Concours Mines MP 2007
Epreuve d'informatique : un corrigé

1 Expressions rationnelles.

1. On a les équivalences suivantes

$$(\varepsilon + \varepsilon) \equiv \varepsilon$$
$$(e_1 + \emptyset) \equiv e_1$$
$$\varepsilon.\emptyset \equiv \emptyset$$
$$(e_1.\varepsilon) \equiv e_1$$
$$((e_1 + \varepsilon) + (e_2 + \varepsilon)) \equiv ((e_1 + e_2) + \varepsilon)$$
$$(e_1.(e_2 + \varepsilon)) \equiv ((e_1.e_2) + e_1)$$
$$((e_1 + \varepsilon).(e_2 + \varepsilon)) \equiv (((e_1.e_2) + (e_1 + e_2)) + \varepsilon)$$

2. On procède par récurrence de construction. Il convient de montrer que la propriété

H_e : "e est équivalente à une expression rationnelle du type voulu"

est vérifiée pour toute expression rationnelle e.

- C'est vrai dans les cas de base où $e = \in \{\varepsilon, \emptyset\} \cup \Sigma$ (e est alors équivalente à elle même).
- Supposons que $e = (f_1 + f_2)$ avec H_{f_1} et H_{f_2} vraies. Il existe des expressions rationnelles e_1, e_2 équivalentes à f_1 et f_2 et ayant le type voulu. e équivaut alors à $(e_1 + e_2)$. Quatre cas sont envisageables pour e_1 et e_2 et il y a donc 16 possibilités. Par symétrie des rôles joués par e_1 et e_2, tous ne sont pas à étudier.
  - Si $e_2 = \emptyset$ alors $e \equiv e_1$ qui est du type voulu.
  - Si $e_2 = \varepsilon$ et
    - si $e_1 = \varepsilon$ alors $e \equiv \varepsilon$ qui est du type voulu.
    - si e_1 est sans ε et $\emptyset$, $e \equiv (e_1 + \varepsilon)$ qui est du type voulu
    - si $e_1 = (e'_1 + \varepsilon)$ où e'_1 est sans ε et $\emptyset$, $e \equiv (e_1 + \varepsilon)$ qui est du type voulu.
  - Si e_2 est sans ε et $\emptyset$ et
    - si e_1 est sans ε et $\emptyset$, $e \equiv (e_1 + e_2)$ qui est du type voulu
    - si $e_1 = (e'_1 + \varepsilon)$ où e'_1 est sans ε et $\emptyset$, $e \equiv ((e'_1 + e_2) + \varepsilon)$ qui est du type voulu.
  - Si $e_2 = (e'_2 + \varepsilon)$ et $e_1 = (e'_1 + \varepsilon)$ où e'_2 et e'_1 sont sans ε et $\emptyset$, alors $e \equiv ((e'_1 + e'_2) + \varepsilon)$ qui est du type voulu.
- Supposons que $e = (f_1.f_2)$ avec H_{f_1} et H_{f_2} vraies. Il existe des expressions rationnelles e_1, e_2 équivalentes à f_1 et f_2 et ayant le type voulu. e équivaut alors à $(e_1.e_2)$.
  - Si $e_2 = \emptyset$ alors $e \equiv \emptyset$ qui est du type voulu.
  - Si $e_1 = \emptyset$ alors $e \equiv \emptyset$ qui est du type voulu.
  - Si $e_2 = \varepsilon$ alors $e \equiv e_1$ qui est du type voulu.
  - Si $e_1 = \varepsilon$ alors $e \equiv e_2$ qui est du type voulu.
  - Si e_2 est sans ε et $\emptyset$ et
    - si e_1 est sans ε et $\emptyset$, $e \equiv (e_1.e_2)$ qui est du type voulu
    - si $e_1 = (e'_1 + \varepsilon)$ où e'_1 est sans ε et $\emptyset$, $e \equiv ((e'_1.e_2) + e_2)$ qui est du type voulu.
  - Si $e_2 = (e'_2 + \varepsilon)$ où e'_2 est sans ε et $\emptyset$ et
    - si e_1 est sans ε et $\emptyset$, $e \equiv ((e_1.e'_2) + e_1)$ qui est du type voulu
    - si $e_1 = (e'_1 + \varepsilon)$ où e'_1 est sans ε et $\emptyset$, $e \equiv (((e'_1.e'_2) + (e'_1 + e'_2)) + \varepsilon)$ qui est du type voulu.
- Supposons que $e = (f_1)^*$ avec H_{f_1} vraie. Il existe une expression rationnelle e_1 équivalente à f_1 et ayant le type voulu. e équivaut alors à $(e_1)^*$.
  - Si $e_1 = \emptyset$ alors $e \equiv \varepsilon$ qui est du type voulu.

- Si $e_1 = \varepsilon$ alors $e \equiv \varepsilon$ qui est du type voulu.
- Si e_1 est sans $\emptyset$ et ε alors $e \equiv (e_1)^*$ qui est du type voulu.
- Si $e_1 = (e'_1 + \varepsilon)$ où e'_1 est sans ε et $\emptyset$, alors $e \equiv (e'_1)^*$ qui est du type voulu.

Dans tous les cas H_e est vraie.

3. On doit voir si l'argument prend l'une des valeurs caractérisant les symboles.

```
let est_symbole n =
(n = ETOILE) or (n = PLUS) or (n = POINT) or (n = P_O) or (n=P_F) ;;
```

4. On se déplace dans le tableau (l'indice i est le numéro de la case à inspecter) en gérant une référence donnant le compte de parenthèses (nombre de (moins nombre de) rencontrées). On continue à se déplacer dans le tableau tant que l'on n'a pas atteint la fin OU que l'une des conditions de l'énoncé soit mise en défaut (compte de parenthèse ou contenu de la case).
En fin de boucle, on a trouvé une case convenable si on n'a pas été au bout du tableau.

```
let cesure expr debut fin =
let i=ref (debut+1)  (* num\'ero de la case \`a inspecter *)
and p=ref 0 in       (* compte de parenth\`ese *)
while !i<fin & (!p<>0 or (expr.(!i)<>PLUS & expr.(!i)<>POINT)) do
   if expr.(!i)=P_O then incr p
   else if expr.(!i)=P_F then decr p ;
   incr i
   done ;
if !i=fin then (-1) else !i ;;
```

Remarque : l'énoncé me semble imprécis. Il parle de "la partie du tableau située de l'indice $(debut + 1)$ *inclus à l'indice* $(i - 1)$ *inclus". Si* $i = debut + 1$*, ceci est ambigu et j'ai considéré que, dans ce cas, la condition correspondante est vérifiée. Supposons que l'expression soit* $e = (.)$*. L'appel de* `cesure` *avec* $debut = 0$ *et* $fin = 2$ *renverra ici* 1.

5. Le cas de base est celui où le bout d'expression ne possède qu'un seul élément ($debut = fin$). C'est alors une expression rationnelle si cet élément est une lettre (c'est à dire si ce n'est pas un symbole).
Dans le cas contraire ($debut < fin$), l'expression est rationnelle si elle est de l'un des types $(e_1 + e_2)$ ou $(e_1.e_2)$ ou $(e)^*$. On distingue donc deux cas :
- si $expr.(fin) = ETOILE$, on teste si il y a une parenthèse ouvrante au début, une parenthèse fermante avant l'étoile et si ce qu'il y a au milieu est une expression rationnelle
- sinon, on teste si on commence par (, si on finit par), on cherche, avec `cesure`, la position du symbole et, si on l'a trouvé, on teste e_1 et e_2.

Dans un cas comme celui évoqué dans la remarque en fin de question 4, on risque de faire un appel récursif avec $debut = 1$ et $fin = 0$. Si on tombe sur un tel cas, on n'est pas en présebce d'une expression rationnelle. Ceci explique le cas envisagé ci-dessous $debut > fin$ où l'on renvoie `false` (appel récursif "illégal" et on n'a pas une expression rationnelle).

```
let rec est_rationnelle expr debut fin =
if debut > fin then false
else if debut = fin then not (est_symbole expr.(debut))
else if expr.(fin)=ETOILE then
          expr.(debut)=P_O
        & expr.(fin-1)=P_F
        & (est_rationnelle expr (debut+1) (fin-2))
else begin
     let i=cesure expr debut fin in
       expr.(debut) = P_O
     & expr.(fin) = P_F
     & (i<> -1)
     & (est_rationnelle expr (debut +1) (i-1))
```

```
    & (est_rationnelle expr (i+1) (fin-1))
    end ;;
```

6. Les expressions considérées ne contenant pas le symbole ε, une expression qui ne contiendrait pas le symbole $*$ ne saurait contenir ε (récurrence de construction). Le principe de l'algorithme récursif demandé est alors le suivant.
- Dans un cas de base (un seul symbole qui est une lettre), on renvoie `flase`.
- Si $e = (f)^*$, on renvoie `true`.
- Si $e = (e_1 + e_2)$, on teste si le langage décrit par e_1 OU celui décrit par e_2 contient ε (appels récursifs).
- Si $e = (e_1.e_2)$, on teste si le langage décrit par e_1 ET celui décrit par e_2 contiennent ε (appels récursifs).

7. La fonction est très proche de celle de la question 5. Comme on SUPPOSE que l'argument est une expression rationnelle, on n'a plus les tests à faire. Les appels récursifs permettent de construire le ou les fils de la racine.

```
let rec expression_vers_arbre expr debut fin =
if debut = fin then Feuille (expr.(debut))
else if expr.(fin)=ETOILE then
        Unaire_ETOILE (expression_vers_arbre expr (debut+1) (fin-2))
else begin
     let i=cesure expr debut fin in
     if expr.(i)=POINT then
       Binaire_POINT ( (expression_vers_arbre expr (debut +1) (i-1)) ,
                       (expression_vers_arbre expr (i+1) (fin-1)) )
     else Binaire_PLUS ( (expression_vers_arbre expr (debut +1) (i-1)) ,
                       (expression_vers_arbre expr (i+1) (fin-1)) )
     end ;;
```

8. Il suffit de suivre le programme décrit en question 6.

```
let rec contient_epsilon arbre =
  match arbre with
  Feuille a -> false
| Unaire_ETOILE e -> true
| Binaire_PLUS (e,f) -> (contient_epsilon e) or (contient_epsilon f)
| Binaire_POINT (e,f) -> (contient_epsilon e) & (contient_epsilon f) ;;
```

2 Langages locaux.

9. Soit L_1 le langage local décrit par le quadruplet.

$$(\{a\}, \{a\}, \{aa\}, false)$$

L_1 ne contient que des mots ayant la lettre a et pas le mot vide. On a donc L_1 qui est inclus dans le langage décrit par $(a(a)^*)$. Réciproquement, tout mot de cet ensemble est non vide, commence et se finit par a et deux lettres consécutives sont des a : cet ensemble est égal à L_1.
Soit L_2 le langage local décrit par le quadruplet

$$(\{a\}, \{b\}, \{ab, ba\}, true)$$

- Si $u \in L_2 \setminus \{\varepsilon\}$, u commence par a, se termine par b, seul un a peut suivre un b et seul un b peut suivre un a. u s'écrit donc $abab\ldots ab = (ab)^n$ où $n \in \mathbb{N}^*$. Le mot $\varepsilon = (ab)^0$ étant dans L_2, on a $L_2 = \{(ab)^n / \ n \in \mathbb{N}\} = ((a.b))^*$.

- Réciproquement, dans tout mot non vide de $((a.b))^*$, la première lettre est a, la dernière est b, seul un a peut suivre un b et seul un b peut suivre un a (et donc les seuls facteurs de longueur 2 possibles sont ab et ba).

10.

- $(a(a)^*)$ et $((a.b))^*$ sont des expression qui décrivent des langages locaux L_1 et L_2 (question précédente). Soit $L = L_1 \cup L_2$; supposons que L soit local décrit par le quadruplet (I, F, P, α). Les mots non vides de L commencent par a et donc $I = \{a\}$. De même, $F = \{a, b\}$ (il existe des mots de L se terminant par a et d'autres par b). aa est un facteur d'un mot de L ainsi que ab et ba et donc $\{aa, ab, ba\} \subset P$. Le mot $abaa$ est non vide, il commence par a, se termine par a, ne contient que des facteurs de longueur 2 dans P et devrait donc être dans L ce qui n'est pas le cas.
  Ainsi, la réunion de deux langages locaux n'est pas forcément un langage local.
- Soient L et L' des langages locaux décrits par les quadruplets (I, F, P, α) et (I', F', P', α'). Soit M le langage local décrit par le quadruplet $(I \cap I', F \cap F', P \cap P', \alpha \wedge \alpha')$. Si un mot non vide est dans M, il commence par un élément de $I \cap I'$, se finit par un élément de $J \cap J'$ et ne contient que des facteurs de longueur 2 dans $P \cap P'$. A fortiori, il est dans $L \cap L'$. Réciproquement, si un mot non vide est dans $L \cap L'$ alors il commence par un élément de I et de I' (et donc de $I \cap I'$), se termine par un élément de J et de J' (et donc de $J \cap J'$) et ses facteurs de longueur 2 sont tous dans P et P' (et donc dans $P \cap P'$). Il est donc dans M. On a montré que $L \cap L' \setminus \{\varepsilon\} = M \setminus \{\varepsilon\}$. Comme $\varepsilon \in M \iff (\alpha \wedge \alpha') = true \iff (\alpha = \alpha' = true) \iff \varepsilon \in L \cap L'$, on a $M = L \cap L'$.
  Ainsi, l'intersection de deux langages locaux est un langage local.
- $(a(a)^*)$ et $((a.b))^*$ sont des expression qui décrivent des langages locaux L_1 et L_2 (question précédente). Soit $L = L_1.L_2$; supposons que L soit local décrit par le quadruplet (I, F, P, α). Les mots non vides de L commencent par a et donc $I = \{a\}$. De même, $F = \{a, b\}$ (il existe des mots de L se terminant par a et d'autres par b). ab est un facteur d'un mot de L et donc $\{ab\} \subset P$. Le mot ab est non vide, il commence par a, se termine par b, ne contient que des facteurs de longueur 2 dans P et devrait donc être dans L ce qui n'est pas le cas (il n'est pas la concaténée d'un mot de L_1 et d'un mot de L_2).
  Ainsi, la concaténation de deux langages locaux n'est pas forcément un langage local.
- Soit L un langage local décrit par le quadruplet (I, F, P, α). Soit M le langage local décrit par le quadruplet $(I, F, P', true)$ avec $P' = P \cup I \times F$.

  - Soit u un mot non vide de L^*. Il s'écrit $u = m_1 \dots . m_k$ avec $m_i \in L$ et $m_i \neq \varepsilon$. Il commence donc, comme m_1 par un élément de I et se termine, comme m_k, par un élément de F. Un facteur de longueur 2 de u est un facteur de longueur 2 d'un m_i ou est composé d'une dernière lettre de m_i et d'une première lettre de m_{i+1}. Tous ces facteurs sont dans P'. On a donc $u \in M$ et ainsi, $L^* \setminus \{\varepsilon\} \subset M$. Comme $\varepsilon \in M$, on a aussi $L^* \subset M$.
  - On montre par récurrence sur sa longueur qu'un mot u de M est dans L^*.
    - Si u est vide, il est dans L^*.
    - Supposons le résultat vrai pour un mot de longueur $\leq n$ (où $n \geq 0$). Soit u un mot non vide de M de longueur $n + 1$. Si tous les facteurs de longueur 2 de u sont dans P alors $u \in L \subset L^*$. Sinon, soit $u_i u_{i+1}$ le premier facteur qui n'est pas dans P (on a alors forcément $u_i \in F$ et $u_{i+1} \in I$). Alors $v = u_1 \dots u_i$ est dans L et $w = u_{i+1} \dots u_{n+1}$ est un mot de M de longueur $\leq n$ et donc dans L^* par hypothèse de récurrence. Ainsi, $u = v.w \in L.L^* \subset L^*$.

    On a donc aussi $M \subset L^*$.

  Ainsi, l'itération d'un langage local est un langage local.

11.

- Le langage des mots dont la lettre initiale est dans I est $I.\Sigma^*$.
- Le langage des mots dont la lettre finale est dans F est $\Sigma^*.F$.
- Le langage des mots de longueur 2 qui ne sont pas dans P est $(\Sigma.\Sigma) \cap (\Sigma^* \setminus P)$.
- Le langage des mots contenant au moins un facteur de longueur 2 qui n'est pas dans P est

$\Sigma^*.\left((\Sigma.\Sigma) \cap (\Sigma^* \setminus P)\right).\Sigma^*$.

- Le langage $L \setminus \{\varepsilon\}$ est alors

$$(I.\Sigma^* \cap \Sigma^*.F) \cap (\Sigma^* \setminus (\Sigma^*.\left((\Sigma.\Sigma) \cap (\Sigma^* \setminus P)\right).\Sigma^*))$$

12. L'ensemble des langages rationnels est stable par intersection, concaténation, itération, complémentation. De plus, tout langage fini est rationnel (donc Σ, P, I, F le sont). La formule précédente montre donc si L est local, $L \setminus \{\varepsilon\}$ est rationnel. $\{\varepsilon\}$ étant rationnel et l'ensemble des langages rationnels étant stable par réunion, L est aussi rationnel (il reste à ajouter ε si nécessaire).

13. Si la première lettre n'est pas dans I ou si la dernière lettre n'est pas dans F, on renvoie `false`. Sinon, on parcourt le mot en testant tous les facteurs $u_{k-1}u_k$ jusqu'à épuiser le mot ou tomber sur un facteur non convenable.

```
let appartient mot I F P =
let l=vect_length mot in
if I.(mot.(0)) & F.(mot.(l-1)) then begin
     let k=ref 1 in
     while (!k<l) & P.(mot.(!k-1)).(mot.(!k)) do incr k done ;
     (!k=l)  (* tous les facteurs rencontr\'es sont dans P *)
     end
else false ;;
```

3 Mots appartenant au langage décrit par une expression rationnelle.

14. Commençons par une remarque sur la réunion ou la concaténation de deux langages locaux particuliers. Soient L_1 et L_2 des langages locaux sur des alphabets **disjoints** Σ_1 et Σ_2. Ils sont caractérisés par des quadruplets $(I_1, F_1, P_1, \alpha_1)$ et $(I_2, F_2, P_2, \alpha_2)$.

- Le langage local R sur l'alphabet $\Sigma_1 \cup \Sigma_2$ caractérisé par le quadruplet $(I_1 \cup I_2, F_1 \cup F_2, P_1 \cup P_2, \alpha_1 \vee \alpha_2)$ est exactement $L_1 \cup L_2$. En effet, le mot vide ne pose pas de problème (on a bien choisi le booléen) et
  - si u est un mot non vide de L_1 ou L_2, il est clairement dans R
  - si u est un mot non vide de R, il s'écrit $u_1 \dots u_k$; si $u_1 \in I_1$, on montre que toutes les lettres sont dans Σ_1 (sinon, on aurait un facteur de taille 2 non convenable), que la dernière lettre est donc dans F_1, que les facteurs de taille 2 sont dans P_1 et qu'ainsi u est dans L_1 ; sinon $u_1 \in I_2$ et alors, de même, $u \in L_2$.
- Si $\alpha_1 = \alpha_2 = false$, le langage local sur l'alphabet $\Sigma_1 \cup \Sigma_2$ caractérisé par le quadruplet $(I_1, F_2, P_1 \cup P_2 \cup F_1 \times I_2, false)$ est exactement $L_1.L_2$. En effet, le mot vide ne pose pas de problème (on a bien choisi le booléen) et
  - si u est un mot non vide de $L_1.L_2$, il est clairement dans R (car le mot vide n'est ni dans L_1 ni dans L_2)
  - si u est un mot non vide de R, il s'écrit $u_1 \dots u_k$ avec $u_1 \in I_1$ et $u_k \in F_2$; il existe une "première lettre" u_i qui n'est pas dans Σ_1 et alors $u = (u_1 \dots u_{i-1})(u_i \dots u_k)$ avec forcément $u_{i-1} \in F_1$ et $u_i \in I_2$ (pour que le facteur de longuer 2 soit acceptable). On montre alors aisément que u est concaténée d'un mot de L_1 et d'un mot de L_2.
- Si $\alpha_1 = true$ et $\alpha_2 = false$, le langage local sur l'alphabet $\Sigma_1 \cup \Sigma_2$ caractérisé par le quadruplet $(I_1 \cup I_2, F_2, P_1 \cup P_2 \cup F_1 \times I_2, false)$ est exactement $L_1.L_2$ (preuve similaire).
- Si $\alpha_1 = false$ et $\alpha_2 = true$, le langage local sur l'alphabet $\Sigma_1 \cup \Sigma_2$ caractérisé par le quadruplet $(I_1, F_1 \cup F_2, P_1 \cup P_2 \cup F_1 \times I_2, false)$ est exactement $L_1.L_2$ (preuve similaire).
- Si $\alpha_1 = \alpha_2 = true$, le langage local sur l'alphabet $\Sigma_1 \cup \Sigma_2$ caractérisé par le quadruplet $(I_1 \cup I_2, F_1 \cup F_2, P_1 \cup P_2 \cup F_1 \times I_2, true)$ est exactement $L_1.L_2$ (preuve similaire).

On peut alors montrer le résultat demandé par récurrence sur la taille des expressions rationnelles. L'hypothèse de récurrence est H_n : "si e est une expression rationnelle de taille n sur Σ dans laquelle toutes les lettres sont distinctes, alors le langage L décrit par e est local".

- Si l'expression est de taille 1, c'est une lettre a (les symbole ε et $\emptyset$ étant exclus) et le langage associé est local caractérisé par le quadruplet $(\{a\}, \{a\}, \emptyset, false)$.
- Supposons le résultat vrai jusqu'à un rang $n \geq 1$. Soit e une expression de taille $n+1$ comme décrite. Trois cas sont envisageables.
  - $e = (e_1 + e_2)$ et on peut appliquer l'hypothèse de récurrence à e_1 et e_2 et utiliser le début de la question pour conclure.
  - $e = (e_1.e_2)$ et on peut appliquer l'hypothèse de récurrence à e_1 et e_2 et utiliser le début de la question pour conclure.
  - $e = (e_1)^*$ et on peut appliquer l'hypothèse de récurrence à e_1 et utiliser la stabilité par itération (question 10) pour conclure.

  On a conclu dans tous les cas et l'hypothèse est vraie au rang $n+1$.

15. On a

$$I(expression1) = \{a, b\}$$
$$F(expression1) = \{c, b\}$$
$$P(expression1) = \{ac, ca\}$$
$$\alpha(expression1) = vrai$$

16. On a

$$I(\{a\}) = \{a\},\ I((e_1)^*) = I(e_1),\ I((e_1 + e_2)) = I(e_1) \cup I(e_2)$$

Pour $I((e_1.e_2))$ contient $I(e_1)$ et éventuellement aussi les éléments de $I(e_2)$ si ε est dans le langage associé à e_1. Ceci nous donne la fonction demandée.

```
let rec calcul_I arbre I =
   match arbre with
   Feuille a -> I.(a)<- true
 | Unaire_ETOILE e -> calcul_I e I
 | Binaire_PLUS (e,f) -> calcul_I e I ; calcul_I f I
 | Binaire_POINT (e,f) -> calcul_I e I ;
                          if contient_epsilon e then calcul_I f I ;;
```

17. On fait décrire à (i, j) l'ensemble $[0..dim - 1]^2$ en mettant à jour les cases de `produit`. Pour éviter quelques itérations, on n'entame la boucle sur j que si $T1.(i) = true$.

```
let ajout_couples produit T1 T2 =
let dim=vect_length T1 in
for i=0 to (dim-1) do
   if T1.(i) then
      for j=0 to (dim-1) do produit.(i).(j)<-produit.(i).(j) or T2.(j) done ;
   done ;;
```

18. $\{a\}$ ne contient aucun facteur de taille 2.

Les facteurs de $(e_1 + e_2)$ sont ceux de e_1 plus ceux de e_2.

Les facteurs de taille 2 de $(e_1.e_2)$ sont ceux de e_1 plus ceux de e_2 plus les ab avec a lettre finale de e_1 et b lettre initiale de e_2.

Les facteurs de taille 2 de (e^*) sont ceux de e plus les ab avec a lettre finale de e et b lettre initiale de e.

```
let rec calcul_P arbre P =
   match arbre with
   Feuille a -> ()
 | Unaire_ETOILE e ->
         let dim=vect_length P.(0) in
```

```
          let I=make_vect dim false in
          let F=make_vect dim false in
          calcul_I e I;
          calcul_F e F;
          calcul_P e P ;
          ajout_couples P F I
  | Binaire_PLUS (e,f) -> calcul_P e P ; calcul_P f P
  | Binaire_POINT (e,f) ->
          let dim=vect_length P.(0) in
          let I=make_vect dim false in
          let F=make_vect dim false in
          calcul_F e F;
          calcul_I f I;
          calcul_P e P ;
          calcul_P f P ;
          ajout_couples P F I ;;
```

19. Soit e une expression rationnelle sur Σ dont toutes les lettres sont distinctes. Le langage L décrit par e est alors un langage local décrit par le quadruplet $(I(e), F(e), P(e), \alpha(e))$. On sait obtenir l'arbre représentant e. On vient alors de voir comment calculer $I(e), F(e), P(e)$. La question 13 permet alors de déterminer si u est dans L.

```
let est_dans expr n mot =  (* n est le nombre de lettres *)
let l=vect_length expr in
let arbre = expression_vers_arbre expr 0 (l - 1) in
let I=make_vect n false in
let F=make_vect n false in
let P=make_matrix n n false in
(calcul_I arbre I);(calcul_F arbre F);(calcul_P arbre P);
appartient mot I F P ;;
```

20. On crée un tableau `lu` tel que $l.(i) = true$ quand la lettre numéro i a été rencontrée. Initialement, toutes les cases de ce tableau valent $false$. On parcourt alors `expr` de gauche à droite en continuant tant que l'on n'est pas au bout et que l'on ne rencontre pas une lettre déjà lue. Si on parvient au bout de `expr`, c'est qu'aucune lettre n'a été rencontrée deux fois.

```
let lettres_distinctes expr n =
let lu=make_vect n false in
let l=vect_length expr in
let i=ref 0 in
while !i<l & ((est_symbole expr.(!i)) or (not lu.(expr.(!i)))) do
    if not (est_symbole expr.(!i)) then lu.(expr.(!i))<-true;
    incr i
    done;
!i=l ;;
```

21. On suppose e représentée par un tableau `expr`.

En parcourant `expr` de gauche à droite et en gérant un compteur indiquant le nombre de lettres (distinctes ou pas) rencontrées, on va pouvoir obtenir un tableau `expr'` représentant e' et construire un tableau `phi` correspondant à l'apllication ϕ ($\Sigma' = \{0, 1, \ldots, q-1\}$ et `phi.(i)` contient $\phi(i)$).

```
let cree_tab expr=
let expr'=copy_vect expr in
let c=ref 0 in  (* compteur de lettres rencontr\'ees *)
let l=ref [] in
for i=0 to (vect_length expr - 1) do
```

```
   if not (est_symbole expr.(i)) then begin
          expr'.(i) <- !c ;
          l:= expr.(i)::(!l) ;
          incr c
          end
   done ;
expr',vect_of_list (rev !l) ;;
```

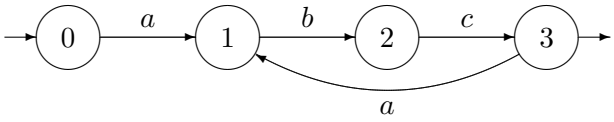
A partir de `phi`, il est facile de créer un tableau `psi` tel que `psi.(i)` contienne l'ensemble des antécédents de i par ϕ.

```
let inverse phi=
let n=ref 0 in
for i=0 to (vect_length phi -1 ) do n:=max !n phi.(i) done ;
let psi=make_vect (!n+1) [] in
for i=0 to (vect_length phi -1 ) do
    psi.(phi.(i)) <- i::(psi.(phi.(i))) done ;
psi ;;
```

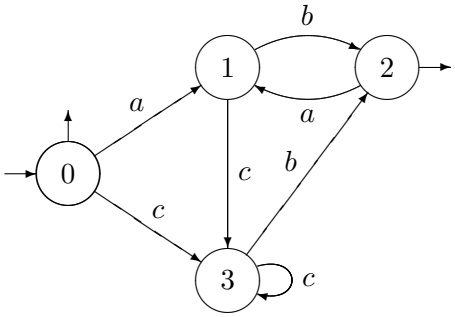
Etant donné un mot $u = u_1 \dots u_p$ (non vide), il reste à tester pour chaque y_i antécédent de u_i par ϕ si $y_1 \dots y_p$ est dans le langage reconnu par e' (ce que l'on sait faire car dans e' une même lettre n'apparaît pas deux fois). L'algorithme est bien sûr de complexité infernale.

4 Algorithme de Glushkov.

22.



23.



24. On va décrire un automate $(\Sigma, Q, T, \mathcal{I}, \mathcal{F})$.
On suppose les éléments de Σ numérotés de 1 à n $(u_1, \dots, u_n)$. L'ensemble des états est $Q = \{0, 1, \dots, n\}$, on a $\mathcal{I} = \{0\}$ (un unique état de départ) et $\mathcal{F}$ contient les numéros des éléments de F ainsi que 0 si $\alpha = true$. Les éléments de T sont les suivants
- $(0, u_i, i)$ quand $u_i \in I$;
- (q, u_i, i) quand $u_q u_i \in P$.

25. On sait obtenir e' à partir de e et l'application ϕ correspondante et on a vu comment calculer $I(e')$, $F(e')$, $P(e')$ et $\alpha(e')$. La question précédente nous permet de construire un automate $\mathcal{A}'$ reconnaissant le langage local L' correspondant au quadruplet $(I(e'), F(e'), P(e'), \alpha(e'))$.
Soit $\mathcal{A}$ l'automate obtenu à partir de $\mathcal{A}'$ en remplaçant l'étiquette i d'une transition par $\phi(i)$. On obtient un automate d'alphabet Σ. Montrons qu'il reconnaît le langage L décrit par e.
- Supposons que $u = x_1 \dots x_n$ est reconnu par $\mathcal{A}$. Il existe donc un calcul réussi dans $\mathcal{A}$ étiqueté par u. Ce calcul correspond à un autre dans $\mathcal{A}'$ qui est réussi (on n'a pas changé la nature

des états) et qui est étiqueté par $y_1 \dots y_n$ tel que $\phi(y_i) = x_i$. Le mot $y_1 \dots y_n$ est dans L' et d'après le résultat admis en fin de partie 3, u est dans L.

- La réciproque est identique et utilise la fait qu'ily équivalence dans le résultat admis.

26. On a

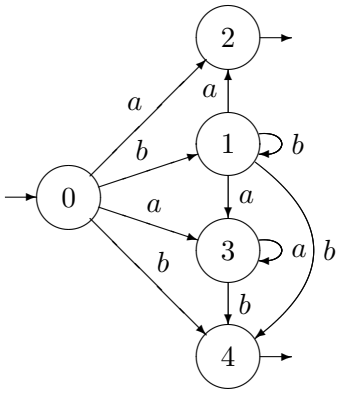
$$expression2' = ((A)^*.(B + ((C)^*.D)))$$

$$I(expression2') = \{A, B, C, D\}$$

$$F(expression2') = \{B, D\}$$

$$P(expression2') = \{(A, A), (A, B), (A, C), (A, D), (C, C), (C, D)\}$$

$$\alpha(expression2') = false$$



27. Le tableau des transitions est

	0	1	2	3	4
a	$2,3$	$2,3$		3	
b	$1,4$	$1,4$		4	

L'algorithme de déterminisation par méthode des sous-ensembles donne la table suivante

	0	$2,3$	$1,4$	3	4
a	$2,3$	3	$2,3$	3	
b	$1,4$	4	$1,4$	4	

et l'automate est donc