1810 Einsendeaufgabe KE 01

Gustavo Nunes Martins

September 30, 2018

Aufgabe 1

a-1

Falsch. Laut "Hopcroft: Introduction to Automata Theory, Languages and Computation 2er Ausgabe, Seite 30":

$$\Sigma^* = \Sigma^0 \bigcup \Sigma^1 \bigcup \Sigma^2 \bigcup ...$$

Es gilt deshalb $\Sigma^* = \{\varepsilon\} = \Sigma^0$ nur wenn $\Sigma^1 \bigcup \Sigma^2 \bigcup ... = \emptyset \Rightarrow \Sigma^1 = \Sigma = \emptyset$, also nur wenn die Alphabet eine leere Menge ist.

Das widerspricht die Definition von Alphabet (gleiches Buch, Seite 28), wobei eine Alphabet keine leere Menge sein darf.

a-2

Falsch, denn jeder regulären Ausdruck hat eine entsprechende endlicher Automat (diese Prozess kann automatisch durchgeführt werden durch den Thompson Algorithmus)

b

TODO

c

$L = \{w \in \{0,1\}^* |$ die Mindestlänge von w ist 3 und endet nicht mit 111$\}$

d

Der Minimalfall trifft zu wenn alle Symbolen eines Wortes **gleich** sind (die String ist der form a...a). Dann gilt es:

- Teilworter Länge 1: {a} (1 Teilwort)
- Teilworter Länge 2: {aa} (1 Teilwort)

- Teilworter Länge 3: { aaa} (1 Teilwort)
- Teilworter Länge n: { aaa...} (1 Teilwort)
- Alle Teilworter von lange 1 bis n: {a,aa,aaa,aaa...}. (n Teilwörter)
- Total: **n** distinkte Teilwörter für das Minimalfall

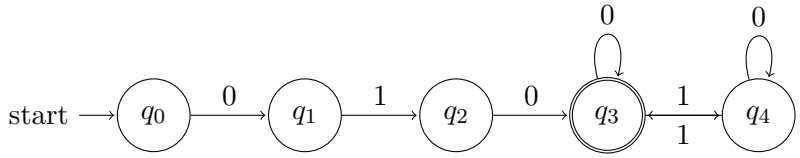
Der Minimalfall trifft zu wenn alle Symbolen eines Wortes **anders** sind (die String ist der Form abcdef...). Dan gilt es:

- Teilworter Länge 1: {a,b,c,d,e,f,...} (n Teilwörter)
- Teilworter Länge 2: {ab,bc,cd,ef,f...,...} (n-1 Teilwörter)
- Teilworter Länge 3: { abc,bcd,cde,def,ef...,f...,...} (n-1 Teilwörter)
- Teilworter Länge n: { abcdef.....} (1 Teilwort)
- Alle Teilworter von lange 1 bis n: n+(n-1)+(n-2)+(n-3)+...+1=(n+1)*n/2.
- Total: **(n+1)*n/2** distinkte Teilwörter für das Maximalfall

e

TODO

f



Aufgabe 2

Listing 1: Lexer Code für das REAL token

```
#define TRUE 1

int gettoken(){
int c;
state = 0; start_state=0;

while (TRUE){
switch(state){
case 11:
```

```
        c = nextchar();
        if isdigit(c)           state=12;
        else if issign(c)       state=13;
        else                    state=next_diagram();
        break;

case 12:
        c = nextchar();
        if c=='.'               state=14;
        else if isdigit(c)      state=12;
        else                    state=next_diagram();
        break;

case 13:
        c=nextchar();
        if isdigit(c)           state=12;
        else                    state=next_diagram();
        break;

case 14:
        c=nextchar();
        if isdigit(c)           state=15;
        else if c=='E'          state=17;
        else                    state=16;
        break;

case 15:
        c=nextchar();
        if isdigit(c)           state=15;
        else if c=='E'          state=17;
        else                    state=16;
        break;

case 16:
        return REAL;
        break;

case 17:
        c=nextchar();
        if isdigit(c)           state=18;
        else if issign(c)       state=19;
        else                    state=next_diagram();
        break;
```

```
case 18;
        c=nextchar(c)
        if isdigit(c)           state=18;
        else                    state=16;
        break;

case 19:
        c=nextchar();
        if isdigit(c)           state=18;
        else                    state=next_diagram();
}
}
}
```

Aufgabe 3

Anmerkungen:

- Das lesen von negativ-Nummern lauft nicht nur durch den Lexer, aber auch durch den Parser. Das erleichtert die Entscheidung von ”-” als negativ-Nummern oder als Subtraktion.
- Punkte (z.B. (3,2) oder (p,5)) und Funktionen ((v, f(v)) sind alle durch Tupeln repräsentiert.
- Relative Bindungskräfte zwischen Multiplikation, Division, Summierung und Subtraktion sind durch %token Vorrang verwirklicht:

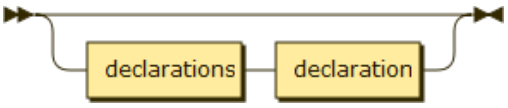
  ```
  %left '+' '-'
  %left '*' '/' "mod"
  ```

- Funktionen sind entweder in Prefix- (setcolor, arc, plot usw) oder Infixform (+, -, mod usw). Alle prefix Funktionen sind durch eine IDENTIFIER RIGHTPARENTHESIS identifiziert (die Programm kann sehr einfach mit extra Funktionen erweitert werden). Infix-Funktionen sind einzeln implementiert
- Der Bison und FLex Code fürs Lexer und Parser ist als zip Datei geliefert.

program

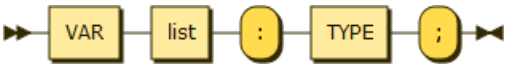

```
program  ::= PICTURE VAL_STRING declarations START commands END
```

declarations



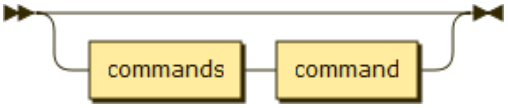

```
declarations ::= empty
| declarations declaration
```

declaration



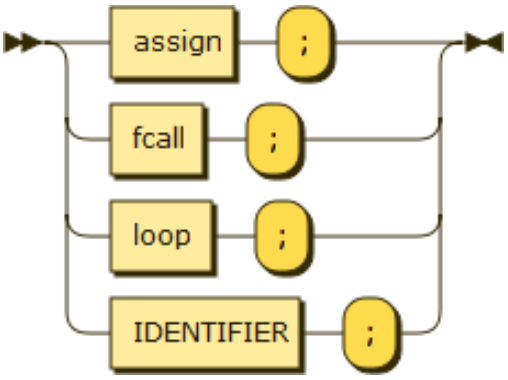

```
declaration ::= VAR list ':' TYPE ';'
```

commands



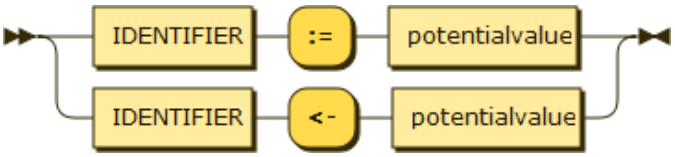

```
commands ::= empty
| commands command
```

command



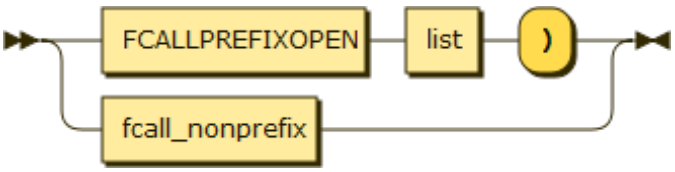

```
command  ::= assign ';'
| fcall ';'
| loop ';'
| IDENTIFIER ';'
```

assign



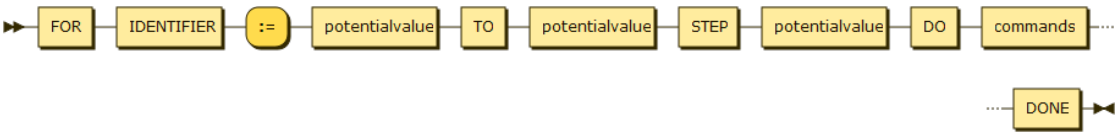

```
assign   ::= IDENTIFIER ':=' potentialvalue
| IDENTIFIER '<-' potentialvalue
```

fcall



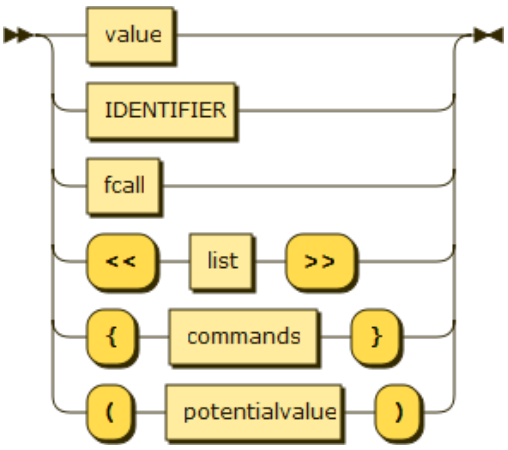

```
fcall    ::= FCALLPREFIXOPEN list ')'
| fcall_nonprefix
```

loop




```
loop     ::= FOR IDENTIFIER ':=' potentialvalue TO potentialvalue STEP potentialvalue DO
```

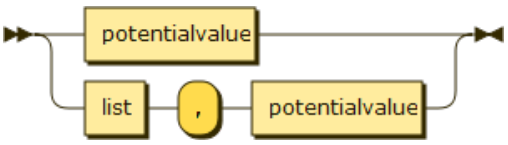
potentialvalue




```
potentialvalue
::= value
```

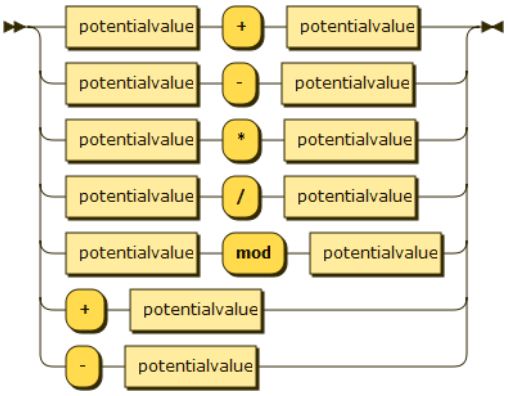
```
| IDENTIFIER
| fcall
| '<<' list '>>'
| '{' commands '}'
| '(' potentialvalue ')'
```

list



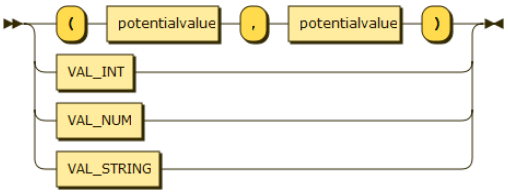

```
list     ::= potentialvalue
| list ',' potentialvalue
```

fcall_nonprefix




```
fcall_nonprefix
::= potentialvalue '+' potentialvalue
| potentialvalue '-' potentialvalue
| potentialvalue '*' potentialvalue
| potentialvalue '/' potentialvalue
| potentialvalue 'mod' potentialvalue
| '+' potentialvalue
| '-' potentialvalue
```

value



```
value    ::= '(' potentialvalue ',' potentialvalue ')'
| VAL_INT
| VAL_NUM
| VAL_STRING
```