## Première partie

# Analyse de billet de concert

# Exercice 1 : Traitement d'une commande de billets

### Spécification de Concert.1.lex : reconnaissance des champs clefs

```
1  alpha [a-zA-Z]
2  codeConcert ^T[0-9]{2,6}
3  codeDossier [0-9]{8}
4  date [0-9][0-9]\/[0-9]{1,2}(\/[0-9][0-9])?
5  FL (\n)
6  heure [0-9][0-9]:[0-9][0-9]
7  ignore [\t ]+
8  nbPlaces [1-9]?[0-9](\ places)
9  nomConcert [A-Z0-9]([A-Z0-9]|(-[A-Z0-9]))*
10 nomPropre {alpha}({alpha}|(-{alpha}))*
11 prenomNom ^{nomPropre}\/{nomPropre}
12
13
14 %%
15 {codeConcert}     {printf("codeConcert");}
16 {codeDossier}     {printf("codeDossier");}
17 {date}         {printf("date");}
18 DOSSIER            {printf("dossier");}
19 <<EOF>>         {printf("FinFichier\n"); return 0;}
20 {FL}               {printf(" FL\n");}
21 {heure}        {printf("heure");}
22 {ignore}       {printf(" ");}
23 {nbPlaces}     {printf("nb places");}
24 {nomConcert}       {printf("nomConcert");}
25 {prenomNom}   {printf("prenomNom");}
```

**Listing 1** – Première spécification en vue d'un test de reconnaissance des différents champs d'une commande de billets

**Spécification de Concert.2.lex : application de la reconnaissance à un besoin 'réel'**

```
%{
    char* codeDossier;
    char* prenomNom;
    int nbPlaces = 0;
    int nbConcerts = 0;
%}

alpha [a-zA-Z]
codeConcert ^T[0-9]{2,6}
codeDossier [0-9]{8}
date [0-9][0-9]\/[0-9]{1,2}(\/[0-9][0-9])?
FL (\n)
heure [0-9][0-9]:[0-9][0-9]
ignore [\t ]+
nbPlaces [1-9]?[0-9]
nomConcert [A-Z0-9]([A-Z0-9]|(-[A-Z0-9]))*
nomPropre {alpha}({alpha}|(-{alpha}))*
prenomNom ^{nomPropre}\/{nomPropre}


%%
{codeConcert}     {nbConcerts++;}
{codeDossier}     {codeDossier=strdup(yytext);}
{date}            {}
DOSSIER           {}
<<EOF>>           {return 0;}
{FL}              {}
{heure}           {}
{ignore}          {}
{nbPlaces}        {nbPlaces+=strtol(yytext, NULL, 10);}
{nomConcert}      {}
places            {}
{prenomNom}       {prenomNom=strdup(yytext);}

%%
int main()
{
    yylex();
    printf("Pour le dossier %s, %s a acheté %i places de %i concerts\n",
       codeDossier, prenomNom, nbPlaces, nbConcerts);
}
```

**Listing 2** – Seconde spécification appliquant la reconnaissance des différents champs d'une commande de billets

Deuxième partie

# Des automates en récursif

# Exercice 2 : Programmation en dur de manière récursive

### Questions de compréhension

**Question :** Si votre automate a N états, combien de fonctions reconnaitRec_i devez vous écrire ?

**Réponse :** Si l'automate a N états alors il faudra écrire N fonctions reconnaitRec_i. En effet, dans les faits nous sommes en train d'implanter un système d'équations aux langages.

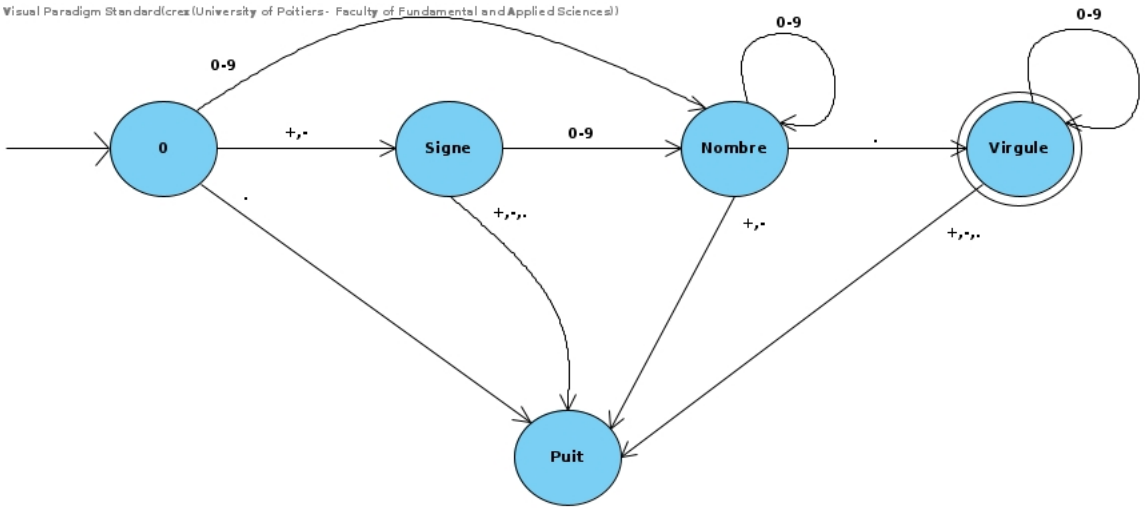
**Question :** Si l'état i est final, que doit retourner reconnaitRec_i("") ? Et si i n'est pas final ?

**Réponse :** Un état i final signifie que reconnaitRec_i("") doit retourner 'true', "" étant le mot vide aussi appelé $\epsilon$. Tout état i non final doit alors retourner 'false' pour le mot vide.

**Question :** Si le paramètre 'mot' n'est pas vide et commence par un caractère c, quelle fonction reconnaitRec_i(mot) doit-elle appeler ? Et avec quel paramètre ?

**Réponse :** Si le paramètre 'mot' n'est pas vide et commence par un caractère c alors on doit appeler la fonction reconnaitRec_i(mot) qui correspond à l'état de destination dans la transition $q_{courant} \xrightarrow{c} q_i$. On appelle alors cette fonction avec pour paramètre le mot 'mot' tronqué de sa première lettre.

### Automate des réels



**Implantation des reconnaitRec**

```
(* ------------------------ *)
(*  Fonctions Auxiliaires  *)
(* ------------------------ *)

let isCiffer c = c >= '0' && c <= '9';;
let isBinOp c = c = '+' || c = '-';;
let isComma c = c = '.';;
let explode m = List.init (String.length m) (String.get m);;


(* ------------------------ *)
(*  Fonctions Principales  *)
(* ------------------------ *)

let rec reconnaitRec_Virgule m =
  match m with
  | [] -> true
  | firstCarac::tl -> if(isCiffer firstCarac) then reconnaitRec_Virgule tl
   else false
;;

let rec reconnaitRec_Nombre m =
  match m with
  | [] -> false
  | firstCarac::tl -> if(isCiffer firstCarac) then reconnaitRec_Nombre tl
                      else if(isComma firstCarac) then reconnaitRec_Virgule
    tl
                      else false
;;

let reconnaitRec_Signe m =
  match m with
  | [] -> false
  | firstCarac::tl -> if(isCiffer firstCarac) then reconnaitRec_Nombre tl
                      else false
;;

let reconnaitRec_0 m =
    match m with
    | [] -> false
    | firstCarac::tl -> if(isCiffer firstCarac) then reconnaitRec_Nombre tl
                        else if(isBinOp firstCarac) then reconnaitRec_Signe
    tl
                        else false
;;
```

**Listing 3** – Début du code source d'automateEnDurReels.ml

### Implantation de l'automate complet

```
let reconnaitReelRec m =
  reconnaitRec_0 (explode m)
;;
```

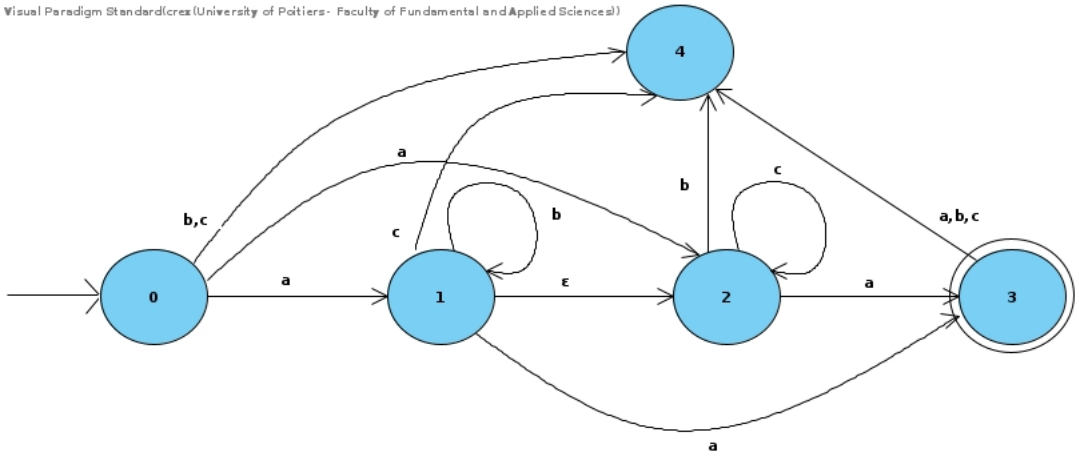
**Listing 4** – Le reste du code source d'automateEnDurReels.ml

### Programme complet

```
(* ============================================ *)
(* AUTEUR: Florian Legendre                     *)
(* OBJECTIF DE CE MODULE: Être le programme     *)
(* final qui sera compilé en un exécutable.     *)
(* Il est basé sur le travail produit dans les *)
(* deux autres .ml de ce dossier.               *)
(* ============================================ *)

open List;;
open AutomateEnDurReels;;

let main =
while true do
  let wordToRead = read_line() in
  if(reconnaitReelRec wordToRead)
  then (print_string "True!"; print_newline();)
  else (print_string "False!"; print_newline();)
done;
;;

main;;
```

**Listing 5** – programme final lisant sans cesse sur le flux d'entrée

# Exercice 3 : Des automates non déterministes représentés dans le code de manière récursive




**Questions de compréhension**

**Question :** Comment dans le code de reconnait_0 allez vous représenter le fait qu'en lisant un a, on puisse aller soit de l'état 0 à l'état 1, soit de l'état 0 à l'état 2 ?

**Réponse :** Un automate non déterministe signifie que l'on peut trouver un chemin valide (et donc reconnaître un mot) en passant soit par un chemin soit par un autre. Ce soit/soit ce représente en programmation par l'opérateur OU. Je vais donc faire pour la lettre 'a' à l'état 0 : reconnaitRec1 resteDuMot || reconnaitRec2 resteDuMot

**Question :** Comment dans le code de reconnait_1 allez vous représenter le fait que l'on peut passer directement, sans rien lire, à l'état 2 ?

**Réponse :** Similairement au cas si dessus je peux faire un OU avec directement l'appel récursif de reconnaitRec2 en donnant en paramètre de reconnaitRec2 le mot complet sans avoir "mangé" de lettre.

**Implantation et Tests de l'automate**

```
(* ------------------------ *)
(*  Fonctions Auxiliaires  *)
(* ------------------------ *)

let explode m = List.init (String.length m) (String.get m);;


(* ------------------------ *)
(*  Fonctions Principales  *)
(* ------------------------ *)

let reconnaitRec_3 m =
  match m with
  | [] -> true
  | firstCarac::tl -> false
;;

let rec reconnaitRec_2 m =
  match m with
  | [] -> false
  | firstCarac::tl -> if(firstCarac = 'c') then reconnaitRec_2 tl
                      else if(firstCarac = 'a') then reconnaitRec_3 tl
                      else false
;;

let rec reconnaitRec_1 m =
  match m with
  | [] -> false
  | firstCarac::tl -> (if(firstCarac = 'b') then reconnaitRec_1 tl
                       else if(firstCarac = 'a') then reconnaitRec_3 tl
                       else false) || reconnaitRec_2 m
;;

let reconnaitRec_0 m =
    match m with
    | [] -> false
    | firstCarac::tl -> if(firstCarac = 'a') then (reconnaitRec_1 tl) || (reconnaitRec_2 tl)
                        else false
;;

let reconnaitL4 m  =
  reconnaitRec_0 (explode m)
;;
```

**Listing 6** – Code source d'automateEnDurEx3.ml

# Exercice 4 : Évaluation du réel correspondant à la chaîne de caractères

### Questions de compréhension

**Question :** Comment allez vous gérer votre position dans la partie décimale ($x^{eme}$ position après la virgule = indique la puissance de dix négative ?), et vous en servir pour prendre en compte la nouvelle décimale lue ?

**Réponse :** La position 'x' après la virgule indique la puissance négative à réaliser sur le nombre évalué. On additionne alors le résultat de cette puissance négative avec la partie décimale déjà ainsi évaluée.

**Question :** Comment allez vous gérer le calcul de la partie entière, lorsqu'un nouveau chiffre est lu ?

**Réponse :** Lorsqu'un nouveau chiffre est lu on fait : partie entière déjà évaluée multipliée par 10 + le nouveau chiffre

**Question :** Comment allez vous gérer la transmission du calcul d'une routine récursive à l'autre ? Variables globales, paramètres d'entrée sortie, valeurs de retour de fonction ?

**Réponse :** Je code sous Ocaml. Je compte utiliser les possibilités du langage en renvoyant des tuples de retour du type (boolean, partie entière, partie décimale) et en injectant des paramètres d'entrée du type (mot, partie entière, position partie décimale, partie décimale)

### Implantation de la fonction d'évaluation des réels

```
1  (* ============================================ *)
2  (* AUTEUR: Florian Legendre                     *)
3  (* OBJECTIF DE CE MODULE: Être une banque       *)
4  (* des fonctions qui implantantent la fonction  *)
5  (* d'évaluation des réels. Les tests étant      *)
6  (* réalisés dans evalueReelsTests.ml            *)
7  (* ============================================ *)
8
9
10 (* ------------------------ *)
11 (*  Fonctions Auxiliaires   *)
12 (* ------------------------ *)
13
14 let isCiffer c = c >= '0' && c <= '9';;
15 let isBinOp c = c = '+' || c = '-';;
16 let isComma c = c = '.';;
17 let explode m = List.init (String.length m) (String.get m);;
18 let getDigit c =
19   match c with
20   | '0' -> 0 | '1' -> 1 | '2' -> 2 | '3' -> 3 | '4' -> 4
21   | '5' -> 5 | '6' -> 6 | '7' -> 7 | '8' -> 8 | '9' -> 9
22   | _ -> failwith("ERROR IN charDigitToInt => This isn't a digit")
23 ;;
```

```
24
25
26 (* ------------------------ *)
27 (*  Fonctions Principales  *)
28 (* ------------------------ *)
29
30 let rec evalueReel_Virgule m partieEnt posDec partieDec =
31   match m with
32   | [] -> (true, partieEnt, posDec, partieDec)
33   | firstCarac::tl -> if(isCiffer firstCarac)
34                       then evalueReel_Virgule tl partieEnt
35                              (posDec+1) (partieDec*10 + (getDigit
     firstCarac))
36                       else (false, 0, 0, 0)
37 ;;
38
39 let rec evalueReel_Nombre m partieEnt posDec partieDec =
40   match m with
41   | [] -> (false, 0, 0, 0)
42   | firstCarac::tl -> if(isCiffer firstCarac)
43                       then evalueReel_Nombre tl (partieEnt*10 + (getDigit
     firstCarac)) 0 0
44                       else if(isComma firstCarac)
45                       then evalueReel_Virgule tl partieEnt 0 0
46                       else (false, 0, 0, 0)
47 ;;
48
49 let evalueReel_Signe m partieEnt posDec partieDec =
50   match m with
51   | [] -> (false, 0, 0, 0)
52   | firstCarac::tl -> if(isCiffer firstCarac)
53                       then evalueReel_Nombre tl (getDigit firstCarac) 0 0
54                       else (false, 0, 0, 0)
55 ;;
56
57 let evalueReel_0 m partieEnt posDec partieDec =
58   match m with
59   | [] -> (false, 0, 0, 0)
60   | firstCarac::tl -> if(isCiffer firstCarac)
61                       then evalueReel_Nombre tl (getDigit firstCarac) 0 0
62                       else if(isBinOp firstCarac)
63                       then evalueReel_Signe tl 0 0 0
64                       else (false, 0, 0, 0)
65 ;;
66
67 let evalueReel m =
68   let (recBoolean, partieEnt, posDec, partieDec) = evalueReel_0 (explode m)
       0 0 0 in
69   let resEval = (float_of_int partieEnt)
70                 +. (float_of_int partieDec) /. 10.**(float_of_int posDec)
     in
71
72   if(recBoolean && m.[0] = '-')
73   then (recBoolean, (-1. *. resEval))
74   else (recBoolean, resEval)
75 ;;
```

```
76
77
78 (* Note:
79 Pas d'inquiétude pour la "division par 0", cf. cas où il n'y a pas de
80 nombre après la virgule, car 10^0 = 1 *)
```

**Listing 7** – Code source d'evalueReels.ml

### Implantation du programme final d'évaluation des réels

```
1  (* ========================================== *)
2  (* AUTEUR: Florian Legendre                   *)
3  (* OBJECTIF DE CE MODULE: Être le programme   *)
4  (* final qui sera compilé en un exécutable.   *)
5  (* Il est basé sur le travail produit dans les *)
6  (* deux autres .ml de ce dossier.             *)
7  (* ========================================== *)
8
9  open EvalueReels;;
10
11 let main =
12 while true do
13   let wordToRead = read_line() in
14   let (isRec, res) = evalueReel wordToRead in
15   if(isRec)
16   then (print_string "True! Value is "; print_float res; print_newline();)
17   else (print_string "False!"; print_newline();)
18 done;
19 ;;
20
21 main;;
```

**Listing 8** – Code source du programme final d'évaluation des réels