# Langages Formels 2018-2019
# TDs + devoir + projet + TP
# Frédéric Gruau

## Plan

Le cours est divisé en six périodes de deux semaines, avec l'enchainement des six thèmes suivants: a-Rappels, approfondissement automates, c- Grammaires Hors Contexte, d- Automates à piles, e- analyse syntaxique ascendantes, f- Machine de Turing. L'enchaînement est une construction logique, c'est à dire qu'il est nécessaire d'assimiler les concepts au fur et à mesure, pour pouvoir continuer à suivre jusqu'au bout. En particulier, les éléves n'ayant jamais vus d'expressions rationnels ni d'automates d'états finis, doivent fournir un effort considérable les deux premières semaines (rappels), pour se mettre à niveau avec le reste. Ce receuil de TD et le support de cours se trouvent à l'adresse:

`https://www.lri.fr/~gruau/`

Il y à 12 cours(1h30) suivi de 12 TD(2h10). Le cours prépare aux TDs. Le déroulement pour chacune des 12 semaines est le suivant:

**a- Rappel**

1. Cours: Panoramique, Langage formels, Expression rationnelle, lemme d'Arden, def. automate d'état fini.
   TD: Egalité langage, Expression rationnelle, Automates simples.

2. Cours: Automate non-déterministe, epsilon transitions, théoreme de kleene
   TDs: Automates, suite et fin, détérminisation, résolution d'équation, début.

**b- Approfondissement automates**

3. Cours: minimisation d'un automate
   TD:résolution d'équation (autre exemple). Construction d'automates, Construction directe de l'automate minimal a partir du langage.

4. Cours pompage, clôture, décidabilité, analyse lexicale.
   TD pompage, clôture.

**c- Grammaires Hors Contexte**

5. Cours: grammaires hors contexte, arbre de dérivation, ambiguité, réécriture droite.
   TD: grammaire d'un langage, langage d'une grammaire, desambiguiser.

6. Cours: nettoyage de grammaire, FN Chomsky, décidabilité, cloture
   TD: analyse lexicale, grammaire d'un vrai langage, est-il-algébrique (1,4,5).

**d- Automates à piles**

7. Cours+TD automates à piles TD est-il-algébrique début

8. Cours: Equivalence automate à pile-grammaire,clôture, premier et suivant.
   TD: est-il-algébrique fin, Analyse ascendente à la main, premier et suivant,.

**e- Analyse Ascendente**

9. Cours+TD: Analyse ascendente

10. Cours automate LR(1) général, LALR(1), intro machine de Turing.
    TD exo d'analyse ascendante (rappel + LR, LALR ). machine de Turing simple

**f- Machine de Turing**

11. Cours décidabilité
    TD machine de Turing compliquée, décidabilité de l'ambiguïté.

12. Cours: NP complétude,
    TP en salle machine: yacc et lex

Les exercices optionnels sont plus difficiles. Ils sont concus pour occuper les meilleurs étudiants. Ils ne sont en général pas traités avec toute la classe par manque de temps. Certains exercices plus importants ou plus difficile sont répartis sur deux TDs: le premier TD traite un exemple facile, et le TD de la semaine suivant un deuxième exemple plus difficile. C'est le cas pour les automates d'état finis (TD 1 et 2) La résolution d'équations de langages (TD 2 et 3) est-il-algébrique (TD 6,7 et 8, il y a plusieurs méthodes pour faire cela) l'analyse ascendente (TD 9 et 10) les machine de Turing (TD 10 et 11).

## Examens, Devoir et Projet

Seules les notes de cours manuscrites, et les poly de cours et d'exercices sont autorisés aux examens. Chaque TD fait l'objet d'un exercice au partiel ou à l'examen. Le partiel et l'examen comprennent aussi des questions de cours non traitées en TD. Les notes de partiels mauvaises pourront être "partiellement" rattrapées grâce à un devoir et un projet à la maison relativement facile, mais sur un coefficient de seulement 10 pourcent par rapport au partiel. Contrôle continu = (8* partiel + devoir + projet)/10. L'énonce du devoir et du projet est inclus dans ce recueil, à la section 7 et 14.

# 1 Démonstration d'égalité entre deux langages

Les egalités suivantes sont elles vraies? si oui, le démontrer sinon donner un contre-exemple.

1. $L^* = L^*.L^* = (L^*)^*$

2. $L.(M \cap N) = (L.M) \cap (L.N)$

3. Optionnel: $(L^*.M)^* = \Lambda + (L + M)^*.M$

# 2 Expression rationnelle

## 2.1 ExprRat d'un langage

$L_1 = \{w \mid w$ commence par $ab\}$
$L_2 = \{w \mid w$ termine par $bb\}$
$L_3 = \{w \mid w$ commence par $ab$ et termine par $bb\}$
$L_4 = \{w \mid w$ contient trois occurrences successives de la lettre $a\}$
$L_5 = \{w \mid w$ ne commence pas par $ba\}$
$L_6 = \{w \mid w$ ne termine pas par $bba\}$

## 2.2 optionnel: ExprRat compliqué.

$L_7 = \{w \mid w$ ne contient pas deux occurrences successives de la lettre $a\}$
$L_8 = \{w \mid w$ ne contient pas trois occurrences successives de la lettre $a\}$
$L_9 = \{w \mid$ le nombre de $a$ dans $w$ est pair $\} = \{w \mid |w|_a = 0 \ (mod\ 2)\}$
$L_{10} = \{w \mid |w|_a = 1 \ (mod\ 3)\}$

## 2.3 ExprRat pour l'analyse lexicale.

La première étape d'un compilateur et l'analyse lexicale, qui découpe le texte d'un programme en unité lexicale appelée token. Un token peut être un mot clef, un identifiant, une constante numérique. On utilise des expression rationnelle pour identifer la nature des différent token. On utilisera la notation "étendue"" plus compacte. Par exemple, $e? = e|\epsilon$ qui signifie que $e$ est optionnel, $[0-9]$ siginifie un chiffre.

Ecrire l'expression rationnelle décrivant:

1. un identificateur comme une lettre suivit d'une suite de lettre ou de chiffre,

2. un entier positif

3. un entier relatif

4. un nombre a virgule

# 3 Automates reconnaissant un langage donné.

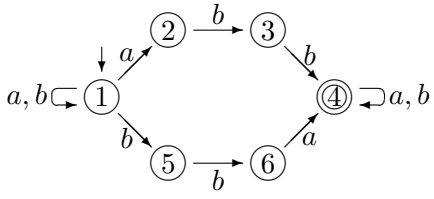
Donner des automates reconnaissant les langages suivants: (pour les entiers comme pour les mots habituel, on considère que les caractères sont lus de gauche a droite, donc en commençant par les bits de poids forts)

- des entiers pairs, des entiers impairs, des puissances de 2,
- $A = \{0,1\}, L = \{w \mid w$ code une puissance de 4$\}$
- $A = \{0,1\}, L = \{w \mid w$ code la somme de deux puissances de 4: $4^k + 4k', k \neq k'\}$.
- $A = \{a,b\}$, $L = \{w \mid w$ commence par $abaaba\}$
- $A = \{a,b\}$, $L = \{w \mid w$ contient $aabaaab\}$
- $A = \{a,b\}$, $L = \{w \mid w$ commence par $abb$ et termine par $bba\}$
- Les écritures de nombre à virgule
- $A = \{/, *, c\}$, $L$ est l'ensemble des commentaires, un commentaire étant un mot de la forme $/ * w * /$, et dans laquelle $w$ ne contient pas d'occurrence de $*/$, ni de $/*$, ($c$ représente l'ensemble des caractères autres que $*$ et $/$).
- $A = \{a,b,c\}, L = \{w \mid w$ contient au moins une fois chacune des trois lettres $\}$
- $A = \{a,b\}$, $L = \{w \mid w$ contient un nombre pair de fois le facteur $bab\}$
- A faire chez vous (possible). $A = \{a,b\}$, $L = \{w \mid |w|_a$ est pair, ainsi que $|w|_b\}$
- A faire chez vous . $A = \{0,1\}$, $L = \{w \mid$ en base 2, $w$ représente un nombre valant 1 modulo 3 $\}$

# 4 Déterminisation

## 4.1 Methode de détérminisation.

(Partiel 2000-01). Déterminisez :




## 4.2 Boum !

Soit $L_n$ l'ensemble des mots sur $\{a,b\}$ de longueur au moins $n$ dont la $n^{\text{ième}}$ lettre avant la fin est un $b$. Donnez un petit automate non-déterministe pour $L_3$. puis son déterminisé. Comparez leur nombre d'états. Au lieu de faire marcher l'algorithme de déterminisation, on commencera par réfléchir quels doivent être les états, puis on rajoutera les transitions.

## 4.3 Le barman boxeur

Cet exercice est à faire à la maison par les étudiants qui n'ont pas vu les notions d'automates d'état fini, afin qu'il se familiarisent.

Un barman et un client jouent au jeu suivant : Le barman met un bandeau sur les yeux qui le rend aveugle, et il met des gants de boxe qui l'empêchent de "sentir" si un verre est à l'endroit ou à l'envers. Devant le barman, se trouve un plateau tournant sur lequel sont placés quatre verres en carré. Ces verres peuvent être à l'envers ou à l'endroit. Le sens des verres est choisi par le client et est inconnu du barman. Si les verres sont tous dans le même sens, alors le barman gagne (Quand le barman gagne, un autre client, "arbitre", annonce qu'il a gagné et le jeu s'arrête.) Le barman peut répéter 10 fois l'opération suivante : Il annonce au client qu'il va retourner certains verres (par exemple le verre en bas à gauche et celui en bas à droite). Le client fait alors tourner le plateau, puis le barman retourne les verres comme il l'a annoncé. Si les verres sont alors tous dans le même sens, le barman gagne.

1)On se place du point de vue du client. Donnez un automate dont les états sont les différentes configurations du plateau, les lettres les coups annoncés par le barman et où les flèches décrivent les évolutions possibles des configurations. Le fait que 1- le client fait tourner le plateau comme il veut, et 2- on ne se préoccupe pas que tout les verres soit a l'endroit, mais seulement qu'ils soient dans le même sens, conduit à beaucoup simplifier: il y a seulement quatre états à distinguer, et seulement trois coups possibles à jouer, pour passer d'un état à un autre.

2) A partir de l'état ou 2 verres a Cote l'un de l'autre dans un sens et les 2 autres dans l'autre, donner une séquence de coup permettant au barman de gagner. Comme on a utilisé une seule lettre pour nommer les coups, cette suite

corresponds à un mot d'un langage formel.

3) Donnez un automate non déterministe (avec éventuellement plusieurs états entrée) qui donne toutes les séquences d'annonces du barman pour lesquelles le client peut gagner (à condition de toujours faire les bons choix).

4) Donnez un automate qui donne les coups qui assurent au barman de gagner quel que soit le comportement du client. On utilisera le résultat suivant: Soit $A$ un automate détérministe complet qui reconnaît un langage $L$, pour obtenir un automate qui reconnaît le complémentaire, il suffit d'inverser final/ non-final. Ce résultat ne marche pas si $A$ est non-deterministe ou non-complet.

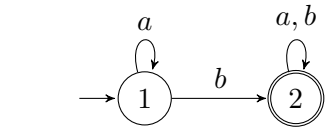
5) Jouez-vous de l'argent contre le barman?

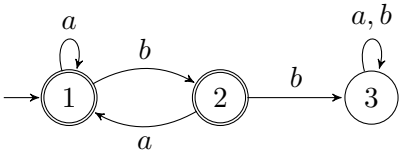
# 5 Resolution d'équations

Rappel de cours: a tout automate on peut associer un système d'équations dont les variables représentent les langages reconnus par cet automate à partir de chacun de ses états.

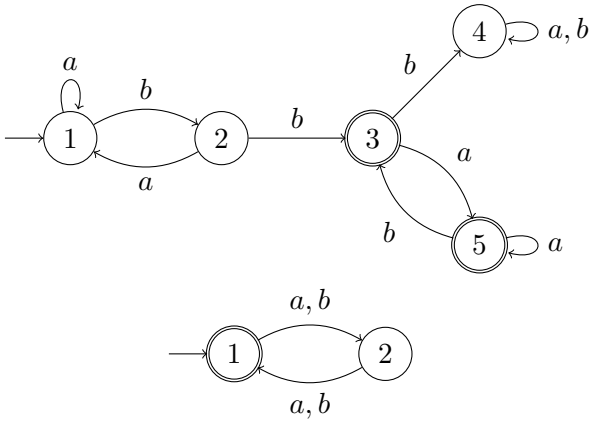
## 5.1 Exemple introductif

À l'aide du système d'équations précédent, que l'on résoudra par élimination et utilisation du lemme d'Arden, déterminer une expression rationnelle correspondant aux automates suivants : ( sur l'alphabet $\mathcal{A} = \{a, b\}$ )




## 5.2 Exemple optionnel.







# 6 Construction d'automates.

$L$ est reconnu par l'automate $A = (\Sigma, Q, \delta, q_0, F)$. Construire des automates reconnaissant :

- $miroir(L) = \{a_n a_{n-1} ... a_3 a_2 a_1 | a_1 a_2 ... a_n \in L\}$
- l'ensemble des mots obtenus a partir des mots de $L$ en effaçant tous les $a$.
- le complémentaire de $L$, en supposant $A$ déterministe.
- Optionnel: l'ensemble des mots obtenus en effaçant un nombre pair de lettres d'un mot de $L$

$L_1$ et $L_2$ sont reconnus par les automates $A_1$ et $A_2$.

On suppose que $A_1$ et $A_2$ sont déterministes complets.

- Donner un algorithme linéaire en $|u|$ pour savoir si $u \in L_1 \cap L_2$.
- En déduire la construction d'un automate déterministe reconnaissant l'intersection.
- Construire également un automate déterministe reconnaisant l'union
- Les constructions précédentes s'adaptent-elles aux automates non complet?non-déterministes ?

# 7 Devoir sur les expressions rationnelles.

Ce devoir est à rendre manuscrit, à votre chargé de TDs, le mardi 19 février pour les groupes 1 et 2, et le mardi 19 février, en cours pour le groupe 3. Il est possible de déposer le sujet chez Sandrine, mardi après midi. Un rendu mercredi 20 février vaut zéro.

Dans ce devoir, on se propose d'écrire une fonction de test d'appartenance d'un mot au langage $L(r)$ défini par une expression rationnelle $r$. Les expressions rationelles seront écrites comme en TD. Elles utilisent donc les symboles suivants: les caractères de l'alphabet, epsilon, ensemble vide, les symboles de parenthésage et d'espacement et les trois symboles '+', '.' et '*'.

1. Définir une fonction booléenne récursive `contient_epsilon` qui détermine si le mot vide $\epsilon$ appartient au langage défini par une expression rationnelle $r$ donnée. On distinguera les différent cas possible: les trois cas terminaux: $r$ est un caractère, le mot vide ou le langage vide, et les trois cas récursifs: $r$ et la somme ou le produit de deux expression rationnelles, ou l'itérée d'une expression rationnelle.

2. On utilise la notion de résidu d'une expression rationnelle par rapport à un caractère qui est le langage :

$$Res(r, c) = \{\ w \mid cw \in L(r)\ \}$$

   Donner $Res(r, c)$ dans le cas où $r = (a|b)^*a$ et $c \in \{a, b\}$.

3. Le langage $Res(r, c)$ peut lui-même être décrit par une expression rationnelle. Donner les expressions rationnelles correspondant aux deux résidus de la question précédente.

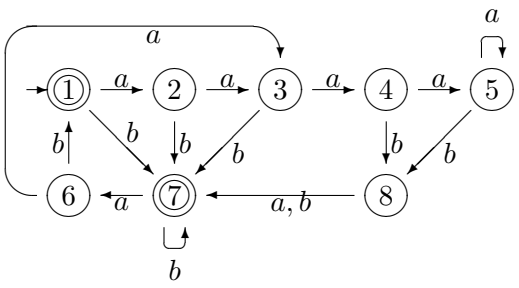
4. Définir une fonction récursive `residu` calculant à partir d'une expression rationnelle $r$ et d'un caractère $c$ une expression rationnelle $r'$ telle que $L(r') = Res(r, c)$. Comme pour `contient_epsilon`, on distinguera les différent cas possible pour $r$.

5. En déduire une fonction `reconnait` qui détermine si une liste de caractères appartient au langage défini par une expression rationnelle donnée.

# 8 Minimisation

## 8.1 Construction de l'automate minimal.

Minimisez l'automates suivant (Partiel 99-00):




## 8.2 Egalité entre automates.

Montrer que les deux automates suivants, (état initial 0), reconnaissent le même langage.

| $\delta$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| a | 1 | 2 | 1 | 3 |
| b | 3 | 1 | 3 | 3 |

état terminal : 1

| $\delta$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| a | 1 | 2 | 3 | 2 | 2 | 5 |
| b | 5 | 4 | 5 | 3 | 4 | 5 |

terminaux: 1,3,4

## 8.3 Construction de l'automate minimal à partir du langage.

### 8.3.1 Rappel de cours.

L'exercice fait intervenir des concepts un peu difficile à digérer en cours, c'est pourquoi on vous les redonne ici, résumé. Si $L$ étant un langage sur l'alphabet $A$, la relation de demi congruence syntaxique sur $A^*$, notée $\sim_L$, est définie par la relation suivante : $x \sim_L y$ ssi $\forall z \in A^*, (xz \in L \Leftrightarrow yz \in L)$ C'est une relation d'équivalence. Deux mots sont en relation pour la demi congruence., si ils ont le même avenir avec $\text{avenir}_L(u)$ qui est $\{v|uv \in L\}$. Si un automate $A$ reconnait $L$, à un état $q$ , on associe un language $Avenir_L(q)$ qui est les mots qui mènent de cet état a un final. Ce langage est précisément celui qui se calcule en

résolvant les équations associé a chaque nœud avec le Lemme d'Arden. L'avenir d'un état et egal a l'avenir d'un mot qui y mène. Deux états peuvent être fusionnés, si ils ont le même avenir. Dans l'automate minimal, il y a donc un seul état par classe d'équivalence, la classe d'un état $p$, c'est l'ensemble des mots qui vont de $q_0$ à $p$, et l'avenir de $p$, c'est les mots qui vont de $p$ a un final.

Theoreme : $L$ reconnaissable ssi si $L$ a un nombre fini de classes, le nombre de classe etant en fait le nombre d'états dans le deteriniste minimal

Si $L$ reconnaissable, alors les classes sont en nombre fini, puisque il y a une classe par état. Si il y a un nombre fini de classe, alors on fait l'automate comme suit : un état par classe, ou, ce qui revient au même et rend la chose plus compréhensible, un état par avenir distinct. On met une flèche de $p$ vers $q$ avec la lettre $a$ si Classe($p$).$a$ est inclus dans classe de $q$, L'etat initial = la classe de epsilon, Un état est final si il a epsilon dans son avenir.

#### 8.3.2 Calcul des classe d'équivalence et construction de l'automate.

Soit $A = \{a, b\}$. Pour chacun des langages cidessous , Déterminer les classes d'équivalences pour la relation de congruence syntaxique. Dire s'il est reconnaissable, et si oui, construire l'automate minimal le reconnaissant, à partir de ces classes.

On procédera en choisissant d'abord des petits mots $u$, en calculant l'avenir de $u$, puis la classe de $u$ qui est l'ensemble des mots qui on le même avenir; On choisis $u$ seulement parmi les mots qui sont des préfixes d'un mot du langage, les autres mots ont tous le même avenir: l'ensemble vide, et sont donc dans la même classe qui corresponds à un état poubelle dans l'automate minimal.

1. $A^*$
2. $\{a\}$
3. $a^*b^*$
4. $abba + ababa$
5. $\{a^n b^n, n \geq 0\}$
6. Optionnel: $\{uu | u \in A^*\}$

# 9 Le lemme de la pompe

## 9.1 Non pompabilité

Montrez, en utilisant le lemme de la pompe que les langages suivants ne sont pas reconnaissables

- $\{a^n b^{2n} | n \geq 0\}$
- $\{(ab)^n c^n | n \geq 0\}$
- $\{a^n b^m \mid n \geq m \geq 0\}$
- $\{a^n b^m \mid m \geq n \geq 0\}$
- $\{a^n b^n | n \geq 0\} + \{a^p b^q | p \neq q[7]\}$
- optionnel $\{a^n b^m | n \neq m\}$

## 9.2 Pompabilité-optionnel

Montrez que le langage suivant est pompable mais pas reconnaissable itemize $\{b^m a^n b^n \mid m > 0,\ n \geq 0\} \cup a(a+b)^*$

# 10 Clôture langages réguliers.

En utilisant les propriétés de clôture des langages rationnels et le fait que $\{a^n b^n\}$ n'est pas rationnel, montrer que les langages suivants ne sont pas rationnels :

- $L1 = \{w \in (a+b)^* \,|\, |w|_a = |w|_b\}$
- $L2 = \{a^n b^p \,|\, n \neq p\}$
- $L3 = \{a^{2n} b^{2n} \,|\, n \geq 0\}$
- $L4 = \{a^n b^p | n \geq p\}$

Pour le dernier, vous pourrez utiliser deux méthodes: Une première qui établit une relation entre $\{a^n b^p | n \geq p\}$ et $\{a^n b^p | n > p\}$. Une deuxième qui utilise la stabilité des reconnaissables par miroir.

.

# 11 Grammaires hors contexte

## 11.1 De la grammaire vers le langage

Déterminer les langages engendrés par les grammaires dont les règles de production sont les suivantes :

1. $S \to \epsilon \ \mid \ aaaS$

2. $S \to ab \mid aSb$

3. $S \to XY \mid Z$ ; $X \to Xa \mid a$ ; $Y \to aYb \mid \epsilon$ ; $Z \to aZb \mid W$ ; $W \to bW \mid b$

4. $S \to SS \mid \epsilon \mid (S)$

5. $S \to SS \mid () \mid [] \mid (S) \mid [S]$

6. $S \to \epsilon \quad S \to a_i S a_i$ pour tout $i$, $1 \leq i \leq n$

7. $S \to bS \mid aT$ ; $T \to aT \mid bU$ ; $U \to aV \mid bS$ ; $V \to aT \mid bU \mid \epsilon$

Ces grammaires sont-elles ambiguës ? Si oui, pouvez-vous donner une grammaire non-ambiguë ?

### 11.2 Du langage vers la grammaire

Trouver des grammaires pour les langages suivants.

1. $(a + (a + b)^*)(ab^*)^*$
2. $\{a^n b^p \mid 0 < p < n\}$
3. $\{a^n b^p \mid 0 \leq n \leq p + 1\}$
4. $\{a^n b^n c^m d^m \mid n, m \in \mathcal{N}\}$
5. $\{a^n b^m c^{n+m} \mid n, m \in \mathcal{N}\}$
6. $\{a^n b^m c^p \mid n = m \text{ ou } m = p\}$
7. optionnel $\{a^n b^m c^p d^q \mid n + q = m + p\}$
8. optionnel $\{a^n b^m c^p d^q \mid n + p = m + q\}$

### 11.3 Désambiguiser à la main

Soit $F_1$ la grammaire

$$E \to E + E \mid E - E \mid (E) \mid id$$

et $G_1$ la grammaire $F_1$ plus les règles:

$$E \to E * E \mid E/E \mid E \wedge E$$

1. Donner tous les arbres de dérivations du mot $id - id - id$. Combien y en a-t-il? Correpondent-ils à des interprétations équivalentes ?

2. Donner des grammaires $F_2$ et $G_2$ telles que $L(F_1) = L(F_2)$, que $L(G_1) = L(G_2)$, que chaque mot $w$ possède une seule dérivation à partir du symbole initial de $G_2$, et que la décomposition en arbre corresponde au regles usuelles de priorité.

## 12 Grammaire et compilation.

Faut avoir parlé d'analyse lexicale en cours.

### 12.1 Analyse lexicale

l'utilisation d'ocamllex n'est pas limitée a l'analyse lexicale des que l'on souhaite analyser un texte (chaîne, fichier, flux) sur la base d'expressions régulières, ocamllex est un outil de choix en particulier pour écrire des filtres, i.e. des programmes traduisant un langage dans un autre par des modifications locales et relativement simples.

Écrire un programme occamlex qui imprime un fichier en ayant préalablement enlevé toutes les lignes vides, et un autre qui compte les occurrences d'un mot dans un texte le mot et le nom du fichier texte sont passés en paramètres

### 12.2 Grammaire d'un Language de programmation

Considérons le petit programme suivant écrit en Pascal:

```
program calcul;
var
      T : array[1..10] of integer;
      S,I : integer;
begin
      S:=0; (* initialisation *)
      for I:= 1 to 10 do
      begin
            read(T[I]);
            S := S + T[I]
      end;
      writeln(S)
end.
```

L'analyseur lexical découpe ce programme en une liste des entités lexicales appelées token dont nous donnons le début . Chaque token est donné par une classe et sa valeur. Pour les identificateurs, la valeur sera la chaine de caractére, ou mieux, l'adresse d'entrée dans une table des symboles.

La table des symboles est supposée découpée en une zone pour les mots-clés 0 à 9 et une zone pour les identificateurs à partir de 50. Cette table est composée d'un champ représentant

la chaîne de caractères et d'un champ pouvant contenir différentes informations utiles à l'analyse sémantique.

On suppose les différentes entités rangées dans l'ordre de leur apparition, en fait les mots-clés sont en général rangés préalablement dans la table.

Les symboles ;: [],.,.. sont associés dans l'ordre à des tokens de classe 11 à 17 comme il n'y a qu'une unité lexicale dans chacune de ces classes il n'est pas nécessaire de passer de valeurs.

| | | | | | |
|---|---|---|---|---|---|
| program <0> | calcul <-1,50> | ; <11> | | | |
| var <1> | T <-1,51> | : <12> | array <2> | [ <13> | 1 <3,1> |

La table des symboles après analyse ressemble à :

| adresse | chaîne | information |
|---|---|---|
| 0 | program | |
| 1 | var | |
| 2 | array | |
| 3 | of | |
| 4 | integer | |
| 5 | begin | |
| 6 | for | |
| 7 | to | |
| 8 | do | |
| 9 | end | |
| ⋮ | ⋮ | |
| 50 | calcul | |
| 51 | T | |
| 52 | S | |
| 53 | I | |
| 54 | read | |
| 55 | writeln | |
| ⋮ | ⋮ | |

1. Comment peut on reconnaitre les mots clefs de facon simple, lors de l'analyse lexicale ?
2. Que code le numéro de classe?
3. Les mots clefs sont ils tous de la même classe?
4. A quoi corresponds la classe -1, sur cet exemple?
5. Quelle valeur a un token identificateur?
6. Quelle valeur a un token mot clef?
7. Quelle valeur a un token constante entiére, et comment la calculer?
8. Proposer une grammaire permettant d'engendrer le langage auquel ce programme appartient.
9. Donner l'arbre de dérivation syntaxique associé à ce programme pour la grammaire précédente. On n'est pas obligé de le dessiner en entier, car il est très grand.

# 13 Optionnel grammaire

## 13.1 Nettoyage de grammaires

On veut nettoyer une grammaire, c'est à dire enlever :

(1) Les non-terminaux impasse, c'est à dire qui ne produisent pas de mots sur $A^*$

(2) les non-terminaux inaccessibles , c'est à dire qui ne figurent dans aucune dérivation faite à partir de $S$.

Donner un algorithme permettant de repérer (et donc d'éliminer) les impasses

Donner un algorithme permettant de repérer (et donc d'éliminer) les inaccessibles

Quand on veut faire un nettoyage complet, l'ordre dans lequel on effectue ces deux opérations est-il indifférent ? Pourquoi ?

Nettoyer la grammaire :

$S \to X \quad X \to Y \quad Z \to W|eS$
$W \to b|fX \quad Y \to aT|TK$
$U \to bdX|Y|dZ \quad K \to cV|Z$
$X \to abcY \quad W \to U$
$T \to aT|\epsilon|ef|aY \quad V \to af$

## 13.2 Désambiguation difficile.

Soient $D_1, D_2$ et $D_3$ les langages suivants :

$$D_1 = \{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$$

$$D_2 = \{w \in \{a,b\}^* \mid |w|_a = |w|_b \text{ et}$$

$$\forall v \text{ préfix de } w \ \ |v|_a \geq |v|_b\}$$

$$D_3 = \{w^R \in \{a,b\}^* \mid w \in D_2\}$$

On veut donner une grammaire pour $D_1$ non ambigue. Montrer comment obtenir $D_1$ avec les opérations de concaténation et d'étoile à partir du langage $D_2$ (Dyck d'ordre 1) et de son miroir $D_3$. Utiliser cette description et les résultats de clôture pour lui trouver une grammaire non-ambiguë.

## 13.3 Grammaire difficile à trouver

Donnez une grammaire pour $\{w \in (a+b)^* \mid |w|_b = 2|w|_a\}$

### 13.4 Grammaires contextuelles

On considère l'ensemble de règles de réécriture suivant :

$S \to aTc \quad aT \to aaTcT \quad cT \to Tc \quad aT \to ab \quad bT \to bb$

Quel est l'ensemble des mots sur $\{a, b, c\}^*$ dérivables à partir de $S$ ?

## 14 Mini Projet à faire à deux.

Le projet est à faire en binôme (par groupe de deux). Vous enverrez une archive **avant le mardi 26 mars 23H59** à votre responsable de TD qui est Frédéric Gruau, gruau@lri.fr pour les groupes G1 et G3, Jonas Sénizergues, senizergues@crans.org pour le groupe G2. Un programme qui ne s'exécute pas, ou un envoi tardif est noté zéro. L'usage du langage Caml est obligatoire. Le projet consiste à implémenter en CAML l'algorithme mis en place dans le devoir. Celui ci prend en entrée une chaine de caractères et une expression rationnelle et détermine si la chaine de caractères est un mot du langage défini par l'expression rationnelle.

Les expressions rationnelles seront représentées par des objets du type Caml suivant:

```
type          expreg =
| Vide
| Epsilon
| Caractere   of char
| Union       of expreg * expreg
| Produit     of expreg * expreg
| Etoile      of expreg
```

1. Ecire une fonction Caml `contient_epsilon : expreg → bool` qui détermine si le mot vide $\epsilon$ appartient au langage défini par une expression rationnelle donnée.

2. On définit le *résidu* d'une expression rationnelle $r$ par rapport à un caractère $c$ de la manière suivante :

$$Res(r, c) = \{\ w \mid cw \in L(r)\ \}$$

   (a) Le langage $Res(r, c)$ peut lui-même être décrit par une expression rationnelle. Écrire une fonction Caml `residu : expreg → char → expreg` calculant à partir de $r$ et de $c$ une expression rationnelle $r'$ telle que $L(r') = Res(r, c)$.

   (b) Tester en calculant `residu` $r$ $c$ dans le cas où $r = (a|b)^*a$ et $c \in \{a, b\}$.

3. En déduire une fonction `reconnait : expreg → char list → bool` qui détermine si une liste de caractères appartient au langage défini par une expression rationnelle donnée.

4. Appliquer cette fonction à la reconnaissance du mot $aba$ par l'expression rationnelle $r = (a|b) * a$

## 15 Automates à pile

Dans la mesure du possible, donnez des automates à pile déterministe.

1. Construire un automate à pile qui reconnaît par état final $\{a^n b^n | n \geq 1\}$, puis $\{a^n b^n | n \geq 0\}$.

2. Construire un automate à pile qui reconnaît le langage $\{a^p b^n c^q | p \geq 0, q \geq 1, n = p + q\}$ par pile vide.

3. Construire un automate à pile qui reconnaît le langage des mots de Dyck sur 1 puis sur 2 types de parentheses.

4. Construire un automate à pile qui reconnaît le langage des mots qui ont autant de a que de b (deux solutions: la premiere n'utilise qu'un état, la seconde qu'un seul symbole de pile)Dans chaque case on met les membre droit possibles, on constate qu'il ny' a qu un choix donc pas de conflit.

5. Construire un automate à pile qui reconnaît le langage des palindromes.

6. Cherchez des automates a pile qui reconnaissent $\{a^n b^n c^n | n \geq 0\}$ et $\{a^n b^m a^n b^m | n, m \geq 0\}$.

## 16 Est il Algébrique? Clôture / pompe.

Il y a quatre méthode possibles pour répondre à cette question.: Un langage est algébrique, si on peut le générer par une grammaire hors contexte. Les propriétés de clôtures permettent également de déterminer si un langage est

algébriques. La contraposée du lemme de la pompe algébrique permet de prouver que un langage n'est pas algébrique. Les algébriques sont aussi ceux que l'on peut générer par automates à pile. Les langages suivants sont-ils algébriques ?

1. $\{a^n b^m \mid m \neq n \text{ et } m \neq 2n\}$
2. $\{a^n b^m a^n b^m \mid n, m \in \mathcal{N}\}$
3. $\{a^{n^2} \mid n \in \mathcal{N}\}$
4. $\{u \mid \ |u|_a = |u|_b = |u|_c\}$
5. $\{uu \mid u \in A^*\}$
6. optionnel Le complémentaire du précédent.
7. $\{u \mid \ |u|_a + 3|u|_b = 2|u|_c\}$
8. $\{u \mid \ |u|_a = 3|u|_b = 2|u|_c\}$
9. $\{a^n b^n a^n b^n | n \geq 0\}$
10. optionnel $\{a^p b^q c^r | p \leq q \leq r\}$
11. optionnel $\{a^n b^n (ab)^n | n \geq 0\}$
12. optionnel $\{f(y) \mid y \in Y\}$ où $Y$ est algébrique et ù $f(a_1\, a_2\, a_3\, a_4\, a_5 ...) = a_1\, a_3\, a_5\, ...$ ($f$ efface les lettres qui sont à une position paire)
13. optionnel $\{a^p b^q c^r d^s e^t f^u | (p, q, r, s, t, u)$ croit ou décroit $\}$

# 17 Analyse Syntaxique.

## 17.1 Analyse ascendente à la main.

Le cours d'analyse syntaxique ascendante sera fait la semaine prochaine, néanmoins ce TD introduit gentiment un exemple concret sur ce thème. De cette façon, les notions du cours, plus abstraites, seront plus facilement comprises. Soit la grammaire suivante:

$$\begin{array}{lll|lll} E & ::= & E+T & E & ::= & T \\ T & ::= & T*F & T & ::= & F \\ F & ::= & id & F & ::= & cte \end{array}$$

1. Que reconnait t'elle? est elle ambigue?
2. Utiliser la grammaire pour générer la chaîne *id*id+cte* par une dérivation droite.
3. On considére un formalisme étendu d'automate à pile qui permet de dépiler un nombre arbitraire de symboles de la pile. Cela change t'il la puissance du modèle?
4. Écrivez l'automate à pile suivant, pour cette grammaire: Il utilise un seul état et deux sortes de transitions: 1- pour chaque régle $X \to \alpha$ ne rien lire du mot, dépiler $\alpha$ empiler $X$. Ces transitions sont appelées "reduction" 2- pour chaque terminal $a$ une transition qui lit $a$ et empile $a$, ces transitions sont appellées "shift", ou "lecture".
5. L'automate de la question précédente permet de reconnaître le langage associé à la grammaire, avec une analyse ascendante, I.e en remontant des feuilles vers la racine. Reconnaître la chaîne *id*id+cte*. On mettra la colonne de l'état de pile à gauche de celle de l'état du mots.
6. Cet automate n'est pas déterministe, préciser pourquoi:
7. Est ce que c'est génant?
8. Ben KesKiFautfaire alors?
9. Un peu d'introspection, vous-même, quelle stratégie avez vous suivi pour orienter vos choix, lorsque vous avez utilisé l'automate à la main.
10. L'analyse LR(1) autorise un automate a pile à consulter quelle est la prochaine lettre du mot à lire, sans pour autant la "consommer" Mais alors, quelle sera cette lettre, lorsqu'on sera arrivé au bout du mot?

## 17.2 Calcul premiers et suivants

Soit la grammaire:

$$\begin{array}{lll|lll} S & ::= & AaB & & & \\ A & ::= & CB & A & ::= & CBb \\ A & ::= & \epsilon & A & ::= & CAd \\ B & ::= & b & & & \\ C & ::= & c & C & ::= & \epsilon \end{array}$$

1. Pour chaque non terminal $X$ calculer premier($X$) (commencer par écrire les équations)
2. Pour chaque non terminal $X$ calculer suivant($X$) (commencer par écrire les équations)

### 17.3 Exemple simple d'automate SLR(1), et son exécution.

Soient les grammaires :

| | | |
|---|---|---|
| S | ::= | L |
| L | ::= | L;A \| A |
| A | ::= | a |

| | | |
|---|---|---|
| S | ::= | L |
| L | ::= | A;L \| A |
| A | ::= | a |

| | | |
|---|---|---|
| S | ::= | L |
| L | ::= | L;L \| A |
| A | ::= | a |

1. Montrer que ces grammaires engendrent le même langage.
2. Pour chaque grammaire si elle est LR(0), si nécessaire construire l'automate d'item et identifiez les conflits, puis essayer de les résoudre en utilisant l'automate SLR(1).
3. Faire tourner l'automate et comparer la taille de la pile lors de l'analyse ascendante du mot $a; a; a$ par les deux premières grammaires. Quelle remarque peut-on faire ?
4. Pour bien montrer que l'automate LR(0) reconnait les mots de piles, pour les deux premières grammaires, dessiner le diagramme de l'automate avec des états et des fléches et calculer le langage qu'il reconnait et vérifier que c'est bien le bon.

### 17.4 Analyse syntaxique des expressions arithmétiques;

Soit la grammaire déja abordée la semaine derniére:

| | | |
|---|---|---|
| E | ::= | E+T —T, |
| T | ::= | T *F —F |
| F | ::= | Id — Cte |

Construire L'automate LR(0) puis l'automate SLR(1) si nécessaire.

### 17.5 Autre example d'analyseur LR

| | | |
|---|---|---|
| S | ::= | E ♯ |
| E | ::= | `id` |
| E | ::= | `id`(E) |
| E | ::= | E + `id` |

Construisez l'automate LR(0) permettant de faire l'analyse ascendante Cet automate présente un conflit, indiquer l'état ou il se trouve, et entre quoi et quoi il y a conflit Expliquer comment résoudre ce conflit

### 17.6 Exemple plus difficile, l'analyse SLR(1) ne marche pas.

On se donne un langage de types permettant de décrire le type des entiers ou celui de fonctions à valeur entière, prenant en argument des entiers ou d'autres fonctions de même nature.
Un type est donc soit la constante `int` soit de la forme $\tau_1 * \ldots * \tau_n \to$ `int` avec $\tau_i$ des types. Pour reconnaître ce langage de types, on se donne la grammaire suivante avec comme ensemble de terminaux $\{\sharp, \to, *,$ `int`$\}$ et comme ensemble de non-terminaux $\{S, A, T\}$ avec $S$ le symbole de départ:

| | | |
|---|---|---|
| S | ::= | T ♯ |
| T | ::= | `int` |
| T | ::= | A →`int` |
| A | ::= | T |
| A | ::= | T * A |

1. Calculer les suivants de $T$ et de $A$.
2. Construire la table d'analyse SLR(1) de cette grammaire en indiquant en cas de conflit les différentes actions possibles. Cette grammaire est-elle SLR(1) ?
3. Expliquer la nature du conflit obtenu en donnant un exemple d'entrée où ce conflit se produit. La grammaire donnée est-elle ambiguë ?
4. Que suggérez-vous pour remédier à ce problème ?

## 18 Machine de Turing

Dessiner une Machine de Turing pour les différentes tâches suivantes. On indiquera à quoi servent les états, on pourra aussi dessiner les états de la machine, en particulier lorsque les conditions de sortie d'une boucle sont vérifiées.

- ajoute 1 à une séquence de 1
- ajoute 1 à un nombre écrit en binaire.
- qui reconnaît le langage $\{a^{2^n} / n \in \mathbb{N}\}$
- qui duplique le mot en entrée
- qui reconnaît le language $\{ww, w \in \{0,1\}^*\}$.
- qui calcule la fonction $n \mapsto n+1$.

# 19 Decidabilité

## 19.1 Decidabilité problème du mot

Soit $H$ une grammaire et $u = u_1\, u_2 \ldots u_{|u|}$ un mot. On voudrait savoir si le mot $u$ est dans le langage engendré par $H$. Pour cela, on met d'abord $h$ sous forme normale de Chomsky, (Rappel les règles sont de la forme $M \to \epsilon; X \to YZ; X \to x$ ; ) puis on remplit un tableau $T$ de type `array[0..|u|,0..|u|] of subset de NT` (où NT est l'ensemble des non terminaux de $H$) avec $M$ dans $T[i,j]$ pour $j \geq i$ ssi $M \to^* u_{i+1}\, u_{i+2} \ldots u_j$. ($T[i,j]$ pour $j < i$ est sans signification). Comment calculer les valeurs de ce tableau ? Comment déduire de ce tableau le fait que $u$ est dans le langage ou non ? Quelle est la complexité de cet algorithme ?
On considérera l'exemple ou la grammaire est $S \to \epsilon; S \to aSb$ qui génére $\{a^n, b^n\}$ On souhaite montrer que $aabb$ est dans le langage, on indexe les lettre $a_1 a_2 b_3 b_4$ et on réécrit dans chaque case le sous mot associé a la case.

## 19.2 Indecidabilité problème de Post

Définition problème de Post: Les données du problème sont deux listes finies $\alpha_1, \ldots, \alpha_N et \beta_1, \ldots, \beta_N$ de mots d'un alphabet A ayant au moins deux symboles. Une solution du problème est une suite d'indices $(i_k)_{1 \leq k \leq K}$ avec $K \geq 1 et 1 \leq i_k \leq N$ pour tous les $k$, telle que les concaténations $\alpha_{i_1} \ldots \alpha_{i_K} et \beta_{i_1} \ldots \beta_{i_K}$ soient égales.

Le problème de correspondance de Post (PCP) consiste à déterminer si une solution existe ou non.

Résoudre ce problème pour les deux exemples suivants:

- $(\alpha_1, \alpha_2, \alpha_3) = (a, ab, bba)$ et $(\beta_1, \beta_2, \beta_3) = (baa, aa, bb)$
- (bb,ab,c) et (b,ba,bc)
- Sur ce pb, on donne les paires $(\alpha_i, \beta_i)$ : (#, #p00000000#), (0, 0), (1, 1), (#, #), (p0, 0p), (p#, q#), (0q, 1p), (1q, q0), (# q0, #q), (#q#, #)

## 19.3 Indecidabilité de l'ambiguïté

A un problème de Post $(u_i, v_i) i \in \{1, \ldots N\}$ sur l'alphabet A = a, ..., z , associons les grammaires suivantes :

```
S2-> $ | #S2 | T
T->T# | U
U->aS2 a | ... | zS2z

S1 ->u1 # S1 # miroir(v1 )
   ->u2 # S1 # miroir(v2 )
....

  -> uN # S1 # miroir(vN )

  ->  u1 # $ # miroir(v1 )
  ->  u2 # $ # miroir(v2 )
 ...
 -> uN # $ # miroir(vN )

S -> S1 | S2
```

- Que génére la grammaire S2
- Que génére la grammaire S1
- A quelle condition la grammaire avec l'axiome S est elle ambigue
- Donner un exemple de grammaire ambiguë en utilisant les système de post donnés en exemple.
- En déduire que le problème de savoir si une grammaire est ambiguë est indécidable .

## 19.4 Indécidabilité de l'intersection.

Considérons le problème suivant: soit $G_1, G_2$ deux grammaires, peut on décider si l'intersection des langages qu'elles génére est non vide?

# 20 TP d'analyse syntaxique.

L'objectif du TP est de comprendre le fonctionnement d'un analyseur syntaxique associé à une grammaire puis d'étendre son fonctionnement en lui ajoutant des éléments. Nous utiliserons pour cela l'outil **Yacc**, un générateur d'analyseurs syntaxique associé au générateur d'analyseurs lexical **Lex** dans leur versions pour Ocaml (ocamlyacc et ocamllex).

## 20.1 Analyseur syntaxique simple

Le squelette de base d'un analyseur syntaxique vous est fournis. Les programmes `express.mli`, `express.ml`, `ana_lex.mll`, `ana_synt.mly` et `test_expr.ml` vous permettent de générer un analyseur syntaxique et de l'utiliser pour effectuer une analyse permettant d'afficher l'arbre de syntaxe abstraite d'une expression arithmétique ne comprenant que des additions et des soustractions.

Dans un premier temps, vous analyserez le squelette de l'application et la modifierez pour ajouter des éléments à la grammaire.

- Lisez les différents fichiers et comprenez leur fonctionnement. Écrivez la grammaire reconnu par l'analyseur syntaxique.
- Compiler le programme en utilisant la commande *make*. Quelle est la signification du message adressé par Ocamlyacc ? Que se passe t-il quand vous exécuter tout de même le programme ?
- Modifier la définition de la grammaire pour enlever l'ambiguïté
- Rajouter le cas de la multiplication et de la division. Celle-ci doivent être prioritaire sur l'addition et la soustraction. Ajouter ensuite le cas des expression parenthésées. *D'autres fichiers sont amenés à être modifier pour observer les résultats*
  On pourra tester $3+4*5+2$ et tester que ca calcule bien 25, i.e que le parenthésage implicite se fait autours de (4*5) à cause des règles de précédence.

## 20.2 Modification du programme pour calculer la valeurs des expressions

Comme vous avez pu le constater précédemment, **Yacc** permet d'exécuter du code lors de la construction de l'arbre syntaxique. En effet, ocamlyacc renverra le résultats du code placé dans les `{}` suivant une règle de grammaire lorsqu'il l'aura entièrement reconnue. Il est possible d'utiliser les variables $\$_i$ représentant la valeur renvoyé par le $i^{eme}$ membre de la règle.

Dans cette partie, vous serez amenez à modifier ce code afin de ne plus afficher l'arbre de dérivation syntaxique des expressions.

- Modifiez l'application pour calculer la valeur plutôt que l'arbre syntaxique de l'expression.
- Rajouter des registres pour que le programme puisse maintenant affecter des variables et les utiliser. Il vous faudra pour cela modifier la grammaire de façon plus significative.

  `[A<-6][B<-7]2*B+A` est une expression valide ou les variables $A$ et $B$ peuvent être utilisé dans le calcul. Ici, l'expression est évalué a 20.