编译原理作业

上交时间：第十三周周二 23:59 截止

作业要求：

1. 请把实验 1 所需要完成的所有单词的正则表达式写出来。
说明：
（1）实验 1 单词的网址: https://rustwiki.org/zh-CN/reference/tokens.html
（2）如果有无法采用正则表达式表示的单词，请在作业中逐个列出来。

```
# 基础字符类与宏（不生成 DFA）
letter        = [A-Za-z]
digit         = [0-9]
lower         = [a-z]
upper         = [A-Z]
underscore    = [_]
hex_digit     = [0-9A-Fa-f]
oct_digit     = [0-7]
bin_digit     = [0-1]
alnum_        = letter | digit | underscore
nonzero       = [1-9]

# 分隔符占位符宏（不生成 DFA）
# 说明：由于 ( ) [ ] 是正则元字符，我们用占位符宏来表示它们
lparen_char   = lparen_placeholder
rparen_char   = rparen_placeholder
lbrack_char   = lbrack_placeholder
rbrack_char   = rbrack_placeholder

# ----------------------
# 标识符 / 原始标识符 / 生命周期（ASCII 近似）
# ----------------------
IDENT1        = ( underscore | letter ) ( alnum_ ) *
RAW_IDENT1    = r # IDENT1
LIFETIME1     = ' ( underscore | letter ( alnum_ ) * )

# ----------------------
# 关键字（按 Rust 稳定关键字 + 常用扩展，匹配时应优先于 IDENT1）
# ----------------------
KW1 = (
  as | break | const | continue | crate | else | enum | extern | false | fn | for | if
  | impl | in | let | loop | match | mod | move | mut | pub | ref | return
```

```
  | Self | self | static | struct | super | trait | true | type | unsafe | use
  | where | while | async | await | dyn | union | yield | macro | macro_rules | try
)

# ----------------------
# 整数字面量（可生成 NFA 的近似规则）
# ----------------------
# 辅助宏（不含数字，不生成 DFA）
HEX_DIGITS   = hex_digit ( hex_digit | underscore ) *
OCT_DIGITS   = oct_digit ( oct_digit | underscore ) *
BIN_DIGITS   = bin_digit ( bin_digit | underscore ) *

# 目标词法单元（含数字，生成 DFA）
INT_DEC1      = 0 | nonzero ( digit | underscore ) *
INT_HEX1      = 0 x HEX_DIGITS
INT_OCT1      = 0 o OCT_DIGITS
INT_BIN1      = 0 b BIN_DIGITS

INT_SUFFIX   = ( u8 | u16 | u32 | u64 | u128 | usize | i8 | i16 | i32 | i64 | i128 | isize )
INT_LIT1      = ( INT_DEC1 | INT_HEX1 | INT_OCT1 | INT_BIN1 ) INT_SUFFIX ?

# 布尔字面量（也可视作关键字）
BOOL_LIT1     = true | false

# ----------------------
# 分隔符与界符
# ----------------------
LPAREN1       = lparen_char
RPAREN1       = rparen_char
LBRACE1       = {
RBRACE1       = }
LBRACK1       = lbrack_char
RBRACK1       = rbrack_char
COMMA1        = ,
SEMI1         = ;
COLON1        = :

# --- 浮点数字面量 ---
# 局限性：无法实现。因为本项目将 . 视为连接运算符，无法用作字面量小数点。
# FLOAT_LIT1 = ( INT_DEC1 . ( INT_DEC1 )? ( ( e | E ) ( + | - )? INT_DEC1 )? ) | ( INT_DEC1 ( e
| E ) ( + | - )? INT_DEC1 )
# --- 字符串与字符 ---
# 局限性：无法实现。any_char_except_quote 是一个概念性占位符，代表“除引号外的任意字符”，
当前项目正则无此表达能力。同时，复杂转义如 \u{...} 也无法处理。
```

```
# 概念性规则：
# STRING_LIT1 = " ( any_char_except_quote | \ " )* "
# CHAR_LIT1   = ' ( any_char_except_quote | \ ' ) '

# --- 注释 ---
# 局限性：行注释无法实现，因缺少“任意非换行符”的表达。块注释的正则仅支持非嵌套形式。
# 概念性规则（行注释）：
# LINE_COMMENT1 = / / any_char_except_newline*
# 近似规则（块注释，非嵌套）：
BLOCK_COMMENT1 = / * ( star | any_char_except_star )* * /
# (其中 star = [*], any_char_except_star 为占位符)
```

表 1 可以采用正则表达式的单词

```
# 完全无法使用正则表示（非正则语言）
# 1) 原始字符串与原始字节串：r"...", r#"..."#, r##"..."## 等
#    原因：需要两侧 # 数量对称匹配，属于 a^n b a^n 形式的上下文相关语言，非正则。
#
# 2) 完整的 Unicode 标识符（XID_Start/XID_Continue）
#    原因：本文件给出的是 ASCII 近似版（IDENT1）。完整支持需要基于 Unicode 表判断，超出简单正则范围。
#
# 3) 由标点组成的多字符运算符与界符
#    原因：例如 :: -> => .. ..= . ? 等，在本项目正则中属于元字符或缺少转义，建议直接在词法器里“最长匹配”。
```

表 2 无法采用正则表达式的单词

2．请按图 1 DFA 图画法的参考图示方式把实验 1 所需要完成的所有单词的 DFA 图画出来。

编译原理作业

(1) 标识符/原始标识符/生命周期

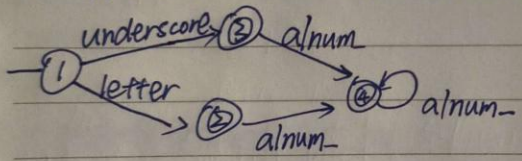
letter = [A-Z a-z]

digit = [0-9]

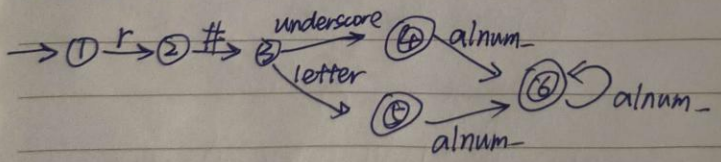
underscore = [_]

&alnum_ = letter | digit | underscore

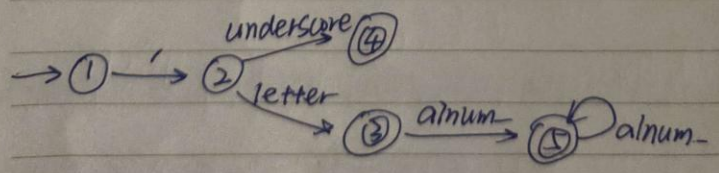
标识符 IDENT1 = (underscore | letter)(&alnum_)*



原始标识符 RAW_IDENT1 = r#IDENT1



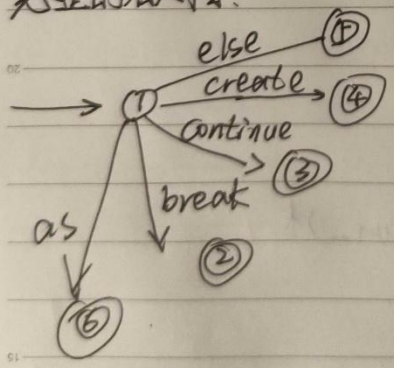
生命周期 lifetime1 = '(underscore | letter(alnum_)*)



(2)关键字

keyword = (as|break|continue|create|else|…)省略剩下的.

完整的看文档.



(3) 整数字面量

辅助宏：定义 HEX_DIGITS = hex_digit (hex_digit | underscore)*

OCT_DIGITS = oct_digit (oct_digit | underscore)*

BIN_DIGITS = bin_digit (bin_digit | underscore)*

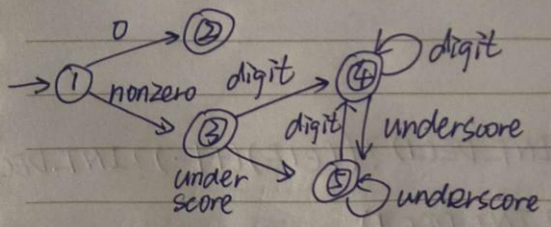
hex_digit = [0-9 a-f A-F]

oct_digit = [0-7]

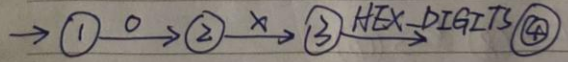
bin_digit = [0-2]

hozero = [1-9]

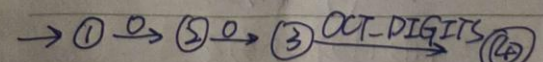
十进制整型字面量 INT_DEC1 = 0 | nonzero (digit | underscore)*



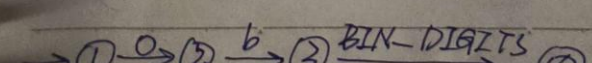
十六进制整型字面量 INT_HEX1 = 0x HEX_DIGITS



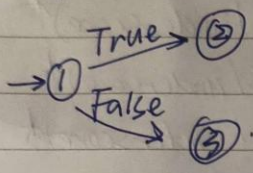
八进制整型字面量 INT_OCT1 = 0o OCT_DIGITS



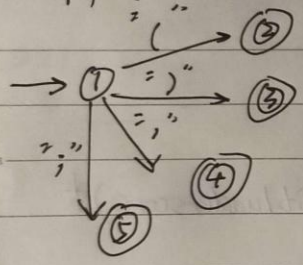
二进制整型字面量 INT_BIN1 = 0b BIN_DIGITS



布尔字面量 BOOL_LIT1 = True/False

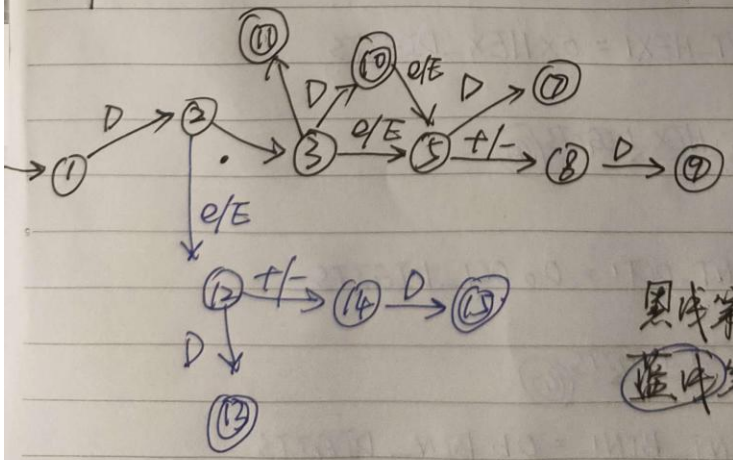


(4)分隔符



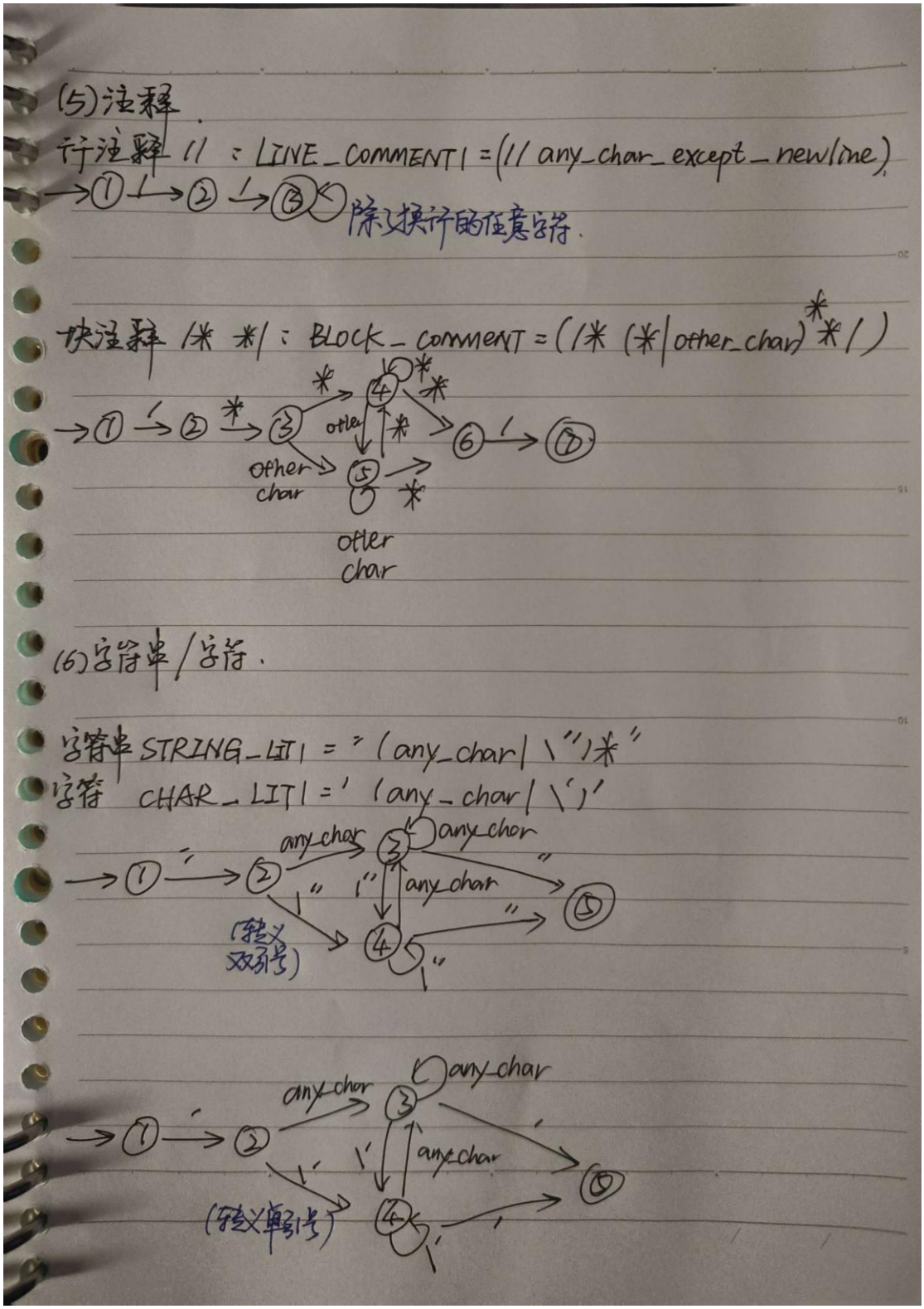
(4)浮点数字面量

FLOAT_LIT1 = (INT_DEC1 . (INT_DEC1)?((e|E)(+|-)?INT_DEC)?) | (INT_DEC1 (e|E)(+|-)? INT_DEC1)



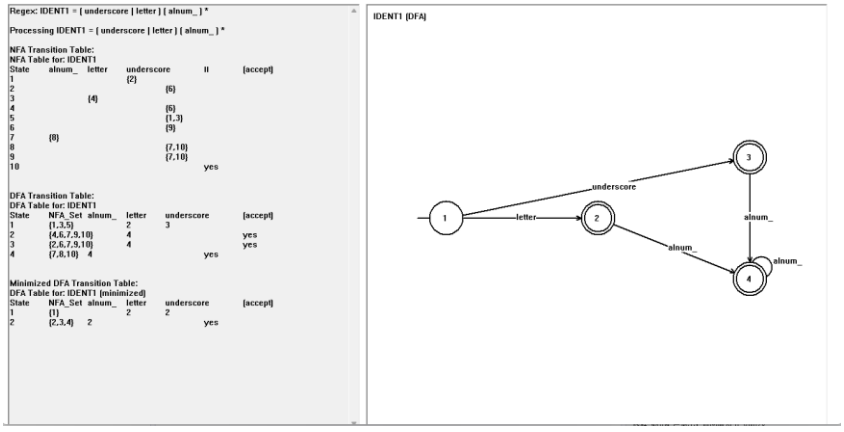
黑线第一个分支，

蓝线第二个分支。



3.运行实验 2 的程序，输入上述作业要求 1 所得到的正则表达式，并把相应的输出结果（最小化的 DFA）进行截图。通过比较上述作业要求 2 所画的 DFA 和实验 2 的输出结果，写出你所实现的实验 2 的测试结果。

IDENT1 = (underscore | letter) (alnum_) *



Regex: IDENT1 = (underscore | letter) (alnum_) *

Processing IDENT1 = (underscore | letter) (alnum_) *

NFA Transition Table:
NFA Table for: IDENT1

State	alnum_	letter	underscore	ll	[accept]
1			{2}		
2				{6}	
3		{4}			
4				{6}	
5				{1,3}	
6				{9}	
7	{8}				
8				{7,10}	
9				{7,10}	
10					yes

DFA Transition Table:
DFA Table for: IDENT1

State	NFA_Set	alnum_	letter	underscore	[accept]
1	{1,3,5}		2	3	
2	{4,6,7,9,10}		4		yes
3	{2,6,7,9,10}		4		yes
4	{7,8,10}	4			yes

Minimized DFA Transition Table:
DFA Table for: IDENT1 [minimized]

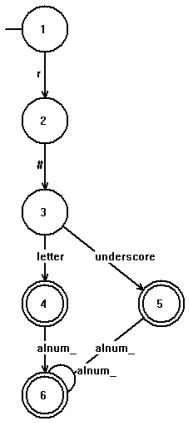
State	NFA_Set	alnum_	letter	underscore	[accept]
1	{1}		2	2	
2	{2,3,4}	2			yes

IDENT1 (DFA)

图表 1 标识符

RAW_IDENT1= r # IDENT1

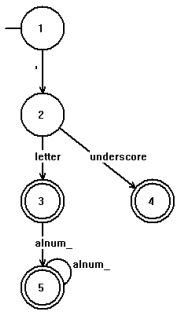
RAW_IDENT1 (DFA)



图表 2 原始标识符

LIFETIME1 = ' (underscore | letter (alnum_) *)

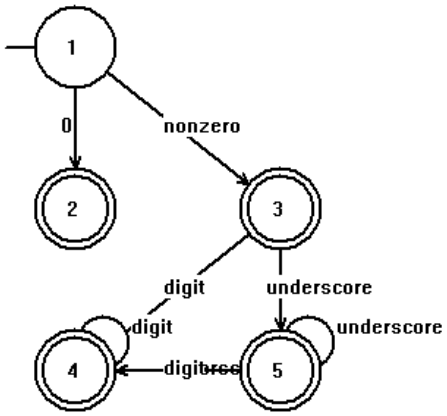
LIFETIME1 (DFA)



图表 3 生命周期

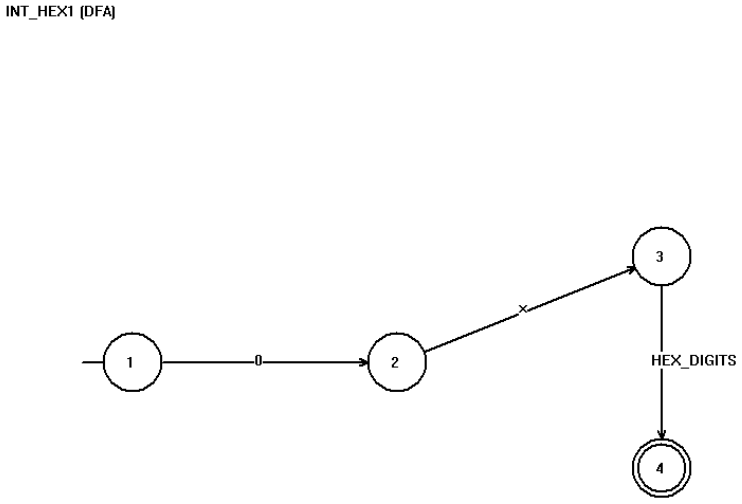
INT_DEC1 = 0 | nonzero (digit | underscore) *

INT_DEC1 (DFA)



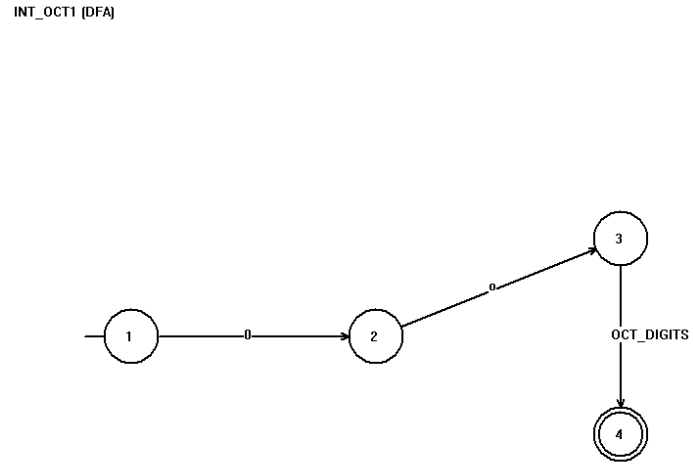
图表 4 十进制整型

INT_HEX1 = 0 x HEX_DIGITS



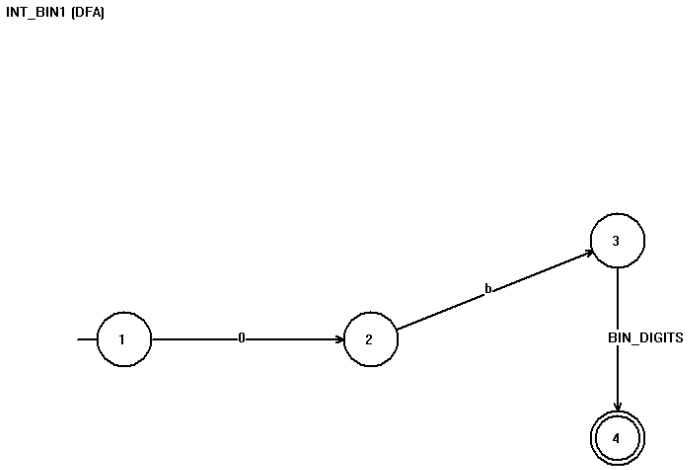
图表 5 十六进制整型

INT_OCT1 = 0 o OCT_DIGITS



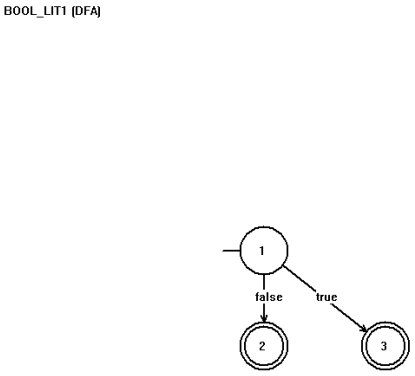
图表 6 八进制整型

INT_BIN1 = 0 b BIN_DIGITS



图表 7 二进制整型

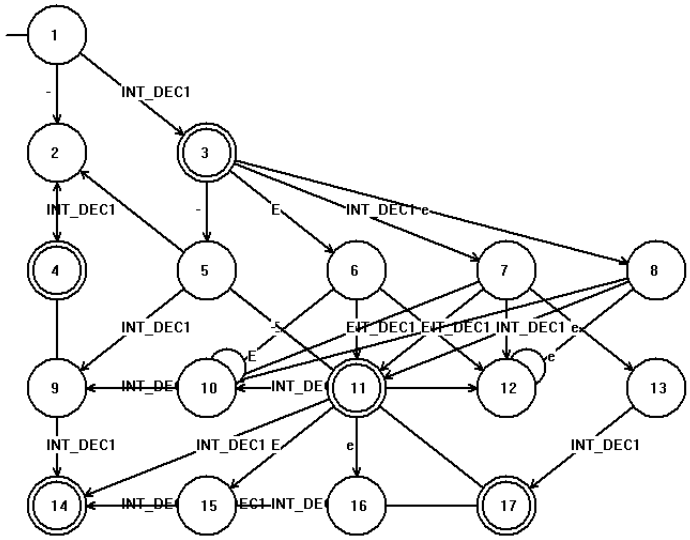
BOOL_LIT1= true | false



图表 8 布尔整型变量

FLOAT_LIT1 = (INT_DEC1 . (INT_DEC1)? ((e | E) (+ | -)? INT_DEC1)?) | (INT_DEC1 (e | E) (+ | -)? INT_DEC1)

FLOAT_LIT1 (DFA)



图表 9 浮点数变量