Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação

SCC0205 – Teoria da Computação e Linguagens Formais

Trabalho 1 – Analisador Léxico e Sintático para Linguagem AWK

Elias Italiano Rodrigues – 7987251
Gabriel Tessaroli Giancristofaro – 4321350
Paulo Augusto de Godoy Patire – 7987060

São Carlos, 2 de outubro de 2014

Sumário

1 Parte 1

1.1 A Linguagem AWK

AWK é uma linguagem de programação interpretada para processamento de texto comumente usada para extração de dados em documentos estruturados em *registro* e *campo*.

Um registro é qualquer quantidade de informação que represente uma entidade e um campo é uma parte constituinte dessa informação. Por exemplo: um arquivo de texto onde cada linha contenha nomes completos de alunos (registro), e os nomes e sobrenomes (campos) estão separados por espaço. Ou ainda, mais comum, um arquivo no formato *.cvs* (Comma-separated values).

Com AWK é possível manipular tais tipos de arquivos para gerar uma nova apresentação ou fazer alterações sistemáticas nos dados.

1.2 A Notação BNF

A notação BNF foi usada para escrever a primeira gramática da linguagem. Foram definidos os conjuntos dos terminais V_t e dos não-terminais V_n, assim como o conjunto de regras P da gramática e o não-terminal inicial <program>. Nessa notação foram usadas em geral as recursões à direita para definir as possíveis cadeias.

1.3 Definição Formal da Gramática em Notação BNF

```
G = (V_n, V_t, P, <program>)

V_n = {
    <program>, <instruction>, <pattern>, <action>, <statement>,
    <command>, <if>, <while>, <do-while>, <for>, <for-in>,
    <delete>, <exit>, <return>, <print>, <getline>, <expr-list>,
    <variable>, <id>, <index>, <lvalue>, <expr>  , <expr01>,
    <expr02>, <expr03>, <expr04>, <expr05>, <expr06>, <expr07>,
    <expr08>, <expr09>, <expr10>, <expr11>, <expr12>, <expr13>,
    <expr14>, <assignment>, <comparison>, <arithmetic>, <unary>,
    <unary-op>, <constant>, <integer>, <float>, <string>,
    <sentence>, <char>, <alpha-numeric>, <number>, <digit>,
    <letter>, <lowercase>, <uppercase>, <symbol>, <sign>,
    <endline>
}

V_t = {
     , !, ", #, $, %, &, ', (, ), *, +, ,, -, ., /, 0, 1,
    2, 3, 4, 5, 6, 7, 8, 9, :, ;, <, =, >, ?, @, A, B, C, D, E,
    F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y,
    Z, [, \, ], ^, _, `, a, b, c, d, e, f, g, h, i, j, k, l, m,
    n, o, p, q, r, s, t, u, v, w, x, y, z, {, |, }, ~,
    BEGIN, END, if, else, do, while, for, in, delete,
    break, continue, next, exit, return, print, getline,
    ARGC, ARGIND, ARGV, BINMODE, CONVFMT, ENVIRON, ERRNO,
```

```
    FIELDWIDTHS, FILENAME, FNR, FS, IGNORECASE, LINT, NF,
    NR, OFMT, OFS, ORS, PROCINFO, RS, RT, RSTART,
    RLENGTH, SUBSEP, TEXTDOMAIN
}

P = {

01.    <program> ::= <instruction> <program>
                   | <instruction>

02.    <instruction> ::= <pattern> { <action> }
                       | { <action> }

03.    <pattern> ::= BEGIN
                   | END
                   | <expr>

04.    <action> ::= <statement> <action>
                  | <statement>

05.    <statement> ::= { <statement> }
                     | <expr> <statement>
                     | <command> <statement>
                     | <expr>
                     | <command>

06.    <command> ::= <if>
                   | <while>
                   | <do-while> <endline>
                   | <for>
                   | <for-in>
                   | <delete> <endline>
                   | break <endline>
                   | continue <endline>
                   | next <endline>
                   | <exit> <endline>
                   | <return> <endline>
                   | <print> <endline>
                   | <getline> <endline>

07.    <if> ::= if ( <expr> ) <statement>
              | if ( <expr> ) <statement> else <statement>

08.    <while> ::= while ( <expr> ) <endline>
                 | while ( <expr> ) <statement>

09.    <do-while> ::= do <statement> while ( <expr> )

10.    <for> ::= for (  ;  ; ) <statement>
              | for ( <expr> ;  ;  ) <statement>
              | for (  ; <expr> ;  ) <statement>
              | for (  ;  ; <expr> ) <statement>
              | for ( <expr> ; <expr> ;  ) <statement>
```

```
            | for ( <expr> ;  ; <expr> ) <statement>
            | for (  ; <expr> ; <expr> ) <statement>
            | for ( <expr> ; <expr> ; <expr> ) <statement>

11.   <for-in> ::= for ( <variable> in <id> ) <statement>

12.   <delete> ::= delete <id> [ <index> ]

13.   <exit> ::= exit | exit <expr>

14.   <return> ::= return | return <expr>

15.   <print> ::= print <expr-list>
                | print

16.   <getline> ::= getline <variable>
                  | getline

17.   <expr-list> ::= <expr> , <expr-list>
                    | <command> , <expr-list>
                    | <variable> , <expr-list>
                    | <constant> , <expr-list>
                    | <expr>
                    | <command>
                    | <variable>
                    | <constant>

18.   <variable> ::= <id>
                   | $<expr>
                   | ARGC | ARGIND | ARGV | BINMODE | CONVFMT
                   | ENVIRON | ERRNO | FIELDWIDTHS | FILENAME
                   | FNR | FS | IGNORECASE | LINT | NF | NR
                   | OFMT | OFS | ORS | PROCINFO | RS| RT
                   | RSTART | RLENGTH | SUBSEP | TEXTDOMAIN

19.   <id> ::= _ <id>
             | <letter> <id>
             | _
             | <alpha-numeric>

20.   <index> ::= <number> | <string>

21.   <lvalue> ::= <variable>
                 | <variable> [ <index> ]

22.   <expr>   ::= <lvalue> <assignment> <expr01> | <expr01>
23.   <expr01> ::= <expr02> ? <expr01> : <expr01> | <expr02>
24.   <expr02> ::= <expr03> || <expr02>           | <expr03>
25.   <expr03> ::= <expr04> && <expr03>           | <expr04>
26.   <expr04> ::= ( <index> ) in <id>            | <expr05>
27.   <expr05> ::= <expr06> in <id>               | <expr06>
28.   <expr06> ::= <expr07> <comparison> <expr06> | <expr07>
29.   <expr07> ::= <expr08> <expr07>              | <expr08>
```

```
30.    <expr08> ::= <expr09> <arithmetic> <expr08> | <expr09>
31.    <expr09> ::= <unary> <expr10>               | <expr10>
32.    <expr10> ::= <expr11> ^ <expr10>            | <expr11>
33.    <expr11> ::= <lvalue> <unary-op>            | <expr12>
34.    <expr12> ::= <unary-op> <lvalue>            | <expr13>
35.    <expr13> ::= ( <expr> )                     | <expr14>
36.    <expr14> ::= <variable> | <constant>

37.    <assignment> ::=   = | -= | += | /= | *= | %= | ^=
38.    <comparison> ::=  >= | >  | == | != | <= | <
39.    <arithmetic> ::=   - | +  | %  |  / | *
40.         <unary> ::=   - | +  | !
41.      <unary-op> ::=  -- | ++

42.    <constant> ::= <integer> | <float> | <string>
43.    <integer> ::= <number> | <sign> <number>

44.    <float> ::= <sign> <number> . <number>
                 | <number> . <number>

45.    <string> ::= " <sentence> "

46.    <sentence> ::= <char> <sentence>
                    | <char>

47.    <char> ::= <alpha-numeric> | <symbol>

48.    <alpha-numeric> ::= <letter> <alpha-numeric>
                         | <digit>  <alpha-numeric>
                         | <letter>
                         | <digit>

49.    <number> ::= <digit> <number>
                  | <digit>

50.    <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
51.    <letter> ::= <lowercase> | <uppercase>

52.    <lowercase> ::= a | b | c | d | e | f | g | h | i | j
                     | k | l | m | n | o | p | q | r | s | t
                     | u | v | w | x | y | z

53.    <uppercase> ::= A | B | C | D | E | F | G | H | I | J
                     | K | L | M | N | O | P | Q | R | S | T
                     | U | V | W | X | Y | Z

54.    <symbol> ::=   | ! | " | # | $ | % | & | ’ | ( | ) | *
                  | + | , | - | . | / | : | ; | < | = | > | ?
                  | @ | [ | \ | ] | ^ | _ | ‘ | { | | | } | ~

55.    <sign> ::= + | -
56.    <endline> ::= ;
}
```

2 Parte 2

Nesta parte do trabalho foram feitas as derivações de três programas em linguagem AWK usando a gramática BNF definida na Parte 1.

2.1 Derivação do programa hello-world.awk

Um programa que usa a pattern BEGIN para imprimir uma mensagem de "Hello World!"

```
BEGIN { print "Hello World!"; }
```

```
<program>
->01 <instruction>
->02 <pattern> { <action> }
->03 BEGIN { <action> }
->04 BEGIN { <statement> }
->05 BEGIN { <command> }
->06 BEGIN { <print> <endline> }
->15 BEGIN { print <expr-list> ; }
->17 BEGIN { print <constant> ; }
->42 BEGIN { print <string> ; }
->45 BEGIN { print " <sentence> " ; }
->46 BEGIN { print " <char> <sentence> " ; }
->47 BEGIN { print " <alpha-numeric> <sentence> " ; }
->48 BEGIN { print " <letter> <sentence> " ; }
->51 BEGIN { print " <uppercase> <sentence> " ; }
->53 BEGIN { print "H <sentence> " ; }
->46 BEGIN { print "H <char> <sentence> " ; }
...
->54 BEGIN { print "Hello World!"; }
```

2.2 Derivação do programa linha-grande.awk

Um programa que considera pequeno um registro que contenha menos de oito campos e grande um registro que contenha oito ou mais campos.

```
{
    if (NF < 8)
        print "Linha pequena:", $0;
    else
        print "Linha grande:", $0;
}
```

```
<program>
->01 <instruction>
->02 { <action> }
->04 { <statement> }
->05 { <command> }
->06 { <if> }
->07 { if ( <expr> ) <statement> else <statement> }
->22 { if ( <expr01> ) <statement> else <statement> }
```

```
->23 { if ( <expr02> ) <statement> else <statement> }
->24 { if ( <expr03> ) <statement> else <statement> }
->25 { if ( <expr04> ) <statement> else <statement> }
->26 { if ( <expr05> ) <statement> else <statement> }
->27 { if ( <expr06> ) <statement> else <statement> }
->28 { if ( <expr07> <comparison> <expr06> ) <statement> else <
   statement> }
->29 { if ( <expr08> <comparison> <expr06> ) <statement> else <
   statement> }
->30 { if ( <expr09> <comparison> <expr06> ) <statement> else <
   statement> }
->31 { if ( <expr10> <comparison> <expr06> ) <statement> else <
   statement> }
->32 { if ( <expr11> <comparison> <expr06> ) <statement> else <
   statement> }
->33 { if ( <expr12> <comparison> <expr06> ) <statement> else <
   statement> }
->34 { if ( <expr13> <comparison> <expr06> ) <statement> else <
   statement> }
->35 { if ( <expr14> <comparison> <expr06> ) <statement> else <
   statement> }
->36 { if ( <variable> <comparison> <expr06> ) <statement> else <
   statement> }
->28 { if ( <variable> <comparison> <expr07> ) <statement> else <
   statement> }
->29 { if ( <variable> <comparison> <expr08> ) <statement> else <
   statement> }
->30 { if ( <variable> <comparison> <expr09> ) <statement> else <
   statement> }
->31 { if ( <variable> <comparison> <expr10> ) <statement> else <
   statement> }
->32 { if ( <variable> <comparison> <expr11> ) <statement> else <
   statement> }
->33 { if ( <variable> <comparison> <expr12> ) <statement> else <
   statement> }
->34 { if ( <variable> <comparison> <expr13> ) <statement> else <
   statement> }
->35 { if ( <variable> <comparison> <expr14> ) <statement> else <
   statement> }
->36 { if ( <variable> <comparison> <constant> ) <statement> else <
   statement> }
->18 { if ( NF <comparison> <constant> ) <statement> else <statement>
   }
->42 { if ( NF <comparison> <integer> ) <statement> else <statement> }
->43 { if ( NF <comparison> <number> ) <statement> else <statement> }
->49 { if ( NF <comparison> <digit> ) <statement> else <statement> }
->50 { if ( NF <comparison> 8 ) <statement> else <statement> }
->38 { if ( NF < 8 ) <statement> else <statement> }
->05 { if ( NF < 8 ) <command> else <statement> }
->05 { if ( NF < 8 ) <command> else <command> }
->06 { if ( NF < 8 ) <print> <endline> else <command> }
->06 { if ( NF < 8 ) <print> <endline> else <print> <endline> }
->56 { if ( NF < 8 ) <print> ; else <print> <endline> }
```

```
->56 { if ( NF < 8 ) <print> ; else <print> ; }
->15 { if ( NF < 8 ) print <expr-list> ; else <print> ; }
->15 { if ( NF < 8 ) print <expr-list> ; else print <expr-list> ; }
->17 { if ( NF < 8 ) print <constant>, <expr-list> ; else print <expr-
   list> ; }
->17 { if ( NF < 8 ) print <constant>, <expr-list> ; else print <
   constant>, <expr-list> ; }
->17 { if ( NF < 8 ) print <constant>, <variable> ; else print <
   constant>, <expr-list> ; }
->17 { if ( NF < 8 ) print <constant>, <variable> ; else print <
   constant>, <variable> ; }
->42 { if ( NF < 8 ) print <string>, <variable> ; else print <constant
   >, <variable> ; }
->42 { if ( NF < 8 ) print <string>, <variable> ; else print <string>,
    <variable> ; }
->45 { if ( NF < 8 ) print " <sentence> ", <variable> ; else print <
   string>, <variable> ; }
->45 { if ( NF < 8 ) print " <sentence> ", <variable> ; else print " <
   sentence> ", <variable> ; }
->46 { if ( NF < 8 ) print " <char> <sentence> ", <variable> ; else
   print " <sentence> ", <variable> ; }
->47 { if ( NF < 8 ) print " <alpha-numeric> <sentence> ", <variable>
   ; else print " <sentence> ", <variable> ; }
->48 { if ( NF < 8 ) print " <letter> <sentence> ", <variable> ; else
   print " <sentence> ", <variable> ; }
->51 { if ( NF < 8 ) print " <uppercase> <sentence> ", <variable> ;
   else print " <sentence> ", <variable> ; }
->53 { if ( NF < 8 ) print "L <sentence> ", <variable> ; else print "
   <sentence> ", <variable> ; }
...
->54 { if ( NF < 8 ) print "Linha pequena:", <variable> ; else print "
    <sentence> ", <variable> ; }
->46 { if ( NF < 8 ) print "Linha pequena:", <variable> ; else print "
    <char> <sentence> ", <variable> ; }
...
->54 { if ( NF < 8 ) print "Linha pequena:", <variable> ; else print "
   Linha grande:", <variable> ; }
->18 { if ( NF < 8 ) print "Linha pequena:", $<expr> ; else print "
   Linha grande:", <variable> ; }
->18 { if ( NF < 8 ) print "Linha pequena:", $<expr> ; else print "
   Linha grande:", $<expr> ; }
...
->50 { if ( NF < 8 ) print "Linha pequena:", $0 ; else print "Linha
   grande:", $0 ; }
```

2.3 Derivação do programa inverte-nomes.awk

Um programa simples que inverte o primeiro e o segundo campo de cada registro de um arquivo.

```
{
    print $2, $1;
}
```

```
<program>
->01 <instruction>
->02 { <action> }
->04 { <statement> }
->05 { <command> }
->06 { <print> <endline> }
->15 { print <expr-list> <endline> }
->17 { print <variable>, <expr-list> <endline> }
->17 { print <variable>, <variable> <endline> }
->18 { print $<expr>, <variable> <endline> }
->22 { print $<expr01>, <variable> <endline> }
->23 { print $<expr02>, <variable> <endline> }
->24 { print $<expr03>, <variable> <endline> }
->25 { print $<expr04>, <variable> <endline> }
->26 { print $<expr05>, <variable> <endline> }
->27 { print $<expr06>, <variable> <endline> }
->28 { print $<expr07>, <variable> <endline> }
->29 { print $<expr08>, <variable> <endline> }
->30 { print $<expr09>, <variable> <endline> }
->31 { print $<expr10>, <variable> <endline> }
->32 { print $<expr11>, <variable> <endline> }
->33 { print $<expr12>, <variable> <endline> }
->34 { print $<expr13>, <variable> <endline> }
->35 { print $<expr14>, <variable> <endline> }
->36 { print $<constant>, <variable> <endline> }
->18 { print $<constant>, $<expr> <endline> }
->22 { print $<constant>, $<expr01> <endline> }
->23 { print $<constant>, $<expr02> <endline> }
->24 { print $<constant>, $<expr03> <endline> }
->25 { print $<constant>, $<expr04> <endline> }
->26 { print $<constant>, $<expr05> <endline> }
->27 { print $<constant>, $<expr06> <endline> }
->28 { print $<constant>, $<expr07> <endline> }
->29 { print $<constant>, $<expr08> <endline> }
->30 { print $<constant>, $<expr09> <endline> }
->31 { print $<constant>, $<expr10> <endline> }
->32 { print $<constant>, $<expr11> <endline> }
->33 { print $<constant>, $<expr12> <endline> }
->34 { print $<constant>, $<expr13> <endline> }
->35 { print $<constant>, $<expr14> <endline> }
->36 { print $<constant>, $<constant> <endline> }
->56 { print $<constant>, $<constant> ; }
->42 { print $<integer>, $<constant> ; }
->42 { print $<integer>, $<integer> ; }
->43 { print $<number>, $<integer> ; }
```

```
->43 { print $<number>, $<number> ; }
->49 { print $<digit>, $<number> ; }
->49 { print $<digit>, $<digit> ; }
->50 { print $2, $<digit> ; }
->50 { print $2, $1 ; }
```

3 Parte 3

3.1 Notação EBNF

Foi usada notação de Wirth em que `[ ]` representa opcionalidade, { } represente zero ou mais repetições, `|` representa alternativa, `=` representa definição e `( )` agrupamento.

3.2 Conversão para notação EBNF

```
G = (V_n, V_t, P, program)

V_n = {
    program, instruction, pattern, action, statement, command,
    if, while, do-while, for, for-in, delete, exit, return,
    print, getline, expr-list, variable, id, index, lvalue, expr,
    expr01, expr02, expr03, expr04, expr05, expr06, expr07,
    expr08, expr09, expr10, expr11, expr12, assign, comparison,
    arithmetic , unary-op, constant, integer, float, string,
    sentence, char, alpha-numeric, number, digit, letter,
    lowercase, uppercase, symbol, sign, endline
}

V_t = {
    " ", "!", """, "#", "$", "%", "&", "'", "(", ")", "*",
    "+", ",", "-", ".", "/", "0", "1", "2", "3", "4", "5", "6",
    "7", "8", "9", ":", ";", "<", "=", ">", "?", "@", "A", "B",
    "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N",
    "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z",
    "[", "\", "]", "^", "_", "`", "a", "b", "c", "d", "e", "f",
    "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r",
    "s", "t", "u", "v", "w", "x", "y", "z", "{", "|", "}", "~",
    "BEGIN", "END", "if", "else", "do", "while",
    "for", "in", "delete", "break", "continue", "next", "exit",
    "return", "print", "getline", "ARGC", "ARGIND", "ARGV",
    "BINMODE", "CONVFMT", "ENVIRON", "ERRNO", "FIELDWIDTHS",
    "FILENAME", "FNR", "FS", "IGNORECASE", "LINT", "NF", "NR",
    "OFMT", "OFS", "ORS", "PROCINFO", "RS", "RT", "RSTART",
    "RLENGTH", "SUBSEP", "TEXTDOMAIN"
}

P = {

01.    program = {instruction}
```

```
02.    instruction = [pattern] "{" action "}"

03.    pattern = [BEGIN | END | expr]

04.    action = statement {statement}

05.    statement = "{" {(expr | command)} "}"
                 | [(expr | command)]

06.    command = if
                | while
                | do-while
                | for
                | for-in
                | delete
                | "break" | "continue" | "next"
                | exit
                | return
                | print
                | getline

07.    if = "if" "(" expr ")" statement ["else" statement]

08.    while = "while" "(" expr ")" statement

09.    do-while = "do" statement "while" "(" expr ")" endline

10.    for = "for" "(" [expr]";" [expr] ";" [expr] ")" statement

11.    for-in = "for" "(" variable "in" id ")" statement

12.    delete = "delete" id "[" index "]" endline

13.    exit = "exit" [expr] endline

14.    return = "return" [expr] endline

15.    print = "print" [expr-list] endline

16.    getline = "getline" [variable] endline

17.    expr-list = (expr | command | variable)
                   {"," (expr | command | variable)}

18.    variable = id
                | "$"expr
                | "ARGC" | "ARGIND" | "ARGV" | "BINMODE"
                | "CONVFMT" | "ENVIRON" | "ERRNO"
                | "FIELDWIDTHS" | "FILENAME" | "FNR" | "FS"
                | "IGNORECASE"  | "LINT" | "NF" | "NR" | "OFMT"
                | "OFS" | "ORS" | "PROCINFO" | "RS" | "RT"
                | "RSTART" | "RLENGTH" | "SUBSEP" | "TEXTDOMAIN"
```

```
19.   id = ("_" | letter) {"_" | alpha-numeric}

20.   index = number | string

21.   lvalue = variable [ "[" index "]" ]

22.   expr   = lvalue assign expr                | expr01
      expr01 = expr02 "?"  expr01 ":" expr01     | expr02
      expr02 = expr03 ("||" | "&&") expr02       | expr03
      expr03 = "(" index ")" "in" id             | expr04
      expr04 = expr05 "in" id                    | expr05
      expr05 = expr06 comparison expr05          | expr06
      expr06 = expr07 expr06                     | expr07
      expr07 = expr08 arithmetic expr07          | expr08
      expr08 = ("-" | "+"  | "!") expr09         | expr09
      expr09 = expr10 "^" expr09                 | expr10
      expr10 = lvalue unary-op | unary-op lvalue | expr11
      expr11 = "(" expr ")"                      | expr12
      expr12 =  variable | constant

23.   assign = "=" | "-=" | "+=" | "/=" | "*=" | "%=" | "^="

24.   comparison = ">=" | ">" | "==" | "!=" | "<=" | "<"

25.   arithmetic  = "-" | "+"  | "%"  |  "/" | "*"

26.   unary-op =  "--" | "++"

27.   constant = integer | float | string

28.   integer = [sign] number

29.   float = [sign] [number] "." number

30.   string = """ [sentence] """

31.   sentence = char {char}

32.   char = alpha-numeric | symbol

33.   alpha-numeric = (letter | digit) {letter | digit}

34.   number = digit {digit}

35.   digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7"
            | "8" | "9"

36.   letter = lowercase | uppercase

37.   lowercase = "a" | "b" | "c" | "d" | "e" | "f" | "g"
                | "h" | "i" | "j" | "k" | "l" | "m" | "n"
                | "o" | "p" | "q" | "r" | "s" | "t"
```

```
                | "u" | "v" | "w" | "x" | "y" | "z"

38.     uppercase = "A" | "B" | "C" | "D" | "E" | "F" | "G"
                  | "H" | "I" | "J" | "K" | "L" | "M" | "N"
                  | "O" | "P" | "Q" | "R" | "S" | "T"
                  | "U" | "V" | "W" | "X" | "Y" | "Z"

39.     symbol = " " | "!" | """ | "#" | "$" | "%" | "&" | "'"
               | "(" | ")" | "*" | "+" | "," | "-" | "." | "/"
               | ":" | ";" | "<" | "=" | ">" | "?" | "@" | "["
               | "\" | "]" | "^" | "_" | "`" | "{" | "|" | "}"
               | "~"

40.     sign = "+" | "-"

41.     endline = ";"
}
```