编译原理

作业 1: 编译原理（第 1，3 章）

中国人民大学　信息学院　崔冠宇　2018202147

一、书中练习题

1. P63, T6. 令 A、B 和 C 是任意正规式，证明以下关系成立:

$$A \mid A = A$$

$$(A^*)^* = A^*$$

$$A^* = \varepsilon \mid AA^*$$

$$(AB)^*A = A(BA)^*$$

$$(A \mid B)^* = (A^*B^*)^* = (A^* \mid B^*)^*$$

$A = b \mid aA$ 当且仅当 $A = a^*b$

证明:

(1) 正规集 $L(A \mid A) = L(A) \cup L(A) = L(A)$，因此正规式 $A \mid A = A$。

(2) 正规集 $L((A^*)^*) = (L(A^*))^* = ((L(A))^*)^*$，正规集 $L(A^*) = (L(A))^*$。

对任意正规集 S，$S^* = \{w^n | w \in S, n \in \mathbb{N}\}$，$(S^*)^* = \{w^m | w \in S^*, m \in \mathbb{N}\}$。

任取 $s \in S^*$，显然有 $s \in (S^*)^*$（取 $w = s, m = 1$ 即可），所以 $S^* \subseteq (S^*)^*$;

任取 $s = s_0^m \in (S^*)^*$，因为 $s_0 \in S^*$，因此存在 $w \in S, n \in \mathbb{N}$ 使得 $s_0 = w^n$，于是 $s = s_0^m = w^{nm} \in S^*$，所以 $(S^*)^* \subseteq S^*$，即 $S^* = (S^*)^*$。

于是正规集 $L((A^*)^*) = ((L(A))^*)^* = (L(A))^* = L(A^*)$，因此正规式 $(A^*)^* = A^*$。

(3) 正规集 $L(\varepsilon | AA^*) = L(\varepsilon) \cup L(AA^*) = \{\varepsilon\} \cup (L(A) \cdot L(A^*)) = \{\varepsilon\} \cup (L(A) \cdot (L(A))^*) = \{\varepsilon\} \cup (\bigcup_{i=1}^{\infty} L^i(A))$ $= \bigcup_{i=0}^{\infty} L^i(A) = (L(A))^* = L(A^*)$，因此正规式 $A^* = \varepsilon | AA^*$。

(4) 先证明一个结论：对于任意正规集 P 和 Q 以及任意自然数 i，都有 $(PQ)^iP = P(QP)^i$。用归纳法：

① (归纳基础) 当 $i=0$ 时，原式为 $P=P$，显然成立。

② (归纳步骤) 假设当 $i=n$ 时，$(PQ)^nP=P(QP)^n$。则当 $i=n+1$ 时，$(PQ)^{n+1}P=(PQ)(PQ)^nP=(PQ)P(QP)^n=P(QP)(QP)^n=P(QP)^{n+1}$，即原式对 $i=n+1$ 也成立。

由归纳法知原式对任意自然数成立。

应用上面的结论，正规集 $L((\mathrm{AB})^*\mathrm{A})=L((\mathrm{AB})^*)\cdot L(\mathrm{A})=(L(\mathrm{AB}))^*\cdot L(\mathrm{A})=(\bigcup_{i=0}^{\infty}L^i(\mathrm{AB}))\cdot L(\mathrm{A})=\bigcup_{i=0}^{\infty}(L^i(\mathrm{AB})L(\mathrm{A}))$

$=\bigcup_{i=0}^{\infty}(L(\mathrm{A})L^i(\mathrm{BA}))=L(\mathrm{A})\cdot(\bigcup_{i=0}^{\infty}L^i(\mathrm{BA}))=L(\mathrm{A})\cdot(L(\mathrm{BA}))^*=L(\mathrm{A})\cdot L(\mathrm{BA}^*)=L(\mathrm{A}(\mathrm{BA})^*)$，因此正规式 $(\mathrm{AB})^*\mathrm{A}=\mathrm{A}(\mathrm{BA})^*$。

(5)

(a) 先证明 (A | B)*=(A*B*)*。

正规集 $L((\mathrm{A}|\mathrm{B})^*)=(L(\mathrm{A}|\mathrm{B}))^*=(L(\mathrm{A})\cup L(\mathrm{B}))^*=\{w_1w_2\cdots w_n|w_i\in L(A)\cup L(B),n\geq 0\}$,

正规集 $L((\mathrm{A}^*\mathrm{B}^*)^*)=(L(\mathrm{A}^*\mathrm{B}^*))^*=(L(\mathrm{A}^*)L(\mathrm{B}^*))^*=((L(\mathrm{A}))^*(L(\mathrm{B}))^*)^*$

$=\{w_1w_2\cdots w_n|w_i\in(L(\mathrm{A}))^*(L(\mathrm{B}))^*\}$。

对任意正规集合 P、Q，任取 $w=w_1w_2\cdots w_n\in(P\cup Q)^*$。将 w 中分成若干组，使得每一组均来自 P 或均来自 Q，而且要求每组尽可能长。即 $w=w_1'w_2'\cdots w_{n'}'$，其中 $w_1'=w_1w_2\cdots w_{k_1}$（$w_1'\in P^*$ 或 $w_1'\in Q^*$），$w_i'=w_{k_{i-1}+1}w_{k_{i-1}+2}\cdots w_{k_i}$（若 $w_{i-1}'\in P^*$，则要求 $w_i'\in Q^*$，或相反）。若 $w_1'\in Q^*$，则在 w 的开头加一个 ε 作为新的 $w_1'\in P^*$。将各 w_i' 两两一组（不为偶数则在后面增加一个 ε），则各 $w_{2i-1}'\in P^*,w_{2i}'\in Q^*$，于是 $w\in(P^*Q^*)^*$，即 $(P\cup Q)^*\subseteq(P^*Q^*)^*$。反之亦然，所以 $(P\cup Q)^*=(P^*Q^*)^*$。因此正规集 $L((\mathrm{A}|\mathrm{B})^*)=(L(\mathrm{A})\cup L(\mathrm{B}))^*=((L(\mathrm{A}))^*(L(\mathrm{B}))^*)^*=L((\mathrm{A}^*\mathrm{B}^*)^*)$，于是正规式 (A | B)* = (A*B*)*。

(b) 再证明 (A* | B*)* = (A*B*)*。

由上面的结论，(A* | B*)* = ((A*)*(B*))*，再根据 (2) 的结论，(A* | B*)* = ((A*)*(B*))*= (A*B*)*。

(6)

(a) 先证明 A = b | aA ⇒ A = a*b。

因为正规式 A = b | aA，将左端 A 不断带入右端，即 A = b | a(b | aA) = b | ab | aaA = b | ab | aa(b | aA) = b | ab | aab | aaaA = ⋯ = a*b。

(b) 再证明 A = a*b ⇒ A = b | aA。

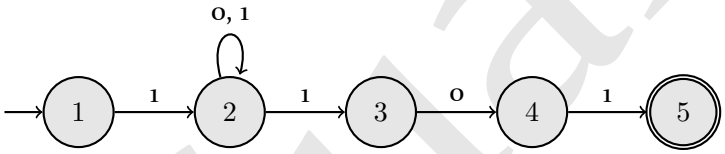
因为正规式 A = a*b，所以正规集 $L(\mathrm{A}) = L(\mathrm{a}^*\mathrm{b}) = L(\mathrm{a}^*) \cdot L(\mathrm{b}) = (L(\mathrm{a}))^* \cdot L(\mathrm{b}) = (\bigcup_{i=0}^{\infty}\{\mathrm{a}\}^i) \cdot \{\mathrm{b}\}$，$L(\mathrm{b} \mid \mathrm{aA}) = L(\mathrm{b}) \cup L(\mathrm{aA}) = \{b\} \cup (L(\mathrm{a}) \cdot L(\mathrm{A})) = \{\mathrm{b}\} \cup (\{\mathrm{a}\} \cdot (\bigcup_{i=0}^{\infty}\{\mathrm{a}\}^i) \cdot \{\mathrm{b}\}) = \{\mathrm{b}\} \cup ((\bigcup_{i=1}^{\infty}\{\mathrm{a}\}^i) \cdot \{\mathrm{b}\}) = (\bigcup_{i=0}^{\infty}\{\mathrm{a}\}^i) \cdot \{\mathrm{b}\} = L(\mathrm{A})$，因此正规式 A = b | aA。

2. P64, T7(1). 构造下列正规式相应的 DFA.

$$1(0|1)^*101$$

解:

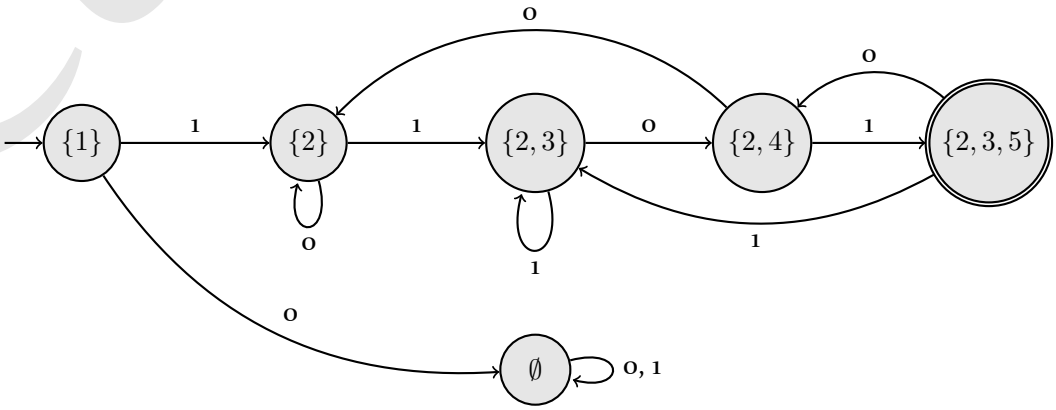
可以先构造一个 NFA 识别该正规语言：



然后用子集法得到 NFA 对应的 DFA 的转移矩阵（新的起始状态为 $\{1\}$，接受状态集合为 $\{\{2,3,5\}\}$）：

画出等价的 DFA 的图表示为：

	0	1
$\{1\}$	$\emptyset$	$\{2\}$
$\emptyset$	$\emptyset$	$\emptyset$
$\{2\}$	$\{2\}$	$\{2,3\}$
$\{2,3\}$	$\{2,4\}$	$\{2,3\}$
$\{2,4\}$	$\{2\}$	$\{2,3,5\}$
$\{2,3,5\}$	$\{2,4\}$	$\{2,3\}$



3. P64, T8(1)-(3) 给出下面正规表达式：

(1) 以 01 结尾的二进制数串；

(2) 能被 5 整除的十进制整数；

(3) 包含奇数个 1 或奇数个 0 的二进制字符串。

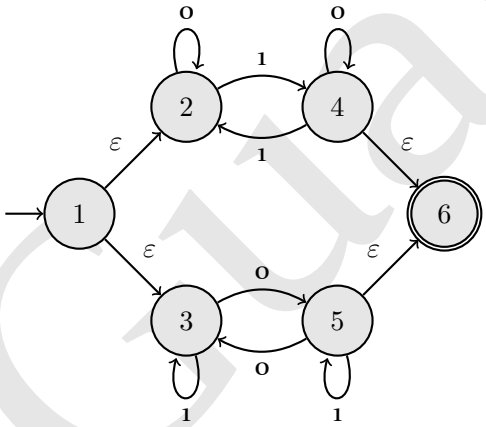
解:

(1) 正规表达式为 $(0 \mid 1)^*01$。

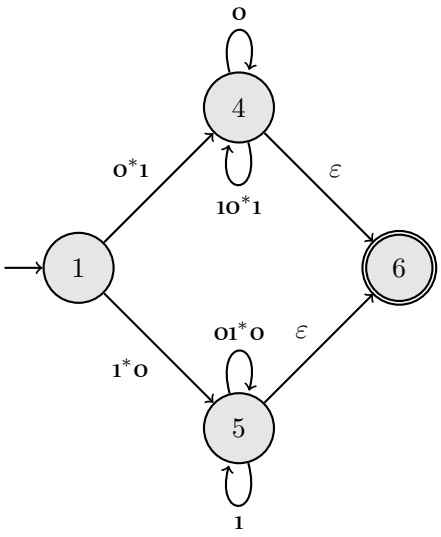
(2) 注意到十进制整数能被 5 整除当且仅当该数末尾为 0 或 5。于是正规表达式为

$0 \mid (1 \mid \cdots \mid 9)(0 \mid 1 \mid \cdots \mid 9)^*(0 \mid 5)$。

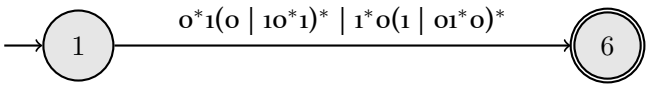
(3) 构造一个 NFA N，状态转换图如下图所示：



约减掉状态 2、3 得到：

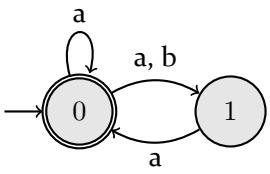


再约减掉状态 4、5 得到：

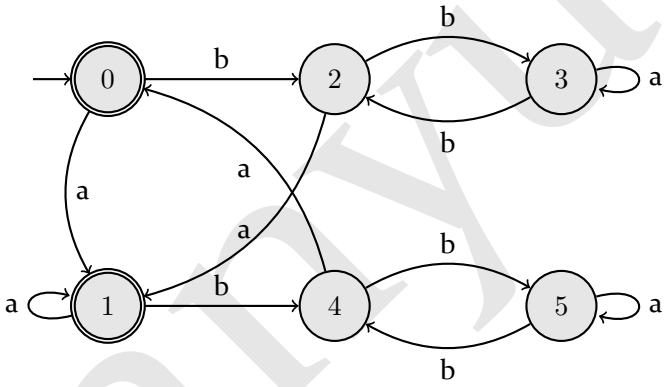


此时得到正规表达式为：0*1(0 | 10*1)* | 1*0(1 | 01*0)*。

4. P64, T12 将图 3.18 的 (a) 和 (b) 分别确定化和最少化。



(a) 需确定化的有限自动机



(b) 需最小化的有限自动机

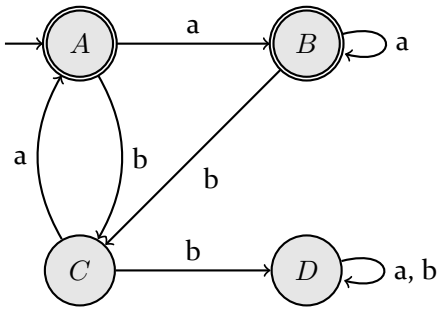
解:

(a) 用子集法确定化。DFA $M = (\Sigma = \{a, b\}, Q = \{\{0\}, \{0, 1\}, \{1\}, \emptyset\}, q_0 = \{0\}, F = \{\{0\}, \{0, 1\}\}, f)$，

先计算转移矩阵：

q	$f(q, \mathrm{a})$	$f(q, \mathrm{b})$
$A = \{0\}$	$\{0, 1\}$	$\{1\}$
$B = \{0, 1\}$	$\{0, 1\}$	$\{1\}$
$C = \{1\}$	$\{0\}$	$\emptyset$
$D = \emptyset$	$\emptyset$	$\emptyset$

画出状态转移图：



用分割的方法进行最小化：

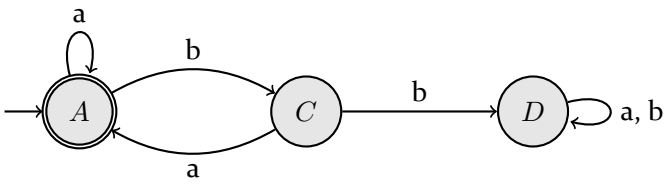
① 先将终态与非终态分开，构成基本分划 $\Pi = \{\{A, B\}, \{C, D\}\}$。

② $f(A,a)=f(B,a)=B$ 在同一子集，$f(A,b)=f(B,b)=C$ 也在同一子集，不分割；

$f(C,a)=A, f(D,a)=D$ 不在同一子集，因此 $\{C,D\}$ 要进行分割，于是得到新分割 $\Pi=\{\{A,B\},\{C\},\{D\}\}$。

③ 与上面相同，$\{A,B\}$ 仍然无法分割，于是上面的 Π 即为最终结果，A,B 可以合并。

最小化后的 DFA 的状态转移图为：



(b) 用分割的方法进行最小化：

① 先将终态与非终态分开，构成基本分划 $\Pi=\{\{0,1\},\{2,3,4,5\}\}$。

② $f(0,\mathtt{a})=f(1,\mathtt{a})=1$ 在同一子集，$f(0,\mathtt{b})=2, f(1,\mathtt{b})=4$ 在同一子集，暂不分割；

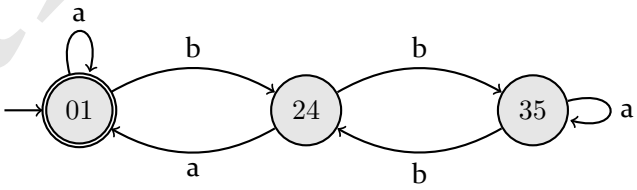
$f(2,\mathtt{a})=1, f(3,\mathtt{a})=3, f(4,\mathtt{a})=0, f(5,\mathtt{a})=5$ 不在同一子集，因此要进行划分，将 $\{2,3,4,5\}$ 划分为 $\{2,4\}$ 和 $\{3,5\}$，得到新的划分 $\Pi=\{\{0,1\},\{2,4\},\{3,5\}\}$。

③ $f(0,\mathtt{a})=f(1,\mathtt{a})=1$ 在同一子集，$f(0,\mathtt{b})=2, f(1,\mathtt{b})=4$ 在同一子集，暂不分割；

$f(2,\mathtt{a})=1, f(4,\mathtt{a})=0$ 在同一子集，$f(2,\mathtt{b})=3, f(4,\mathtt{b})=5$ 在同一子集，暂不分割；

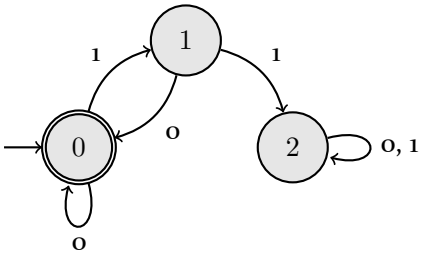
$f(3,\mathtt{a})=3, f(5,\mathtt{a})=5$ 在同一子集，$f(3,\mathtt{b})=2, f(5,\mathtt{b})=4$ 在同一子集，暂不分割。于是上面的 Π 即为最终结果，0 和 1、2 和 4 以及 3 和 5 都可以合并。

最小化后的 DFA 的状态转移图为：



5. P65, T14. 构造一个 DFA，它接受 $\Sigma=\{0,1\}$ 上所有满足如下条件的字符串：每个 **1** 都有 **0** 直接跟在右边。

解: 直接设计 DFA，状态 0 表示当前读到的是 **0**，它是起始状态，也是接受状态；状态 1 表示当前读到的是 **1**，需要根据下一个字符进行判定；状态 2 表示有字符串有 **11** 子串。



6. P65, T15. 给定右线性文法 G：

$$S \to 0S \mid 1S \mid 1A \mid 0B$$

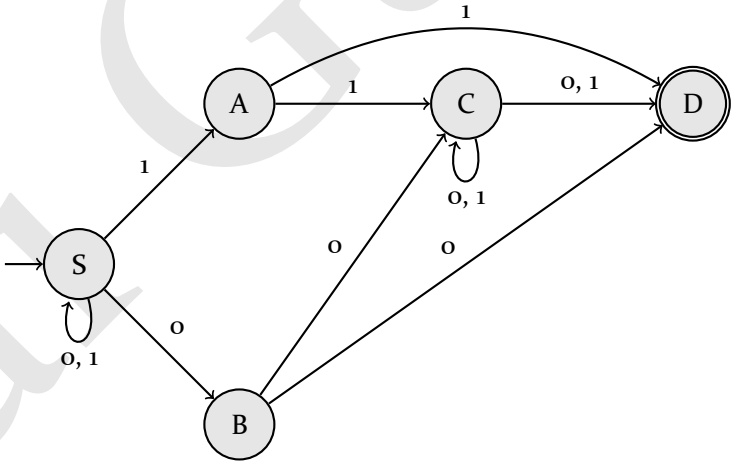
$$A \to 1C \mid 1$$

$$B \to 0C \mid 0$$

$$C \to 0C \mid 1C \mid 0 \mid 1$$

求出一个与 G 等价的左线性文法。

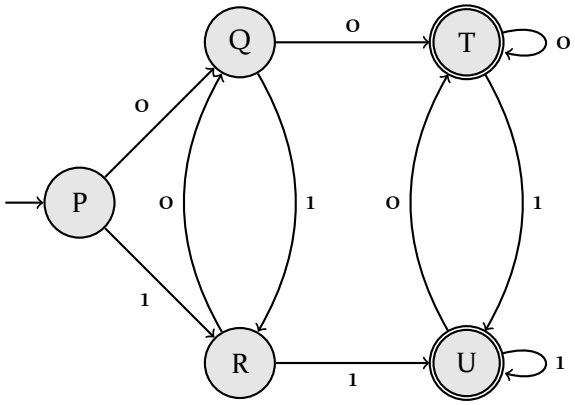
解: 先根据右线性文法构造接受该语言的 NFA $N = (Q, \Sigma, \delta, S, F)$。其中状态集 $Q = \{S, A, B, C, D\}$，字符集 $\Sigma = \{0, 1\}$，起始状态为 S，结束状态集合 $F = \{D\}$，转移函数 δ 按照下图给出：



使用子集法将其转为 DFA，其转移矩阵为（状态子集简写）：

	0	1
P = S	SB	SA
Q = SB	SBCD	SA
R = SA	SB	SACD
T = SBCD	SBCD	SACD
U = SACD	SBCD	SACD

画出状态转移图：



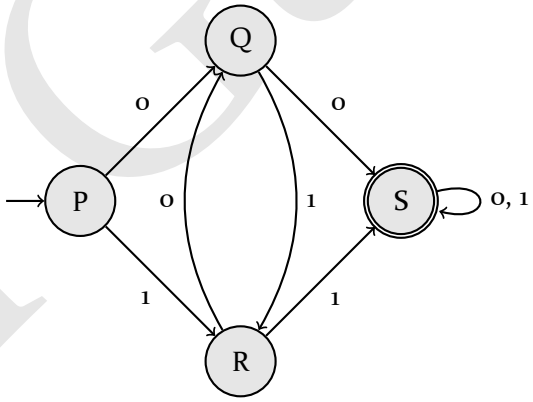
进行最小化：

① $\Pi = \{\{P, Q, R\}, \{T, U\}\}$；

② $\{T, U\}_0 = \{T\}$，$\{T, U\}_1 = \{U\}$ 均在一个子集，不用划分。$\{P, Q, R\}_0 = \{Q, T\}$，$\{P, Q, R\}_1 = \{R, U\}$ 不在同一个集合，需要划分。其中 $\{P, R\}_0 = \{Q\}$，因此 $\{Q\}$ 与 $\{P, R\}$ 应当分开；同时 $\{P, Q\}_1 = \{R\}$，因此 $\{R\}$ 与 $\{P, Q\}$ 应当分开。因此新的划分为 $\Pi = \{\{P\}, \{Q\}, \{R\}, \{T, U\}\}$。

③ $\{T, U\}$ 无法再分，结束。

将 T, U 两状态合并，得到最小化的 DFA：



直接写出左线性文法：

$$S \to So \mid S1 \mid Qo \mid R1$$

$$Q \to Ro \mid o$$

$$R \to Q1 \mid 1$$

二、补充习题

1. 名词解释：翻译程序，编译程序，解释程序，交叉编译程序。

解:

(1) 翻译程序：翻译程序是这样一种程序，它将一种语言的程序（源语言程序）转换成与之逻辑上等价的，另一种语言的程序（目标语言程序）。

(2) 编译程序：编译程序是将高级语言转换成等价的低级语言的一种特殊的翻译程序。

(3) 解释程序：某语言的解释程序是这样的一种程序，它以该语言的源程序作为输入，但不产生目标程序，而是边解释边执行源程序本身。

(4) 交叉编译程序：产生不同于其宿主机的机器代码的编译程序。

2. 画出编译程序总框图，并详细阐述编译程序总框中各模块的功能。

解: 框图如下：

各模块的功能如下：

(1) 词法分析器，又称扫描器，对输入的源程序进行词法分析，输出单词符号。

(2) 语法分析器，简称分析器，对单词符号串进行语法分析（根据语法规则进行推导和归约），识别出各类语法单位，最终判断输入串是否构成语法上正确的程序。

(3) 语义分析与中间代码产生器，按照语义规则对语法分析器归约或推导出的语法单位进行语义分析并把它们翻译成一定形式的中间代码。

(4) 优化器，对中间代码进行优化处理。

(5) 目标代码生成器，把中间代码翻译成目标程序。

除此之外，编译程序还应包含错误处理以及符号表管理两部分。

3. 利用编译程序生成中“T 形图”，说明“移植”、“自编译”方式产生编译程序的方法。

解:

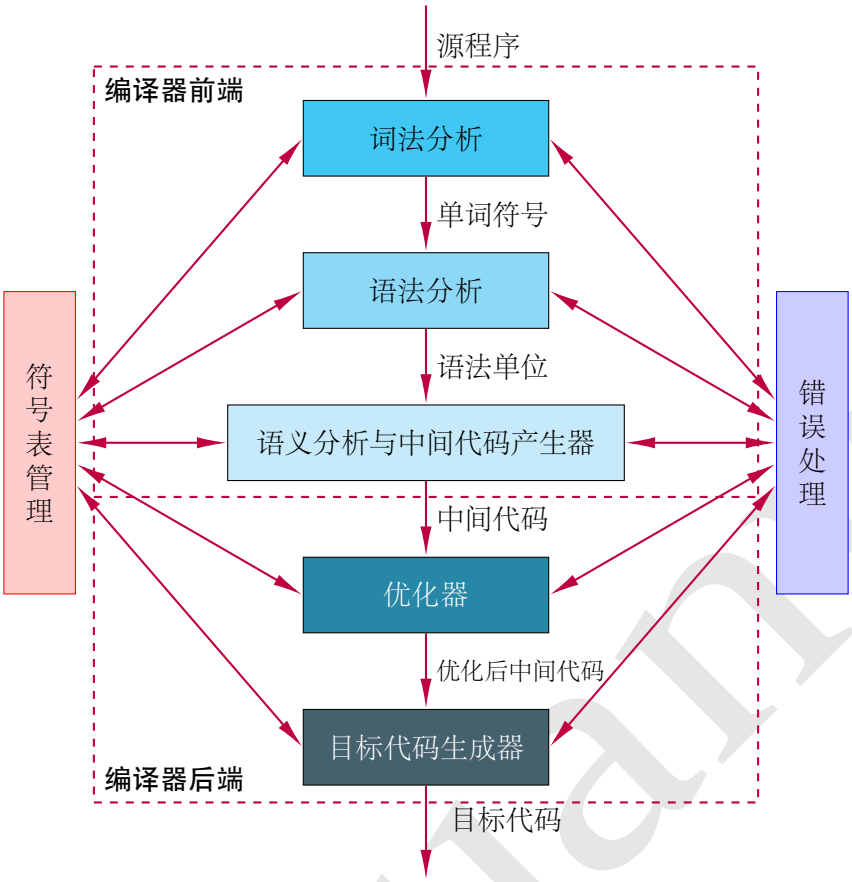


图 2: 编译器结构框图

A. “移植”是指在 A 机器上已经有了某高级语言 H 的编译器（如下图 T 形图）的基础上，希望构建一个 B 机器上的 H 的编译器。

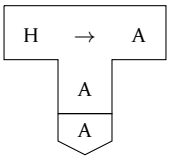


图 3: A 机器（倒五角形）上已有的将高级语言 H 编译成 A 机器码的**编译器 1**

(1) 用 H 语言写一个将 H 语言编译成 B 机器语言的编译器源程序，用**编译器 1** 编译得到**编译器 2**。

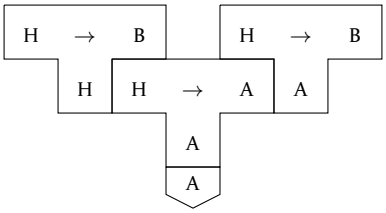


图 4: 编译器 1 编译编译器 2 的源程序，得到**编译器 2**

(2) 用**编译器 2** 再次编译编译器 2 的源程序，于是得到所需目标——**编译器 3**。

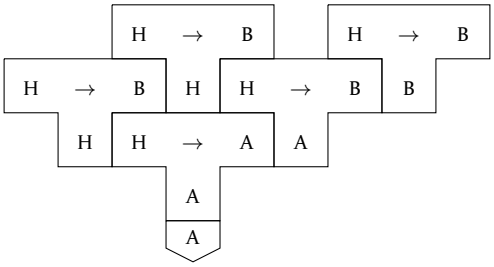


图 5: 编译器 2 编译编译器 2 源程序得到**编译器 3**

之后便可以将目标编译器 3 移到 B 机器上使用了。

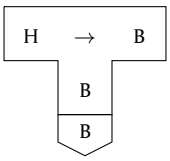


图 6: 将**编译器 3** 移至 B 机器上使用

B. “自编译”的方式是指某机器 A 上没有某高级语言 H 的编译器，需要以“滚雪球”的方式，逐步构建一个 H 语言编译器。

(1) 用 A 机器码构建 H 语言核心部分 H_{K1} 的**编译器 1**。

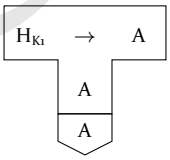


图 7: 将高级语言 H 的核心部分 H_{K1} 编译成 A 机器码的**编译器 1**

(2) 再用 H_{K1} 语言写一个将 H_{K1} 的特性扩大一些后的 H_{K2} 语言编译成 A 机器码的编译器源程序，用**编译器 1** 编译得到**编译器 2**。

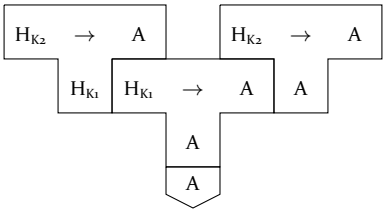


图 8: 编译器 1 编译编译器 2 的源程序，得到**编译器 2**

(3) 像 (2) 一样一步步做下去，不断扩展语言特性，然后用特性更多的语言写编译器，用底下一层的编译器编译，直到得到 H 的编译器。

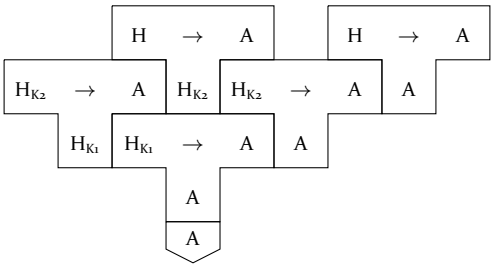


图 9: 编译器 2 编译编译器 2 源程序得到**编译器 3**

三、选做题

1. P64, T11(3). 用某种高级语言写出 DFA 状态最少化的算法。

解: 由于时间所限，我只写出了给出 DFA，计算状态间等价关系的算法。

Algorithm 1 计算 DFA 状态间的等价关系

```
// 初始化等价关系为所有状态对
equiv ← {}
for i ← 0 to n - 1 do
    for j ← i to n - 1 do
        equiv.insert((states[i], states[j]))
    end for
end for
// 开始逐步删除不等价的状态对
while true do
    modified ← true
    for statepair in equiv do
        modified ← false
        // 考虑每一对状态，计算它们经过每一个字符后的状态
        for ch in alphabet do
            stateA ← f(statepair[0], ch)
            stateB ← f(statepair[1], ch)
            // 如果两个状态不等价，删除该对状态
            if (stateA, stateB) not in equiv and (stateB, stateA) not in equiv then
                equiv.erase(statepair)
                modified ← true
            end if
        end for
    end for
    if modified == false then
        break
    end if
end while
```

下面是源代码:

```
#include <iostream>
#include <fstream>
#include <any>
#include <string>
#include <set>
#include <unordered_set>
#include <map>

class DFA
{
    public:
        DFA(const std::unordered_set<std::string> & states,
            const std::unordered_set<char> & alphabet,
            const std::map<std::pair<std::string, char>, std::string> &
   transFunc,
            const std::string & startState,
            const std::unordered_set<std::string> & acceptStates);
        ~DFA();
        // 字符串是否是状态
        bool isState(const std::string & s);
        // 是否是初始状态
        bool isStartState(const std::string & s);
        // 是否是接受态
        bool isAcceptState(const std::string & s);
```

```
24          // 是否是有效的 DFA
25          bool isValid();
26          // 打印 DFA
27          void print();
28          // 模拟字符串计算
29          bool calcString(const std::string & s);
30          // 最小化
31          void minimization();
32      private:
33          // 状态集: 每个状态是一个字符串
34          std::unordered_set<std::string> states;
35          // 字符表: 字符的集合
36          std::unordered_set<char> alphabet;
37          // 转移函数: states x alphabet -> states,
38          // <state, char> -> state
39          std::map<std::pair<std::string, char>, std::string> transFunc;
40          // 开始状态: 一个字符串, 必须在状态集中
41          std::string startState;
42          // 接受状态: 字符串的集合, 必须在状态集中
43          std::unordered_set<std::string> acceptStates;
44  };
45
46  DFA::DFA(const std::unordered_set<std::string> & states,
47          const std::unordered_set<char> & alphabet,
```

```
48          const std::map<std::pair<std::string, char>, std::string> &
    transFunc,
49          const std::string & startState,
50          const std::unordered_set<std::string> & acceptStates)
51 : states(states), alphabet(alphabet), transFunc(transFunc), startState(
    startState), acceptStates(acceptStates)
52 {}
53
54 DFA::~DFA(){}
55
56 bool DFA::isState(const std::string & s)
57 {
58     return states.find(s) != states.end();
59 }
60
61 bool DFA::isStartState(const std::string & s)
62 {
63     return (s == startState);
64 }
65
66 bool DFA::isAcceptState(const std::string & s)
67 {
68     return acceptStates.find(s) != acceptStates.end();
69 }
```

```
70
71 bool DFA::isValid()
72 {
73     // 状态集合非空
74     bool flag = !states.empty();
75     // 字符集非空
76     flag &= !alphabet.empty();
77     // 转移函数都有定义
78     for(auto & state : states)
79     {
80         for(auto & ch : alphabet)
81         {
82             flag &= (transFunc.find(std::make_pair(state, ch)) !=
    transFunc.end());
83         }
84     }
85     // 开始状态必须在状态集合中
86     flag &= isState(startState);
87     // 接受状态必须包含于状态集合中
88     for(auto & state : acceptStates)
89     {
90         flag &= isState(state);
91     }
92     return flag;
```

```
93 }
94
95 void DFA::print()
96 {
97     if(!this -> isValid())
98     {
99         std::cout << "Warning: an invalid DFA." << std::endl;
100         return;
101     }
102
103     std::cout << "States: " << std::endl;
104     for(auto & state : states)
105     {
106         std::cout << state << " ";
107     }
108     std::cout << std::endl;
109
110     std::cout << "Alphabet: " << std::endl;
111     for(auto & ch : alphabet)
112     {
113         std::cout << ch << " ";
114     }
115     std::cout << std::endl;
116
```

```
117     std::cout << "Transive function: " << std::endl;
118     for(auto & state : states)
119     {
120         for(auto & ch : alphabet)
121         {
122             std::cout << state << "-" << ch << "->" << transFunc[std::
    make_pair(state, ch)] << " ";
123         }
124         std::cout << std::endl;
125     }
126 
127     std::cout << "Start state: " << startState << std::endl;
128 
129     std::cout << "Accept states: " << std::endl;
130     for(auto & state : acceptStates)
131     {
132         std::cout << state << " ";
133     }
134     std::cout << std::endl;
135 }
136 
137 bool DFA::calcString(const std::string & s)
138 {
139     // 检查是否有效
```

```
140     if(!this -> isValid())
141     {
142         std::cout << "Invalid DFA." << std::endl;
143         return false;
144     }
145     // 初始状态
146     std::string state = this -> startState;
147     // 逐个读取
148     for(auto & i : s)
149     {
150         state = transFunc[std::make_pair(state, i)];
151     }
152     // 最终结果在接受状态中
153     return isAcceptState(state);
154 }
155
156 void DFA::minimization()
157 {
158     // 计算等价关系
159     std::set<std::pair<std::string, std::string>> equiv;
160     // 初始化
161     for(auto i = states.begin(); i != states.end(); i++)
162     {
163         for(auto j = i; j != states.end(); j++)
```

```
164         {
165             equiv.emplace(std::make_pair(*i, *j));
166         }
167     }
168     // 将终态与非终态分开
169     for(auto i = equiv.begin(); i != equiv.end(); )
170     {
171         // 二者不都是终态，就删除
172         if((isAcceptState(i -> first) && !isAcceptState(i -> second))
173         || (!isAcceptState(i -> first) && isAcceptState(i -> second)))
174         {
175             i = equiv.erase(i);
176         }
177         else
178         {
179             i++;
180         }
181     }
182     // 开始删除不等价的状态
183     while(true)
184     {
185         bool modified = true;
186         // 考虑每一对状态
187         for(auto i = equiv.begin(); i != equiv.end(); )
```

```
188         {
189             // 认为没有修改
190             modified = false;
191             // 考虑每个字符
192             for(auto j = alphabet.begin(); j != alphabet.end(); j++)
193             {
194                 // sa = f(i -> first, *j)
195                 // sb = f(i -> second, *j)
196                 std::string stateA = transFunc[std::make_pair(i -> first,
     *j)];
197                 std::string stateB = transFunc[std::make_pair(i -> second
    , *j)];
198                 // <sa, sb> <sb, sa> 都不在等价关系
199                 if(equiv.find(std::make_pair(stateA, stateB)) == equiv.
    end()
200                     && equiv.find(std::make_pair(stateB, stateA)) ==
    equiv.end())
201                 {
202                     // 则需要删除该对
203                     modified = true;
204                     break;
205                 }
206             }
207             // 删除该对
```

```
208             if(modified)
209             {
210                 i = equiv.erase(i);
211             }
212             // 考虑下一对
213             else
214             {
215                 i++;
216             }
217         }
218         // 如果本轮没有删除的
219         if(!modified)
220         {
221             break;
222         }
223     }
224
225     // 合并
226     for(auto & i : equiv)
227     {
228         if(i.first != i.second)
229         {
230             std::cout << "Merge: (" << i.first << ", " << i.second << ")"
    << std::endl;
```

```
231         }
232     }
233 }
234
235 int main()
236 {
237     std::unordered_set<std::string> states = {"0", "1", "2", "3", "4", "5
        "};
238     std::unordered_set<char> alphabet = {'a', 'b'};
239     std::map<std::pair<std::string, char>, std::string> transFunc = \
240     {
241         {{"0", 'a'}, "1"}, {{"0", 'b'}, "2"},
242         {{"1", 'a'}, "1"}, {{"1", 'b'}, "4"},
243         {{"2", 'a'}, "1"}, {{"2", 'b'}, "3"},
244         {{"3", 'a'}, "3"}, {{"3", 'b'}, "2"},
245         {{"4", 'a'}, "0"}, {{"4", 'b'}, "5"},
246         {{"5", 'a'}, "5"}, {{"5", 'b'}, "4"}
247     };
248     std::string startState = "0";
249     std::unordered_set<std::string> acceptStates = {"0", "1"};
250     DFA M(states, alphabet, transFunc, startState, acceptStates);
251
252     std::cout << "Is M valid: " << M.isValid() << std::endl;
253     M.print();
```

```
254     //std::cout << "Is aa accepted: " << M.calcString("aa") << std::endl;
255     //std::cout << "Is aabb accepted: " << M.calcString("aabb") << std::
    endl;
256     M.minimization();
257     // std::cout << "After minimization:" << std::endl;
258     // M.print();
259     return 0;
260 }
```