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

Listings

Liste des tableaux

Table des figures

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

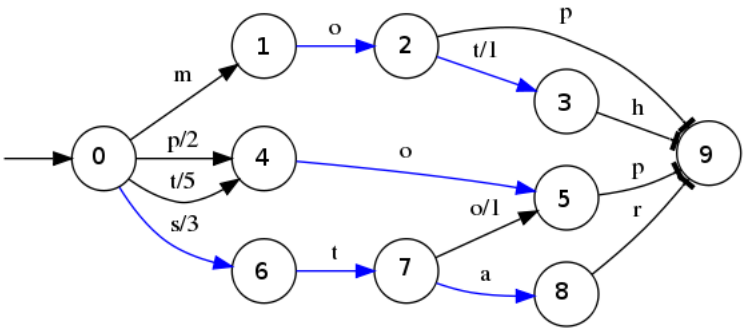

FIGURE 1 – La FST associée avec le Tableau 3 page 7

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

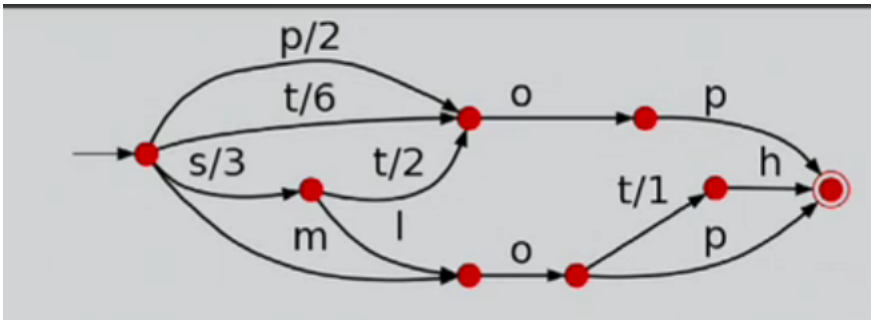

FIGURE 2 – La FST associée avec le Tableau 4 page 7

A.3 Générer à la volée du code C

A.3.1 Script shell

```
#!/bin/bash

FST="$1"

sort "$FST" > "$FST.sort"

FIRST_CALL=1

cat <<EOF
#include <stdio.h>

int compute_fst(const char* token)
{
    int pos=0;
    int total=0;

EOF

while read DEP ARR CHAR WEIGHT ; do
    # If it's a final node
    if [[ ! "$CHAR" ]]; then
        WEIGHT=${ARR:-0}
        cat <<EOF
    default:
        return -1;
    }

NODE_$DEP :
EOF
    (( WEIGHT != 0 )) &&
        echo "    total += $WEIGHT;"
    echo "    goto END;"

        continue
    fi

    : ${WEIGHT:=0}

    if [[ $DEP != $PREV_DEP ]]; then
        if [[ ! "$FIRST_CALL" ]]; then
            cat <<EOF
    default:
        return -1;
    }

EOF
        fi

        cat <<EOF
NODE_$DEP :
    pos++;
    switch (token[pos-1]) {
EOF
    fi

    echo "    case '$CHAR':"
```

```
    (( WEIGHT != 0 )) &&
        echo "        total += $WEIGHT;"
    echo "        goto NODE_$ARR;"

    PREV_DEP=$DEP
    FIRST_CALL=
done < "$FST.sort"

cat <<EOF

END :
    return total;
}

int main(int argc, const char *argv[])
{
    if (argc < 2)
        return 1;

    printf("%d\n", compute_fst(argv[1]));
    return 0;
}
EOF

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

```c
#include <stdio.h>

int compute_fst(const char* token)
{
    int pos=0;
    int total=0;

NODE_0 :
    pos++;
    switch (token[pos-1]) {
    case 'M':
        goto NODE_1;
    case 'P':
        total += 2;
        goto NODE_4;
    case 'T':
        total += 5;
        goto NODE_4;
    case 'S':
        total += 3;
        goto NODE_6;
    default:
        return -1;
    }

NODE_1 :
    pos++;
    switch (token[pos-1]) {
    case 'O':
        goto NODE_2;
    default:
        return -1;
    }

NODE_2 :
    pos++;
    switch (token[pos-1]) {
    case 'T':
        total += 1;
        goto NODE_3;
    case 'P':
        goto NODE_9;
    default:
        return -1;
    }

NODE_3 :
    pos++;
    switch (token[pos-1]) {
    case 'H':
        goto NODE_9;
    default:
        return -1;
    }

NODE_4 :
    pos++;
    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

```c
#include <stdio.h>

int compute_fst(const char* token)
{
    int pos=0;
    int total=0;

NODE_0 :
    pos++;
    switch (token[pos-1]) {
    case 'P':
        total += 2;
        goto NODE_1;
    case 'T':
        total += 6;
        goto NODE_1;
    case 'S':
        total += 3;
        goto NODE_3;
    case 'M':
        goto NODE_4;
    default:
        return -1;
    }

NODE_1 :
    pos++;
    switch (token[pos-1]) {
    case 'O':
        goto NODE_2;
    default:
        return -1;
    }

NODE_2 :
    pos++;
    switch (token[pos-1]) {
    case 'P':
        goto NODE_7;
    default:
        return -1;
    }

NODE_3 :
    pos++;
    switch (token[pos-1]) {
    case 'T':
        total += 2;
        goto NODE_1;
    case 'L':
        goto NODE_4;
    default:
        return -1;
    }

NODE_4 :
    pos++;
    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 '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

```
#!/bin/bash

FST="$1"
oIFS=$IFS

sort "$FST" > "$FST.sort"

FIRST_CALL=1

cat <<EOF
        .file   "3.c"
        .text
        .globl  compute_fst
        .type   compute_fst, @function

compute_fst:
        pushq   %rbp            # remember old base pointer
        movq    %rsp, %rbp      # set new base pointer
        movq    %rdi, -24(%rbp) # put content of rdi (token) in -24(%rbp)
        movl    \$0, -8(%rbp)    # pos   (-8(%rbp)) = 0
        movl    \$0, -4(%rbp)    # total (-4(%rbp)) = 0

EOF

while read DEP ARR CHAR WEIGHT ; do
    # If it's a final node
    if [[ ! "$CHAR" ]]; then
        WEIGHT=${ARR:-0}
        cat <<EOF
        movl \$-1, %eax          # default : return -1
        jmp .RET

.NODE_$DEP:
EOF
    (( WEIGHT != 0 )) &&
        echo "  addl  \$$WEIGHT, -4(%rbp)         # total += Weight"
    echo "       jmp .END                # goto END"

        continue
    fi

    : ${WEIGHT:=0}

    if [[ $DEP != $PREV_DEP ]]; then
        if [[ ! "$FIRST_CALL" ]]; then
            cat <<EOF
        movl \$-1, %eax          # default : return -1
        jmp .RET

EOF
        fi

        if [[ ${#tmp[@]} != 0 ]] ; then
            IFS=$'\n'
            echo "${tmp[*]}"
            IFS=$oIFS
```

```
            tmp=()
        fi

        cat <<EOF
.NODE_$DEP:
        addl    \$1, -8(%rbp)     # pos++
        movl    -8(%rbp), %eax    # eax = pos
        leaq    -1(%rax), %rdx    # rdx = pos - 1
        movq    -24(%rbp), %rax   # load token (address) in rax
        addq    %rdx, %rax        # rax = &(token[pos-1])
        movzbl  (%rax), %eax      # eax = token[pos-1]

EOF
    fi

    printf -v CHAR_INT '%d' "\"$CHAR"

    echo "       cmpl \$$CHAR_INT, %eax           # case '$CHAR'"

    if (( WEIGHT != 0 )) ; then
        echo "  je .NODE_${DEP}_$CHAR"
        tmp+=(
            ".NODE_${DEP}_$CHAR:"
            "   addl  \$$WEIGHT, -4(%rbp)      # total += $WEIGHT"
            "   jmp .NODE_$ARR"
            ""
            )
    else
        echo "  je .NODE_$ARR"
    fi

    PREV_DEP=$DEP
    FIRST_CALL=
done < "$FST.sort"

cat <<EOF

.END:
        movl    -4(%rbp), %eax  # Put return value in eax
.RET:
        popq    %rbp
        ret
        .size   compute_fst, .-compute_fst
        .section        .rodata

.PRINTF_FMT:
        .string "%d\n"
        .text
        .globl  main
        .type   main, @function
main:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    \$16, %rsp
        movl    %edi, -4(%rbp)
        movq    %rsi, -16(%rbp)
        cmpl    \$1, -4(%rbp)         # if (argc < 2)
        jg      .DO_MAIN
        movl    \$1, %eax             # return 1;
        jmp     .END_MAIN
```

```
117 .DO_MAIN:
118         movq    -16(%rbp), %rax      # rax = &(argv[0])
119         addq    \$8, %rax             # rax = &(argv[1])
120         movq    (%rax), %rax         # rax = argv[1]
121         movq    %rax, %rdi           # rdi = argv[1]
122         call    compute_fst
123         movl    %eax, %esi           # esi = compute_fst()
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 définié dans le Tableau 3

```
        .file   "3.c"
        .text
        .globl  compute_fst
        .type   compute_fst, @function

compute_fst:
        pushq   %rbp            # remember old base pointer
        movq    %rsp, %rbp      # set new base pointer
        movq    %rdi, -24(%rbp) # put content of rdi (token) in -24(%rbp)
        movl    $0, -8(%rbp)    # pos   (-8(%rbp)) = 0
        movl    $0, -4(%rbp)    # total (-4(%rbp)) = 0

.NODE_0:
        addl    $1, -8(%rbp)    # pos++
        movl    -8(%rbp), %eax  # eax = pos
        leaq    -1(%rax), %rdx  # rdx = pos - 1
        movq    -24(%rbp), %rax # load token (address) in rax
        addq    %rdx, %rax      # rax = &(token[pos-1])
        movzbl  (%rax), %eax    # eax = token[pos-1]

        cmpl $77, %eax          # case 'M'
        je .NODE_1
        cmpl $80, %eax          # case 'P'
        je .NODE_0_P
        cmpl $84, %eax          # case 'T'
        je .NODE_0_T
        cmpl $83, %eax          # case 'S'
        je .NODE_0_S
        movl $-1, %eax          # default : return -1
        jmp .RET

.NODE_0_P:
        addl  $2, -4(%rbp)      # total += 2
        jmp .NODE_4

.NODE_0_T:
        addl  $5, -4(%rbp)      # total += 5
        jmp .NODE_4

.NODE_0_S:
        addl  $3, -4(%rbp)      # total += 3
        jmp .NODE_6

.NODE_1:
        addl    $1, -8(%rbp)    # pos++
        movl    -8(%rbp), %eax  # eax = pos
        leaq    -1(%rax), %rdx  # rdx = pos - 1
        movq    -24(%rbp), %rax # load token (address) in rax
        addq    %rdx, %rax      # rax = &(token[pos-1])
        movzbl  (%rax), %eax    # eax = token[pos-1]

        cmpl $79, %eax          # case 'O'
        je .NODE_2
        movl $-1, %eax          # default : return -1
        jmp .RET

.NODE_2:
        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 'T'
66          je .NODE_2_T
67          cmpl $80, %eax           # case 'P'
68          je .NODE_9
69          movl $-1, %eax           # default : return -1
70          jmp .RET
71
72  .NODE_2_T:
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 'H'
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 'O'
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 'P'
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 'T'
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 'O'
137         je .NODE_7_O
138         cmpl $65, %eax          # case 'A'
139         je .NODE_8
140         movl $-1, %eax          # default : return -1
141         jmp .RET
142
143 .NODE_7_O:
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 'R'
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)        # if (argc < 2)
183         jg      .DO_MAIN
184         movl    $1, %eax            # return 1;
185         jmp     .END_MAIN
186 .DO_MAIN:
187         movq    -16(%rbp), %rax     # rax = &(argv[0])
188         addq    $8, %rax            # rax = &(argv[1])
189         movq    (%rax), %rax        # rax = argv[1]
190         movq    %rax, %rdi          # rdi = argv[1]
191         call    compute_fst
192         movl    %eax, %esi          # esi = compute_fst()
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 définié dans le Tableau 4

```
        .file   "3.c"
        .text
        .globl  compute_fst
        .type   compute_fst, @function

compute_fst:
        pushq   %rbp            # remember old base pointer
        movq    %rsp, %rbp      # set new base pointer
        movq    %rdi, -24(%rbp) # put content of rdi (token) in -24(%rbp)
        movl    $0, -8(%rbp)    # pos   (-8(%rbp)) = 0
        movl    $0, -4(%rbp)    # total (-4(%rbp)) = 0

.NODE_0:
        addl    $1, -8(%rbp)    # pos++
        movl    -8(%rbp), %eax  # eax = pos
        leaq    -1(%rax), %rdx  # rdx = pos - 1
        movq    -24(%rbp), %rax # load token (address) in rax
        addq    %rdx, %rax      # rax = &(token[pos-1])
        movzbl  (%rax), %eax    # eax = token[pos-1]

        cmpl $80, %eax          # case 'P'
        je .NODE_0_P
        cmpl $84, %eax          # case 'T'
        je .NODE_0_T
        cmpl $83, %eax          # case 'S'
        je .NODE_0_S
        cmpl $77, %eax          # case 'M'
        je .NODE_4
        movl $-1, %eax          # default : return -1
        jmp .RET

.NODE_0_P:
        addl  $2, -4(%rbp)      # total += 2
        jmp .NODE_1

.NODE_0_T:
        addl  $6, -4(%rbp)      # total += 6
        jmp .NODE_1

.NODE_0_S:
        addl  $3, -4(%rbp)      # total += 3
        jmp .NODE_3

.NODE_1:
        addl    $1, -8(%rbp)    # pos++
        movl    -8(%rbp), %eax  # eax = pos
        leaq    -1(%rax), %rdx  # rdx = pos - 1
        movq    -24(%rbp), %rax # load token (address) in rax
        addq    %rdx, %rax      # rax = &(token[pos-1])
        movzbl  (%rax), %eax    # eax = token[pos-1]

        cmpl $79, %eax          # case 'O'
        je .NODE_2
        movl $-1, %eax          # default : return -1
        jmp .RET

.NODE_2:
        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 'P'
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 'T'
79          je .NODE_3_T
80          cmpl $76, %eax           # case 'L'
81          je .NODE_4
82          movl $-1, %eax           # default : return -1
83          jmp .RET
84
85  .NODE_3_T:
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 'O'
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 'T'
111         je .NODE_5_T
112         cmpl $80, %eax           # case 'P'
113         je .NODE_7
114         movl $-1, %eax           # default : return -1
115         jmp .RET
116
117 .NODE_5_T:
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 'H'
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)         # if (argc < 2)
157         jg      .DO_MAIN
158         movl    $1, %eax             # return 1;
159         jmp     .END_MAIN
160 .DO_MAIN:
161         movq    -16(%rbp), %rax      # rax = &(argv[0])
162         addq    $8, %rax             # rax = &(argv[1])
163         movq    (%rax), %rax         # rax = argv[1]
164         movq    %rax, %rdi           # rdi = argv[1]
165         call    compute_fst
166         movl    %eax, %esi           # esi = compute_fst()
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