

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 A2: Διονύσης Νικολόπουλος

Username στο Github

Διονύσης Νικολόπουλος : jnu
Θανάσης Αναγνωστόπουλος : ThanasisAnagno
Αριστείδης Αναγνωστόπουλος : Aris-Anag
Σπυρίδων Φλώρος : spirosl

Η εργασία αυτή πραγματοποιήθηκε με χρήση L^AT_EX

Περιεχόμενα

1	Εισαγωγή	2
2	Περιγραφή σύνταξης της Uni-C σε BNF	4
3	Διαγράμματα Ενιαίου Αυτόματου	5
3.1	Γενικευμένο Διάγραμμα Ενιαίου Αυτόματου	5
3.2	Λεπτομερές Διάγραμμα Ενιαίου Αυτόματου	6
4	Εξήγηση Κώδικα σε FSM	7
5	Ολοκληρωμένος Κώδικας σε FSM	13
6	Αποτελέσματα δοκιμών	17
6.1	Εισαγωγή <code>int metavliti = 23.3e19 ;</code>	17
6.2	Εισαγωγή <code>str1 = "really long str_1ing() -+!#%^&*"</code> 18	18
6.3	Εισαγωγή <code>if m1 == 2 m2 == 0 then m = "\n"</code>	20
6.4	Εισαγωγή <code>mistake === this!</code>	21
6.5	Αποδείξεις για τα αποτελέσματα	22

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 προσέγγιση 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).

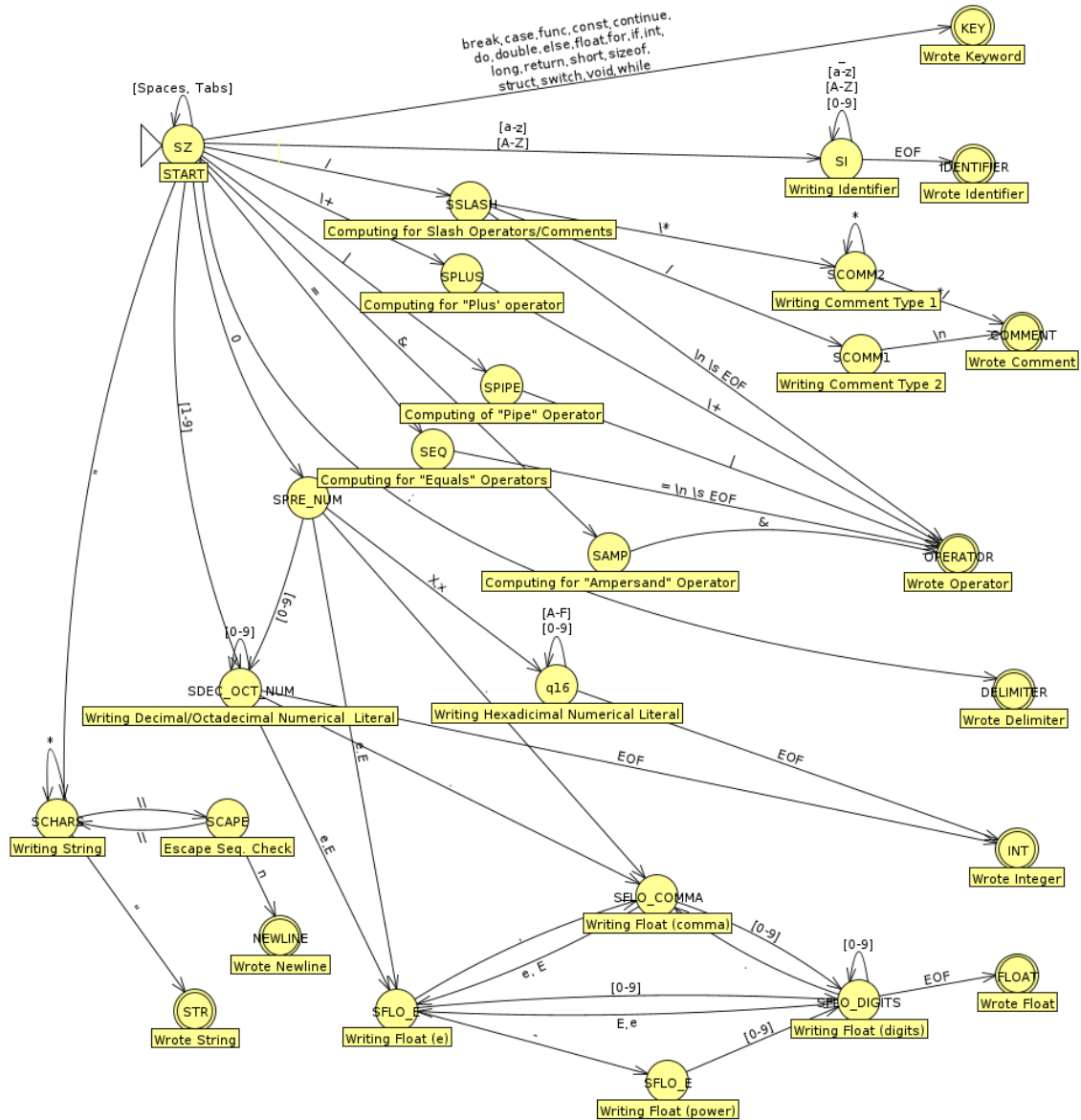
Οπότε, με την βοήθεια του περιβάλλοντος λεκτικού επεξεργαστή Emacs, γράψαμε τον εξής συμβολισμό.

```
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
```


3.2 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:

```
1 START=SZ
2 SZ: % -> OPERATOR
3     ! < > \- = \* -> SEQ
4     \+ -> SPLUS
5     & -> SAMP
6     ; -> DELIMITER
7     | -> SPIPE
8     a-z A-Z _ -> SI
9     \n \s -> SZ
10    / -> SSLASH
11    " -> SCHARS
12    0 -> SPRE_NUM
13    1-9 -> SDEC_OCT_NUM
14    * -> BAD
15    EOF -> GOOD
16
```

Εδώ, βλέπουμε ότι ο αναλυτής μας περιμένει τον πρώτο, ουσιαστικά χαρακτήρα από 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 είναι 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 DELIMITER) 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 από 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 εισαχθεί ο χαρακτήρας `\`.

Στην τελευταία περίπτωση, ο αναλυτής έρχεται στην κατάσταση SCAPE, την οποία περιμένει να εισαχθεί 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 πάει 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).

```
1 SSLASH: / -> SCOMM1
2         = \n \s -> OPERATOR
3         \* -> SCOMM2
4         * -> BAD
5
6 SEQ:    = \n \s -> OPERATOR
7         * -> BAD
8
9 SPLUS:  = \+ -> OPERATOR
10        * -> BAD
11
12 SAMP:   & \s -> OPERATOR
13        * -> BAD
14
15 SPIPE:  | -> OPERATOR
16        * -> BAD
17
```

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

```
1 SCOMM1:  * -> SCOMM1
2          \n -> SZ
3
4 SCOMM2:  \* -> SCOMM21
5          * -> SCOMM2
6
7 SCOMM21: / -> COMMENT
8          * -> SCOMM2
9
10 COMMENT: \n \s -> SZ
11          EOF -> GOOD
12
```

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

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

```

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

```

Εδώ βλέπουμε τις 'τελικές' καταστάσεις στις οποίες φτάνει ο αναλυτής όταν κρίνει μια λέξη να πληρεί τις προϋποθέσεις για να χαρακτηριστεί ως μια συγκεκριμένη λεκτική μονάδα.

Παρατηρούμε ότι, με βάση την γλώσσα Uni-C, οι λεκτικές μονάδες είναι δυνατό να ακολουθηθούν από συγκεκριμένες λεκτικές μονάδες (παράδειγμα οι τελεστές μονάχα από ένα νούμερο ή identifier, για αυτό οι μοναδικές επιτρεπτές καταστάσεις που μεταβαίνει ο αναλυτής μετά από την κατάσταση OPERATOR είναι η ενδιαμέση κατάσταση είτε αριθμού είτε 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 ενδιαμέσες 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 δύο κατηγορίες.
 (πχ. ακέραιοι: 0, 954, 4235 πχ. πραγματικοί: 3.14 2.43E31, 35.3e10, 34e12, 0e0, 0.2222)

```

1 SPRE_NUM:    x X -> SHEX
2              0-9 -> SDEC_OCT_NUM
3              \. -> SFLO1
4              e E -> SFLO2
5              * -> BAD
6              \s -> INT
7
8 SDEC_OCT_NUM: 0-9      -> SDEC_OCT_NUM
9              \.      -> SFLO_COMMA
10             e E      -> SFLO_E
11             \s \n ; -> FLOAT
12             * -> BAD
13
14 SFLO_COMMA:  \s \n ; -> FLOAT
15             e E      -> SFLO_E
16             0-9      -> SFLO_DIGITS
17             * -> BAD
18
19 SFLO_DIGITS: 0-9      -> SFLO_DIGITS
20             \n \s ; -> FLOAT
21             e E      -> SFLO_E
22             \.      -> SFLO_COMMA
23             * -> BAD
24
25 SFLO_E:      \.      -> SFLO_COMMA
26             \-      -> SFLO_POWER
27             0-9      -> SFLO_DIGITS
28             * -> BAD
29
30 SFLO_POWER:  0-9      -> SFLO_DIGITS
31             * -> BAD
32
33 SHEX:       A-F      -> SHEX
34             0-9      -> SHEX
35             \s \n ; -> INT
36             * -> BAD
37

```

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 αυτόματου μας:

```
1 START=SZ
2 SZ: % -> OPERATOR
3     ! < > \- = \* -> SEQ
4     \+ -> SPLUS
5     & -> SAMP
6     ; -> DELIMITER
7     | -> SPIPE
8     a-z A-Z _ -> SI
9     \n \s -> SZ
10    / -> SSLASH
11    " -> SCHARS
12    0 -> SPRE_NUM
13    1-9 -> SDEC_OCT_NUM
14    * -> BAD
15    EOF -> GOOD
16
17
18 SCHARS: * -> SCHARS
19         " -> STR
20         \\ -> SCAPE
21
22 SCAPE:  n -> NEWLINE
23         \\ -> SCHARS
24         * -> BAD
25
26 NEWLINE: " -> STR
27          * -> SCHARS
28
29 SSLASH: / -> SCOMM1
30         = \n \s -> OPERATOR
31         \* -> SCOMM2
32
33 SEQ:    = \n \s -> OPERATOR
34         * -> BAD
35
36 SPLUS:  = \+ -> OPERATOR
37         * -> BAD
38
```

```

39 SAMP:      & \s  -> OPERATOR
40          * -> BAD
41
42 SPIPE:     |      -> OPERATOR
43          * -> BAD
44
45 SI:        a-z A-Z 0-9 _ -> SI
46          ;          -> IDENTIFIER
47          \s \n      -> IDENTIFIER
48          EOF        -> IDENTIFIER
49          \+ \- =    -> IDENTIFIER
50          * -> BAD
51
52 SCOMM1:    * -> SCOMM1
53          \n -> SZ
54
55 SCOMM2:    \* -> SCOMM21
56          * -> SCOMM2
57
58 SCOMM21:   / -> COMMENT
59          * -> SCOMM2
60
61 COMMENT:   \n \s -> SZ
62
63 OPERATOR:  a-z A-Z _ -> SI
64          0 -> SPRE_NUM
65          1-9 -> SDEC_OCT_NUM
66          " -> SCHARS
67          * -> BAD
68          \s -> SZ
69
70 DELIMITER: * -> SZ
71
72 SPRE_NUM:  x X -> SHEX
73          0-9 -> SDEC_OCT_NUM
74          \. -> SFLO1
75          e E -> SFLO2
76          * -> BAD
77          \s -> INT
78
79 SDEC_OCT_NUM: 0-9      -> SDEC_OCT_NUM
80          \.      -> SFLO_COMMA
81          e E      -> SFLO_E
82          \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

```

```

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 STR:      \s \n -> SZ
54      ;      -> DELIMITER
55      EOF      -> GOOD
56
57 BAD(OK): \n -> SZ
58      * -> BAD
59 GOOD(OK):
60
61

```

6 Αποτελέσματα δοκιμών

Ακολουθούν 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:

```
1 int metavliti = 23.3e19 ;
2 sz i -> si
3 si n -> si
4 si t -> si
5 si \s -> identifier
6 identifier m -> si
7 si e -> si
8 si t -> si
9 si a -> si
10 si v -> si
11 si l -> si
12 si i -> si
13 si t -> si
14 si i -> si
15 si \s -> identifier
16 identifier = -> seq
17 seq \s -> operator
18 operator 2 -> sdec_oct_num
19 sdec_oct_num 3 -> sdec_oct_num
20 sdec_oct_num . -> sflo_comma
21 sflo_comma 3 -> sflo_digits
22 sflo_digits e -> sflo_e
23 sflo_e 1 -> sflo_digits
24 sflo_digits 9 -> sflo_digits
25 sflo_digits \s -> float
26 float ; -> delimiter
27 delimiter \n -> sz
28
```

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 10ng str_1ing() -+!#%^&*()"`

Αποτέλεσμα στο τερματικό:

```
1 $ ./fsm -trace automaton.fsm
2 str1 = "really 10ng str_1ing() -+!@#%^&*()";
3 sz s -> si
4 si t -> si
5 si r -> si
6 si l -> si
7 si \s -> identifier
8 identifier = -> seq
9 seq \s -> operator
10 operator " -> schars
11 schars r -> schars
12 schars e -> schars
13 schars a -> schars
14 schars l -> schars
15 schars l -> schars
16 schars y -> schars
17 schars \s -> schars
18 schars l -> schars
19 schars 0 -> schars
20 schars n -> schars
21 schars g -> schars
22 schars \s -> schars
23 schars s -> schars
24 schars t -> schars
25 schars r -> schars
26 schars _ -> schars
27 schars l -> schars
28 schars i -> schars
29 schars n -> schars
30 schars g -> schars
31 schars ( -> schars
32 schars ) -> schars
33 schars \s -> schars
34 schars - -> schars
35 schars + -> schars
36 schars ! -> schars
37 schars @ -> schars
38 schars # -> schars
39 schars % -> schars
40 schars ^ -> schars
41 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:

```
1 if m1 == 2 && m2 == 0 then m = "\n"
2 sz i -> si
3 si f -> si
4 si \s -> identifier
5 identifier m -> si
6 si 1 -> si
7 si \s -> identifier
8 identifier = -> seq
9 seq = -> operator
10 operator \s -> sz
11 sz 2 -> sdec_oct_num
12 sdec_oct_num \s -> float
13 float & -> samp
14 samp & -> operator
15 operator \s -> sz
16 sz m -> si
17 si 2 -> si
18 si \s -> identifier
19 identifier = -> seq
20 seq = -> operator
21 operator \s -> sz
22 sz 0 -> spre_num
23 spre_num \s -> int
24 int t -> si
25 si h -> si
26 si e -> si
27 si n -> si
28 si \s -> identifier
29 identifier m -> si
30 si \s -> identifier
31 identifier = -> seq
32 seq \s -> operator
33 operator " -> schars
34 schars \ -> scape
35 scape n -> newline
36 newline " -> str
37 str \n -> sz
38
```

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:

```
1 ./fsm -trace automaton.fsm
2 mistake === this!
3 sz m -> si
4 si i -> si
5 si s -> si
6 si t -> si
7 si a -> si
8 si k -> si
9 si e -> si
10 si \s -> identifier
11 identifier = -> seq
12 seq = -> operator
13 operator = -> bad
14 bad \s -> bad
15 bad t -> bad
16 bad h -> bad
17 bad i -> bad
18 bad s -> bad
19 bad ! -> bad
20 bad \n -> sz
21
```

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 κεφαλαίων 6.1, 6.2, 6.3, 6.4):

Σχήμα 3: Για αποτέλεσμα 6.1

```
dennis (Ψ main) ~/uniBackup/classes/Eksamino_6
$ ./fsm -trace automaton.fsm
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
|
```


Σχήμα 4: Για αποτέλεσμα 6.2

```
dennis (main) ~/uniBackup/classes/Eksam
$ ./fsm -trace automaton.fsm
str1 = "really long str_1ing() -+!@##%^&*(";
sz s -> si
si t -> si
si r -> si
si l -> 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 l -> 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
schars * -> schars
schars ( -> schars
schars ) -> schars
schars " -> str
str ; -> delimiter
delimiter \n -> sz
[]
```

Σχήμα 5: Για αποτέλεσμα 6.3

```
└─ dennis (Ψ main) ~/uniBackup/classes/EL
└─ $ ./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

```

Σχήμα 6: Για αποτέλεσμα 6.4

```
└─┐ A dennis ($main) ~/uniBackup/classes/f
└─┐ $ ./fsm -trace automaton.fsm
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
└─┐ A dennis ($main) ~/uniBackup/classes/f
```