ÉCOLE NATIONALE DES PONTS ET CHAUSSÉES,
ÉCOLES NATIONALES SUPÉRIEURES DE L'AÉRONAUTIQUE ET DE L'ESPACE,
DE TECHNIQUES AVANCÉES, DES TÉLÉCOMMUNICATIONS,
DES MINES DE PARIS, DES MINES DE SAINT-ÉTIENNE, DES MINES DE NANCY,
DES TÉLÉCOMMUNICATIONS DE BRETAGNE,
ÉCOLE POLYTECHNIQUE (FILIÈRE TSI)

CONCOURS D'ADMISSION 2007

ÉPREUVE D'INFORMATIQUE

Filière MP
Durée de l'épreuve : 3 heures.
L'usage de calculettes est autorisé. L'utilisation d'un ordinateur est interdite.

Sujet mis à disposition des concours : ENSAE (Statistique), ENSTIM, TPE-EIVP, Cycle international
Les candidats sont priés de mentionner de façon apparente sur la première page de la copie :

INFORMATIQUE - MP

L'énoncé de cette épreuve comporte 12 pages.

RECOMMANDATIONS AUX CANDIDATS

- **Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.**
- **Tout résultat fourni dans l'énoncé peut être utilisé pour les questions ultérieures même s'il n'a pas été démontré.**
- **Il ne faut pas hésiter à formuler les commentaires qui semblent pertinents même lorsque l'énoncé ne le demande pas explicitement.**

COMPOSITION DE L'ÉPREUVE

L'épreuve comporte un seul problème, pages 2 à 12.

Expressions rationnelles, langages locaux, algorithme de Glushkov

Chaque partie du problème peut dépendre des parties précédentes. Les indications données dans une partie sont souvent utiles pour les parties suivantes.

Un **alphabet** Σ est un ensemble fini d'éléments appelés **lettres**. Un **mot** sur Σ est une suite finie de lettres de Σ ; la **longueur** d'un mot est le nombre de lettres le composant ; le mot de longueur nulle est noté ε ; une lettre de Σ est aussi un mot de longueur 1. On désigne par Σ^* l'ensemble des mots sur Σ, y compris le mot ε. Un **langage** est une partie de Σ^*. Soit $u = x_1x_2\ldots x_p$ un mot de longueur $p \geq 1$, avec $x_i \in \Sigma$ pour tout i compris entre 1 et p ; x_1 est la **lettre initiale** de u et x_p est sa **lettre finale** ; par ailleurs, si i et j sont deux entiers vérifiant $1 \leq i \leq j \leq p$, le mot $v = x_ix_{i+1}\ldots x_j$ est un **facteur** de u.

La concaténation de deux langages L_1 et L_2 est notée $L_1.L_2$. L'intersection de deux langages L_1 et L_2 est notée $L_1 \cap L_2$. L'union de deux langages L_1 et L_2 est notée $L_1 \cup L_2$. Le complémentaire d'un langage L par rapport à Σ^* est notée $\overline{L}$. La différence ensembliste est notée avec le signe '–'. Le cardinal d'un ensemble E quelconque est noté $|E|$.

On suppose dans tout le problème que l'alphabet Σ ne contient aucun des sept symboles ci-dessous :

*	+	.	(	)	$\varnothing$	ε

Préliminaire concernant la programmation : il faudra écrire des fonctions ou des procédures à l'aide d'un langage de programmation qui pourra être soit **Caml**, soit **Pascal**, tout autre langage étant exclu. **Indiquer en début de problème le langage de programmation choisi ; il est interdit de modifier ce choix au cours de l'épreuve**. Certaines questions du problème sont formulées différemment selon le langage de programmation ; cela est indiqué chaque fois que cela est nécessaire. Lorsque le candidat écrira une fonction ou une procédure, il pourra faire appel à une autre fonction ou procédure définie dans les questions précédentes. Enfin, si les paramètres d'une fonction ou d'une procédure à écrire sont supposés vérifier certaines hypothèses, il ne sera pas utile dans l'écriture de cette fonction ou de cette procédure de tester si les hypothèses sont bien vérifiées.

Dans l'énoncé du problème, un même identificateur écrit dans deux polices de caractères différentes désignera la même entité mais du point de vue mathématique pour une police (en italique ; par exemple a) et du point de vue informatique pour l'autre (en romain ; par exemple `a`).

Première partie : expressions rationnelles

On appelle **expression rationnelle** sur l'alphabet Σ toute formule obtenue récursivement à partir des règles suivantes :

- $\varnothing$ et ε sont des expressions rationnelles ;
- pour tout a dans Σ, a est une expression rationnelle ;
- si e et f sont des expressions rationnelles, $(e + f)$, $(e.f)$ et $(e)^*$ sont des expressions rationnelles (les caractères d'espacement dans ces expressions sont facultatifs et seront ignorés).

ATTENTION : **Dans ce problème, on ne fera jamais de simplification dans l'écriture d'expressions rationnelles**. **Une expression rationnelle simplifiée ne sera pas considérée comme étant une expression rationnelle.**

La **longueur** d'une expression rationnelle est son nombre total de caractères (lettres de l'alphabet Σ ou symboles '*', '+', '.', '(', ')', '$\varnothing$', 'ε'). Par exemple, la longueur de l'expression $(((a.c))^* + b)$, où a, b et c appartiennent à l'alphabet, vaut 12.

On fait correspondre à chaque expression rationnelle sur l'alphabet Σ un **langage** dit **rationnel** sur l'alphabet Σ ; ce langage sera dit **décrit par l'expression rationnelle**. On rappelle que :

- l'expression $\varnothing$ décrit le langage vide ;
- l'expression ε décrit le langage qui ne contient que le mot ε,
- si a est dans Σ, l'expression a décrit le langage réduit au mot a,
- si e_1 décrit un langage L_1 et si e_2 décrit un langage L_2, $(e_1 + e_2)$ décrit le langage $L_1 \cup L_2$ et $(e_1.e_2)$ décrit le langage $L_1.L_2$,

- si e décrit un langage L, $(e)^*$ décrit l'**itération**, ou **étoile**, de L, notée L^*, c'est-à-dire l'union de toutes les puissances de L : $L^* = \bigcup_{p \geq 0} L^p$, avec $L^0 = \{\varepsilon\}$, $L^1 = L$ et, pour $p \geq 2$, $L^p = L.L^{p-1}$.

Deux expressions rationnelles sont dites **équivalentes** si elles décrivent le même langage.
Un langage est rationnel si et seulement s'il existe un automate qui le reconnaît.

On veut d'abord montrer la propriété (P) suivante : toute expression rationnelle e est équivalente

- soit à l'expression rationnelle $\varnothing$,
- soit à l'expression rationnelle ε,
- soit à une expression rationnelle e' qui ne contient pas les symboles $\varnothing$ et ε,
- soit à une expression rationnelle de la forme $(e' + \varepsilon)$, où e' est une expression rationnelle qui ne contient pas les symboles $\varnothing$ et ε.

❒ 1 – Soient *e1* et *e2* deux expressions rationnelles ne contenant pas les symboles $\varnothing$ et ε ; en exhibant sans démonstration des expressions rationnelles équivalentes, montrer que la propriété (P) est exacte pour les huit expressions suivantes : $(\varepsilon + \varepsilon)$, $(e1 + \varnothing)$, $(\varepsilon.\varnothing)$, $(e1.\varnothing)$, $(e1.\varepsilon)$, $((e1 + \varepsilon) + (e2 + \varepsilon))$, $(e1.(e2 + \varepsilon))$, $((e1 + \varepsilon).(e2 + \varepsilon))$.

❒ 2 – Esquisser une démonstration par récurrence de la propriété (P) dans le cas général. En particulier, on fera apparaître les différents cas.

***ATTENTION* : dans toute la suite du problème, on ne considère que des expressions rationnelles ne contenant ni le symbole $\varnothing$, ni le symbole ε, même si cela n'est pas rappelé**.

On appelle **expression sur Σ** une suite composée des symboles '*', '+', '.', '(', ')' et de lettres de l'alphabet Σ. Une expression n'est pas nécessairement rationnelle ; par exemple, l'expression $(a + b$ n'est pas rationnelle alors que l'expression $(a + b)$ l'est ; l'expression $(a.b)^*$ n'est pas rationnelle alors que l'expression $((a.b))^*$ l'est ; l'expression $(a + b).(a.b)$ n'est pas rationnelle alors que l'expression $((a + b).(a.b))$ l'est.

Les expressions rationnelles seront codées par des tableaux indicés à partir de 0 et contenant des entiers de signes quelconques de la façon suivante.

On associe des constantes entières négatives distinctes aux cinq symboles utilisés ; plus précisément, on associe :

- une constante nommée ETOILE au symbole '*' avec ETOILE = –1,
- une constante nommée PLUS au symbole '+' avec PLUS = –2,
- une constante nommée POINT au symbole '.' avec POINT = –3,
- une constante nommée P_O au symbole '(' avec P_O = –4,
- une constante nommée P_F au symbole ')' avec P_F = –5.

Dans tout le problème, ces constantes seront manipulées par leur nom et non par leur valeur.

Par ailleurs, les lettres de l'alphabet Σ sont numérotées à partir de 0 ; par exemple, si $\Sigma = \{a, b, c\}$, la lettre 'a' est numérotée 0, la lettre 'b' est numérotée 1 et la lettre 'c' est numérotée 2. En ce sens, l'alphabet est considéré comme ordonné.

L'expression rationnelle est alors codée par un tableau obtenu en remplaçant les symboles par les constantes associées et les lettres de l'alphabet Σ par leur numéro.

Voici deux exemples pour $\Sigma = \{a, b, c\}$.

Exemple 1 : l'expression rationnelle *expression1* = $(((a.c))^* + b)$ est codée par le tableau *expr1* ci-dessous :

0	1	2	3	4	5	6	7	8	9	10	11
P_O	P_O	P_O	0	POINT	2	P_F	P_F	ETOILE	PLUS	1	P_F

Exemple 2 : l'expression rationnelle *expression2* = $((b)^*.(a + ((a)^*.b)))$ est codée par le tableau *expr2* ci-dessous :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P_O	P_O	1	P_F	ETOILE	POINT	P_O	0	PLUS	P_O	P_O	0	P_F	ETOILE	POINT	1	P_F	P_F	P_F

Une partie d'un tableau peut aussi coder une expression rationnelle ; c'est le cas par exemple de la partie du tableau *expr1* située de l'indice 2 inclus à l'indice 6 inclus qui code l'expression rationnelle $(a.c)$. Quand on dira simplement qu'un tableau code une expression rationnelle, cette expression rationnelle sera celle qui est codée à partir de l'indice 0 du tableau.

Premières indications pour la programmation
Caml

On définit des constantes par les instructions ci-dessous :

```
let ETOILE = -1;;
let PLUS = -2;;
let POINT = -3;;
let P_O = -4;;
let P_F = -5;;
```

Les expressions sont codées dans des tableaux ayant un nombre de cases exactement égal à la longueur de l'expression. Si un tableau `expr` code une expression, `vect_length expr` donne la longueur de l'expression.

Fin des premières indications pour Caml

Pascal

Dans tout le problème, on suppose qu'on écrit les différentes fonctions ou procédures dans un fichier contenant les définitions suivantes :

```
const MAX = 100;
      ETOILE = -1;
      PLUS = -2;
      POINT = -3;
      P_O = -4;
      P_F = -5;

type Table_Integer = array[0 .. MAX - 1] of Integer;
type Table_Boolean = array[0 .. MAX - 1] of Boolean;
type Matrice_Boolean = array[0 .. MAX - 1, 0 .. MAX - 1] of Boolean;
```

Les expressions sont codées dans des tableaux de type `Table_Integer` à partir de l'indice 0. On suppose que toutes les expressions traitées ont une longueur inférieure ou égale à la constante `MAX` ; on suppose aussi que l'alphabet n'a pas plus de `MAX` lettres.

Fin des premières indications pour Pascal

❒ 3 – Écrire dans le langage de programmation choisi une fonction nommée `est_symbole` prenant un paramètre entier et qui renvoie une valeur booléenne indiquant si l'entier reçu en paramètre a ou non une valeur égale à l'une des cinq constantes `ETOILE`, `PLUS`, `POINT`, `P_O`, `P_F`.

❒ 4 – On considère un tableau *expr* codant une expression de longueur $\ell \geq 1$ et deux indices notés *debut* et *fin* vérifiant $0 \leq debut < fin < \ell$.
Il s'agit d'écrire en langage de programmation une fonction à valeurs entières nommée *cesure* qui prend *debut* et *fin* comme paramètres et qui cherche s'il existe un indice i vérifiant simultanément :

- $debut < i < fin$,
- la case d'indice i du tableau *expr* contient soit la valeur PLUS soit la valeur POINT,
- dans la partie du tableau *expr* située de l'indice $(debut + 1)$ inclus à l'indice $(i - 1)$ inclus, il y a autant de cases contenant la constante P_O que de cases contenant la constante P_F.

Si au moins un indice i vérifiant ces propriétés existe, la fonction renvoie le plus petit tel indice ; dans le cas contraire, la fonction renvoie la valeur –1.
Par exemple, si on applique cette fonction à *expr2* avec *debut* qui vaut 0 et *fin* qui vaut 18, la fonction renvoie 5. Si on l'applique à *expr1* avec *debut* qui vaut 1 et *fin* qui vaut 7, la fonction renvoie –1.

Caml : écrire en Caml une fonction `cesure` telle que :

- si `expr` est un tableau codant une expression de longueur ℓ,
- si `debut` et `fin` sont deux indices vérifiant $0 \leq$ `debut` $<$ `fin` $< \ell$,

alors `cesure expr debut fin` renvoie la valeur définie ci-dessus.

Pascal : écrire en Pascal une fonction `cesure` telle que :
- si `expr` est de type `Table_Integer` et code une expression de longueur ℓ,
- si `debut` et `fin` sont deux indices vérifiant $0 \leq$ `debut` $<$ `fin` $< \ell$,

alors `cesure(expr, debut, fin)` renvoie la valeur définie ci-dessus.

❒ 5 – Il s'agit d'écrire en langage de programmation une fonction nommée *est_rationnelle* qui permet de déterminer si une expression est ou non rationnelle.

Caml : écrire en Caml une fonction récursive `est_rationnelle` telle que :
- si `expr` est un tableau codant une expression de longueur ℓ,
- si `debut` et `fin` sont deux entiers vérifiant $0 \leq$ `debut` $\leq$ `fin` $< \ell$,

alors `est_rationnelle expr debut fin` renvoie une valeur booléenne indiquant si la partie du tableau `expr` située de l'indice `debut` inclus à l'indice `fin` inclus code ou non une expression rationnelle.

Pascal : écrire en Pascal une fonction récursive `est_rationnelle` telle que :
- si `expr` est de type `Table_Integer` et code une expression de longueur ℓ,
- si `debut` et `fin` sont deux entiers vérifiant $0 \leq$ `debut` $\leq$ `fin` $< \ell$,

alors `est_rationnelle(expr, debut, fin)` renvoie une valeur booléenne indiquant si la partie du tableau `expr` située de l'indice `debut` inclus à l'indice `fin` inclus code ou non une expression rationnelle.

Désormais, toutes les expressions considérées seront rationnelles. On rappelle qu'on ne considère que des expressions rationnelles ne contenant ni le symbole $\varnothing$ ni le symbole ε.

❒ 6 – Soit e une expression rationnelle. Indiquer le principe d'un algorithme récursif permettant de tester si le langage décrit par e contient ou non le mot ε.

L'objectif est maintenant de construire un arbre représentant une expression rationnelle.

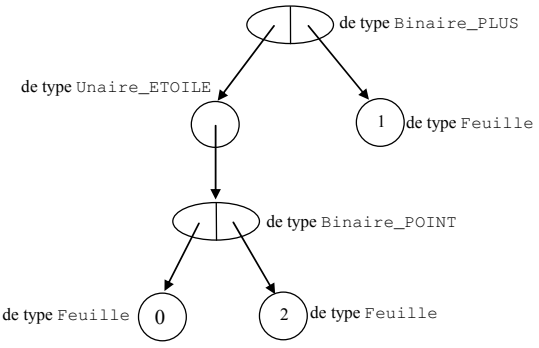
Indications pour la programmation
Caml

En plus des définitions indiquées plus haut, on définit les types suivants :

```
type Arbre =    Feuille of int
              | Unaire_ETOILE of Arbre
              | Binaire_PLUS of Arbre * Arbre
              | Binaire_POINT of Arbre * Arbre;;
```

Si `num` est un entier, on peut créer une feuille d'un arbre contenant la valeur `num` en invoquant `Feuille(num)`. Si `fils` est de type `Arbre`, on peut créer un nœud de type `Unaire_ETOILE` en invoquant `Unaire_ETOILE(fils)`. Si `fils1` et `fils2` sont de type `Arbre`, on peut créer un nœud de type `Binaire_PLUS` en invoquant `Binaire_PLUS(fils1, fils2)` et un nœud de type `Binaire_POINT` en invoquant `Binaire_POINT(fils1, fils2)`.

L'expression *expression1* = $(((a.c))^* + b)$ est décrite en Caml par l'arbre représenté ci-contre :



Fin des indications pour Caml

Pascal

En plus des définitions indiquées plus haut, on définit les types suivants :

```
type Arbre = ^Noeud;
     Noeud = record
        num : Integer;
        fils1, fils2: Arbre;
     end;
```

Une variable de type `Noeud` est un enregistrement. Un enregistrement contient des champs (aussi appelés des membres) ; par exemple, ici, une variable de type `Noeud` contient les champs `num`, `fils1`, `fils2`. Une variable `A` de type `Arbre` est appelée pointeur vers une variable de type `Noeud` ; la variable `A` permet de créer une variable de type `Noeud` pendant l'exécution du programme ; cela se fait en invoquant `new(A)`. L'enregistrement créé est dit pointé par `A` et se note `A^` ; les champs de cet enregistrement sont accessibles en faisant suivre `A^` d'un point puis du nom du champ considéré, comme cela est fait un peu plus bas.

Lorsqu'un arbre est construit pour représenter une expression rationnelle, il faut créer un enregistrement de type `Noeud` pour chacun des nœuds de l'arbre. Cela se fait en utilisant les trois fonctions ci-dessous.

La fonction `nouvelle_feuille` permet de créer une feuille contenant la valeur de l'entier reçu en paramètre ; on donne aux champs `fils1` et `fils2` la valeur `nil`, ce qui signifie que le nœud créé n'a ni fils gauche, ni fils droit.

```
function nouvelle_feuille(numero : Integer) : Arbre;
var
   A : Arbre;
begin
   new(A);
   A^.num := numero;
   A^.fils1 := nil;
   A^.fils2 := nil;
   nouvelle_feuille := A;
end;
```

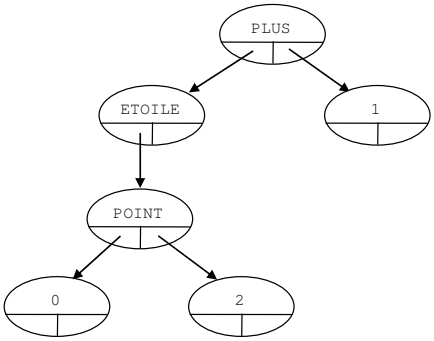
La fonction `nouvel_unaire` permet de créer un nœud interne ayant un seul fils qui est le fils gauche. Elle correspond à l'opérateur '*' appliqué à l'expression rationnelle représentée par le sous-arbre `fils` reçu en paramètre ; on donne au champ `fils2` la valeur `nil`, ce qui signifie qu'il n'y a pas de fils droit.

```
function nouvel_unaire(fils : Arbre) : Arbre;
var
   A : Arbre;
begin
   new(A);
   A^.num := ETOILE;
   A^.fils1 := fils;
   A^.fils2 := nil;
   nouvel_unaire := A;
end;
```

La fonction `nouveau_binaire` permet de créer un nœud interne ayant deux fils. Elle correspond aux opérateurs '+' ou '.' appliqués aux expressions rationnelles représentées par les sous-arbres `fils1` et `fils2` reçus en paramètres ; le paramètre `numero` vaudra selon le cas la valeur `PLUS` ou la valeur `POINT`.

```
function nouveau_binaire(numero : Integer; fils1, fils2 : Arbre) : Arbre;
var
   A : Arbre;
begin
   new(A);
   A^.num := numero;
   A^.fils1 := fils1;
   A^.fils2 := fils2;
   nouveau_binaire := A;
end;
```

L'expression *expression1* = (((*a*.*c*))* + *b*) est décrite en Pascal par l'arbre représenté ci-contre ; les cases vides contiennent la valeur `nil`.



Fin des indications pour Pascal

❒ 7 – Il s'agit d'écrire une fonction nommée `expression_vers_arbre` qui permette de construire un arbre représentant une expression rationnelle.

Caml : écrire en Caml une fonction récursive `expression_vers_arbre` telle que :
- si `expr` est un tableau codant une expression rationnelle *e* de longueur $\ell \geq 1$,
- si `debut` et `fin` sont deux entiers vérifiant $0 \leq$ `debut` $\leq$ `fin` $< \ell$ pour lesquels la partie du tableau `expr` située entre l'indice `debut` inclus et l'indice `fin` inclus code une expression rationnelle notée *e*′,

alors `expression_vers_arbre debut fin` renvoie une valeur de type `Arbre` donnant la racine d'un arbre qui représente l'expression *e*′.

Pascal : écrire en Pascal une fonction récursive `expression_vers_arbre` telle que :
- si `expr` est de type `Table_Integer` et code une expression rationnelle *e* de longueur $\ell \geq 1$,
- si `debut` et `fin` sont deux entiers vérifiant $0 \leq$ `debut` $\leq$ `fin` $< \ell$ pour lesquels la partie du tableau `expr` située entre l'indice `debut` inclus et l'indice `fin` inclus code une expression rationnelle notée *e*′,

alors `expression_vers_arbre(debut, fin)` renvoie une valeur de type `Arbre` donnant la racine d'un arbre qui représente l'expression *e*′.

❒ 8 – Il s'agit d'écrire en langage de programmation une fonction nommée *contient_epsilon* permettant de tester si le langage décrit par une expression rationnelle contient ou non le mot ε. Ce test est fait à partir de l'arbre représentant l'expression rationnelle.

Caml : écrire en Caml une fonction récursive `contient_epsilon` telle que :
- si `arbre` est de type `Arbre` et représente une expression rationnelle *e*,

alors `contient_epsilon arbre` renvoie une valeur booléenne indiquant si le langage décrit par *e* contient ou non le mot ε.

Pascal : écrire en Pascal une fonction récursive `contient_epsilon` telle que :
- si `A` est de type `Arbre` et représente une expression rationnelle *e*,

alors `contient_epsilon(A)` renvoie une valeur booléenne indiquant si le langage décrit par *e* contient ou non le mot ε.

Deuxième partie : langages locaux

On note Σ^2 l'ensemble des mots de longueur 2 sur un alphabet Σ. On dit qu'un langage *L* sur un alphabet Σ est un **langage local** s'il existe simultanément :
- un sous-ensemble *I* de Σ,
- un sous-ensemble *F* de Σ,
- un sous-ensemble *P* de Σ^2

tels qu'un mot *u* de Σ^* autre que le mot ε appartienne à *L* si et seulement si :

- la lettre initiale de u est dans I,
- la lettre finale de u est dans F,
- si la longueur de u est au moins 2, tout facteur de longueur 2 de u est dans P.

Un langage local L peut contenir ou non le mot ε.

Un mot de longueur 1 appartient à un langage local si la lettre qui le compose est à la fois dans I et dans F.

On peut ainsi définir un langage local L par l'ensemble I des lettres initiales permises, l'ensemble F des lettres finales permises, l'ensemble P des facteurs de longueur 2 permis et le fait de contenir ou non le mot ε. En posant $\alpha = vrai$ si L contient ε et $\alpha = faux$ si L ne le contient pas, on dit alors dans ce problème que le langage local L est caractérisé par le quadruplet (I, F, P, α).

Les ensembles I, F et P peuvent être vides. Si I est vide, le langage L est alors soit vide, soit réduit au mot ε ; il en est de même si F est vide.

❒ 9 – On considère l'alphabet $\Sigma = \{a, b\}$. Montrer que les langages décrits par les expressions rationnelles $(a.(a)^*)$ et $((a.b))^*$ sont locaux. On donnera pour chacun des langages un quadruplet qui le caractérise.

❒ 10 – Déterminer celles des propositions ci-dessous qui sont exactes. Dans chaque cas, on indiquera clairement la réponse et on justifiera celle-ci.

- *Proposition sur l'union* : l'union de deux langages locaux est un langage local.
- *Proposition sur l'intersection* : l'intersection de deux langages locaux est un langage local.
- *Proposition sur la concaténation* : la concaténation de deux langages locaux est un langage local.
- *Proposition sur l'itération* : l'itération L^* d'un langage local L est un langage local.

❒ 11 – Soit L un langage défini sur l'alphabet Σ. On suppose que L est local et caractérisé par le quadruplet (I, F, P, α). Exprimer à l'aide des langages I, F, P, Σ, Σ^* et en utilisant uniquement les opérations d'intersection, de concaténation et de complémentation par rapport à Σ^* :

- le langage des mots dont la lettre initiale est dans I,
- le langage des mots dont la lettre finale est dans F,
- le langage des mots de longueur 2 qui ne sont pas dans P,
- puis le langage des mots contenant au moins un facteur de longueur 2 qui n'est pas dans P,
- puis le langage $L - \{\varepsilon\}$.

❒ 12 – Déduire de la question précédente qu'un langage local est un langage rationnel.

Un sous-ensemble E de lettres de l'alphabet Σ est représenté à l'aide d'un tableau de valeurs booléennes ; pour i compris entre 0 et $|\Sigma| - 1$, la case d'indice i du tableau contient la valeur *vrai* si la lettre de numéro i appartient à E et la valeur *faux* sinon.

Par exemple, le sous-ensemble $\{a, c\}$ de l'alphabet $\Sigma = \{a, b, c, d\}$ est codé par le tableau :

0	1	2	3
vrai	*faux*	*vrai*	*faux*

Un ensemble P de mots de longueur 2 sur l'alphabet Σ est représenté à l'aide d'une matrice de valeurs booléennes ; pour i compris entre 0 et $|\Sigma| - 1$ et pour j compris entre 0 et $|\Sigma| - 1$, la case de ligne i et de colonne j de la matrice contient la valeur *vrai* si le mot de longueur 2 composé de la lettre de numéro i suivie de la lettre de numéro j appartient à P (on dit alors que cette case correspond au mot considéré) et la valeur *faux* sinon.

Par exemple, l'ensemble de mots $\{ac, bc, cd, bb, ad, db, ca\}$ sur l'alphabet $\Sigma = \{a, b, c, d\}$ est codé par la matrice ci-contre. Dans cette matrice, la case d'indices 1 et 2 correspond au mot bc.

	0	1	2	3
0	*faux*	*faux*	*vrai*	*vrai*
1	*faux*	*vrai*	*vrai*	*faux*
2	*vrai*	*faux*	*faux*	*vrai*
3	*faux*	*vrai*	*faux*	*faux*

Les mots sont codés par un tableau d'entiers donnant les indices de leurs lettres successives dans l'alphabet Σ. Par exemple, le mot $acbba$ sur l'alphabet $\Sigma = \{a, b, c\}$ est codé par le tableau :

0	1	2	3	4
0	2	1	1	0

❒ 13 – Il s'agit de définir une fonction nommée *appartient* déterminant si un mot sur Σ autre que ε appartient ou non au langage local caractérisé par le quadruplet (I, F, P, α).

Caml : écrire en Caml une fonction nommée `appartient` telle que :
- si `mot` est un tableau de longueur ℓ codant un mot u sur Σ autre que ε et de longueur $\ell \geq 1$,
- si `I` et `F` sont deux tableaux de longueur $|\Sigma|$ codant des sous-ensembles I et F de Σ,
- si `P` est une matrice carrée dont les deux dimensions valent $|\Sigma|$ et qui code un ensemble P de mots de longueur 2 sur Σ,

alors `appartient mot I F P` renvoie une valeur booléenne indiquant si le mot u appartient ou non au langage local caractérisé par le quadruplet (I, F, P, α).

Pascal : écrire en Pascal une fonction nommée `appartient` telle que :
- si `mot` est de type `Table_Integer` et contient le code d'un mot u sur Σ autre que ε,
- si `longueur` est un entier ayant pour valeur la longueur de u,
- si `I` et `F` sont de type `Table_Boolean` et codent des sous-ensembles I et F de Σ,
- si `P` est de type `Matrice_Boolean` et code un ensemble P de mots de longueur 2 sur Σ,

alors `appartient(mot, longueur, I, F, P)` renvoie une valeur booléenne indiquant si le mot u appartient ou non au langage local caractérisé par le quadruplet (I, F, P, α).

Troisième partie : mots appartenant au langage décrit par une expression rationnelle

❒ 14 – On considère une expression rationnelle e sur un alphabet Σ dans laquelle toute lettre de Σ figure au plus une fois, c'est-à-dire dans laquelle toutes les lettres sont distinctes. Montrer que le langage L décrit par e est un langage local.

Soit e une expression rationnelle et soit L le langage décrit par e. On note $I(e)$ l'ensemble des lettres initiales des mots de L, $F(e)$ l'ensemble des lettres finales des mots de L, $P(e)$ l'ensemble des facteurs de longueur 2 des mots de L et $\alpha(e)$ la valeur *vrai* ou *faux* selon que L contient ou non le mot ε.

❒ 15 – On rappelle qu'on a défini : *expression1* = $(((a.c))^* + b)$. Indiquer sans démonstration les valeurs de I(*expression1*), F(*expression1*), P(*expression1*) et α(*expression1*).

❒ 16 – Étant donnée une expression rationnelle e sur un alphabet Σ, il s'agit d'écrire en langage de programmation une fonction ou une procédure nommée *calcul_I* qui permette de déterminer l'ensemble $I(e)$.

Caml : écrire en Caml une fonction récursive `calcul_I` telle que :
- si `arbre` est de type `Arbre` et représente une expression rationnelle e sur un alphabet Σ comme cela a été décrit avant la question ❒ 7,
- si `I` est un tableau de longueur $|\Sigma|$ destiné à coder un sous-ensemble de Σ de la manière expliquée avant la question ❒ 13,

alors `calcul_I arbre I` modifie le tableau `I` en affectant la valeur `true` aux cases dont l'indice est égal au numéro d'une lettre de $I(e)$ (la fonction ne modifie pas les autres cases de `I`).
Remarque : on suppose qu'avant le premier appel de la fonction `calcul_I`, un tableau de variables booléennes a été créé et initialisé en attribuant à toutes les cases la valeur `false` ; c'est ce tableau qui contiendra à la fin la description de l'ensemble I ; cette initialisation n'est pas à écrire.

Pascal : écrire en Pascal une procédure récursive `calcul_I` telle que :
- si `A` est de type `Arbre` et représente une expression rationnelle e sur un alphabet Σ comme cela a été décrit avant la question ❒ 7,
- si `I` est de type `Table_Boolean` et est destiné à coder un sous-ensemble de Σ de la manière expliquée avant la question ❒ 13,

alors `calcul_I(A, I)` modifie le tableau `I` en affectant la valeur `true` aux cases dont l'indice est égal au numéro d'une lettre de $I(e)$ (la procédure ne modifie pas les autres cases de `I`).

Remarque : on suppose qu'avant le premier appel de la procédure `calcul_I`, un tableau de variables booléennes a été défini et initialisé en attribuant à toutes les cases la valeur `false` ; c'est ce tableau qui contiendra à la fin la description de l'ensemble I ; cette initialisation n'est pas à écrire.

Étant donnée une expression rationnelle e sur un alphabet Σ, on admet qu'on peut écrire en langage de programmation une fonction ou une procédure nommée *calcul_F* qui permette de déterminer l'ensemble $F(e)$ en codant le tableau représentant $F(e)$ de manière analogue à ce qui a été fait pour $I(e)$. Cette fonction ou cette procédure aurait les mêmes types de paramètres que la fonction ou la procédure *calcul_I*. On ne demande pas d'écrire la fonction ou la procédure *calcul_F* mais on pourra y faire appel dans la suite du problème.

❒ 17 – Il s'agit d'écrire une fonction ou une procédure qui servira pour le calcul ultérieur de l'ensemble $P(e)$.

Caml : écrire en Caml une fonction nommée `ajout_couples` telle que :
- si `produit` est une matrice carrée de booléens de dimension $dim \times dim$,
- si `T1` et `T2` sont deux tableaux de booléens de dimension dim,

alors `ajout_couples produit T1 T2` modifie la matrice `produit` en affectant, pour tout i et tout j vérifiant $0 \leq i < dim$ et $0 \leq j < dim$, la valeur `true` à la case d'indices i et j si les cases d'indice i du tableau `T1` et d'indice j du tableau `T2` contiennent toutes deux la valeur `true` (la fonction ne modifie pas les autres cases de la matrice).

Pascal : écrire en Pascal une procédure nommée `ajout_couples` telle que :
- si `produit` est de type `Matrice_Boolean`,
- si `T1` et `T2` sont de type `Table_Boolean`,
- si `dim` est un entier,

alors `ajout_couples(produit, T1, T2, dim)` modifie la matrice `produit` en affectant, pour tout i et tout j vérifiant $0 \leq i <$ `dim` et $0 \leq j <$ `dim`, la valeur `true` à la case d'indices i et j si les cases d'indice i du tableau `T1` et d'indice j du tableau `T2` contiennent toutes deux la valeur `true` (la procédure ne modifie pas les autres cases de la matrice).

❒ 18 – Étant donnée une expression rationnelle e sur un alphabet Σ, il s'agit d'écrire en langage de programmation une fonction ou une procédure nommée *calcul_P* qui permette de déterminer l'ensemble $P(e)$.

Caml : écrire en Caml une fonction récursive `calcul_P` telle que :
- si `arbre` est de type `Arbre` et représente une expression rationnelle e sur un alphabet Σ,
- si `P` est une matrice carrée dont les deux dimensions valent $|\Sigma|$ et qui est destinée à coder un ensemble de mots de longueur 2 sur Σ comme expliqué avant la question ❒ 13,

alors `calcul_P arbre P` modifie le tableau `P` en affectant la valeur `true` aux cases correspondant aux éléments de $P(e)$ (la fonction ne modifie pas les autres cases de la matrice).

Pascal : écrire en Pascal une procédure récursive `calcul_P` telle que :
- si `A` est de type `Arbre` et représente une expression rationnelle e sur un alphabet Σ,
- si `P` est de type `Matrice_Boolean` et est destiné à coder un ensemble de mots de longueur 2 sur Σ de la manière expliquée avant la question ❒ 13,

alors `calcul_P A P` modifie le tableau `P` en affectant la valeur `true` aux cases correspondant aux éléments de $P(e)$ (la procédure ne modifie pas les autres cases de la matrice).

❒ 19 – On considère une expression rationnelle e sur un alphabet Σ dans laquelle toutes les lettres sont distinctes ; on note L le langage décrit par e. Soit u un mot sur Σ autre que ε. Décrire un algorithme utilisant les fonctions ou procédures écrites précédemment pour déterminer si le mot u appartient ou non à L.

❒ 20 – On considère une expression rationnelle e sur un alphabet Σ ; il s'agit d'écrire en langage de programmation une fonction nommée *lettres_distinctes* qui permette de déterminer si toutes les lettres de e sont distinctes, c'est-à-dire si toute lettre de Σ figure au plus une fois dans e.

Caml : écrire en Caml une fonction `lettres_distinctes` telle que,
- si `expr` est un tableau codant une expression e sur un alphabet Σ,
- si `n` est un entier qui donne le cardinal de l'alphabet Σ,

alors `lettres_distinctes expr n` renvoie une valeur booléenne indiquant si les lettres de l'expression e sont ou non distinctes.

Pascal : écrire en Pascal une fonction `lettres_distinctes` telle que,
- si `expr` est de type `Table_Integer` et code une expression rationnelle e sur un alphabet Σ,
- si `longueur` est un entier qui donne la longueur de l'expression e,
- si `n` est un entier qui donne le cardinal de l'alphabet Σ,

alors `lettres_distinctes(expr, longueur, n)` renvoie une valeur booléenne indiquant si les lettres de l'expression e sont ou non distinctes.

On considère une expression rationnelle e sur un alphabet Σ. On traduit cette expression en une autre expression rationnelle e' en utilisant un second alphabet Σ' de cardinal aussi grand que nécessaire. Pour cela, on traduit le codage de e en le codage de e' de la manière suivante. Les lettres de l'alphabet Σ' sont numérotées par 0, 1, 2… comme cela est fait pour l'alphabet Σ. Pour passer du codage de e au codage de e', toutes les valeurs représentant les symboles de e sont recopiées à la même place dans le codage de e'. Les numéros des lettres du codage de e sont remplacés dans le codage de e' de la gauche vers la droite successivement par 0, 1, 2, 3, etc. Par exemple, l'expression rationnelle *expression2* = $((b)^*.(a + ((a)^*.b)))$ sur l'alphabet $\Sigma = \{a, b\}$ est transformée en l'expression rationnelle *expression2'* sur Σ' dont le codage est le tableau ci-dessous :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P_O	P_O	0	P_F	ETOILE	POINT	P_O	1	PLUS	P_O	P_O	2	P_F	ETOILE	POINT	3	P_F	P_F	P_F

En supposant que l'alphabet Σ' est l'ensemble ordonné $\{A, B, C, D, \ldots\}$, *expression2* est donc traduite en *expression2'* = $((A)^*.(B + ((C)^*.D)))$.

Soit q le nombre de lettres intervenant dans l'expression rationnelle e, distinctes ou non. Par exemple, si e est *expression2*, le nombre q vaut 4. On définit une application ϕ des q premières lettres de Σ' dans les lettres de Σ ; si une occurrence de la lettre x se trouvant dans e a été remplacée dans e' par une lettre y, on pose $\phi(y) = x$. Si e est *expression2*, on a ainsi $\phi(A) = b$, $\phi(B) = a$, $\phi(C) = a$, $\phi(D) = b$.

Soit $u = x_1x_2\ldots x_p$ un mot sur l'alphabet Σ. Avec les notations précédentes, on admet l'équivalence suivante : « le mot u appartient au langage décrit par e si et seulement s'il existe un mot $y_1y_2\ldots y_p$ appartenant au langage décrit par e' et vérifiant, pour i compris entre 1 et p, $\phi(y_i) = x_i$ ».

❒ 21 – Soient e une expression rationnelle sur un alphabet Σ et L le langage décrit par e. Décrire sommairement un algorithme qui permette de déterminer si un mot u sur Σ autre que ε appartient ou non à L.

Quatrième partie : algorithme de Glushkov

Cette dernière partie a pour objectif de décrire l'algorithme de Glushkov ; cet algorithme permet de construire un automate fini qui reconnaît le langage décrit par une expression rationnelle.

Les rappels de définitions qui suivent permettent de fixer la terminologie et les notations.

Un *automate fini* $\mathcal{A}$ est décrit par une structure $<\Sigma, Q, T, I, F>$, où :
- Σ est un alphabet ;
- Q est un ensemble fini et non vide appelé *ensemble des états* de $\mathcal{A}$;
- $T \subseteq Q \times \Sigma \times Q$ est appelé l'*ensemble des transitions* ; étant donnée une transition $(p, a, q) \in T$, on dit qu'elle va de l'état p vers l'état q et qu'elle est d'étiquette a ; on pourra la noter $p \xrightarrow{a} q$;
- $I \subseteq Q$ est appelé ensemble des *états initiaux* de $\mathcal{A}$;
- $F \subseteq Q$ est appelé ensemble des *états finals* de $\mathcal{A}$.

❒ 22 – On considère le langage local L_1 sur l'alphabet $\Sigma = \{a, b, c\}$ caractérisé par le quadruplet $(I_1, F_1, P_1, \alpha_1)$ avec :

- $I_1 = \{a\}$;
- $F_1 = \{c\}$;
- $P_1 = \{ab, bc, ca\}$;
- $\alpha_1 = faux$.

Dessiner sans démonstration un automate déterministe $\mathcal{A}_1$ possédant quatre états et reconnaissant le langage L_1.

❒ 23 – On considère le langage local L_2 sur l'alphabet $\Sigma = \{a, b, c\}$ caractérisé par le quadruplet $(I_2, F_2, P_2, \alpha_2)$ avec :

- $I_2 = \{a, c\}$;
- $F_2 = \{b\}$;
- $P_2 = \{ab, ac, ba, cb, cc\}$;
- $\alpha_2 = vrai$.

Dessiner sans démonstration un automate déterministe $\mathcal{A}_2$ possédant quatre états et reconnaissant le langage L_2 ; l'automate $\mathcal{A}_2$ doit être tel qu'aucune transition ne va vers l'état initial et que toutes les transitions allant vers un même état q portent la même étiquette ; à part l'état initial, il doit y avoir un état par lettre de l'alphabet.

❒ 24 – Soit L un langage sur l'alphabet Σ. On suppose que L est un langage local caractérisé par un quadruplet (I, F, P, α). Décrire un automate $\mathcal{A}$ déterministe qui reconnaisse le langage L et ayant $|\Sigma| + 1$ états. On ne demande pas de justifier la construction.

❒ 25 – On considère une expression rationnelle e sur un alphabet Σ. On note L le langage décrit par e. En utilisant ce qui a été fait précédemment, en particulier la transformation de e en une expression e' où toutes les lettres sont distinctes et la construction d'un automate reconnaissant un langage local, décrire un algorithme permettant de construire un automate reconnaissant L. On montrera que l'automate obtenu reconnaît le langage L.

❒ 26 – Appliquer l'algorithme de la question précédente à $expression2 = ((b)^*.(a + ((a)^*.b)))$ pour construire un automate reconnaissant le langage décrit par cette expression. On traduira $expression2$ en une expression $expression2'$ en utilisant un alphabet $\Sigma' = \{A, B, C, D, \ldots\}$ comme il a été fait dans les indications précédant la question ❒ 21. On donnera sans démonstration les ensembles $I(expression2')$, $F(expression2')$, $P(expression2')$ et la valeur de $\alpha(expression2')$.

❒ 27 – Utiliser une méthode au choix pour transformer l'automate obtenu à la question précédente en un automate déterministe équivalent. On se contentera de dessiner l'automate obtenu en indiquant sur la figure à quoi correspondent les nouveaux états.