Feuilles de TD Langages Formels 2016-2017

1 Expression rationnelle

1.1 Expression rationnelle 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 $bba\}$

1.2 Expression rationnelle utilisée en pratique

1. L'outil fgrep sous unix permet de filtrer les lignes d'un fichier qui contiennent une sous chaine matchant une expression rationnelle. On utilise le caractère | pour écrire la somme + et le produit '.' est implicite. Ecrire une commande fgrep qui filtre les lignes contenant le mot malloc ou free d'un fichier toto.c
2. Lorsque un script saisit une adresse email sur une page web, il peut commencer par reprérer une eventuelle faute de frappe en vérifiant le plus possible que l'utilisateur a bien rentré quelque chose qui ressemble à une adresse email ; si cela n'est pas le cas, il peut redemander l'adresse. Pour cela, le script peut utiliser l'automate associé à l'expression rationnelle décrivant les adresses email. On suppose que les identifiant contiennent seulement des lettres , et eventuellement un '.' pour séparer le prénom du nom. Ecrire cette expression. on utilisera les notation simplifié standard $e^+ = e.e^*$
3. La premiére étape d'un compilateur et l'analyse lexicale, qui découpe le texte d'un programme en unité lexicale appélée token. Un token peut etre un mot clef, un identifiant, une constante numérique. On utilise des expression rationnelle pour identifer la nature des différent token. Ecrire l'expression rationnelle decrivant un identificateur comme une lettre suivit d'une suite de lettre ou de chiffre, un entier, un flottant. On utilisera LA simplifiation de notation $e? = e|\epsilon$ qui signifie que e est optionnel.
4. Comment peut faire l'analyseur lexical pour répérer les different mot clefs de facon simple ?

1.3 Expression rationnelle d'un language, plus 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 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. $(L^*.M)^* = \Lambda + (L + M)^*.M$

3 Automates reconnaissant un langage donné.

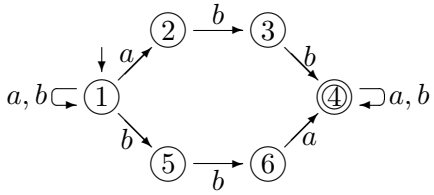
Donner des automates reconnaissant les langages suivants : (pour les entiers comme pour les mots habituel, on considéres que les caractères sont lus de gauche a droite, donc en commencant par les bits de poids forts)

- 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$\}$
- $A = \{a, b\}$, $L = \{w \mid w$ 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 flottants
— $A = \{/, *, c\}$, L est l'ensemble des commentaires, un commentaire étant un mot de la forme $/ * w * /$, et dans laquelle w ne contient pas d'occurence 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

(Partiel 2000-01). Determininisez :



4.1 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.

4.2 Le barman,boxeur aveugle

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 decrivent les évolutions possibles des configurations.

2) 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).

3) Donnez un automate qui donne les coups qui assurent au barman de gagner quel que soit le comportement du client.

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

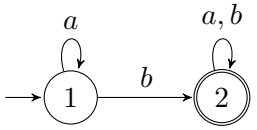
Cible : On commence par bien maîtriser l'algorithme de minimisation sur deux exemples, puis l'exercice sur le calcul des résudus, (futur) permet d'avoir plus de recul par rapports au fondement théorique de la minimisation. Enfin l'exercice sur les congruences permet de se familiariser avec les relations d'équivalences.

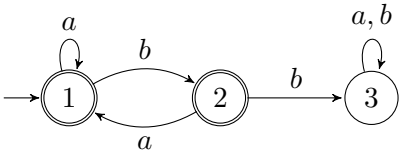
5 Resolution d'équations

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.

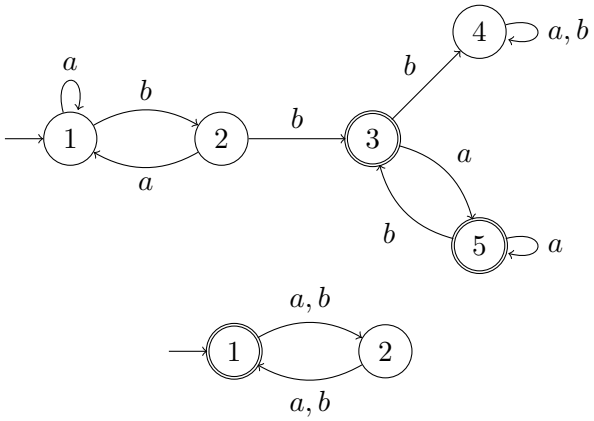
5.1 Exemple introductif

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 : (surl'alphabet $\mathcal{A} = \{a, b\}$)





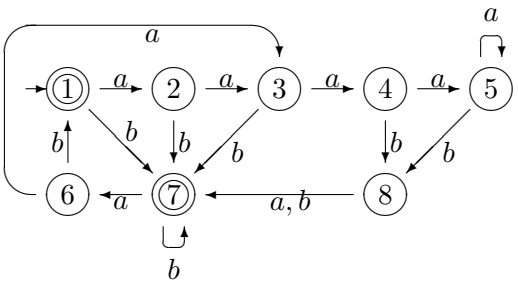
5.2 Exemple optionnel.



6 Minimisation

6.1 Construction de l'automate minimal.

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



6.2 Egalité entre automates.

Montrer que les deux automates suivants reconnaissent le même langage.

δ	0	1	2	3
a	1	2	1	3
b	3	1	3	3

Etat initial : 0, état terminal : 1

δ	0	1	2	3	4	5
a	1	2	3	2	2	5
b	5	4	5	3	4	5

Etat initial : 0, états terminaux : 1,3,4

6.3 Congruences

Rappel de cours : 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'equivalence. 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 menent de cet etat a un final. Ce language est précisement celui qui se calcule en résolvant les equations associé a chaque noeud avec le Lemme d'arden. L'avenir d'un etat et egal a l'avenir d'un mot qui y mene. Deux etats peuvent etre mergés, si ils ont le meme avenir. Dans l'automate minimal, il y a donc un seul etat par classe d'equivalence, 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'etat dans le deteriniste minimal

Si L reconnaisable, alors les classes sont en nombre fini, puisque il y a une classe par etat. Si il y a un nombre fini de classe, alors on fait l'automate comme suit : un etat par classe, ou, ce qui revient au même et rend la chose plus comprehensible, un etat par avenir distinct. On met une fleche de p vers q avec la lettre a si Classe(p).a est inclus dans classe de q, ou (equivalent) si $a^{-1}Avenir(p) = Avenir(q)$ L'etat initial = celui de la classe contenant epsilon, ou dont l'avenir est celui de epsilon. Un etat est final si il a epsilon dans son avenir.

6.3.1 Calcul de classe d'equivalence

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, donner l'automate minimal le reconnaissant.

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 chosis u seulement parmis les mots qui sont des prefixes d'un mot du language, les autres mots ont tous le meme ave-

nir : 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. $\{a^nb^n, n \geq 0\}$
6. $\{uu|u \in A^*\}$
7. Les mots de longueur au moins égale à 4 et dont la 4eme lettre est un b

6.4 Raisonnent sur des relations d'equivalence (difficile).

Soit L un langage, et $\sim_L$ sa demi-congruence syntaxique.

On note $\bowtie_L$ la relation sur A^* définie par

$$u \bowtie v \iff (\forall x, y \in A^*, xuy \in L \Leftrightarrow xvy \in L)$$

Montrez que :

— $\bowtie$ est une relation d'équivalence.
— Si $u \bowtie v$ alors $u \sim v$
— Si $u \bowtie v$ et $\bar{u} \bowtie \bar{v}$ alors $u\bar{u} \bowtie v\bar{v}$
— L est reconnaissable ssi le nombre de classes d'équivalences pour $\bowtie$ est fini.
— S'il y a N classes d'équivalence pour $\sim$, combien au maximum y-en-t-il pour $\bowtie$?

7 Transformation,construction d'automates.

L est reconnu par l'automate A. 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.
— l'ensemble des mots obtenus en effaçant un nombre pair de lettres d'un mot de L
— le complémentaire de L, en supposant A déterministe.

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

— Construire un automate reconnaissant l'union $L_1 + L_2$.
— Si A_1 et A_2 n'ont pas d'epsilon transitions, peut-on construire facilement un automate sans epsilon-transitions pour l'union ?

On suppose que A_1 et A_2 sont deterministes 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 precedentes s'adaptent-elles aux automates non complet ?non-deterministes ?

8 Clôture langages réguliers.

8.1 Cloture simple, sans morphisme

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

— $\{w \in (a+b)^* \,|\, |w|_a = |w|_b\}$
— $\{a^nb^p \,|\, n \neq p\}$
— $\{a^{2n}b^{2n} \,|\, n \geq 0\}$
— $\{a^nb^p | n \geq p\}$

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

8.2 Image inverse de morphisme

Soit ϕ le morphisme défini par $\phi(a) = a.b$, $\phi(b) = a$, $\phi(c) = b.a$ et $\phi(d) = bbb$.

Décrire $\phi(abcd)$, $\phi((ab)^*)$, $\phi^{-1}(aba)$, $\phi^{-1}((bba)^*)$, $\phi^{-1}((ab)^*a)$, $\phi^{-1}((bb)^*)$

8.3 Cloture avec Morphisme

Montrer que les langages suivants ne sont pas rationnels :

— $\{a^nb^nc^n \,|\, n \geq 0\}$
— $\{a^ib^jc^k \,|\, i + j = k \geq 0\}$
— $\{a^nb^{2n} \,|\, n \geq 0\}$
— $\{(ab)^{2n}(cd)^{2n} \mid n \geq 0\}$
— $\{a^nba^n \,|\, n \geq 0\}$
— $\{w \in (a+b)^* \,|\, w$ est un palindrome $\}$

8.4 Optionel

Soit f la fonction qui efface toutes les lettres en position paire. $f(a_1a_2a_3...) = a_1a_3a_5...$

Exemple, $f(abcdefghijklmn) = acegikm$, $f(abbababba) = abbba$.

Montrez que si L est reconnaissable, alors $f(L)$ aussi.

8.5 Demonstration d'egalité entre langage, utilisant les morphismes.

Soit L et M deux langages et soit ϕ un homomorphisme. Montrer ou infirmer (en exhibant un contre-exemple) les propriétés suivantes :

1. $\phi(L \cup M) = \phi(L) \cup \phi(M)$
2. $\phi(L.M) = \phi(L).\phi(M)$
3. $\phi(L \cap M) = \phi(L) \cap \phi(M)$
4. $\phi^{-1}(L \cup M) = \phi^{-1}(L) \cup \phi^{-1}(M)$
5. $\phi^{-1}(L \cap M) = \phi^{-1}(L) \cap \phi^{-1}(M)$
6. $\phi^{-1}(\phi(L)) = L$

9 Le lemme de la pompe

9.1 Non pompabilité

Montrez, en utilisant le lemme de la pompe (version du cours) que les langages suivants ne sont pas reconnaissables

— $\{a^n b^{2n} | n \geq 0\}$
— $\{(ab)^n c^n | n \geq 0\}$
— $\{a^n b^n | n \geq 0\} + \{a^p b^q | p \neq q[7]\}$
— $\{a^n b^m \mid n \geq m \geq 0\}$
— $\{a^n b^m \mid m \geq n \geq 0\}$
— $\{a^n b^m | n \neq m\}$
— $\{a^p b^q | p = q[2]\} + \{ba^{p+r}b^p | p, r \geq 0\} + \{b^s a^p b^{p+r} | s \geq 2, p, r \geq 0\}$
— $\{a^p \mid p \text{ premier }\}$

9.2 Pompabilité

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 Grammaires hors contexte

10.1 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. $S \to \epsilon \qquad S \to a_i S a_i$ 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 ?

10.2 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 $\{a^n b^m c^p d^q \mid n + p = m + q\}$

10.3 Desambiguiser 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.

11 Grammaire et compilation.

Faut avoir parlé d'analyse lexicale en cours.

11.1 Analyse lexicale

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

Ecrire un programme occamlex qui imprime un fichier en ayant prealablement 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 paramétres

11.2 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 l'adresse d'entrée dans une table des symboles. Par exemple, la classe est 1 pour les mot clés, 2 pour les identificateurs.

La table des symboles est supposée découpée en une zone pour les mots-clés 0 à 50 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 16 comme il n'y a qu'une unité lexicale dans chacune de ces classes il n'est pas nécessaire de passer de valeurs.

program <1,0>	calcul <2,50>	; <11>						
var <1,1>	T <2,51>	: <12>	array <1,2>	[<13>	1 <3,1>	.. <8>	10 <3,10>	] <14>
S <2,52>	, <15>	I <2,53>	: <12>	integer <1,4>	; <11>			
begin <1,5>	S <2,52>	:= <6>	0 <3,0>	; <11>				
for <1,6>	I <2,53>	:= <3,1>	1 <8>	to <1,7>	10 <3,10>	do <1,8>	... ...	

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	read	
10	end	
11	writeln	
⋮	⋮	
50	calcul	
51	T	
52	S	
53	I	
⋮	⋮	

1. Proposer une grammaire permettant d'engendrer le langage auquel ce programme appartient.
2. 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.

12 Exercices optionnel, approfondissement grammaire

A ne faire que s'il reste du temps.

12.1 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-termianux 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 :

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

12.2 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.

12.3 Grammaire difficile à trouver

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

12.4 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 ?

13 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 $\{a^n b^m a^n b^m | n, m \geq 0\}$.

14 Analyse Syntaxique.

14.1 Fonctionnemment de l'automate a pile non deterministe

Le cours d'analyse syntaxique ascendente sera fait la semaine prochaine, néanmoins le 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 :

E	$::= E + T$	E	$::= T$
T	$::= T * F$	T	$::= F$
F	$::= id$	F	$::= cte$

1. Que reconnait t'elle ? est elle ambigue ?
2. Utiliser la grammaire pour générer la chaîne *id*id+cte* par une dérivation gauche.
3. Ecriver l'automate à pile qui permet de reconnaitre le langage associé à cette grammaire en utilisant la méthode vue en cours.

4. Utiliser l'automate de la question précédente pour reconnaitre la chaîne *id*id+cte*
5. Cet automate n'est pas déterministe, préciser pourquoi :
6. Est ce que c'est génant ?
7. Ben KesKiFautfaire alors ?
8. Un peu d'introspection, vous-même, quelle stratégie avez vous suivi pour orienter vos choix d'expansion de S, lorsque vous avez utilisé l'automate à la main.
9. 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 ?

14.2 Calcul premiers et suivants

Soit la grammaire :

S	$::= AaB$		
A	$::= CB$	A	$::= CBb$
A	$::= \epsilon$	A	$::= CAd$
B	$::= b$		
C	$::= c$	C	$::= \epsilon$

1. Pour chaque non terminal X calculer premier(X) (commencer par ecrire les equations)
2. Pour chaque non terminal X calculer suinvant(X) (commencer par ecrire les equations)

14.3 Exemple simple d'automate SLR(1), avec exécution de l'automate LR

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'autoamte 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 ?

14.4 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 conflite

14.5 Exemple un peu plus difficile, ou l'analyse SLR(1) ne marche pas, donné en Examen.

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 $\tau_1 * \ldots * \tau_n \rightarrow$ `int` avec τ_i des types. Pour reconnaître ce langage de types, on se donne la grammaire suivante avec comme ensemble de terminaux $\{\sharp, \rightarrow, *,$ `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 l'ensemble des 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 ?

15 Clotures algébriques

Les langages suivants sont-ils algébriques ?

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

16 Machine de Turing

Décrire une Machine de Turing pour les différentes tâches suivantes :

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

17 Decidabilité

17.1 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.

17.2 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 (α_i, β_i) : (#, #p00000000#), (0, 0), (1, 1), (#, #), (p0, 0p), (p#, q#), (0q, 1p), (1q, q0), (# q0, #q), (#q#, #)

17.3 Indecidabilité de l'ambiguité

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 example de grammaire ambigue en utilisant les système de post donnés en exemple.
— En déduire que le problème de savoir si une grammaire est ambigue est indecidable .

17.4 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 ?