

Institut Supérieur d'Électronique de Paris
Projet de Fin d'Études

Reponsable: M. Hugueney

Finite State Transducers Just-In-Time Compiling

Do you hear the bytecode ?

Émilien Boulben
Victor Delepine
Corentin Peuvrel

17 juin 2015
Paris

Table des matières

Introduction	1
1 Analyse préalable	2
2 Des FST en JIT	3
3 Optimiser : comment ?	4
Conclusion	5
A Annexe : tests avec un script shell	6
A.1 Dictionnaires	6
A.2 FST	7
A.3 Générer à la volée du code C	8
A.3.1 Script shell	8
A.3.2 Code C généré pour la FST définie dans le Tableau 3	10
A.3.3 Code C généré pour la FST définie dans le Tableau 4	13
A.4 Générer à la volée du code assembleur x86	15
A.4.1 Script shell	15
A.4.2 Code assembleur x86 généré pour la FST défini dans le Tableau 3	18
A.4.3 Code assembleur x86 généré pour la FST défini dans le Tableau 4	22

Listings

1	Script pour générer un code C à la volée d'une FST	8
2	Code C généré pour la FST définie dans Tableau 3	10
3	Code C généré pour la FST définie dans Tableau 4	13
4	Script pour générer un code assembleur x86 à la volée d'une FST	15
5	Code assembleur x86 généré pour la FST définie dans Tableau 3	18
6	Code assembleur x86 généré pour la FST définie dans Tableau 4	22

Liste des tableaux

1	Dictionnaire à utiliser avec la FST dans le Tableau 3	6
2	Dictionnaire à utiliser avec la FST dans le Tableau 4	6
3	FST utilisée avec le dictionnaire Tableau 1, voir Figure 1 page 7	7
4	FST utilisée avec le dictionnaire Tableau 2, voir Figure 2 page 7	7

Table des figures

1	La FST associée avec le Tableau 3 page 7	7
2	La FST associée avec le Tableau 4 page 7	7

Introduction

1 Analyse préalable

2 Des FST en JIT

3 Optimiser : comment ?

Conclusion

A Annexe : tests avec un script shell

A.1 Dictionnaires

Value	Word
0	mop
1	moth
2	pop
3	star
4	stop
5	top

TABLEAU 1 – Dictionnaire à utiliser avec la FST dans le Tableau 3

Value	Word
0	mop
1	moth
2	pop
3	slop
4	sloth
5	stop
6	top

TABLEAU 2 – Dictionnaire à utiliser avec la FST dans le Tableau 4

A.2 FST

Nœu	0	0	0	0	1	2	3	2	4	6	7	5	7	8	
Nœu suivant	1	4	4	6	2	3	9	9	5	7	5	9	8	9	
Nœu final															9
Caractère	M	P	T	S	O	T	H	P	O	T	O	P	A	R	
Poids			2	5	3			1				1			

TABLEAU 3 – FST utilisée avec le dictionnaire Tableau 1, voir Figure 1 page 7

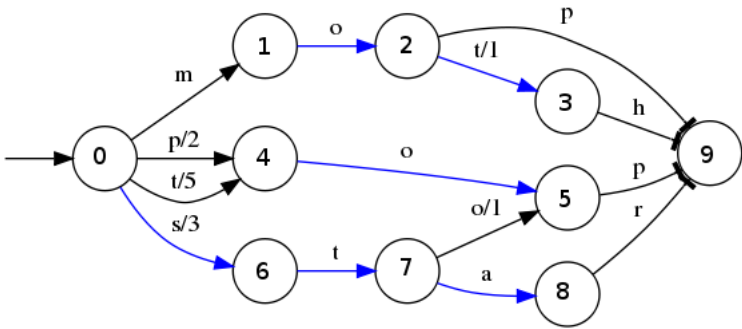


FIGURE 1 – La FST associée avec le Tableau 3 page 7

Nœu	0	0	0	0	3	3	1	2	4	5	6	5	
Nœu suivant	1	1	3	4	1	4	2	7	5	6	7	7	
Nœu final													7
Caractère	P	T	S	M	T	L	O	P	O	T	H	P	
Poids		2	6	3		2					1		

TABLEAU 4 – FST utilisée avec le dictionnaire Tableau 2, voir Figure 2 page 7

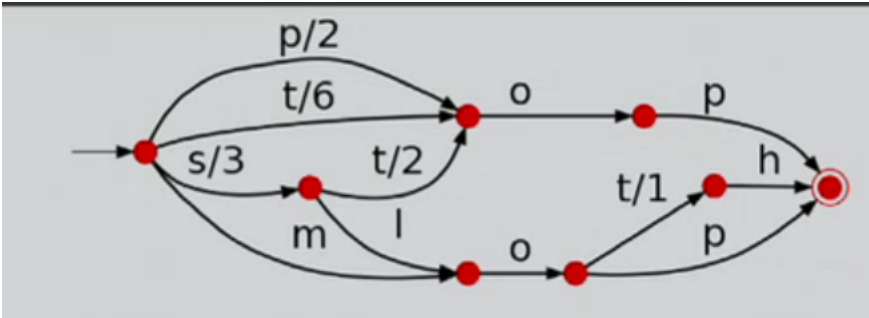


FIGURE 2 – La FST associée avec le Tableau 4 page 7

A.3 Générer à la volée du code C

A.3.1 Script shell

```

1  #!/bin/bash
2
3  FST="$1"
4
5  sort "$FST" > "$FST.sort"
6
7  FIRST_CALL=1
8
9  cat <<EOF
10 #include <stdio.h>
11
12 int compute_fst(const char* token)
13 {
14     int pos=0;
15     int total=0;
16
17 EOF
18
19 while read DEP ARR CHAR WEIGHT ; do
20     # If it's a final node
21     if [[ ! "$CHAR" ]]; then
22         WEIGHT=${ARR:-0}
23         cat <<EOF
24         default:
25             return -1;
26     }
27
28 NODE_$DEP :
29 EOF
30     (( WEIGHT != 0 )) &&
31     echo "        total += $WEIGHT;"
32     echo "        goto END;"
33
34     continue
35 fi
36
37 : ${WEIGHT:=0}
38
39 if [[ $DEP != $PREV_DEP ]]; then
40     if [[ ! "$FIRST_CALL" ]]; then
41         cat <<EOF
42         default:
43             return -1;
44     }
45
46 EOF
47     fi
48
49     cat <<EOF
50 NODE_$DEP :
51     pos++;
52     switch (token[pos-1]) {
53 EOF
54     fi
55
56     echo "        case '$CHAR':"
```

```
57      (( WEIGHT != 0 )) &&
58          echo "          total += $WEIGHT;"
59      echo "          goto NODE_$ARR;"
60
61      PREV_DEP=$DEP
62      FIRST_CALL=
63  done < "$FST.sort"
64
65  cat <<EOF
66
67  END :
68      return total;
69  }
70
71  int main(int argc, const char *argv[])
72  {
73      if (argc < 2)
74          return 1;
75
76      printf("%d\n", compute_fst(argv[1]));
77      return 0;
78  }
79  EOF
80
81  rm "$FST.sort"
```

Listing 1 – Script pour générer un code C à la volée d'une FST

A.3.2 Code C généré pour la FST définie dans le Tableau 3

```
1  #include <stdio.h>
2
3  int compute_fst(const char* token)
4  {
5      int pos=0;
6      int total=0;
7
8  NODE_0 :
9      pos++;
10     switch (token[pos-1]) {
11     case 'M':
12         goto NODE_1;
13     case 'P':
14         total += 2;
15         goto NODE_4;
16     case 'T':
17         total += 5;
18         goto NODE_4;
19     case 'S':
20         total += 3;
21         goto NODE_6;
22     default:
23         return -1;
24     }
25
26  NODE_1 :
27      pos++;
28      switch (token[pos-1]) {
29      case 'O':
30          goto NODE_2;
31      default:
32          return -1;
33      }
34
35  NODE_2 :
36      pos++;
37      switch (token[pos-1]) {
38      case 'T':
39          total += 1;
40          goto NODE_3;
41      case 'P':
42          goto NODE_9;
43      default:
44          return -1;
45      }
46
47  NODE_3 :
48      pos++;
49      switch (token[pos-1]) {
50      case 'H':
51          goto NODE_9;
52      default:
53          return -1;
54      }
55
56  NODE_4 :
57      pos++;
58      switch (token[pos-1]) {
```

```
59     case 'O':
60         goto NODE_5;
61     default:
62         return -1;
63 }
64
65 NODE_5 :
66     pos++;
67     switch (token[pos-1]) {
68     case 'P':
69         goto NODE_9;
70     default:
71         return -1;
72     }
73
74 NODE_6 :
75     pos++;
76     switch (token[pos-1]) {
77     case 'T':
78         goto NODE_7;
79     default:
80         return -1;
81     }
82
83 NODE_7 :
84     pos++;
85     switch (token[pos-1]) {
86     case 'O':
87         total += 1;
88         goto NODE_5;
89     case 'A':
90         goto NODE_8;
91     default:
92         return -1;
93     }
94
95 NODE_8 :
96     pos++;
97     switch (token[pos-1]) {
98     case 'R':
99         goto NODE_9;
100    default:
101        return -1;
102    }
103
104 NODE_9 :
105     goto END;
106
107 END :
108     return total;
109 }
110
111 int main(int argc, const char *argv[])
112 {
113     if (argc < 2)
114         return 1;
115
116     printf("%d\n", compute_fst(argv[1]));
117     return 0;
```


118 || }

Listing 2 – Code C généré pour la FST définie dans Tableau 3

A.3.3 Code C généré pour la FST définie dans le Tableau 4

```
1  #include <stdio.h>
2
3  int compute_fst(const char* token)
4  {
5      int pos=0;
6      int total=0;
7
8  NODE_0 :
9      pos++;
10     switch (token[pos-1]) {
11     case 'P':
12         total += 2;
13         goto NODE_1;
14     case 'T':
15         total += 6;
16         goto NODE_1;
17     case 'S':
18         total += 3;
19         goto NODE_3;
20     case 'M':
21         goto NODE_4;
22     default:
23         return -1;
24     }
25
26  NODE_1 :
27      pos++;
28      switch (token[pos-1]) {
29      case 'O':
30          goto NODE_2;
31      default:
32          return -1;
33      }
34
35  NODE_2 :
36      pos++;
37      switch (token[pos-1]) {
38      case 'P':
39          goto NODE_7;
40      default:
41          return -1;
42      }
43
44  NODE_3 :
45      pos++;
46      switch (token[pos-1]) {
47      case 'T':
48          total += 2;
49          goto NODE_1;
50      case 'L':
51          goto NODE_4;
52      default:
53          return -1;
54      }
55
56  NODE_4 :
57      pos++;
58      switch (token[pos-1]) {
```

```
59     case '0':
60         goto NODE_5;
61     default:
62         return -1;
63     }
64
65 NODE_5 :
66     pos++;
67     switch (token[pos-1]) {
68     case 'T':
69         total += 1;
70         goto NODE_6;
71     case 'P':
72         goto NODE_7;
73     default:
74         return -1;
75     }
76
77 NODE_6 :
78     pos++;
79     switch (token[pos-1]) {
80     case 'H':
81         goto NODE_7;
82     default:
83         return -1;
84     }
85
86 NODE_7 :
87     goto END;
88
89 END :
90     return total;
91 }
92
93 int main(int argc, const char *argv[])
94 {
95     if (argc < 2)
96         return 1;
97
98     printf("%d\n", compute_fst(argv[1]));
99     return 0;
100 }
```

Listing 3 – Code C généré pour la FST définie dans Tableau 4

A.4 Générer à la volée du code assembleur x86

A.4.1 Script shell

```

1  #!/bin/bash
2
3  FST="$1"
4  oIFS=$IFS
5
6  sort "$FST" > "$FST.sort"
7
8  FIRST_CALL=1
9
10 cat <<EOF
11     .file      "3.c"
12     .text
13     .globl    compute_fst
14     .type     compute_fst, @function
15
16 compute_fst:
17     pushq    %rbp                # remember old base pointer
18     movq     %rsp, %rbp          # set new base pointer
19     movq     %rdi, -24(%rbp)      # put content of rdi (token) in -24(%rbp)
20     movl     \0, -8(%rbp)         # pos (-8(%rbp)) = 0
21     movl     \0, -4(%rbp)         # total (-4(%rbp)) = 0
22
23 EOF
24
25 while read DEP ARR CHAR WEIGHT ; do
26     # If it's a final node
27     if [[ ! "$CHAR" ]]; then
28         WEIGHT=${ARR:-0}
29         cat <<EOF
30             movl \0-1, %eax        # default : return -1
31             jmp .RET
32
33 .NODE_$DEP:
34 EOF
35         (( WEIGHT != 0 )) &&
36         echo "    addl  \$$WEIGHT, -4(%rbp)          # total += Weight"
37         echo "    jmp   .END                        # goto END"
38
39         continue
40     fi
41
42     : ${WEIGHT:=0}
43
44     if [[ $DEP != $PREV_DEP ]]; then
45         if [[ ! "$FIRST_CALL" ]]; then
46             cat <<EOF
47                 movl \0-1, %eax        # default : return -1
48                 jmp .RET
49
50 EOF
51             fi
52
53             if [[ ${#tmp[@]} != 0 ]] ; then
54                 IFS=$'\n'
55                 echo "${tmp[*]}"
56                 IFS=$oIFS

```

```

57         tmp=()
58     fi
59
60     cat <<EOF
61 .NODE_$DEP:
62     addl    \$1, -8(%rbp)    # pos++
63     movl    -8(%rbp), %eax   # eax = pos
64     leaq    -1(%rax), %rdx   # rdx = pos - 1
65     movq    -24(%rbp), %rax  # load token (address) in rax
66     addq    %rdx, %rax       # rax = &(token[pos-1])
67     movzbl  (%rax), %eax     # eax = token[pos-1]
68
69 EOF
70 fi
71
72 printf -v CHAR '%d' "\"$CHAR"
73
74 echo "      cmpl \$\$CHAR, %eax      # case '$CHAR'"
75
76 if (( WEIGHT != 0 )) ; then
77     echo "      je .NODE_${DEP}_$CHAR"
78     tmp+=(
79         ".NODE_${DEP}_$CHAR:"
80         "      addl  \$$WEIGHT, -4(%rbp)      # total += $WEIGHT"
81         "      jmp  .NODE_$ARR"
82         ""
83     )
84 else
85     echo "      je .NODE_$ARR"
86 fi
87
88 PREV_DEP=$DEP
89 FIRST_CALL=
90 done < "$FST.sort"
91
92 cat <<EOF
93
94 .END:
95     movl    -4(%rbp), %eax   # Put return value in eax
96 .RET:
97     popq    %rbp
98     ret
99     .size   compute_fst, .-compute_fst
100    .section .rodata
101
102 .PRINTF_FMT:
103     .string "%d\n"
104     .text
105     .globl  main
106     .type   main, @function
107 main:
108     pushq   %rbp
109     movq    %rsp, %rbp
110     subq    \$16, %rsp
111     movl    %edi, -4(%rbp)
112     movq    %rsi, -16(%rbp)
113     cmpl    \$1, -4(%rbp)
114     jg      .DO_MAIN
115     movl    \$1, %eax
116     jmp     .END_MAIN

```

```
117 | .DO_MAIN:
118 |     movq    -16(%rbp), %rax
119 |     addq    \$.8, %rax
120 |     movq    (%rax), %rax
121 |     movq    %rax, %rdi
122 |     call    compute_fst
123 |     movl    %eax, %esi
124 |     movl    \$.PRINTF_FMT, %edi
125 |     movl    \$.0, %eax
126 |     call    printf
127 |     movl    \$.0, %eax
128 | .END_MAIN:
129 |     leave
130 |     ret
131 |     .size   main, .-main
132 |     .ident  "GCC: (GNU) 4.9.2 20150212 (Red Hat 4.9.2-6)"
133 |     .section        .note.GNU-stack,"",@progbits
134 | EOF
135 |
136 | rm "$FST.sort"
```

Listing 4 – Script pour générer un code assembleur x86 à la colée d'une FST

A.4.2 Code assembleur x86 généré pour la FST défini dans le Tableau 3

```

1      .file      "3.c"
2      .text
3      .globl    compute_fst
4      .type     compute_fst, @function
5
6  compute_fst:
7      pushq     %rbp                # remember old base pointer
8      movq     %rsp, %rbp          # set new base pointer
9      movq     %rdi, -24(%rbp)      # put content of rdi (token) in -24(%rbp)
10     movl     $0, -8(%rbp)         # pos (-8(%rbp)) = 0
11     movl     $0, -4(%rbp)         # total (-4(%rbp)) = 0
12
13     .NODE_0:
14         addl     $1, -8(%rbp)      # pos++
15         movl     -8(%rbp), %eax    # eax = pos
16         leaq     -1(%rax), %rdx    # rdx = pos - 1
17         movq     -24(%rbp), %rax   # load token (address) in rax
18         addq     %rdx, %rax        # rax = &(token[pos-1])
19         movzbl   (%rax), %eax      # eax = token[pos-1]
20
21         cmpl     $77, %eax         # case '77'
22         je      .NODE_1
23         cmpl     $80, %eax         # case '80'
24         je      .NODE_0_80
25         cmpl     $84, %eax         # case '84'
26         je      .NODE_0_84
27         cmpl     $83, %eax         # case '83'
28         je      .NODE_0_83
29         movl     $-1, %eax         # default : return -1
30         jmp     .RET
31
32     .NODE_0_80:
33         addl     $2, -4(%rbp)      # total += 2
34         jmp     .NODE_4
35
36     .NODE_0_84:
37         addl     $5, -4(%rbp)      # total += 5
38         jmp     .NODE_4
39
40     .NODE_0_83:
41         addl     $3, -4(%rbp)      # total += 3
42         jmp     .NODE_6
43
44     .NODE_1:
45         addl     $1, -8(%rbp)      # pos++
46         movl     -8(%rbp), %eax    # eax = pos
47         leaq     -1(%rax), %rdx    # rdx = pos - 1
48         movq     -24(%rbp), %rax   # load token (address) in rax
49         addq     %rdx, %rax        # rax = &(token[pos-1])
50         movzbl   (%rax), %eax      # eax = token[pos-1]
51
52         cmpl     $79, %eax         # case '79'
53         je      .NODE_2
54         movl     $-1, %eax         # default : return -1
55         jmp     .RET
56
57     .NODE_2:
58         addl     $1, -8(%rbp)      # pos++

```

```

59      movl    -8(%rbp), %eax # eax = pos
60      leaq    -1(%rax), %rdx # rdx = pos - 1
61      movq    -24(%rbp), %rax # load token (address) in rax
62      addq    %rdx, %rax     # rax = &(token[pos-1])
63      movzbl  (%rax), %eax   # eax = token[pos-1]
64
65      cmpl    $84, %eax      # case '84'
66      je     .NODE_2_84
67      cmpl    $80, %eax      # case '80'
68      je     .NODE_9
69      movl    $-1, %eax      # default : return -1
70      jmp     .RET
71
72 .NODE_2_84:
73      addl    $1, -4(%rbp)    # total += 1
74      jmp     .NODE_3
75
76 .NODE_3:
77      addl    $1, -8(%rbp)    # pos++
78      movl    -8(%rbp), %eax  # eax = pos
79      leaq    -1(%rax), %rdx  # rdx = pos - 1
80      movq    -24(%rbp), %rax # load token (address) in rax
81      addq    %rdx, %rax     # rax = &(token[pos-1])
82      movzbl  (%rax), %eax   # eax = token[pos-1]
83
84      cmpl    $72, %eax      # case '72'
85      je     .NODE_9
86      movl    $-1, %eax      # default : return -1
87      jmp     .RET
88
89 .NODE_4:
90      addl    $1, -8(%rbp)    # pos++
91      movl    -8(%rbp), %eax  # eax = pos
92      leaq    -1(%rax), %rdx  # rdx = pos - 1
93      movq    -24(%rbp), %rax # load token (address) in rax
94      addq    %rdx, %rax     # rax = &(token[pos-1])
95      movzbl  (%rax), %eax   # eax = token[pos-1]
96
97      cmpl    $79, %eax      # case '79'
98      je     .NODE_5
99      movl    $-1, %eax      # default : return -1
100     jmp     .RET
101
102 .NODE_5:
103     addl    $1, -8(%rbp)    # pos++
104     movl    -8(%rbp), %eax  # eax = pos
105     leaq    -1(%rax), %rdx  # rdx = pos - 1
106     movq    -24(%rbp), %rax # load token (address) in rax
107     addq    %rdx, %rax     # rax = &(token[pos-1])
108     movzbl  (%rax), %eax   # eax = token[pos-1]
109
110     cmpl    $80, %eax      # case '80'
111     je     .NODE_9
112     movl    $-1, %eax      # default : return -1
113     jmp     .RET
114
115 .NODE_6:
116     addl    $1, -8(%rbp)    # pos++
117     movl    -8(%rbp), %eax  # eax = pos
118     leaq    -1(%rax), %rdx  # rdx = pos - 1

```



```

119     movq    -24(%rbp), %rax # load token (address) in rax
120     addq    %rdx, %rax      # rax = &(token[pos-1])
121     movzbl  (%rax), %eax     # eax = token[pos-1]
122
123     cmpl    $84, %eax        # case '84'
124     je      .NODE_7
125     movl    $-1, %eax        # default : return -1
126     jmp     .RET
127
128 .NODE_7:
129     addl    $1, -8(%rbp)     # pos++
130     movl    -8(%rbp), %eax   # eax = pos
131     leaq    -1(%rax), %rdx   # rdx = pos - 1
132     movq    -24(%rbp), %rax  # load token (address) in rax
133     addq    %rdx, %rax      # rax = &(token[pos-1])
134     movzbl  (%rax), %eax     # eax = token[pos-1]
135
136     cmpl    $79, %eax        # case '79'
137     je      .NODE_7_79
138     cmpl    $65, %eax        # case '65'
139     je      .NODE_8
140     movl    $-1, %eax        # default : return -1
141     jmp     .RET
142
143 .NODE_7_79:
144     addl    $1, -4(%rbp)     # total += 1
145     jmp     .NODE_5
146
147 .NODE_8:
148     addl    $1, -8(%rbp)     # pos++
149     movl    -8(%rbp), %eax   # eax = pos
150     leaq    -1(%rax), %rdx   # rdx = pos - 1
151     movq    -24(%rbp), %rax  # load token (address) in rax
152     addq    %rdx, %rax      # rax = &(token[pos-1])
153     movzbl  (%rax), %eax     # eax = token[pos-1]
154
155     cmpl    $82, %eax        # case '82'
156     je      .NODE_9
157     movl    $-1, %eax        # default : return -1
158     jmp     .RET
159
160 .NODE_9:
161     jmp     .END             # goto END
162
163 .END:
164     movl    -4(%rbp), %eax   # Put return value in eax
165 .RET:
166     popq    %rbp
167     ret
168     .size   compute_fst, .-compute_fst
169     .section        .rodata
170
171 .PRINTF_FMT:
172     .string "%d\n"
173     .text
174     .globl  main
175     .type   main, @function
176 main:
177     pushq   %rbp
178     movq    %rsp, %rbp

```

```

179      subq    $16, %rsp
180      movl    %edi, -4(%rbp)
181      movq    %rsi, -16(%rbp)
182      cmpl    $1, -4(%rbp)
183      jg      .DO_MAIN
184      movl    $1, %eax
185      jmp     .END_MAIN
186 .DO_MAIN:
187      movq    -16(%rbp), %rax
188      addq    $8, %rax
189      movq    (%rax), %rax
190      movq    %rax, %rdi
191      call    compute_fst
192      movl    %eax, %esi
193      movl    $.PRINTF_FMT, %edi
194      movl    $0, %eax
195      call    printf
196      movl    $0, %eax
197 .END_MAIN:
198      leave
199      ret
200      .size    main, .-main
201      .ident   "GCC: (GNU) 4.9.2 20150212 (Red Hat 4.9.2-6)"
202      .section .note.GNU-stack,"",@progbits

```

Listing 5 – Code assembleur x86 généré pour la FST définie dans Tableau 3

A.4.3 Code assembleur x86 généré pour la FST défini dans le Tableau 4

```

1      .file      "3.c"
2      .text
3      .globl    compute_fst
4      .type     compute_fst, @function
5
6  compute_fst:
7      pushq     %rbp                # remember old base pointer
8      movq      %rsp, %rbp          # set new base pointer
9      movq      %rdi, -24(%rbp)     # put content of rdi (token) in -24(%rbp)
10     movl      $0, -8(%rbp)        # pos (-8(%rbp)) = 0
11     movl      $0, -4(%rbp)        # total (-4(%rbp)) = 0
12
13  .NODE_0:
14     addl      $1, -8(%rbp)        # pos++
15     movl      -8(%rbp), %eax       # eax = pos
16     leaq      -1(%rax), %rdx       # rdx = pos - 1
17     movq      -24(%rbp), %rax      # load token (address) in rax
18     addq      %rdx, %rax           # rax = &(token[pos-1])
19     movzbl    (%rax), %eax         # eax = token[pos-1]
20
21     cmpl      $80, %eax            # case '80'
22     je        .NODE_0_80
23     cmpl      $84, %eax            # case '84'
24     je        .NODE_0_84
25     cmpl      $83, %eax            # case '83'
26     je        .NODE_0_83
27     cmpl      $77, %eax            # case '77'
28     je        .NODE_4
29     movl      $-1, %eax            # default : return -1
30     jmp       .RET
31
32  .NODE_0_80:
33     addl      $2, -4(%rbp)         # total += 2
34     jmp       .NODE_1
35
36  .NODE_0_84:
37     addl      $6, -4(%rbp)         # total += 6
38     jmp       .NODE_1
39
40  .NODE_0_83:
41     addl      $3, -4(%rbp)         # total += 3
42     jmp       .NODE_3
43
44  .NODE_1:
45     addl      $1, -8(%rbp)        # pos++
46     movl      -8(%rbp), %eax       # eax = pos
47     leaq      -1(%rax), %rdx       # rdx = pos - 1
48     movq      -24(%rbp), %rax      # load token (address) in rax
49     addq      %rdx, %rax           # rax = &(token[pos-1])
50     movzbl    (%rax), %eax         # eax = token[pos-1]
51
52     cmpl      $79, %eax            # case '79'
53     je        .NODE_2
54     movl      $-1, %eax            # default : return -1
55     jmp       .RET
56
57  .NODE_2:
58     addl      $1, -8(%rbp)        # pos++

```

```

59      movl    -8(%rbp), %eax # eax = pos
60      leaq    -1(%rax), %rdx # rdx = pos - 1
61      movq    -24(%rbp), %rax # load token (address) in rax
62      addq    %rdx, %rax     # rax = &(token[pos-1])
63      movzbl  (%rax), %eax   # eax = token[pos-1]
64
65      cmpl    $80, %eax      # case '80'
66      je      .NODE_7
67      movl    $-1, %eax      # default : return -1
68      jmp     .RET
69
70 .NODE_3:
71      addl    $1, -8(%rbp)   # pos++
72      movl    -8(%rbp), %eax # eax = pos
73      leaq    -1(%rax), %rdx # rdx = pos - 1
74      movq    -24(%rbp), %rax # load token (address) in rax
75      addq    %rdx, %rax     # rax = &(token[pos-1])
76      movzbl  (%rax), %eax   # eax = token[pos-1]
77
78      cmpl    $84, %eax      # case '84'
79      je      .NODE_3_84
80      cmpl    $76, %eax      # case '76'
81      je      .NODE_4
82      movl    $-1, %eax      # default : return -1
83      jmp     .RET
84
85 .NODE_3_84:
86      addl    $2, -4(%rbp)   # total += 2
87      jmp     .NODE_1
88
89 .NODE_4:
90      addl    $1, -8(%rbp)   # pos++
91      movl    -8(%rbp), %eax # eax = pos
92      leaq    -1(%rax), %rdx # rdx = pos - 1
93      movq    -24(%rbp), %rax # load token (address) in rax
94      addq    %rdx, %rax     # rax = &(token[pos-1])
95      movzbl  (%rax), %eax   # eax = token[pos-1]
96
97      cmpl    $79, %eax      # case '79'
98      je      .NODE_5
99      movl    $-1, %eax      # default : return -1
100     jmp     .RET
101
102 .NODE_5:
103     addl    $1, -8(%rbp)   # pos++
104     movl    -8(%rbp), %eax # eax = pos
105     leaq    -1(%rax), %rdx # rdx = pos - 1
106     movq    -24(%rbp), %rax # load token (address) in rax
107     addq    %rdx, %rax     # rax = &(token[pos-1])
108     movzbl  (%rax), %eax   # eax = token[pos-1]
109
110     cmpl    $84, %eax      # case '84'
111     je      .NODE_5_84
112     cmpl    $80, %eax      # case '80'
113     je      .NODE_7
114     movl    $-1, %eax      # default : return -1
115     jmp     .RET
116
117 .NODE_5_84:
118     addl    $1, -4(%rbp)   # total += 1

```

```

119     jmp .NODE_6
120
121 .NODE_6:
122     addl    $1, -8(%rbp)    # pos++
123     movl    -8(%rbp), %eax  # eax = pos
124     leaq    -1(%rax), %rdx  # rdx = pos - 1
125     movq    -24(%rbp), %rax # load token (address) in rax
126     addq    %rdx, %rax      # rax = &(token[pos-1])
127     movzbl  (%rax), %eax    # eax = token[pos-1]
128
129     cmpl    $72, %eax       # case '72'
130     je .NODE_7
131     movl    $-1, %eax       # default : return -1
132     jmp .RET
133
134 .NODE_7:
135     jmp .END                # goto END
136
137 .END:
138     movl    -4(%rbp), %eax  # Put return value in eax
139 .RET:
140     popq    %rbp
141     ret
142     .size   compute_fst, .-compute_fst
143     .section        .rodata
144
145 .PRINTF_FMT:
146     .string  "%d\n"
147     .text
148     .globl   main
149     .type    main, @function
150 main:
151     pushq    %rbp
152     movq     %rsp, %rbp
153     subq     $16, %rsp
154     movl     %edi, -4(%rbp)
155     movq     %rsi, -16(%rbp)
156     cmpl     $1, -4(%rbp)
157     jg       .DO_MAIN
158     movl     $1, %eax
159     jmp      .END_MAIN
160 .DO_MAIN:
161     movq     -16(%rbp), %rax
162     addq     $8, %rax
163     movq     (%rax), %rax
164     movq     %rax, %rdi
165     call     compute_fst
166     movl     %eax, %esi
167     movl     $.PRINTF_FMT, %edi
168     movl     $0, %eax
169     call     printf
170     movl     $0, %eax
171 .END_MAIN:
172     leave
173     ret
174     .size    main, .-main
175     .ident    "GCC: (GNU) 4.9.2 20150212 (Red Hat 4.9.2-6)"
176     .section        .note.GNU-stack, "", @progbits

```

Listing 6 – Code assembleur x86 généré pour la FST définie dans Tableau 4