223|3407 王飞雨　　编译第五次作业.

练习1：

1. (1) $L(A|A) = L(A) \cup L(A) = L(A)$

$\therefore A|A = A$

(2) 由定义，$A^*$表示对正则表达式$A$进行零次或多次的连接运算。

而对$(A^*)^*$，其结果仍是$L(A)$的所有可能排列组合，故$(A^*)^* = A^*$

(3) $L(\varepsilon|AA^*) = L(\varepsilon) \cup L(A)L(A^*) = L(\varepsilon) \cup L(A)(L(A))^*$

$= L(\varepsilon) \cup L(A)((L(A))^0 \cup (L(A))^1 \cup \cdots)$

$= L(\varepsilon) \cup ((L(A))^1 \cup (L(A))^2 \cup \cdots)$

$= (L(A))^* = L(A^*)$　　$\because A^* = \varepsilon|AA^*$

(4) 对分配律和结合律：

$(AB)^*A = ((AB)^0|(AB)^1|\cdots)A = \varepsilon A|(AB)^1A|(AB)^2A|\cdots$

$= A\varepsilon|A(BA)^1|A(BA)^2|\cdots = A(\varepsilon|(BA)^1|(BA)^2|\cdots)$

$= A(BA)^*$

(5) ~~$L(A) \subseteq L(A) \cup L(B)$~~　~~$L(B) \subseteq L(A) \cup L(B)$~~　$\because$ 由(2)知 $(L(A) \cup L(B))^* \subseteq (L(A)^* \cdot L(B)^*)^*$ ①

$\because L(A) \subseteq L(A)^*L(B)^*$　　$L(A) \subseteq L(A) \cup L(B)$　　$L(B) \subseteq L(A) \cup L(B)$

$L(B) \subseteq L(A)^*L(B)^*$　　$\therefore L(A)^* \subseteq (L(A) \cup L(B))^*$　$L(B)^* \subseteq (L(A) \cup L(B))^*$

$\therefore L(A) \cup L(B) \subseteq L(A)^*L(B)^*$　　$\therefore L(A)^* \cdot L(B)^* \subseteq (L(A) \cup L(B))^*$

$\therefore (L(A)^* \cdot L(B)^*)^* \subseteq ((L(A) \cup L(B))^*)^* = (L(A) \cup L(B))^*$ ②

由①②可知：$(A|B)^* = (A^*B^*)^*$

~~$\because L((A^*B^*)^*) = (L(A)^* L(B)^*)^*$~~

$\because$ A，B 是任意正则表达式 $\therefore (A^*|B^*)^* = ((A^*)^*(B^*)^*)^*$

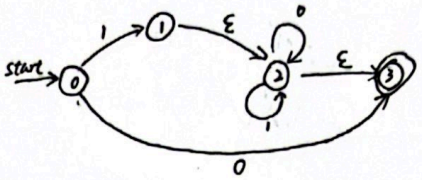
$\because (A^*)^* = A^*$ $(B^*)^* = B^*$ $\therefore (A^*|B^*)^* = (A^*B^*)^*$

$\therefore (A|B)^* = (A^*B^*)^* = (A^*|B^*)^*$

2.

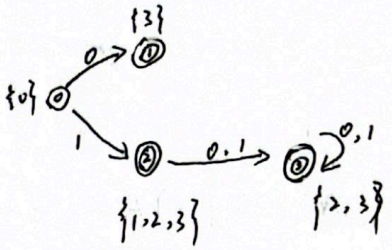
(1)

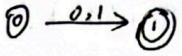
由题，可构造NFA：




| | I | $I_0$ | $I_1$ |
|---|---|---|---|
| ① | {0} | {3} ④ | {1,2,3} ② |
| ④ | {3} | ∅ - | ∅ - |
| ② | {1,2,3} | {2,3} ⑤ | {2,3} ⑤ |
| ⑤ | {2,3} | {2,3} ⑤ | {2,3} ⑤ |

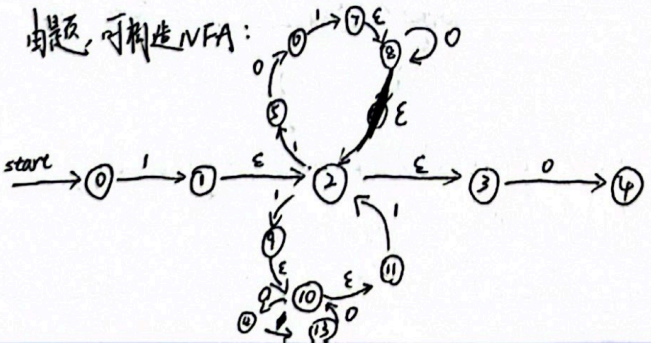
$\therefore$ DFA：



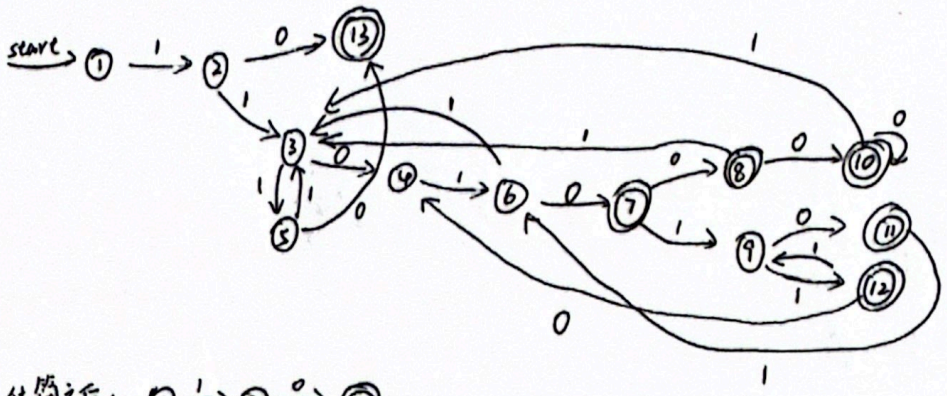

$\therefore$ 化简后： 


(2).

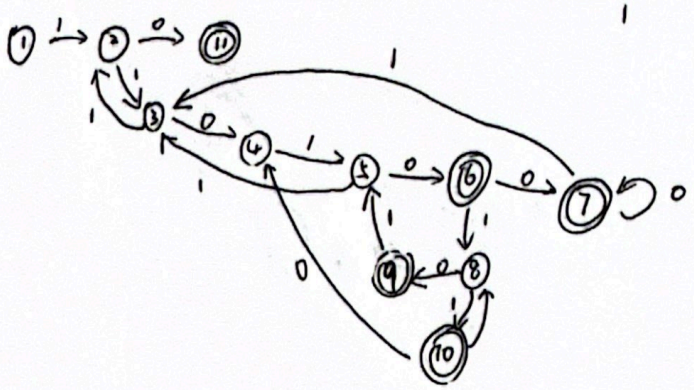
由题，可构造NFA：



化为DFA得：

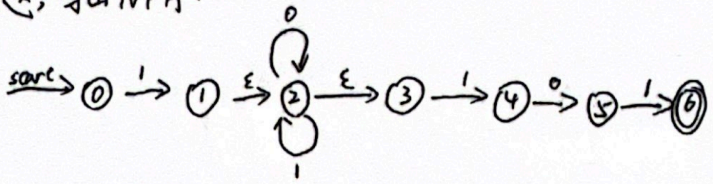



化简之后：

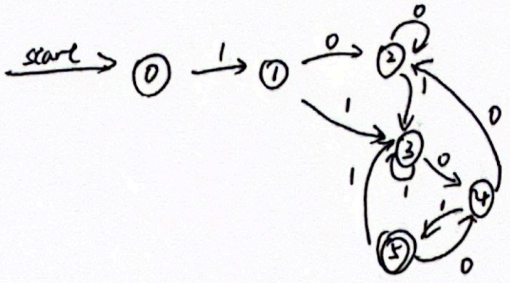


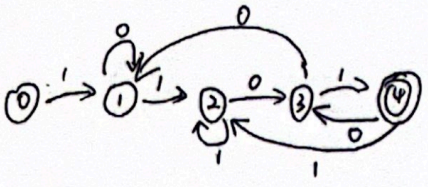

(3)

由题，写出NFA：

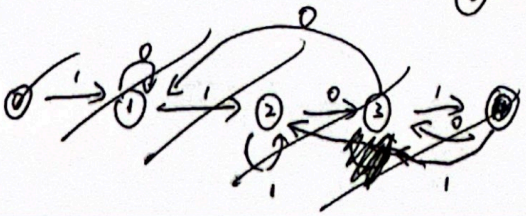



化为DFA得：



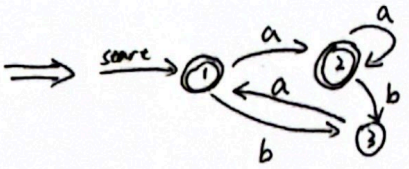

化简之后：



4. (a).

确定化：

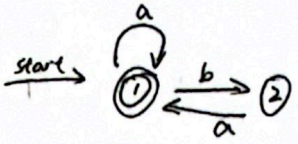
| | $I$ | $I_a$ | $I_b$ |
|---|---|---|---|
| ① | {0} | {0,1} ② | {1} ③ |
| ② | {0,1} | {0,1} ② | {1} ③ |
| ③ | {1} | {0} ① | ∅ |




最小化：

| | a | b | |
|---|---|---|---|
| 1 | 2 | 3 | ① |
| 2 | 2 | 3 | |
| 3 | 1 | - | ② |

∴ 化简后：

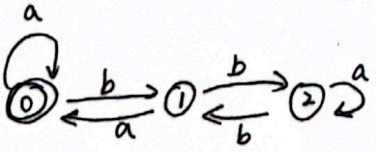


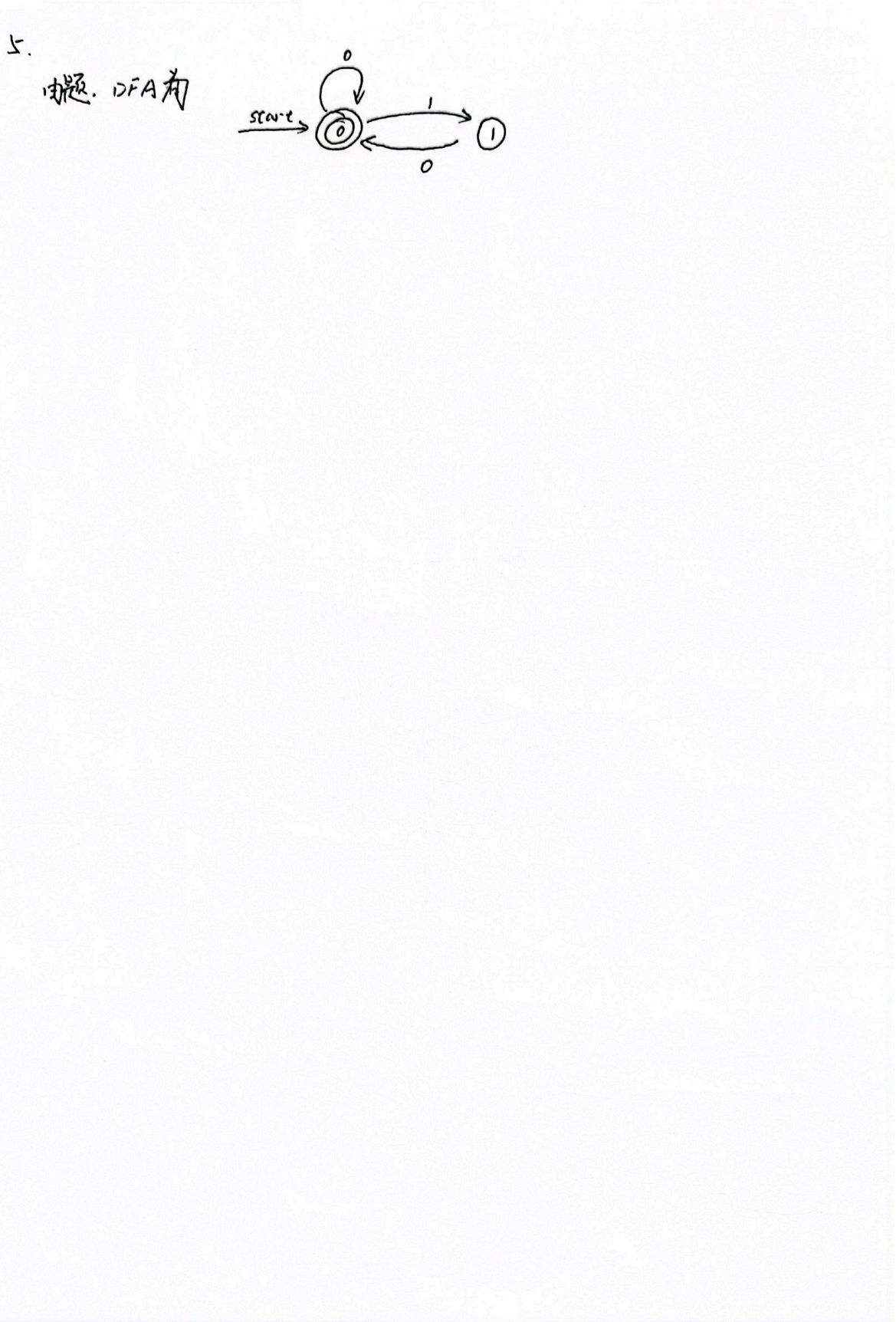

(b).

原图已经确定化：

最小化：

| | a | b | |
|---|---|---|---|
| 0 | 1 | 2 | ① |
| 1 | 1 | 4 | |
| 2 | 1 | 3 | ① |
| 3 | 3 | 2 | |
| 4 | 0 | 5 | ② |
| 5 | 5 | 4 | |






## 选做：Lex 工具如何实现自动词法分析模块

Lex 是一个用于生成词法分析器的工具，它使用正则表达式来定义词法单元。Lex 文件中定义了源代码中的模式和与之关联的动作。Lex 通过读取这些模式生成一个可以识别源代码中的词法单元的 C 语言程序。原理是：给定RE → NFA → DFA → 极小化，从而自动生成词法分析程序。

Lex 文件的基本结构如下：

```
%{
    // 这里是全局变量
    int num_lines = 0;
%}

%%
    // 正则表达式             动作
    [a-zA-Z][a-zA-Z0-9]*   { printf("IDENTIFIER: %s\n", yytext); }
    [0-9]+                 { printf("NUMBER: %s\n", yytext); }
    \n                     { num_lines++; }
    .                      { printf("OTHER: %s\n", yytext); }
%%

int main() {
    yylex(); // 启动词法分析器
    printf("Total lines: %d\n", num_lines);
}
```

## 主要部分说明：

1. **声明部分**（`%{ ... %}`）：在这个部分中可以定义 C 语言的头文件和全局变量。在上述例子中，`num_lines` 用来记录行数。
2. **规则部分**（`%%` 之间的部分）：每一行由正则表达式和相应的动作组成。当源代码匹配某个正则表达式时，将执行其对应的动作。
   - `[a-zA-Z][a-zA-Z0-9]*` 匹配标识符并输出。
   - `[0-9]+` 匹配整数并输出。
   - `\n` 用于匹配换行符，并增加行数计数器。
   - `.` 匹配任意字符。
3. **用户代码部分**：在最后的 `%%` 后面，可以编写完整的 C 语言代码。在这里，`yylex()` 启动词法分析，逐行扫描输入。

## 示例

```
%{
    //声明部分
    int num_lines = 0;   // 用于记录行数
    int num_chars = 0;   // 用于记录字符数
%}

%%
    // 规则部分：正则表达式 + 动作

    // 匹配标识符 (字母开头，后面可以跟字母或数字)
    [a-zA-Z_][a-zA-Z0-9_]*   { printf("IDENTIFIER: %s\n", yytext); }

    // 匹配整数
    [0-9]+                   { printf("NUMBER: %s\n", yytext); }

    // 匹配运算符 + - * /
    [+\-*/]                  { printf("OPERATOR: %s\n", yytext); }

    // 匹配换行符，并统计行数
```

```
    \n                      { num_lines++; }

    // 匹配其他任意字符
    .                       { num_chars++; }
%%

int main() {
    // 主程序入口，启动词法分析器
    printf("Starting lexical analysis...\n");
    yylex();  // 调用词法分析器开始工作

    // 输出总行数和字符数
    printf("Total lines: %d\n", num_lines);
    printf("Total characters: %d\n", num_chars);

    return 0;
}

int yywrap() {
    // 告诉词法分析器处理完毕
    return 1;
}
```

**文件解释：**

1. **声明部分**（`%{ ... %}`）：在这里我们声明了两个全局变量 `num_lines` 和 `num_chars` 分别用于记录总行数和字符数。
2. **规则部分**（`%%` 之间的部分）：定义了一系列正则表达式与相应的动作：
   - `[a-zA-Z_][a-zA-Z0-9_]*` 匹配标识符，并在控制台打印出识别的标识符。
   - `[0-9]+` 匹配整数，并打印出识别的数字。
   - `[+\-*/]` 匹配常见的运算符，如 `+`、`-`、`*`、`/`，并打印出识别的运算符。
   - `\n` 匹配换行符，并增加行数计数器。
   - `.` 匹配所有其他字符，并增加字符计数器。
3. **用户代码部分**：在 `%%` 之后的部分是常规的 C 语言代码：
   - 在 `main()` 函数中调用 `yylex()` 启动词法分析器。
   - `yywrap()` 函数用于告诉 Lex 文件词法分析器何时结束分析。返回 `1` 表示结束。

**编译与执行**

要编译并运行这个 Lex 文件，您可以按照以下步骤进行：

1. 保存为 `example.l`。
2. 使用 Lex 工具生成 C 代码：

```
lex example.l
```

3. 使用 C 编译器编译生成的 `lex.yy.c` 文件：

```
gcc lex.yy.c -o lexer -ll
```

4. 运行生成的词法分析器：

```
./lexer < input.txt
```

其中，`input.txt` 是包含待分析代码的文件。例如，`input.txt` 文件内容如下：

```
int a = 10 + 20;
```

运行输出将类似如下：

```
IDENTIFIER: int
IDENTIFIER: a
OPERATOR: =
NUMBER: 10
OPERATOR: +
NUMBER: 20
OPERATOR: ;
Total lines: 1
Total characters: 19
```