=τρυε =βλυε =ςψαν

Εργασία Μεταγλωττιστών
Τμήμα Α2

Ομάδα 15

29 Μαρτίου 2021

Ομάδα 15

Αναλυτικά τα μέλη:

Διονύσης Νικολόπουλος AM : 18390126

Θανάσης Αναγνωστόπουλος AM : 18390043

Αριστείδης Αναγνωστόπουλος AM : 16124

Σπυρίδων Φλώρος AM : 141084

Αναλυτικά οι ρόλοι:

Γενικός Συντονιστής: Διονύσης Νικολόπουλος

Υπεύθυνος Τμήματος Εργασίας Α2: Διονύσης Νικολόπουλος

Usernames στο Github

Διονύσης Νικολόπουλος : jiud

Θανάσης Αναγνωστόπουλος : ThanasisAnagno

Αριστείδης Αναγνωστόπουλος : Aris-Anag

Σπυρίδων Φλώρος : spirosfl

Η εργασία αυτή πραγματοποιήθηκε με χρήση LaTeX

Περιεχόμενα

1 Εισαγωγή

Τι πραγματεύεται η εργασία

Στην εργαστηριακή εργασία αυτή ασχοληθήκαμε με την δημιουργία ενός ντετερμινιστικού αυτόματου πεπερασμένων καταστάσεων.

Ο σκοπός του αυτομάτου

Το αυτόματο αυτό έχει ως σκοπό να λειτουργεί σαν Λεκτικός Αναλυτής της γλώσσας Uni-C, ενός παρακλαδιού της γλώσσας C, απο το Πανεπιστήμιο Δυτικής Αττικής.

Βήματα Υλοποίησης

Πριν πραγματοποιηθεί φυσικά η κωδικοποίηση για την προσσέγγιση του προβλήματος αυτού, έγινε η περιγραφή της γλώσσας Uni-C σε μορφή BNF.

Η κωδικοποίηση του έχει γίνει καταρχήν με την βοήθεια του προγράμματος *FSM*, που επίσης είναι προγραμματισμένο πάνω στην γλώσσα C.

Ο σχεδιασμός των διαγραμμάτων έχει γίνει με το πρόγραμμα *JFLAP*.

Σελίδα στο Github

Η σελίδα μας στο Github, με την οποία διοργανώσαμε την εργασία και δουλέψαμε όλοι μαζί είναι (κάντε κλικ για το λινκ):

Uni-C-Analyser

Μπορείτε να δείτε αναλυτικά όλη μας την πρόοδο εκεί.

Εξήγηση για το τι ακολουθεί

Σε αυτό το PDF αρχικά θα παρατεθεί η περιγραφή της Uni-C σε BNF.

Έπειτα, θα παρατεθούν επίσης και τα διαγράμματα του εναίου αυτόματου που βοηθούν στην κωδικοποίηση του αλλά και στην σύλληψη νοητά του τι περιμένουμε να κάνει το αυτόματό μας, πως θα το υλοποιήσουμε τεχνικά.

Στην συνέχεια, θα ακολουθήσει ο κώδικας του αυτόματου τμηματικά με εξηγήσεις της λειτουργίας του.

Τελος, εφόσον ο κώδικας παραδωθεί και ολοκληρωμένα, θα παρατεθούν ορισμένοι ελέγχοι πραγματικής λειτουργίας του αυτόματου σε εικόνες, καταλίγοντας και σε εικόνες που αποδεικνουν ότι οι έλεγχοι έγιναν.

2 Περιγραφή σύνταξης της Uni-C σε BNF

Πριν την υλοποίηση του αυτομάτου σε εκτελέσιμο κώδικα, όπως αναφέραμε παραπάνω, έγινε η ανάλυση της σύνταξης της γλώσσας Uni-C σε μορφή BNF (Backus-Naur Form ή Μορφή Μπάκους-Ναούρ).
Οπότε, με την βοήθεια του περιβάλλοντος λεκτικού επεξεργαστή Emacs, γράψαμε τον εξής συμβολισμό.

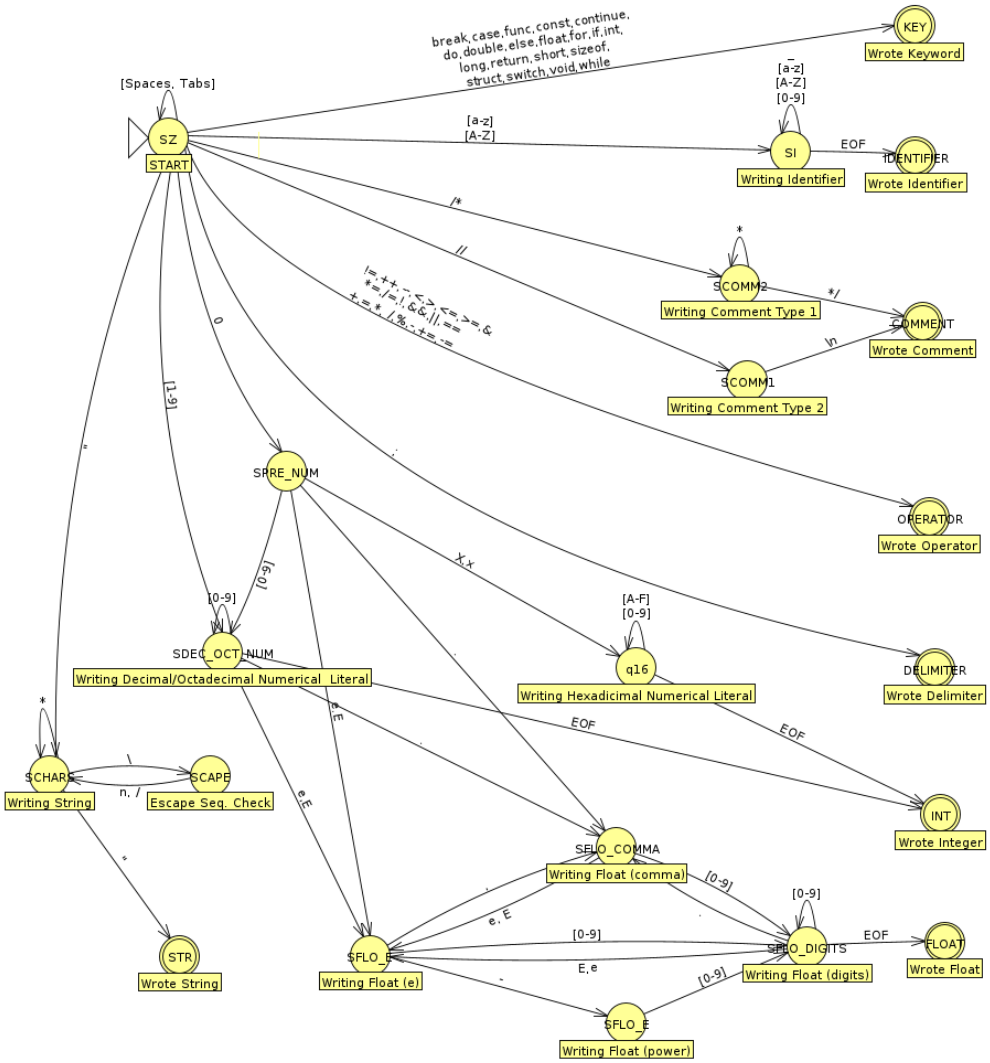
```
1  symbol         ::= "|" | " " | "!" | "#" | "$" | "%" | "&" | "(" | ")
       " | "*" | "+" | "," | "-" | "." | "/" | ":" | "\;" | ">" | "=" |
       "<" | "?" | "@" | "[" | "\" | "]" | "^" | "_" | "`" | "{" | "}" |
        "~"
2  letter         ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "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"
3  digit          ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
       "8" | "9"
4  character      ::= letter | symbol | digit
5  character1     ::= character | "'"
6  character2     ::= character | '"'
7  text1          ::= "" | character1 text1
8  text2          ::= '' | character2 text2
9  literal        ::= '"' text1 '"' | "'" text2 "'"
10 identifier     ::= character text1
11 operator       ::= "+" | "-" | "*" | "/" | "%" | "=" | "+=" | "-=" |
       "*=" | "" | "" | "--" | "=" | "=" | "&" | "==" | "!=" | "++" |
       "&&" | "||"
12 number1        ::= digit number1 | "'"
13 number2        ::= digit number2 | '"'
14 number         ::= number1 | number2
15 integer        ::= number | "0X" number | "0x" number | "0"
16 float          ::= number "." number | number "e" number | number "e"
        number "-" number
17
```

3 Διαγράμματα Ενιαίου Αυτόματου

3.1 Γενικευμένο Διάγραμμα Ενιαίου Αυτόματου

Εδώ είναι το γενικευμένο ενιαίο αυτόματο που σχεδιάσαμε:

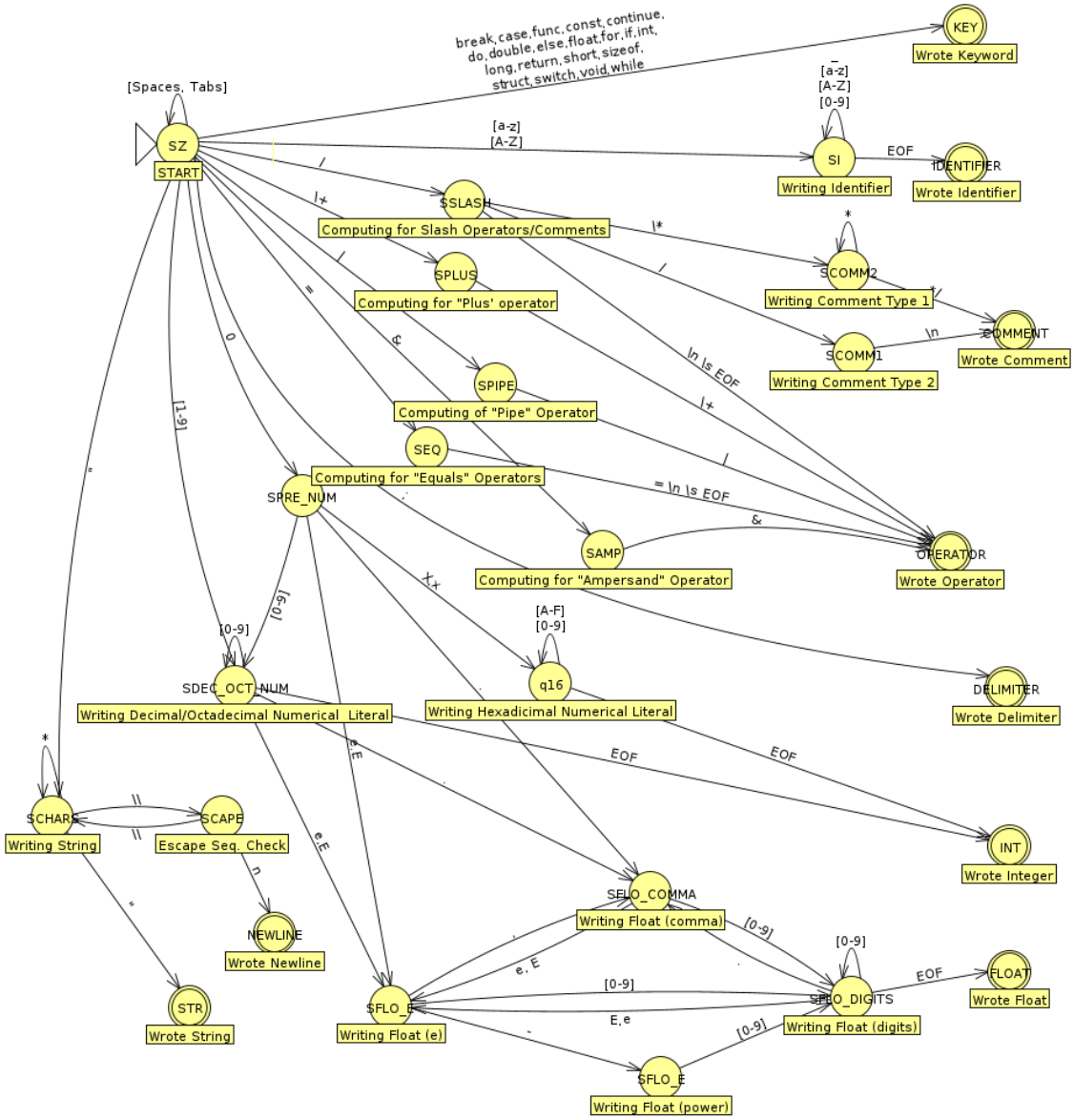
Σχήμα 1: Γενικευμένο Αυτόματο



3.2 Λεπτομερές Διάγραμμα Ενιαίου Αυτόματου

Εδώ είναι το λεπτομερές ενιαίο αυτόματο που σχεδιάσαμε:

Σχήμα 2: Λεπτομερές Αυτόματο



4 Εξήγηση Κώδικα σε FSM

Το πρώτο κομμάτι κώδικα σε FSM είναι, ασφαλώς, η αρχική μας κατάσταση:

```
START=SZ
SZ: % -> OPERATOR
    ! < > \- = \* -> SEQ
    \+ -> SPLUS
    & -> SAMP
    ; -> DELIMITER
    | -> SPIPE
    a-z A-Z _ -> SI
    \n \s -> SZ
    / -> SSLASH
    " -> SCHARS
    0 -> SPRE_NUM
    1-9 -> SDEC_OCT_NUM
    * -> BAD
    EOF -> GOOD

```

Εδώ, βλέπουμε ότι ο αναλυτής μας περιμένει τον πρώτο, ουσιαστικά χαρακτήρα απο τον χρήστη, ο οποίος θα έχει μεγάλη σημασία στο να καθορίσουμε ¨τι πάει να γράψει ο χρήστης¨.

Βλέπουμε πολλές καταστάσεις, των οποίων η φιλοσοφία εξηγείται ως εξής.

- Οι καταστάσεις που το όνομα τους αρχίζει με "S" είναι για τον αναλυτή μας οι ¨μεταβατικές' μας καταστάσεις, οι καταστάσεις στις οποίες δεν έιναι απόλυτα σίγουρο το τι θέλει να γράψει ο χρήστης, αλλά είναι σίγουροι, με βάση κανόνες και ντετερμινιστικά οι ¨δρόμοι' που μπορεί να ακολουθήσει ο χρήστης.
- Οι υπόλοιπες καταστάσεις, καταστάσεις όπως η DELIMITER που βλέπουμε εδώ, εκφράζουν επιστροφή του αναγνωριστικού (και άρα επιτυχής αναγνώριση) της λεκτικής μονάδας (πχ εδώ το token (λεκτική μονάδα) είναι DELIMETER) καθώς ο αναλυτής μας είναι σίγουρος, *με βάση κανόνες που ακολουθεί ντετερμινιστικά*, ότι η λέξη η οποία ο χρήστης εισήγαγε είναι ενός συγκεκριμένου τύπου.

Συνεχίζοντας, βλέπουμε 2 καταστάσεις που αφορούν την αναγνώριση συμβολοσειράς, στις οποίες είναι δυνατόν να μεταβεί το αυτόματο, αν ο χρήστης εισάγει σε ορισμένες καταστάσεις τον χαρακτήρα ".

Αυτό ισχύει για την αρχική κατάσταση, όπως είδαμε παραπάνω, αλλά και για τις καταστάσεις IDENTIFIER, INT, FLOAT, αφού είναι πιθανό ο χρήστης να γράφει μια συμβολοσειρά έπειτα απο τους τύπους λέξεων αυτούς.

Η κατάσταση SCHARS συγκεκριμένα είναι αυτή στην οποία δίνονται οι χαρακτήρες της συμβολοσειράς, και αφήνεται ένα ενδεχόμενο είτε να τερματισθεί η συμβολοσειρά με το να εισαχθεί ξανά ο χαρακτήρας " (και να μεταβεί στην κατάσταση STR), είτε να να εισαχθεί ο χαρακτήρας \.

Στην τελευταία περίπτωση, ο αναλυτής έρχεται στην κατάσταση SCAPE, την οποία περιμένει να εισαχθέι είτε ο χαρακτήρας n, οπότε να επιστραφεί η κατάσταση NEWLINE, είτε να εισαχθεί ο χαρακτήρας \ξανά, οπότε να επιστρέψει στην κατάσταση SCHARS, είτε να κάνει λάθος και να εισαχθεί οποιοσδήποτε άλλος χαρακτήρας, οπότε να παέι στην κατάσταση BAD.

Στην κατάσταση NEWLINE, οποιοσδήποτε χαρακτήρας μας πάει πίσω στην κατάσταση SCHARS, ενώ ο χαρακτήρας " στην κατάσταση STR

```
1  SCHARS: * -> SCHARS
2          " -> STR
3          \\ -> SCAPE
4
5  SCAPE:  n -> NEWLINE
6          \\ -> SCHARS
7          * -> BAD
8
9  NEWLINE: " -> STR
10           * -> SCHARS
11
12 STR:        \s \n -> SZ
13             ;     -> DELIMITER
14             EOF   -> GOOD
15
```

Με παρόμοια λογική βλέπουμε διάφορους χαρακτήρες που είναι *δυνητικά* τελεστές (ο πρώτος είναι δυνητικά και σχόλιο).

```
SSLASH: / -> SCOMM1
        = \n \s -> OPERATOR
        \* -> SCOMM2
         * -> BAD

SEQ:     = \n \s -> OPERATOR
         * -> BAD

SPLUS:   = \+  -> OPERATOR
         * -> BAD

SAMP:    & \s  -> OPERATOR
         * -> BAD

SPIPE:   |     -> OPERATOR
         * -> BAD

```

Επίσης, εδώ βλέπουμε, με παρόμοιο συλλογισμό με τις καταστάσεις για τις συμβολοσειρές, και την επεξεργασία για τα σχόλια (με τελική κατάσταση την κατάσταση COMMENT)

```
SCOMM1:     * -> SCOMM1
           \n -> SZ

SCOMM2:    \* -> SCOMM21
           * -> SCOMM2

SCOMM21:    / -> COMMENT
            * -> SCOMM2

COMMENT:    \n \s -> SZ
            EOF -> GOOD

```

Παρατηρούμε και την κατάσταση SI, η οποία δέχεται τους επιτρεπτούς χαρακτήρες ως κομμάτια της πιθανής λέξης ενός identifier, που με τον χαρακτηρα ;, με κενό, ή με τέλος αρχείου γίνεται η μεταβολή του αναλυτή στην κατάσταση IDENTIFIER.

```
SI:      a-z A-Z 0-9 _ -> SI
         ;             -> IDENTIFIER
         \s            -> IDENTIFIER
```

```
4           EOF             -> IDENTIFIER
5           \+ \- =         -> IDENTIFIER
6           * -> BAD
7
```

Εδώ βλέπουμε τις ¨τελικές' καταστάσεις στις οποίες φτάνει ο αναλυτής οταν κρίνει μια λέξη να πληρεί τις προϋποθέσεις για να χαρακτηρισθεί ως μια συγκεκριμένη λεκτική μονάδα.
Παρατηρούμε ότι, με βάση την γλώσσα Uni-C, οι λεκτικές μονάδες είναι δυνατό να ακολουθηθούν απο συγκεκριμένες λεκτικές μονάδες (παράδειγμα οι τελεστές μονάχα από ένα νούμερο ή identifier, για αυτό οι μοναδικές επιτρεπτές καταστάσεις που μεταβαίνει ο αναλυτής μετά απο την κατάσταση OPERATOR είναι η ενδιάμεση κατάσταση έιτε αριθμού έιτε identifier)

```
1  OPERATOR:  a-z A-Z _ -> SI
2             0 -> SPRE_NUM
3             1-9 -> SDEC_OCT_NUM
4             " -> SCHARS
5             * -> BAD
6             \s -> SZ
7
8  DELIMITER: * -> SZ
9  INT:       \s \n -> SZ
10            A-Z a-z _ -> SI
11            % -> OPERATOR
12            ! < > \- = \* -> SEQ
13            \+ -> SPLUS
14            & -> SAMP
15            ; -> DELIMITER
16            | -> SPIPE
17            / -> SSLASH
18            0 -> SPRE_NUM
19            1-9 -> SDEC_OCT_NUM
20            * -> BAD
21            " -> SCHARS
22            EOF -> GOOD
23
24 FLOAT:     \s \n -> SZ
25            A-Z a-z _ -> SI
26            % -> OPERATOR
27            ! < > \- = \* -> SEQ
28            \+ -> SPLUS
29            & -> SAMP
30            ; -> DELIMITER
```

```
31              | -> SPIPE
32              / -> SSLASH
33              0 -> SPRE_NUM
34              1-9 -> SDEC_OCT_NUM
35              * -> BAD
36              EOF -> GOOD
37
38  IDENTIFIER: \s \n -> SZ
39              A-Z a-z _ -> SI
40              % -> OPERATOR
41              ! < > \- = \* -> SEQ
42              \+ -> SPLUS
43              & -> SAMP
44              ; -> DELIMITER
45              | -> SPIPE
46              / -> SSLASH
47              0 -> SPRE_NUM
48              1-9 -> SDEC_OCT_NUM
49              * -> BAD
50              " -> SCHARS
51              EOF -> GOOD
52
53
54
```

Τέλος, έχουμε τις ενδιάμεσες καταστάσεις με τις οποίες ο αναλυτής κρίνει αν ο αριθμός που εισάγεται είναι πραγματικός ή ακέραιος.
Με βάση λοιπόν την σύνταξη της Uni-C, διαχωρίζονται αυτές οι δυό κατηγορίες.
(πχ. ακέραιοι: 0, 954, 4235 πχ. πραγματικοί: 3.14 2.43E31, 35.3e10, 34e12, 0e0, 0.2222)

```
SPRE_NUM:    x X -> SHEX
             0-9 -> SDEC_OCT_NUM
             \.  ->  SFLO1
             e E -> SFLO2
             *   -> BAD
             \s  -> INT

SDEC_OCT_NUM:    0-9      -> SDEC_OCT_NUM
                 \.       -> SFLO_COMMA
                 e E      -> SFLO_E
                 \s \n ; -> FLOAT
                 * -> BAD

SFLO_COMMA:      \s \n ; -> FLOAT
                 e E      -> SFLO_E
                 0-9      -> SFLO_DIGITS
                 * -> BAD

SFLO_DIGITS:     0-9      -> SFLO_DIGITS
                 \n \s ; -> FLOAT
                 e E      -> SFLO_E
                 \.       -> SFLO_COMMA
                 * -> BAD

SFLO_E:          \.        -> SFLO_COMMA
                 \-        -> SFLO_POWER
                 0-9      -> SFLO_DIGITS
                 * -> BAD

SFLO_POWER:      0-9      -> SFLO_DIGITS
                 * -> BAD

SHEX:        A-F      -> SHEX
             0-9      -> SHEX
             \s \n ; -> INT
             * -> BAD

```

5 Ολοκληρωμένος Κώδικας σε FSM

Ο κώδικας του αυτοματού μας μπορεί να βρεθεί ως αρχείο και στον φάκελο *Source_Code* που βρήκεται μέσα στον κύριο φάκελο του παραδοτέου.
Ακολουθεί ο ολοκληρωμένος κώδικας σε FSM του αυτόματού μας:

```
START=SZ
SZ: % -> OPERATOR
    ! < > \- = \* -> SEQ
    \+ -> SPLUS
    & -> SAMP
    ; -> DELIMITER
    | -> SPIPE
    a-z A-Z _ -> SI
    \n \s -> SZ
    / -> SSLASH
    " -> SCHARS
    0 -> SPRE_NUM
    1-9 -> SDEC_OCT_NUM
    * -> BAD
    EOF -> GOOD


SCHARS: * -> SCHARS
        " -> STR
        \\ -> SCAPE

SCAPE:  n -> NEWLINE
        \\ -> SCHARS
        * -> BAD

NEWLINE: " -> STR
         * -> SCHARS

SSLASH: / -> SCOMM1
        = \n \s -> OPERATOR
        \* -> SCOMM2

SEQ:     = \n \s -> OPERATOR
        * -> BAD

SPLUS:   = \+  -> OPERATOR
        * -> BAD

```

```
SAMP:     & \s  -> OPERATOR
         * -> BAD

SPIPE:    |     -> OPERATOR
         * -> BAD

SI:       a-z A-Z 0-9 _ -> SI
          ;           -> IDENTIFIER
          \s \n       -> IDENTIFIER
          EOF         -> IDENTIFIER
          \+ \- =     -> IDENTIFIER
          * -> BAD

SCOMM1:      * -> SCOMM1
            \n -> SZ

SCOMM2:     \* -> SCOMM21
            * -> SCOMM2

SCOMM21:     / -> COMMENT
             * -> SCOMM2

COMMENT:     \n \s -> SZ

OPERATOR:    a-z A-Z _ -> SI
             0 -> SPRE_NUM
             1-9 -> SDEC_OCT_NUM
             " -> SCHARS
             * -> BAD
             \s -> SZ

DELIMITER:   * -> SZ

SPRE_NUM:    x X -> SHEX
             0-9 -> SDEC_OCT_NUM
             \.  ->  SFLO1
             e E -> SFLO2
             *   -> BAD
             \s  -> INT

SDEC_OCT_NUM:   0-9     -> SDEC_OCT_NUM
                \.      -> SFLO_COMMA
                e E     -> SFLO_E
                \s \n ; -> FLOAT
```

```
83                  * -> BAD
84
85  SFLO_COMMA:     \s \n ; -> FLOAT
86                  e E     -> SFLO_E
87                  0-9     -> SFLO_DIGITS
88                  * -> BAD
89
90  SFLO_DIGITS:    0-9     -> SFLO_DIGITS
91                  \n \s ; -> FLOAT
92                  e E     -> SFLO_E
93                  \.      -> SFLO_COMMA
94                  * -> BAD
95
96  SFLO_E:         \.       -> SFLO_COMMA
97                  \-       -> SFLO_POWER
98                  0-9     -> SFLO_DIGITS
99                  * -> BAD
100
101 SFLO_POWER:     0-9     -> SFLO_DIGITS
102                 * -> BAD
103
104 SHEX:       A-F     -> SHEX
105             0-9     -> SHEX
106             \s \n ; -> INT
107             * -> BAD
108
109 INT:        \s \n -> SZ
110             A-Z a-z _ -> SI
111             % -> OPERATOR
112             ! < > \- = \* -> SEQ
113             \+ -> SPLUS
114             & -> SAMP
115             ; -> DELIMITER
116             | -> SPIPE
117             / -> SSLASH
118             0 -> SPRE_NUM
119             1-9 -> SDEC_OCT_NUM
120             * -> BAD
121             " -> SCHARS
122             EOF -> GOOD
123
124 FLOAT:      \s \n -> SZ
125             A-Z a-z _ -> SI
126             % -> OPERATOR
```

```
127             ! < > \- = \* -> SEQ
128             \+ -> SPLUS
129             & -> SAMP
130             ; -> DELIMITER
131             | -> SPIPE
132             / -> SSLASH
133             0 -> SPRE_NUM
134             1-9 -> SDEC_OCT_NUM
135             * -> BAD
136             EOF -> GOOD
137
138 IDENTIFIER: \s \n -> SZ
139             A-Z a-z _ -> SI
140             % -> OPERATOR
141             ! < > \- = \* -> SEQ
142             \+ -> SPLUS
143             & -> SAMP
144             ; -> DELIMITER
145             | -> SPIPE
146             / -> SSLASH
147             0 -> SPRE_NUM
148             1-9 -> SDEC_OCT_NUM
149             * -> BAD
150             " -> SCHARS
151             EOF -> GOOD
152
153 STR:        \s \n -> SZ
154             ;     -> DELIMITER
155             EOF   -> GOOD
156
157 BAD(OK): \n -> SZ
158          * -> BAD
159 GOOD(OK):
160
161
```

6 Αποτελέσματα δοκιμών

Ακολοθούν ορισμένα απο τα αποτελέσματα που είχαμε όταν τράξαμε το πρόγραμμα στο bash terminal σε λειτουργικό σύστημα Linux.
Όλα τα τρεξίματα του αυτόματου με το fsm γίνονται με την εντολή

```
./fsm -trace automaton.fsm
```

6.1 Εισαγωγή `int metavliti = 23.3e19 ;`

Αποτέλεσμα στο τερματικό:

```
int metavliti = 23.3e19 ;
sz i -> si
si n -> si
si t -> si
si \s -> identifier
identifier m -> si
si e -> si
si t -> si
si a -> si
si v -> si
si l -> si
si i -> si
si t -> si
si i -> si
si \s -> identifier
identifier = -> seq
seq \s -> operator
operator 2 -> sdec_oct_num
sdec_oct_num 3 -> sdec_oct_num
sdec_oct_num . -> sflo_comma
sflo_comma 3 -> sflo_digits
sflo_digits e -> sflo_e
sflo_e 1 -> sflo_digits
sflo_digits 9 -> sflo_digits
sflo_digits \s -> float
float ; -> delimiter
delimiter \n -> sz

```

Βλέπουμε ότι το πρόγραμμα μας αναγνώρισε επιτυχώς όλες τις λέξεις! Αναγνώρισε τις λέξεις int και metavliti ως identifier, την λέξη 23.3e19 ως float, την λέξη ¨;' ως delimiter.

6.2 Εισαγωγή str1 = "really l0ng str_1ing() -+!#%^&*()"

Αποτέλεσμα στο τερματικό:

```
$ ./fsm -trace automaton.fsm
str1 = "really l0ng str_1ing() -+!@#%^&*()";
sz s -> si
si t -> si
si r -> si
si 1 -> si
si \s -> identifier
identifier = -> seq
seq \s -> operator
operator " -> schars
schars r -> schars
schars e -> schars
schars a -> schars
schars l -> schars
schars l -> schars
schars y -> schars
schars \s -> schars
schars l -> schars
schars 0 -> schars
schars n -> schars
schars g -> schars
schars \s -> schars
schars s -> schars
schars t -> schars
schars r -> schars
schars _ -> schars
schars 1 -> schars
schars i -> schars
schars n -> schars
schars g -> schars
schars ( -> schars
schars ) -> schars
schars \s -> schars
schars - -> schars
schars + -> schars
schars ! -> schars
schars @ -> schars
schars # -> schars
schars % -> schars
schars ^ -> schars
schars & -> schars
```

```
42 schars * -> schars
43 schars ( -> schars
44 schars ) -> schars
45 schars " -> str
46 str ; -> delimiter
47 str \n -> sz
48
```

Εδώ βλέπουμε ότι παρόλο που η συμβολοσειρά που δώθηκε είναι αρκετά ¨δύσκολη¨, το πρόγραμμά μας κατάφερε να ερμηνεύσει πάλι σωστά κάθε λέξη, από το αρχικό όνομα, τον τελεστή, την γραμματοσειρά μέχρι τον delimiter στο τέλος!

6.3 Εισαγωγή `if m1 == 2  m2 == 0 then m = "\n"`

Αποτέλεσμα στο τερματικό:

```
if m1 == 2 && m2 == 0 then m = "\n"
sz i -> si
si f -> si
si \s -> identifier
identifier m -> si
si 1 -> si
si \s -> identifier
identifier = -> seq
seq = -> operator
operator \s -> sz
sz 2 -> sdec_oct_num
sdec_oct_num \s -> float
float & -> samp
samp & -> operator
operator \s -> sz
sz m -> si
si 2 -> si
si \s -> identifier
identifier = -> seq
seq = -> operator
operator \s -> sz
sz 0 -> spre_num
spre_num \s -> int
int t -> si
si h -> si
si e -> si
si n -> si
si \s -> identifier
identifier m -> si
si \s -> identifier
identifier = -> seq
seq \s -> operator
operator " -> schars
schars \ -> scape
scape n -> newline
newline " -> str
str \n -> sz

```

Ξανά, αν και σχετικά πολύπλοκη η εισαγωγή μας,το πρόγραμμα κατάφερε ορθώς να καταλάβει τους αριθμούς, τα ονόματα, την συμβολοσειρά και τα newlines μέσα σε αυτή.

6.4 Εισαγωγή `mistake === this!`

Αποτέλεσμα στο τερματικό:

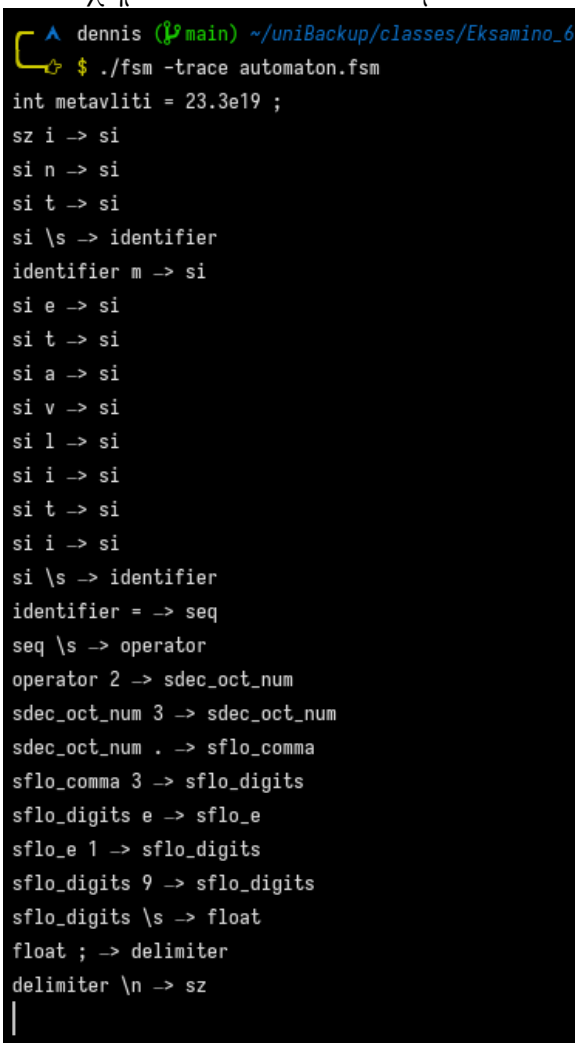
```
./fsm -trace automaton.fsm
mistake === this!
sz m -> si
si i -> si
si s -> si
si t -> si
si a -> si
si k -> si
si e -> si
si \s -> identifier
identifier = -> seq
seq = -> operator
operator = -> bad
bad \s -> bad
bad t -> bad
bad h -> bad
bad i -> bad
bad s -> bad
bad ! -> bad
bad \n -> sz

```

Τώρα εισάγαμε μια λανθασμένη εντολή στο πρόγραμμα, για να δούμε άμα την χειρησθεί σωστά.
Πράγματι, η εντολή δεν είναι σωστή, και θα πρέπει, όπως και το πρόγραμμα μας έκανε, να θεωρήται κάθε λέξη μετά απο το λάθος στην γραμμή αυτή λανθασμένη επίσης.

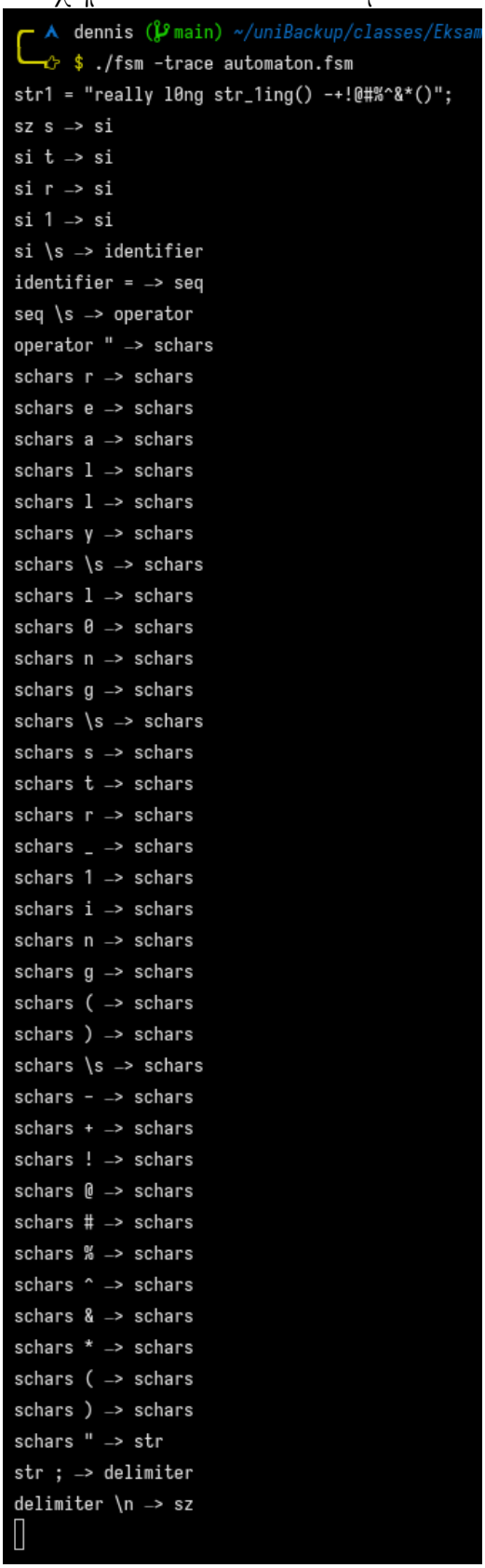
6.5 Αποδείξεις για τα αποτελέσματα

Ακολουθούν οι αποδείξεις ότι το πρόγραμμα πράγματι έδωσε τα παραπάνω αποτελέσματα (των κεφαλάιων 6.1, 6.2, 6.3, 6.4):

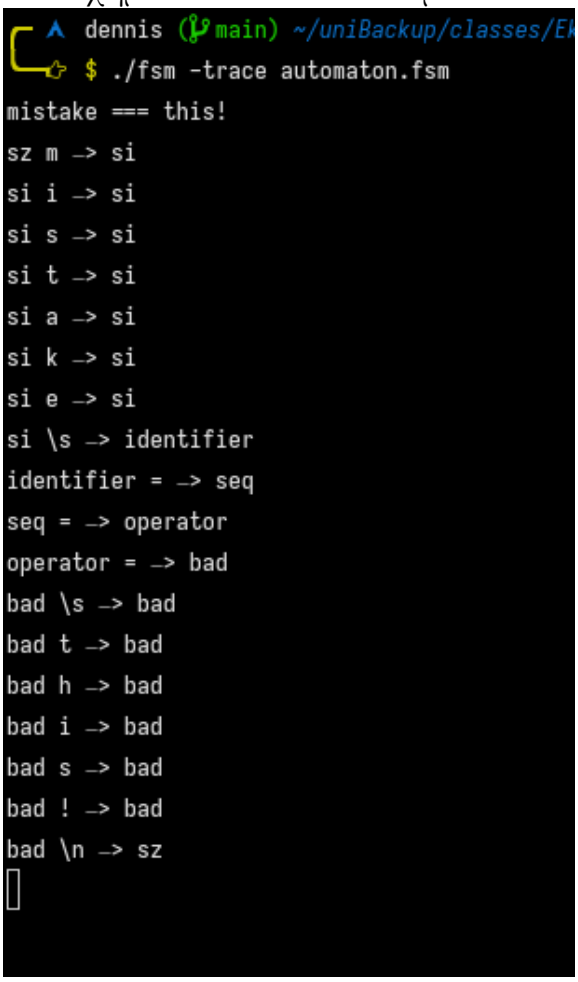
Σχήμα 3: Για αποτέλεσμα 6.1



Σχήμα 4: Για αποτέλεσμα 6.2



Σχήμα 5: Για αποτέλεσμα 6.3



Σχήμα 6: Για αποτέλεσμα 6.4