xrightarrow{x_2} s_2 \cdots s_{n-1} \xrightarrow{x_n} s_n \}$$

Example (续上例)

$L(M) = L(ID) = \{$由字母和 _ 组成的非空字符串$\}$

DFA 所识别的语言

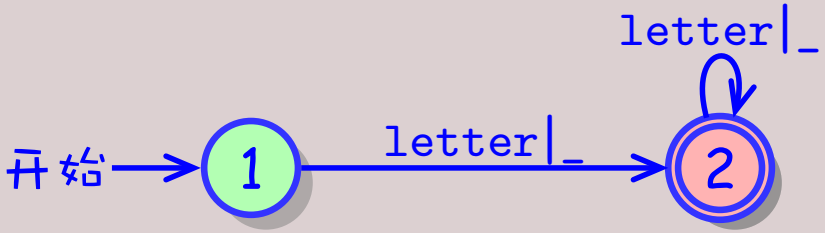
定义

设 M 是一个 DFA，M 所识别的语言：

$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_n, (s_i \in S \wedge s_n \in F) \wedge s_0 \xrightarrow{x_1} s_1 \xrightarrow{x_2} s_2 \cdots s_{n-1} \xrightarrow{x_n} s_n \}$$

Example (续上例)

$L(M) = L(ID) = \{$由字母和 _ 组成的非空字符串$\}$



DFA 所识别的语言

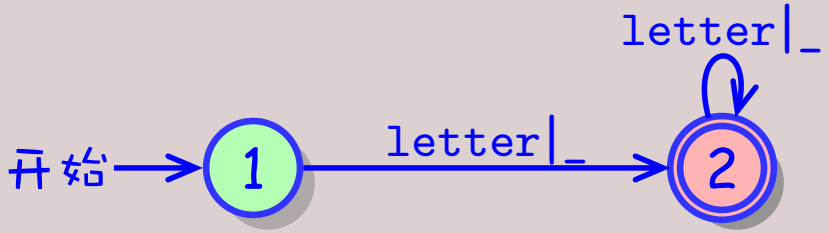
定义

设 M 是一个 DFA，M 所识别的语言：

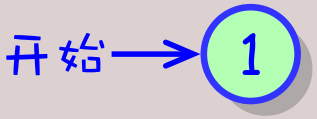
$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_n, (s_i \in S \wedge s_n \in F) \wedge s_0 \xrightarrow{x_1} s_1 \xrightarrow{x_2} s_2 \cdots s_{n-1} \xrightarrow{x_n} s_n \}$$

Example (续上例)

$L(M) = L(ID) = \{$由字母和 _ 组成的非空字符串$\}$



接受 ``_ABC'' 的过程：



DFA 所识别的语言

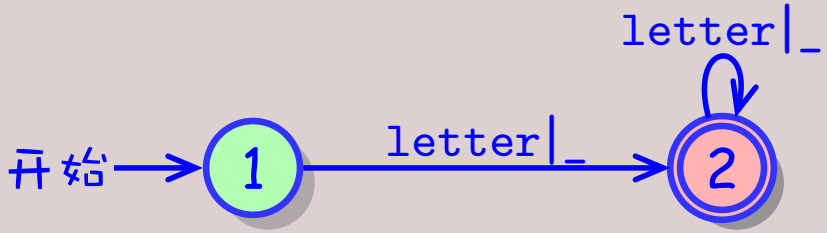
定义

设 M 是一个 DFA，M 所识别的语言：

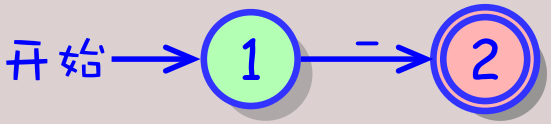
$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_n, (s_i \in S \wedge s_n \in F) \wedge s_0 \xrightarrow{x_1} s_1 \xrightarrow{x_2} s_2 \cdots s_{n-1} \xrightarrow{x_n} s_n \}$$

Example (续上例)

$L(M) = L(ID) = \{$由字母和 _ 组成的非空字符串$\}$



接受 ``_ABC'' 的过程：



DFA 所识别的语言

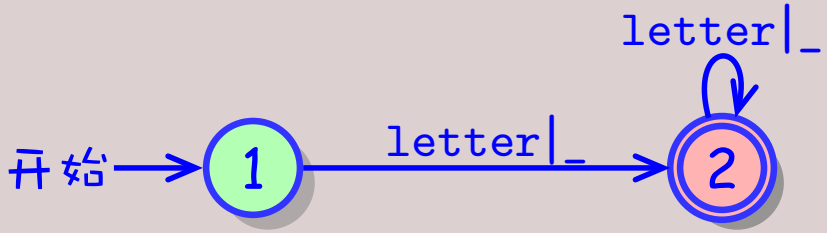
定义

设 M 是一个 DFA，M 所识别的语言：

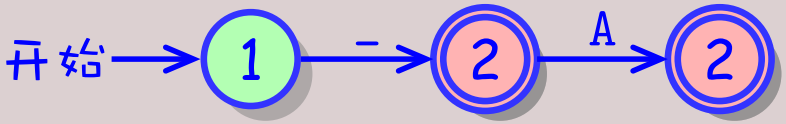
$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_n, (s_i \in S \wedge s_n \in F) \wedge s_0 \xrightarrow{x_1} s_1 \xrightarrow{x_2} s_2 \cdots s_{n-1} \xrightarrow{x_n} s_n \}$$

Example (续上例)

$L(M) = L(ID) = \{$由字母和 _ 组成的非空字符串$\}$



接受 ``_ABC'' 的过程：



DFA 所识别的语言

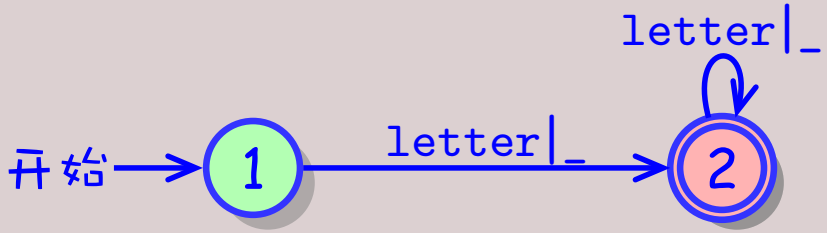
定义

设 M 是一个 DFA，M 所识别的语言：

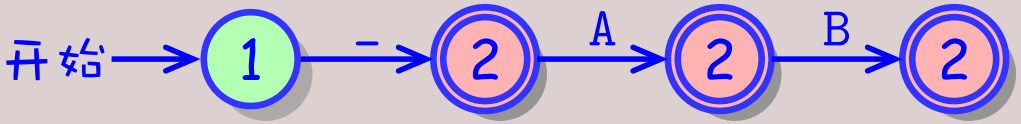
$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_n, (s_i \in S \wedge s_n \in F) \wedge s_0 \xrightarrow{x_1} s_1 \xrightarrow{x_2} s_2 \cdots s_{n-1} \xrightarrow{x_n} s_n \}$$

Example (续上例)

$L(M) = L(ID) = \{$由字母和 _ 组成的非空字符串$\}$



接受 ``_ABC'' 的过程：



DFA 所识别的语言

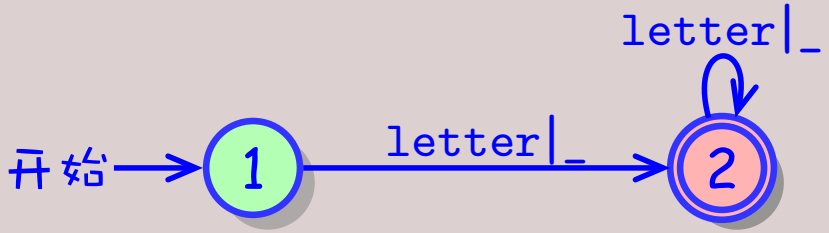
定义

设 M 是一个 DFA，M 所识别的语言：

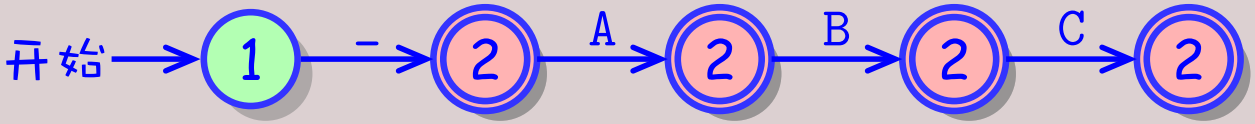
$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_n, (s_i \in S \wedge s_n \in F) \wedge s_0 \xrightarrow{x_1} s_1 \xrightarrow{x_2} s_2 \cdots s_{n-1} \xrightarrow{x_n} s_n \}$$

Example (续上例)

$L(M) = L(ID) = \{$由字母和 _ 组成的非空字符串$\}$



接受 ``_ABC'' 的过程：



DFA 的设计

找出状态转换图中的关键路径.

Example: 实数的科学表示

$$R = \underbrace{(+|-|\varepsilon)\underline{digit\ digit^*}}_{\text{integer part}}\ \underbrace{((.\underline{digit\ digit^*})|\varepsilon)}_{\text{decimal part}}\ \underbrace{((E(+|-|\varepsilon)\underline{digit\ digit^*})|\varepsilon)}_{\text{exponential part}}$$

设计步骤

©hfwang

DFA 的设计

找出状态转换图中的关键路径.

Example: 实数的科学表示

$$R = \underbrace{(+|-|\varepsilon)\underline{digit\ digit^*}}_{\text{integer part}}\ \underbrace{((.\underline{digit\ digit^*})|\varepsilon)}_{\text{decimal part}}\ \underbrace{((E(+|-|\varepsilon)\underline{digit\ digit^*})|\varepsilon)}_{\text{exponential part}}$$

设计步骤

DFA 的设计

找出状态转换图中的关键路径.

Example: 实数的科学表示

$$R = \underbrace{(+|-|\varepsilon)\underline{digit\ digit^*}}_{\text{integer part}}\ \underbrace{((.\underline{digit\ digit^*})|\varepsilon)}_{\text{decimal part}}\ \underbrace{((E(+|-|\varepsilon)\underline{digit\ digit^*})|\varepsilon)}_{\text{exponential part}}$$

设计步骤

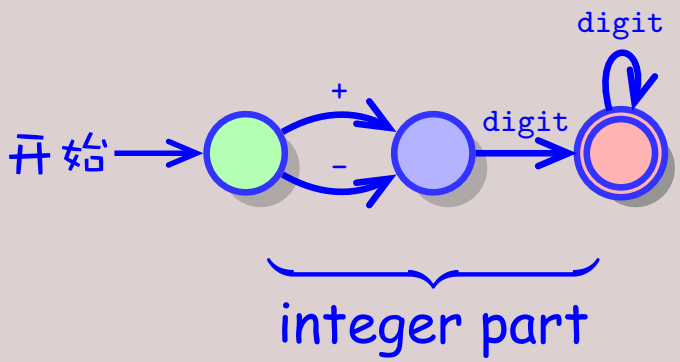
DFA 的设计

找出状态转换图中的关键路径.

Example: 实数的科学表示

$$R = \underbrace{(+|-|\varepsilon)\underline{\text{digit digit}^*}}_{\text{integer part}}\ \underbrace{((.\underline{\text{digit digit}^*})|\varepsilon)}_{\text{decimal part}}\ \underbrace{((E(+|-|\varepsilon)\underline{\text{digit digit}^*})|\varepsilon)}_{\text{exponential part}}$$

设计步骤



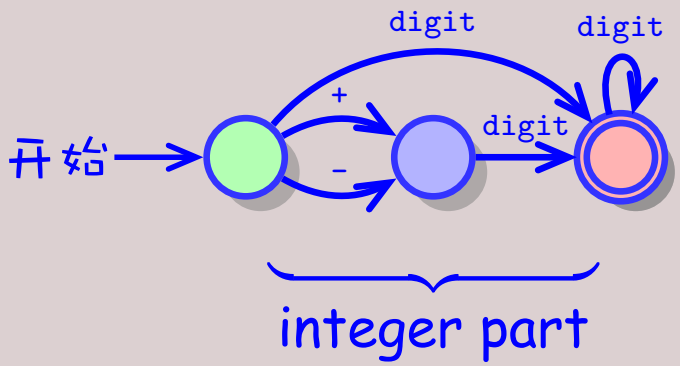
DFA 的设计

找出状态转换图中的关键路径.

Example: 实数的科学表示

$$R = \underbrace{(+|-|\varepsilon)\underline{\text{digit digit}^*}}_{\text{integer part}}\ \underbrace{((.\underline{\text{digit digit}^*})|\varepsilon)}_{\text{decimal part}}\ \underbrace{((E(+|-|\varepsilon)\underline{\text{digit digit}^*})|\varepsilon)}_{\text{exponential part}}$$

设计步骤



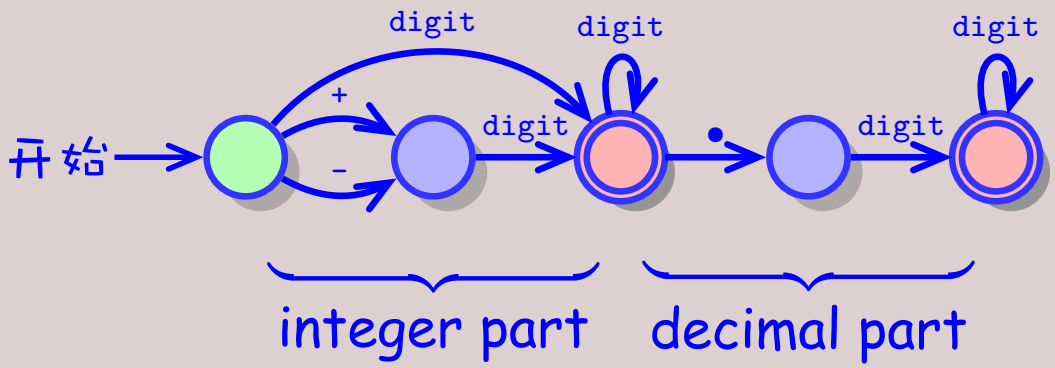
DFA 的设计

找出状态转换图中的关键路径.

Example: 实数的科学表示

$$R = \underbrace{(+|-|\varepsilon)\underline{\text{digit digit}^*}}_{\text{integer part}}\ \underbrace{((.\underline{\text{digit digit}^*})|\varepsilon)}_{\text{decimal part}}\ \underbrace{((E(+|-|\varepsilon)\underline{\text{digit digit}^*})|\varepsilon)}_{\text{exponential part}}$$

设计步骤



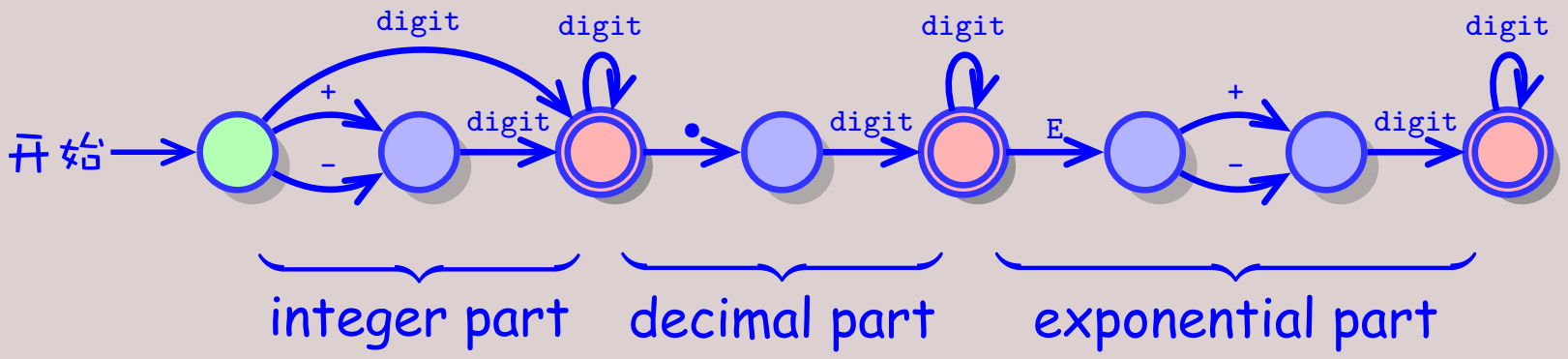
DFA 的设计

找出状态转换图中的关键路径.

Example: 实数的科学表示

$$R = \underbrace{(+|-|\varepsilon)\underline{\text{digit digit}^*}}_{\text{integer part}}\ \underbrace{((.\underline{\text{digit digit}^*})|\varepsilon)}_{\text{decimal part}}\ \underbrace{((E(+|-|\varepsilon)\underline{\text{digit digit}^*})|\varepsilon)}_{\text{exponential part}}$$

设计步骤



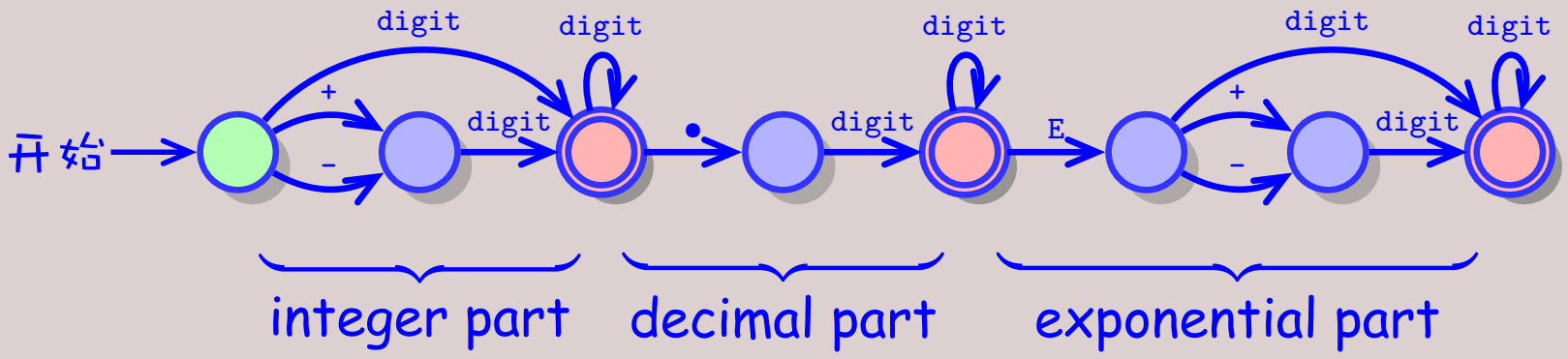
DFA 的设计

找出状态转换图中的关键路径.

Example: 实数的科学表示

$$R = \underbrace{(+|-|\varepsilon)\underline{\text{digit digit}^*}}_{\text{integer part}}\ \underbrace{((.\underline{\text{digit digit}^*})|\varepsilon)}_{\text{decimal part}}\ \underbrace{((E(+|-|\varepsilon)\underline{\text{digit digit}^*})|\varepsilon)}_{\text{exponential part}}$$

设计步骤



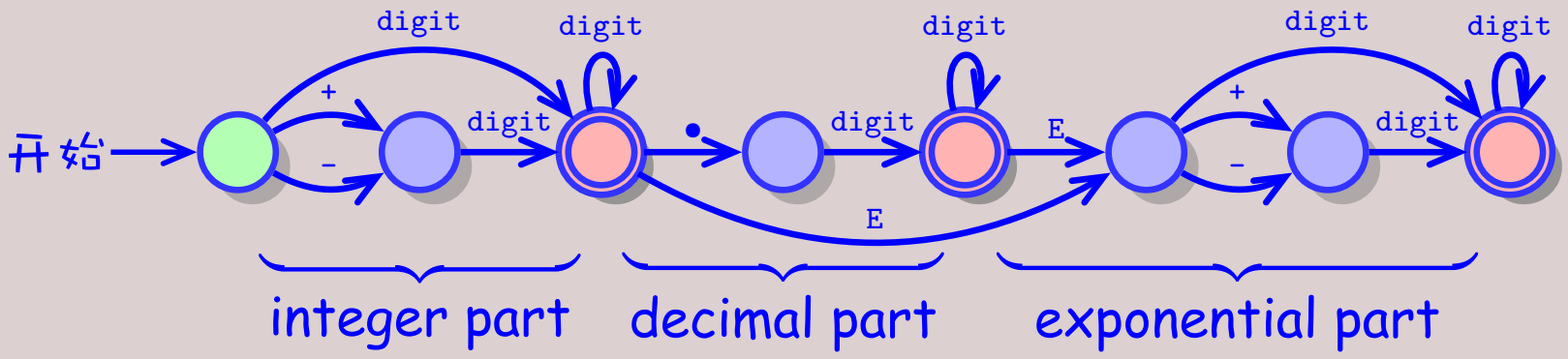
DFA 的设计

找出状态转换图中的关键路径.

Example: 实数的科学表示

$$R = \underbrace{(+|-|\varepsilon)\underline{\text{digit digit}^*}}_{\text{integer part}}\ \underbrace{((.\underline{\text{digit digit}^*})|\varepsilon)}_{\text{decimal part}}\ \underbrace{((E(+|-|\varepsilon)\underline{\text{digit digit}^*})|\varepsilon)}_{\text{exponential part}}$$

设计步骤



Example 1/5

r = (a | b)*abb

©hfwang

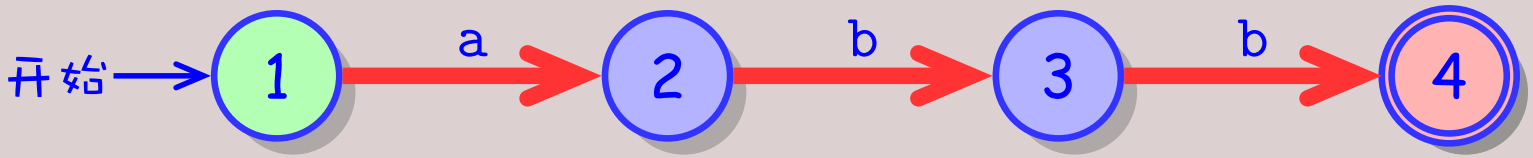
Example 1/5

r = (a | b)*abb

©hfwang

Example 1/5

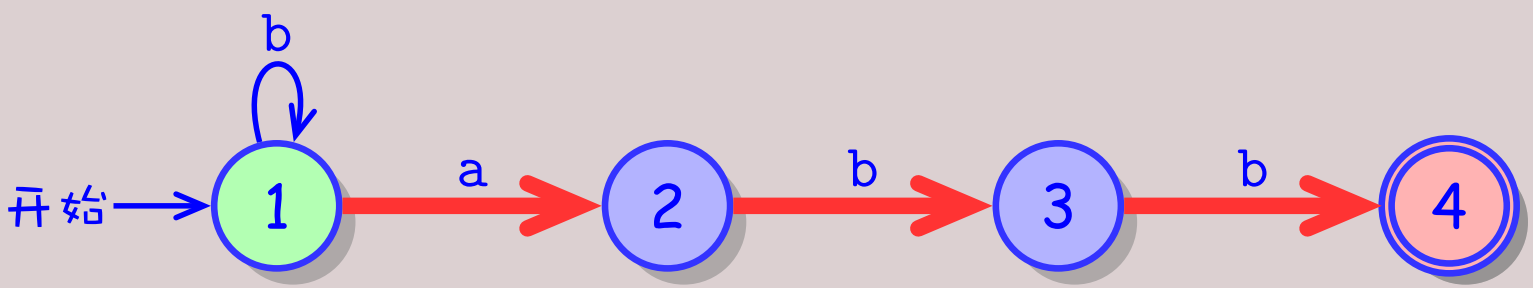
r = (a | b)*abb



©hfwang

Example 1/5

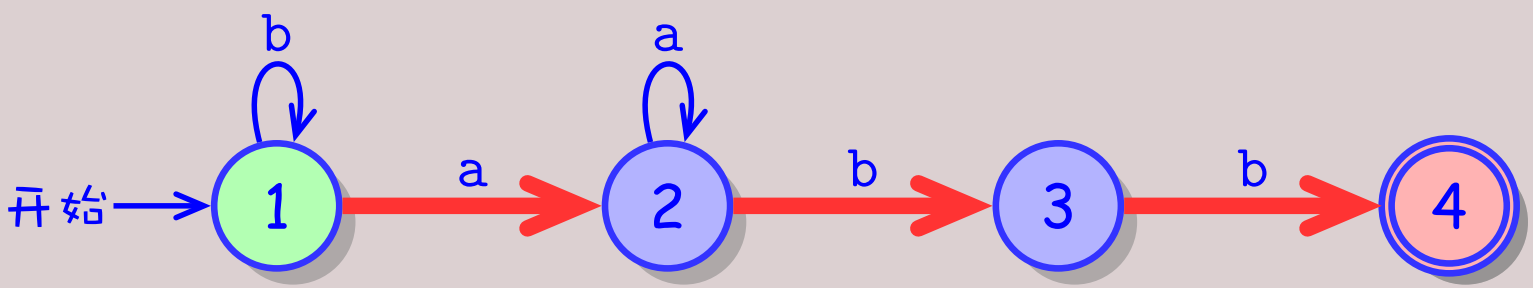
r = (a | b)*abb



©hfwang

Example 1/5

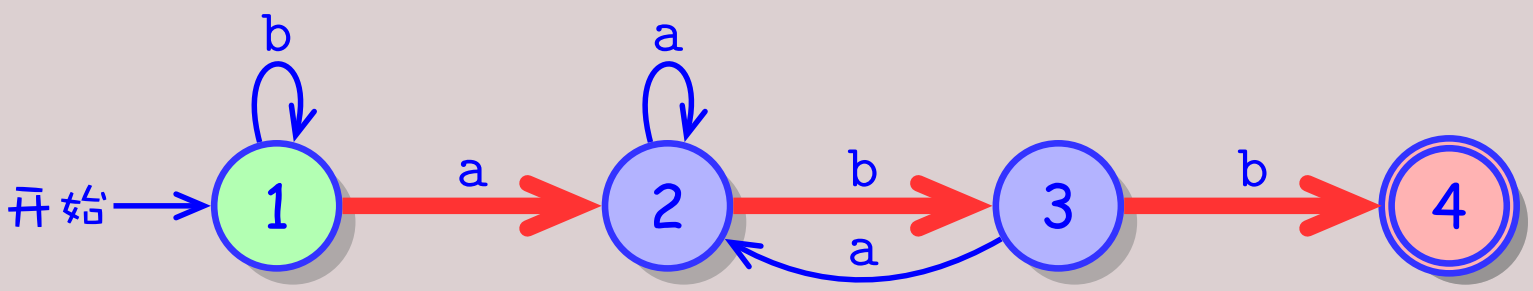
r = (a | b)*abb



©hfwang

Example 1/5

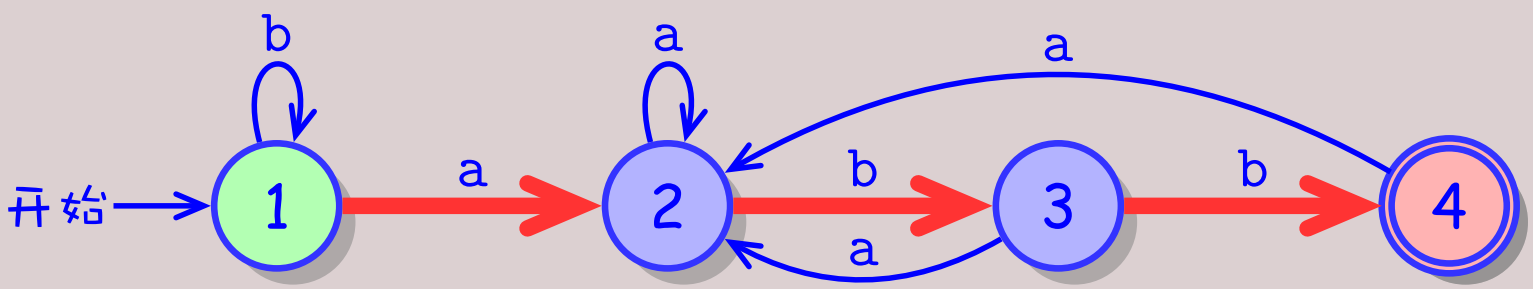
r = (a | b)*abb



©hfwang

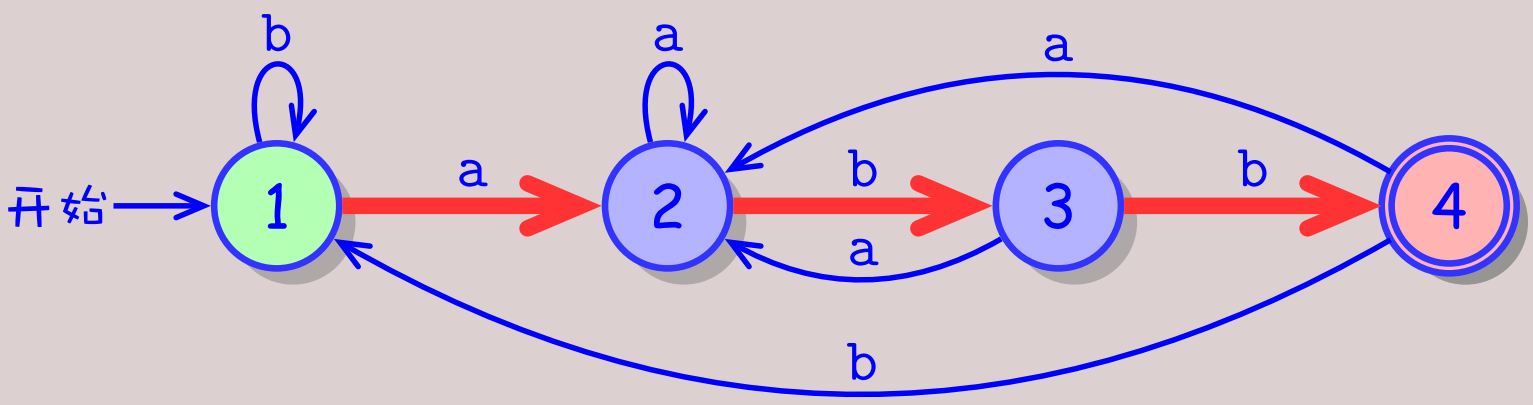
Example 1/5

r = (a | b)*abb



Example 1/5

r = (a | b)*abb



Example 2/5

C 语言的注释

©hfwang

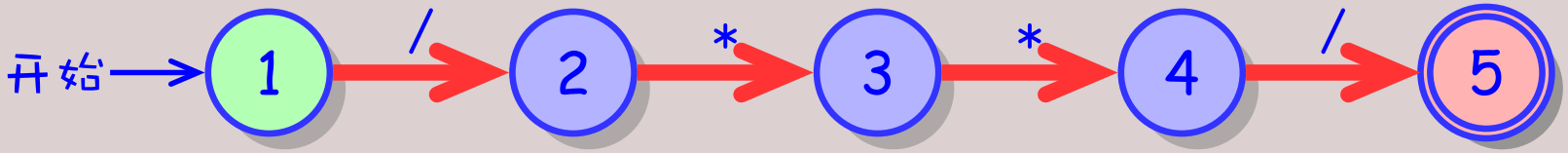
Example 2/5

C 语言的注释

©hfwang

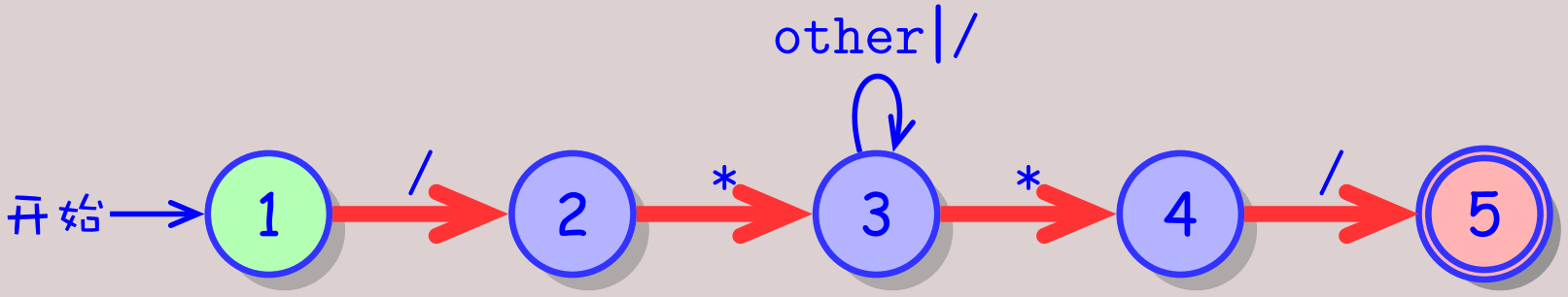
Example 2/5

C 语言的注释



Example 2/5

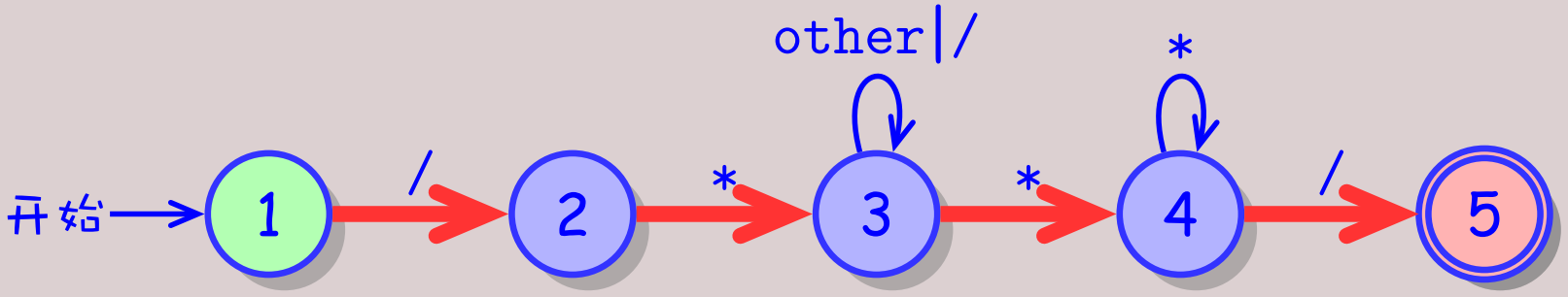
C 语言的注释



©hfwang

Example 2/5

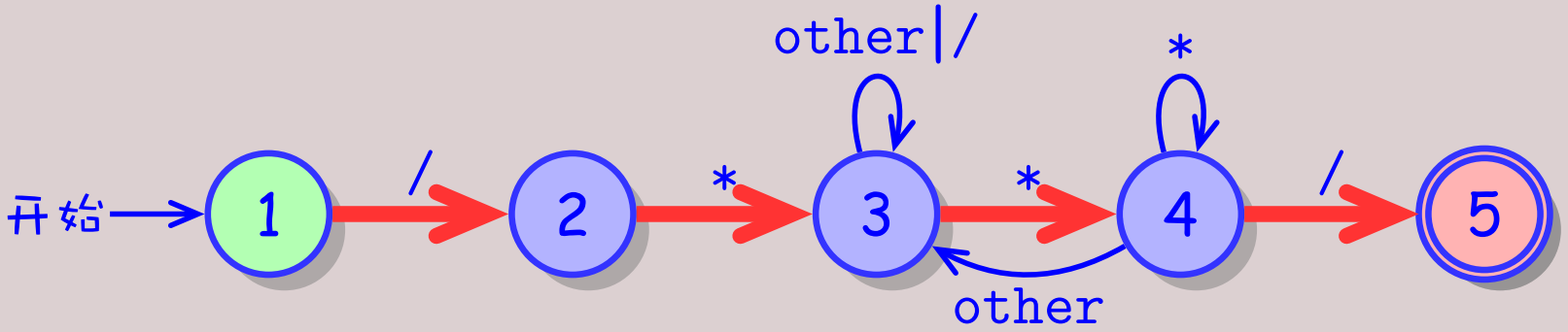
C 语言的注释



©hfwang

Example 2/5

C 语言的注释



©hfwang

Example 3/5 (解 1)

L(M) = {a 和 b 组成的串，a 出现的次数为偶数}

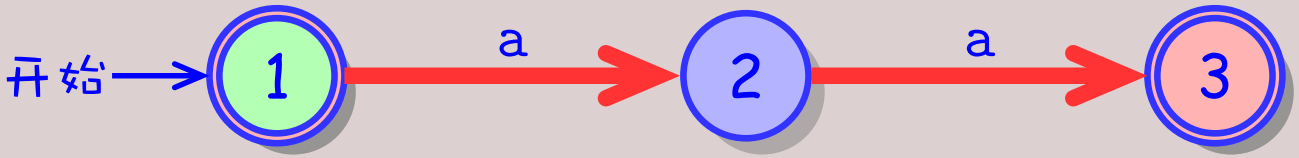
Example 3/5 (解 1)

L(M) = {a 和 b 组成的串，a 出现的次数为偶数}

©hfwang

Example 3/5 (解 1)

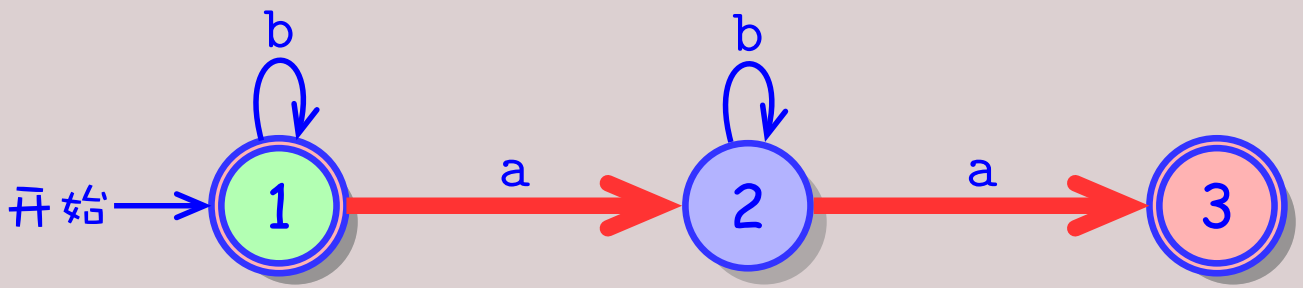
L(M) = {a 和 b 组成的串，a 出现的次数为偶数}



©hfwang

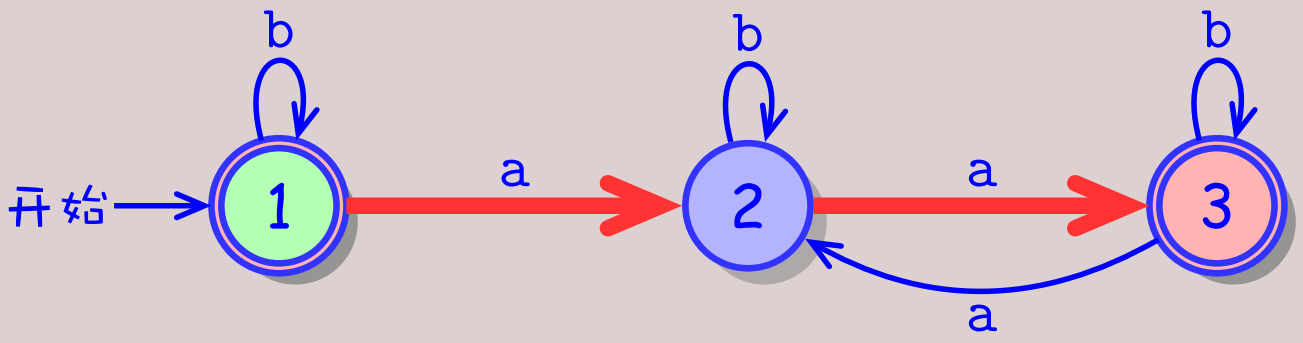
Example 3/5 (解 1)

L(M) = {a 和 b 组成的串，a 出现的次数为偶数}



Example 3/5 (解 1)

L(M) = {a 和 b 组成的串，a 出现的次数为偶数}

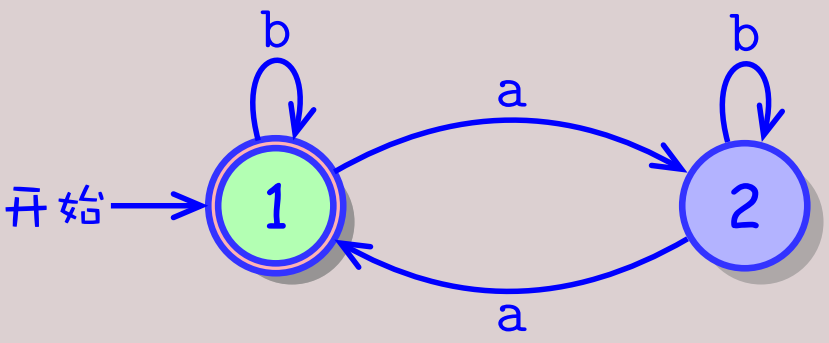


Example 3/5 (解 2)

L(M) = {a 和 b 组成的串，a 出现的次数为偶数}

Example 3/5 (解 2)

L(M) = {a 和 b 组成的串，a 出现的次数为偶数}



Example 4/5

Example

L(M) = {a 和 b 组成的串，a 和 b 出现的次数均为偶数}

Example 4/5

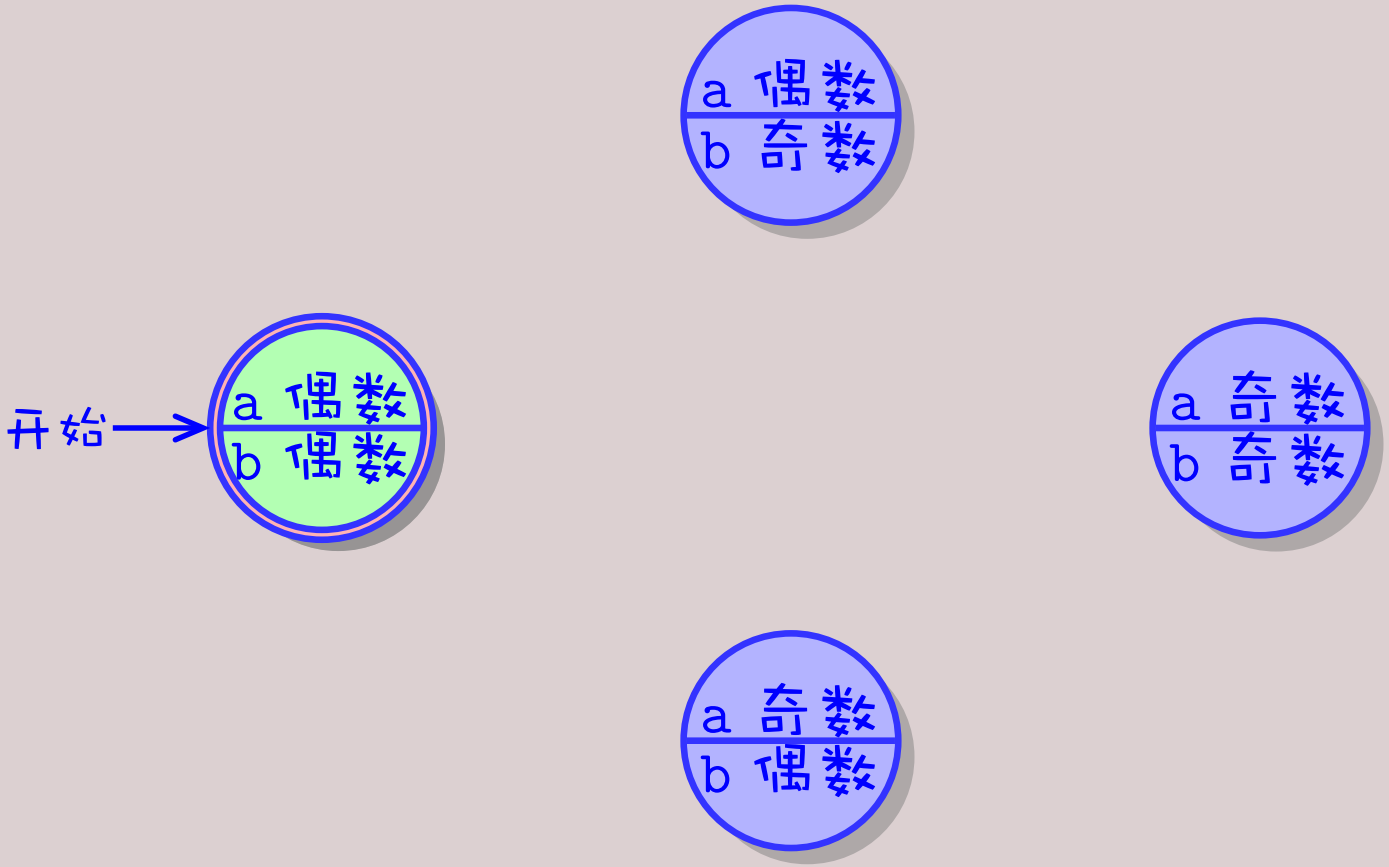
Example

L(M) = {a 和 b 组成的串，a 和 b 出现的次数均为偶数}

Example 4/5

Example

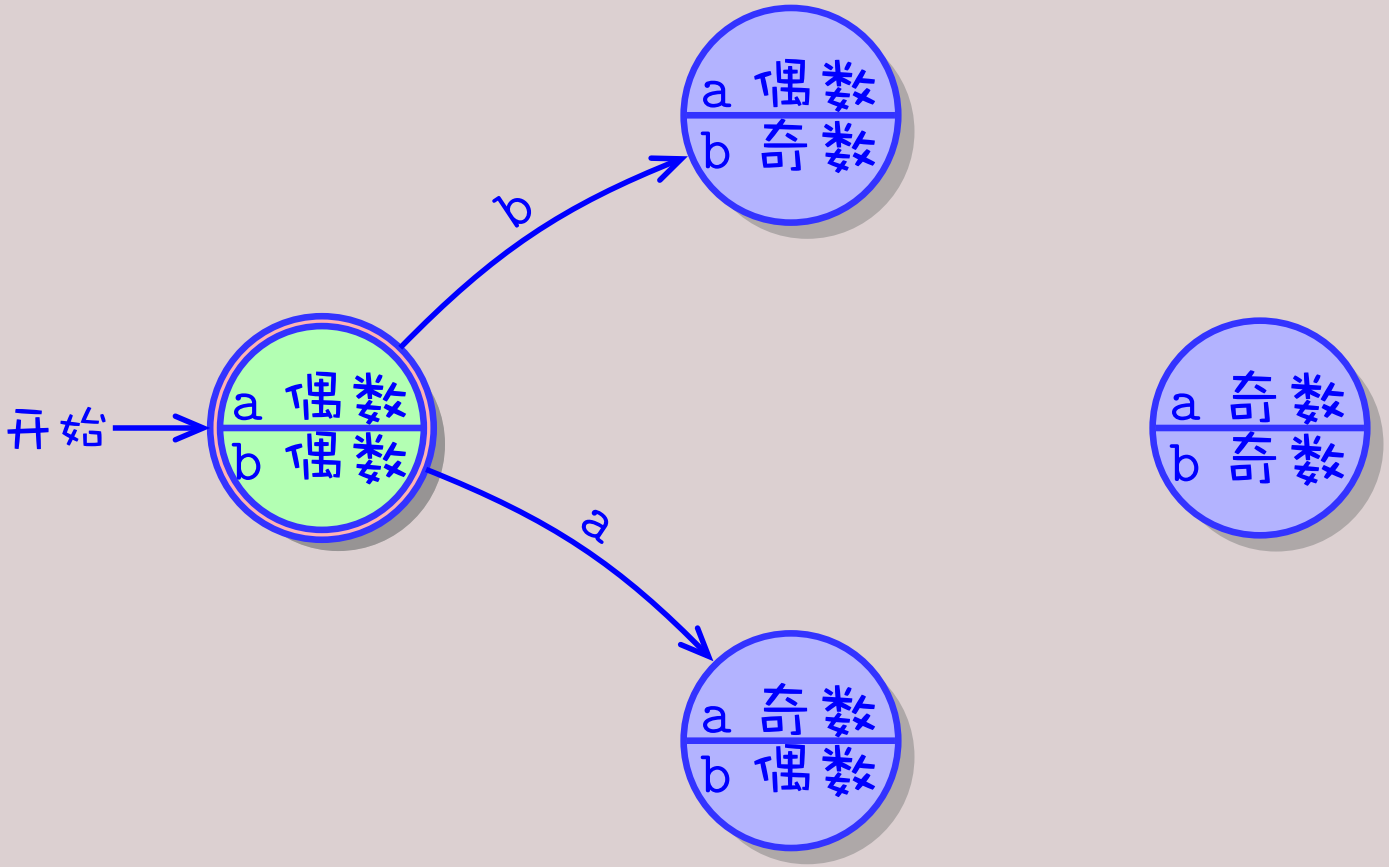
L(M) = {a 和 b 组成的串，a 和 b 出现的次数均为偶数}



Example 4/5

Example

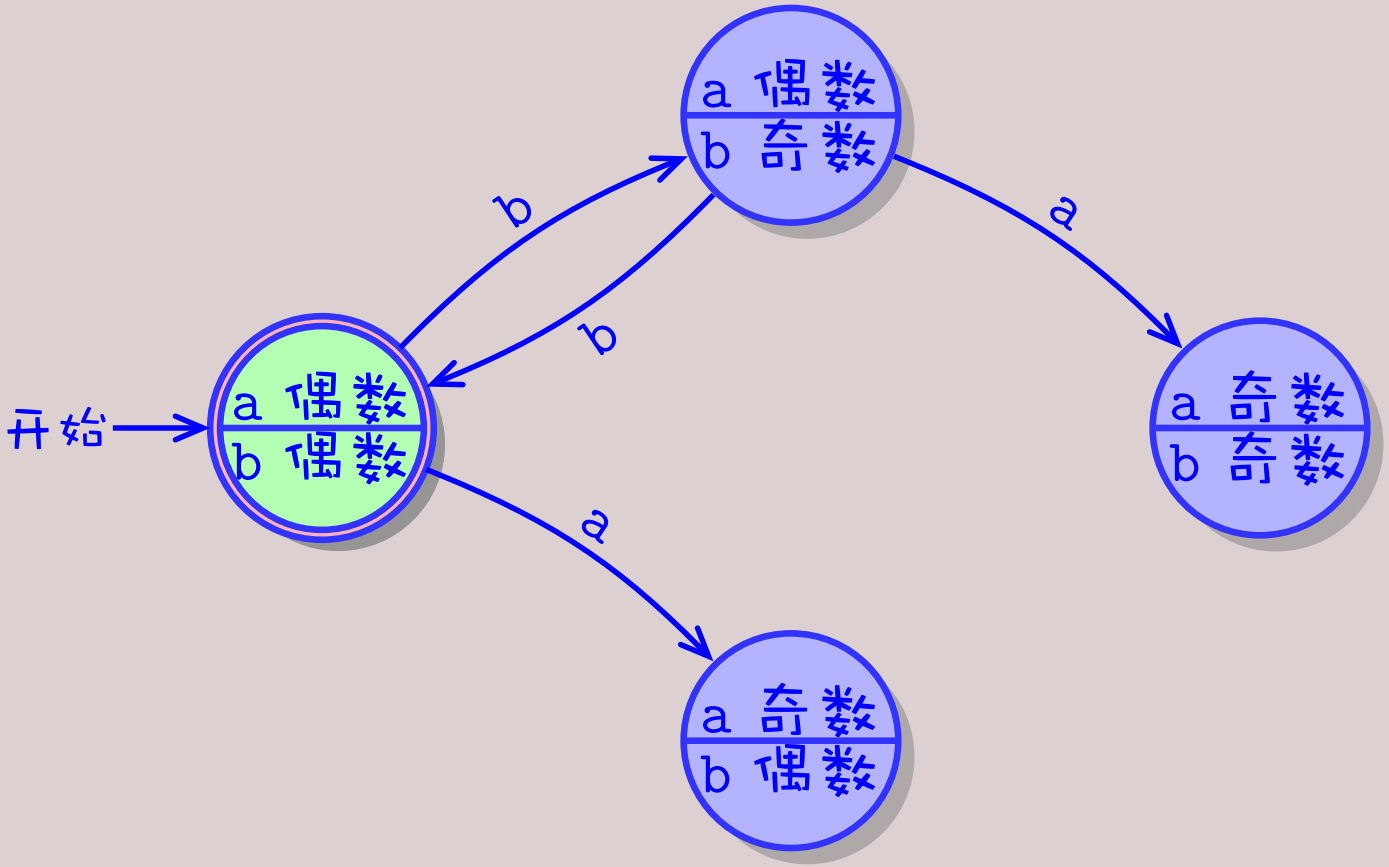
L(M) = {a 和 b 组成的串，a 和 b 出现的次数均为偶数}



Example 4/5

Example

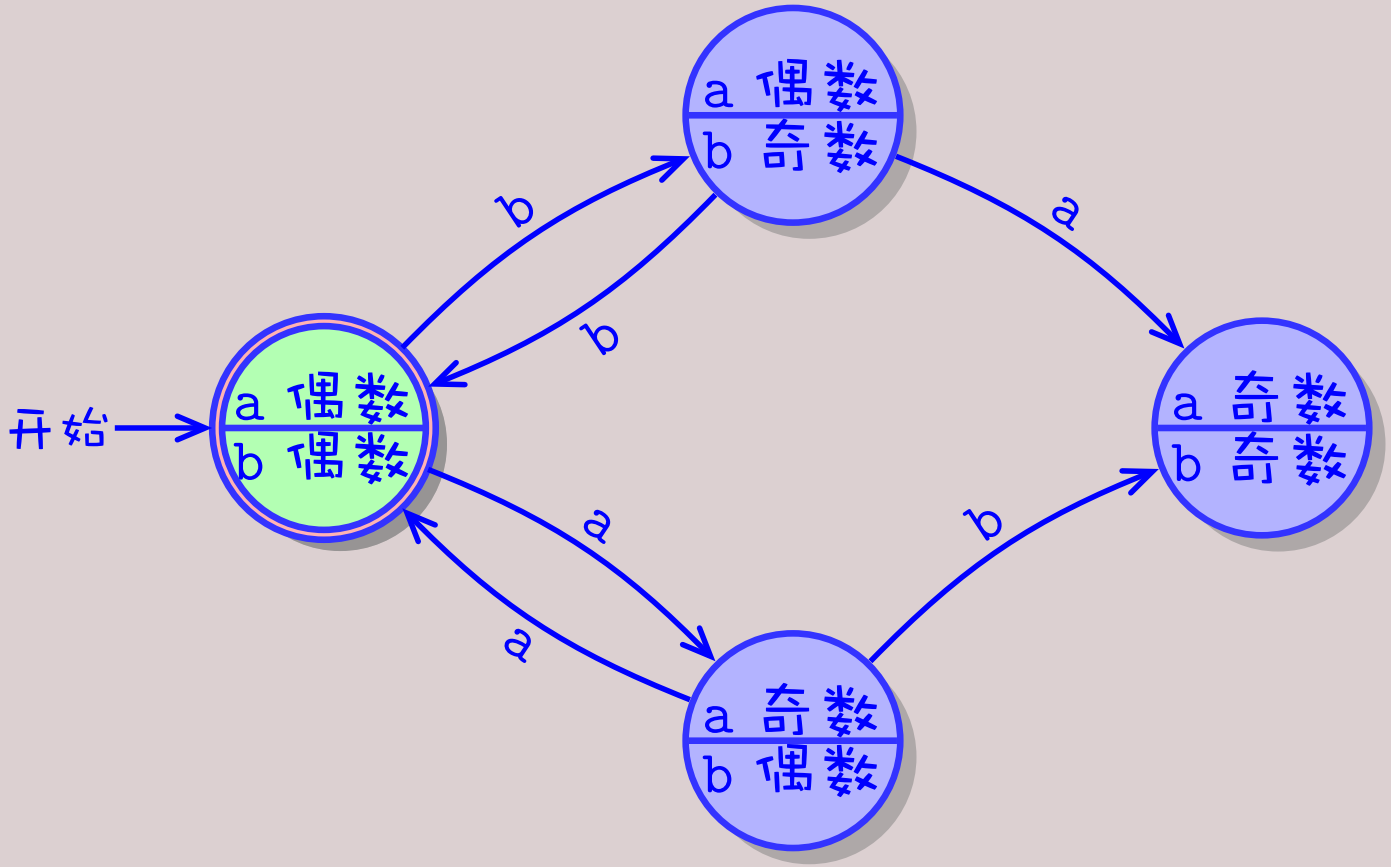
L(M) = {a 和 b 组成的串，a 和 b 出现的次数均为偶数}



Example 4/5

Example

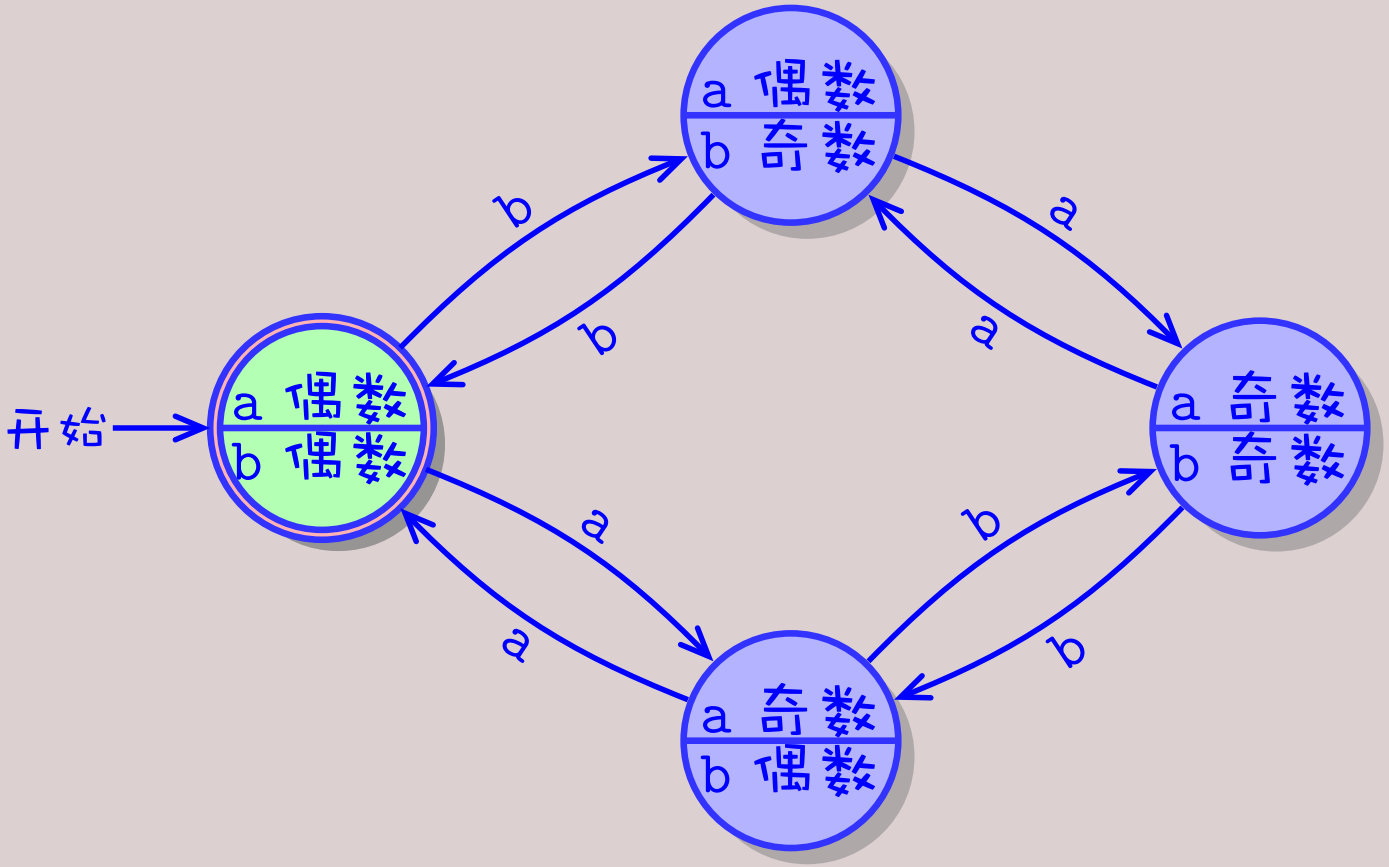
L(M) = {a 和 b 组成的串，a 和 b 出现的次数均为偶数}



Example 4/5

Example

L(M) = {a 和 b 组成的串，a 和 b 出现的次数均为偶数}

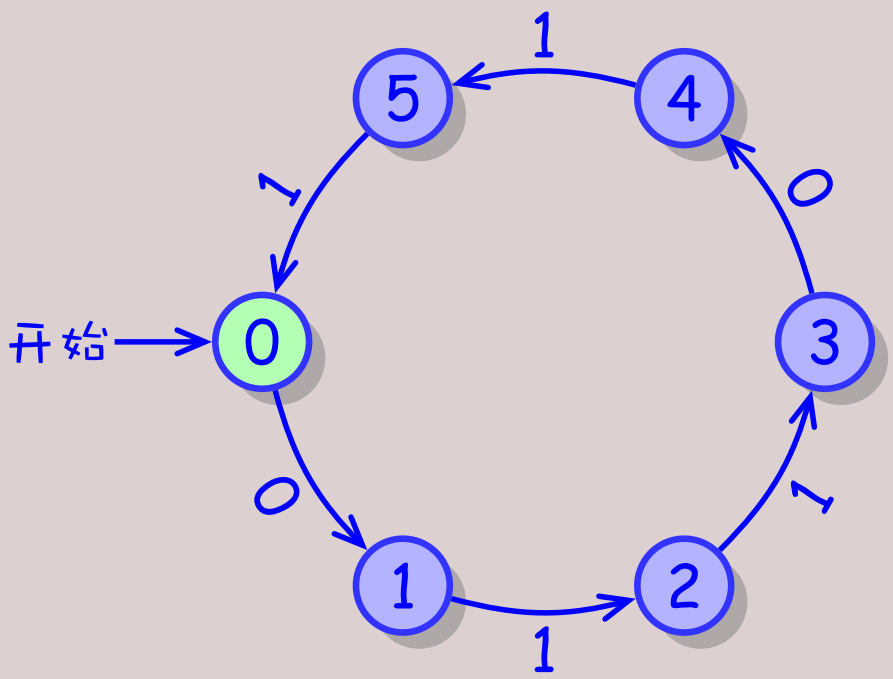


Example 5/5

$\{s \mid s \in \{0,1\}^* \wedge s$ has even number of occurrences of $011\}$

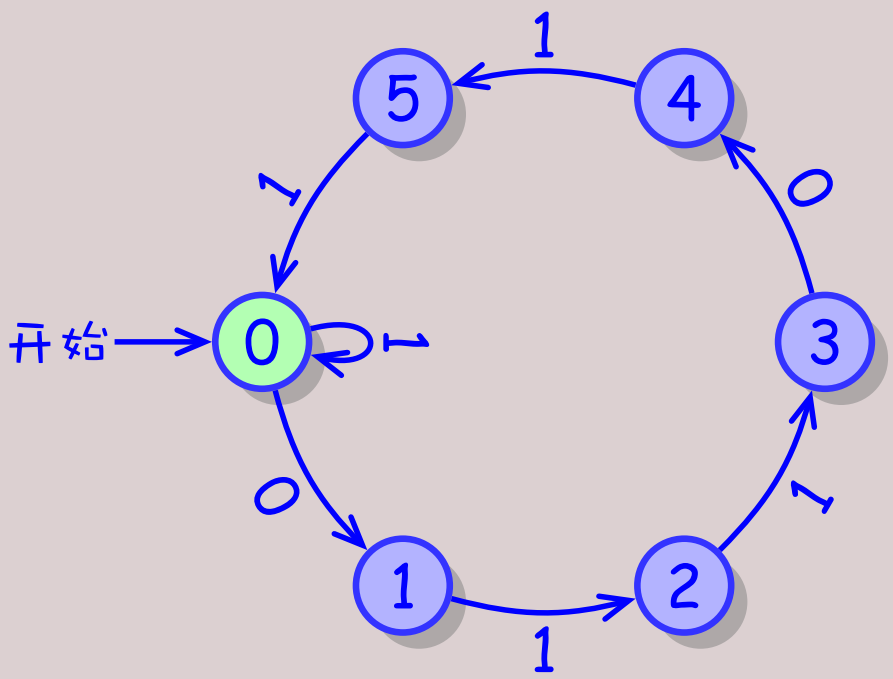
Example 5/5

$\{s \mid s \in \{0,1\}^* \wedge s \text{ has even number of occurrences of } 011\}$



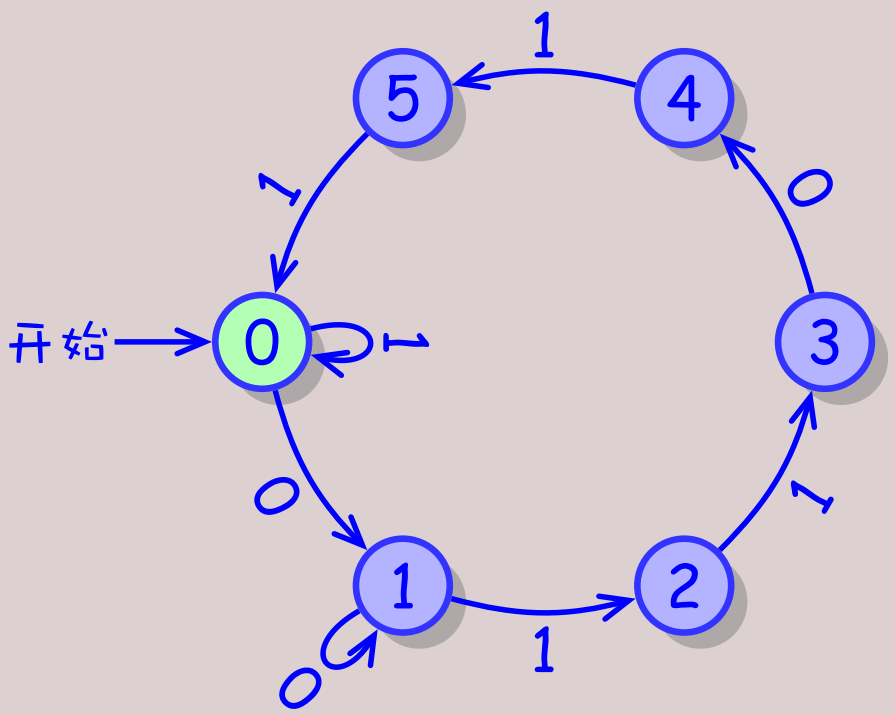
Example 5/5

$\{s \mid s \in \{0,1\}^* \wedge s \text{ has even number of occurrences of } 011\}$



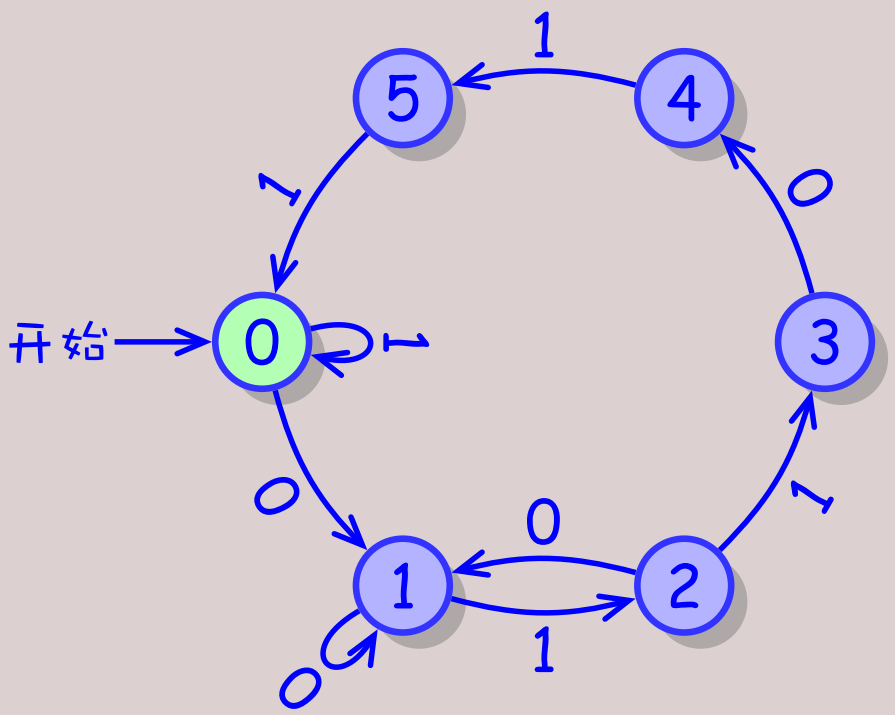
Example 5/5

$\{s \mid s \in \{0,1\}^* \wedge s \text{ has even number of occurrences of } 011\}$



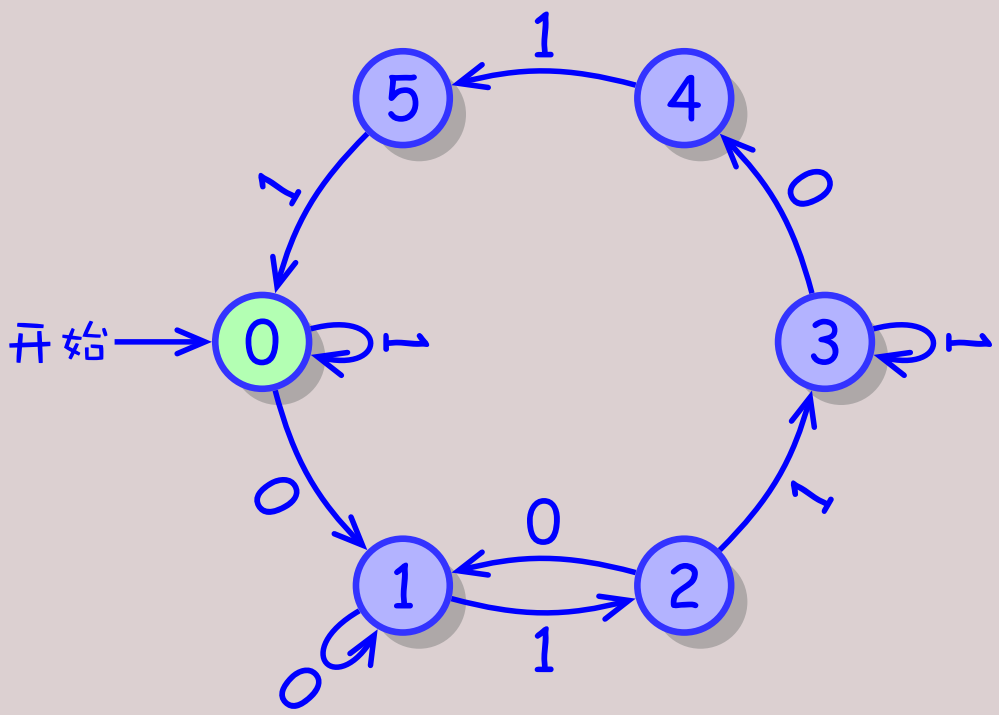
Example 5/5

$\{s \mid s \in \{0,1\}^* \wedge s \text{ has even number of occurrences of } 011\}$



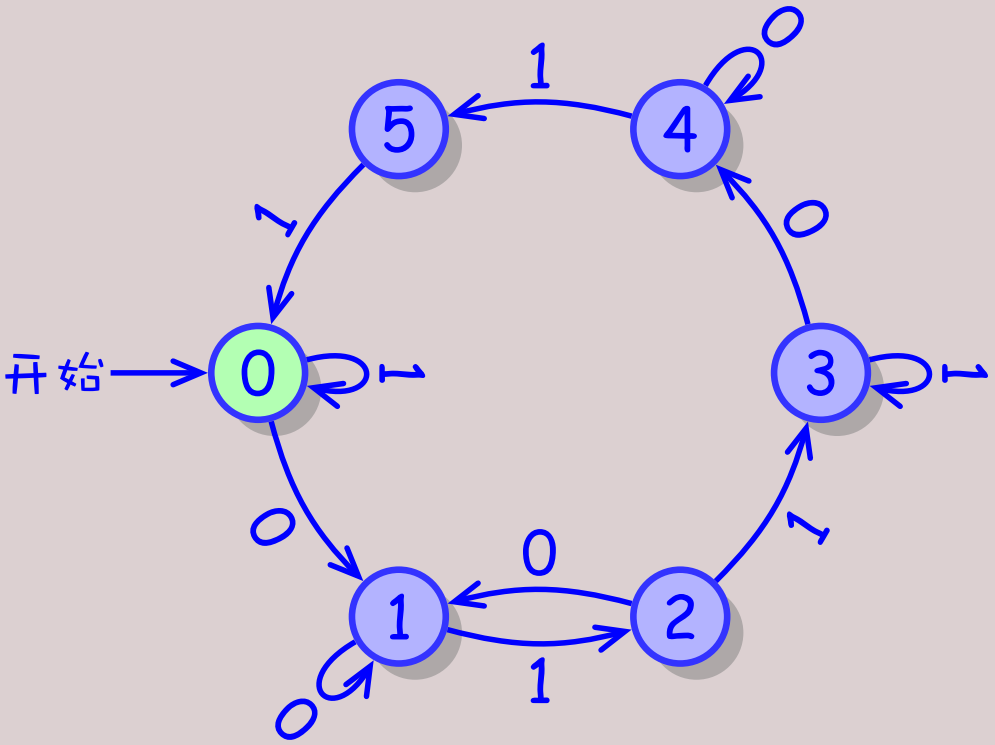
Example 5/5

$\{s \mid s \in \{0,1\}^* \wedge s \text{ has even number of occurrences of } 011\}$



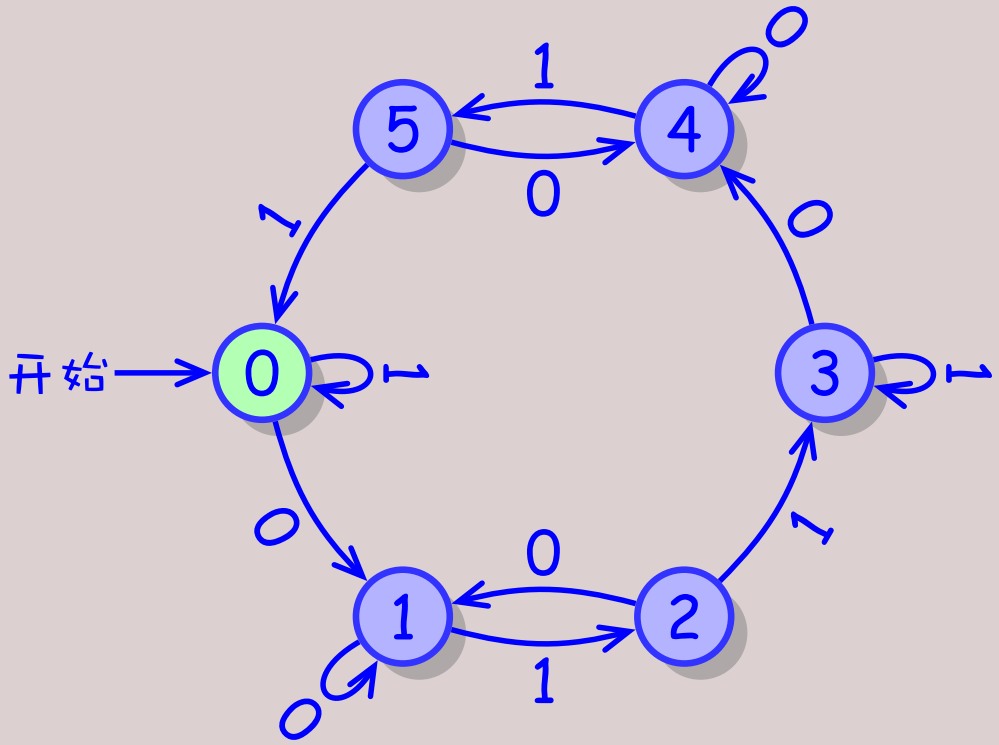
Example 5/5

$\{s \mid s \in \{0,1\}^* \wedge s \text{ has even number of occurrences of } 011\}$



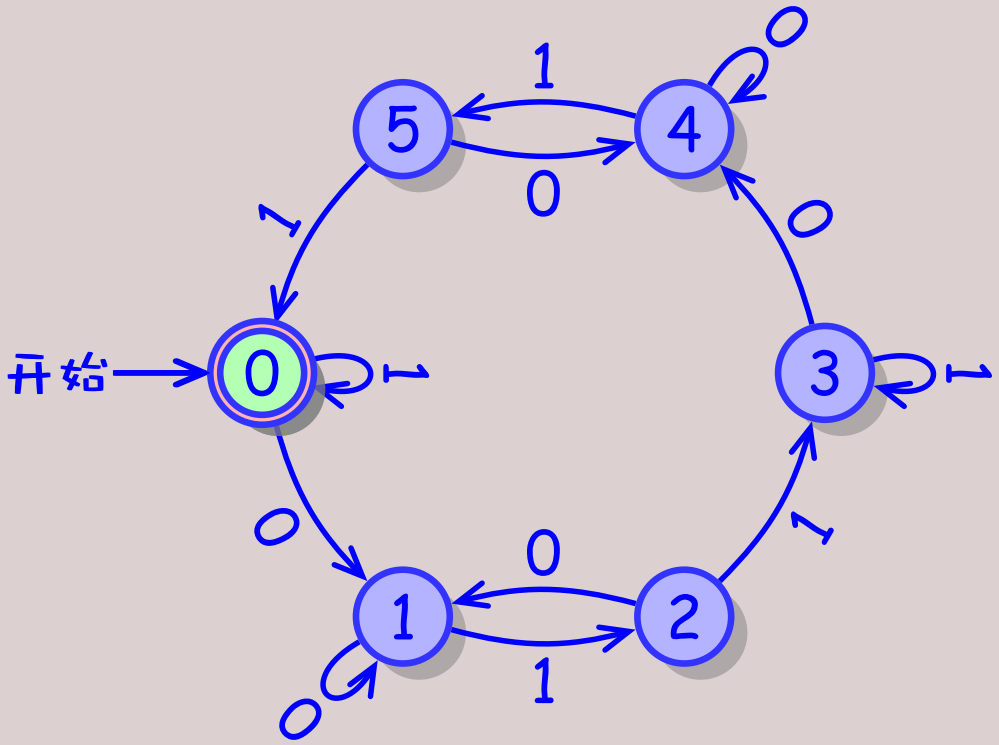
Example 5/5

$\{s \mid s \in \{0,1\}^* \wedge s \text{ has even number of occurrences of } 011\}$



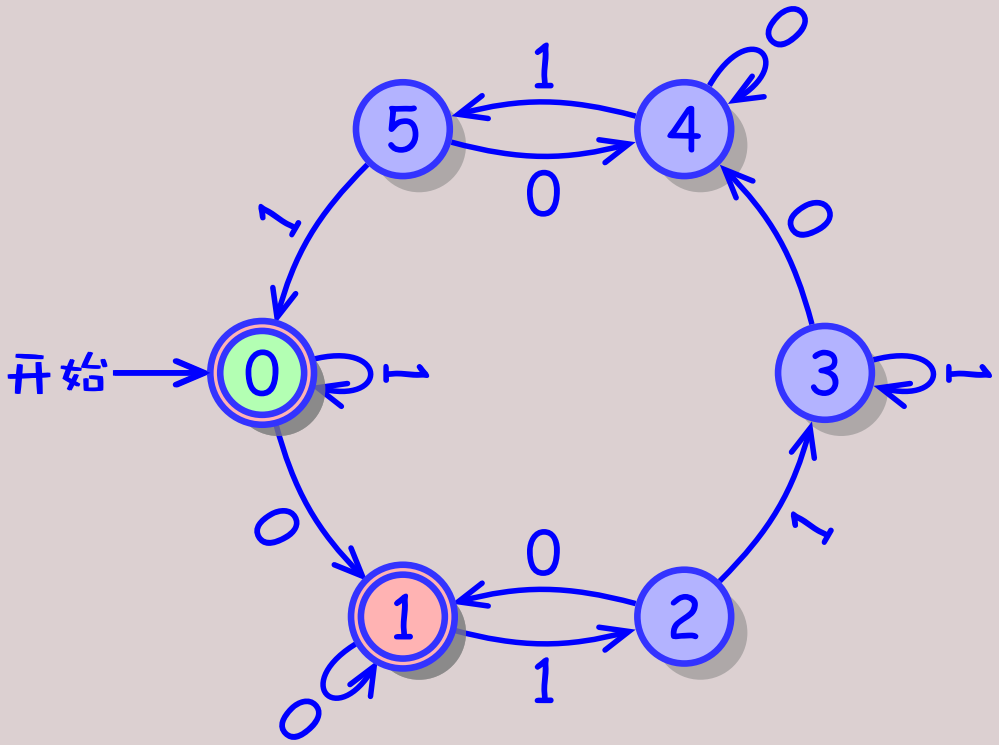
Example 5/5

$\{s \mid s \in \{0,1\}^* \wedge s \text{ has even number of occurrences of } 011\}$



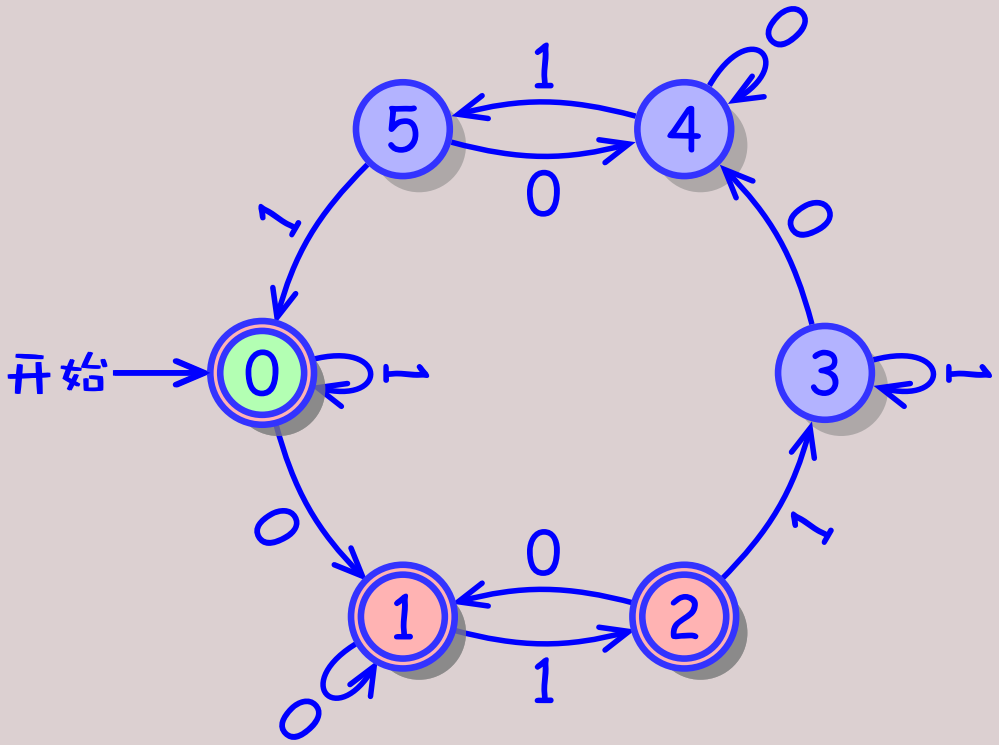
Example 5/5

$\{s \mid s \in \{0,1\}^* \wedge s \text{ has even number of occurrences of } 011\}$



Example 5/5

$\{s \mid s \in \{0,1\}^* \wedge s \text{ has even number of occurrences of } 011\}$



DFA 算法 -- table-driven

无分词

整个输入串看成一个要识别的整体，没有分词处理，即：给定 DFA M $\forall w \in \Sigma^*, w \in L(M)$?

```
state Trans[states][symbols];
  /* 状态转换矩阵 Trans[s][a] = Error
     表示在状态s下，没有标记为a的边 */
symbol getnext(void);
  /* 读一个字符， 如果end of input，返回 EOI */
int scan(void)
{
  state s = s0;
  symbol a = getnext();
  while ( a != EOI && s != Error) {
    s = Trans[s][a];
    a = getnext();
    }
  if ( a == EOI && s ∈ F)
    return 1;
  return 0;
}
```

DFA 算法 --- 有分词 1/2.

- 分词问题：如果 DFA 的接受状态还有引出的边到其他的状态，扫描程序必须要要继续向前扫描搜索到一个出错状态，再回退到扫描过程的某个接受状态，从而实现分词：
  - Ex: L(M) = L((abcd)*), Input string: ``abcdabce'', DFA(M) 在识别 ``abcd'' 到接受状态，但是该状态还可以继续接受字符，DFA 在扫描最后一个 ``e'' 时出错，扫描程序必须将 ``abcd'' 作为识别串并恢复缓冲区到该串之后第一个字符.
- 最长匹配原则.
  - Ex: L(M) = {ab, abc, abcd}, Input string: ``abcdef'', 必须要识别前缀 ``abcd'' 作为匹配成功子串，而不是 ``ab'' 或 ``abc''.
- 子串问题：
  - Ex: L(M) = {cbedef, bcde}, Input string: ``abcdef'', 如果自动机 M 在识别串 ``abcd'' 上停止，则 M 已经到达了接受状态，但有的子串可能还没有扫描到，因此需要

DFA 算法 --- 有分词 1/2.

- 分词问题：如果 DFA 的接受状态还有引出的边到其他的状态，扫描程序必须要要继续向前扫描搜索到一个出错状态，再回退到扫描过程的某个接受状态，从而实现分词：
  - Ex: L(M) = L((abcd)$^+$), Input string: ``abcdabce'', DFA(M) 在识别 ``abcd'' 到接受状态，但是该状态还可以继续接受字符，DFA 在扫描最后一个 ``e'' 时出错，扫描程序必须将 ``abcd'' 作为识别串并恢复缓冲区到该串之后第一个字符.
- 最长匹配原则.
  - Ex: L(M) = {ab, abc, abcd}, Input string: ``abcdabce'', 扫描程序必须识别最长 ``abcd'' 作为匹配成功字串，而不是 ``ab'' 或 ``abc''.
- 子串问题：
  - Ex: L(M) = {abcdef, bcde}, Input string: ``abcdefg'', 扫描程序必须识别 ``abcdef'' 作为匹配成功字串，而不是 ``bcde''.

DFA 算法 --- 有分词 1/2

- 分词问题：如果 DFA 的接受状态还有引出的边到其他的状态，扫描程序必须要要继续向前扫描搜索到一个出错状态，再回退到扫描过程的某个接受状态，从而实现分词：
  - Ex: $L(M) = L((abcd)^+)$, Input string: ``abcdabce'', DFA(M) 在识别 ``abcd'' 到接受状态，但是该状态还可以继续接受字符，DFA 在扫描最后一个 ``e'' 时出错，扫描程序必须将 ``abcd'' 作为识别串并恢复缓冲区到该串之后第一个字符.
- 最长匹配原则.
  - Ex: $L(M) = \{ab, abc, abcd\}$, Input string: ``abcdef'', 算法将识别前缀 ``abcd'' 作为匹配 M的子串，而非 ``ab'' 或 ``abc''.
- 子串问题：
  - Ex: $L(M) = \{abcdef, pqrst\}$, Input string: ``abcdefpqrst'', 算法在扫描 M 的过程中，发现 ``abcdef'' 是 M 的子串，但是 ``pqrst'' 也是 M 的子串，因为 ``pqrst'' 在 ``abcdef'' 之后，因此 ``abcdef'' 不是 M 的子串.

DFA 算法 --- 有分词 1/2

- 分词问题：如果 DFA 的接受状态还有引出的边到其他的状态，扫描程序必须要要继续向前扫描搜索到一个出错状态，再回退到扫描过程的某个接受状态，从而实现分词：
  - Ex: $L(M) = L((abcd)^+)$, Input string: ``abcdabce'', DFA(M) 在识别 ``abcd'' 到接受状态，但是该状态还可以继续接受字符，DFA 在扫描最后一个 ``e'' 时出错，扫描程序必须将 ``abcd'' 作为识别串并恢复缓冲区到该串之后第一个字符.
- 最长匹配原则.
  - Ex: $L(M) = \{ab, abc, abcd\}$, Input string: ``abcdef''，算法将识别前缀 ``abcd'' 作为匹配 M的子串，而非 ``ab'' 或 ``abc''.
- 子串问题：
  - Ex: $L(M) = \{abcdef, bcde\}$, Input string: ``abcdefg''，算法将识别前缀 ``abcdef'' 作为匹配 M的子串，而非 ``bcde''.

DFA 算法 --- 有分词 1/2

- 分词问题：如果 DFA 的接受状态还有引出的边到其他的状态，扫描程序必须要要继续向前扫描搜索到一个出错状态，再回退到扫描过程的某个接受状态，从而实现分词：
  - Ex: $L(M) = L((abcd)^+)$, Input string: ``abcdabce'', DFA(M) 在识别 ``abcd'' 到接受状态，但是该状态还可以继续接受字符，DFA 在扫描最后一个 ``e'' 时出错，扫描程序必须将 ``abcd'' 作为识别串并恢复缓冲区到该串之后第一个字符.
- 最长匹配原则.
  - Ex: $L(M) = \{ab, abc, abcd\}$, Input string: ``abcdef'', 算法将识别前缀 ``abcd'' 作为匹配 M的子串，而非 ``ab'' 或 ``abc''.
- 子串问题：
  - Ex: $L(M) = \{abcdef, bcde\}$, Input string: ``abcdeg'', 如果自动机从 ``a'' 识别到 ``g'' 出错，则要从 ``b'' 重新开始，因为可以已扫描过的子串能被该自动机识别.

DFA 算法 --- 有分词 1/2

- 分词问题：如果 DFA 的接受状态还有引出的边到其他的状态，扫描程序必须要要继续向前扫描搜索到一个出错状态，再回退到扫描过程的某个接受状态，从而实现分词：
  - Ex: $L(M) = L((abcd)^+)$, Input string: ``abcdabce'', DFA(M) 在识别 ``abcd'' 到接受状态，但是该状态还可以继续接受字符，DFA 在扫描最后一个 ``e'' 时出错，扫描程序必须将 ``abcd'' 作为识别串并恢复缓冲区到该串之后第一个字符.
- 最长匹配原则.
  - Ex: $L(M) = \{ab, abc, abcd\}$, Input string: ``abcdef''，算法将识别前缀 ``abcd'' 作为匹配 M的子串，而非 ``ab'' 或 ``abc''.
- 子串问题：
  - Ex: $L(M) = \{abcdef, bcde\}$, Input string: ``abcdeg''，如果自动机从 ``a'' 识别到 ``g'' 出错，则要从 ``b'' 重新开始，因为可以已扫描过的子串能被该自动机识别.

DFA 算法 --- 有分词 1/2

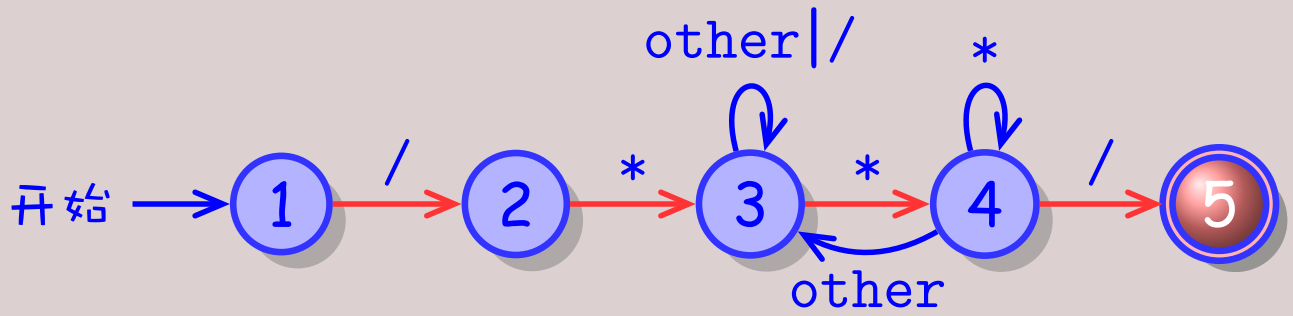
- 分词问题：如果 DFA 的接受状态还有引出的边到其他的状态，扫描程序必须要要继续向前扫描搜索到一个出错状态，再回退到扫描过程的某个接受状态，从而实现分词：
  - Ex: $L(M) = L((abcd)^+)$, Input string: ``abcdabce'', DFA(M) 在识别 ``abcd'' 到接受状态，但是该状态还可以继续接受字符，DFA 在扫描最后一个 ``e'' 时出错，扫描程序必须将 ``abcd'' 作为识别串并恢复缓冲区到该串之后第一个字符.
- 最长匹配原则.
  - Ex: $L(M) = \{ab, abc, abcd\}$, Input string: ``abcdef''，算法将识别前缀 ``abcd'' 作为匹配 M的子串，而非 ``ab'' 或 ``abc''.
- 子串问题：
  - Ex: $L(M) = \{abcdef, bcde\}$, Input string: ``abcdeg''，如果自动机从 ``a'' 识别到 ``g'' 出错，则要从 ``b'' 重新开始，因为可以已扫描过的子串能被该自动机识别.

DFA 算法 --- 有分词 2/2

```
symbol *current = 缓冲区当前指针, *begin_token;
int scan(void)
{
  symbol *p = NULL,   /* 指向最近一次接受字符在缓冲区的位子 */
  begin_token = current;
  while ( 1 ) {
    state s = s0;  /* start state */
    while ( *current != EOI && s != Error ) {
      s = Trans[s][*current];
      if ( s ∈ F )
        p = current; /* 获得当前接受字符在缓冲区的指针 */
      current ++;
      }
    if ( p != NULL ) { /* 分词及最长匹配 */
      执行识别单词的相关操作;
      current = p + 1;
      continue;
      }
    if ( s == Error ) {
      current = begin_token + 1;  /* 跳过未识别的symbol */
      continue;
      }
    if ( *current == EOI )  return 1;
  }
}
```

Example: DFA 在程序设计中的应用 (1/2)

识别 C 语言注释的 DFA



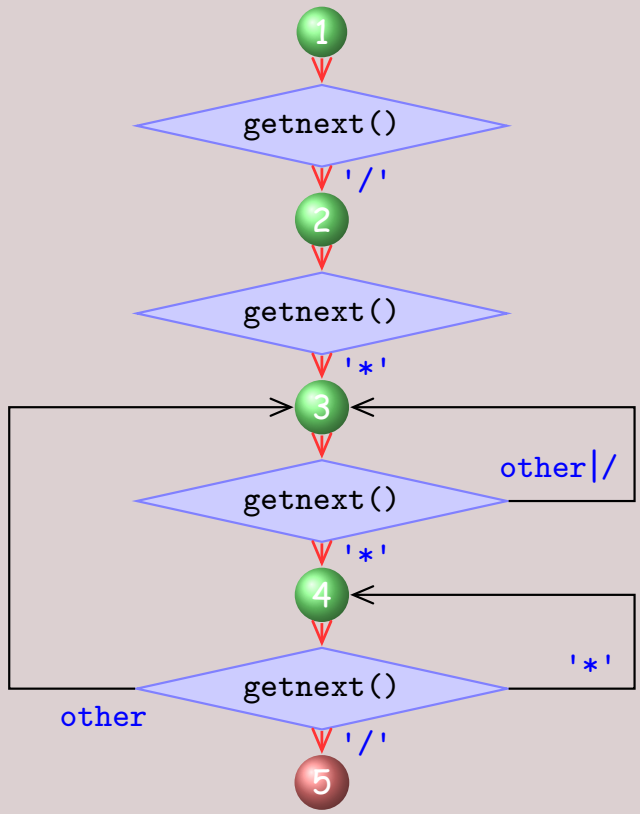
Automata-based programming

Automata-based programming is a programming paradigm in which the program or its part is thought of as a model of a finite state machine or any other (often more complicated) formal automaton.

基于自动机的编程时一种编程范式，用于问题可以被考虑为一种有限自动机的情形

Example: DFA 在程序设计中的应用 (2/2)

状态图 ⟺ 流程图



C~语言源程序

```
void comment(void)
{
  int c;
  if (getnext()=='/' && getnext()=='*')
    for ( ; ; ) { /*  state 3 */
      while ((c=getnext())!='*' && (c!=EOI));
      if ( c == '*' ) {
        while ( (c = getnext()) == '*' ) ;
                                /*  state 4 */
        if ( c == '/' )
          break; /*  state 5 */
        }
      if ( c == EOI ) {
        error("end of input in comments!");
        break;
        }
      }   /*  state 3 */
}
```

1 基本概念
- 定义
- 分析方法
- 正则表达式

2 确定的有限自动机
- 基本概念
- DFA 的设计
- DFA 算法

3 不确定的有限自动机
- 基本概念
- 正则表达式到 NFA 的转换
- NFA 到 DFA 的转换
- DFA 状态最小化

4 相关问题

©hfwang

基本概念

问题

正则表达式 (RE) 描述词法，DFA 算法实现，如何实现：RE $\Longrightarrow$ DFA

- 描述能力是否相同？
- 能否自动转换？

Example：手工设计识别正则表达式 (a | b)*abb 所表示语言的 DFA

基本概念

问题

正则表达式 (RE) 描述词法，DFA 算法实现，如何实现：RE $\Longrightarrow$ DFA

- 描述能力是否相同？
- 能否自动转换？

Example: 手工设计识别正则表达式 (a | b)*abb 所表示语言的 DFA

基本概念

问题

正则表达式 (RE) 描述词法，DFA 算法实现，如何实现：RE $\Longrightarrow$ DFA

- 描述能力是否相同？
- 能否自动转换？

Example: 手工设计识别正则表达式 (a | b)*abb 所表示语言的 DFA

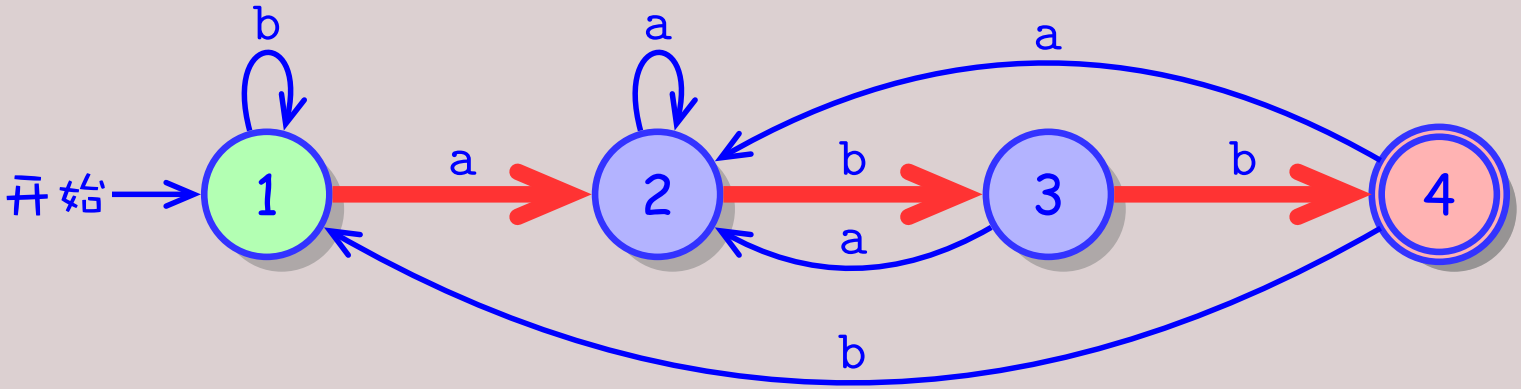
基本概念

问题

正则表达式 (RE) 描述词法，DFA 算法实现，如何实现：RE $\Longrightarrow$ DFA

- 描述能力是否相同？
- 能否自动转换？

Example：手工设计识别正则表达式 (a | b)*abb 所表示语言的 DFA

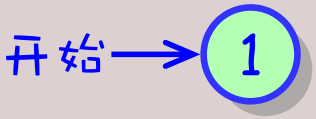


自动转换程序所遇到的困难

Example: (a | b)*abb 到 DFA 自动转换程序的设计思路

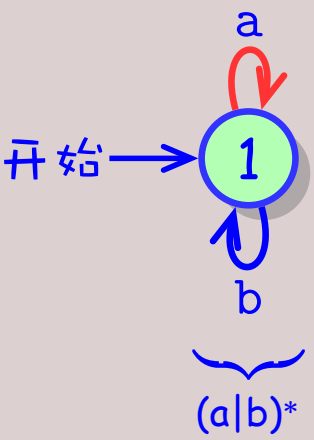
自动转换程序所遇到的困难

Example: (a | b)*abb 到 DFA 自动转换程序的设计思路



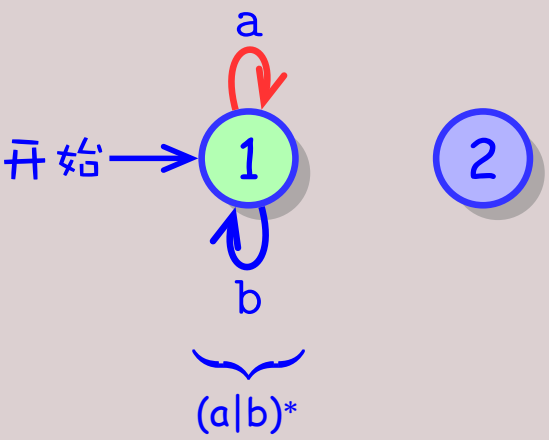
自动转换程序所遇到的困难

Example: (a | b)*abb 到 DFA 自动转换程序的设计思路



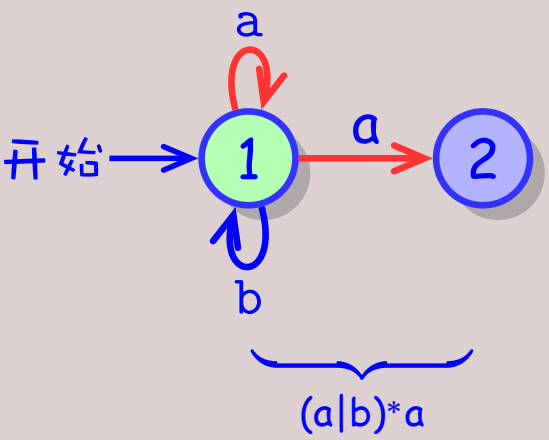
自动转换程序所遇到的困难

Example: (a | b)*abb 到 DFA 自动转换程序的设计思路



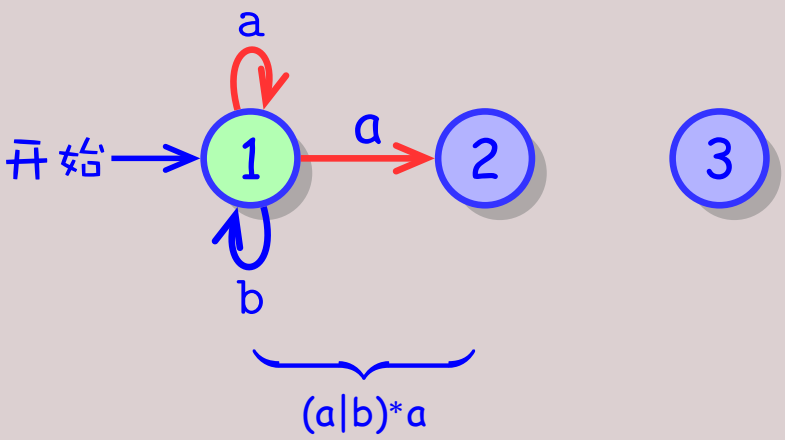
自动转换程序所遇到的困难

Example: (a | b)*abb 到 DFA 自动转换程序的设计思路



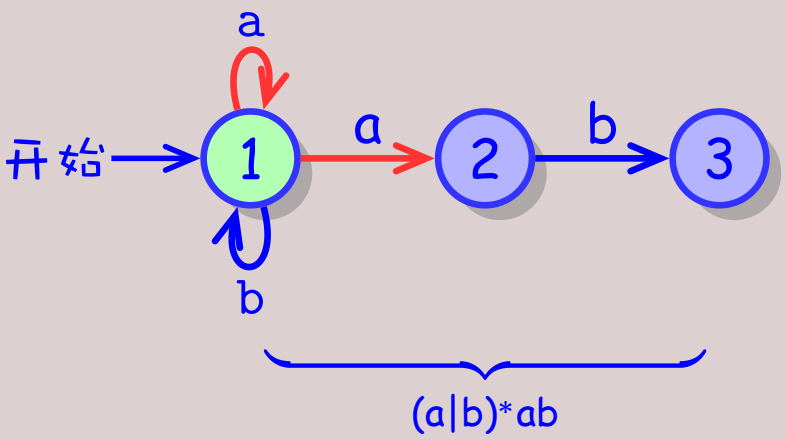
自动转换程序所遇到的困难

Example: (a | b)*abb 到 DFA 自动转换程序的设计思路



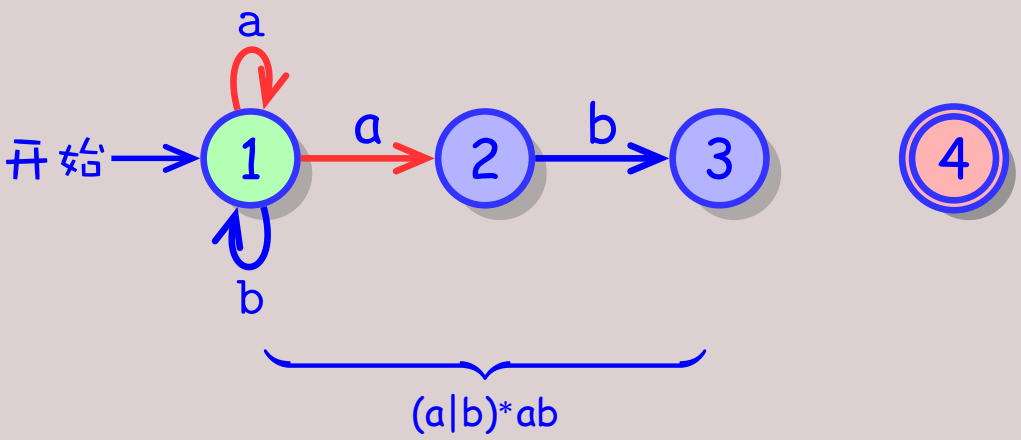
自动转换程序所遇到的困难

Example: (a | b)*abb 到 DFA 自动转换程序的设计思路



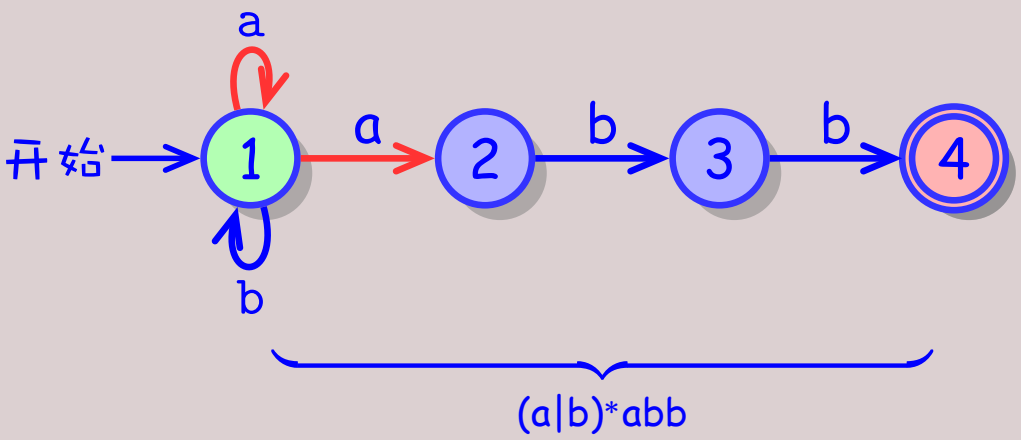
自动转换程序所遇到的困难

Example: (a | b)*abb 到 DFA 自动转换程序的设计思路



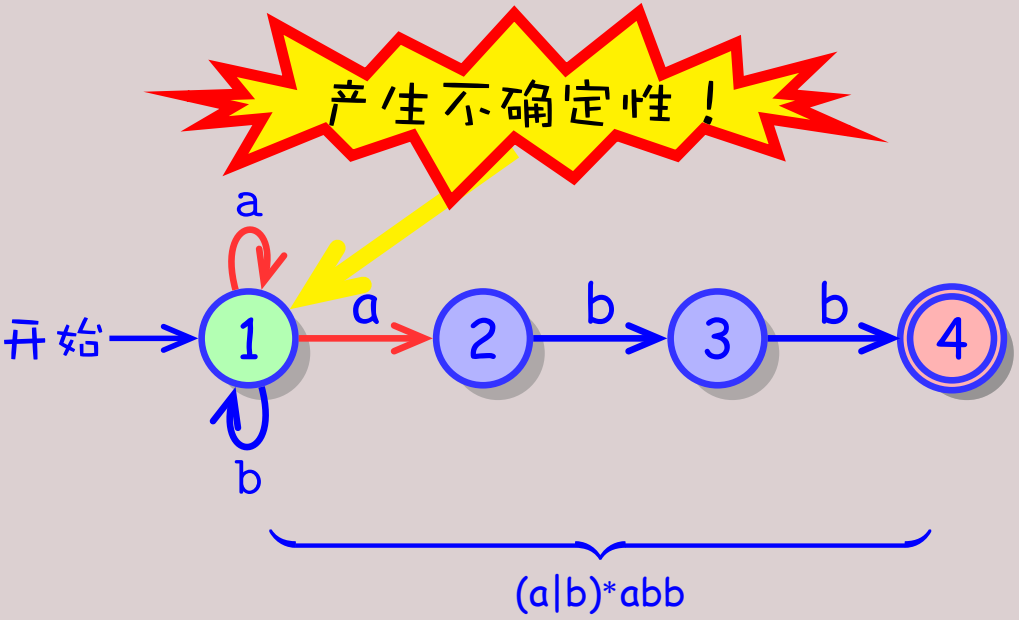
自动转换程序所遇到的困难

Example: (a | b)*abb 到 DFA 自动转换程序的设计思路



自动转换程序所遇到的困难

Example: (a | b)*abb 到 DFA 自动转换程序的设计思路



不确定的有限自动机 (Nondeterministic Finite Automata).

NFA 的定义

$M = \langle \Sigma, S, Move, s_0, F \rangle$, 其中:

- Σ 是字母表.
- S 是有限的的状态集合.
- $Move$ 是 $S \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(S)$.$\langle s, a \rangle \mapsto T \subseteq S$ 的部分函数, 称之为状态转换函数(状态转换图).
- $s_0 \in S$ 是开始状态.
- $F \subseteq S$ 称之为终止状态集合.

Example

$\Sigma = \{a, b\}$, $S = \{1, 2, 3, 4\}$ $s_0 = 1$, $F = \{4\}$, Move:

	a	b	ε
1	1, 2	1	/
2	/	2	/
3	/	4	/
3	/	/	/

状态转换图

不确定的有限自动机 (Nondeterministic Finite Automata).

NFA 的定义

$M = \langle \Sigma, S, Move, s_0, F \rangle$, 其中:

- Σ 是字母表.
- S 是有限的的状态集合.
- $Move$ 是 $S \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(S)$.$\langle s, a \rangle \mapsto T \subseteq S$ 的部分函数, 称之为状态转换函数(状态转换图).
- $s_0 \in S$ 是开始状态.
- $F \subseteq S$ 称之为终止状态集合.

Example

$\Sigma = \{a, b\}$, $S = \{1, 2, 3, 4\}$ $s_0 = 1$, $F = \{4\}$, Move:

	a	b	ε
1	1,2	1	/
2	/	2	/
3	/	4	/
3	/	/	/

状态转换图

不确定的有限自动机 (Nondeterministic Finite Automata).

NFA 的定义

$M = \langle \Sigma, S, Move, s_0, F \rangle$, 其中:

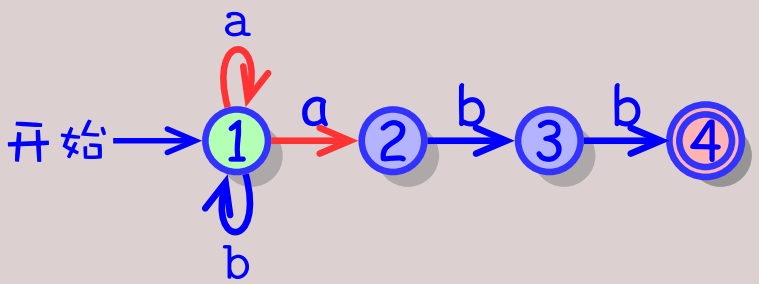
- Σ 是字母表.
- S 是有限的的状态集合.
- $Move$ 是 $S \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(S)$.$\langle s, a \rangle \mapsto T \subseteq S$ 的部分函数, 称之为状态转换函数(状态转换图).
- $s_0 \in S$ 是开始状态.
- $F \subseteq S$ 称之为终止状态集合.

Example

$\Sigma = \{a, b\}$, $S = \{1, 2, 3, 4\}$ $s_0 = 1$, $F = \{4\}$, Move:

	a	b	ε
1	1,2	1	/
2	/	2	/
3	/	4	/
3	/	/	/

状态转换图



NFA 所识别的语言

定义

设 M 是一个 NFA，M 所识别的语言：

$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_m, (s_i \in S \wedge s_m \in F) \wedge s_0 \xrightarrow{y_1} s_1 \xrightarrow{y_2} s_2 \cdots s_{m-1} \xrightarrow{y_m} s_m,\ y_i \in \Sigma \cup \{\varepsilon\},\ s_{i+1} \in Move(s_i, y_i),\ y_1y_2\cdots y_m = x_1x_2\cdots x_n \}$$

Example：有两条路经接受 ``abb''

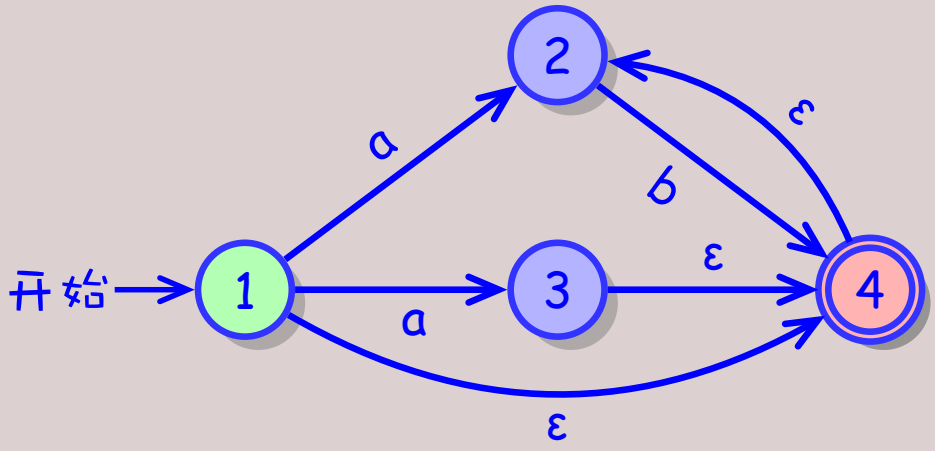
NFA 所识别的语言

定义

设 M 是一个 NFA，M 所识别的语言：

$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_m, (s_i \in S \wedge s_m \in F) \wedge s_0 \xrightarrow{y_1} s_1 \xrightarrow{y_2} s_2 \cdots s_{m-1} \xrightarrow{y_m} s_m,\ y_i \in \Sigma \cup \{\varepsilon\},\ s_{i+1} \in Move(s_i, y_i),\ y_1y_2\cdots y_m = x_1x_2\cdots x_n \}$$

Example：有两条路经接受 ``abb''



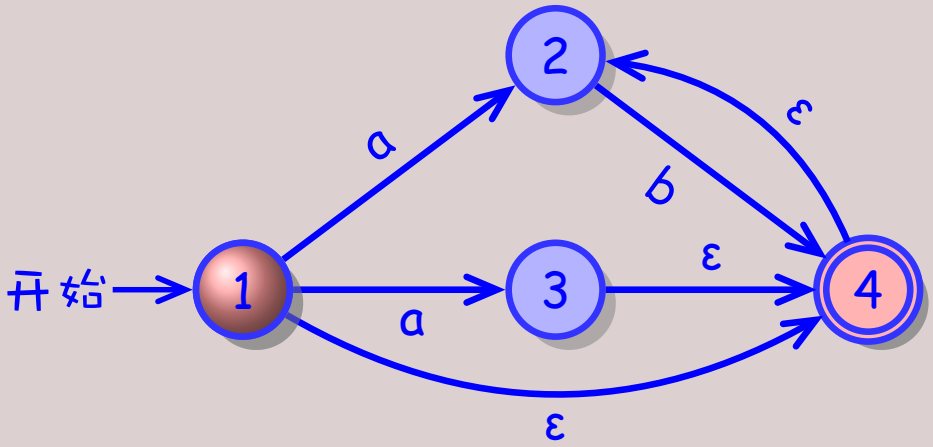
NFA 所识别的语言

定义

设 M 是一个 NFA，M 所识别的语言：

$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_m, (s_i \in S \wedge s_m \in F) \wedge s_0 \xrightarrow{y_1} s_1 \xrightarrow{y_2} s_2 \cdots s_{m-1} \xrightarrow{y_m} s_m,\ y_i \in \Sigma \cup \{\varepsilon\},\ s_{i+1} \in Move(s_i, y_i),\ y_1y_2\cdots y_m = x_1x_2\cdots x_n \}$$

Example：有两条路经接受 ``abb''



(1)：开始 → 1

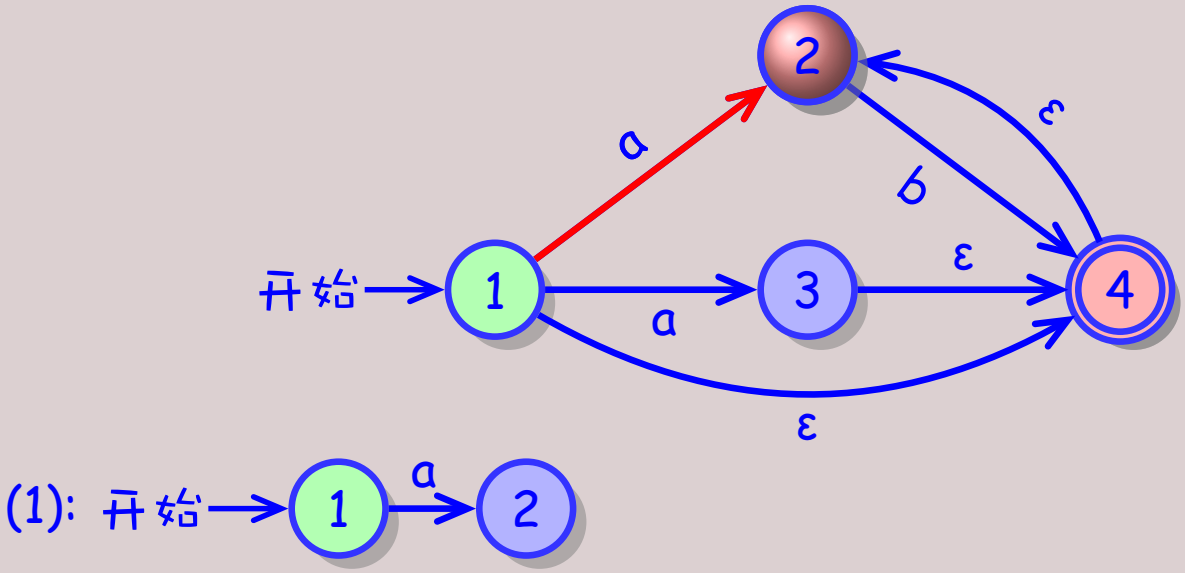
NFA 所识别的语言

定义

设 M 是一个 NFA，M 所识别的语言：

$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_m, (s_i \in S \wedge s_m \in F) \wedge s_0 \xrightarrow{y_1} s_1 \xrightarrow{y_2} s_2 \cdots s_{m-1} \xrightarrow{y_m} s_m,\ y_i \in \Sigma \cup \{\varepsilon\},\ s_{i+1} \in Move(s_i, y_i),\ y_1y_2\cdots y_m = x_1x_2\cdots x_n \}$$

Example：有两条路经接受 ``abb''



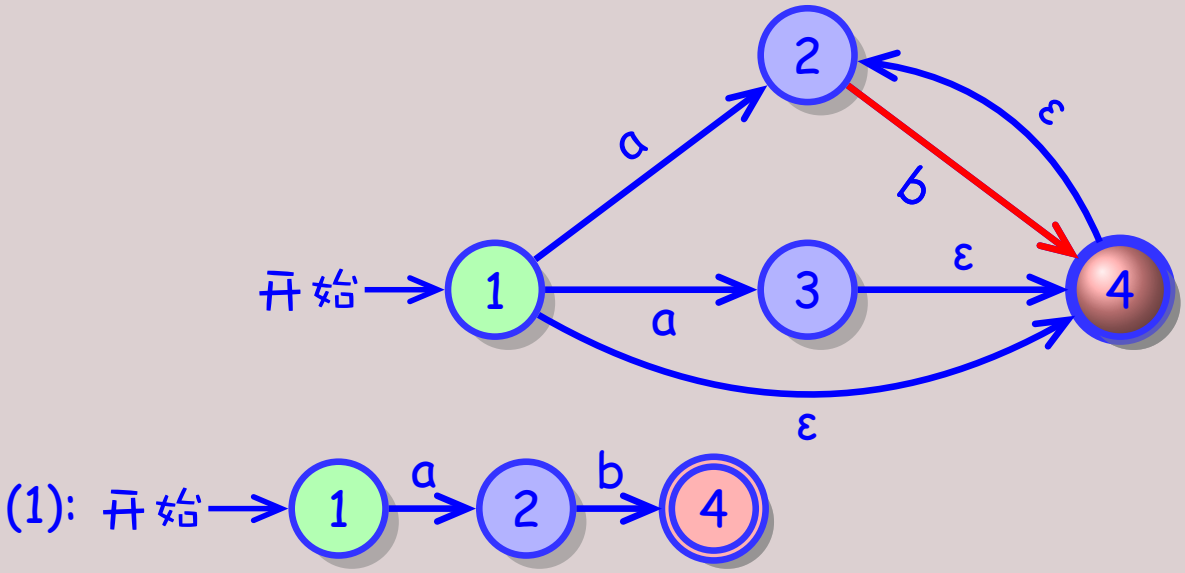
NFA 所识别的语言

定义

设 M 是一个 NFA，M 所识别的语言：

$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_m, (s_i \in S \wedge s_m \in F) \wedge s_0 \xrightarrow{y_1} s_1 \xrightarrow{y_2} s_2 \cdots s_{m-1} \xrightarrow{y_m} s_m,\ y_i \in \Sigma \cup \{\varepsilon\},\ s_{i+1} \in Move(s_i, y_i),\ y_1y_2\cdots y_m = x_1x_2\cdots x_n \}$$

Example：有两条路经接受 ``abb''



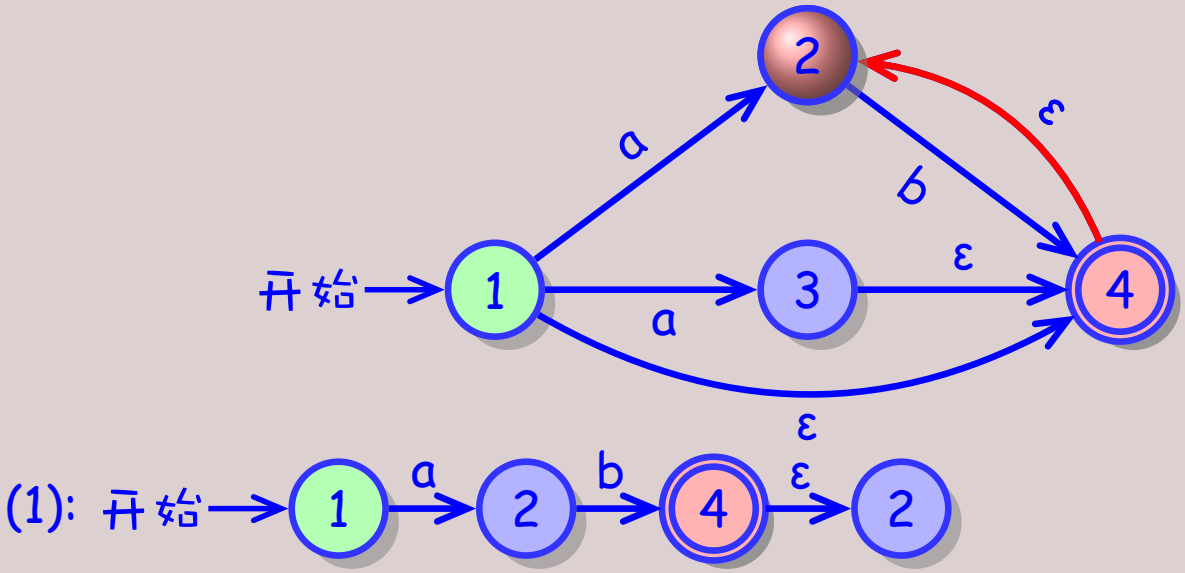
NFA 所识别的语言

定义

设 M 是一个 NFA，M 所识别的语言：

$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_m, (s_i \in S \wedge s_m \in F) \wedge s_0 \xrightarrow{y_1} s_1 \xrightarrow{y_2} s_2 \cdots s_{m-1} \xrightarrow{y_m} s_m,\ y_i \in \Sigma \cup \{\varepsilon\},\ s_{i+1} \in Move(s_i, y_i),\ y_1y_2\cdots y_m = x_1x_2\cdots x_n \}$$

Example：有两条路经接受 ``abb''



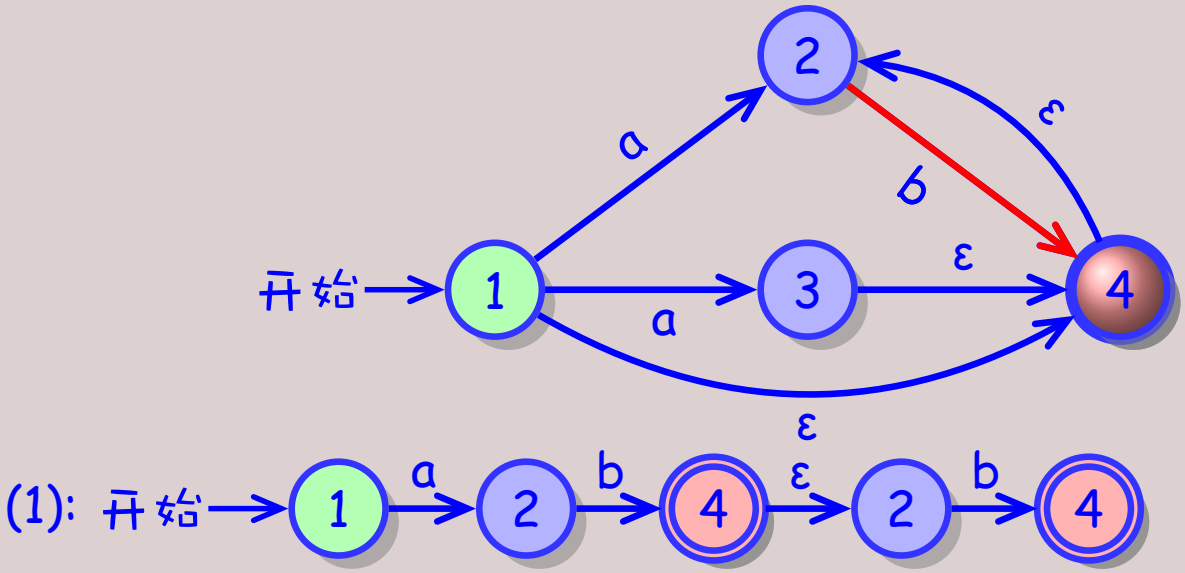
NFA 所识别的语言

定义

设 M 是一个 NFA，M 所识别的语言：

$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_m, (s_i \in S \wedge s_m \in F) \wedge s_0 \xrightarrow{y_1} s_1 \xrightarrow{y_2} s_2 \cdots s_{m-1} \xrightarrow{y_m} s_m,\ y_i \in \Sigma \cup \{\varepsilon\},\ s_{i+1} \in Move(s_i, y_i),\ y_1y_2\cdots y_m = x_1x_2\cdots x_n \}$$

Example：有两条路经接受 ``abb''



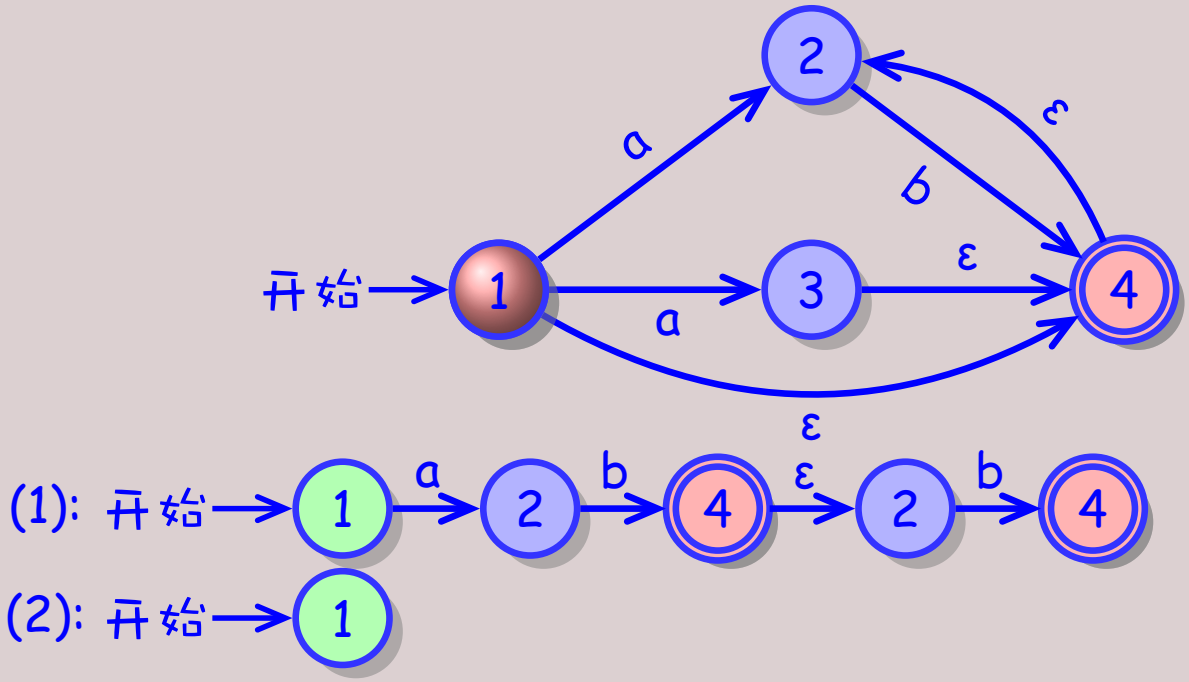
NFA 所识别的语言

定义

设 M 是一个 NFA，M 所识别的语言：

$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_m, (s_i \in S \wedge s_m \in F) \wedge s_0 \xrightarrow{y_1} s_1 \xrightarrow{y_2} s_2 \cdots s_{m-1} \xrightarrow{y_m} s_m,\ y_i \in \Sigma \cup \{\varepsilon\},\ s_{i+1} \in Move(s_i, y_i),\ y_1y_2\cdots y_m = x_1x_2\cdots x_n \}$$

Example：有两条路经接受 ``abb''



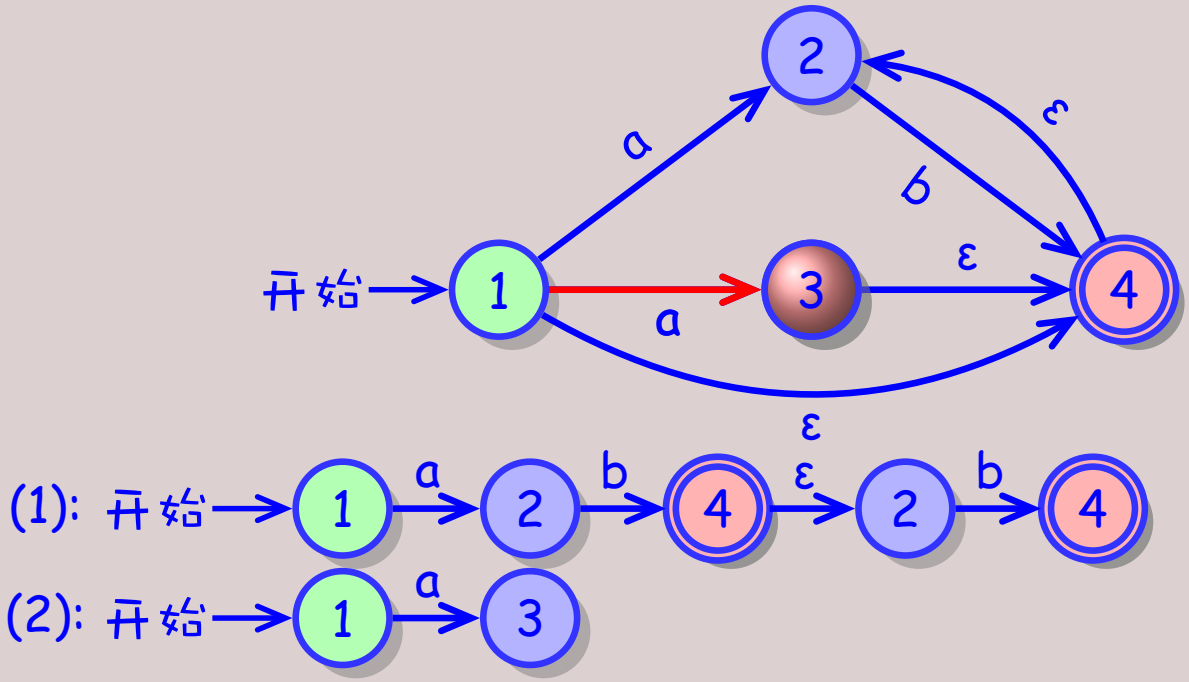
NFA 所识别的语言

定义

设 M 是一个 NFA，M 所识别的语言：

$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_m, (s_i \in S \wedge s_m \in F) \wedge s_0 \xrightarrow{y_1} s_1 \xrightarrow{y_2} s_2 \cdots s_{m-1} \xrightarrow{y_m} s_m,\ y_i \in \Sigma \cup \{\varepsilon\},\ s_{i+1} \in Move(s_i, y_i),\ y_1y_2\cdots y_m = x_1x_2\cdots x_n \}$$

Example：有两条路经接受 ``abb''



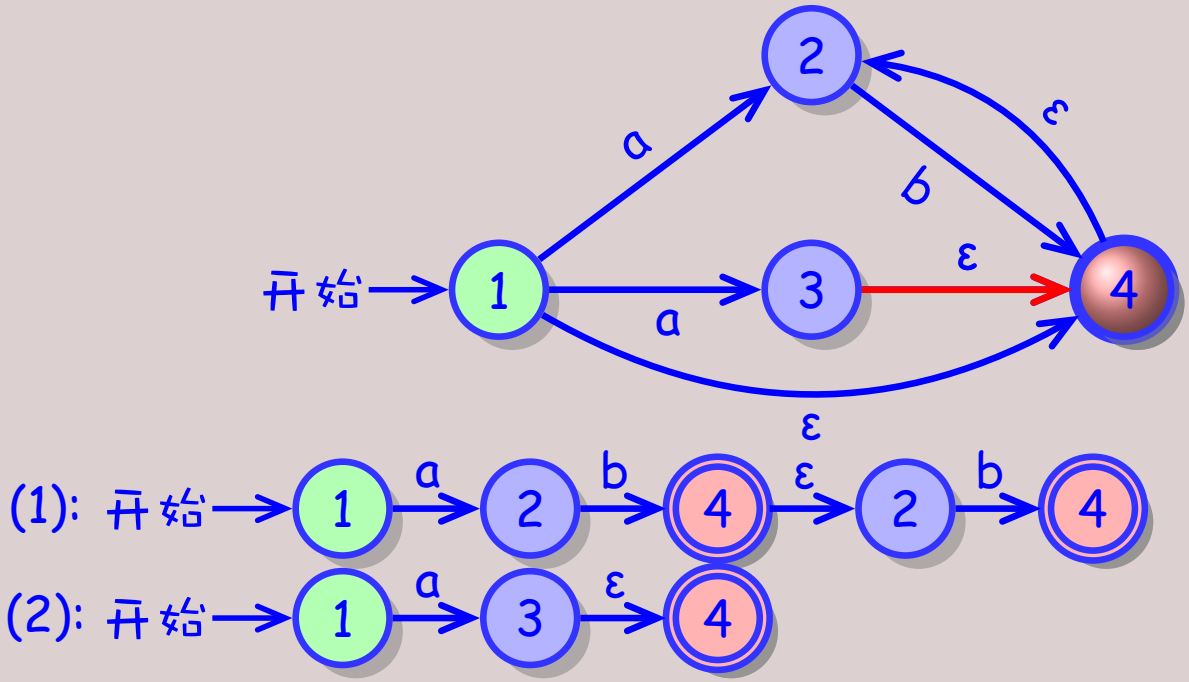
NFA 所识别的语言

定义

设 M 是一个 NFA，M 所识别的语言：

$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_m, (s_i \in S \wedge s_m \in F) \wedge s_0 \xrightarrow{y_1} s_1 \xrightarrow{y_2} s_2 \cdots s_{m-1} \xrightarrow{y_m} s_m,\ y_i \in \Sigma \cup \{\varepsilon\},\ s_{i+1} \in Move(s_i, y_i),\ y_1y_2\cdots y_m = x_1x_2\cdots x_n \}$$

Example：有两条路经接受 ``abb''



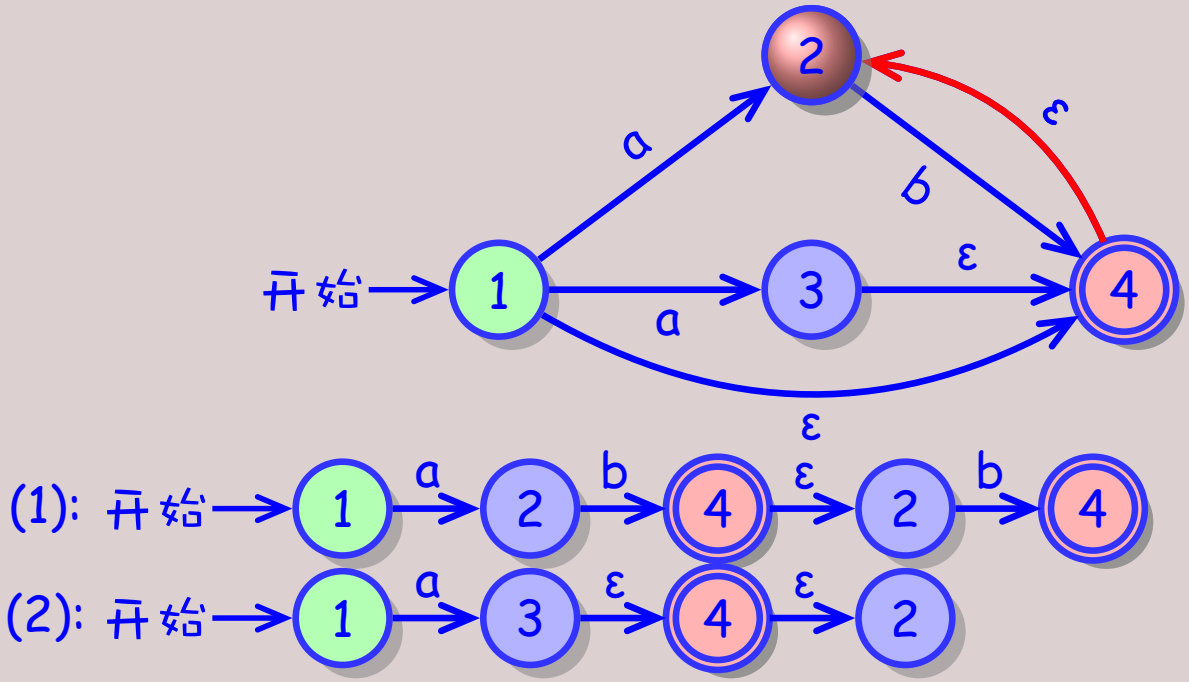
NFA 所识别的语言

定义

设 M 是一个 NFA，M 所识别的语言：

$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_m, (s_i \in S \wedge s_m \in F) \wedge s_0 \xrightarrow{y_1} s_1 \xrightarrow{y_2} s_2 \cdots s_{m-1} \xrightarrow{y_m} s_m,\ y_i \in \Sigma \cup \{\varepsilon\},\ s_{i+1} \in Move(s_i, y_i),\ y_1y_2\cdots y_m = x_1x_2\cdots x_n \}$$

Example：有两条路经接受 ``abb''



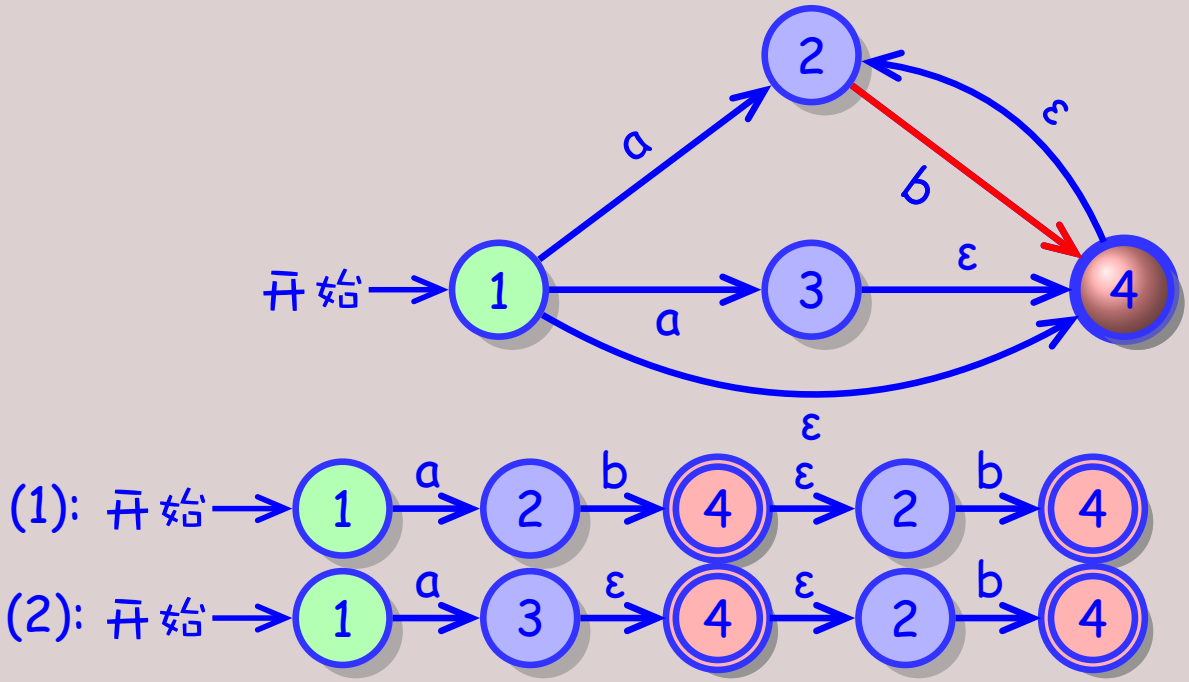
NFA 所识别的语言

定义

设 M 是一个 NFA，M 所识别的语言：

$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_m, (s_i \in S \wedge s_m \in F) \wedge s_0 \xrightarrow{y_1} s_1 \xrightarrow{y_2} s_2 \cdots s_{m-1} \xrightarrow{y_m} s_m,\ y_i \in \Sigma \cup \{\varepsilon\},\ s_{i+1} \in Move(s_i, y_i),\ y_1y_2\cdots y_m = x_1x_2\cdots x_n \}$$

Example：有两条路经接受 ``abb''



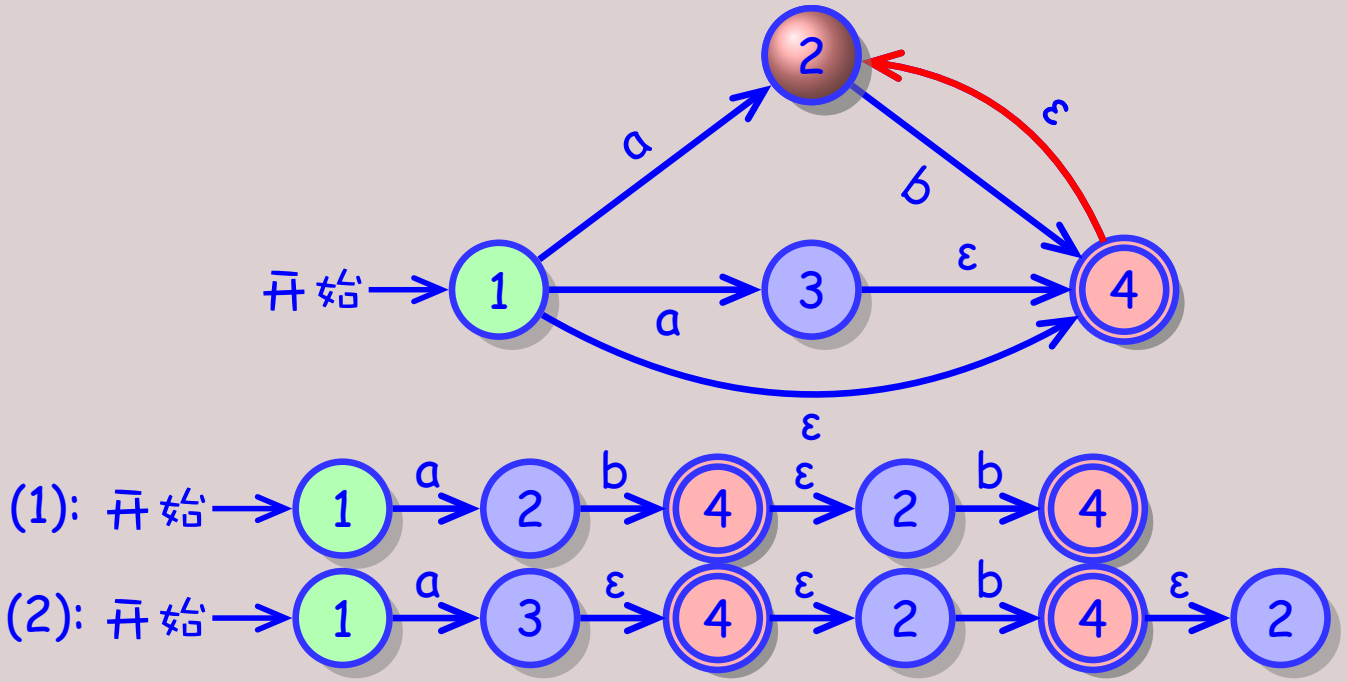
NFA 所识别的语言

定义

设 M 是一个 NFA，M 所识别的语言：

$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_m, (s_i \in S \wedge s_m \in F) \wedge s_0 \xrightarrow{y_1} s_1 \xrightarrow{y_2} s_2 \cdots s_{m-1} \xrightarrow{y_m} s_m,\ y_i \in \Sigma \cup \{\varepsilon\},\ s_{i+1} \in Move(s_i, y_i),\ y_1y_2\cdots y_m = x_1x_2\cdots x_n \}$$

Example：有两条路经接受 ``abb''



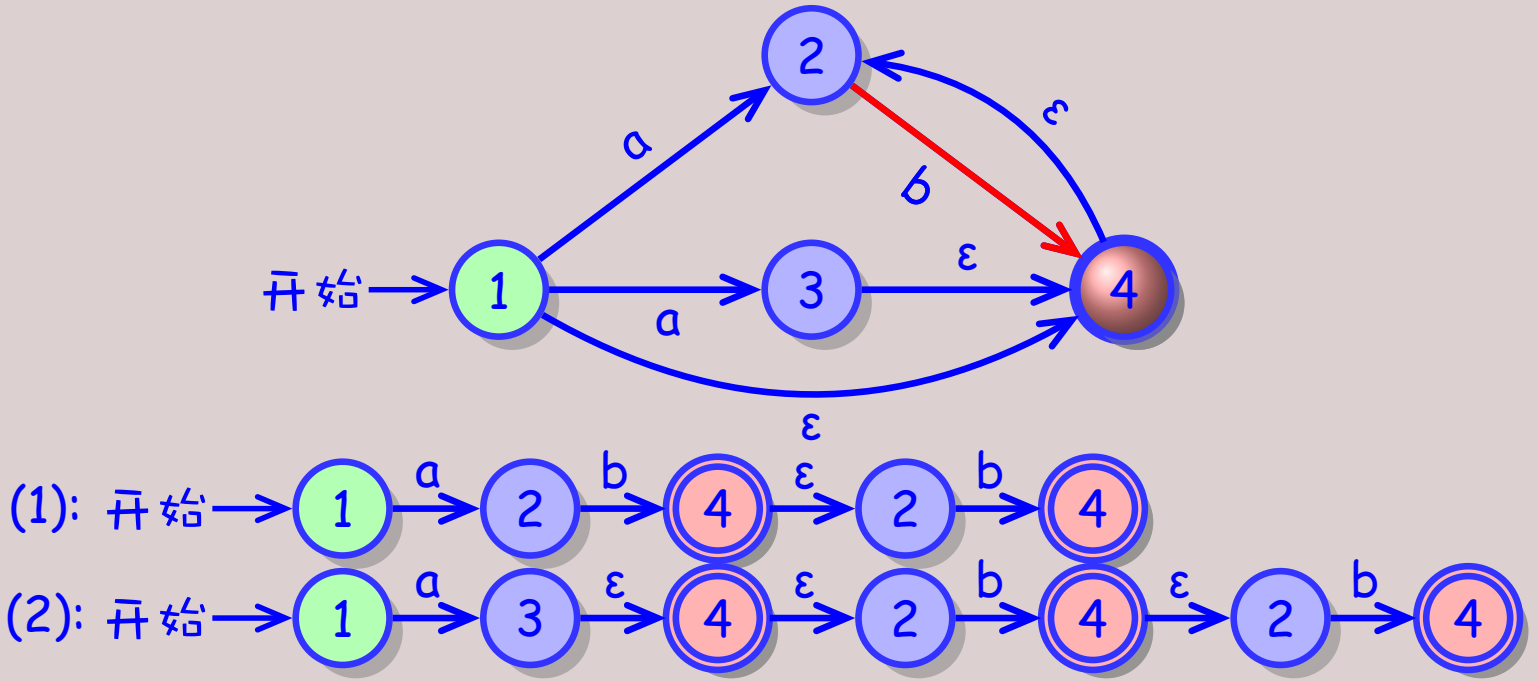
NFA 所识别的语言

定义

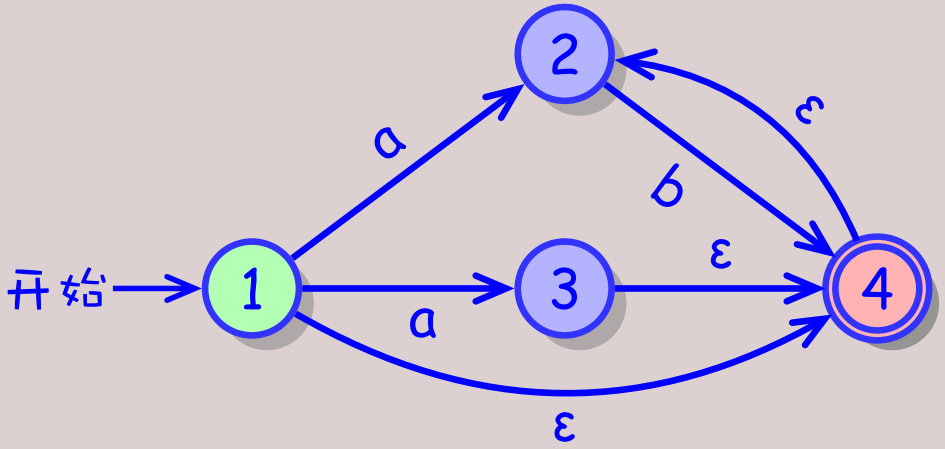
设 M 是一个 NFA，M 所识别的语言：

$$L(M) = \{ x_1x_2\cdots x_n \mid x_i \in \Sigma, i = 1,\cdots,n \wedge \exists s_1, s_2, \cdots, s_m, (s_i \in S \wedge s_m \in F) \wedge s_0 \xrightarrow{y_1} s_1 \xrightarrow{y_2} s_2 \cdots s_{m-1} \xrightarrow{y_m} s_m,\ y_i \in \Sigma \cup \{\varepsilon\},\ s_{i+1} \in Move(s_i, y_i),\ y_1y_2\cdots y_m = x_1x_2\cdots x_n \}$$

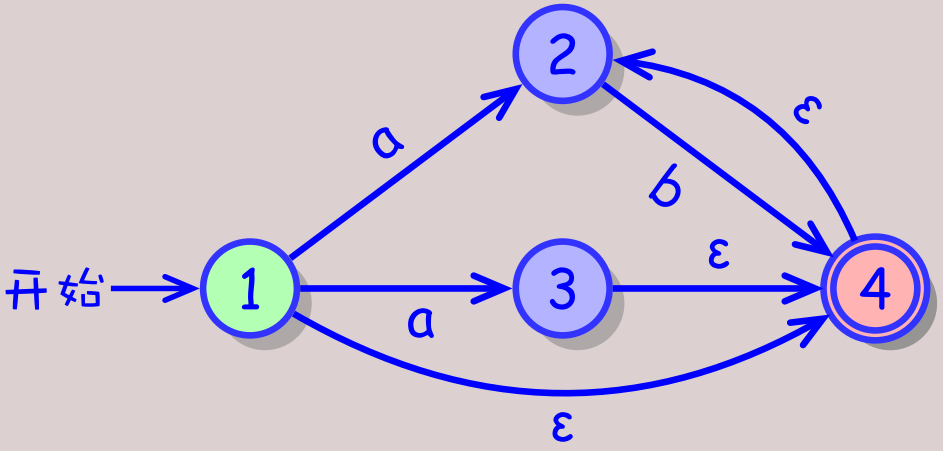
Example：有两条路经接受 ``abb''



Example：不接受 aba.

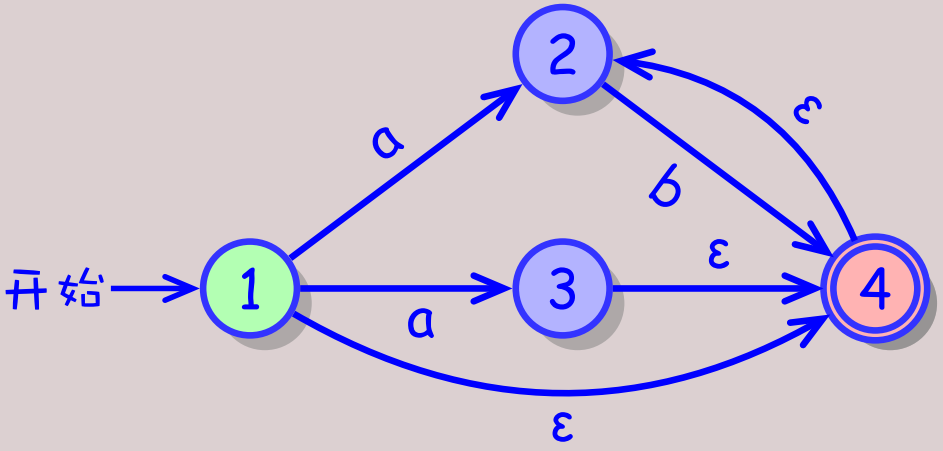


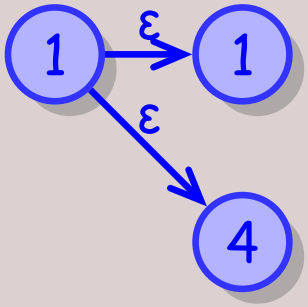
Example：不接受 aba.



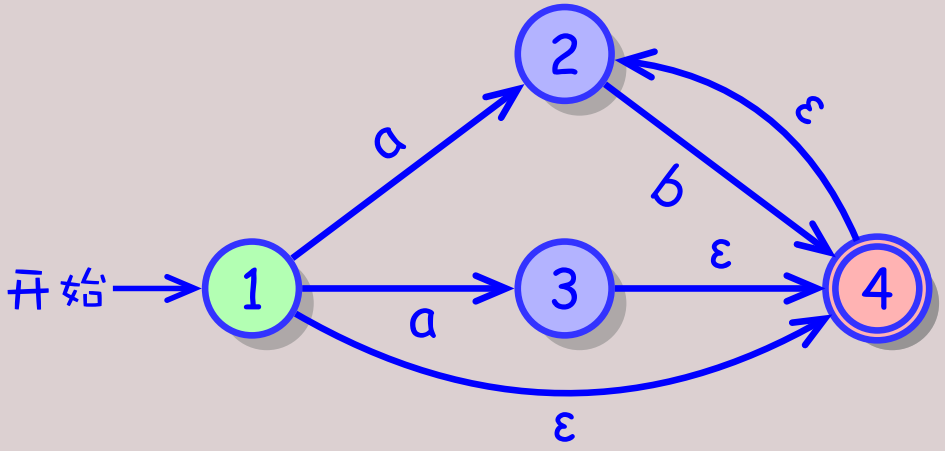
1

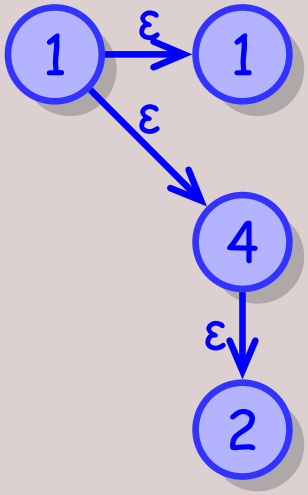
Example：不接受 aba.



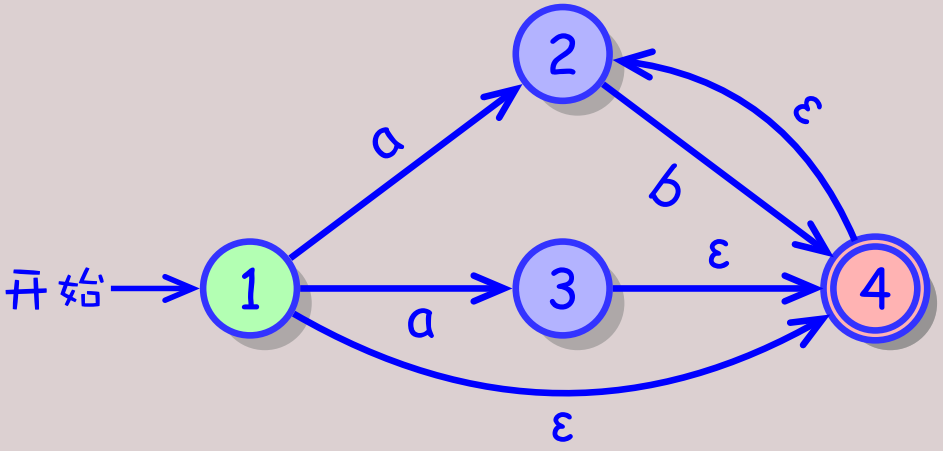


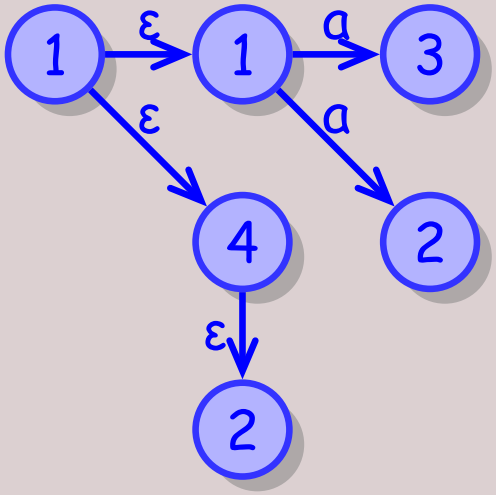
Example: 不接受 aba.



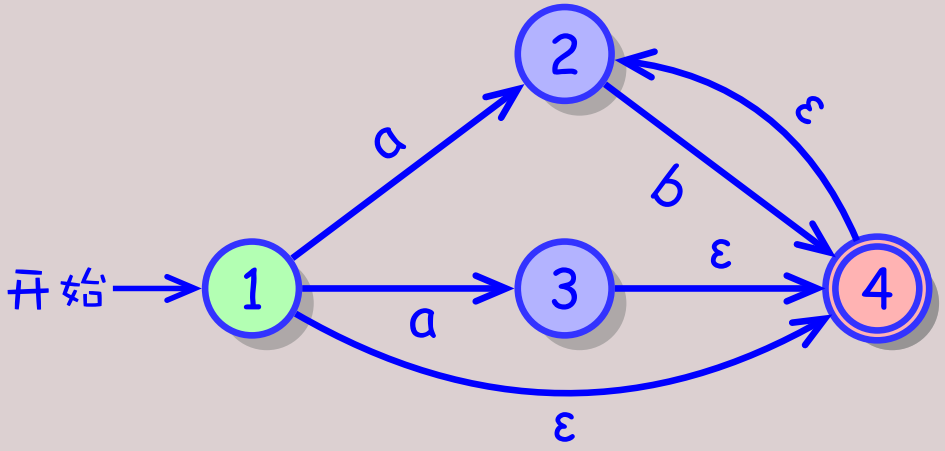


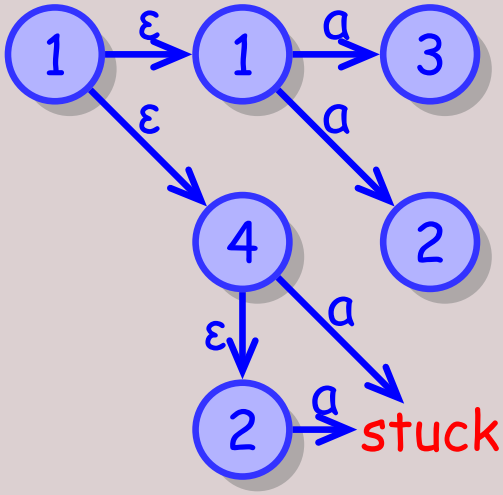
Example：不接受 aba.



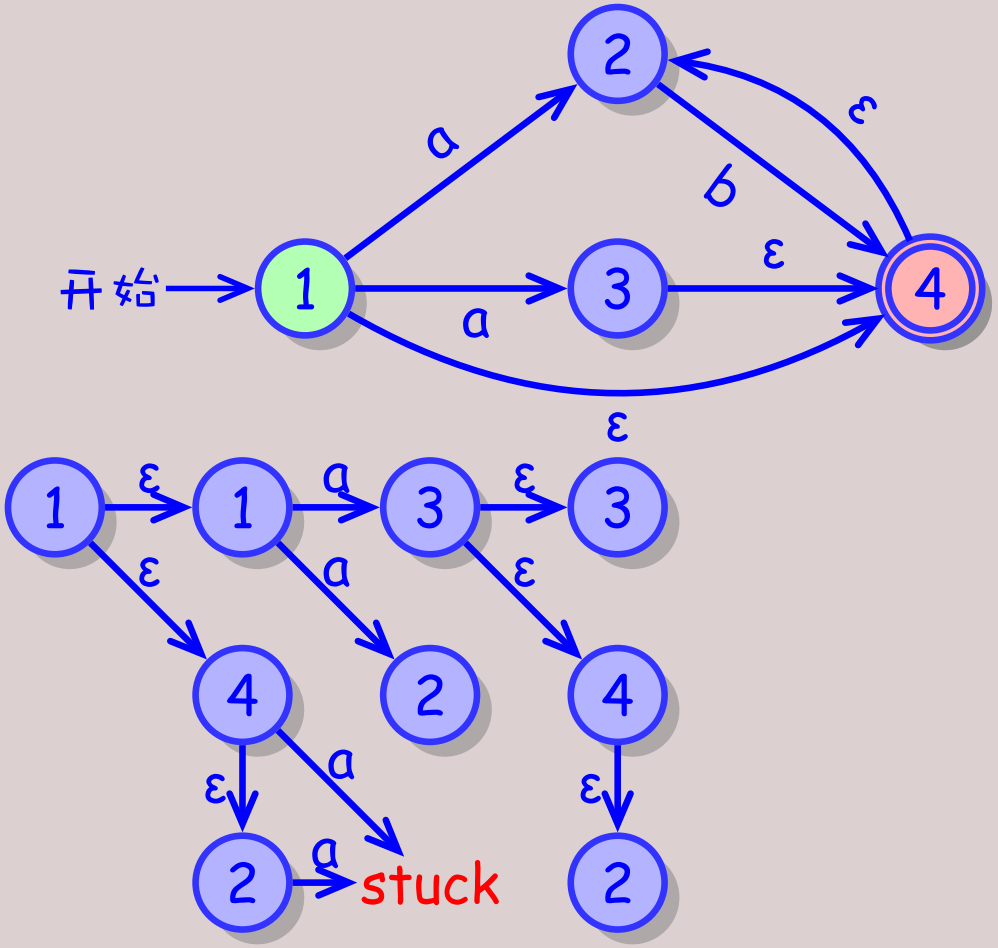


Example: 不接受 aba.

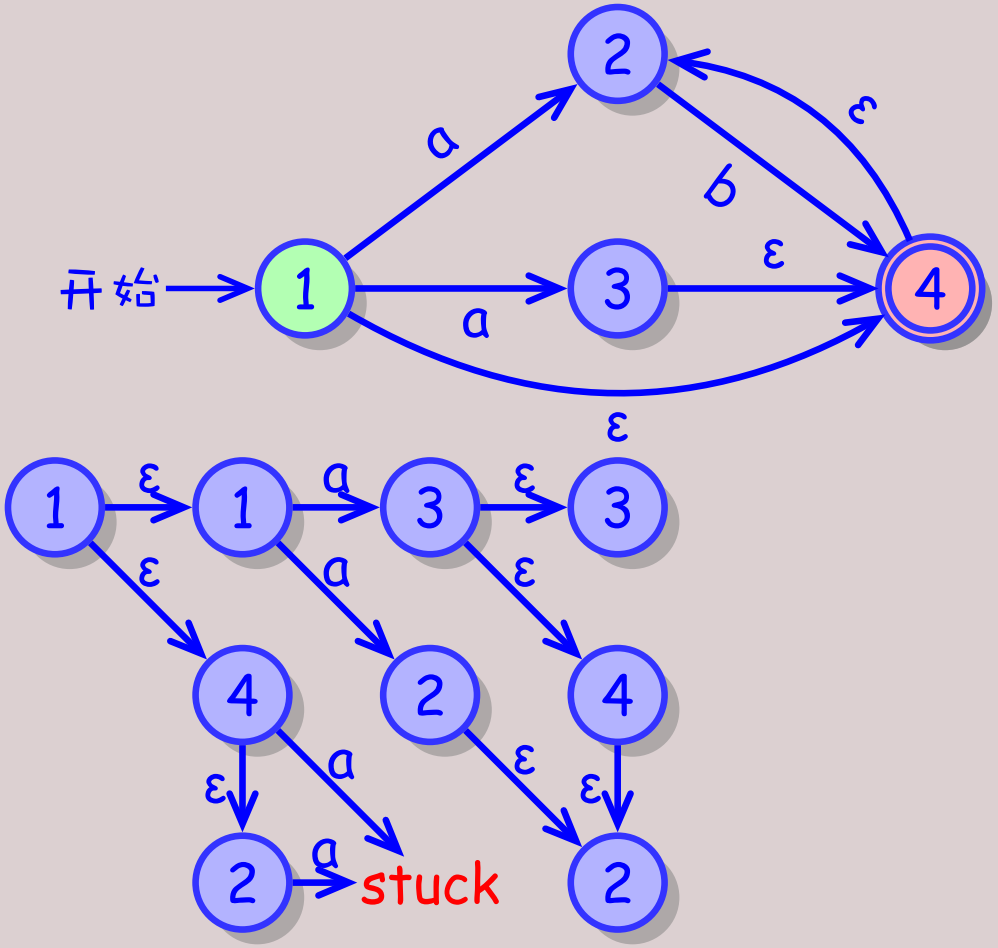




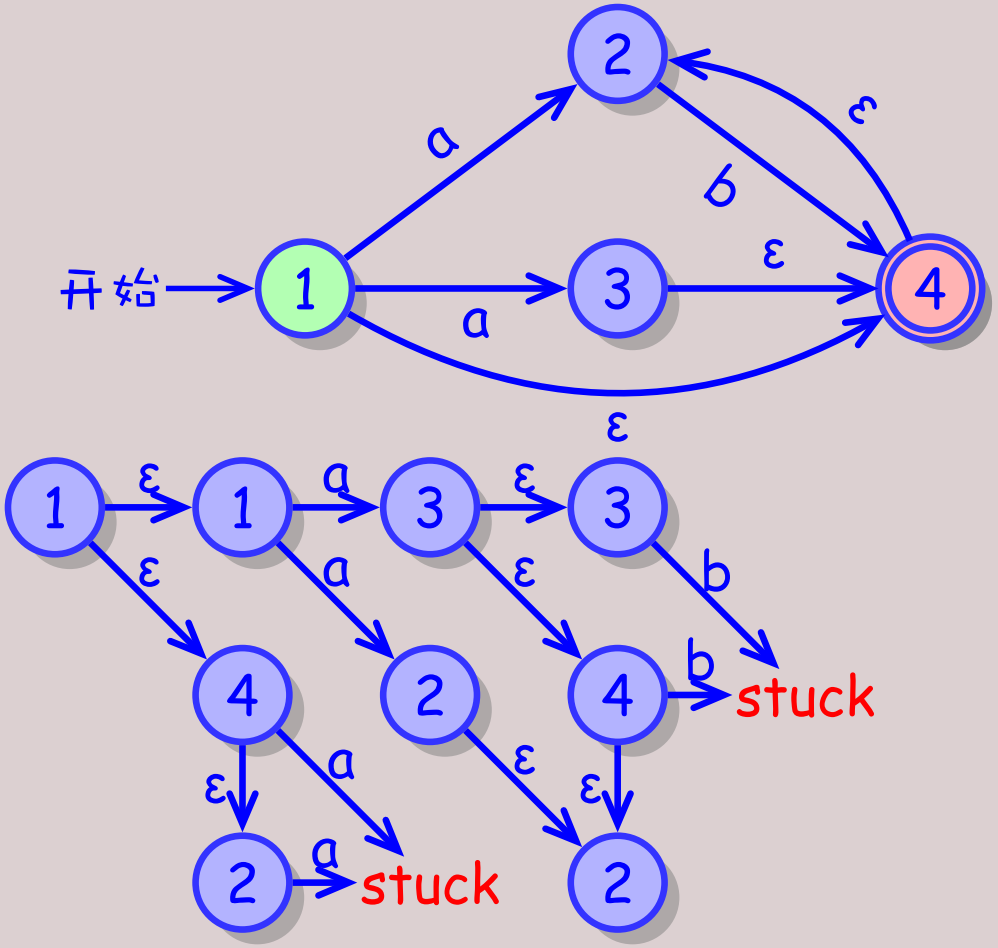
Example: 不接受 aba.



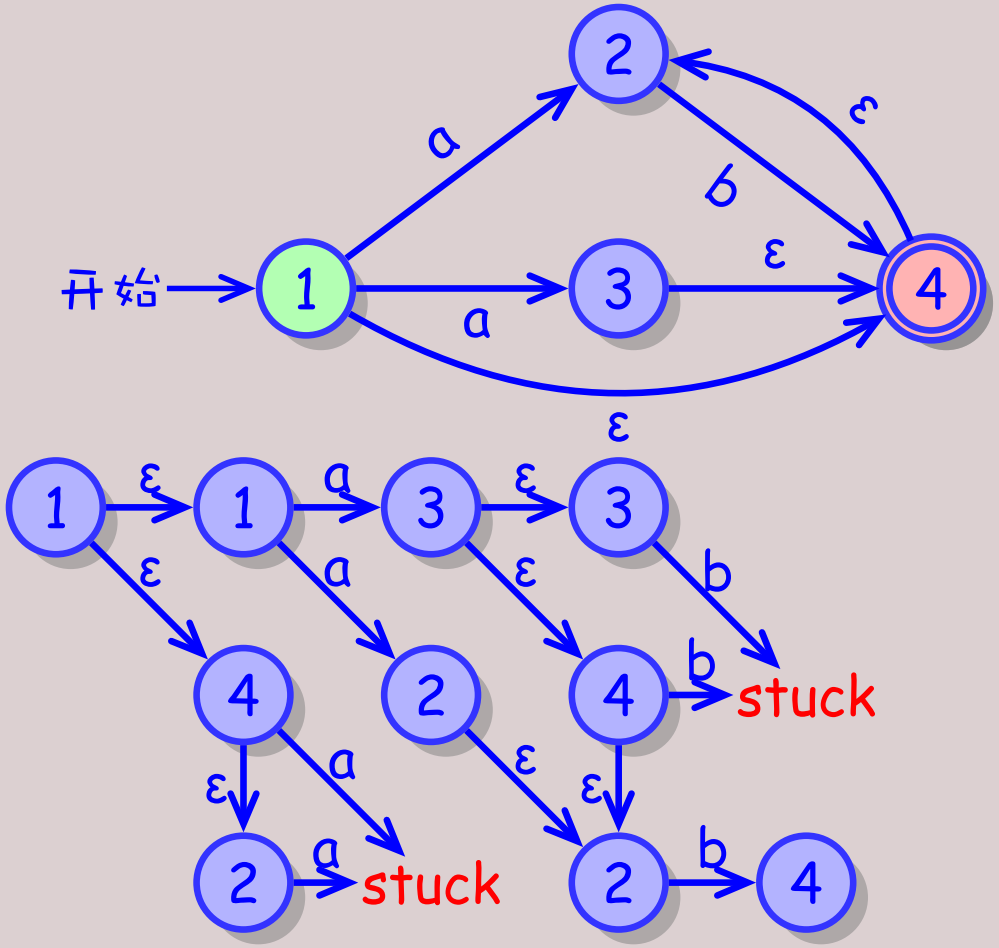
Example: 不接受 aba.



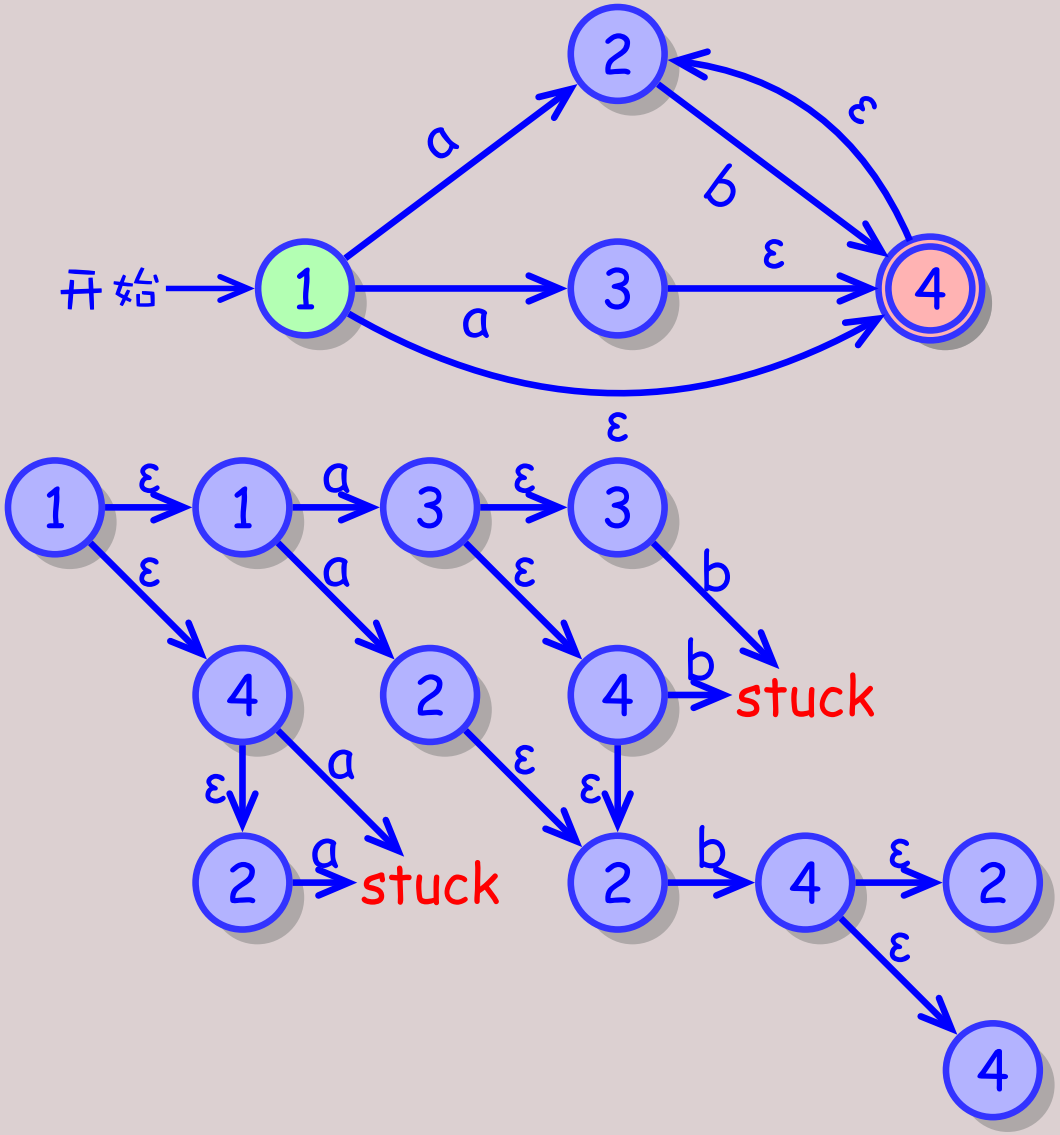
Example: 不接受 aba.



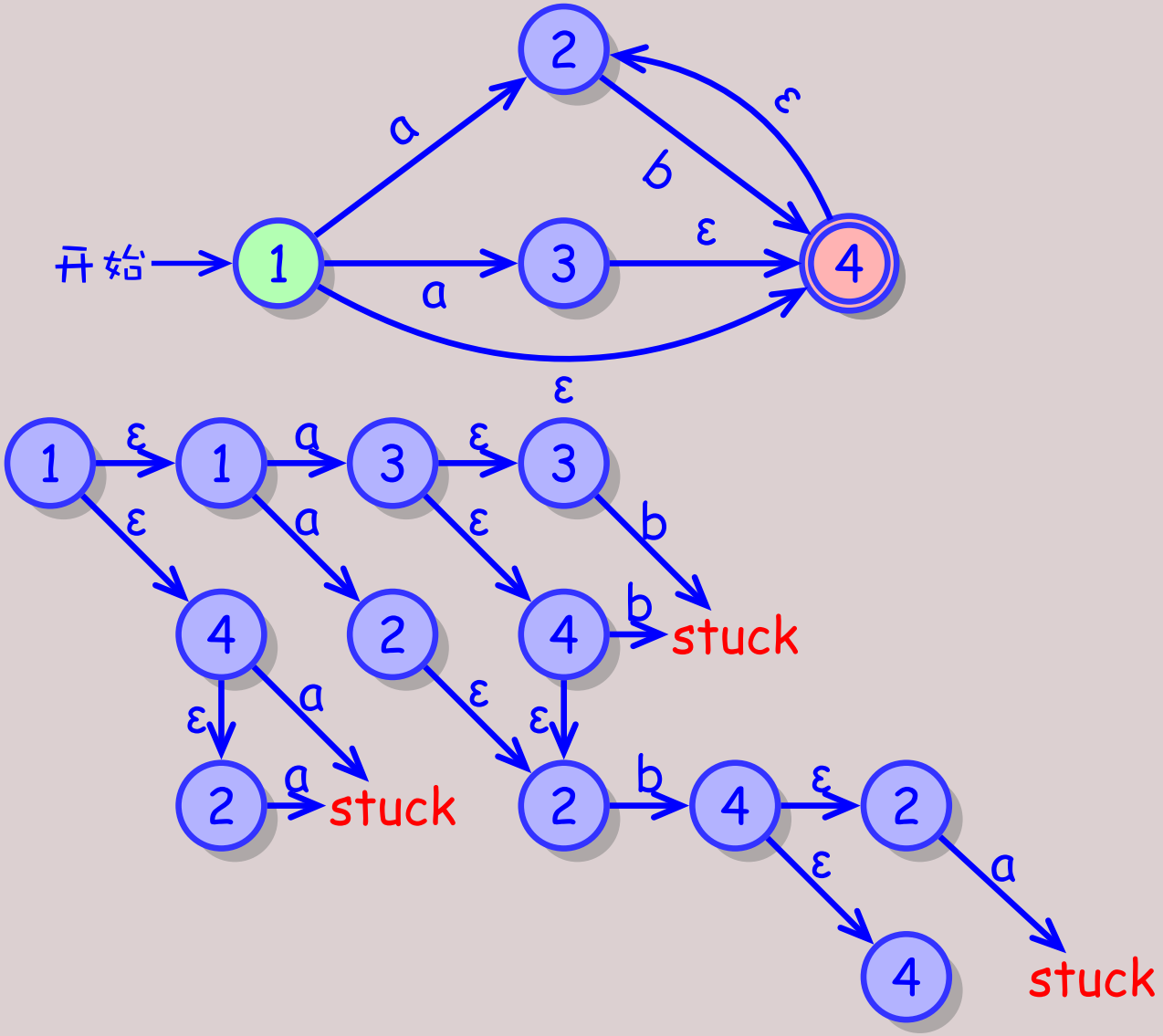
Example: 不接受 aba.



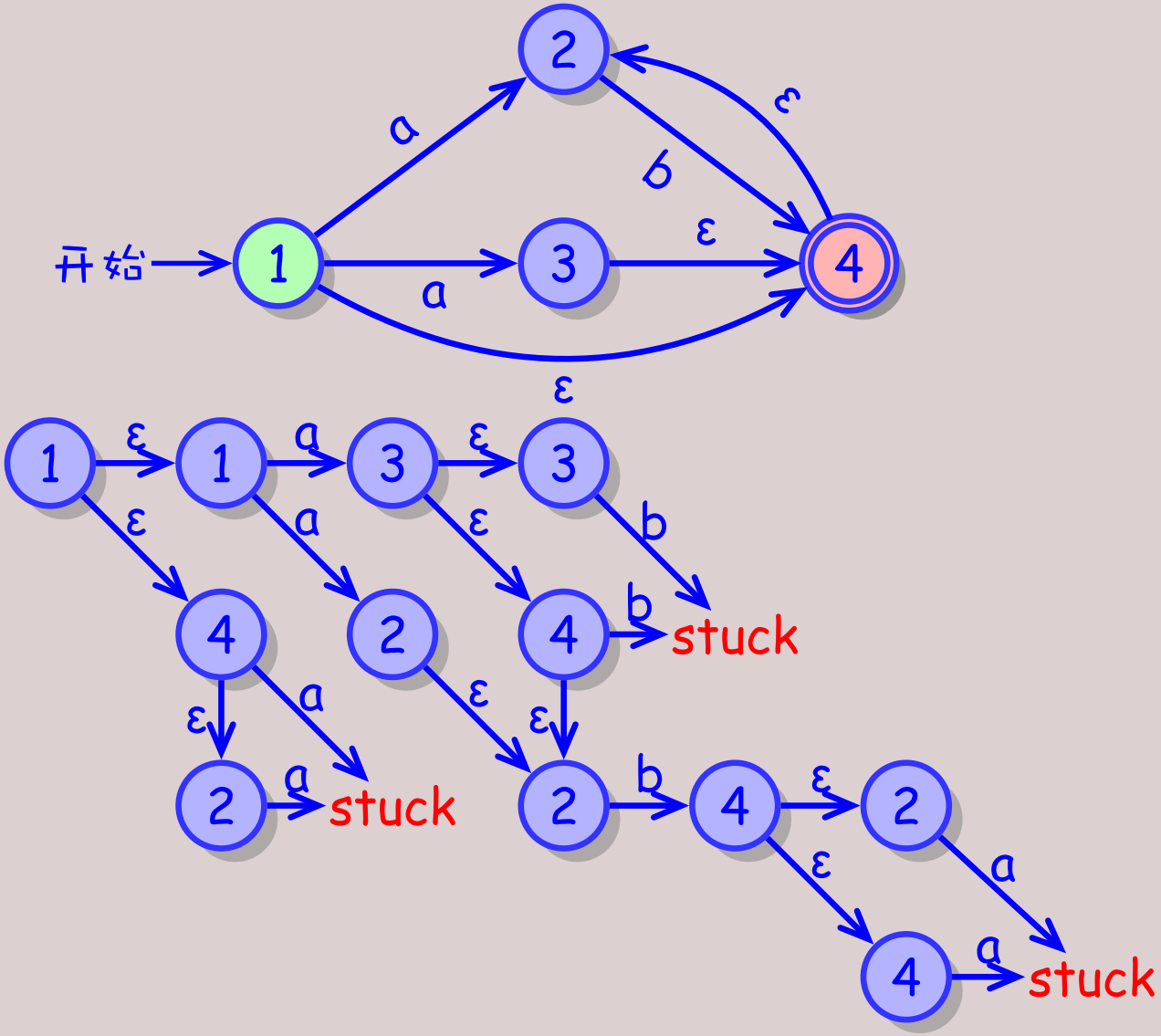
Example: 不接受 aba.



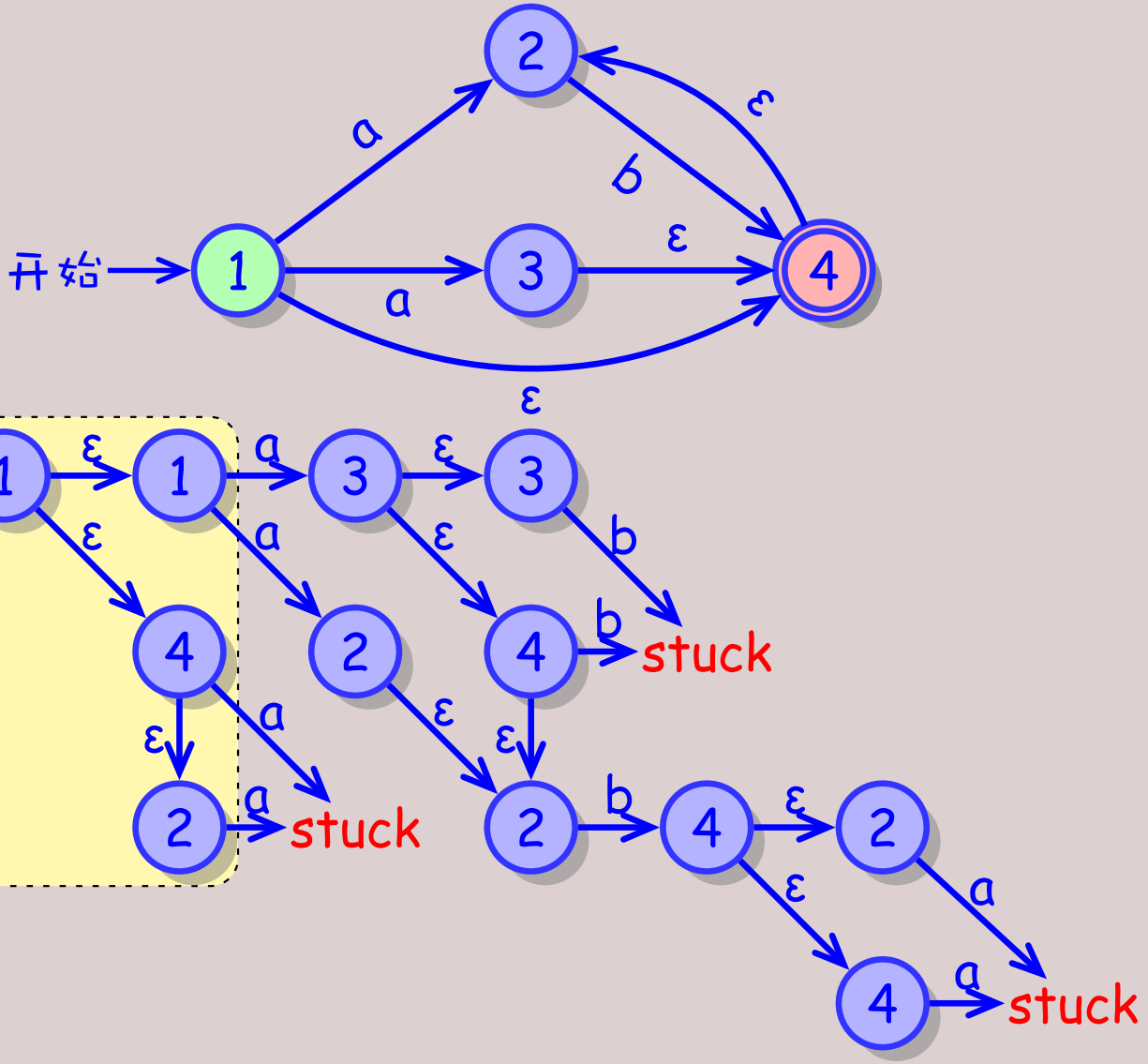
Example: 不接受 aba.



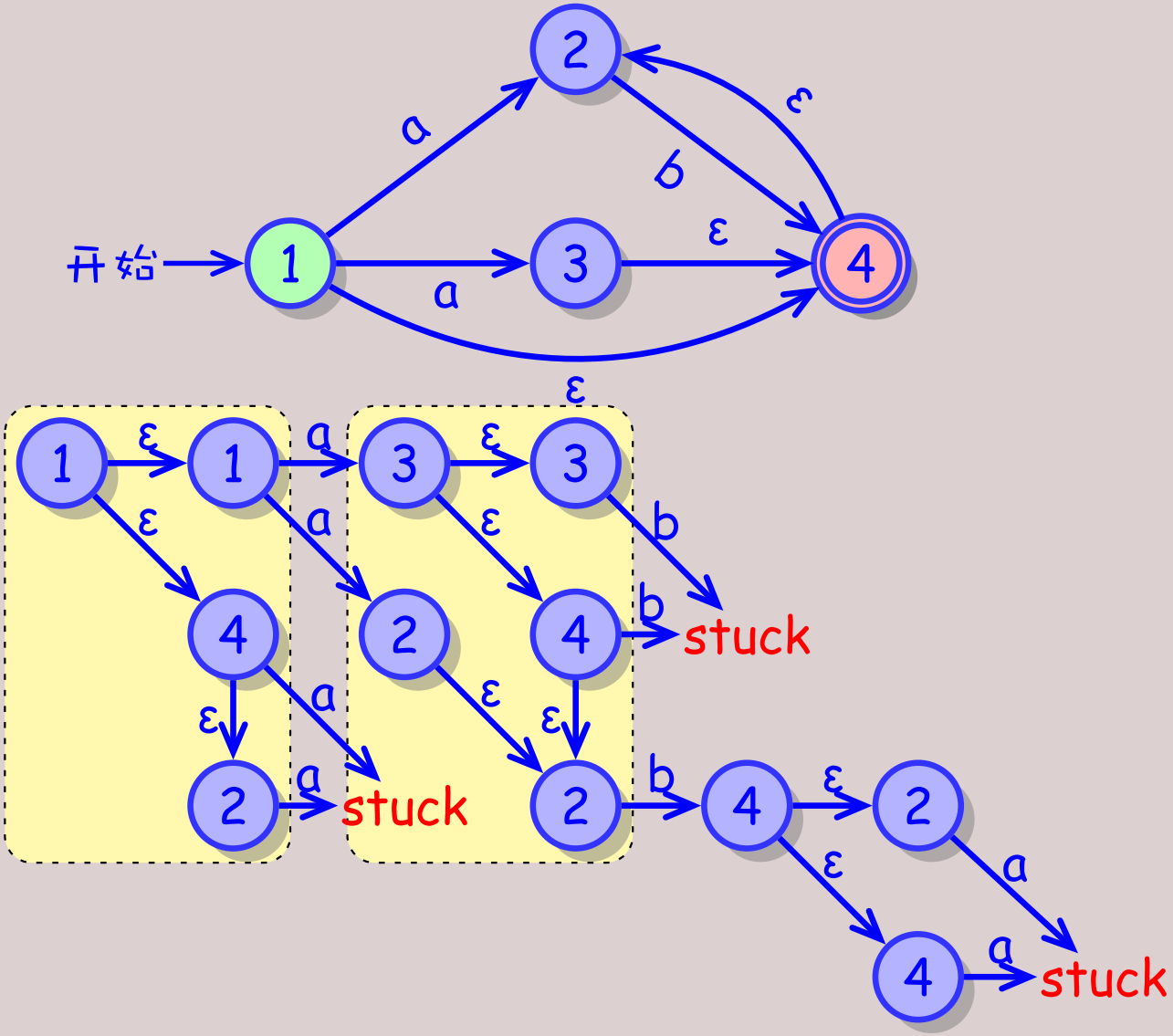
Example: 不接受 aba.



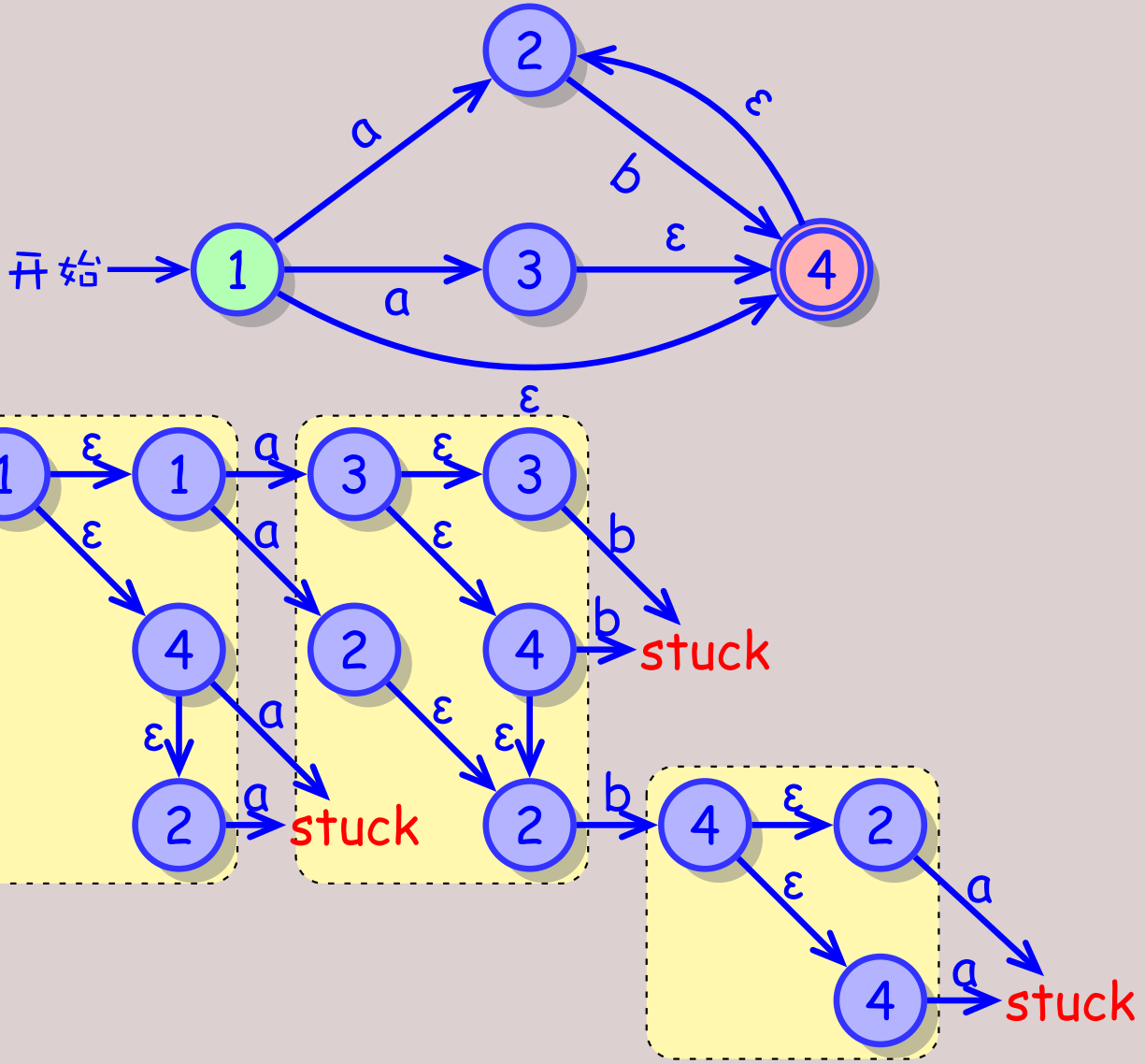
Example：不接受 aba.



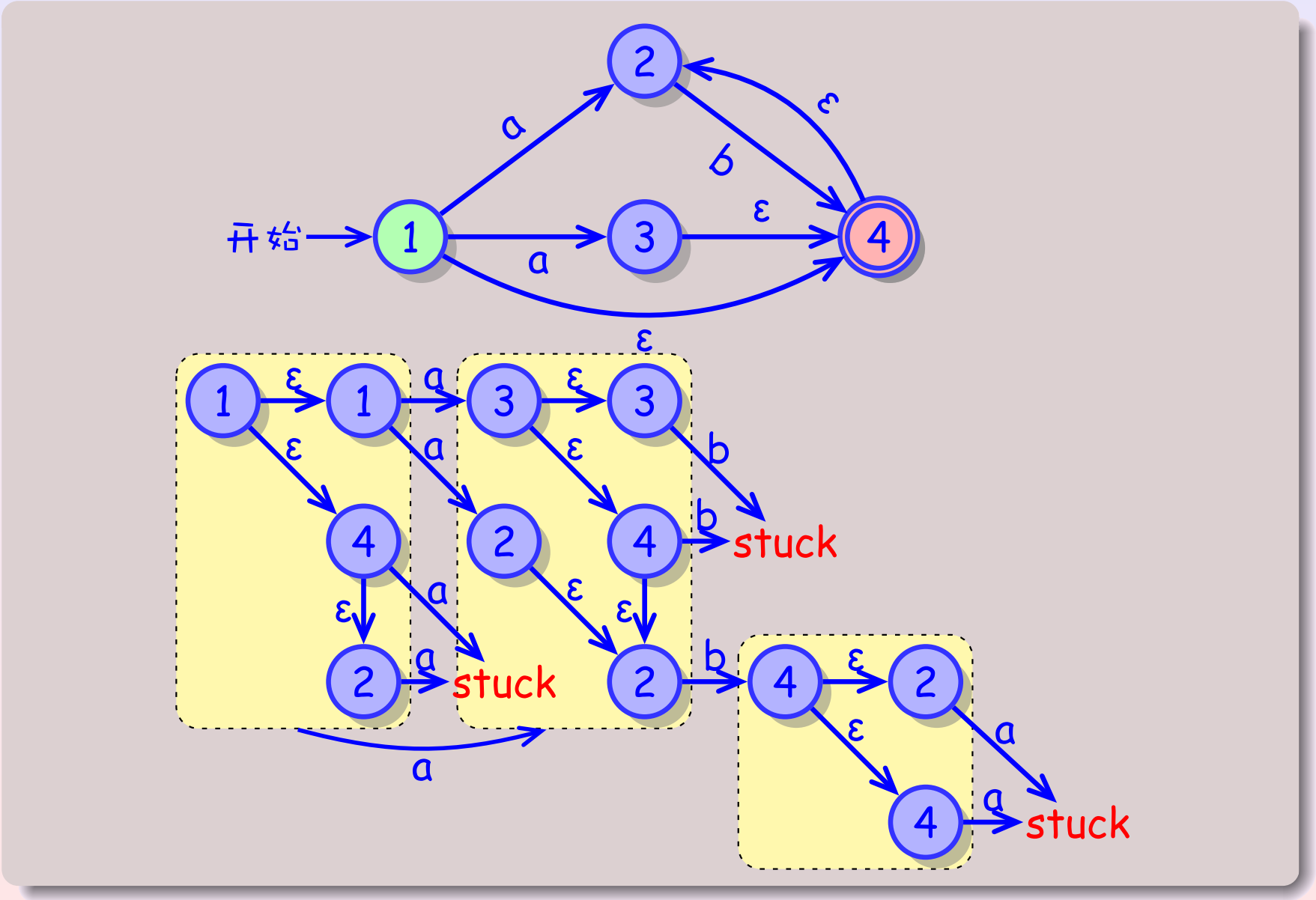
Example：不接受 aba.



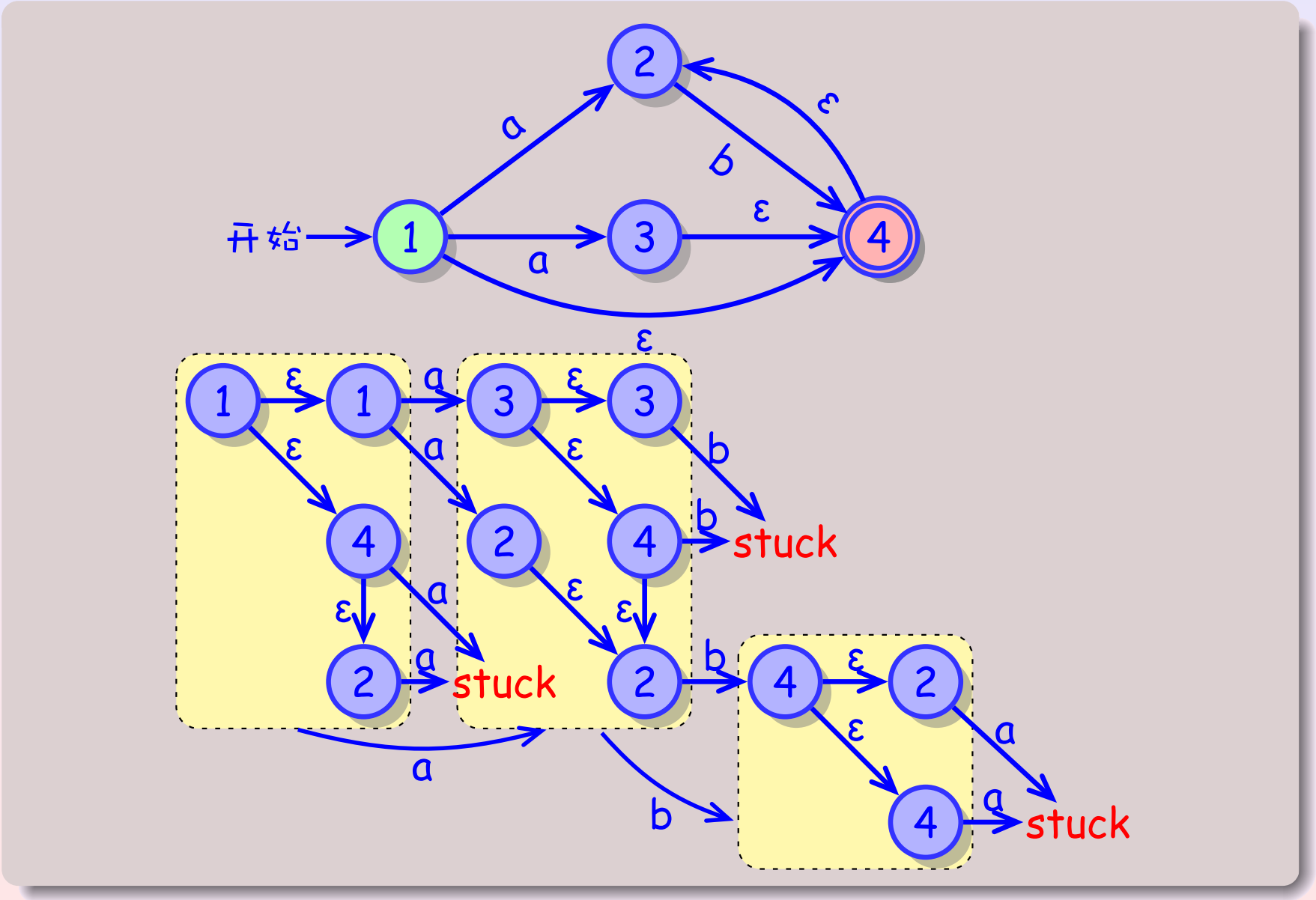
Example：不接受 aba.



Example: 不接受 aba.



Example: 不接受 aba.



框架

为了实现从描述词法的正则表达式到以确定的有限自动机为支撑的识别该词法的词法分析器的自动转换，我们利用 NFA 作为该转换的桥，通过Thompson构造将正则表达式转换为与之等价的 NFA，再用子集构造法将所得到的 NFA 转换为与之等价的 DFA，从而构造出以该 DFA 状态转换表为驱动的识别给定正则表达式的词法分析器.

Thompson 构造法

利用正则表达式的递归结构，实现正则表达式到 NFA 的自动转换.

Thompson's Construction

- 分别对每个字母 $a \in \Sigma$ 和 ε 构造出唯一的接受态的 NFA;
- 如果正则表达式 r 和 s 对应的 NFA M_r 和 M_s 已构造，则

定理

设 M_r 是按上述方法对正则表达式 r 所构造的 NFA，
则 $L(M_r) = L(r)$，即两者等价.

Thompson 构造法

利用正则表达式的递归结构，实现正则表达式到 NFA 的自动转换.

Thompson's Construction

- 分别对每个字母 $a \in \Sigma$ 和 ε 构造由唯一的接受状的 NFA;
- 如果正则表达式 r 和 s 对应的 NFA M_r 和 M_s 已构造，则
  1. 正则表达式 $r|s$ 对应的自动机是将 M_r 和 M_s 并联后所得到的自动机
  2. 正则表达式 rs 对应的自动机是将 M_r 和 M_s 串联所得到的自动机
  3. 正则表达式 r^* 对应的自动机是将 M_r 闭包后所得到的自动机

定理

设 M_r 是按上述方法对正则表达式 r 所构造的 NFA，
则 $L(M_r) = L(r)$，即两者等价.

Thompson 构造法

利用正则表达式的递归结构，实现正则表达式到 NFA 的自动转换.

Thompson's Construction

- 分别对每个字母 $a \in \Sigma$ 和 ε 构造由唯一的接受状的 NFA;
- 如果正则表达式 r 和 s 对应的 NFA M_r 和 M_s 已构造，则
  1. 正则表达式 $r|s$ 对应的自动机是将 M_r 和 M_s "并联" 后所得到的自动机.
  2. 正则表达式 rs 对应的自动机是将 M_r 和 M_s "串联" 后所得到的自动机
  3. 正则表达式 r^* 对应的自动机是将 M_r 即加上自环后所得到的自动机.

定理

设 M_r 是按上述方法对正则表达式 r 所构造的 NFA，则 $L(M_r) = L(r)$，即两者等价.

Thompson 构造法

利用正则表达式的递归结构，实现正则表达式到 NFA 的自动转换.

Thompson's Construction

- 分别对每个字母 $a \in \Sigma$ 和 ε 构造由唯一的接受状的 NFA;
- 如果正则表达式 r 和 s 对应的 NFA M_r 和 M_s 已构造，则
  1. 正则表达式 $r \mid s$ 对应的自动机是将 M_r 和 M_s ``并联'' 后所得到的自动机.
  2. 正则表达式 rs 对应的自动机是将 M_r 和 M_s ``串联'' 后所得到的自动机.
  3. 正则表达式 r^* 对应的自动机是将 M_r 中加上短路边和回路自动机.

定理

设 M_r 是按上述方法对正则表达式 r 所构造的 NFA，
则 $L(M_r) = L(r)$，即两者等价.

Thompson 构造法

利用正则表达式的递归结构，实现正则表达式到 NFA 的自动转换.

Thompson's Construction

- 分别对每个字母 $a \in \Sigma$ 和 ε 构造由唯一的接受状的 NFA;
- 如果正则表达式 r 和 s 对应的 NFA M_r 和 M_s 已构造，则
  1. 正则表达式 $r \mid s$ 对应的自动机是将 M_r 和 M_s ``并联'' 后所得到的自动机.
  2. 正则表达式 rs 对应的自动机是将 M_r 和 M_s ``串联'' 后所得到的自动机.
  3. 正则表达式 r^* 对应的自动机是将 M_r 中加上短路边和回路自动机.

定理

设 M_r 是按上述方法对正则表达式 r 所构造的 NFA，
则 $L(M_r) = L(r)$，即两者等价.

Thompson 构造法

利用正则表达式的递归结构，实现正则表达式到 NFA 的自动转换.

Thompson's Construction

- 分别对每个字母 $a \in \Sigma$ 和 ε 构造由唯一的接受状的 NFA;
- 如果正则表达式 r 和 s 对应的 NFA M_r 和 M_s 已构造，则
  1. 正则表达式 $r \mid s$ 对应的自动机是将 M_r 和 M_s ``并联'' 后所得到的自动机.
  2. 正则表达式 rs 对应的自动机是将 M_r 和 M_s ``串联'' 后所得到的自动机.
  3. 正则表达式 r^* 对应的自动机是将 M_r 中加上短路边和回路自动机.

定理

设 M_r 是按上述方法对正则表达式 r 所构造的 NFA，
则 $L(M_r) = L(r)$，即两者等价.

Thompson 构造法

利用正则表达式的递归结构，实现正则表达式到 NFA 的自动转换.

Thompson's Construction

- 分别对每个字母 $a \in \Sigma$ 和 ε 构造由唯一的接受状的 NFA;
- 如果正则表达式 r 和 s 对应的 NFA M_r 和 M_s 已构造，则
  1. 正则表达式 $r \mid s$ 对应的自动机是将 M_r 和 M_s ``并联'' 后所得到的自动机.
  2. 正则表达式 rs 对应的自动机是将 M_r 和 M_s ``串联'' 后所得到的自动机.
  3. 正则表达式 r^* 对应的自动机是将 M_r 中加上短路边和回路自动机.

定理

设 M_r 是按上述方法对正则表达式 r 所构造的 NFA，
则 $L(M_r) = L(r)$，即两者等价.

Thompson 构造法

利用正则表达式的递归结构，实现正则表达式到 NFA 的自动转换.

Thompson's Construction

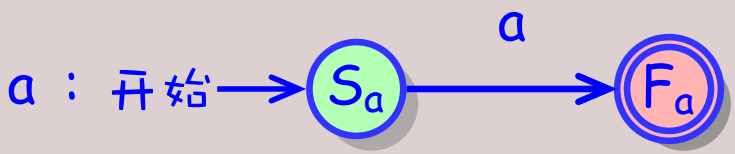
- 分别对每个字母 $a \in \Sigma$ 和 ε 构造由唯一的接受状的 NFA;
- 如果正则表达式 r 和 s 对应的 NFA M_r 和 M_s 已构造，则
  1. 正则表达式 r | s 对应的自动机是将 M_r 和 M_s ``并联'' 后所得到的自动机.
  2. 正则表达式 rs 对应的自动机是将 M_r 和 M_s ``串联'' 后所得到的自动机.
  3. 正则表达式 r* 对应的自动机是将 M_r 中加上短路边和回路自动机.

定理

设 M_r 是按上述方法对正则表达式 r 所构造的 NFA，
则 $L(M_r) = L(r)$，即两者等价.

Thompson 构造法 --- 归纳基础

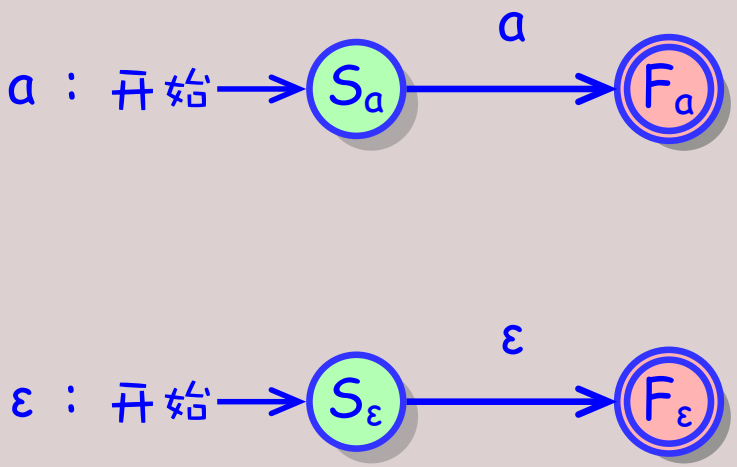
字母 a 和空串 ε 对应的自动机



©hfwang

Thompson 构造法 --- 归纳基础

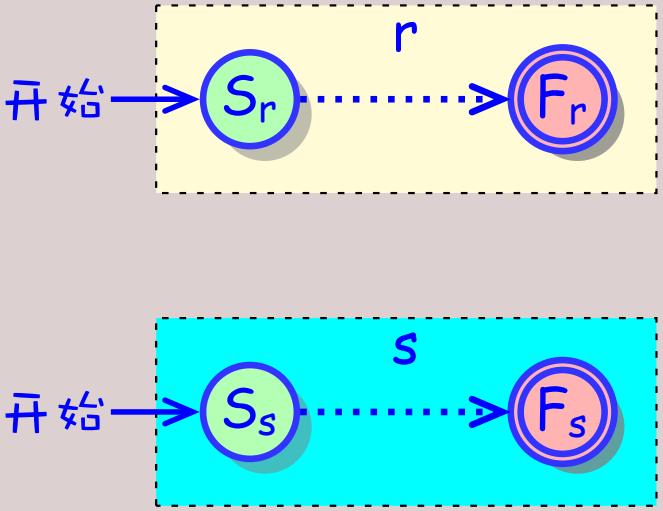
字母 a 和空串 ε 对应的自动机



©hfwang

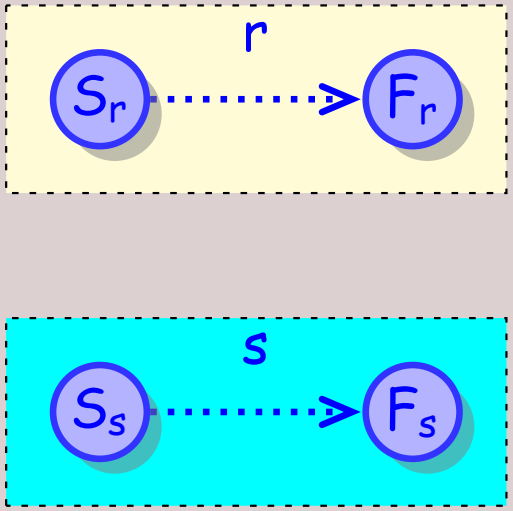
Thompson 构造法 --- 并运算：并联

r | s 对应的自动机



Thompson 构造法 --- 并运算：并联

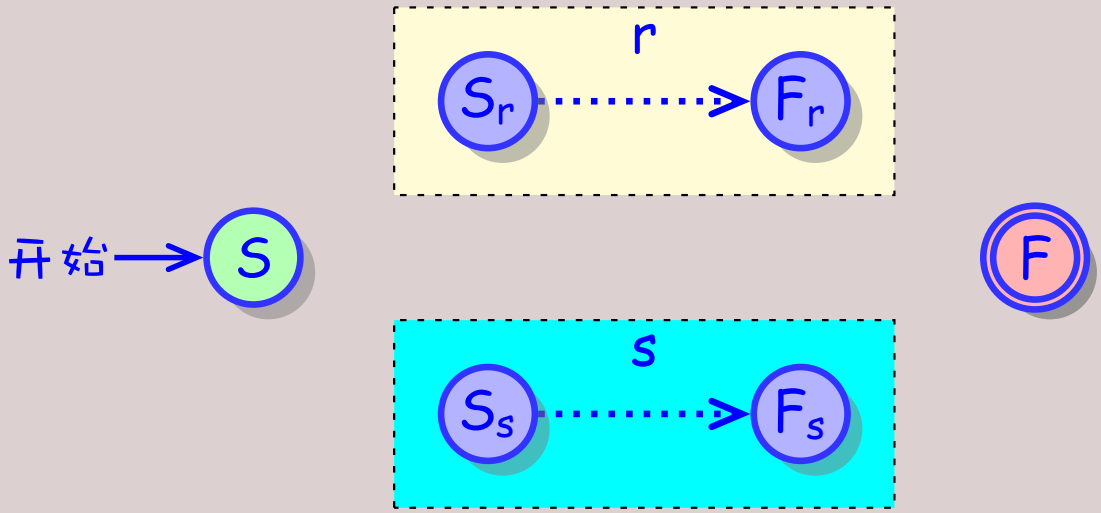
r | s 对应的自动机



©hfwang

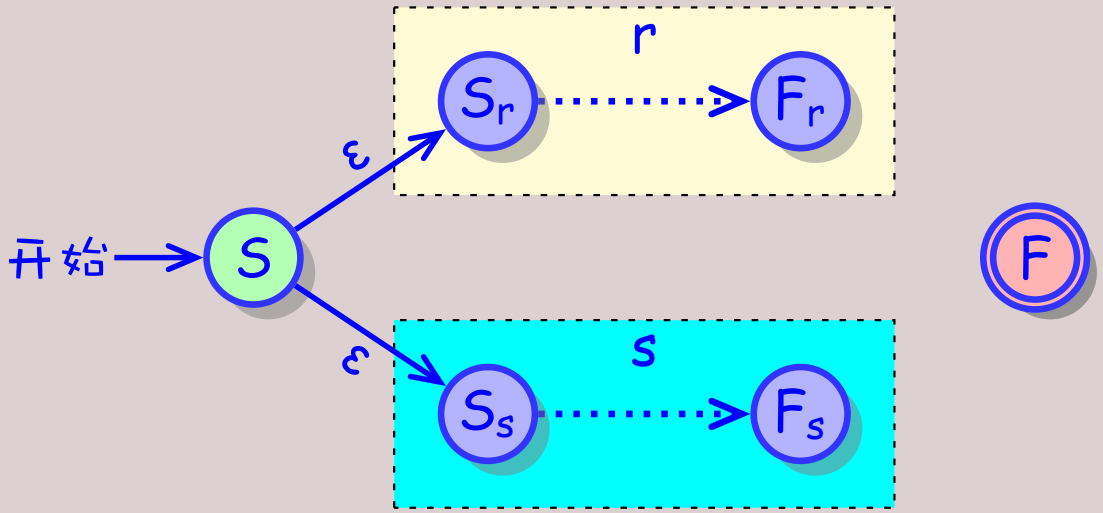
Thompson 构造法 --- 并运算：并联

r | s 对应的自动机



Thompson 构造法 --- 并运算：并联

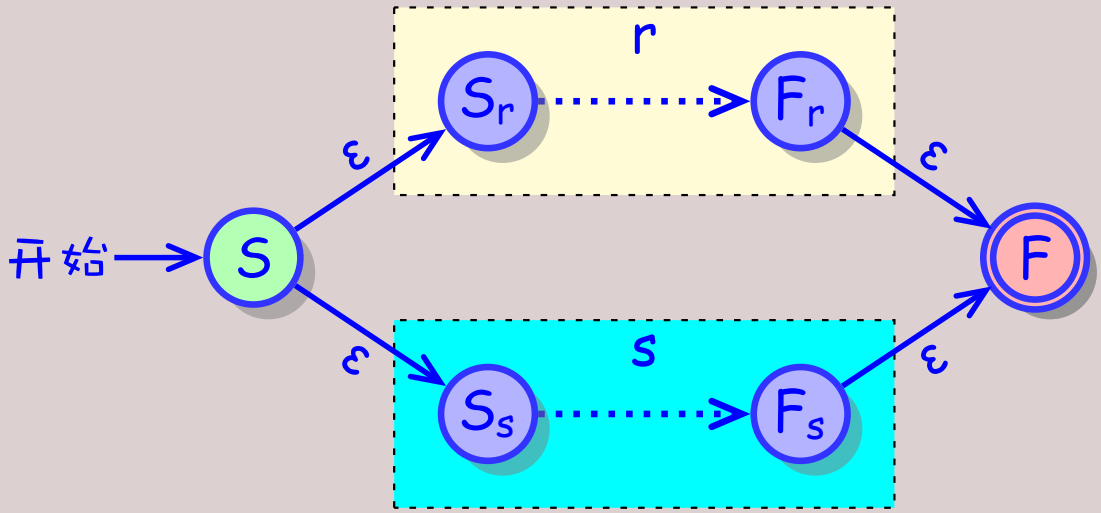
r | s 对应的自动机



©hfwang

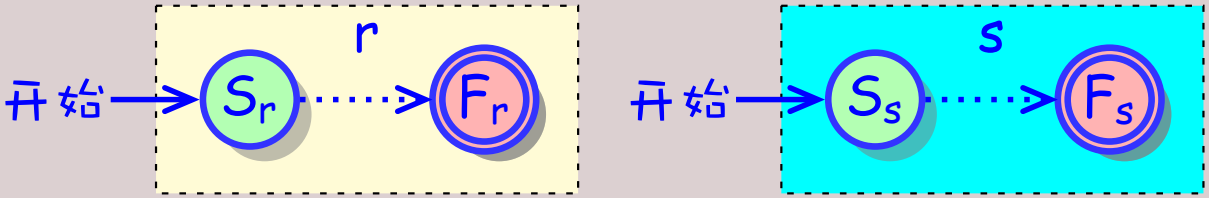
Thompson 构造法 --- 并运算：并联

r | s 对应的自动机



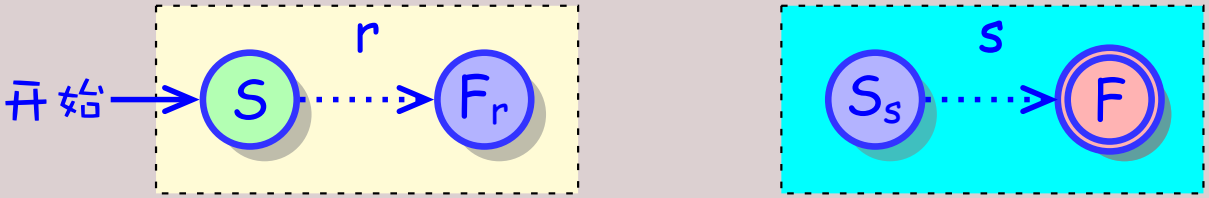
Thompson 构造法 --- 连接运算：串连

rs 对应的自动机



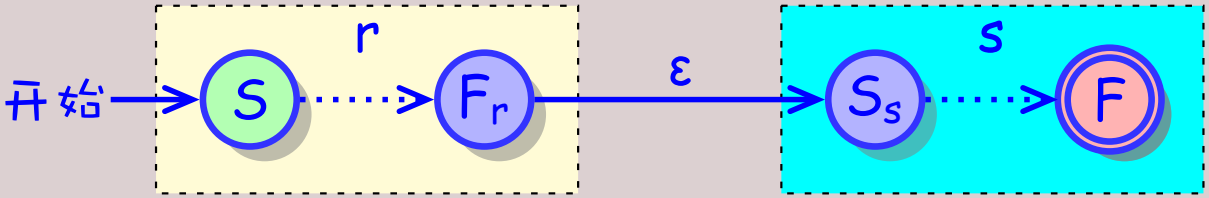
Thompson 构造法 --- 连接运算：串连

rs 对应的自动机



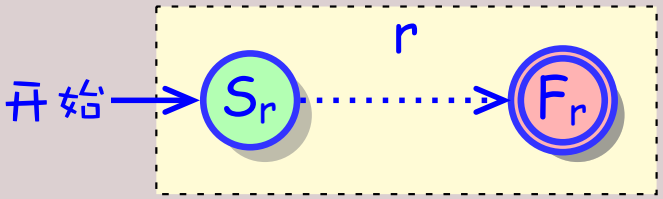
Thompson 构造法 --- 连接运算：串连

rs 对应的自动机



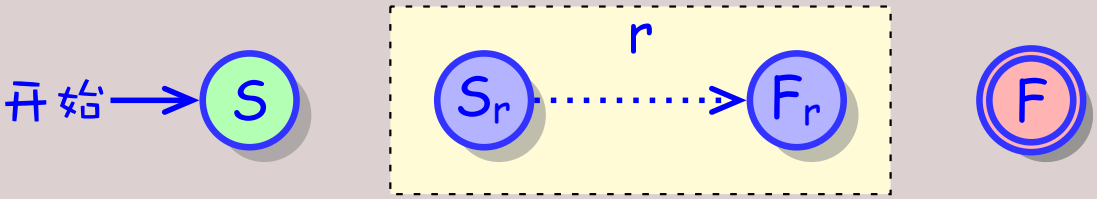
Thompson 构造法 --- Kleene Closure：回路+短路.

r* 对应的自动机



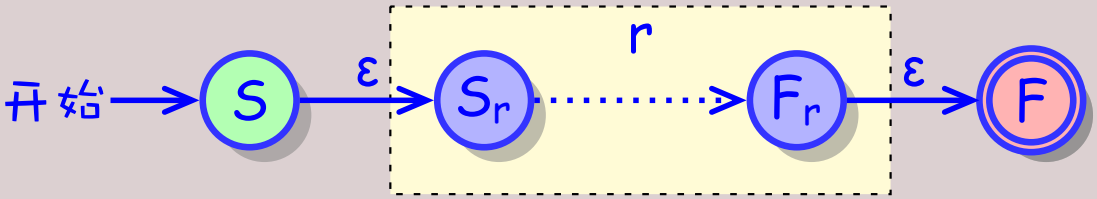
Thompson 构造法 --- Kleene Closure：回路+短路.

r* 对应的自动机



Thompson 构造法 --- Kleene Closure：回路+短路.

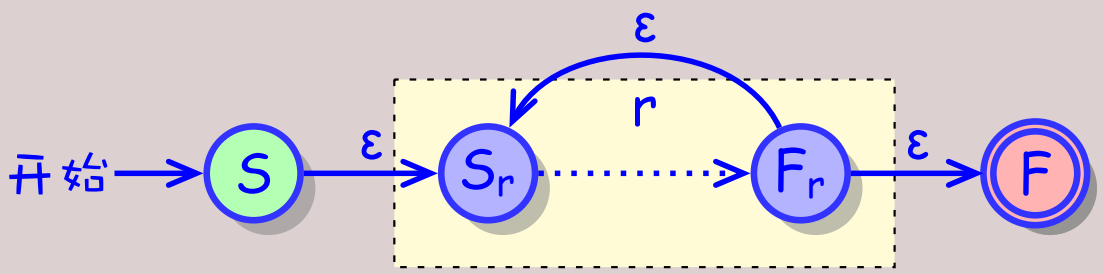
r* 对应的自动机



©hfwang

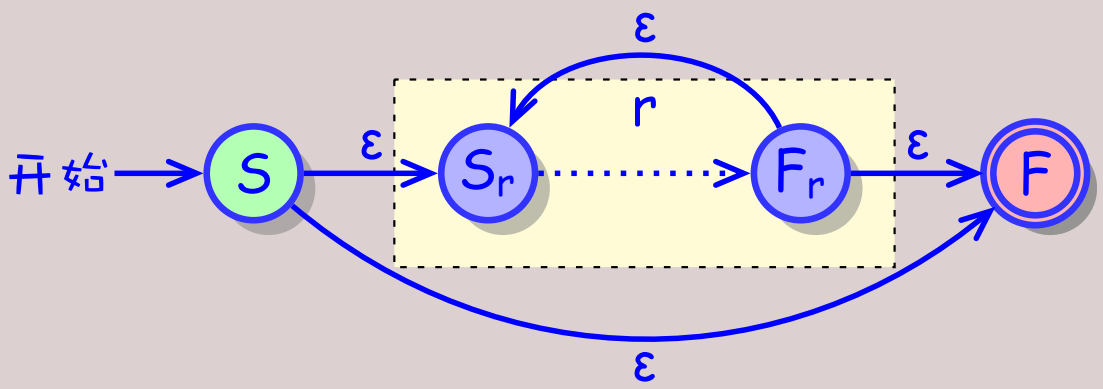
Thompson 构造法 --- Kleene Closure：回路+短路.

r* 对应的自动机



Thompson 构造法 --- Kleene Closure：回路+短路.

r* 对应的自动机

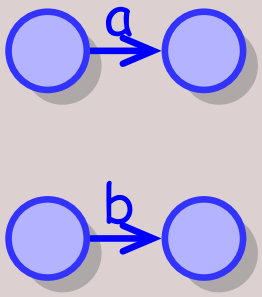


©hfwang

Example

求与 (a | b)*abb 等价的 NFA

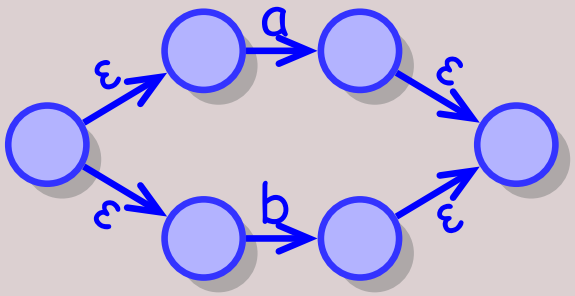
a和 b



Example

求与 (a | b)*abb 等价的 NFA

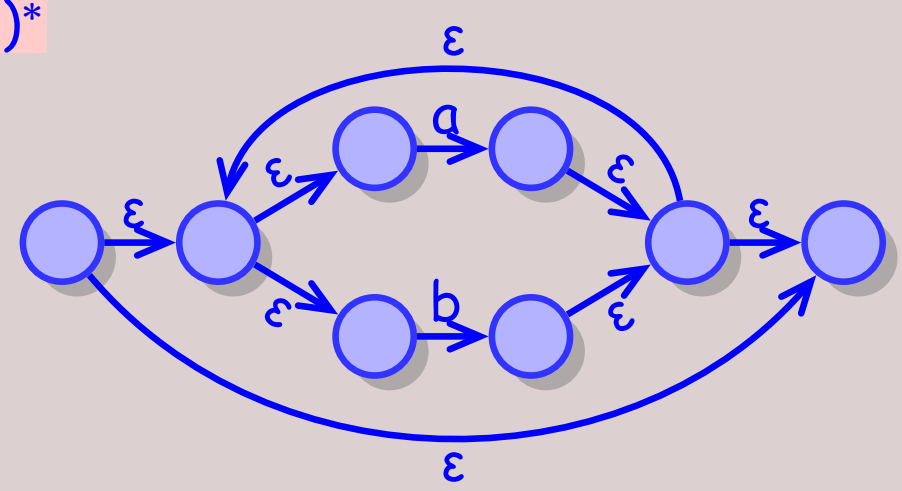
a | b



Example

求与 (a | b)*abb 等价的 NFA

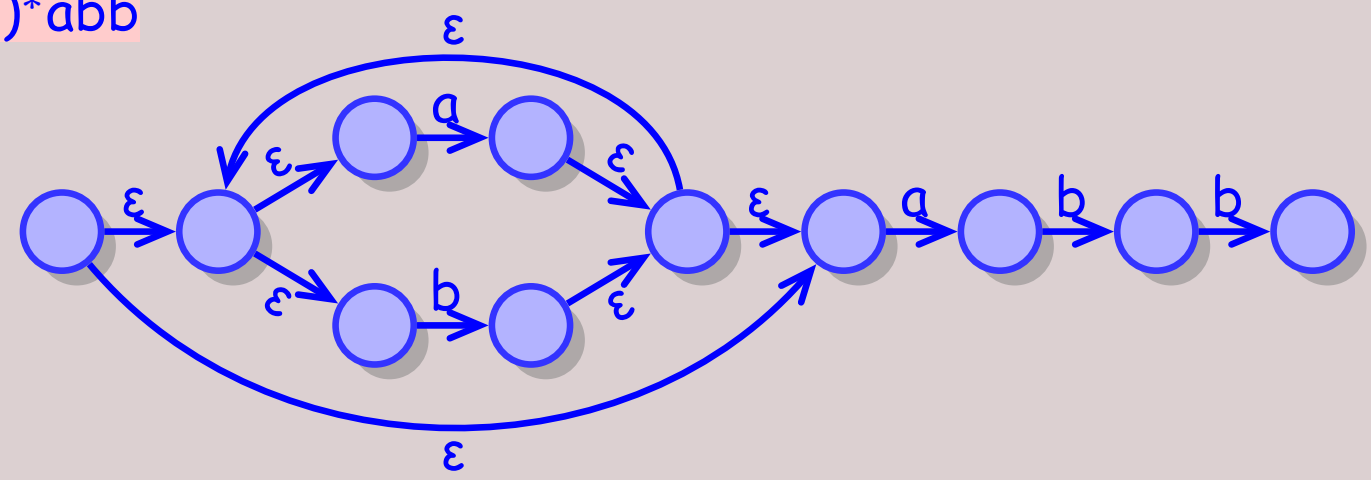
(a | b)*



Example

求与 (a | b)*abb 等价的 NFA

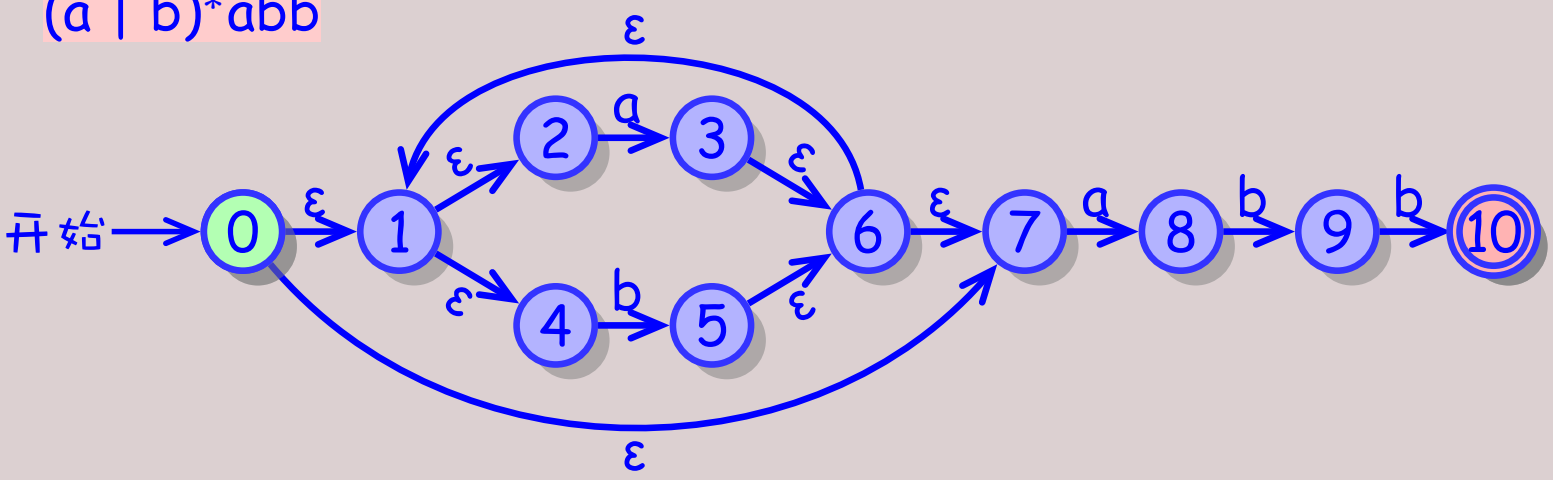
(a | b)*abb



Example

求与 (a | b)*abb 等价的 NFA

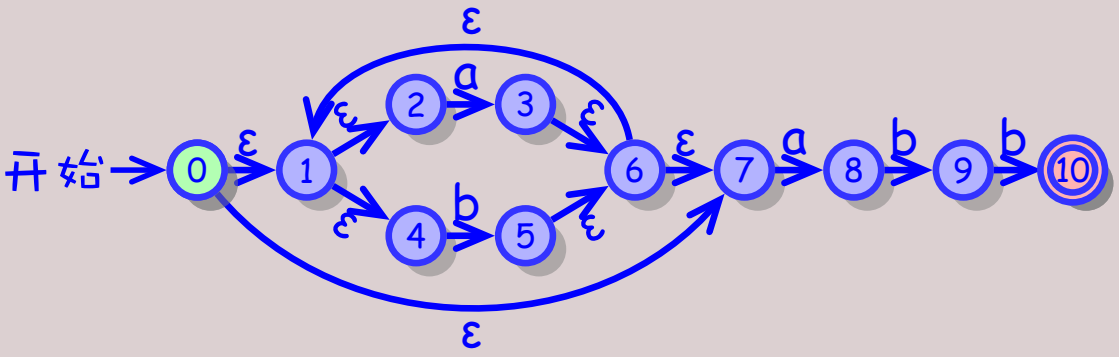
(a | b)*abb



ε – closure

解决 NFA 的不确定性 (多条路经接受一个输入串): 枚举所有可能的路径.

Example: NFA of (a | b)*abb

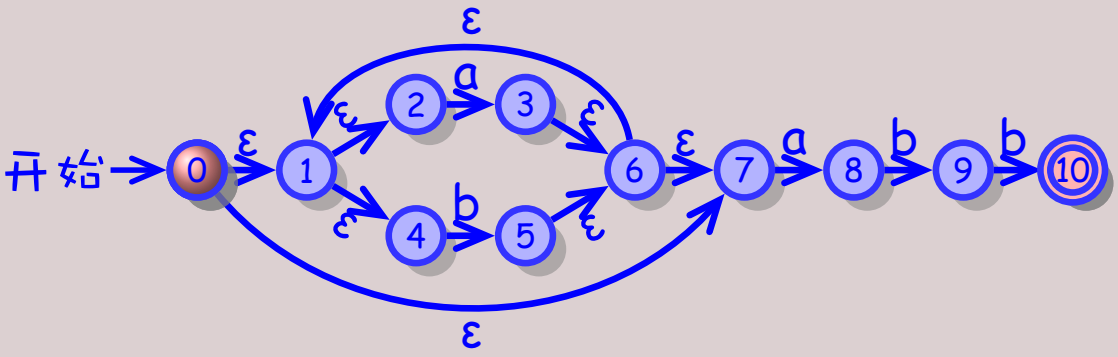


接受 aabb 所有可能的 paths

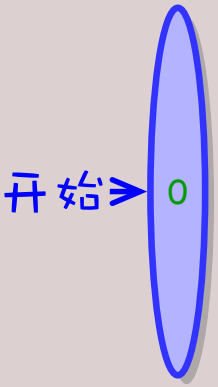
ε – closure

解决 NFA 的不确定性 (多条路经接受一个输入串): 枚举所有可能的路径.

Example: NFA of (a | b)*abb



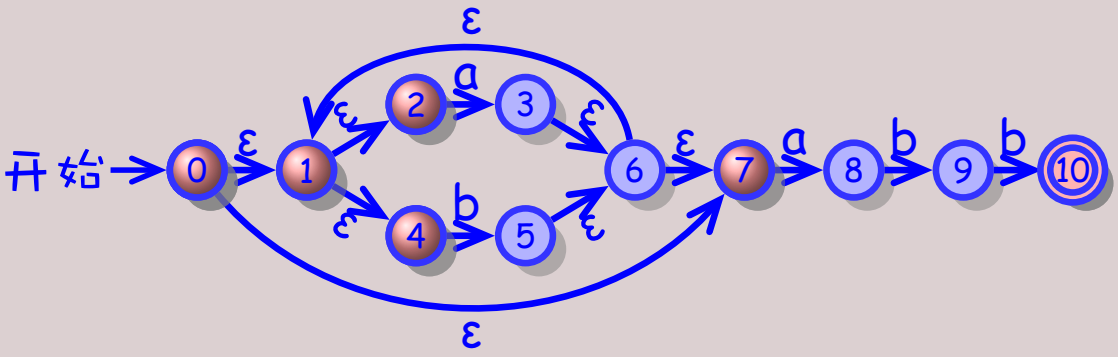
接受 aabb 所有可能的 paths



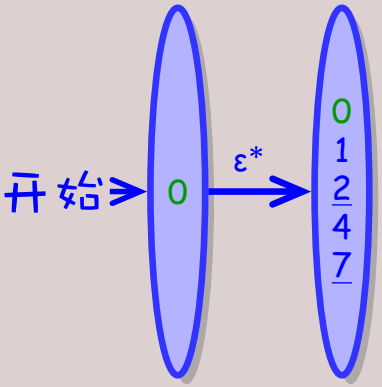
ε – closure

解决 NFA 的不确定性 (多条路经接受一个输入串): 枚举所有可能的路径.

Example: NFA of (a | b)*abb



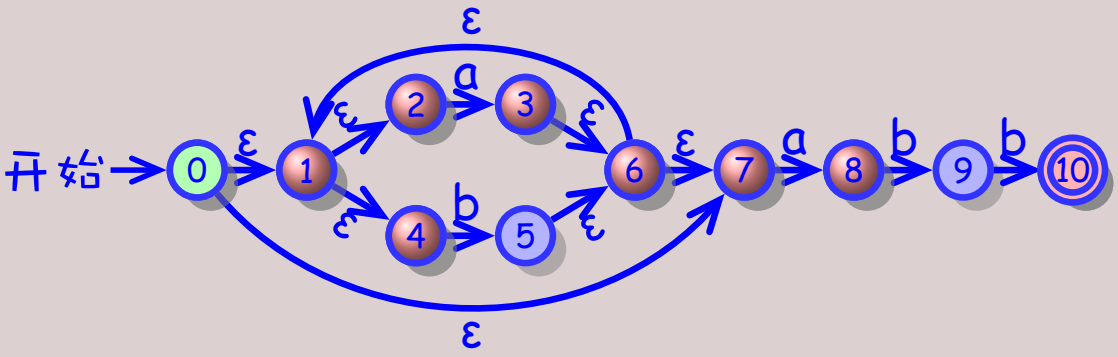
接受 aabb 所有可能的 paths



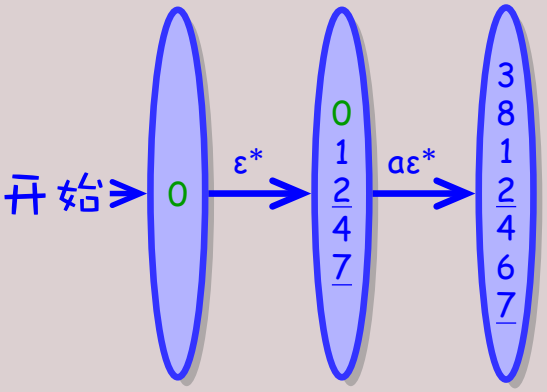
ε – closure

解决 NFA 的不确定性 (多条路经接受一个输入串): 枚举所有可能的路径.

Example: NFA of (a | b)*abb



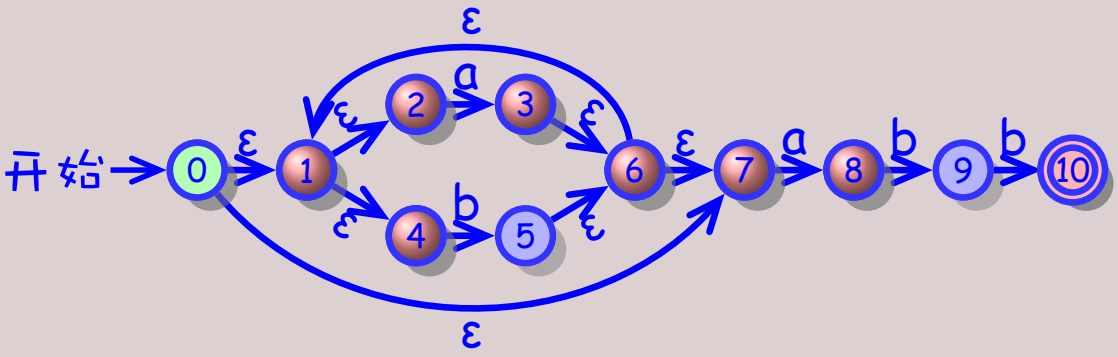
接受 aabb 所有可能的 paths



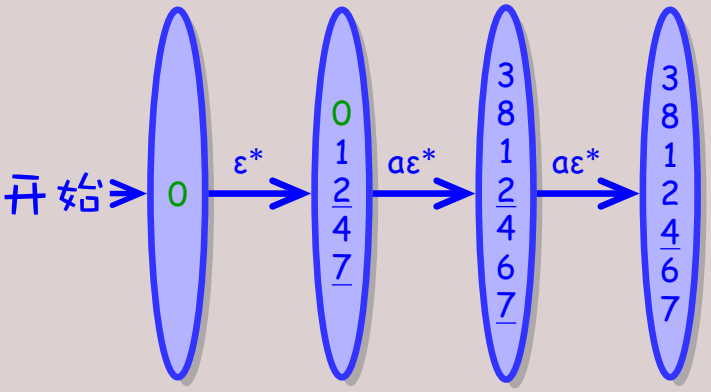
ε – closure

解决 NFA 的不确定性 (多条路经接受一个输入串): 枚举所有可能的路径.

Example: NFA of (a | b)*abb



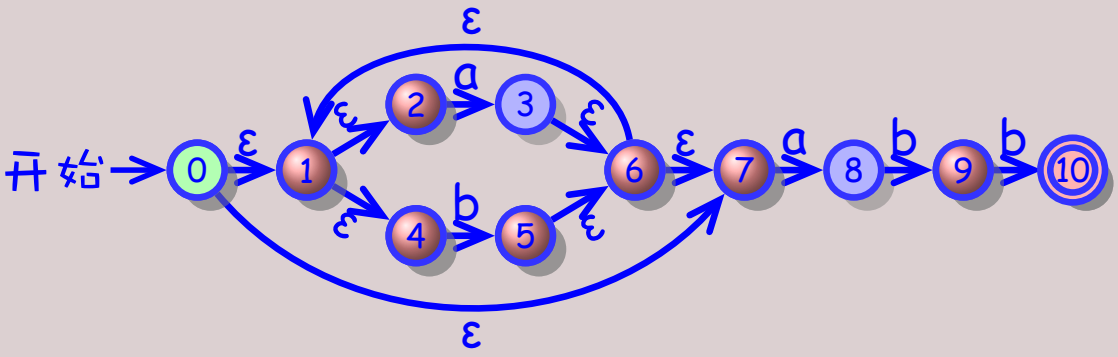
接受 aabb 所有可能的 paths



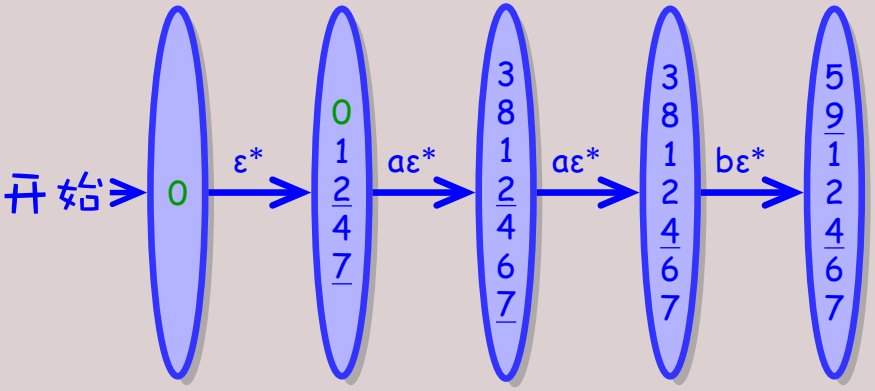
ε – closure

解决 NFA 的不确定性 (多条路经接受一个输入串): 枚举所有可能的路径.

Example: NFA of (a | b)*abb



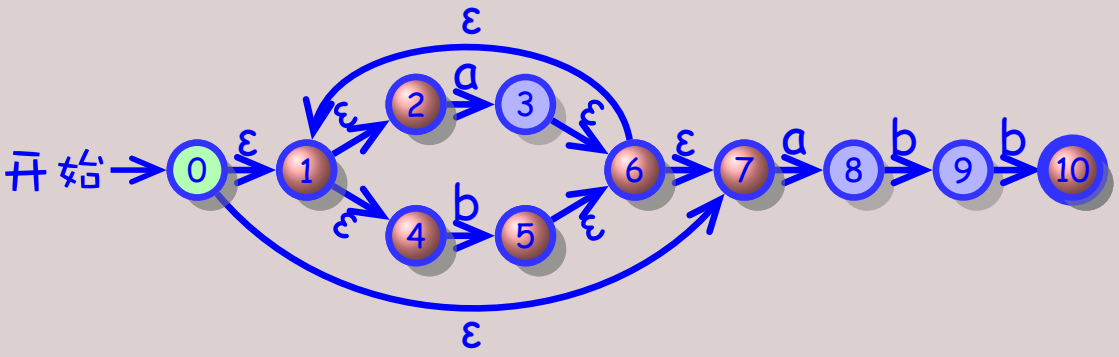
接受 aabb 所有可能的 paths



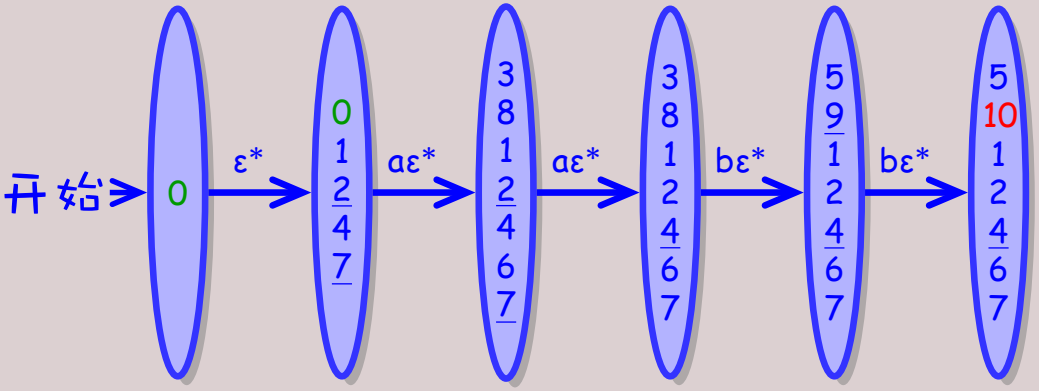
ε – closure

解决 NFA 的不确定性 (多条路经接受一个输入串): 枚举所有可能的路径.

Example: NFA of (a | b)*abb



接受 aabb 所有可能的 paths



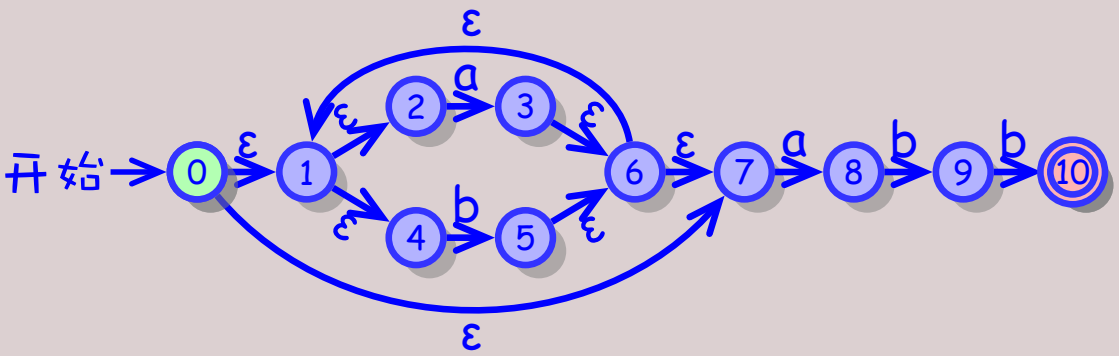
ε-closure

定义

设 M 是 NFA, $B \subseteq S$ 是 M 的一个状态子集合:

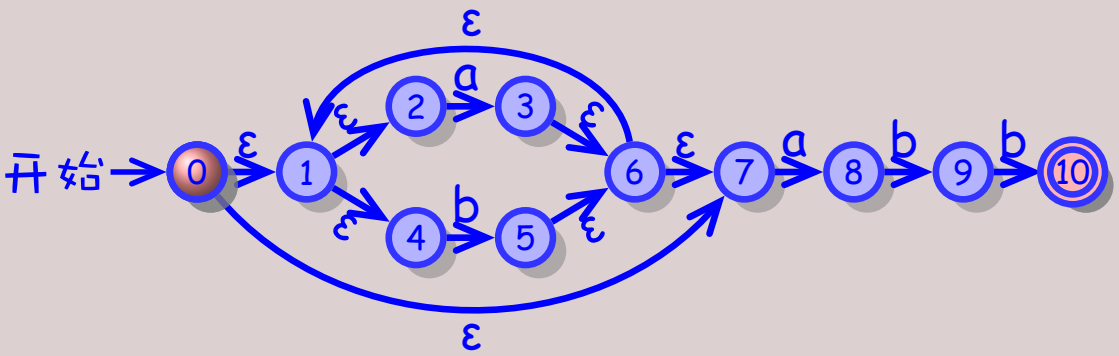
$$\begin{aligned} \varepsilon\text{-closure}(B) &= B \cup \{s' \mid \exists s \in B \wedge s \xrightarrow{\varepsilon} s_1 \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s'\} \\ &\triangleq \overline{B} \end{aligned}$$

Example: NFA of (a | b)*abb

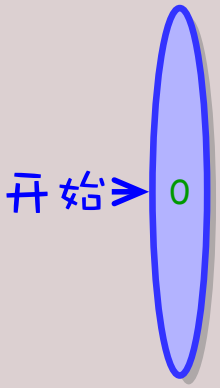


ε-closure

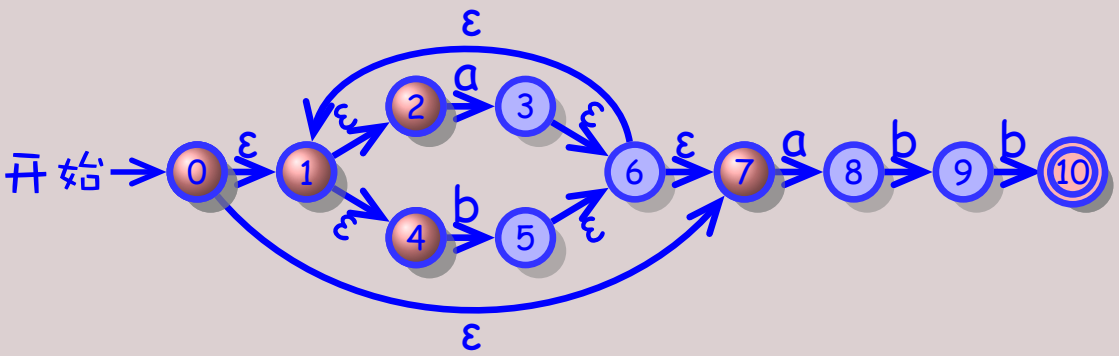
Example: NFA of (a | b)*abb



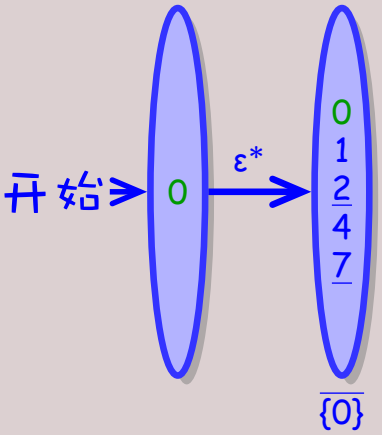
ε-closure



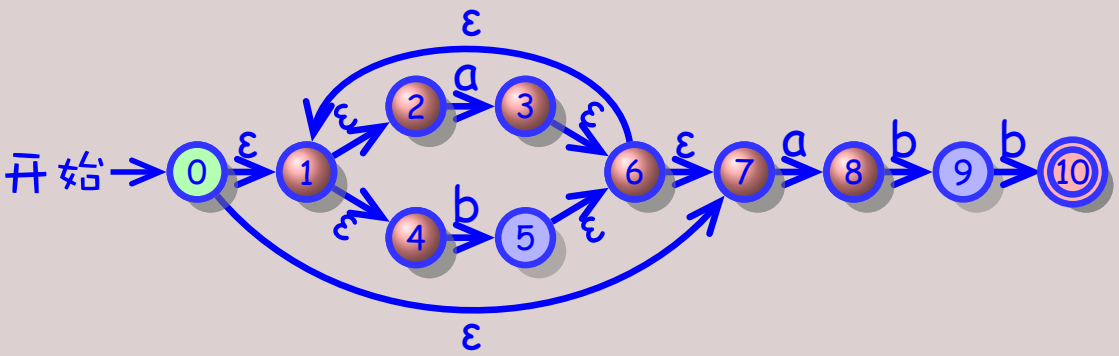
Example: NFA of (a | b)*abb



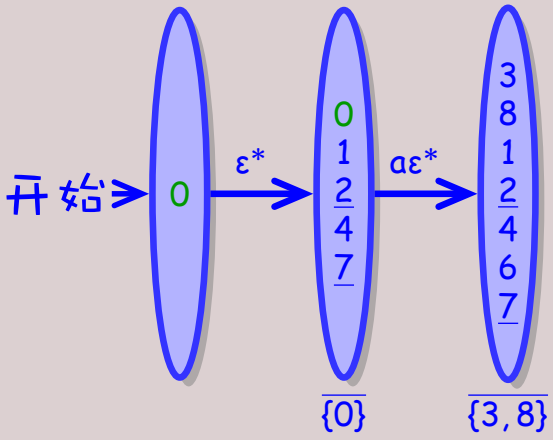
ε-closure



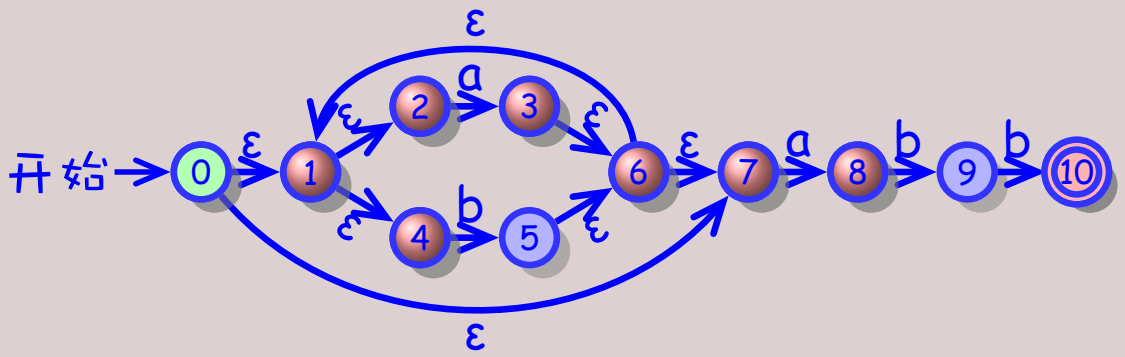
Example: NFA of (a | b)*abb



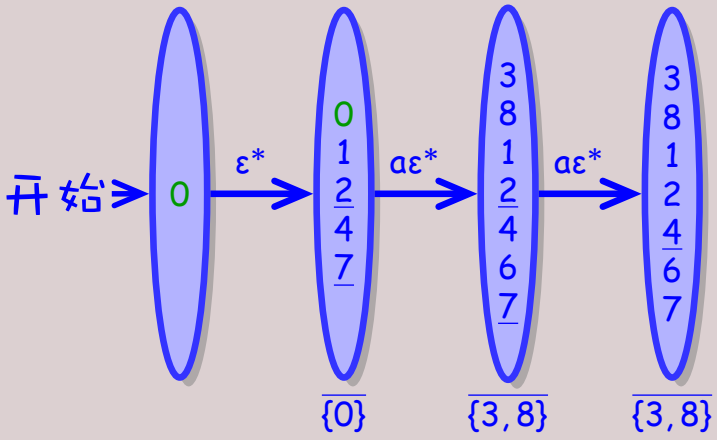
ε-closure



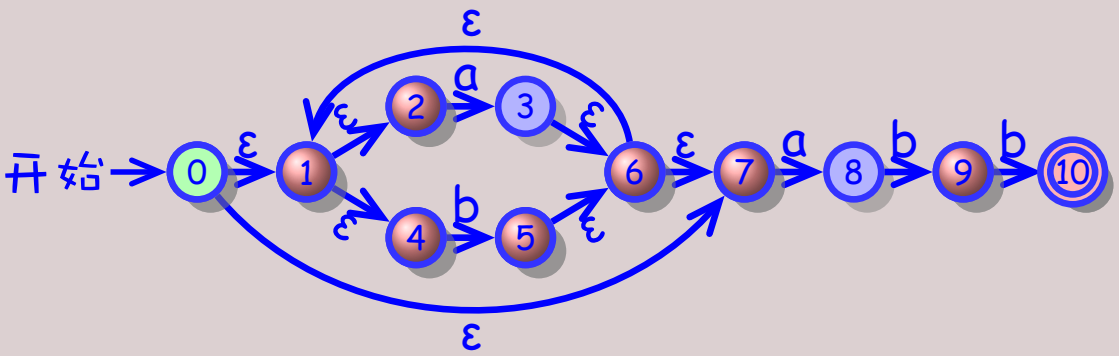
Example: NFA of (a | b)*abb



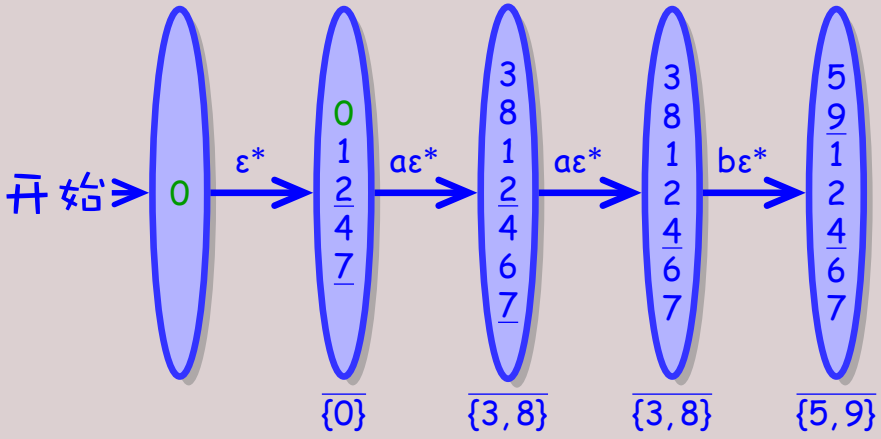
ε-closure



Example: NFA of (a | b)*abb

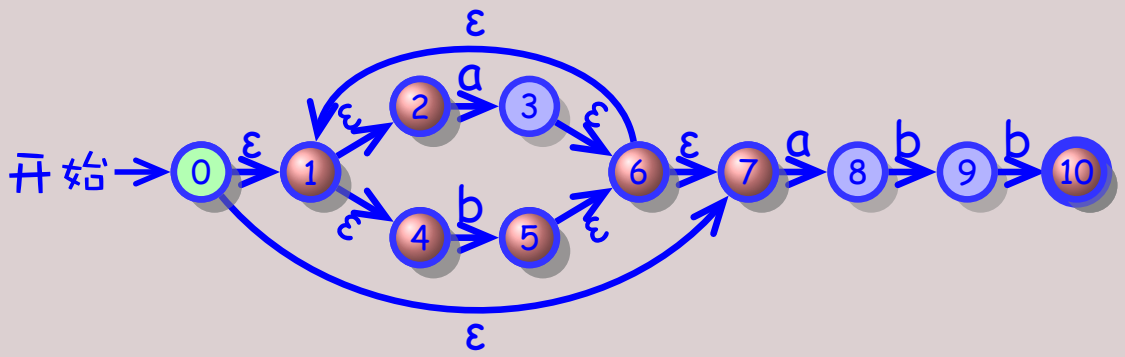


ε-closure

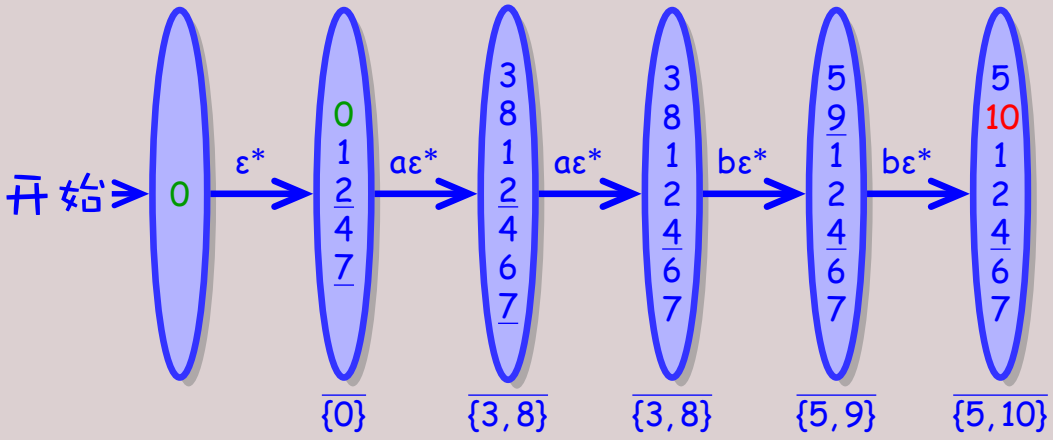


©hfwang

Example: NFA of (a | b)*abb



ε-closure



Algorithm

ε-closure

```
set ε-closure( set B )
{
  stack Stack;  /* 记录还没有传播的状态 */
  set B̄ = ∅;  /* 记录已经传播的状态 */
  push all elements of B in Stack;
  while (!empty(Stack)) {
    state s, s1;
    s = pop(Stack);
    if (s ∉ B̄) {
        /* 对没有传播的状态进行传播 */
      B̄ = B̄ ∪ { s };
        /* 加入 B̄, 表示已经传播 */
      for every state s1 ∈ Move(s,ε) {
        if (s1 ∉ B̄)
          push(s1);  /* 加入新的可转播的状态 */
      }
    }
  }
  return B̄;
}
```

注释

算法的收斂

stack 中的状态是没有传播的状态，保证每个状态仅被传播一次，这样，即使状态转换图中存在 ε 的回路，也不会产生死循环. 另外，状态数目是有限的，所以 while 循环一定会终止.

算法的复杂性

Depth First Search: $O(n+m)$, n 为 NFA 的状态数，m 为边数.

Example

在 Compiler_cd 中 ``\holub\SRC\COMPILER\LEX\terp.c''

注释

算法的收斂

stack 中的状态是没有传播的状态，保证每个状态仅被传播一次，这样，即使状态转换图中存在 ε 的回路，也不会产生死循环. 另外，状态数目是有限的，所以 while 循环一定会终止.

算法的复杂性

Depth First Search：$O(n+m)$, n 为 NFA 的状态数，m 为边数.

Example

在 Compiler_cd 中 ``\holub\SRC\COMPILER\LEX\terp.c''

注释

算法的收敛

stack 中的状态是没有传播的状态，保证每个状态仅被传播一次，这样，即使状态转换图中存在 ε 的回路，也不会产生死循环. 另外，状态数目是有限的，所以 while 循环一定会终止.

算法的复杂性

Depth First Search: $O(n+m)$, n 为 NFA 的状态数，m 为边数.

Example

在 Compiler_cd 中 ``\holub\SRC\COMPILER\LEX\terp.c''

©hfwang

NFA 到 DFA 的转换

DFA of NFA

设 $M = \langle \Sigma, S, Move, s_0, F \rangle$ 是 NFA，其对应的 DFA $D = \langle \Sigma, \mathcal{S}, Trans, \int_0, \mathcal{F} \rangle$ 定义如下：

- $\mathcal{S} = \{\overline{T} \mid T \subseteq S\}$
- $\int_0 = \overline{\{s_0\}}$
- $\mathcal{F} = \{\overline{T} \mid \overline{T} \cap F \neq \emptyset\}$
- $Trans(\overline{T}, a) = \overline{\bigcup_{t \in \overline{T}} Move(t, a)}$

定理

由上定义得到的 DFA D与原 NFA M 等价，即：

$$L(M) = L(D)$$

说明

DFA：$A \xrightarrow{a} B \Leftrightarrow$ NFA：$s_i \xrightarrow{\varepsilon} s_k \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s_m \xrightarrow{a} s_n \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s_q$

NFA 到 DFA 的转换

DFA of NFA

设 $M = \langle \Sigma, S, Move, s_0, F \rangle$ 是 NFA，其对应的 DFA $D = \langle \Sigma, \mathcal{S}, Trans, \int_0, \mathcal{F} \rangle$ 定义如下：

- $\mathcal{S} = \{\overline{T} \mid T \subseteq S\}$
- $\int_0 = \overline{\{s_0\}}$
- $\mathcal{F} = \{\overline{T} \mid \overline{T} \cap F \neq \emptyset\}$
- $Trans(\overline{T}, a) = \overline{\bigcup_{t \in \overline{T}} Move(t, a)}$

定理

由上定义得到的 DFA D与原 NFA M 等价，即：

$$L(M) = L(D)$$

说明

DFA：$A \xrightarrow{a} B \Leftrightarrow$ NFA：$s_i \xrightarrow{\varepsilon} s_k \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s_m \xrightarrow{a} s_n \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s_q$

NFA 到 DFA 的转换

DFA of NFA

设 $M = \langle \Sigma, S, Move, s_0, F \rangle$ 是 NFA，其对应的 DFA $D = \langle \Sigma, \mathcal{S}, Trans, \int_0, \mathcal{F} \rangle$ 定义如下：

- $\mathcal{S} = \{\overline{T} \mid T \subseteq S\}$
- $\int_0 = \overline{\{s_0\}}$
- $\mathcal{F} = \{\overline{T} \mid \overline{T} \cap F \neq \emptyset\}$
- $Trans(\overline{T}, a) = \overline{\bigcup_{t \in \overline{T}} Move(t, a)}$

定理

由上定义得到的 DFA D与原 NFA M 等价，即：

$$L(M) = L(D)$$

说明

DFA：$A \xrightarrow{a} B \Leftrightarrow$ NFA：$s_i \xrightarrow{\varepsilon} s_k \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s_m \xrightarrow{a} s_n \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s_q$

NFA 到 DFA 的转换

DFA of NFA

设 $M = \langle \Sigma, S, Move, s_0, F \rangle$ 是 NFA，其对应的 DFA $D = \langle \Sigma, \mathcal{S}, Trans, \int_0, \mathcal{F} \rangle$ 定义如下：

- $\mathcal{S} = \{\overline{T} \mid T \subseteq S\}$
- $\int_0 = \overline{\{s_0\}}$
- $\mathcal{F} = \{\overline{T} \mid \overline{T} \cap F \neq \emptyset\}$
- $Trans(\overline{T}, a) = \overline{\bigcup_{t \in \overline{T}} Move(t, a)}$

定理

由上定义得到的 DFA D与原 NFA M 等价，即：

$$L(M) = L(D)$$

说明

DFA：$A \xrightarrow{a} B \Leftrightarrow$ NFA：$s_i \xrightarrow{\varepsilon} s_k \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s_m \xrightarrow{a} s_n \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s_q$

NFA 到 DFA 的转换

DFA of NFA

设 $M = \langle \Sigma, S, Move, s_0, F \rangle$ 是 NFA，其对应的 DFA $D = \langle \Sigma, \mathcal{S}, Trans, \int_0, \mathcal{F} \rangle$ 定义如下：

- $\mathcal{S} = \{\overline{T} \mid T \subseteq S\}$
- $\int_0 = \overline{\{s_0\}}$
- $\mathcal{F} = \{\overline{T} \mid \overline{T} \cap F \neq \emptyset\}$
- $Trans(\overline{T}, a) = \overline{\bigcup_{t \in \overline{T}} Move(t, a)}$

定理

由上定义得到的 DFA D与原 NFA M 等价，即：

$$L(M) = L(D)$$

说明

DFA：$A \xrightarrow{a} B \Leftrightarrow$ NFA：$s_i \xrightarrow{\varepsilon} s_k \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s_m \xrightarrow{a} s_n \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s_q$

NFA 到 DFA 的转换

DFA of NFA

设 $M = \langle \Sigma, S, Move, s_0, F \rangle$ 是 NFA，其对应的 DFA $D = \langle \Sigma, \mathcal{S}, Trans, \int_0, \mathcal{F} \rangle$ 定义如下：

- $\mathcal{S} = \{\overline{T} \mid T \subseteq S\}$
- $\int_0 = \overline{\{s_0\}}$
- $\mathcal{F} = \{\overline{T} \mid \overline{T} \cap F \neq \emptyset\}$
- $Trans(\overline{T}, a) = \overline{\bigcup_{t \in \overline{T}} Move(t, a)}$

定理

由上定义得到的 DFA D 与原 NFA M 等价，即：

$$L(M) = L(D)$$

说明

DFA： $A \xrightarrow{a} B \Leftrightarrow$ NFA： $s_i \xrightarrow{\varepsilon} s_k \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s_m \xrightarrow{a} s_n \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s_q$

NFA 到 DFA 的转换

DFA of NFA

设 $M = \langle \Sigma, S, Move, s_0, F \rangle$ 是 NFA，其对应的 DFA $D = \langle \Sigma, \mathcal{S}, Trans, \int_0, \mathcal{F} \rangle$ 定义如下：

- $\mathcal{S} = \{\overline{T} \mid T \subseteq S\}$
- $\int_0 = \overline{\{s_0\}}$
- $\mathcal{F} = \{\overline{T} \mid \overline{T} \cap F \neq \emptyset\}$
- $Trans(\overline{T}, a) = \overline{\bigcup_{t \in \overline{T}} Move(t, a)}$

定理

由上定义得到的 DFA D 与原 NFA M 等价，即：

$$L(M) = L(D)$$

说明

DFA：$A \xrightarrow{a} B \Leftrightarrow$ NFA：$s_i \xrightarrow{\varepsilon} s_k \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s_m \xrightarrow{a} s_n \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s_q$

©hfwang

注释

DFA 的状态数

状态集合是 NFA 状态集合的幂集合，所以 DFA D 的状态数可能有 2^n, n 为 NFA M 的状态数.

定义

DFA 的一个状态称为可达状态 iff 存在一条从开始状态到该状态的有向路径.

说明

DFA D 的状态绝大多数是不可达状态.

注释

DFA 的状态数

状态集合是 NFA 状态集合的幂集合，所以 DFA D 的状态数可能有 2^n, n 为 NFA M 的状态数.

定义

DFA 的一个状态称为可达状态 iff 存在一条从开始状态到该状态的有向路径.

说明

DFA D 的状态绝大多数是不可达状态.

注释

DFA 的状态数

状态集合是 NFA 状态集合的幂集合，所以 DFA D 的状态数可能有 2^n, n 为 NFA M 的状态数.

定义

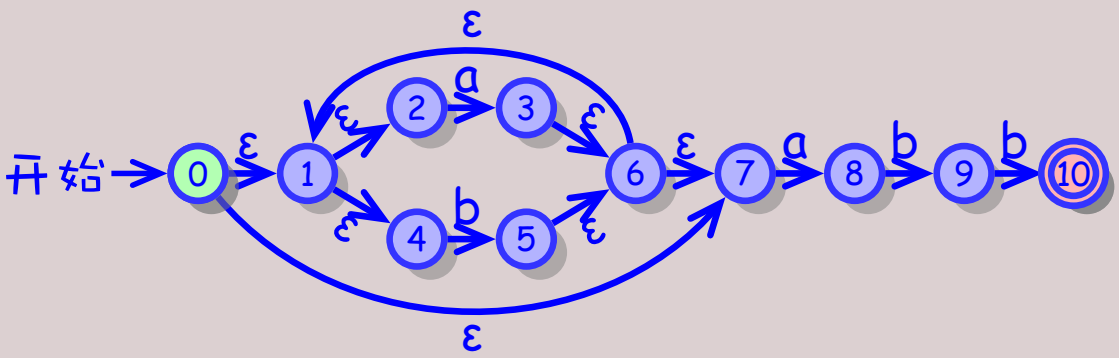
DFA 的一个状态称为可达状态 iff 存在一条从开始状态到该状态的有向路径.

说明

DFA D 的状态绝大多数是不可达状态.

求可达状态

Example: NFA of (a | b)*abb

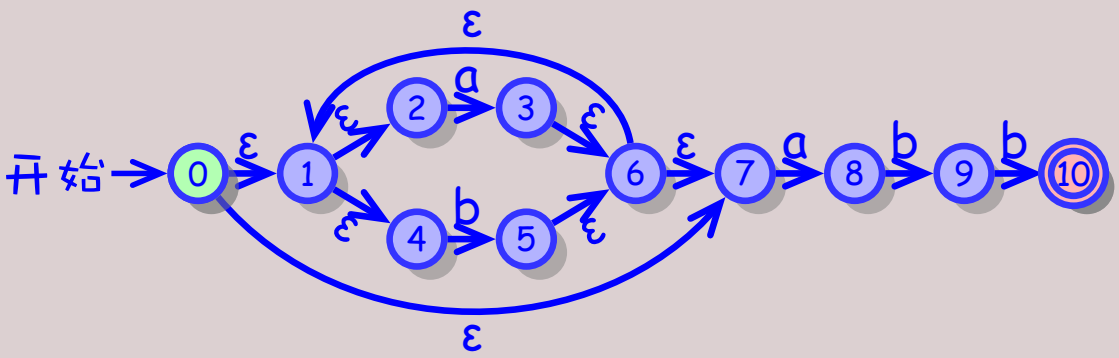


求可达状态

1. $\overline{\{0\}} = \{0,1,2,4,7\} =$
   /* 开始状态 */ $= A$;
2. $Trans(A,a) = \{3,8\} =$
   $= \{2,3,6,7,1,4\} = B$
3. $Trans(A,b) = \{5\} =$
   $= \{6,7,1,2,4\} = C$
4. $Trans(B,a) = \{3,8\} = B$
5. $Trans(B,b) = \{5,9\} =$
   $= \{6,7,1,2,4\} = D$
6. $Trans(C,a) = \{3,8\} = B$
7. $Trans(C,b) = \{5\} = C$
8. $Trans(D,a) = \{3,8\} = B$
9. $Trans(D,b) = \{5,10\} =$
   $= \{6,7,1,2,4\} = E$
10. $Trans(E,a) = \{3,8\} = B$
11. $Trans(E,b) = \{5\} = C$

求可达状态

Example: NFA of (a | b)*abb



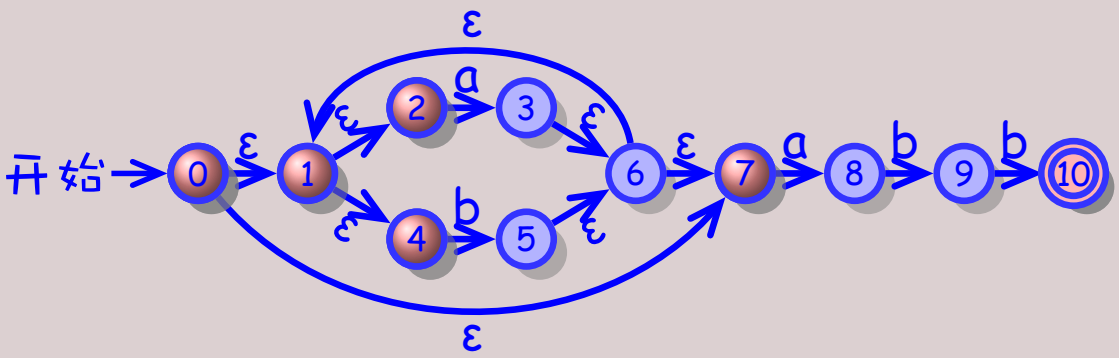
求可达状态

1. $\overline{\{0\}} = \{0,1,2,4,7\} =$ /* 开始状态 */ $\triangleq A$;
2. $\text{Trans}(A,a) = \overline{\{3,8\}} = \{3,8,6,7,1,2,4\} \triangleq B$;
3. $\text{Trans}(A,b) = \overline{\{5\}} = \{5,6,7,1,2,4\} \triangleq C$;
4. $\text{Trans}(B,a) = \overline{\{3,8\}} = B$;
5. $\text{Trans}(B,b) = \overline{\{5,9\}} = \{5,6,7,1,2,4,9\} \triangleq D$;
6. $\text{Trans}(C,a) = \overline{\{3,8\}} = B$;
7. $\text{Trans}(C,b) = \overline{\{5\}} = C$;
8. $\text{Trans}(D,a) = \overline{\{3,8\}} = B$;
9. $\text{Trans}(D,b) = \overline{\{5,10\}} = \{5,6,7,1,2,4,10\} \triangleq E$;
10. $\text{Trans}(E,a) = \overline{\{3,8\}} = B$;
11. $\text{Trans}(E,b) = \overline{\{5\}} = C$;

©hfwang

求可达状态

Example: NFA of (a | b)*abb

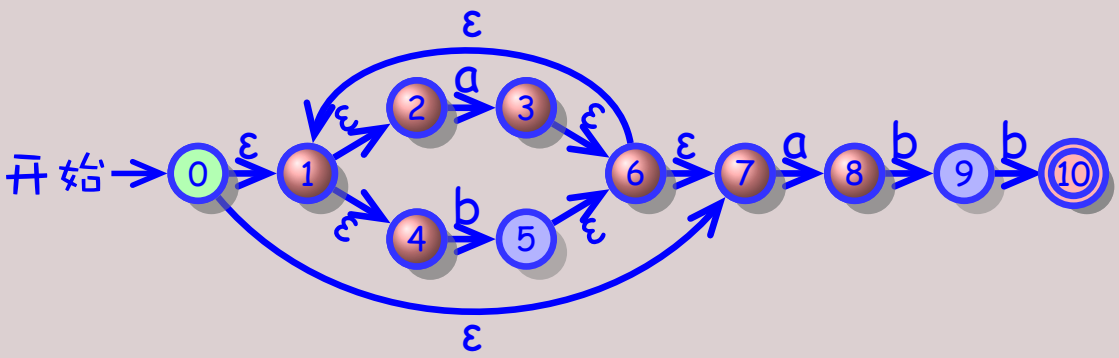


求可达状态

1. $\overline{\{0\}} = \{0,1,2,4,7\} =$ /* 开始状态 */ $\triangleq A$;
2. $\text{Trans}(A,a) = \overline{\{3,8\}} = \{3,8,6,7,1,2,4\} \triangleq B$;
3. $\text{Trans}(A,b) = \overline{\{5\}} = \{5,6,7,1,2,4\} \triangleq C$;
4. $\text{Trans}(B,a) = \overline{\{3,8\}} = B$;
5. $\text{Trans}(B,b) = \overline{\{5,9\}} = \{5,6,7,1,2,4,9\} \triangleq D$;
6. $\text{Trans}(C,a) = \overline{\{3,8\}} = B$;
7. $\text{Trans}(C,b) = \overline{\{5\}} = C$;
8. $\text{Trans}(D,a) = \overline{\{3,8\}} = B$;
9. $\text{Trans}(D,b) = \overline{\{5,10\}} = \{5,6,7,1,2,4,10\} \triangleq E$;
10. $\text{Trans}(E,a) = \overline{\{3,8\}} = B$;
11. $\text{Trans}(E,b) = \overline{\{5\}} = C$;

求可达状态

Example: NFA of (a | b)*abb

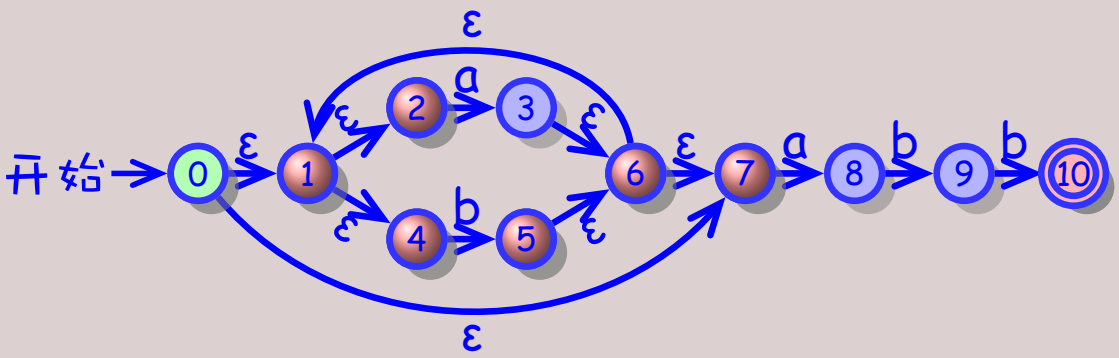


求可达状态

1. $\overline{\{0\}} = \{0,1,2,4,7\} =$ /* 开始状态 */ $\triangleq$ A;
2. Trans(A, a) = $\overline{\{3,8\}} = \{3,8,6,7,1,2,4\} \triangleq$ B;
3. Trans(A, b) = $\overline{\{5\}} = \{5,6,7,1,2,4\} \triangleq$ C;
4. Trans(B, a) = $\overline{\{3,8\}}$ = B;
5. Trans(B, b) = $\overline{\{5,9\}} = \{5,6,7,1,2,4,9\} \triangleq$ D;
6. Trans(C, a) = $\overline{\{3,8\}}$ = B;
7. Trans(C, b) = $\overline{\{5\}}$ = C;
8. Trans(D, a) = $\overline{\{3,8\}}$ = B;
9. Trans(D, b) = $\overline{\{5,10\}} = \{5,6,7,1,2,4,10\} \triangleq$ E;
10. Trans(E, a) = $\overline{\{3,8\}}$ = B;
11. Trans(E, b) = $\overline{\{5\}}$ = C;

求可达状态

Example: NFA of (a | b)*abb



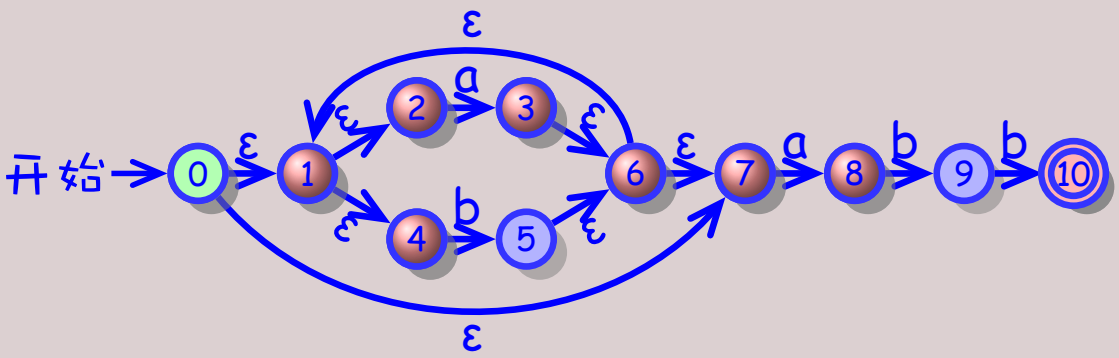
求可达状态

1. $\overline{\{0\}} = \{0,1,2,4,7\} =$ /* 开始状态 */ $\triangleq A$;
2. $\text{Trans}(A,a) = \overline{\{3,8\}} = \{3,8,6,7,1,2,4\} \triangleq B$;
3. $\text{Trans}(A,b) = \overline{\{5\}} = \{5,6,7,1,2,4\} \triangleq C$;
4. $\text{Trans}(B,a) = \overline{\{3,8\}} = B$;
5. $\text{Trans}(B,b) = \overline{\{5,9\}} = \{5,6,7,1,2,4,9\} \triangleq D$;
6. $\text{Trans}(C,a) = \overline{\{3,8\}} = B$;
7. $\text{Trans}(C,b) = \overline{\{5\}} = C$;
8. $\text{Trans}(D,a) = \overline{\{3,8\}} = B$;
9. $\text{Trans}(D,b) = \overline{\{5,10\}} = \{5,6,7,1,2,4,10\} \triangleq E$;
10. $\text{Trans}(E,a) = \overline{\{3,8\}} = B$;
11. $\text{Trans}(E,b) = \overline{\{5\}} = C$;

©hfwang

求可达状态

Example: NFA of (a | b)*abb



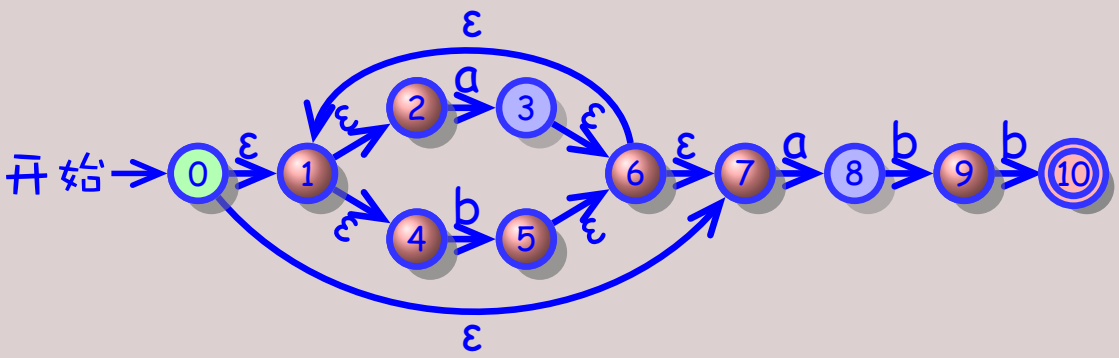
求可达状态

1. $\overline{\{0\}} = \{0,1,2,4,7\} =$ /* 开始状态 */ $\triangleq A$;
2. $\text{Trans}(A,a) = \overline{\{3,8\}} = \{3,8,6,7,1,2,4\} \triangleq B$;
3. $\text{Trans}(A,b) = \overline{\{5\}} = \{5,6,7,1,2,4\} \triangleq C$;
4. $\text{Trans}(B,a) = \overline{\{3,8\}} = B$;
5. $\text{Trans}(B,b) = \overline{\{5,9\}} = \{5,6,7,1,2,4,9\} \triangleq D$;
6. $\text{Trans}(C,a) = \overline{\{3,8\}} = B$;
7. $\text{Trans}(C,b) = \overline{\{5\}} = C$;
8. $\text{Trans}(D,a) = \overline{\{3,8\}} = B$;
9. $\text{Trans}(D,b) = \overline{\{5,10\}} = \{5,6,7,1,2,4,10\} \triangleq E$;
10. $\text{Trans}(E,a) = \overline{\{3,8\}} = B$;
11. $\text{Trans}(E,b) = \overline{\{5\}} = C$;

©hfwang

求可达状态

Example: NFA of (a | b)*abb



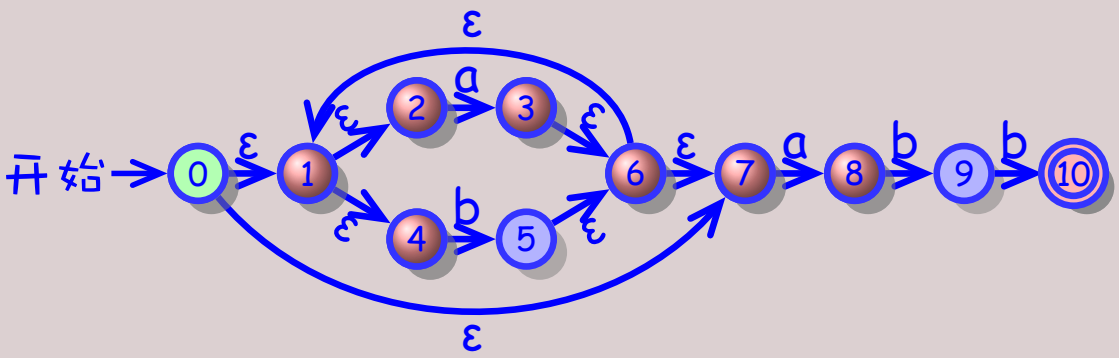
求可达状态

1. $\overline{\{0\}} = \{0,1,2,4,7\} =$ /* 开始状态 */ $\triangleq A$;
2. $\text{Trans}(A,a) = \overline{\{3,8\}} = \{3,8,6,7,1,2,4\} \triangleq B$;
3. $\text{Trans}(A,b) = \overline{\{5\}} = \{5,6,7,1,2,4\} \triangleq C$;
4. $\text{Trans}(B,a) = \overline{\{3,8\}} = B$;
5. $\text{Trans}(B,b) = \overline{\{5,9\}} = \{5,6,7,1,2,4,9\} \triangleq D$;
6. $\text{Trans}(C,a) = \overline{\{3,8\}} = B$;
7. $\text{Trans}(C,b) = \overline{\{5\}} = C$;
8. $\text{Trans}(D,a) = \overline{\{3,8\}} = B$;
9. $\text{Trans}(D,b) = \overline{\{5,10\}} = \{5,6,7,1,2,4,10\} \triangleq E$;
10. $\text{Trans}(E,a) = \overline{\{3,8\}} = B$;
11. $\text{Trans}(E,b) = \overline{\{5\}} = C$;

©hfwang

求可达状态

Example: NFA of (a | b)*abb

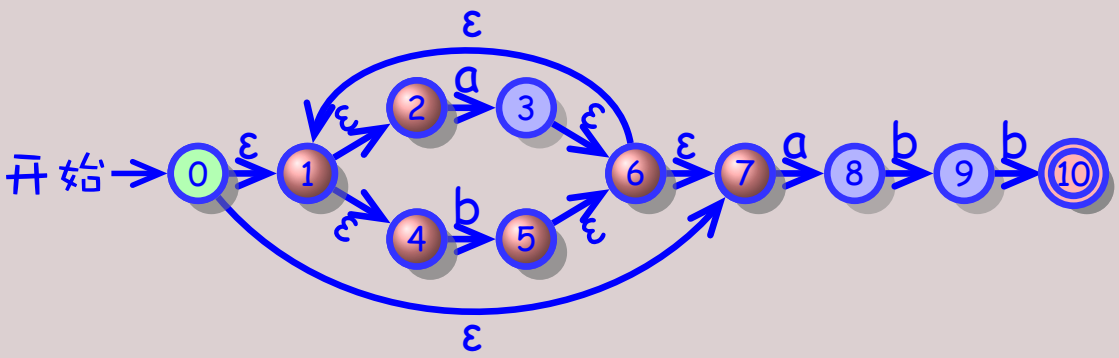


求可达状态

1. $\overline{\{0\}} = \{0,1,2,4,7\} =$ /* 开始状态 */ $\triangleq$ A;
2. Trans(A, a) = $\overline{\{3,8\}} = \{3,8,6,7,1,2,4\} \triangleq$ B;
3. Trans(A, b) = $\overline{\{5\}} = \{5,6,7,1,2,4\} \triangleq$ C;
4. Trans(B, a) = $\overline{\{3,8\}}$ = B;
5. Trans(B, b) = $\overline{\{5,9\}} = \{5,6,7,1,2,4,9\} \triangleq$ D;
6. Trans(C, a) = $\overline{\{3,8\}}$ = B;
7. Trans(C, b) = $\overline{\{5\}}$ = C;
8. Trans(D, a) = $\overline{\{3,8\}}$ = B;
9. Trans(D, b) = $\overline{\{5,10\}} = \{5,6,7,1,2,4,10\} \triangleq$ E;
10. Trans(E, a) = $\overline{\{3,8\}}$ = B;
11. Trans(E, b) = $\overline{\{5\}}$ = C;

求可达状态

Example: NFA of (a | b)*abb



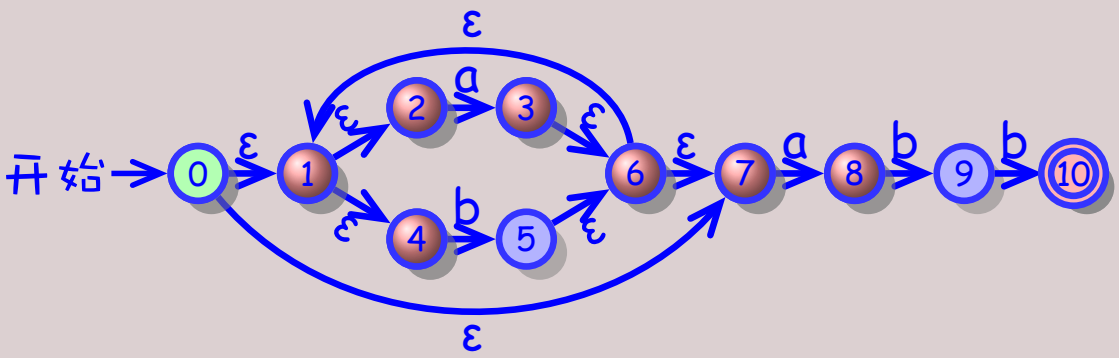
求可达状态

1. $\overline{\{0\}} = \{0,1,2,4,7\} =$ /* 开始状态 */ $\triangleq A$;
2. $\text{Trans}(A,a) = \overline{\{3,8\}} = \{3,8,6,7,1,2,4\} \triangleq B$;
3. $\text{Trans}(A,b) = \overline{\{5\}} = \{5,6,7,1,2,4\} \triangleq C$;
4. $\text{Trans}(B,a) = \overline{\{3,8\}} = B$;
5. $\text{Trans}(B,b) = \overline{\{5,9\}} = \{5,6,7,1,2,4,9\} \triangleq D$;
6. $\text{Trans}(C,a) = \overline{\{3,8\}} = B$;
7. $\text{Trans}(C,b) = \overline{\{5\}} = C$;
8. $\text{Trans}(D,a) = \overline{\{3,8\}} = B$;
9. $\text{Trans}(D,b) = \overline{\{5,10\}} = \{5,6,7,1,2,4,10\} \triangleq E$;
10. $\text{Trans}(E,a) = \overline{\{3,8\}} = B$;
11. $\text{Trans}(E,b) = \overline{\{5\}} = C$;

©hfwang

求可达状态

Example: NFA of (a | b)*abb

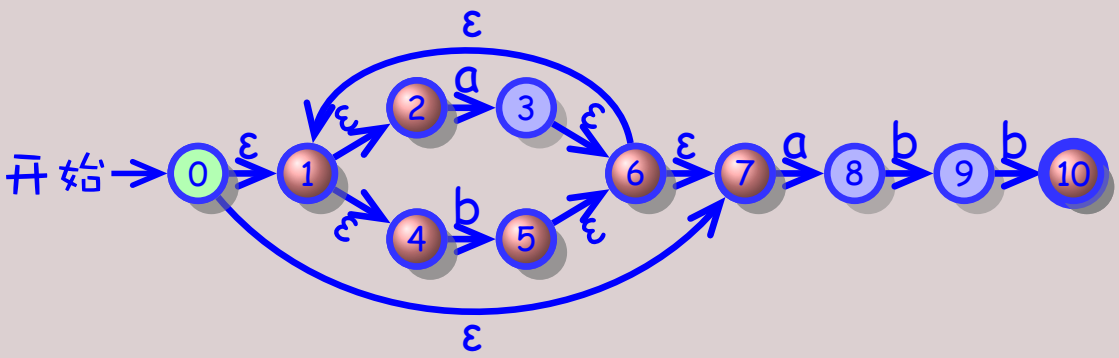


求可达状态

1. $\overline{\{0\}} = \{0,1,2,4,7\} =$ /* 开始状态 */ $\triangleq$ A;
2. Trans(A, a) = $\overline{\{3,8\}} = \{3,8,6,7,1,2,4\} \triangleq$ B;
3. Trans(A, b) = $\overline{\{5\}} = \{5,6,7,1,2,4\} \triangleq$ C;
4. Trans(B, a) = $\overline{\{3,8\}}$ = B;
5. Trans(B, b) = $\overline{\{5,9\}} = \{5,6,7,1,2,4,9\} \triangleq$ D;
6. Trans(C, a) = $\overline{\{3,8\}}$ = B;
7. Trans(C, b) = $\overline{\{5\}}$ = C;
8. Trans(D, a) = $\overline{\{3,8\}}$ = B;
9. Trans(D, b) = $\overline{\{5,10\}} = \{5,6,7,1,2,4,10\} \triangleq$ E;
10. Trans(E, a) = $\overline{\{3,8\}}$ = B;
11. Trans(E, b) = $\overline{\{5\}}$ = C;

求可达状态

Example: NFA of (a | b)*abb



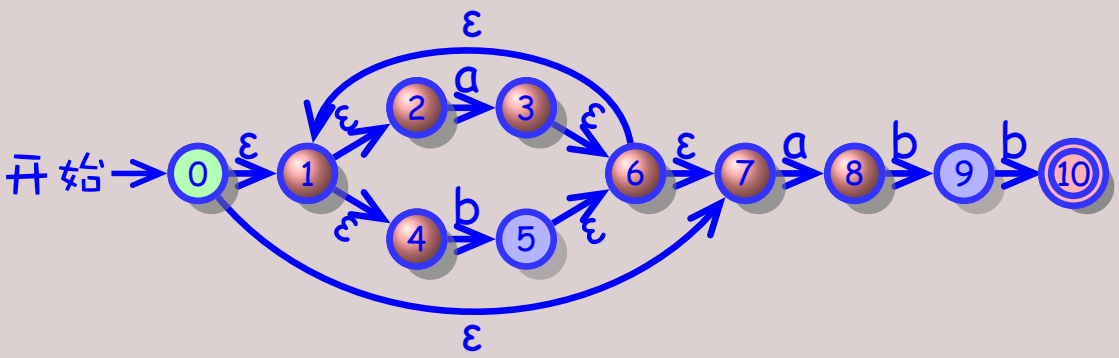
求可达状态

1. $\overline{\{0\}} = \{0,1,2,4,7\} =$ /* 开始状态 */ $\triangleq$ A;
2. Trans(A, a) = $\overline{\{3,8\}} = \{3,8,6,7,1,2,4\} \triangleq$ B;
3. Trans(A, b) = $\overline{\{5\}} = \{5,6,7,1,2,4\} \triangleq$ C;
4. Trans(B, a) = $\overline{\{3,8\}}$ = B;
5. Trans(B, b) = $\overline{\{5,9\}} = \{5,6,7,1,2,4,9\} \triangleq$ D;
6. Trans(C, a) = $\overline{\{3,8\}}$ = B;
7. Trans(C, b) = $\overline{\{5\}}$ = C;
8. Trans(D, a) = $\overline{\{3,8\}}$ = B;
9. Trans(D, b) = $\overline{\{5,10\}} = \{5,6,7,1,2,4,10\} \triangleq$ E;
10. Trans(E, a) = $\overline{\{3,8\}}$ = B;
11. Trans(E, b) = $\overline{\{5\}}$ = C;

©hfwang

求可达状态

Example: NFA of (a | b)*abb



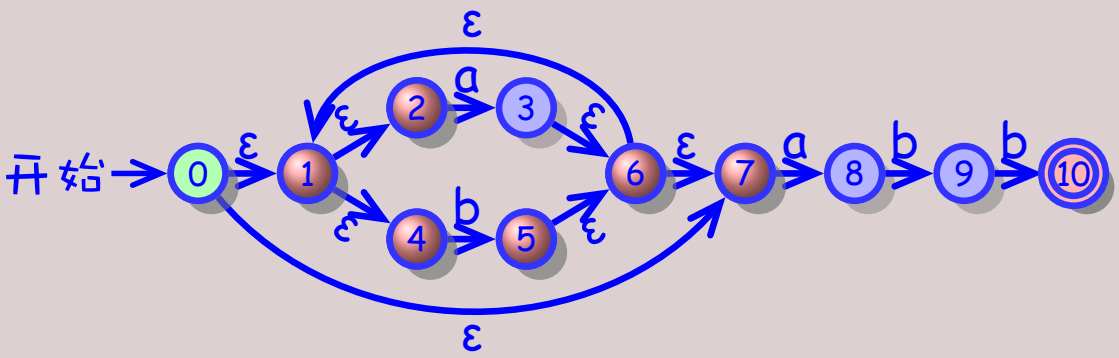
求可达状态

1. $\overline{\{0\}} = \{0,1,2,4,7\} =$ /* 开始状态 */ $\triangleq$ A;
2. Trans(A, a) = $\overline{\{3,8\}} = \{3,8,6,7,1,2,4\} \triangleq$ B;
3. Trans(A, b) = $\overline{\{5\}} = \{5,6,7,1,2,4\} \triangleq$ C;
4. Trans(B, a) = $\overline{\{3,8\}}$ = B;
5. Trans(B, b) = $\overline{\{5,9\}} = \{5,6,7,1,2,4,9\} \triangleq$ D;
6. Trans(C, a) = $\overline{\{3,8\}}$ = B;
7. Trans(C, b) = $\overline{\{5\}}$ = C;
8. Trans(D, a) = $\overline{\{3,8\}}$ = B;
9. Trans(D, b) = $\overline{\{5,10\}} = \{5,6,7,1,2,4,10\} \triangleq$ E;
10. Trans(E, a) = $\overline{\{3,8\}}$ = B;
11. Trans(E, b) = $\overline{\{5\}}$ = C;

©hfwang

求可达状态

Example: NFA of (a | b)*abb

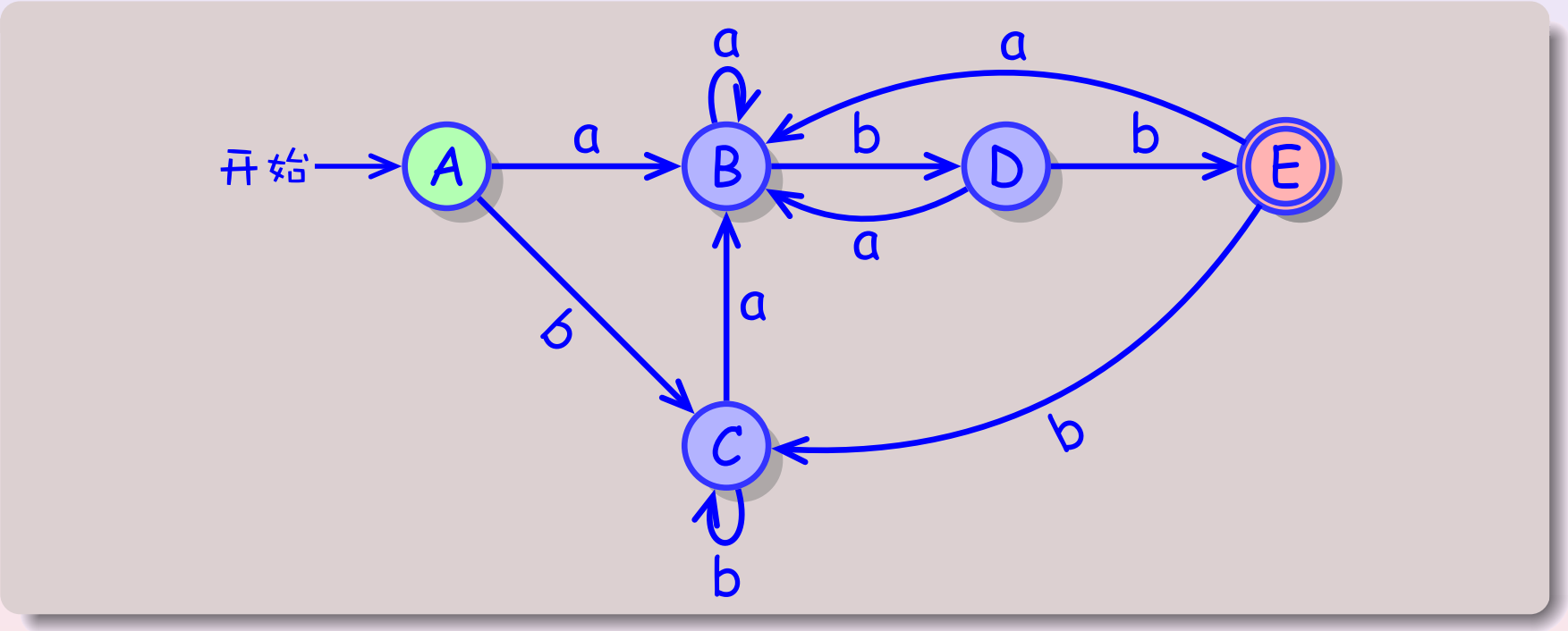


求可达状态

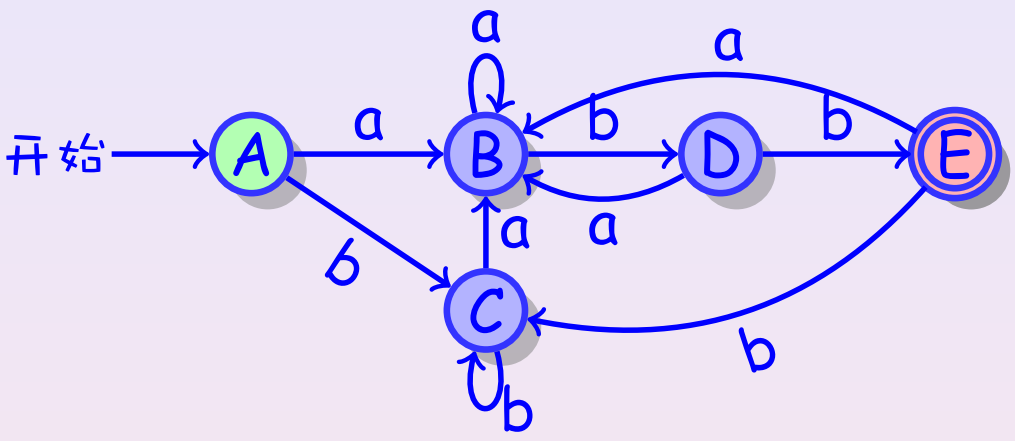
1. $\overline{\{0\}} = \{0,1,2,4,7\} =$ /* 开始状态 */ $\triangleq$ A;
2. Trans(A, a) = $\overline{\{3,8\}} = \{3,8,6,7,1,2,4\} \triangleq$ B;
3. Trans(A, b) = $\overline{\{5\}} = \{5,6,7,1,2,4\} \triangleq$ C;
4. Trans(B, a) = $\overline{\{3,8\}}$ = B;
5. Trans(B, b) = $\overline{\{5,9\}} = \{5,6,7,1,2,4,9\} \triangleq$ D;
6. Trans(C, a) = $\overline{\{3,8\}}$ = B;
7. Trans(C, b) = $\overline{\{5\}}$ = C;
8. Trans(D, a) = $\overline{\{3,8\}}$ = B;
9. Trans(D, b) = $\overline{\{5,10\}} = \{5,6,7,1,2,4,10\} \triangleq$ E;
10. Trans(E, a) = $\overline{\{3,8\}}$ = B;
11. Trans(E, b) = $\overline{\{5\}}$ = C;

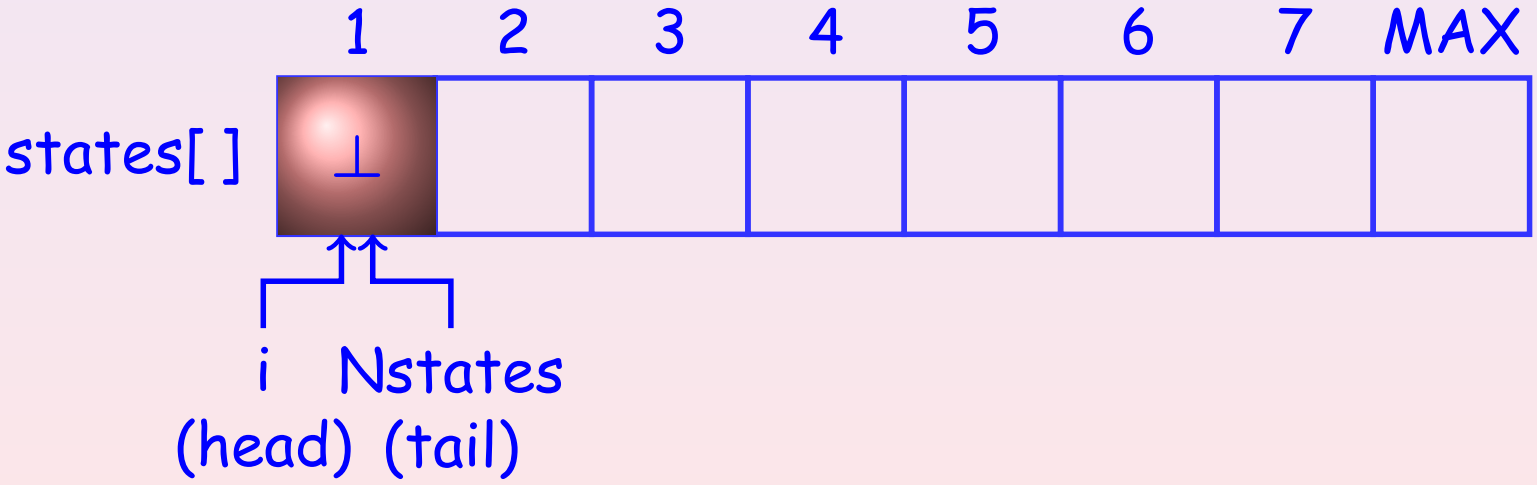
©hfwang

DFA of (a | b)*abb

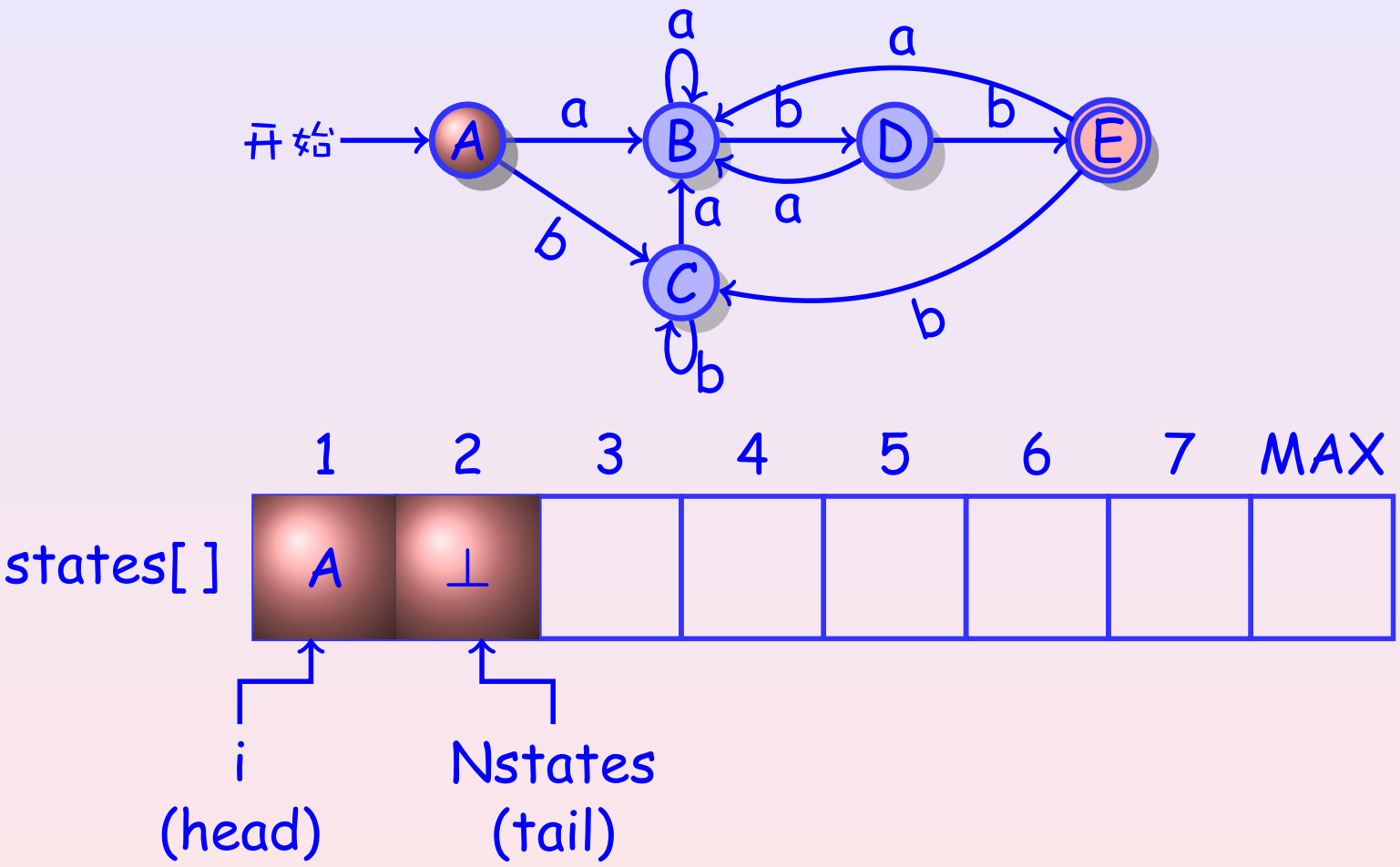


Example: Breath First Search 求 DFA 可达状态.

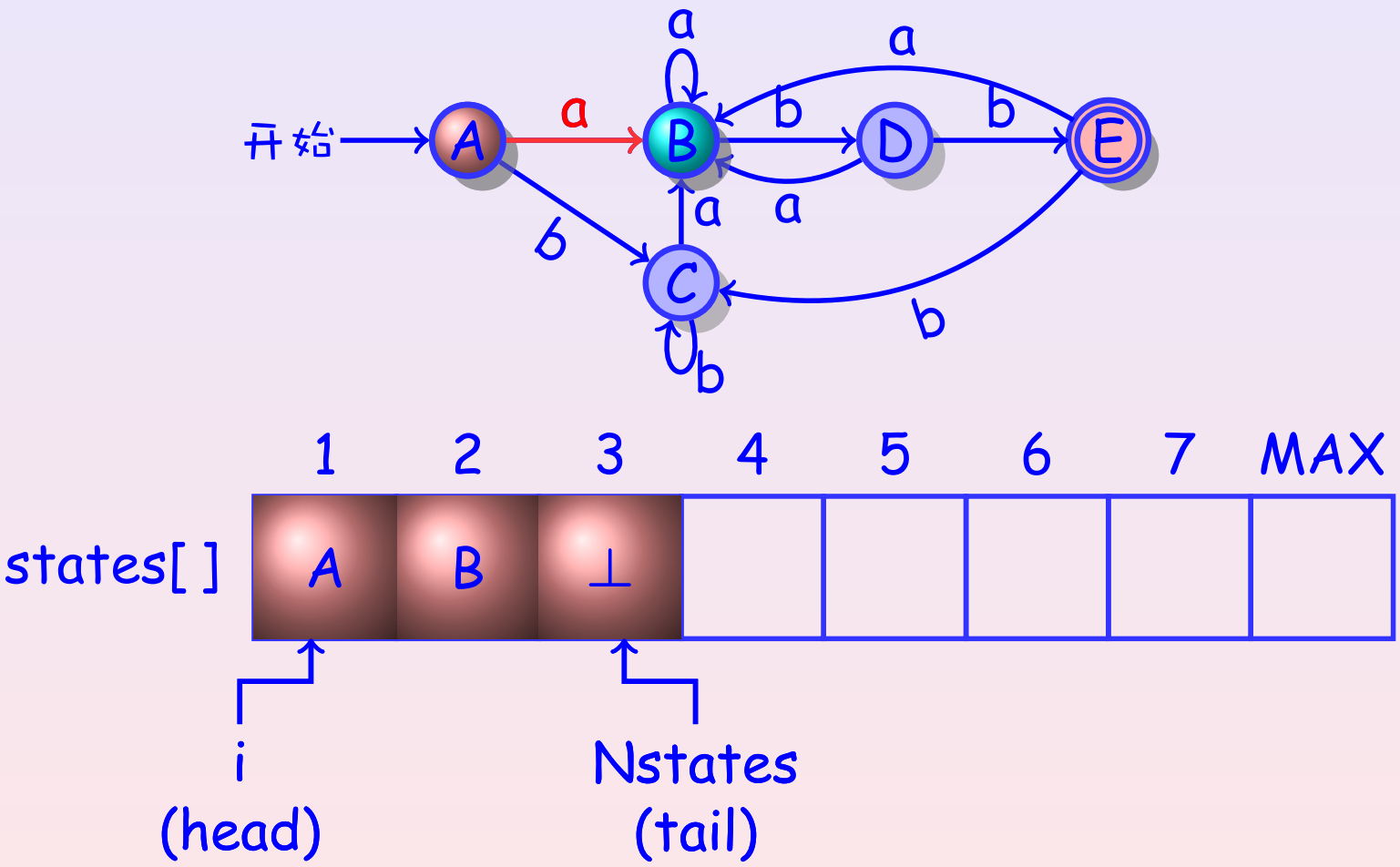




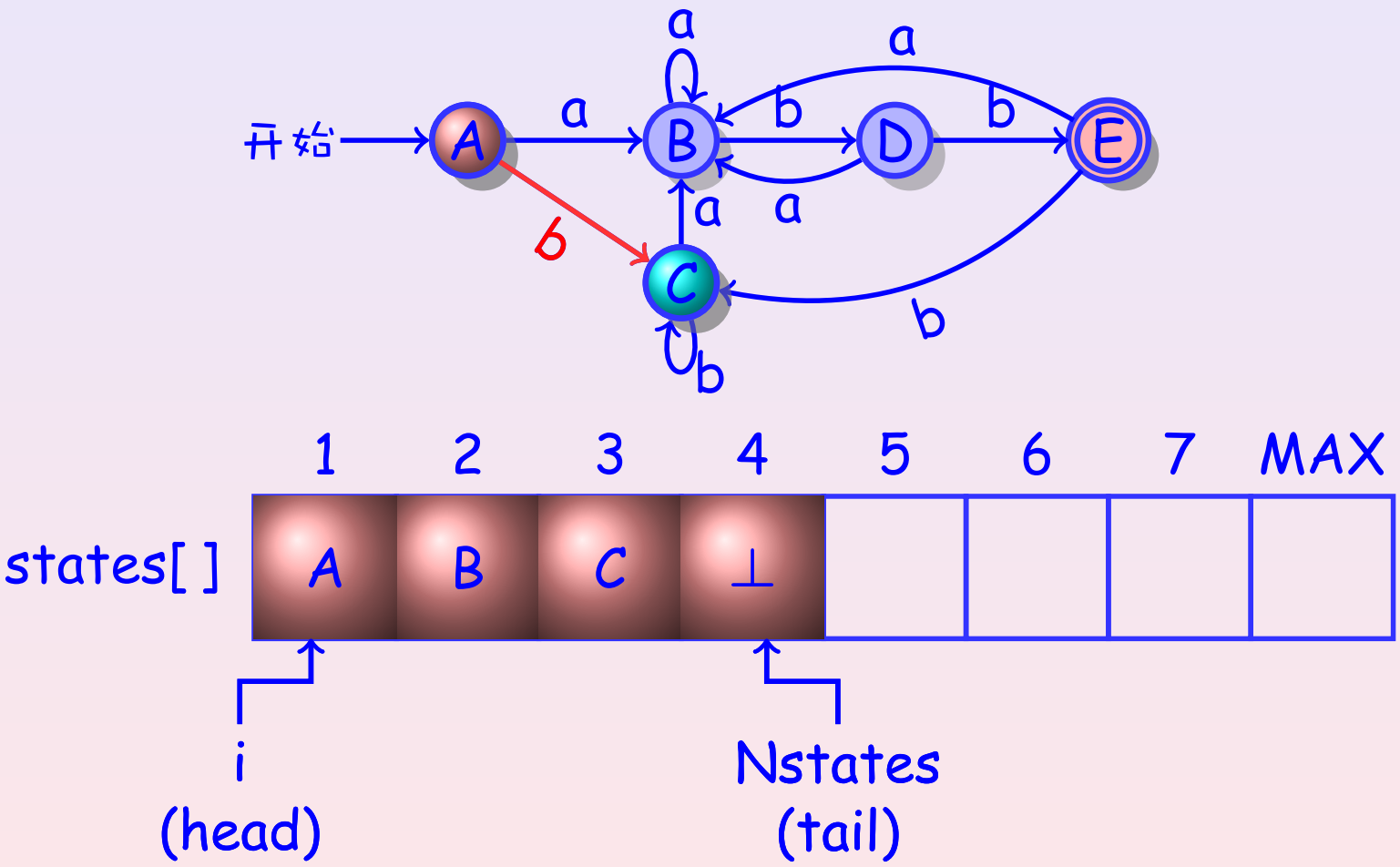
Example: Breath First Search 求 DFA 可达状态.



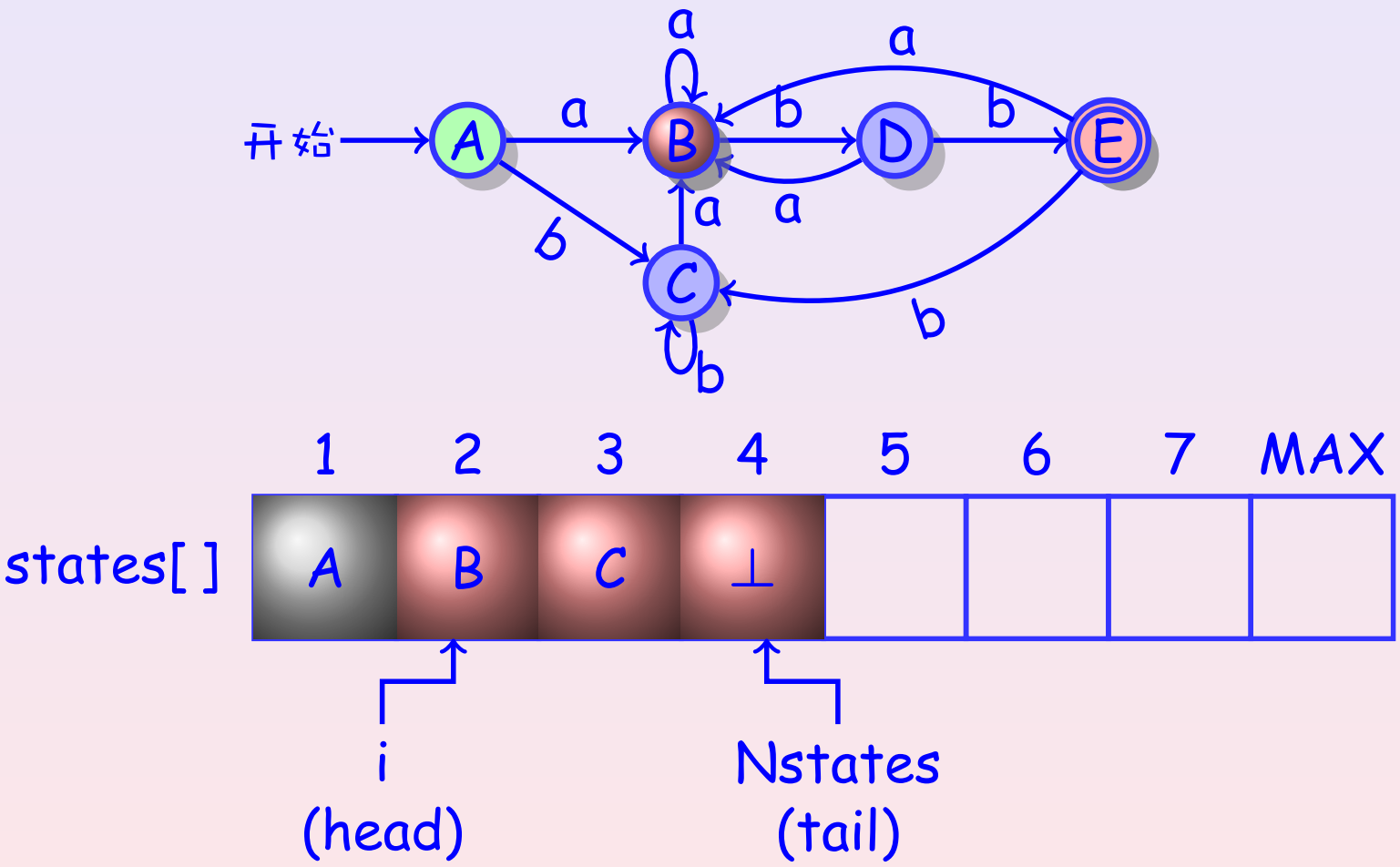
Example: Breath First Search 求 DFA 可达状态.



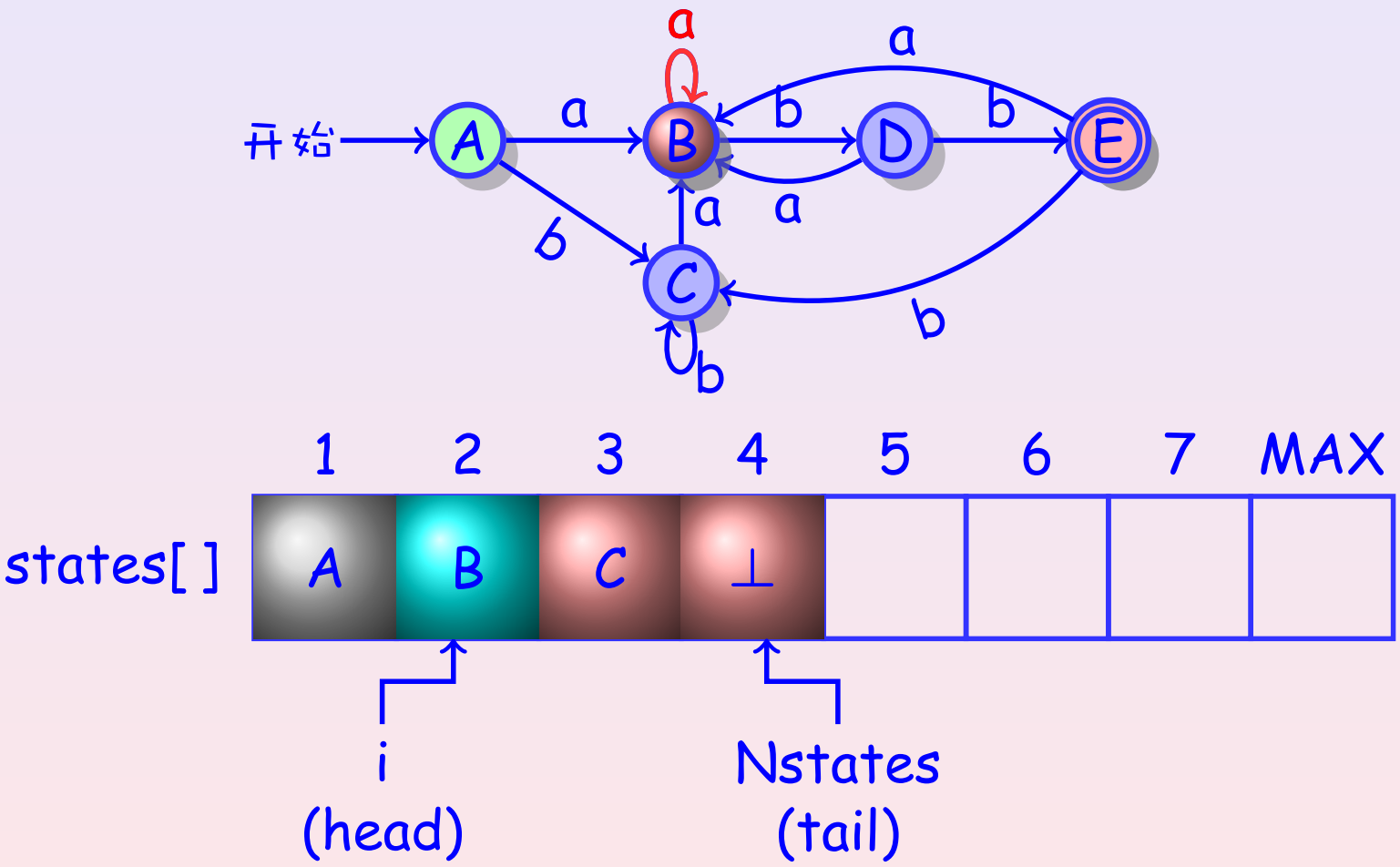
Example: Breath First Search 求 DFA 可达状态.



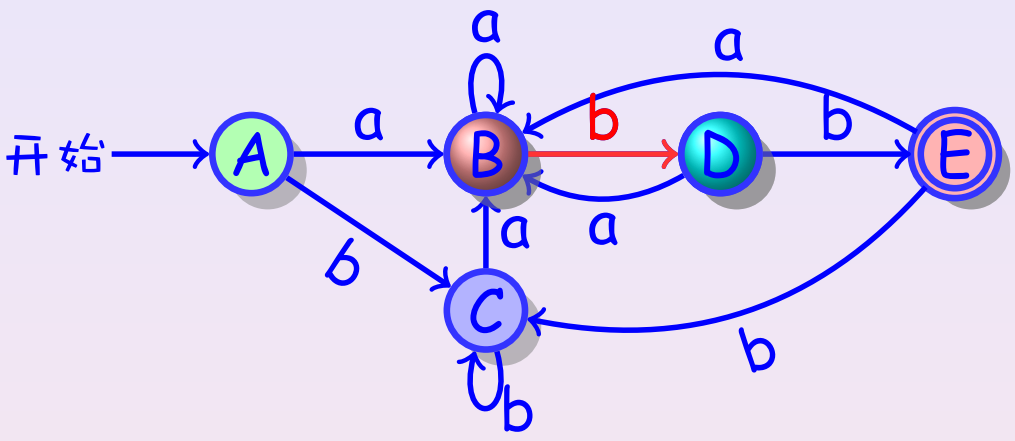
Example: Breath First Search 求 DFA 可达状态.

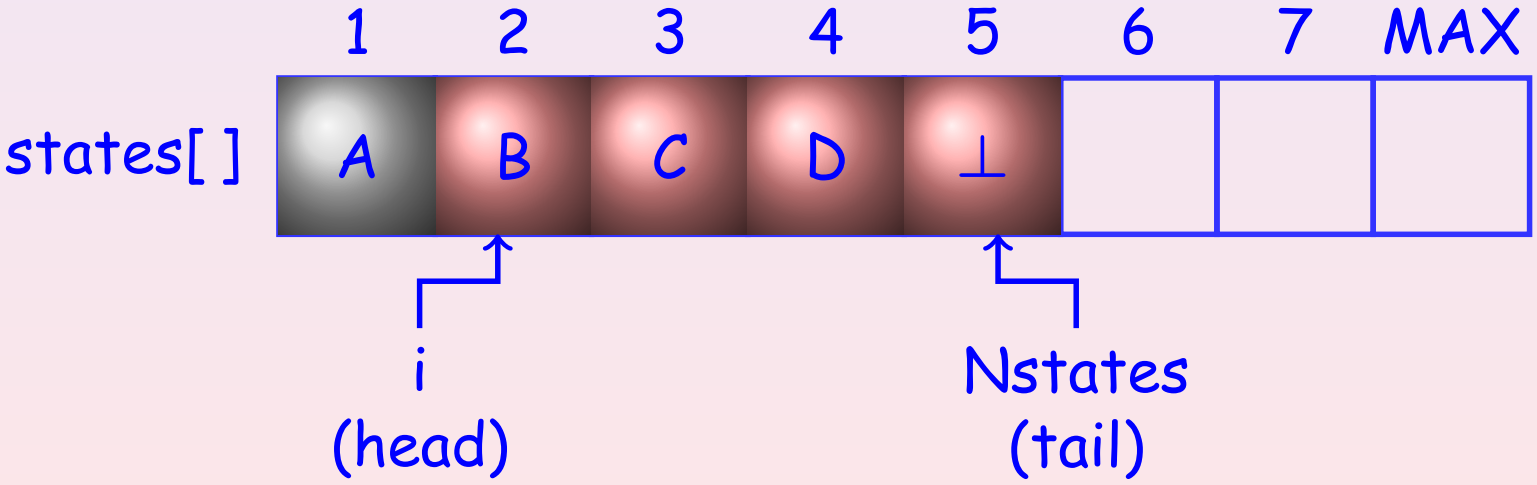


Example: Breath First Search 求 DFA 可达状态.

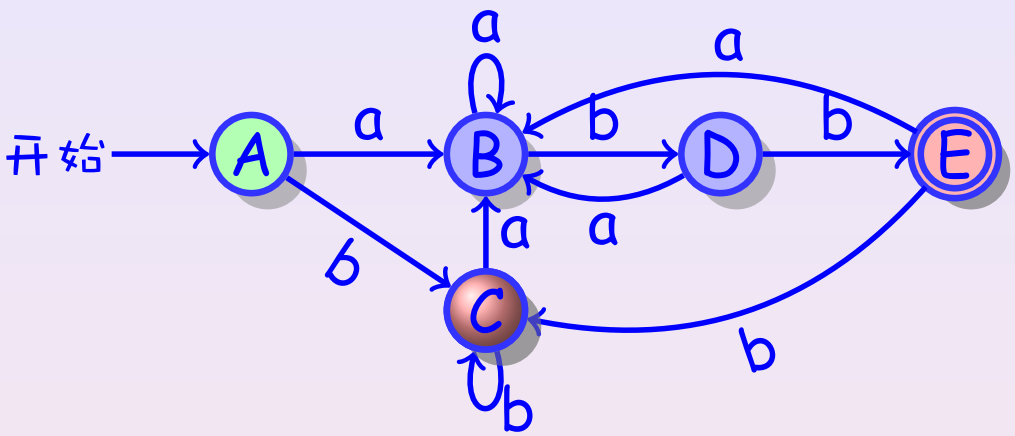


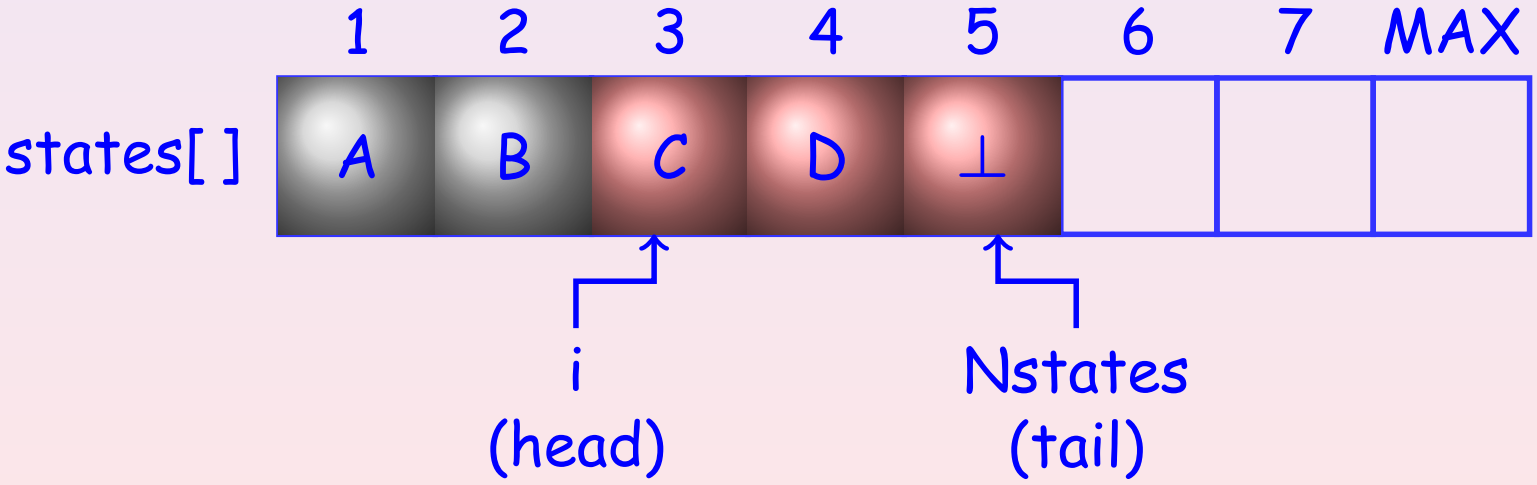
Example: Breath First Search 求 DFA 可达状态.



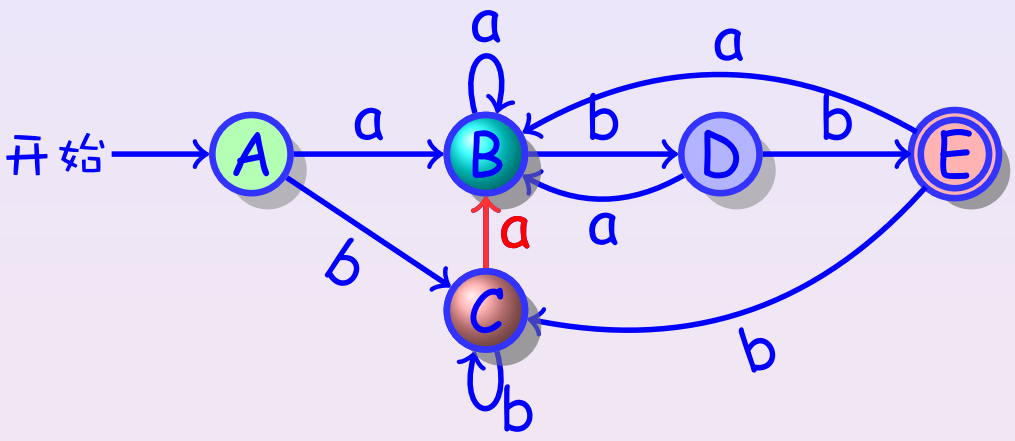


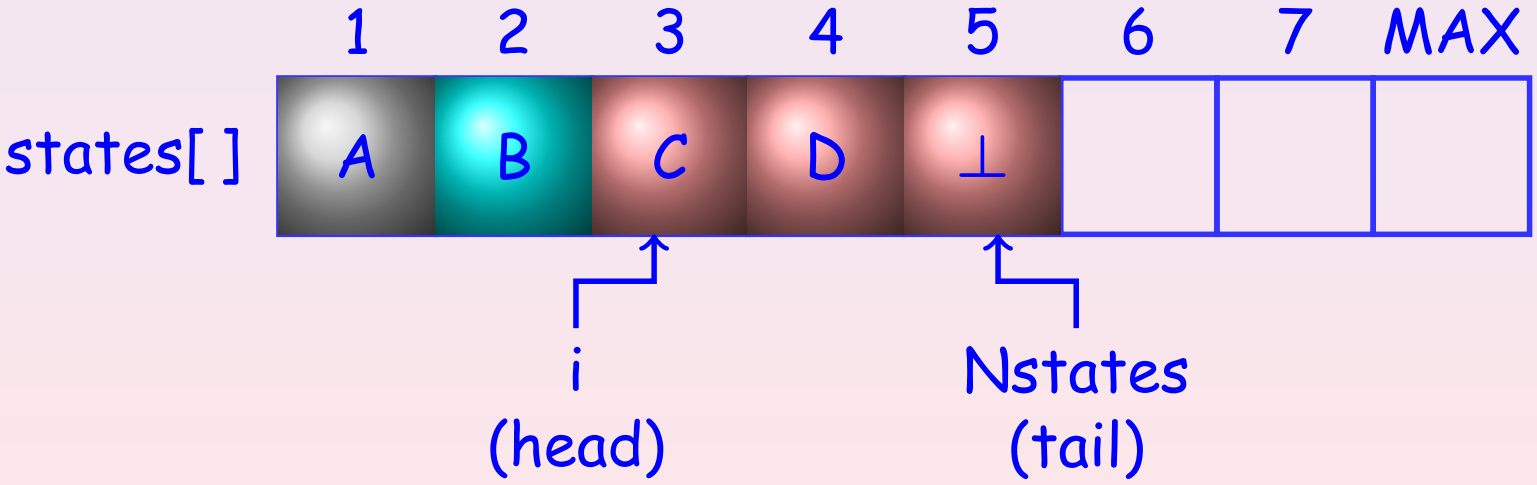
Example: Breath First Search 求 DFA 可达状态.



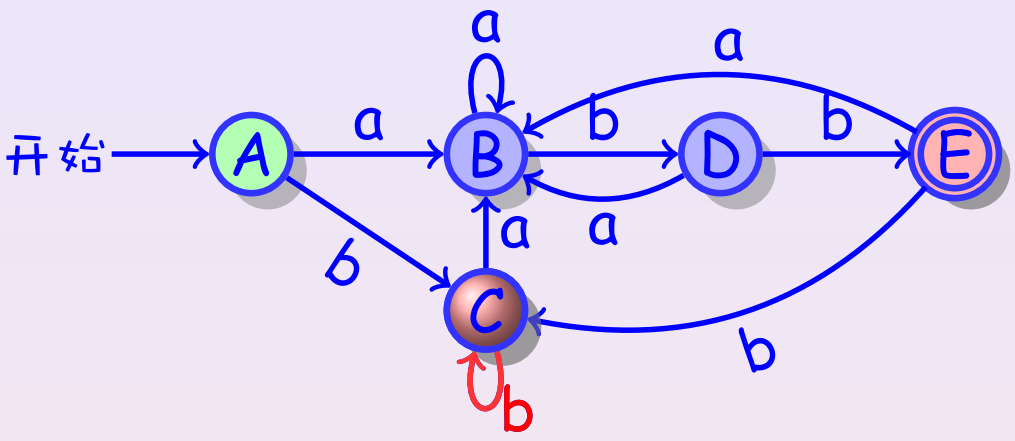


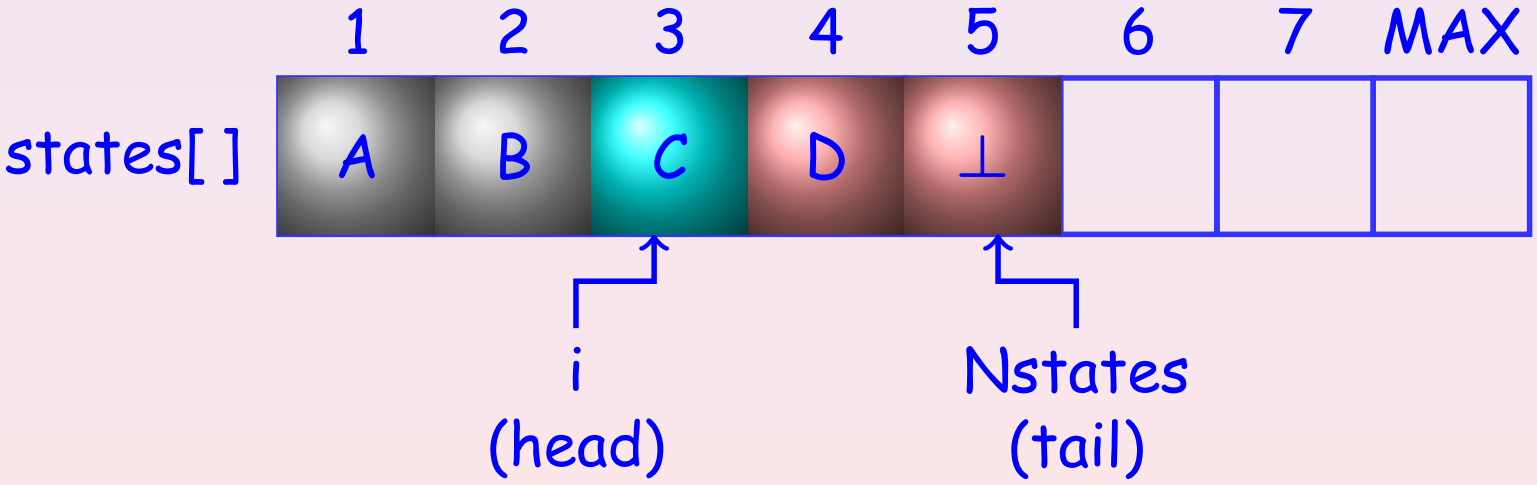
Example: Breath First Search 求 DFA 可达状态.



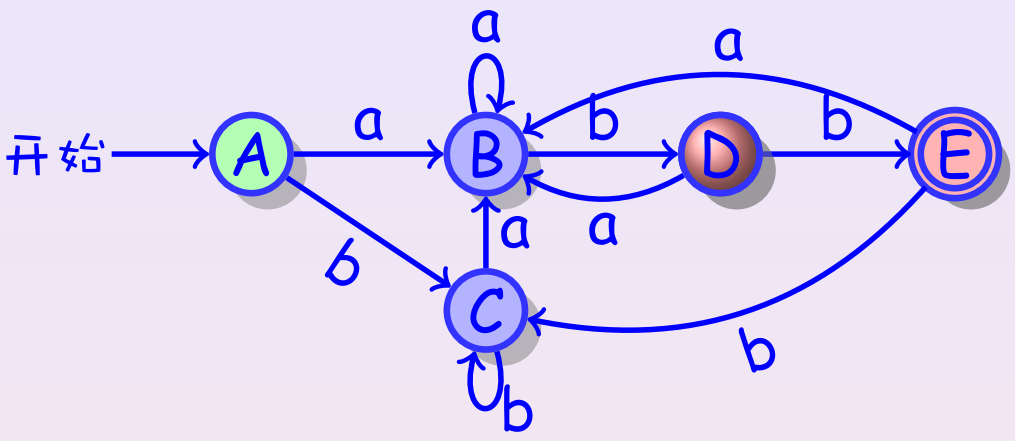


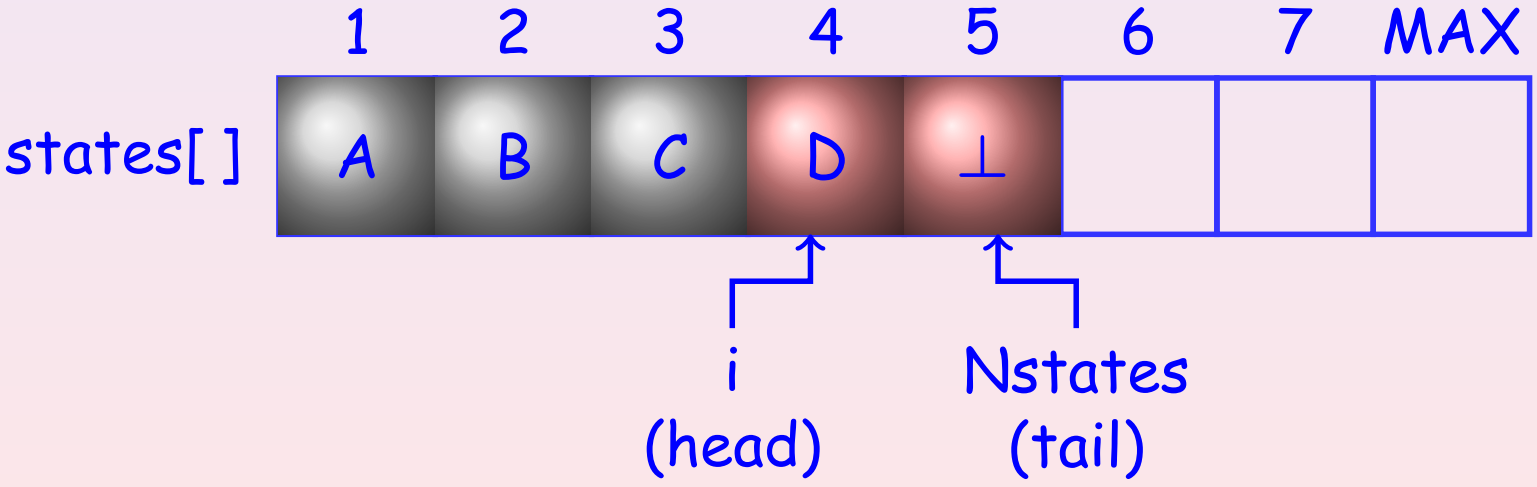
Example: Breath First Search 求 DFA 可达状态.



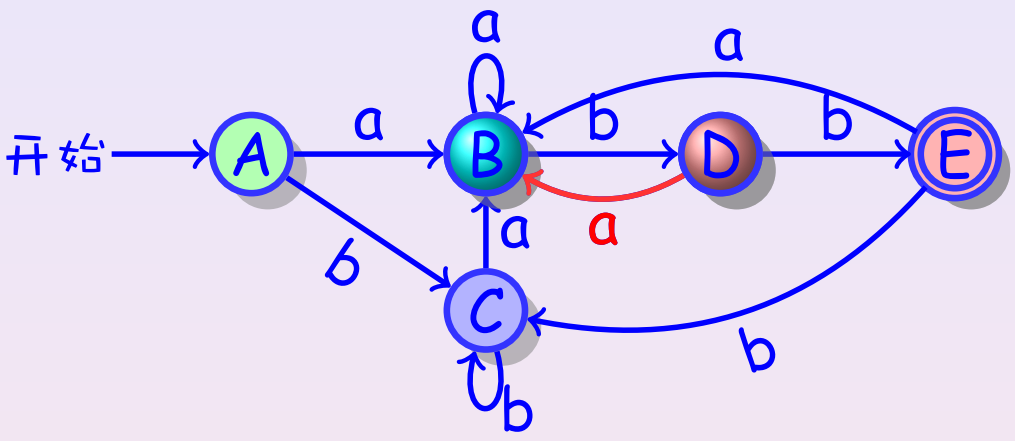


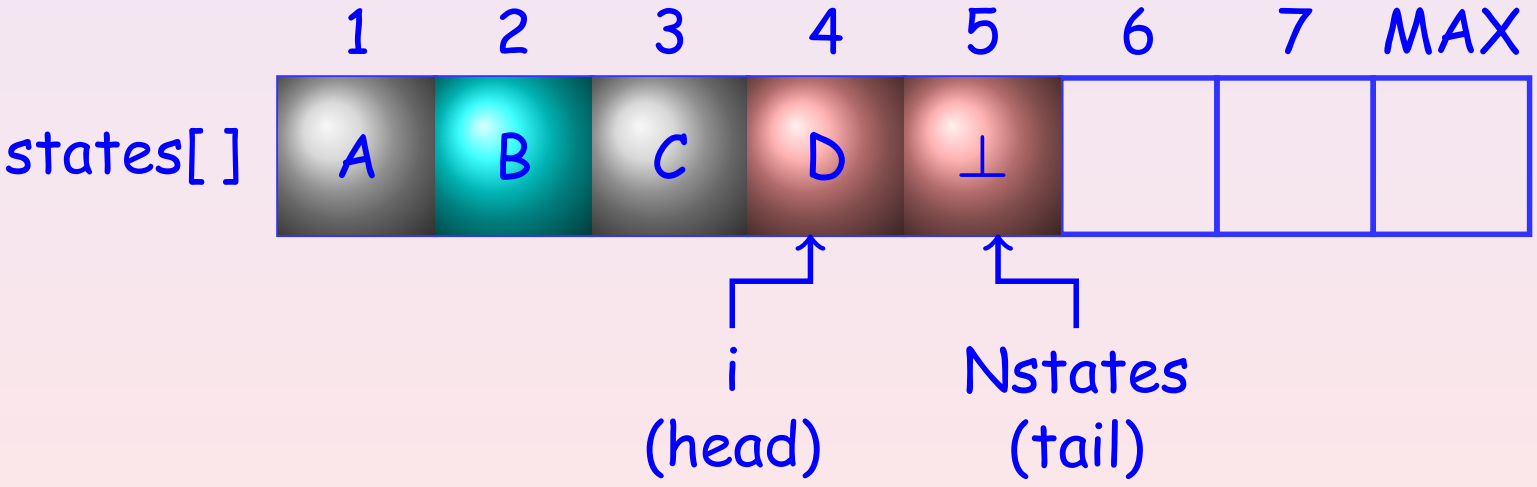
Example: Breath First Search 求 DFA 可达状态.



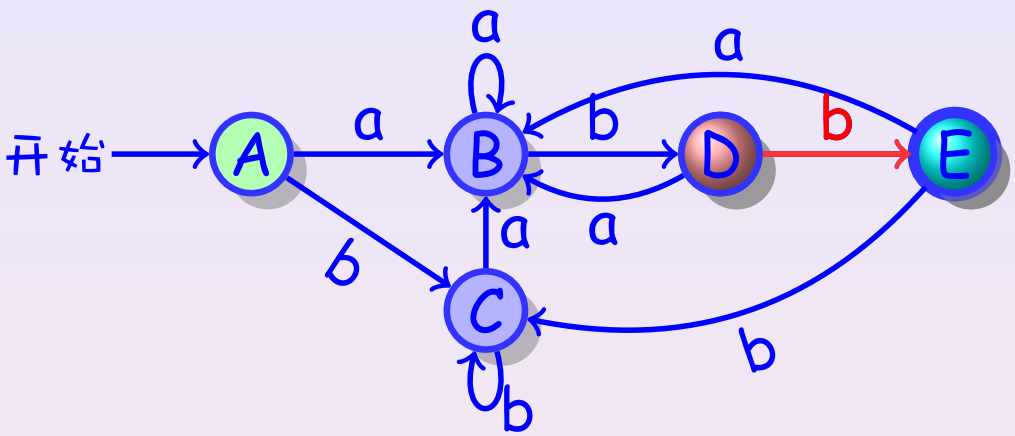


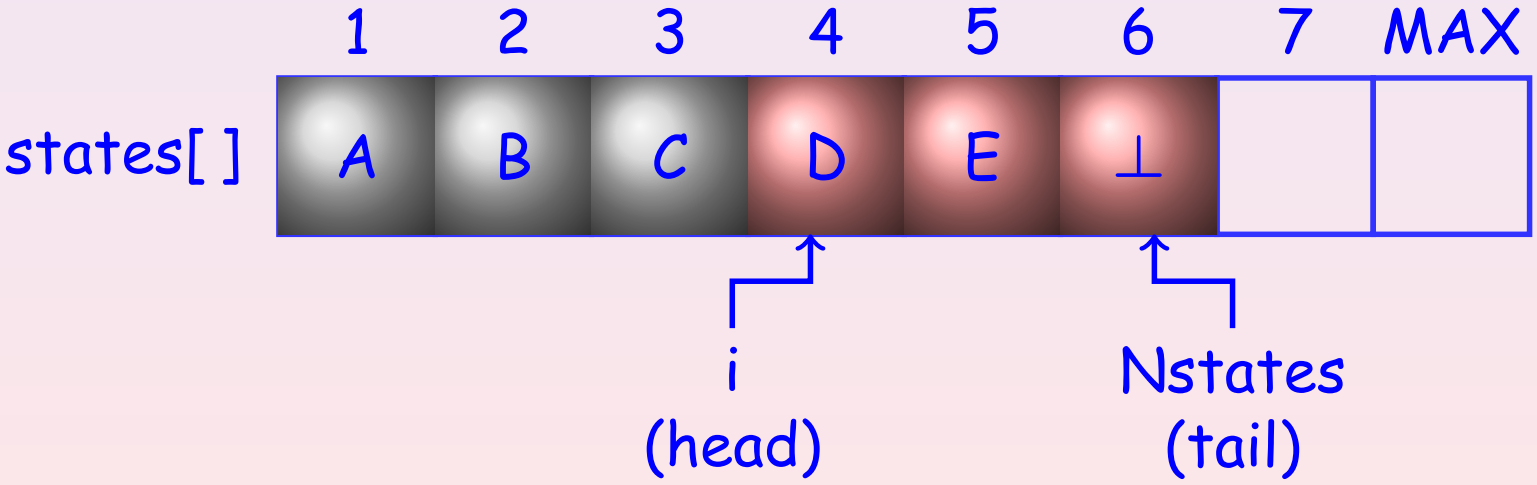
Example: Breath First Search 求 DFA 可达状态.



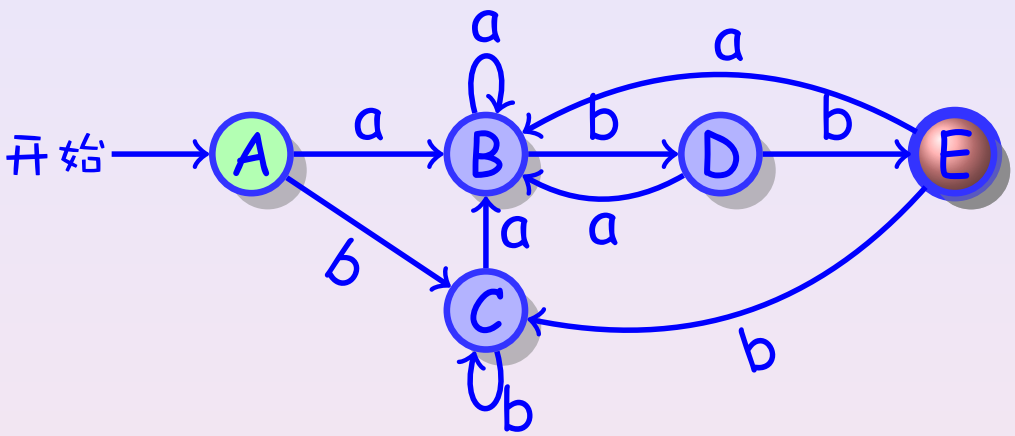


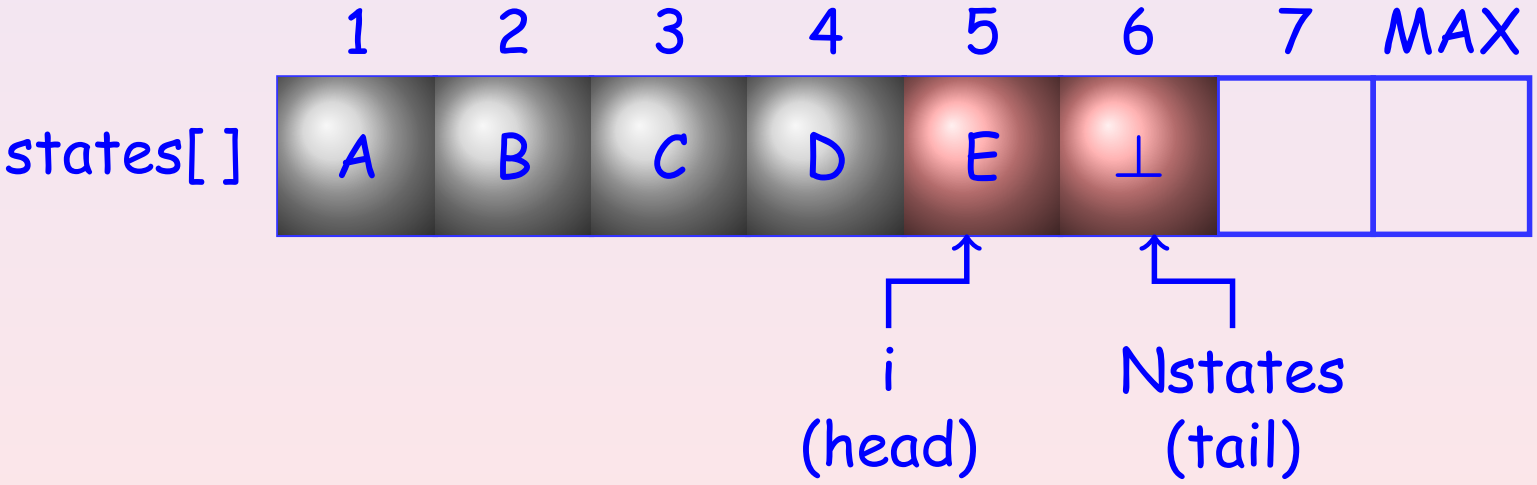
Example: Breath First Search 求 DFA 可达状态.



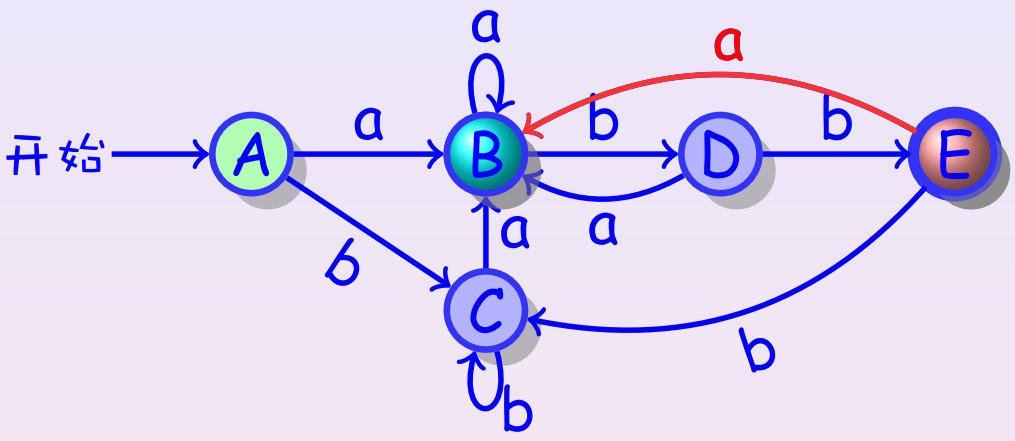


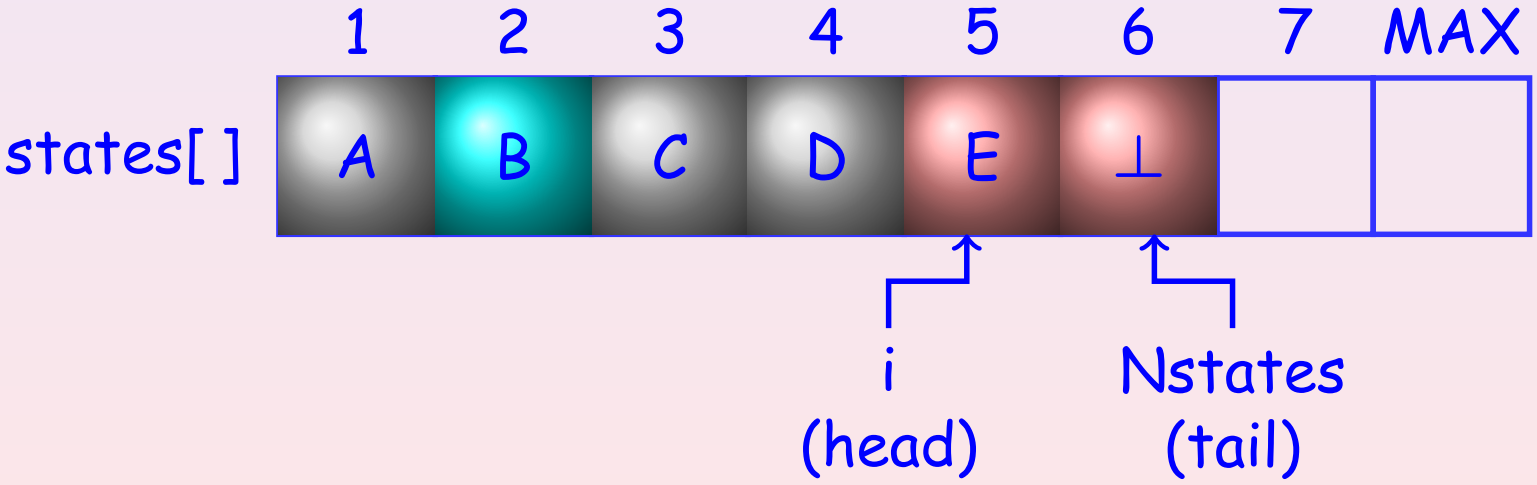
Example: Breath First Search 求 DFA 可达状态.



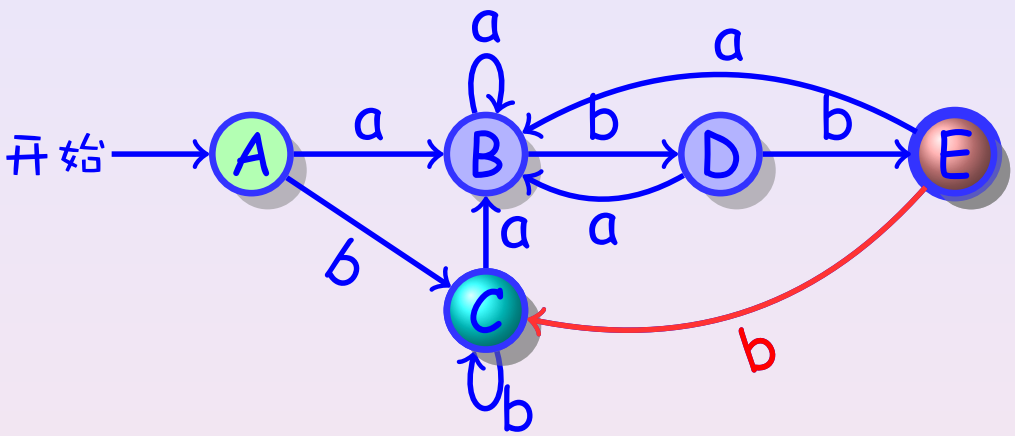


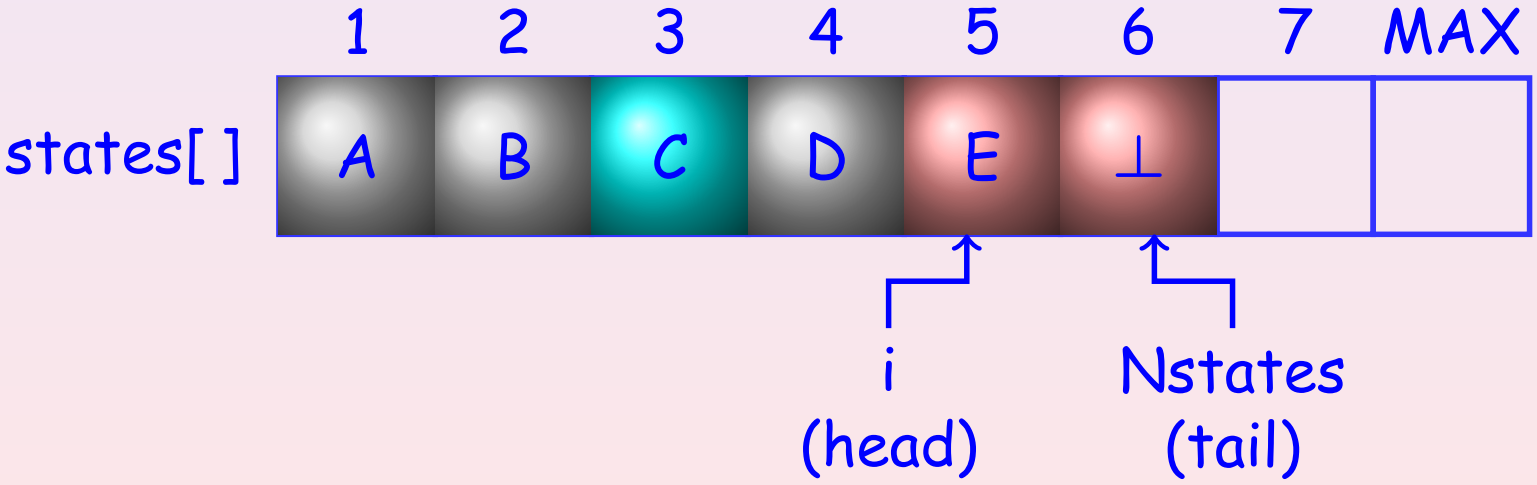
Example: Breath First Search 求 DFA 可达状态.



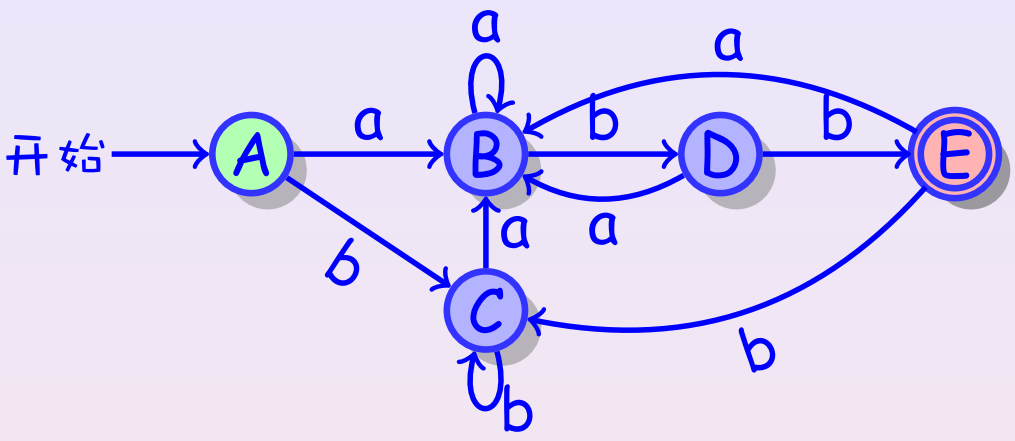


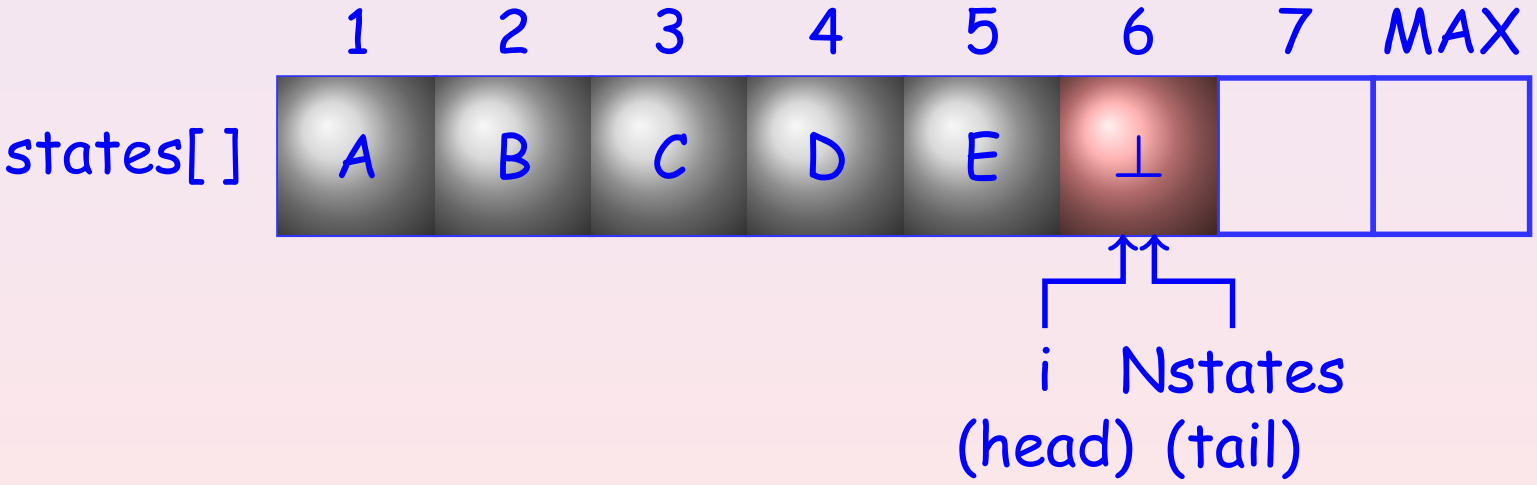
Example: Breath First Search 求 DFA 可达状态.





Example: Breath First Search 求 DFA 可达状态.





子集构造算法

数据结构:

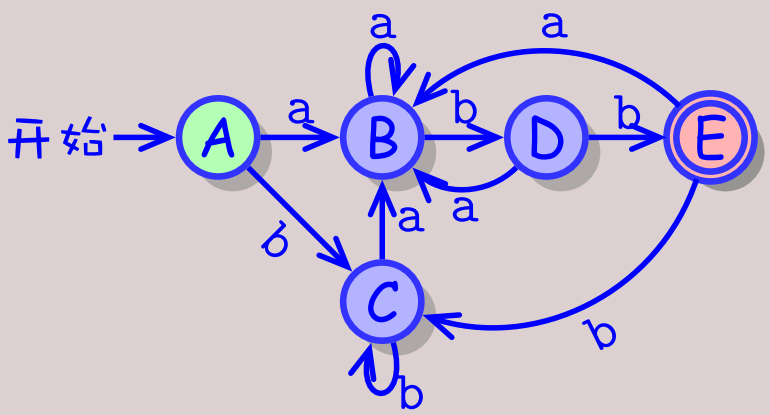
1、states[MAX] DFA状态数组，其元素的类型是NFA的状态Set;
2、int Trans [int][char] 状态转换矩阵;
3、int i; /* 当前的DFA状态号 */
4、int Nstates; /* 下一个可用的状态编号 */

```
int dfa(NFA M)
   /* 输出DFA最大状态数 */
{
  states[0] = ε-closure(M的初态);
  Nstates = 1;
  i = 0;
  Set s = ∅;
  Trans初始化为error;
```

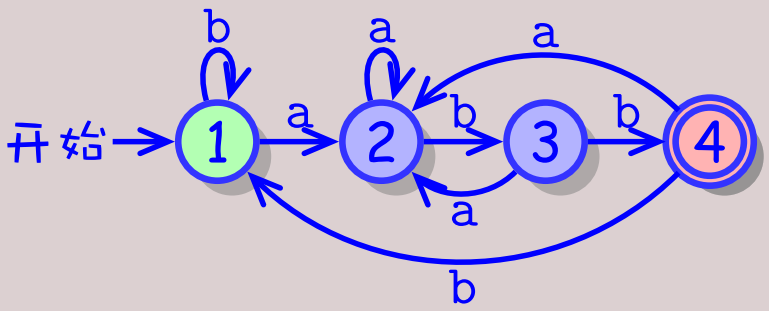
```
  while ( i < Nstates ) {
    for ( every c ∈ Σ)
      for (every states s ∈ s[i])
        t = ⋃ ε-closure(M.Move[s][c]);
      if ( t != ∅) {
        for ( j= 0; j < Nstates; j++ )
          if (states[j] == t) then
            break;
        Trans[i][c] = j;
        if ( j == Nstates ) Nstates++;
        }
      }
    i++;
  }
  return Nstates;
}
```

Example

算法和手工得到的 (a | b)*abb 对应的自动机的比较



(a) 算法转换

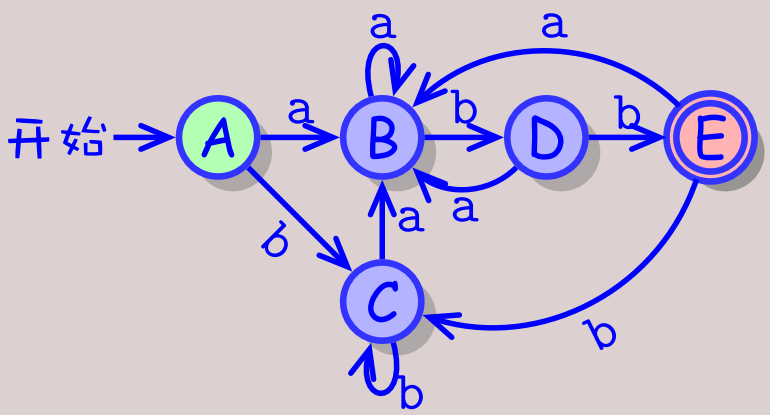


(b) 手工设计

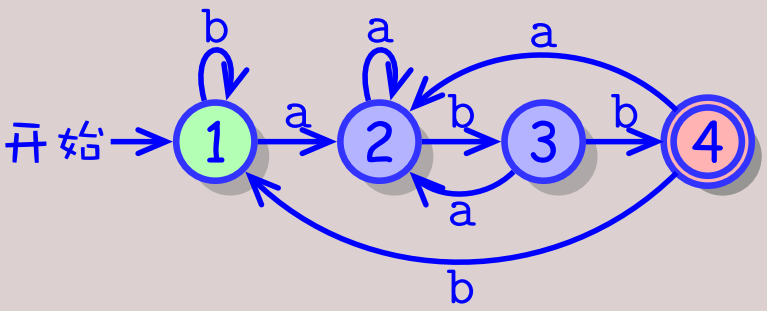
- 两者等价，但是后者状态数少，对应的扫描程序分析效率一样，但是后者可节省存储空间.
- 前者多了一个Ⓒ，从状态Ⓒ到Ⓔ所接受的和Ⓐ到Ⓔ接受的串一样，都要经过Ⓑ→Ⓐ→Ⓔ，两状态合并就和第二个自动机一样.

Example

算法和手工得到的 (a | b)*abb 对应的自动机的比较



(a) 算法转换

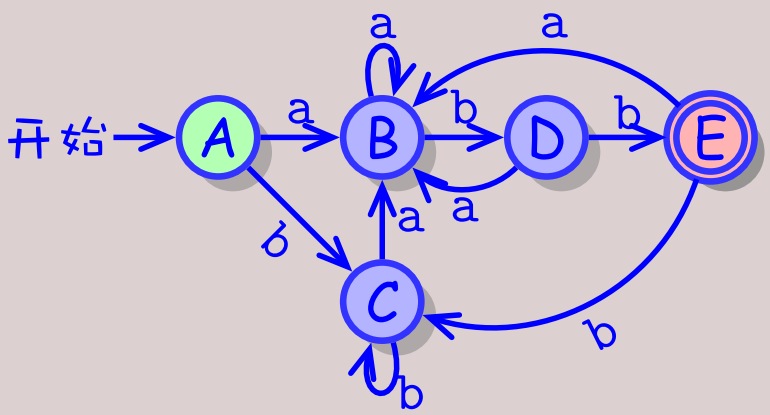


(b) 手工设计

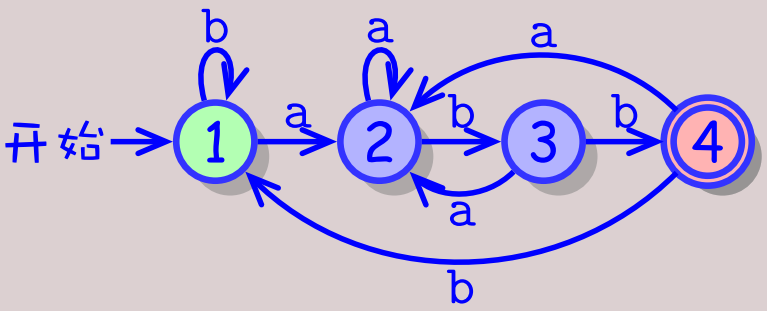
- 两者等价，但是后者状态数少，对应的扫描程序分析效率一样，但是后者可节省存储空间.
- 前者多了一个Ⓒ，从状态Ⓒ到Ⓔ所接受的和Ⓐ到Ⓔ接受的串一样，都要经过Ⓑ→Ⓐ→Ⓔ，两状态合并就和第二个自动机一样.

Example

算法和手工得到的 (a | b)*abb 对应的自动机的比较



(a) 算法转换



(b) 手工设计

- 两者等价，但是后者状态数少，对应的扫描程序分析效率一样，但是后者可节省存储空间。
- 前者多了一个C，从状态C到E所接受的和A到E接受的串一样，都要经过B→A→E，两状态合并就和第二个自动机一样。

可区别的状态

定义

设Ⓐ和Ⓑ是 DFA M 的两个状态，称Ⓐ和Ⓑ是可区别的 iff $\exists a \in \Sigma^*$和两个状态Ⓒ和Ⓓ，使得 $Ⓐ \xrightarrow{a} Ⓒ$，$Ⓑ \xrightarrow{a} Ⓓ$，并且，Ⓒ和Ⓓ有一个并仅有一个是接受状态.

性质

接受状态和非接受状态是可区别的，因为：若Ⓐ是非接受状态，Ⓑ是接受状态，则：

$$Ⓐ \xrightarrow{\varepsilon} Ⓐ,\ Ⓑ \xrightarrow{\varepsilon} Ⓑ.$$

性质

如果Ⓐ和Ⓑ是可区别的，并且$Ⓒ \xrightarrow{a} Ⓐ$，$Ⓓ \xrightarrow{a} Ⓑ$，则，Ⓒ和Ⓓ是可区别的. 因为：

$$Ⓒ \xrightarrow{a} Ⓐ \xrightarrow{a} Ⓖ$$
$$Ⓓ \xrightarrow{a} Ⓑ \xrightarrow{a} Ⓗ$$

可区别的状态

定义

设Ⓐ和Ⓑ是 DFA M 的两个状态，称Ⓐ和Ⓑ是可区别的 iff $\exists a \in \Sigma^*$和两个状态Ⓒ和Ⓓ，使得 $A \xrightarrow{a} C$，$B \xrightarrow{a} D$，并且，Ⓒ和Ⓓ有一个并仅有一个是接受状态.

性质

接受状态和非接受状态是可区别的，因为：若Ⓐ是非接受状态，Ⓑ是接受状态，则：

$$A \xrightarrow{\varepsilon} A,\ B \xrightarrow{\varepsilon} B.$$

性质

如果Ⓐ和Ⓑ是可区别的，并且$C \xrightarrow{a} A$，$D \xrightarrow{a} B$，则，Ⓒ和Ⓓ是可区别的. 因为：

$$C \xrightarrow{a} A \xrightarrow{a} G$$
$$D \xrightarrow{a} B \xrightarrow{a} H$$

可区别的状态

定义

设Ⓐ和Ⓑ是 DFA M 的两个状态，称Ⓐ和Ⓑ是可区别的 iff $\exists a \in \Sigma^*$和两个状态Ⓒ和Ⓓ，使得 $A \xrightarrow{a} C$，$B \xrightarrow{a} D$，并且，Ⓒ和Ⓓ有一个并仅有一个是接受状态.

性质

接受状态和非接受状态是可区别的，因为：若Ⓐ是非接受状态，Ⓑ是接受状态，则：

$$A \xrightarrow{\varepsilon} A,\ B \xrightarrow{\varepsilon} B.$$

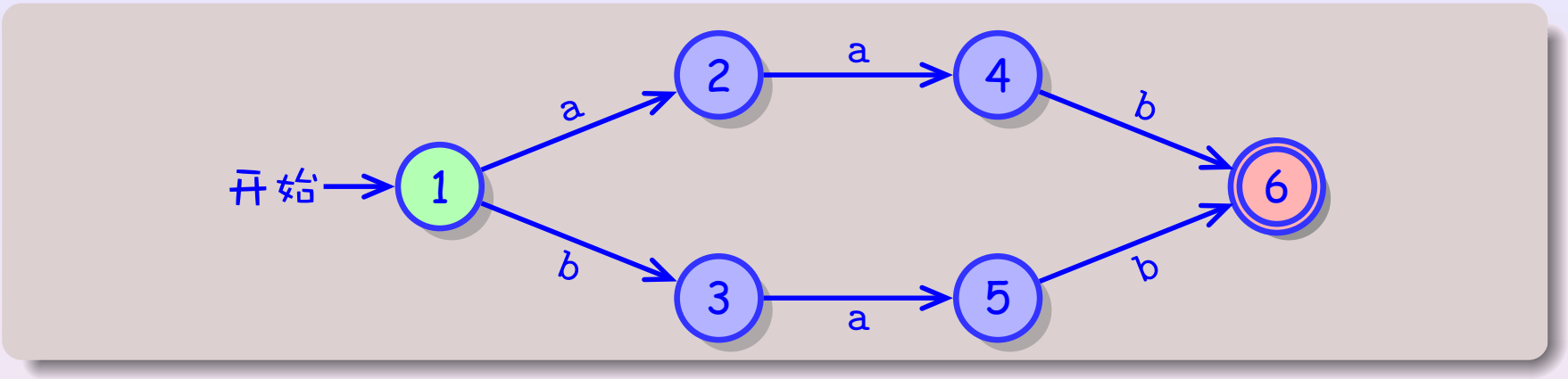
性质

如果Ⓐ和Ⓑ是可区别的，并且$C \xrightarrow{a} A$，$D \xrightarrow{a} B$，则，Ⓒ和Ⓓ是可区别的. 因为：

$$C \xrightarrow{a} A \xrightarrow{a} G$$
$$D \xrightarrow{a} B \xrightarrow{a} H$$

©hfwang

Example

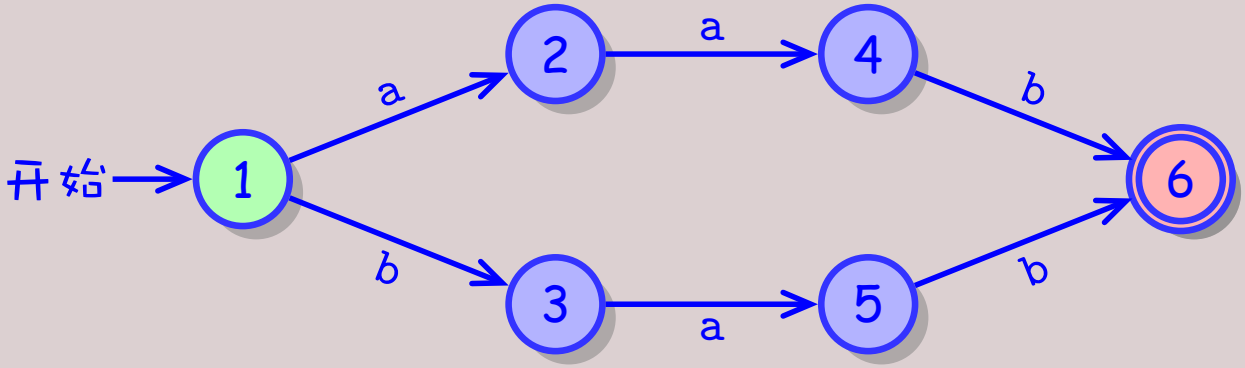


- 增加一个出错状态 (trap state), 使状态转换图为完全图.
- (6)与(2,3,4,5)中每个状态是可区别的.
- (2)与(1和4)是可区别的, (2)与(1和3)是可区别的.
- (2)与(3和4)与(5)是不可区别的.

命题

如果 A 和 B 是不可区别的, 则从 A 状态出发接受的串和从 B 状态出发接受的串相同, 称之为等价状态. 合并所有的价状态所得到的自动机一定是状态数最少的自动机.

Example

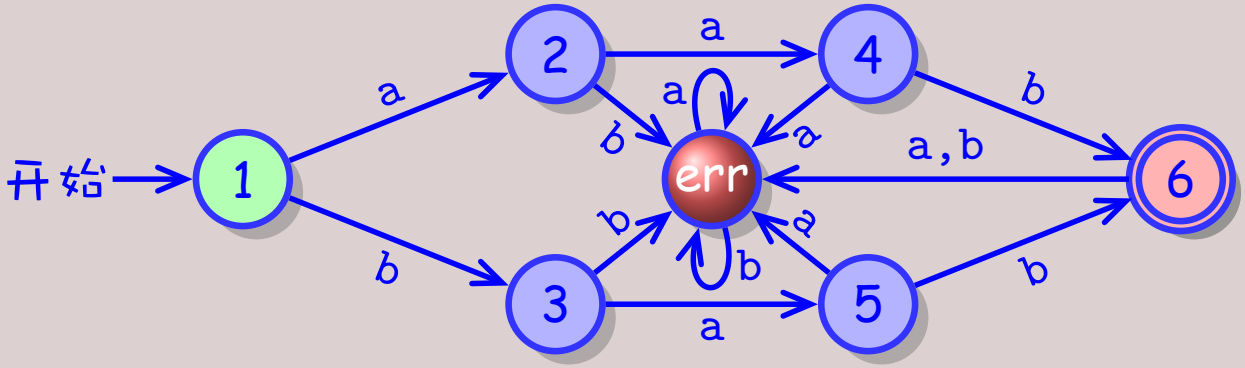


- 增加一个出错状态 (trap state), 使状态转换图为完全图.
- ⑥与①, ②, ③, ④, ⑤和err是可区别的.
- ②与①和④. ③与①和⑤是可区别的.
- ②与③; ④与⑤是不可区别的.

命题

如果 A 和 B 是不可区别的, 则从 A 状态出发接受的串和从 B 状态出发接受的串相同, 称之为等价状态. 合并所有的价状态所得到的自动机一定是状态数最少的自动机.

Example

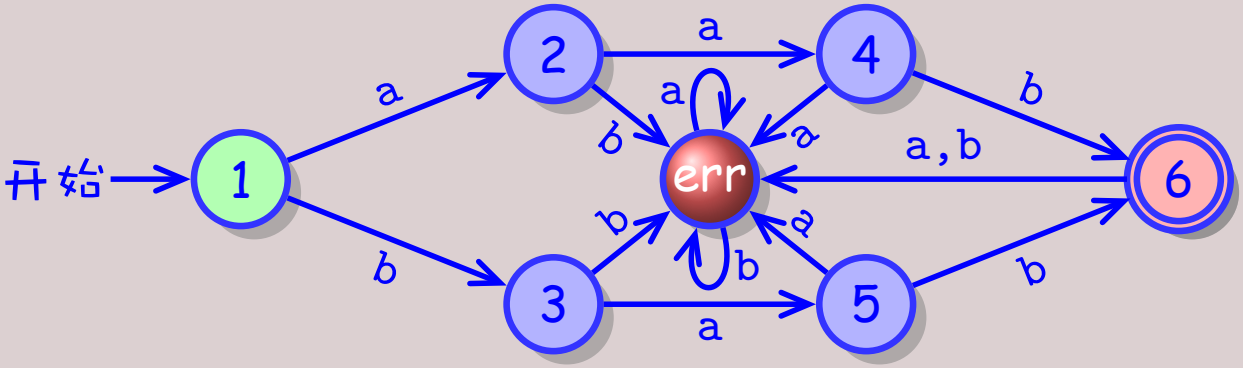


- 增加一个出错状态 (trap state), 使状态转换图为完全图.
- ⑥与①, ②, ③, ④, ⑤和err是可区别的.
- ②与①和④. ③与①和⑤是可区别的.
- ②与③; ④与⑤是不可区别的.

命题

如果 A 和 B 是不可区别的, 则从 A 状态出发接受的串和从 B 状态出发接受的串相同, 称之为等价状态. 合并所有的价状态所得到的自动机一定是状态数最少的自动机.

Example

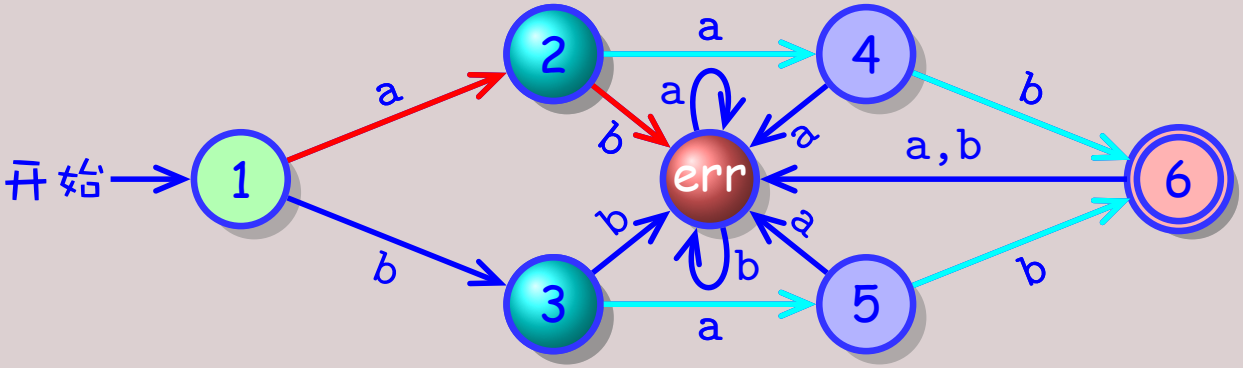


- 增加一个出错状态 (trap state), 使状态转换图为完全图.
- ⑥与①，②，③，④，⑤和err是可区别的.
- ②与①和④. ③与①和⑤是可区别的.
- ②与③; ④与⑤是不可区别的.

命题

如果 A 和 B 是不可区别的, 则从 A 状态出发接受的串和从 B 状态出发接受的串相同，称之为等价状态. 合并所有的价状态所得到的自动机一定是状态数最少的自动机.

Example

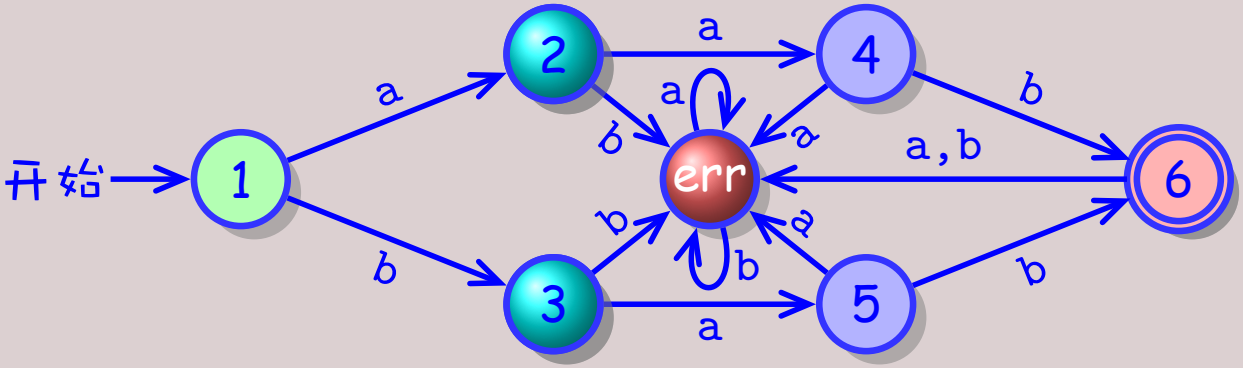


- 增加一个出错状态 (trap state), 使状态转换图为完全图.
- ⑥与①，②，③，④，⑤和err是可区别的.
- ②与①和④. ③与①和⑤是可区别的.
- ②与③; ④与⑤是不可区别的.

命题

如果 A 和 B 是不可区别的, 则从 A 状态出发接受的串和从 B 状态出发接受的串相同，称之为等价状态. 合并所有的价状态所得到的自动机一定是状态数最少的自动机.

Example

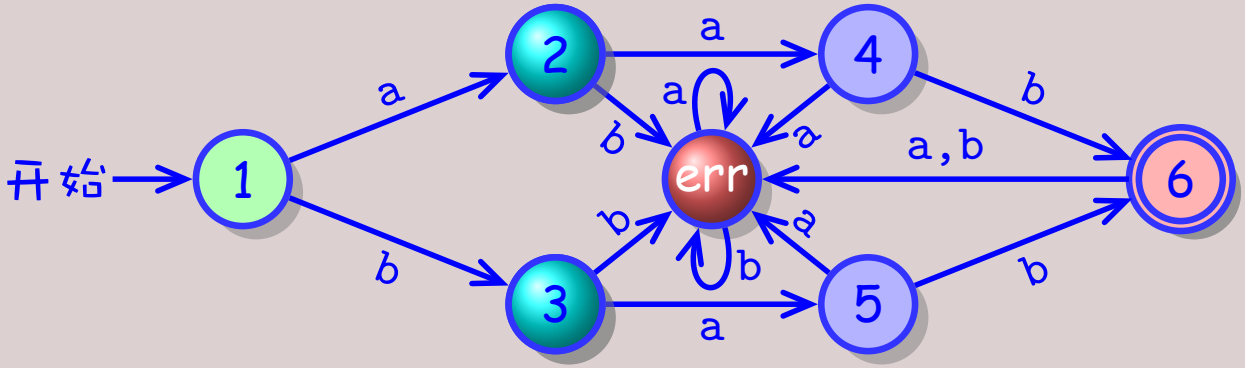


- 增加一个出错状态 (trap state), 使状态转换图为完全图.
- 6与1, 2, 3, 4, 5和err是可区别的.
- 2与1和4. 3与1和5是可区别的.
- 2与3; 4与5是不可区别的.

命题

如果 A 和 B 是不可区别的, 则从 A 状态出发接受的串和从 B 状态出发接受的串相同, 称之为等价状态. 合并所有的价状态所得到的自动机一定是状态数最少的自动机.

Example

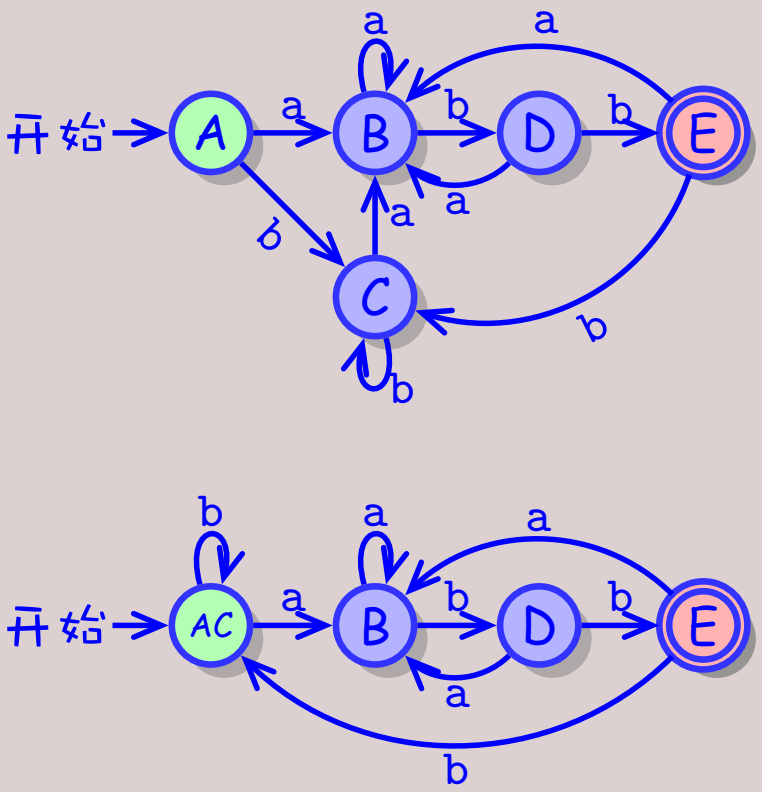


- 增加一个出错状态 (trap state), 使状态转换图为完全图.
- ⑥与①, ②, ③, ④, ⑤和err是可区别的.
- ②与①和④. ③与①和⑤是可区别的.
- ②与③; ④与⑤是不可区别的.

命题

如果 A 和 B 是不可区别的, 则从 A 状态出发接受的串和从 B 状态出发接受的串相同, 称之为等价状态. 合并所有的价状态所得到的自动机一定是状态数最少的自动机.

Partition 算法



算法

增加一个Trap state，使DFA状态图是一个完全图.

1、首先分解成两个可区分的集合， S - F, F;

2、if ∃a，使得 s1, s2∈Si，而Trans(s1,a)和Trans(s2,a)在不同的分区，则细分Si:

Ex:

	{A, B, C, D}				{E}
a	↓	↓	↓	↓	
	B	B	B	B	
b	↓	↓	↓	↓	
	C	D	C	E	

⇒

	{A, B, C}			{D}	{E}
b	↓	↓	↓		
	C	D	C		

⇒

	{A, C}	{B}		{D}	{E}
b	↓	↓			
	C	C			

3、重复2直到不能再细分为止，合并状态后的自动机是最小化自动机.

4、该自动机对状态转换函数是相容的，即分区的每个状态在接受接受一个字符一定跳转到同一个分区，否则还可以进一步细分.

©hfwang

表达能力等价

公式

$$\mathcal{L}(RE) = \mathcal{L}(NFA) = \mathcal{L}(DFA)$$

定义

由正则表达式生成的语言称为正则语言.

定理

L 是正则语言，则 $\Sigma^* - L$ 也是正则语言.

Example

表达能力等价

公式

$$\mathcal{L}(RE) = \mathcal{L}(NFA) = \mathcal{L}(DFA)$$

定义

由正则表达式生成的语言称为正则语言.

定理

L 是正则语言，则 $\Sigma^* - L$ 也是正则语言.

Example

表达能力等价

公式

$$\mathcal{L}(RE) = \mathcal{L}(NFA) = \mathcal{L}(DFA)$$

定义

由正则表达式生成的语言称为正则语言.

定理

L 是正则语言，则 $\Sigma^* - L$ 也是正则语言.

Example

表达能力等价

公式

$$\mathcal{L}(RE) = \mathcal{L}(NFA) = \mathcal{L}(DFA)$$

定义

由正则表达式生成的语言称为正则语言.

定理

L 是正则语言，则 $\Sigma^* - L$ 也是正则语言.

Example

$L = \{s \mid s \in \{a,b,c\}^*, s$中一定没有bc子串$\}$

$\Sigma^* - L = \{s \mid s \in \{a,b,c\}^*, s$中可以有bc子串$\}$

表达能力等价

公式

$$\mathcal{L}(RE) = \mathcal{L}(NFA) = \mathcal{L}(DFA)$$

定义

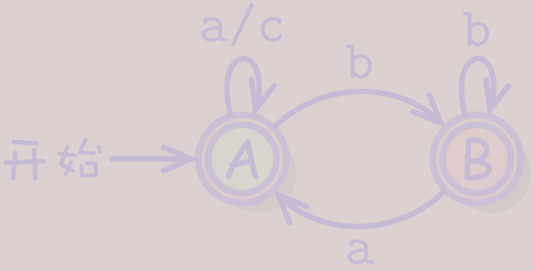
由正则表达式生成的语言称为正则语言.

定理

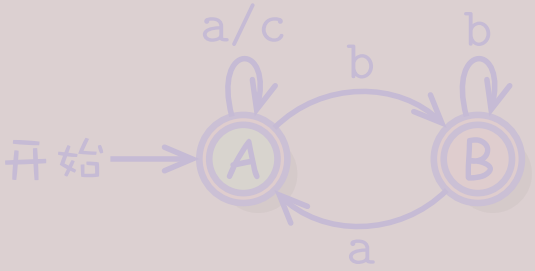
L 是正则语言，则 $\Sigma^* - L$ 也是正则语言.

Example

$L = \{s \mid s \in \{a,b,c\}^*, s$中一定没有 bc子串$\}$



$\Sigma^* - L = \{s \mid s \in \{a,b,c\}^*, s$中一定有 bc子串$\}$



表达能力等价

公式

$$\mathcal{L}(RE) = \mathcal{L}(NFA) = \mathcal{L}(DFA)$$

定义

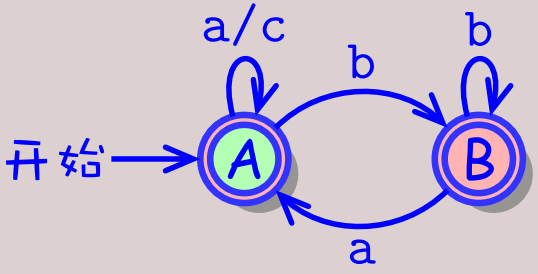
由正则表达式生成的语言称为正则语言.

定理

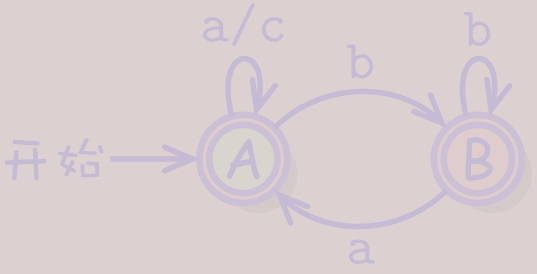
L 是正则语言，则 $\Sigma^* - L$ 也是正则语言.

Example

$L = \{s \mid s \in \{a,b,c\}^*, s$中一定没有 bc子串$\}$



$\Sigma^* - L = \{s \mid s \in \{a,b,c\}^*, s$中一定有 bc子串$\}$



表达能力等价

公式

$$\mathcal{L}(RE) = \mathcal{L}(NFA) = \mathcal{L}(DFA)$$

定义

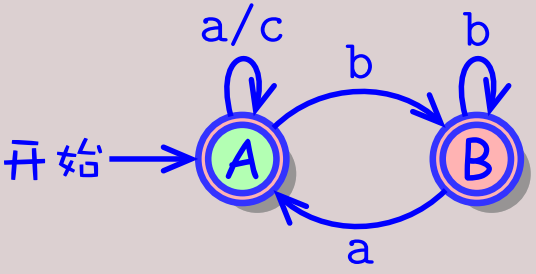
由正则表达式生成的语言称为正则语言.

定理

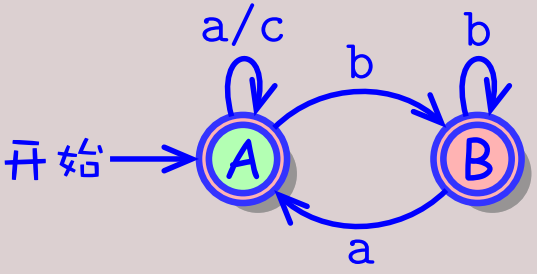
L 是正则语言，则 $\Sigma^* - L$ 也是正则语言.

Example

$L = \{s \mid s \in \{a,b,c\}^*, s$中一定没有 bc子串$\}$



$\Sigma^* - L = \{s \mid s \in \{a,b,c\}^*, s$中一定有 bc子串$\}$



表达能力等价

公式

$$\mathcal{L}(RE) = \mathcal{L}(NFA) = \mathcal{L}(DFA)$$

定义

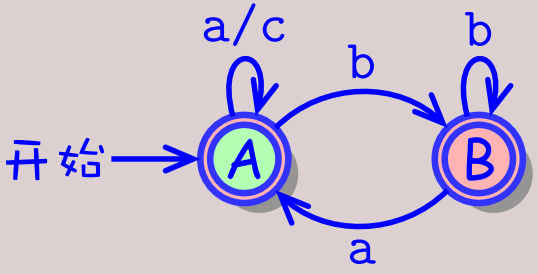
由正则表达式生成的语言称为正则语言.

定理

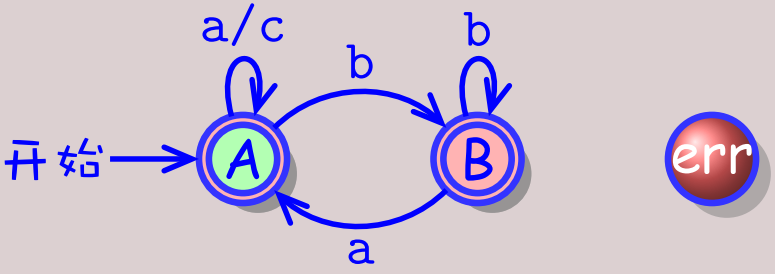
L 是正则语言，则 $\Sigma^* - L$ 也是正则语言.

Example

$L = \{s \mid s \in \{a,b,c\}^*, s$中一定没有 bc子串$\}$



$\Sigma^* - L = \{s \mid s \in \{a,b,c\}^*, s$中一定有 bc子串$\}$



表达能力等价

公式

$$\mathcal{L}(RE) = \mathcal{L}(NFA) = \mathcal{L}(DFA)$$

定义

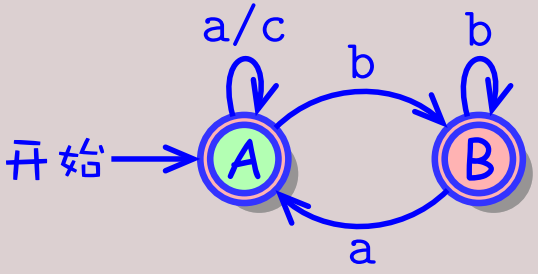
由正则表达式生成的语言称为正则语言.

定理

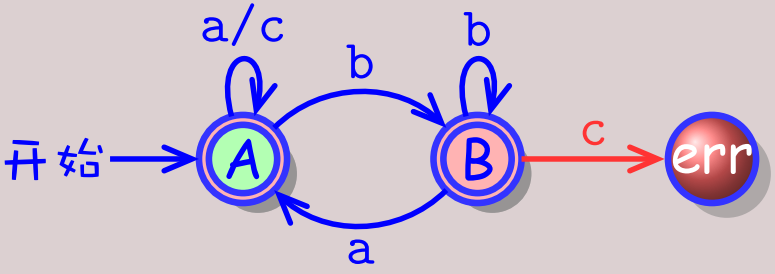
L 是正则语言，则 $\Sigma^* - L$ 也是正则语言.

Example

$L = \{s \mid s \in \{a,b,c\}^*, s$中一定没有 bc子串$\}$



$\Sigma^* - L = \{s \mid s \in \{a,b,c\}^*, s$中一定有 bc子串$\}$



表达能力等价

公式

$$\mathcal{L}(RE) = \mathcal{L}(NFA) = \mathcal{L}(DFA)$$

定义

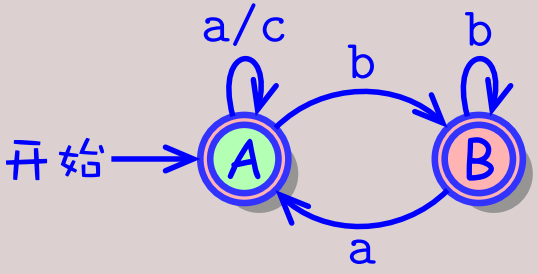
由正则表达式生成的语言称为正则语言.

定理

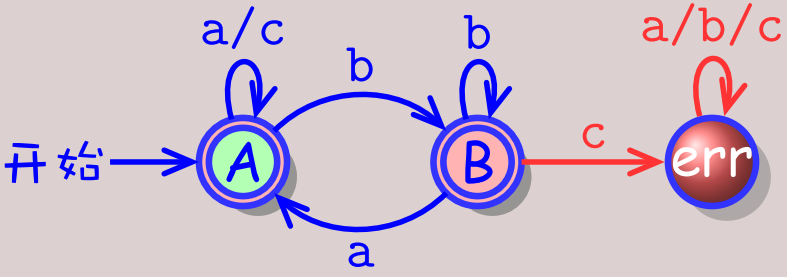
L 是正则语言，则 $\Sigma^* - L$ 也是正则语言.

Example

$L = \{s \mid s \in \{a,b,c\}^*, s$中一定没有 bc子串$\}$



$\Sigma^* - L = \{s \mid s \in \{a,b,c\}^*, s$中一定有 bc子串$\}$



表达能力等价

公式

$$\mathcal{L}(RE) = \mathcal{L}(NFA) = \mathcal{L}(DFA)$$

定义

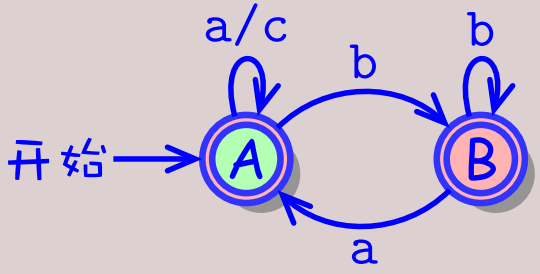
由正则表达式生成的语言称为正则语言.

定理

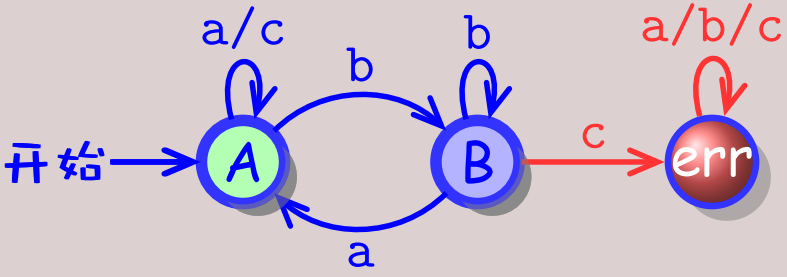
L 是正则语言，则 $\Sigma^* - L$ 也是正则语言.

Example

$L = \{s \mid s \in \{a,b,c\}^*, s$中一定没有 bc子串$\}$



$\Sigma^* - L = \{s \mid s \in \{a,b,c\}^*, s$中一定有 bc子串$\}$



表达能力等价

公式

$$\mathcal{L}(RE) = \mathcal{L}(NFA) = \mathcal{L}(DFA)$$

定义

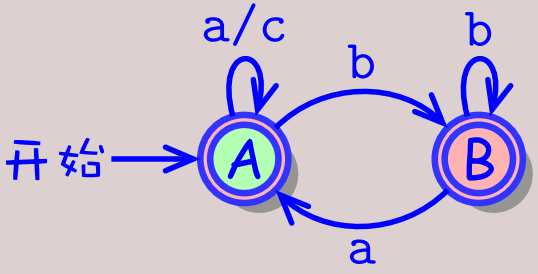
由正则表达式生成的语言称为正则语言.

定理

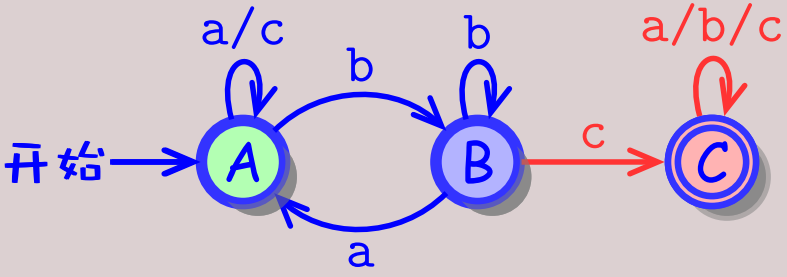
L 是正则语言，则 $\Sigma^* - L$ 也是正则语言.

Example

$L = \{s \mid s \in \{a,b,c\}^*, s$中一定没有 bc子串$\}$



$\Sigma^* - L = \{s \mid s \in \{a,b,c\}^*, s$中一定有 bc子串$\}$



相关性质

可判定性问题

定理：存在一个算法对任意的两个正则表达式 r 和 s，判断其是否等价. 因为：$r = s$ iff $MDFA(r) \cong MDFA(s)$.

语言运算

定理：设 L_1 和 L_2 是正则语言，则 $L_1 \cup L_2$，$L_1 \cap L_2$，$L_1 - L_2$，$L_1 L_2$ 和 L_1^* 也是正则语言

©hfwang

相关性质

可判定性问题

定理: 存在一个算法对任意的两个正则表达式 r 和 s, 判断其是否等价. 因为: $r = s$ iff $MDFA(r) \cong MDFA(s)$.

语言运算

定理: 设 L_1 和 L_2 是正则语言, 则 $L_1 \cdot L_2$, $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$ 和 L_1^* 也是正则语言.

相关性质

可判定性问题

定理： 存在一个算法对任意的两个正则表达式 r 和 s，判断其是否等价. 因为：$r = s$ iff $MDFA(r) \cong MDFA(s)$.

语言运算

定理： 设 L_1 和 L_2 是正则语言，则 $L_1 \cdot L_2$, $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$ 和 L_1^* 也是正则语言.

©hfwang

相关性质

可判定性问题

定理： 存在一个算法对任意的两个正则表达式 r 和 s，判断其是否等价. 因为：$r = s$ iff $MDFA(r) \cong MDFA(s)$.

语言运算

定理： 设 L_1 和 L_2 是正则语言，则 $L_1 \cdot L_2$, $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$ 和 L_1^* 也是正则语言.

©hfwang

本章小节

1. 基本概念
   - 定义
   - 分析方法
   - 正则表达式
2. 确定的有限自动机
   - 基本概念
   - DFA 的设计
   - DFA 算法
3. 不确定的有限自动机
   - 基本概念
   - 正则表达式到 NFA 的转换
   - NFA 到 DFA 的转换
   - DFA 状态最小化
4. 相关问题

©hfwang