Mathematical Foundation of Computer Sciences I

Regular Languages and Finite Automata

Guoqiang Li

School of Software, Shanghai Jiao Tong University

Instructor and Textbook

Instructors

Guoqiang Li, 1-8, Automata Theory

Xiaodong Gu, 9-16, Optimization Theory

Xubo Yang, 17-24, Scientific Computing

Instructor

- Guoqiang LI

Instructor

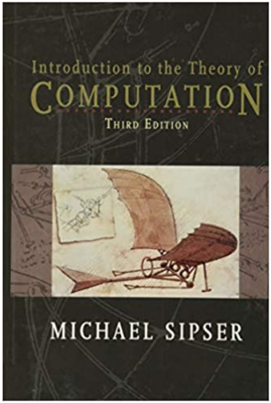
- Guoqiang LI
  - Homepage: https://basics.sjtu.edu.cn/%7Eliguoqiang
  - Course page: https://basics.sjtu.edu.cn/%7Eliguoqiang/teaching/SE2324/
  - Email: li.g@outlook.com
  - Office: Rm. 1212, Building of Software
  - Phone: 3420-4167

Instructor

- Guoqiang LI
  - Homepage: https://basics.sjtu.edu.cn/%7Eliguoqiang
  - Course page: https://basics.sjtu.edu.cn/%7Eliguoqiang/teaching/SE2324/
  - Email: li.g@outlook.com
  - Office: Rm. 1212, Building of Software
  - Phone: 3420-4167
- TA:
  - Jingyang LI: 94598772 (AT) qq (DOT) com

Textbook

[Sip12] *Introduction to the Theory of Computation*, Michael Sipser, 2012



Scoring Policy

30% Homework.

25% Mid-term Exam.

20% Report.

25% Final Exam.

Scoring Policy

30% Homework.

- Each part 10 pt.
- Automata part: 3 homework.

25% Mid-term Exam.

- This is for Automata part.

20% Report.

- This is for Optimization part.

25% Final Exam.

- This is for Scientific computing part.

Regular Languages and DFA

Deterministic Finite Automata

Definition (DFA)

A deterministic finite automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

这个转化方程实质上就是一个函数，注意在画DFA是不要漏情况(trap状态的使用)

1. Q is a finite set called the states,
2. Σ is a finite set called the alphabet,
3. $\delta : Q \times \Sigma \to Q$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

F状态可以有多个，构成一个集合

Formal Definition of Computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \dots w_n$ be a string with $w_i \in \Sigma$ for all $i \in [n]$. Then M accepts w if a sequence of states $r_0, r_1, \dots, r_n$ in Q exists with:

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, \dots, n-1$, and
3. $r_n \in F$.

Formal Definition of Computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \ldots w_n$ be a string with $w_i \in \Sigma$ for all $i \in [n]$. Then M accepts w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with:

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, \ldots, n-1$, and
3. $r_n \in F$.

We say that M recognizes A if

$$A = \{w \mid M \text{ accepts } w\}$$

A实质上就是一种 语言，即能够被一个DFA接受的string的集合

Regular Languages

Definition (Regular languages)

A language is called regular if some finite automaton recognizes it.

没有强调DFA

Examples of Regular Languages

$\{(ab)^n \mid \forall n \geq 0\}$

Examples of Regular Languages

$\{(ab)^n \mid \forall n \geq 0\}$

$\{a^n b^n \mid \forall n \geq 0\}$

Examples of Regular Languages

$\{(ab)^n \mid \forall n \geq 0\}$

可能为空串，且当n无穷大时不是regular

$\{a^n b^n \mid \forall n \geq 0\}$

可能为空串，且当n无穷大时不是regular

$\{ab, a^2 b^2, \ldots a^n b^n\}$

不可能为空串，(此题中的n为有限的正整数)

The Regular Operators

Definition

Let A and B be languages. We define the regular operations union, concatenation, and star as follows:

The Regular Operators

Definition

Let A and B be languages. We define the regular operations union, concatenation, and star as follows:

- Union: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.

The Regular Operators

Definition

Let A and B be languages. We define the regular operations union, concatenation, and star as follows:

- Union: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.
- Concatenation: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$.

Definition

Let A and B be languages. We define the regular operations union, concatenation, and star as follows:

- Union: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.
- Concatenation: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$.
- Kleene star: $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

Closure under Union

Theorem

The class of regular languages is closed under the union operation.

In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

closed:运算封闭，指的是满足某个性质的参数再参加某个运算之后的结果仍有该性质，则称这个性质对该运算方式封闭

Pre-Proof

For $i \in [2]$ let $M_i = (Q_i, \Sigma_i, \delta_i, q_{0_i}, F_i)$ recognize A_i. We can assume without loss of generality $\Sigma_1 = \Sigma_2$:

Pre-Proof

For $i \in [2]$ let $M_i = (Q_i, \Sigma_i, \delta_i, q_{0_i}, F_i)$ recognize A_i. We can assume without loss of generality $\Sigma_1 = \Sigma_2$:

- Let $a \in \Sigma_2 - \Sigma_1$.

Pre-Proof

For $i \in [2]$ let $M_i = (Q_i, \Sigma_i, \delta_i, q_{0_i}, F_i)$ recognize A_i. We can assume without loss of generality $\Sigma_1 = \Sigma_2$:

- Let $a \in \Sigma_2 - \Sigma_1$.
- We add $\delta_1(r, a) = r_{trap}$, where r_{trap} is a new state with $\delta_1(r_{trap}, w) = r_{trap}$ for every w.

Proof

We construct $M = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$:

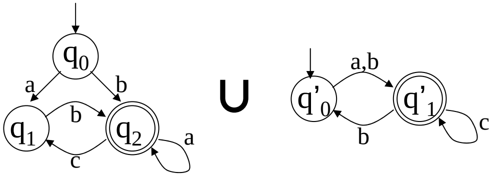
1. $Q = Q_1 \times Q_2 = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$.
2. $\Sigma = \Sigma_1 = \Sigma_2$.
3. For each $(r_1, r_2) \in Q$ and $a \in \Sigma$ we let

   $$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$$

4. $q_0 = (q_1, q_2)$.
5. $F = (F_1 \times Q_2) \cup (Q_1 \times F_2) = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$.

该转化方程实际上就是在模拟A1 UA2一起计算和A1和A2分开计算然后去并集的过程，但是该算法有较多的空间浪费，因为有的状态并集组合根本取不到

A Sample



Closure under Concatenation

Theorem

The class of regular languages is closed under the concatenation operation.

In other words, if A_1 and A_2 are regular languages, so is $A_1 \circ A_2$.

Closure under Concatenation

Theorem

The class of regular languages is closed under the concatenation operation.

In other words, if A_1 and A_2 are regular languages, so is $A_1 \circ A_2$.

We prove the above theorem by nondeterministic finite automata.

Nondeterministic Finite Automata

Nondeterminism

Definition (NFA)

A nondeterministic finite automaton (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the states,
2. Σ is a finite set called the alphabet,
3. $\delta : Q \times \Sigma_\varepsilon \to \mathscr{P}(Q)$ is the transition function, where $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

NFA相对DFA的特点：1.其转化方程最终的转化结果为初始状态的一个子集，即可能一个输入字符存在不同的输出分支；2.NFA的输入字符集合中允许有空串。

Formal Definition of Computation

Let $N = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic finite automaton and let $w = w_1 w_2 \ldots w_m$ be a string with $w_i \in \Sigma_\varepsilon$ for all $i \in [m]$. Then N accepts w if a sequence of states $r_0, r_1, \ldots, r_m$ in Q exists with:

1. $r_0 = q_0$,
2. $r_{i+1} \in \delta(r_i, w_{i+1})$ for $i = 0, \ldots, m-1$, and
3. $r_m \in F$.

Formal Definition of Computation

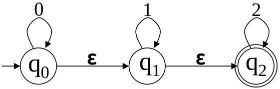
Let $N = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic finite automaton and let $w = w_1 w_2 \dots w_m$ be a string with $w_i \in \Sigma_\varepsilon$ for all $i \in [m]$. Then N accepts w if a sequence of states $r_0, r_1, \dots, r_m$ in Q exists with:

1. $r_0 = q_0$,
2. $r_{i+1} \in \delta(r_i, w_{i+1})$ for $i = 0, \dots, m-1$, and
3. $r_m \in F$.

We say that N recognizes A if

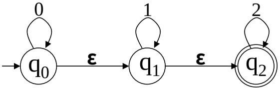
$$A = \{w \mid M \text{ accepts } w\}$$

Examples of NFA



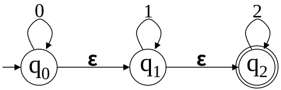
空串实际上相当于什么也没做，故空串这个字符表示可以随意添加在输入字符串的任意位置

Examples of NFA



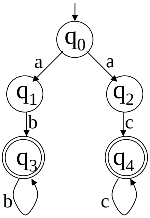
Accepts $\{0^*1^*2^*\}$

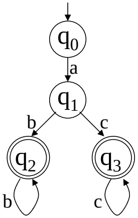
Examples of NFA

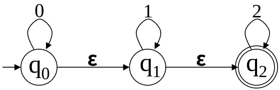


Accepts $\{0^*1^*2^*\}$

如此图所示，q0出来的a的线条有两条，这实际上就是NFA在进行尝试以及猜测的过程。

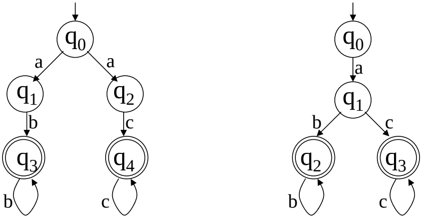






Accepts $\{0^*1^*2^*\}$

*号表示从0开始的任意长度；+号表示从1开始的任意长度。



Accepts $\{ab^+, ac^+\}$

Equivalence of NFAs and DFAs

Theorem

Every NFA has an equivalent DFA, i.e., they recognize the same language.

Proof

Proof.

Proof

Proof.

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A. We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing the same A.

Proof

Proof.

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A. We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing the same A.

First assume N has no "ε" arrows.

Proof

Proof.

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A. We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing the same A.

First assume N has no "ε" arrows.

1. $Q' = \mathscr{P}(Q)$.

Proof

Proof.

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A. We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing the same A.

First assume N has no "ε" arrows.

1. $Q' = \mathscr{P}(Q)$.
2. Let $R \in Q'$ and $a \in \Sigma$. Then we define

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

构造NFA对应的DFA

Proof

Proof.

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A. We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing the same A.

First assume N has no "ε" arrows.

1. $Q' = \mathscr{P}(Q)$.
2. Let $R \in Q'$ and $a \in \Sigma$. Then we define

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

3. $q_0' = \{q_0\}$.

Proof

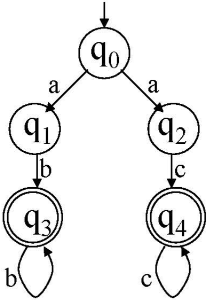
Proof.

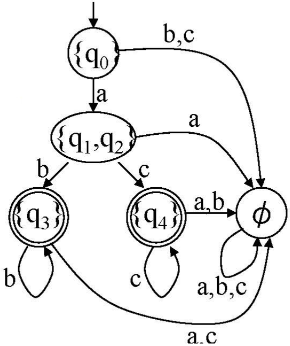
Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A. We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing the same A.

First assume N has no "ε" arrows.

1. $Q' = \mathscr{P}(Q)$.
2. Let $R \in Q'$ and $a \in \Sigma$. Then we define
$$\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$$
3. $q_0' = \{q_0\}$.
4. $F' = \{R \in Q' \mid R \cap F \neq \emptyset\}$.

Determinization





Proof (cont'd)

Proof.

Now we allow "ε" arrows.

For every $R \in Q'$, i.e., $R \subseteq Q$, let

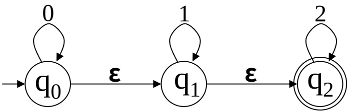
$$E(R) = \{q \in Q \mid \quad q \text{ can be reached from } R \text{ by traveling along 0 and more } \varepsilon \text{ arrows } \}$$

1. $Q' = \mathscr{P}(Q)$.
2. Let $R \in Q'$ and $a \in \Sigma$. Then we define

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

3. $q_0' = E(\{q_0\})$.
4. $F' = \{R \in Q' \mid R \cap F \neq \emptyset\}$.

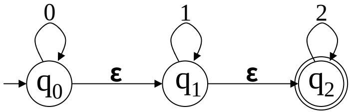
Example of ε-Transition Removal



Put a new transition $\xrightarrow{a}$ where $\xrightarrow{\varepsilon^* a \varepsilon^*}$

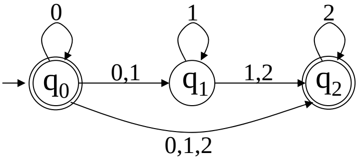
If $q_0 \xrightarrow{\varepsilon^*} q_f$ for $q_f \in F$, add q_0 to F

Example of ε-Transition Removal



Put a new transition $\xrightarrow{a}$ where $\xrightarrow{\varepsilon^* a \varepsilon^*}$

If $q_0 \xrightarrow{\varepsilon^*} q_f$ for $q_f \in F$, add q_0 to F



Corollary (推论)

Corollary

A language is regular if and on if some nondeterministic finite automaton recognizes it.

Second Proof of the Closure under Union

以下几个证明均是基于NFA理论

For $i \in [2]$ let $N_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$ recognize A_i. We construct an $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$:

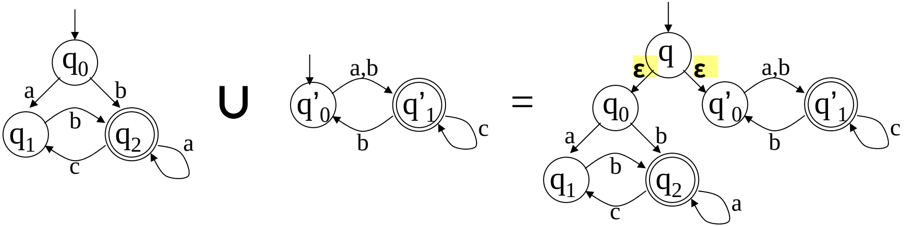
注意这里的写法。NFA由于其每次转化的结果的不确定性，故可以将其转化的终点写成多个状态构成的集合的形式，而对于DFA，其每次转化的结果是固定的，故只能将其转化的终点写成笛卡尔积的形式。

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$.
2. q_0 is the start state. q0是额外加入的一个新的起始状态
3. $F = F_1 \cup F_2$.
4. For any $q \in Q$ and any $a \in \Sigma_\varepsilon$

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \\ \delta_2(q, a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$

注意对于特殊情况的讨论，分类要分准确，不能有二义性

Union



Closure under Concatenation

Theorem

The class of regular languages is closed under the concatenation operation.

Proof

For $i \in [2]$ let $N_i = (Q_i, \Sigma_i, \delta_i, q_i, F_i)$ recognize A_i. We construct an $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$:

1. $Q = Q_1 \cup Q_2$.
2. The start state q_1 is the same as the start state of N_1.
3. The accept states F_2 are the same as the accept states of N_2.
4. For any $q \in Q$ and any $a \in \Sigma_\varepsilon$

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 - F_1 \\ \delta_1(q, a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q, a) \cup \{q_2\} & q \in F_1 \text{ and } a = \varepsilon \\ \delta_2(q, a) & q \in Q_2 \end{cases}$$

Closure under Kleene Star

Theorem

The class of regular languages is closed under the star operation.

Proof

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1. We construct an $N = (Q, \Sigma, \delta, q_0, F)$ to recognize A_1^*:

1. $Q = \{q_0\} \cup Q_1$.
2. The start state q_0 is the new start state.
3. $F = \{q_0\} \cup F_1$. q0处理的是空串的情况
4. For any $q \in Q$ and any $a \in \Sigma_\varepsilon$

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 - F_1 \\ \delta_1(q, a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q, a) \cup \{q_1\} & q \in F_1 \text{ and } a = \varepsilon \\ \{q_1\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$

第三行那里应该为q0

Regular Expression

Definition

We say that R is a regular expression if R is

1. a for some $a \in \Sigma$,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions,
6. R_1^*, where R_1 is a regular expression.

Regular Expression

Definition

We say that R is a regular expression if R is

1. a for some $a \in \Sigma$,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions,
6. R_1^*, where R_1 is a regular expression.

We often write $R_1 R_2$ instead of $(R_1 \circ R_2)$ if no confusion arises.

Language Defined by Regular Expressions

regular expression R	language $L(R)$
a	$\{a\}$
ε	$\{\varepsilon\}$
$\emptyset$	$\emptyset$
$R_1 \cup R_2$	$L(R_1) \cup L(R_2)$
$R_1 \circ R_2$	$L(R_1) \circ L(R_2)$
R_1^*	$L(R_1)^*$

Equivalence with Finite Automata

Theorem

A language is regular if and only if some regular expression describes it.

The Languages Defined by Regular Expressions Are Regular

1. $R = a$: Let $N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$, where $\delta(q_1, a) = \{q_2\}$ and $\delta(r, b) = \emptyset$, for all $r \neq q_1$ or $b \neq a$.

The Languages Defined by Regular Expressions Are Regular

<==

1. $R = a$: Let $N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$, where $\delta(q_1, a) = \{q_2\}$ and $\delta(r, b) = \emptyset$, for all $r \neq q_1$ or $b \neq a$.
2. $R = \varepsilon$: Let $N = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$, where $\delta(r, b) = \emptyset$, for all r and b.
3. $R = \emptyset$: Let $N = (\{q_1\}, \Sigma, \delta, q_1, \emptyset)$, where $\delta(r, b) = \emptyset$, for all r and b.
4. $R = R_1 \cup R_2$: $L(R) = L(R_1) \cup L(R_2)$.
5. $R = R_1 \circ R_2$: $L(R) = L(R_1) \circ L(R_2)$.
6. $R = R_1^*$: $L(R) = L(R_1)^*$.

要理解空集出现的情况：
1.没有F状态集合
2.有F状态集合但是无法达到。

Regular languages can be defined by regular expressions

==>

We need generalized nondeterministic finite automata (GNFA)-nondeterministic finite automata where in the transition arrows may have any regular expressions as labels.

1. The start state has transition arrows going to every other state but no arrows coming in from any other state.
2. There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
3. Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.

要注意这里对于GNFA的定义：start：只出不进，并且指向所有状态；final：只进不出，并且所有状态都要指向这里；并且终止状态只有一个；其余状态和之前NFA同

Generalized nondeterministic finite automata

Definition

A GNFA is a 5-tuple $(Q, \Sigma, \delta, q_{start}, q_{accept})$, where

- Q is a finite set of states,
- Σ is a finite alphabet,
- $\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \to R$ is the transition function, where R is the set of regular expressions,
- q_{start} is the start state, and
- q_{accept} is the accept state.

Formal definition of computation

A GNFA accepts a string $w \in \Sigma^*$ if $w = w_1 w_2 \dots w_k$, where each $w_i \in \Sigma^*$ and a sequence of states $q_0, q_1, \dots, q_k$ exists such that

- $q_0 = q_{start}$ is the start state,
- $q_k = q_{accept}$ is the accept state, and
- for each $i \in [k]$, we have $w_i \in L(R_i)$, where $R_i = \delta(q_{i-1}, q_i)$.

Regular languages can be defined by regular expressions

Let M be the DFA for language A.

- We convert M to a GNFA G by adding a new start state and a new accept state and additional transition arrows as necessary.
    1. The start state has transition arrows going to every other state but no arrows coming in from any other state.
    2. There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
    3. Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.
- Then we use a procedure convert on G to return an equivalent regular expression.

此算法是将之前转化好的GNFA对应的regular expression提取出来

1. Let k be the number of states of G.
2. If $k = 2$, then return the regular expression R labelling the arrow from q_{start} to q_{accept}.

此方法是一种不是很严格的数学归纳法

3. If $k > 2$, we select any state $q_{rip} \in Q - \{q_{start}, q_{accept}\}$ and let $G' = (Q', \Sigma, \delta', q_{start}, q_{accept})$ be the GNFA, where

$$Q' = Q - \{q_{rip}\}$$

此处去除q_rip实际上是在进行删除中间节点的过程，最终只剩下start和final，得到regular expression

and for any $q_i \in Q' - \{q_{accept}\}$ and $q_j \in Q' - \{q_{start}\}$, let

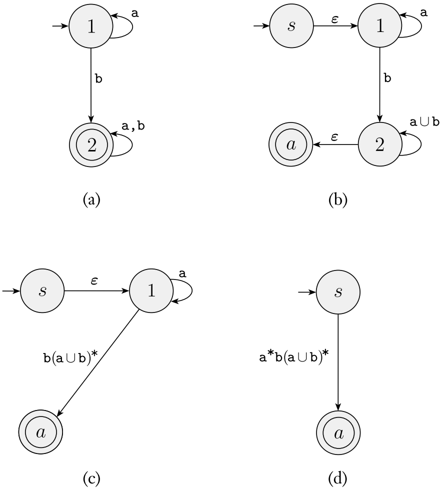
$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4)$$

for $R_1 = \delta(q_i, q_{rip})$, $R_2 = \delta(q_{rip}, q_{rip})$, $R_3 = \delta(q_{rip}, q_j)$, and $R_4 = \delta(q_i, q_j)$.

4. compute convert(G') and return this value.

模型图见regular-附页

An Example



Non-Regular Languages

Languages need counting

$$C = \{w \in \{0,1\} \mid w \text{ has an equal number of 0s and 1s}\}$$

$$D = \left\{ w \in \{0,1\} \;\middle|\; \begin{array}{l} w \text{ has an equal number of occurrences} \\ \text{of 01 and 10 as substrings} \end{array} \right\}$$

C、D均不是regular language

Quiz: D is regular.

Pumping Lemma

Lemma (Pumping Lemma)

If A is a regular language, then there is a number p (i.e., the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. *for each $i \geq 0$, we have $xy^iz \in A$,* y部分可以为空
2. $|y| > 0$, *and* 注意引理的使用条件！
3. $|xy| \leq p$.

Any string xyz in A can be pumped along y.

Proof

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA recognizing A, and $p := |Q|$.

Let $s = s_1 s_2 \ldots s_n$ be a string in A with $n \geq p$. Let $r_1, \ldots, r_{n+1}$ be the sequence of states that A enters while processing s, i.e.,

$$r_{i+1} = \delta(r_i, s_i)$$

for $i \in [n]$.

Among the first $p + 1$ states in the sequence, two must be the same, say r_j and r_ℓ with $j < \ell \leq p + 1$. We define 鸽巢原理

$$x = s_1 \ldots s_{j-1}, y = s_j \ldots s_{\ell-1}, \text{ and } z = s_\ell \ldots s_n$$

形式图见regular-附页

Example

The language $L =| \{0^n1^n \mid n \geq 0\}$ is not regular.

Example

The language $L =| \{0^n1^n \mid n \geq 0\}$ is not regular.

Proof.

Example

The language $L = | \{0^n 1^n \mid n \geq 0\}$ is not regular.

Proof. 此处的证明是基于给定的pumping length p而言的，用的是反证法

If it is regular, consider $s = 0^p 1^p$ where p is pumping length. By the Pumping lemma, $s = xyz$ with $xy^i z \in L$ for all $i \geq 0$.

Example

The language $L =| \{0^n 1^n \mid n \geq 0\}$ is not regular.

Proof.

If it is regular, consider $s = 0^p 1^p$ where p is pumping length. By the Pumping lemma, $s = xyz$ with $xy^i z \in L$ for all $i \geq 0$.

- $y \in 0^+$, then $xyyz$ has more 0s than 1s, a contradiction.

Example

The language $L = | \{0^n 1^n \mid n \geq 0\}$ is not regular.

Proof.

If it is regular, consider $s = 0^p 1^p$ where p is pumping length. By the Pumping lemma, $s = xyz$ with $xy^i z \in L$ for all $i \geq 0$.

- $y \in 0^+$, then $xyyz$ has more 0s than 1s, a contradiction.
- $y \in 1^+$, then $xyyz$ has more 1s than 0s, again a contradiction.

Example

The language $L =| \{0^n 1^n \mid n \geq 0\}$ is not regular.

Proof.

If it is regular, consider $s = 0^p 1^p$ where p is pumping length. By the Pumping lemma, $s = xyz$ with $xy^i z \in L$ for all $i \geq 0$.

- $y \in 0^+$, then $xyyz$ has more 0s than 1s, a contradiction.
- $y \in 1^+$, then $xyyz$ has more 1s than 0s, again a contradiction.
- y consists of both 0s and 1s, then $xyyz$ have 0 and 1 interleaved.

Example

The language $L = \{w \mid w \text{ has an equal number of 0s and 1s }\}$ is not regular.

Example

The language $L = \{w \mid w \text{ has an equal number of 0s and 1s }\}$ is not regular.

Proof.

Example

The language $L = \{w \mid w \text{ has an equal number of 0s and 1s } \}$ is not regular.

Proof.

Choose p be the pumping length and consider $s = 0^p 1^p$. By the Pumping Lemma, $s = xyz$ with $|xy| < p$ and $xy^i z \in L$.

此处是一种常用方法：通过已知的语言形式，
给出一个较为容易证明的特例，并证明其矛盾性
即可

Example

The language $L = \{w \mid w \text{ has an equal number of 0s and 1s }\}$ is not regular.

Proof.

Choose p be the pumping length and consider $s = 0^p 1^p$. By the Pumping Lemma, $s = xyz$ with $|xy| < p$ and $xy^i z \in L$.

for all $i \geq 0$. Thus $xy \in 0^+$ and the contradiction follows easily.

Quiz

The language $L = \{ww \mid w \in \{0,1\}^*\}$ is not regular.

Quiz

The language $L = \{ww \mid w \in \{0,1\}^*\}$ is not regular.

The language $L = \{0^m 1^n \mid m \neq n\}$ is not regular.

Other Computations

Models and Specifications

A finite automaton can be used to describe behaviours of a system or an (intra-procedure) program. Thus regarded it as a model $\mathcal{M}$.

Models and Specifications

A finite automaton can be used to describe behaviours of a system or an (intra-procedure) program. Thus regarded it as a model $\mathcal{M}$.

A finite automaton can also be used to describe regulations of a system or an (intra-procedure) program. Thus regarded it as a specification φ.

Usually, we should guarantee

$$\mathcal{M} \models \varphi$$

Models and Specifications

A finite automaton can be used to describe behaviours of a system or an (intra-procedure) program. Thus regarded it as a model $\mathcal{M}$.

A finite automaton can also be used to describe regulations of a system or an (intra-procedure) program. Thus regarded it as a specification φ.

Usually, we should guarantee

$$\mathcal{M} \models \varphi$$

In the automata terminology, we should guarantee

$$L(\mathcal{M}) \subseteq L(\varphi)$$

An Algorithmic Problem of FA

Given two automata M and N,

$$L(M) \subseteq L(N)$$

An Algorithmic Problem of FA

Given two automata M and N,

$$L(M) \subseteq L(N)$$

Two approaches:

$$L(M) \cap L(N^c) = \emptyset$$

An Algorithmic Problem of FA

Given two automata M and N,

$$L(M) \subseteq L(N)$$

Two approaches:

$$L(M) \cap L(N^c) = \emptyset$$

and,

$$L(M^c) \cup L(N) = \Sigma^*$$

Σ *指的是全集

New Operations

intersection (交集)

complement (补集)

emptiness (空集)

universality (全集)

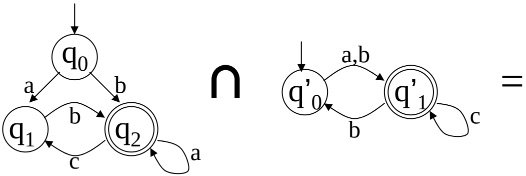
Intersection of Automata

$A = (S, \Sigma, \delta, q_0, F)$, $B = (S', \Sigma, \delta', q_0', F')$

An Automaton that accepts $L(A) \cap L(B)$

$$(S \times S', \Sigma, \delta \times \delta', (q_0, q_0'), F \times F')$$

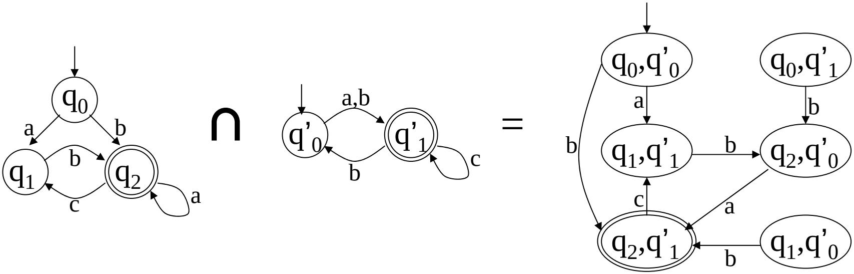
此处的S和转化方程形式与并集相似，F的形式的意思是:因为取交集，所以需要该语言能够被两个FA同时接收，即必须同时进入两个FA的终止状态。



$A = (S, \Sigma, \delta, q_0, F)$, $B = (S', \Sigma, \delta', q_0', F')$

An Automaton that accepts $L(A) \cap L(B)$

$$(S \times S', \Sigma, \delta \times \delta', (q_0, q_0'), F \times F')$$



Complement of Automata

$A = (S, \Sigma, \delta, q_0, F)$

Complement of Automata

$A = (S, \Sigma, \delta, q_0, F)$

- if A is deterministic, $A^c = (S, \Sigma, \delta, q_0, S - F)$.

Complement of Automata

$A = (S, \Sigma, \delta, q_0, F)$

- if A is deterministic, $A^c = (S, \Sigma, \delta, q_0, S - F)$.
- if A is non-deterministic,

此处的S-F需要是DFA才能成立，因为若为NFA则可能出现分支，且会有不同分支终止状态不同步的问题，导致出错；同时应该要求这个DFA必须是定义完备的，即为一个全函数。

$A = (S, \Sigma, \delta, q_0, F)$

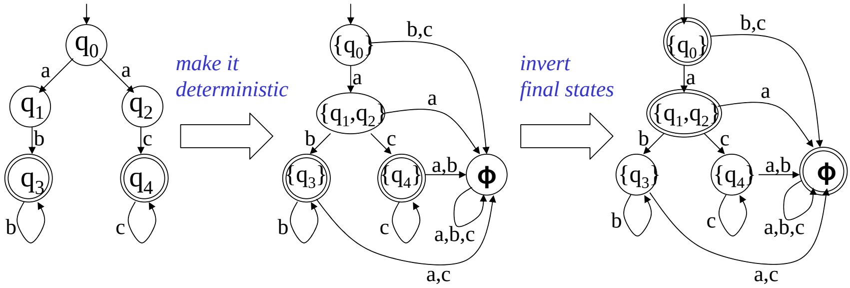
- if A is deterministic, $A^c = (S, \Sigma, \delta, q_0, S - F)$.
- if A is non-deterministic, make A deterministic first

Assume that A is without ε-transition. Then

$$(P(S), \Sigma, \{(X, a, \{y \mid x \xrightarrow{a} y \text{ for } x \in X\})\}, \{q_0\}, \{X \mid X \cap F = \emptyset\})$$

若为NFA,应该先转化为DFA，之后的过程同上DFA。

Example of Complement



Quiz

Emptiness?

判断一个语言是否是空集合，主要是看其起始状态是否有状态能够达到Final状态。
但是若将FA视为一个图，对其进行搜索时搜索路径最长为|Q|（状态总数，因为最多每个状态遍历一遍），但是根据Pumping Lemma可知，若存在一个长度大于|Q|的路径那么一定存在一个L满足pumping lemma，即为regular language，但是集合为空集内部没有任何元素，所以矛盾，故只需遍历能否达到F状态即可。（一种可能的特殊情况是，Q中本来就没有F状态）