université
PARIS-SACLAY

Langages Formels 2025—2026
TDs + DM, Frédéric Gruau

Plan

Le cours est divisé en trois périodes de 5, puis 4, puis 3 semaines, avec les trois grands thèmes suivants : 1 - automates et expressions rationnelles ; 2 - grammaires hors contexte et automates à piles ; 3 - analyse syntaxique.

L'enchaînement est une construction logique, il est nécessaire d'assimiler les concepts au fur et à mesure, pour suivre. Ce recueil de TD et le support de cours se trouvent sur ma home page `https://www.lri.fr/~gruau/` Attention, prendre la version L2. Les exos sont accompagnés d'un timing indicatif, pour s'assurer qu'on reste dans les clous du programme, au fur et à mesure de l'avancement du TD. Également, j'utilise un marquage « Huge Bold Font » pour indiquer un découpage en semaines. Mon expérience est que souvent, il y a une disparité importante suivant les groupes, et j'essaie ainsi d'éviter cela.

Corrigé.

Vous trouverez les corrigés de tous les exercices à la fin de ce document. Attention, ils sont d'abord destinés aux TD-wo-men, donc un peu elliptiques. Vous pouvez les utiliser si vous ratez un TD, afin de rester en piste, ou pendant le TD si vous êtes trop sec. Parfois par manque de temps, les corrigés sont sous forme de pdf produits par mes étudiants eux-mêmes, dans le drive public `https://tinyurl.com/correctionTdLf`. C'était pour la version du cours destinée au L3, donc les exos ne sont pas toujours les mêmes ; j'y fais référence pour vous simplifier.

Exercices marqués "optionnels"

Ils sont plus difficiles, conçus pour occuper les étudiants très à l'aise, ou vous permettre de travailler chez vous et approfondir. Ils ne sont pas traités en classe par manque de temps. Vous trouverez aussi un corrigé tout à la fin dans une section séparée.

Examens, DM, Rattrapage.

Seules les notes manuscrites, et les polys de cours et d'exercices sont autorisés aux examens. Chaque TD fait l'objet d'un exercice au partiel ou à l'examen. Vous aurez aussi des questions fléchées "questions de cours" non traitées en TD. Le programme du partiel (resp. de l'examen) porte sur les tds et cours fait avant (resp. après) le partiel. L'examen de rattrapage, en juin, porte sur tout le semestre, il est donc naturellement un peu plus difficile. Je souhaite encourager les étudiants à fournir un travail régulier. Deux semaines avant chacun des trois examens, vous aurez l'annale de l'année dernière avec son corrigé, sur ecampus. Pour 2026 ne soyez pas surpris car l'annale ne corresponds pas tout a fait au cours. En effet, en 2025 ce cours était destiné au L3 et pas au L2. Le DM est relativement facile, mais sur un coefficient de seulement 10 pourcent par rapport au partiel. Contrôle continu = (9* partiel + DM)/10. L'énonce du DM est inclus dans ce recueil. Le DM met le focus sur un objectif pédagogique estimé pertinent. Le DM permet de rattraper un peu les notes de partiels d'un ou deux points. Il n'est pas optionnel, donc avoir la note zéro en revanche, va baisser votre note de partiel d'un ou deux points, ce qui est un peu dommage, vu que ce DM ne vous prendra pas plus que quelques heures. Le DM est manuscrit, cela diminue les risques de pompage. Si vous pompez pour le DM, cela n'est pas un bon calcul, car il y aura un exo dans le même style a l'examen, sur lequel vous n'allez pas briller. Deux semaines après le partiel, vos copies seront distribuées en fin de TD, vous pourrez les regarder, et ensuite, vous les rendez. Si vous ratez ce TD, vous ne pourrez pas les consulter ensuite. Pour l'examen, je ne suis pas à Paris pour organiser un 2eme distribution. Sachez que je compte les points deux fois pour être sûr de l'addition. Les étudiants qui sont (très) profondément convaincu que je me suis trompé peuvent demander un scan, mais soyez sympas, car c'est du boulot en plus.

—semaine 1 —

1 s1 Langage formels

Cette feuille est en deux parties : la première partie est relativement facile : d'abord qq courts exos, puis un exo un peu plus costaud pour se familiariser avec l'étoile. Il faut gazer pour avoir le temps de faire la deuxième partie, laquelle comprend deux exos de à peu près une demi-heure chacun qui introduit le raisonnement mathématique avec la double inclusion.

1.1 Produit de langage 10m

Considérons les deux langages sur l'alphabet $\{a, b\}$ définis par $L_1 = \{a, ba\}$ et $L_2 = \{\epsilon, b, aa\}$. Calculer L_1L_2, L_2L_1, L_1^2, L_2^2. Peut on caractériser le cardinal de L^2 à partir du cardinal de L ?

1.2 Puissance et étoile, premiére approche 10m

Dans l'exo précédent caractériser en francais, L_1^*, L_2^* . NB on pose souvent ce genre de questions en examen. L'objectif est de tester votre capacité à prendre du recul, vous devez donnez une description synthétique, compréhensible par votre petite soeur de ans.

Soit $L_3 = \{a\}$, et $L_4 = \{a, \epsilon\}$. Calculer L_3^n, L_4^n pour $n = 0, 1, 2, 3, 10$, pour finir, calculer L_3^*, L_4^*

1.3 Bonjour messieurs les vides : epsilon et $\emptyset$, 5m

L'ensemble vide, et epsilon sont des célébres peaux de bananes sur lesquelles on glisse facilement ; Pour un langage L arbitraire, que valent $L \cup \emptyset$, $L \cap \emptyset$ et $L \cdot \emptyset$? Existe-t-il un langage L_0 tel que pour tout L, $LL_0 = L_0L = L$?

1.4 Deuxiéme approche de l'Etoile, plus fouillée, 45mm

On considère les langages définis sur l'alphabet $A = \{a, b\}$ par $L_1 = \{a\}^*.\{b\}^*$, $L_2 = (\{a\} \cup \{b\})^*$, $L_3 = \{ab\}^*$, $L_4 = \{a\}^* \cup \{b\}^*$ et $L_5 = (\{ab, a\})^*$.

Les questions 1 et 2 de cet exercice sont étroitement imbriquées, on peut les traiter en même temps pour chacun des langages.

1. Les mots suivants appartiennent-ils à chacun de ces langages : a, b, aa, ab, ba, aab, $abab$?
2. Trouver la forme générale des mots de chacun de ces langages, en utilisant les puissances.
3. Comparez ces langages (égalités, inclusions).

1.5 Démonstration d'égalité entre deux langages formels 50m

Les égalités suivantes sont-elles vraies ? Si oui, le démontrer ; sinon, donner un contre-exemple. Objectif : Étudier deux façon pour démontrer qu'un langage L_1 est inclus dans un autre Langage L_2

1. Dans le premier exo, on va prendre un élément quelconque dans L_1, et on montre qu'il appartient à L_2
2. Dans le deuxième exo, on va montrer que L_2 s'écrit comme L_1, union autre chose.

.

1. $L.(M \cap N) = (L.M) \cap (L.N)$ 30mm . Il s'agit ici de reprendre la démonstration du cours que $L.(M \cup N) = (L.M) \cup (L.N)$, voir que cela marche dans un sens et pas dans l'autre, et se servir de cet échec pour trouver un contre-exemple.
2. $L^* = L^*.L^* = (L^*)^*$ 20mm. Objectif : complètement démystifier l'étoile de Kleene ; la clef pour résoudre est vraiment de prendre conscience que L^* sont les mots formés d'une séquence de mots de L. En effet, le terme "séquence " est parlant
3. Optionnel : $(L^*.M)^* = \{\epsilon\} + (L + M)^*.M$

— semaine 2 —

2 s2 Expression rationnelle et automate

Objectif : se familiariser avec ces concepts nouveaux et riches. On considère plusieurs langages, on demande pour chacun de trouver l'expression rationnelles qui le décrit, et l'automate qui le reconnait. On précisera s'il est déterministe ou pas, et s'il ne l'est pas, on essaiera de la modifier pour obtenir une version déterministe. Corrigés automates du prof dans correctionFredAutomate.pdf

2.1 Langage facile sur {0,1} 40m

les entiers sont codés en binaire. Pour les entiers comme pour les mots habituels, on considère que les caractères sont lus de gauche a droite, donc en commençant par les bits de poids forts.

1. L_1 codes binaires des entiers puissance de 2.
2. L_2 codes binaires des entiers puissance de 4.
3. L_3 codes binaires sans zéro redondant. Par exemple, $\ll 01 \gg$ a un zéro redondant.
4. L_4 codes binaires de nombres pairs.
5. L_5 nombres égaux à 1 modulo 4.

2.2 Langage sur {a,b} plus dur 50m

1. $L_5 = \{w \mid w$ commence par ab et termine par $bb\}$
2. $L_6 = \{w \mid w$ contient trois occurrences successives de la lettre $a\}$
3. $L_7 = \{w \mid w$ ne commence pas par $ba\}$
4. $L_8 = \{w \mid w$ ne termine pas par $bba\}$
5. $L_9 = \{w \mid w$ ne contient pas deux occurrences successives de la lettre $a\}$
6. $L_{10} = \{w \mid w$ mots de longueur paire $\}$
7. L_{11}=$\{$ensemble des mots w de la forme $w = a^n b^n\}$;
8. $L_{12} = \{$ensemble des mots w de la forme $w = a^n b^p\}$;

2.3 ExprRat analyse lexicale 30m

La première étape d'un compilateur est l'analyse lexicale, qui découpe le texte d'un programme en unités lexicales appelées "token". Un token peut être un mot clef, un identifiant, une constante numérique. On utilise des expressions rationnelles pour identifier la nature des différents tokens. On utilisera les deux notations suivantes qui permettent une écriture plus compacte : $e? = e|\epsilon$ qui signifie que e est optionnel, $[0-9]$ signifie un chiffre, un caractère dont le code ascii est compris entre 0 et 9. Ces deux notations appartiennent à un ensemble d'une vingtaine d'autres formant ce qu'on appelle les expressions rationnelles étendues, très pratiques lorsqu'on les utilise en vrai pour faire de l'analyse lexicale. Trouver l'expression rationnelle et l'automate associé pour décrire :

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 à virgule

2.4 0ptionnel

$L_{12} = \{w \mid$ le nombre de a et nombre de b dans w sont pair $\} = \{w \mid |w|_a = 0 | w|_b = 0 \ (mod\ 2)\}$

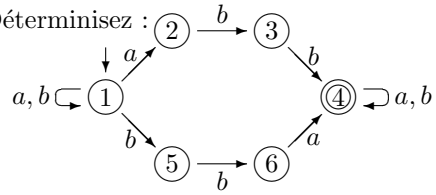
L_{13} = les codes binaires de nombres egaux a 1 modulo 3

— semaine 3—

3 s3 Déterminisation

3.1 L'algo de déterminisation 25m

Déterminisez :



3.2 Boum ! 20m

Intérêt pédagogique :

1. montrez un exemple simple ou effectivement, le nombre d'état du déterminisé, augmente exponentiellement.
2. Prendre conscience que le plus important dans un automate, ce sont les états, et qu'on peut commencer d'abord par en faire la liste et seulement ensuite, et préoccuper des transitions, et savoir qui est final, qui est initial.

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 à quels doivent être les états, puis on rajoutera les transitions.

3.3 Définir via les états 20m

Donner l'automate reconnaissant les mot sur l'alphabe $\Sigma = \{a, b, c\}$ qui contiennent au moins une fois chacune des trois lettres.
Objectif pédagogique : Cet exercice ne fait pas intervenir le non determinisme ; on le met ici car comme pour le précédent exercice, on prendre conscience une 2eme fois que le plus important dans un automate, c'est les états, et que on peut commencer d'abord par en faire la liste et ensuite a se préocupper des transitions, et voir qui est final, qui est initial. Cet ensemble d'état pourrait trés bien être infini. Il le deviendra, par exemple, lorsqu'on adjoindra une pile non bornée.

3.4 Optionnel : Le barman boxeur

Un très bon exercice ludique classique. Il met en jeu des techniques de construction d'automate ; il permet de bien comprendre que le non-déterminisme est effectivement un outil de modélisation.

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 mm̂ême 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 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. Attention, cette méthode ne marche pas si A est non-deterministe ou si A est non-complet.

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

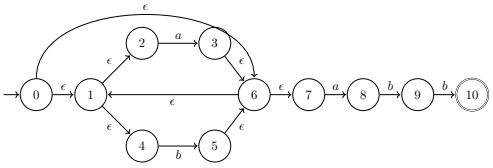
4 Epsilon clotures.

4.1 Un exo simple 10m

On veut construire un automate déterministe qui reconnait les mots sur l'alphabet a,b telle que y a jamais deux lettres identique de suite. On dessine d'abord deux états q_1 et q_2 finaux tels que $q_1 - a - > q_2 - b - > q_1$. Effectivement, on constate que cela évite deux 'a' ou deux 'b' de suite. Rajouter un état initial et deux transitions epsilon, puis éliminer ces transitions epsilon avec la technique des epsilon-clôtures.

4.2 Un exo casse-gueule 20mm

Il enlevez les epsilon transitions de l'automate ci-après. Interet pédagogique : montrez que la construction des epsilon cloture constituent bel et bien une étape intermédiaire dans la déterminisation. N'y restez pas trop longtemps. Cet exo n'est pas anecdotique, mais si on s'y prends mal on peut facilement y passer tout le TD.



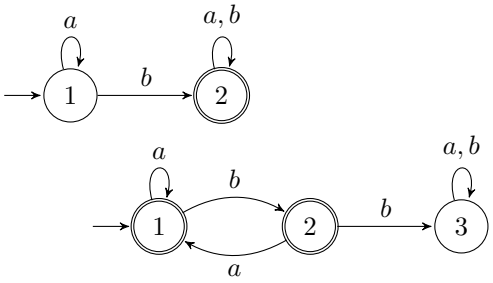
5 Résolution d'équations 25m

Objectif pédagogique : prendre conscience qu'on peut faire des calculs algébriques avec des langages, de la même façon qu'on a l'habitude d'en faire sur les nombres réels. Il faut un peu accumuler un savoir-faire pour s'en sortir à l'examen.

5.1 Exemple 1 a faire en TD

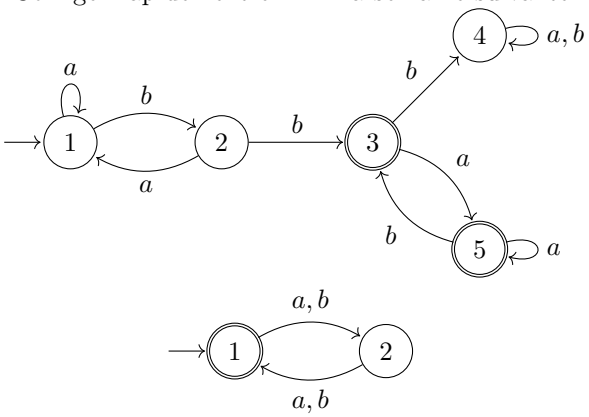
Rappel de cours : à tout automate, on peut associer un système d'équations dont les variables représentent les langages reconnus par cet automate à partir de chacun de ses états.

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 2 et 3 a faire chez soi

Corriger rapidement en TD la semaine suivante.



— semaine 4—

6 s4 Construction d'automates.

6.1 Construction classiques 30m

Un automate est définit principalement par une fonction : la fonction de transition, et quelques ensembles : états, finaux, alphabet. Cet exercice permet de bien s'approprier cet aspect mathématique sur la nature d'un automate d'état fini. Soit un langage régulier L reconnu par l'automate $A = (\Sigma, Q, \delta, q_0, F)$. Construire des automates reconnaissant :

1. $miroir(L) = \{a_n a_{n-1} ... a_2 a_1 | a_1 a_2 ... a_n \in L\}$
2. L'ensemble des mots obtenus à partir des mots de L en effaçant tous les a.
3. le complémentaire de L, en supposant A déterministe.

4. Optionnel : les mots obtenus en effaçant un nombre pair de lettres d'un mot de L

6.2 Produit synchronisé 15m

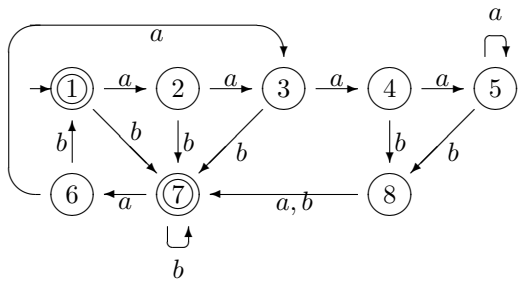
L_1 et L_2 sont reconnus par les automates A_1 et A_2, supposés déterministes complets.

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

7 Minimisation 35m

7.1 Algo de minimisation 25m.

Objectif : maîtriser l'algorithme de minimisation. Reliser l'algo de minimisation dans le cours. Puis minimisez l'automates suivant :



7.2 Egalité entre automates 10m.

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

δ	0	1	2	3
a	1	2	1	3
b	3	1	3	3

état terminal : 1

δ	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 Clôture langages réguliers. 20m

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 :

1. $L1 = \{w \in (a+b)^* \,|\, |w|_a = |w|_b\}$
2. $L2 = \{a^n b^p \,|\, n \neq p\}$
3. $L3 = \{a^{2n} b^{2n} \,|\, n \geq 0\}$
4. $L4 = \{a^n b^p | n \geq p\}$ Vous pourrez établir une relation entre $\{a^n b^p | n \geq p\}$ et $\{a^n b^p | n > p\}$.

— semaine 5—

9 s5 Lemme pompe 90m

Rappel de cours : La version simple du lemme de la pompe est la suivante : Si L est reconnaissable, alors il existe N tel que si $u \in L$ et $|u| \geq N$, alors il existe x, y, z tel que $u = xyz$ et $0 < |y| \leq N$ et $xy^*z \subset L$. En clair, des que le mot est assez long, je peux pomper quelquechose plus petit que N.

Version plus fine, du cours, consiste à dire que l'on peut pomper dans le début du mot, on remplace la condition $0 < |y| \leq N$ par $0 < |y|$ et $|xy| \leq N$.

Non pompable (et donc non reconnaisable) : pour tout N, il existe un mot u dans le langage, de longeueur $\geq N$ tq pour toute decomposition $u = xyz$ avec $0 < |y|$ et $|xy| \leq N$, on a un k tel que $xy^k z \notin L$. On obtient ca en appliquant les regles des negations, c'est vu en cours. Il existe une version encore plus fine : je peux pomper dans tout facteur de longueur au moins N (pas seulement au début)

9.1 Non pompabilité

Objectif pédagogique : savoir démontrer des propriétés utilisant des quantificateurs. Montrez, en utilisant le lemme de la pompe que les langages suivants ne sont pas reconnaissables

1. $\{a^n b^{2n} | n \geq 0\}$
2. $\{(ab)^n c^n | n \geq 0\}$
3. $\{a^n b^m \mid n \geq m \geq 0\}$
4. $\{a^n b^m \mid m \geq n \geq 0\}$
5. $\{a^n b^n | n \geq 0\} + \{a^p b^q | p \neq q[7]\}$
6. optionnel $\{a^n b^m | n \neq m\}$

9.2 Pompabilité-optionnel

Pour une fois, il est possible qu'il y ait le temps de le faire, celui-là. Montrez que le langage suivant est pompable mais pas reconnaissable : $\{b^m a^n b^n \mid m > 0,\, n \geq 0\} \cup a(a+b)^*$

10 Nettoyer-Décider 30m

1. Donnez un exemple d'automate pas propre
2. Pourquoi la définition d'un automate n'impose pas la propreté ?
3. À quoi ressemble le nettoyé d'un automate tel que pour toute lettre x,$\delta(q_0, x) = \emptyset$
4. Est-il décidable de savoir si le langage d'un automate A est fini ?

5. Est il décidable de savoir si deux automates reconnaissent le même langage ?
6. Que se passe-t-il si on minimise un automate pas propre ?

— semaine 6—

11 s6 Grammaire hors context

11.1 De grammaire vers langage 50m

Objectif : visiter le zoo des grammaires classiques, représentatives de ce qu'on peut faire.
Déterminer les langages engendrés par les grammaires suivantes. Dire si la grammaire est ambiguë, justifier, Et si oui, essayer de proposer une version non non-ambiguë.

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. $S \to SS \mid \epsilon \mid (S) \mid [S]$
6. $S \to \epsilon \quad S \to a_i S a_i$ pour tout i, $1 \leq i \leq n$ ou les lettres a_i représentent n terminaux, i.e. $\Sigma_T = \{a_1, \dots a_n\}$
7. $S \to bS \mid aT$; $T \to aT \mid bU$; $U \to aV \mid bS$; $V \to aT \mid bU \mid \epsilon$

11.2 De langage vers grammaire 40m

Trouver des grammaires qui génèrent les langages suivants.

1. L∪ M, L.M L*, ou L et M sont deux langages reconnu par des grammaire d'axiome respectivement X et Y
2. $\{a^n b^p \mid 0 < p < n\}$
3. $\{a^n b^n c^m d^m \mid n, m \in \mathcal{N}\}$
4. $\{a^n b^m c^{n+m} \mid n, m \in \mathcal{N}\}$
5. $\{a^n b^m c^p \mid n = m \text{ ou } m = p\}$
6. optionnel $\{a^n b^m c^p d^q \mid n + q = m + p\}$
7. optionnel $\{a^n b^m c^p d^q \mid n + p = m + q\}$

11.3 Désambiguïser à la main 30m

Objectif : Se familiariser avec la grammaire qui génère les expressions arithmétiques pour les langages de programmations, comprendre comment on peut désambiguïser en introduisant des nouveaux non terminaux.

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 deux arbres de dérivations du mot $id - id - id$. Combien y en a-t-il ? Correspondent-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.

—semaine 7—

12 s7 Vrai langage 90m

Faut avoir parlé d'analyse lexicale en cours. Objectif pédagogique : Se préparer au cours de compilation en L3. 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 ici le début :

program <0>	calcul <-1,50>	; <11>	var <1>	T <-1,51>	: <12>
array <2>	[<13>	1 <-2,1>	.. <14>	10 <-2,10>	

Chaque token est donné par une classe et sa valeur, s'il y en a une. Pour les identificateurs, la valeur sera la chaine de caractères, ou mieux, l'adresse d'entrée dans un tableau de chaine de caractères, qu'on appelle table des symboles en compilation.

Lorqu'elle rencontre des identificateurs, l'analyse lexicale les range dans la table des symboles. Après l'analyse lexicale, celle-ci sera :

adresse	chaîne
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
⋮	⋮

La table est découpée en une zone pour les mots-clés, occupée ici de 0 à 9 et une zone pour les identificateurs à partir de 50. On suppose les différentes entités rangées dans l'ordre de leur apparition, sauf les mots-clés qui sont chargés 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.

On souhaite écrire une grammaire permettant de générer des programmes Pascal, et en particulier notre programme. Plus précisément, la grammaire doit générer non pas le texte du programme mais le "mot" représentant la suite de tokens de ce programme. On commence par quelques questions pour déjà mieux comprendre ce que c'est que ce "mot".

1. À quoi correspond la classe d'un token pour cette grammaire ?
2. À quoi correspond la classe -1, sur cet exemple ?
3. À quoi correspond la classe -2, sur cet exemple ?
4. Que représente la valeur d'un token constante entière, à quelle étape on la calcule, et comment la calculer ?
5. Proposer une grammaire permettant d'engendrer le langage auquel ce programme appartient. On conviendra qu'un non-terminal commence par une majuscule et que les terminaux qui sont aussi des chaînes de caractères commencent par une minuscule.
6.
7. Donner l'arbre de dérivation syntaxique associé à ce programme. Il couvre plusieurs pages, on pourra le finir chez soi. Indiquez les valeurs des tokens.
8. Lorsqu'on compile, on construit une version résumée de l'arbre d'analyse appelée "Arbre de Syntaxe Abstraite" (AST). Elle contient juste les informations utiles. Proposer un AST pour ce programme.

13 Nettoyage grammaire 30m

Algorithme nettoyage On veut nettoyer une grammaire, en enlevant :

(1) Les non-terminaux non-productifs, qui ne produisent pas de mots sur A^*

(2) les non-terminaux non atteignables, qui ne figurent dans aucune dérivation faite à partir de S.

Donner un algorithme permettant de repérer (et donc d'éliminer) les non-productifs, puis un autre algorithme permettant de repérer (et donc d'éliminer) les non atteignables

L'ordre dans lequel on effectue ces deux opérations est-il indifférent ? Pourquoi ?

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

14 DM notez que c'est noté !

Ce DM reprends le TD sur la grammaire du language Pascal. Ce TD va trop vite pour que tout le monde comprenne. Le DM vous permettra de 1- dissocier le travail fait par l'analyse lexicale 2- apprendre à écrire des grammaires "grandeur nature" 3- Aborder la notion de syntaxe abstraite. Egalement, il vous prépare a l'examen ou vous aurai la grammaire d'un mini langage à écrire. Le rendu est à faire SUR PAPIER, et a remettre au TD-wo-men de la semaine du 9 mars.

On considére le programme PASCAL suivant :

```
program factoriel ;
var
  i,n : integer ;
  f : longint ;
begin
  write(' Donner un entier : ') ;
  readln(n) ; f:=1 ;
  for i:=2 to n do
    f:= f * i ;
  writeln('Le  factoriel est ',  f);
End.
```

14.1 Analyse lexicale : Classe, valeur

L'analyseur lexical découpe ce programme en token (comprenant toujours une classe et parfois une valeur) et range les identificateurs dans une table des symboles.

1. Que code le numéro de classe d'un token ?
2. Les identificateurs sont ils tous de la même classe ?

3. Les mots clefs sont ils tous de la même classe ?
4. Quelle valeur a un token identificateur ?
5. Quelle valeur a un token mot clef ?
6. Comment peut on reconnaître les mots clefs de facon simple, lors de l'analyse lexicale ? (On suppose que les mot clefs sont préchargés dans cette table.)
7. Décrivez l'état de la table des symboles, aprés l'analyse lexicale.
8. Ecrire la suite de tokens générée par ce programme, avec les classes et les valeurs des tokens. pour le bout suivant

```
for i:=2 to n do   f:= f * i ;
```

14.2 Grammaire, Analyse syntaxique, syntaxe abstraite.

1. Construction de la grammaire : Proposer une grammaire permettant d'engendrer un langage auquel ce programme appartient. On convient qu'un non-terminal commence par une majuscule et que les terminaux (correspondant aux classe de tokens) commencent par une minuscule
2. Analyse syntaxique : Dessiner l'arbre de dérivation syntaxique généré pour l'analyse de ce programme. Comme il est très grand, utiliser plusieurs pages. On indiquera aussi la valeur des tokens, lorqu'il y en a une.
3. Syntaxe abstraite : Proposer un AST résumant les informations contenue dans le programme.

15 Optionnel grammaire

15.1 Désambiguïsation 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 ambiguë. 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.

— semaine 8—

16 s8 Automates à pile 2H

Construire un automate à pile, si possible déterministe, pour les langages suivants. On précisera à chaque fois son mode de reconnaissance (par état final, par pile vide, ou par les deux).

1. $L_1 = \{a^n b^n | n \geq 1\}$, puis $L_2 = \{a^n b^n | n \geq 0\}$.
2. $L_3 = \{a^p b^n c^q | p \geq 0, q \geq 1, n = p + q\}$. Etudier le cas $q = 0$
3. L_4 = mots de Dyck sur 1 puis sur 2 types de parenthésés.
4. L_5 = mots avec des a et des b qui ont autant de a que de b. On donnera deux solutions : la première n'utilise qu'un état, la seconde qu'un seul symbole de pile.
5. L_6 = les palindromes.
6. $L_7 = \{a^n b^n c^n | n \geq 0\}$
7. $L_8 = \{a^n b^m a^n b^m | n, m \geq 0\}$.

— semaine 9—

17 s9 Est-il algébrique ? 2H

Les langages suivants sont-ils algébriques ? On utilisera les quatre méthodes possibles pour répondre à cette question : On montre qu'un langage est algébrique, en 1- le générant par une grammaire hors contexte ou 2-le reconnaissant par un automate à pile. On montre par l'absurde qu'un langage n'est pas algébrique avec 3- les propriétés de clôtures 4- La contraposée du lemme de la pompe algébrique. Les exercices optionnels utilisent le fait que l'image ou l'image inverse d'un langage algébrique, par un morphisme, reste algébrique.

1. $\{a^n b^m \mid m \neq n \text{ et } m \neq 2n\}$
2. $L_2 = \{a^n b^m a^n b^m \mid n, m \in \mathcal{N}\}$
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. optionnel $\{u \mid \ |u|_a + 3|u|_b = 2|u|_c\}$
8. optionnel $\{u \mid \ |u|_a = 3|u|_b = 2|u|_c\}$
9. optionnel $\{a^n b^n a^n b^n | n \geq 0\}$
10. optionnel $\{a^p b^q c^r | p \leq q \leq r\}$
11. optionnel $\{f(y) \mid y \in Y\}$ 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)
12. optionnel $\{a^p b^q c^r d^s e^t f^u | (p, q, r, s, t, u)$ croit ou décroit $\}$

— semaine 10—

18 s10 Analyse Descendante.

18.1 Calcul premiers suivants 20mm

Soit la grammaire :

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

1. Qu'est-ce que le membre droit (resp. gauche) d'une règle ? Qu'est-ce qui les distingue ?
2. Pour chaque non-terminal X, calculer premier(X). Commencer par écrire les équations et, pour cela, regarder les règles où X apparaît dans le membre GAUCHE.
3. Pour chaque non-terminal X, calculer suivant(X). Commencer par écrire les équations et, pour cela, regarder les règles où X apparaît dans le membre DROIT.

18.2 Table LL(1), build & use, 40m

Grammaire cours $G3 = S- > aSb|\epsilon$ **10m**
Commencez par rappeler pourquoi cette grammaire est difficile, construire la table LL(1) et analyser la chaîne suivante :`aabb#`

Expressions bien parenthésées 30m Sur l'ensemble des terminaux :{( )}, $S \rightarrow (S)S \mid \epsilon$
Construire la table LL(1) et analyser la chaîne suivante : `(()())`

18.3 Est elle LL(1) ? 30m

Objectif : mieux cerner les bonnes grammaires de celles qui ne fonctionnent pas pour l'analyse descendante. Soient les grammaires suivantes :

```
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, dire si elle n'est pas LL(1).

18.4 Recap grammaire charnue 30m

Objectif : Maniper une grammaire dont on ne perçoit pas d'emblée ce qu'elle fait.S'assurer ainsi que l'on maîtrise bien les trois étapes de l'analyse descendante (construction de la table, est-elle LL(1), analyse d'un mot) que nous avons déjà expérimentées, mais pour l'instant seulement sur des grammaires simples en termes de nombre de règles et non-terminaux.

Soit la grammaire G suivante :

$$
\begin{aligned}
S &\rightarrow XaY \\
X &\rightarrow W|T \\
W &\rightarrow bc \\
T &\rightarrow ac \\
Y &\rightarrow eY|f|\varepsilon
\end{aligned}
$$

Voici ses tables Premier et Suivant, le non-terminal Y peut engendrer le mot vide :

	Premier	Suivant
S	a,b	#
X	a,b	a
W	b	a
T	a	a
Y	e,f	#

1. Construire la table d'analyse de G avec l'algorithme vu en cours. Est elle $LL(1)$?
2. Analyser les mots `acaebe` et `acaeee` avec l'algorithme vu en cours.

— semaine 11 —

19 s11 Analyse Ascendante

19.1 Analyse "à la main" 40m

Faire l'analyse à la main permet de rendre plus digeste, les notions du cours abstraites. Soit la grammaire suivante :

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

1. Que reconnait-elle ? est elle ambiguë ?
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 $X \rightarrow \alpha$, une transition appelée "réduction" qui ne lit pas le mot (epsilon transition), qui dépile α et empile X. 2- pour chaque terminal a une transition appelée "lecture" ou "shift" (traduc anglais) qui lit a et empile a.
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 mot.
6. Cet automate n'est pas déterministe, préciser pourquoi

7. Est-ce que c'est gênant ?
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 à 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 ?

19.2 Exemple simple SLR(1) 1H20

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. Pour chaque grammaire, si elle est LR(0), si nécessaire, construire l'automate d'item et identifier les conflits, puis essayer de les résoudre en utilisant l'automate SLR(1).
2. 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.
3. Quelle remarque peut-on faire sur la taille de pile ?
4. Pour les deux premières grammaires, montrez que le langage reconnu par l'automate LR(0) est le langage des mots de pile.

— semaine 12 —

19.3 s12 Analyse syntaxique des expressions arithmétiques 30m

Soit la grammaire des expressions arithmétiques déjà vue en cours :

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

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

19.4 Autre exemple charnu 40m

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

Objectif : voir un cas ou un peu plus compliqué ou on ne vois pas immédiatement c'est quoi le langage considéré.
Construisez l'automate LR(0). 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.

19.5 Les cas LR(1) et LALR(1) 50m

Soit la grammaire suivante :

S	:=	X#	X	:=	`dc`
X	:=	Ma	X	:=	`bda`
X	:=	bMc	M	:=	`d`

Cette grammaire est elle LR(O), SLR(1), LALR(1) ?

19.6 Optionel, cause echec SLR(1) ?

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 l'axiome :

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 ?

20 Corrigés exo optionnels

20.1 égalité entre deux langages

$(L^*.M)^* = \{\epsilon\} + (L+M)^*.M$ On utilise donc la double inclusion comme cela a déjà était fait en cours et en TD.

Dans le sens $(L^*.M)^* \subseteq \{\epsilon\} + (L+M)^*.M$

Si $w = \epsilon$, trivial. Si $w \neq \epsilon$, alors $w \in (L^*.M)^n$ $(n > 0)$, donc il peut s'écrire comme $k_1 \ldots . k_n$ avec $k_i \in (L^*.M)$. On écrit chaque k_i comme $l_i.m_i$ avec $l_i \in L^*$ and $m_i \in M$. On a $l_i \in L^* \subseteq (L+M)^*$ et $m_i \in (L+M)^*$, donc $u = l_1.m_1 \ldots . l_{n-1}.m_{n-1}.l_n \subseteq (L+M)^* \ldots . (L+M)^* \subseteq (L+M)^*$ et donc $w = u.m_n \in (L+M)^*.M$.

Dans l'autre sens $\{\epsilon\} + (L+M)^*.M \subseteq (L^*.M)^*$

Si $w = \epsilon$, alors $w \in (L^*.M)^*$. Si $w \neq \epsilon$, $w = v.m$ avec $v \in (L+M)^*$ et $m \in M$. On peut écrire v comme $k_1 \ldots . k_n$ avec $k_i \in (L+M)$. On démontre par récurrence sur n que $v.m \in (L^*.M)^*$. Si $n = 0$, trivial. Suppose vrai jusque $n-1$. Si pour tout h, v_h est dans L, alors trivial, sinon soit g le plus petit indice tel que v_g est dans M. On applique l'hypothèse de récurrence à $k_{g+1}...k_n.m$ et on dit que $k_1...k_g \in L^*M$

20.2 Expression rationnelle

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

20.3 Automate reconnaissant un langage donné.

— Nombre pair de facteur u :faire deux occurences de l'automate cherchant le facteur u. Dans la premiere version, j'ai lu le facteur un nombre pair de fois (sauf à l'extremité de l'automate ou je viens de lire une occurence de plus), dans la deuxieme impair. Quand j'ai reussi a lire un u de plus, la parite change, on se dirige donc vers l'autre automate. Donc pour chacune de deux versions, remplacer la flèche qui conduit à l'état le plus à droite, par une flèche qui passe sur l'autre version, dans l'état ou on a deja lu v, v etant le plus long préfixe de u qui est aussi suffixe de u, avec v different de u.
Pour $u = bab$, ca fait six etats (1, a, 1) (1, b, 2) (2, b, 2) (2, a, 3) (3, a, 1) (3, b, 2') (1', a, 1') (1', b, 2') (2', b, 2') (2', a, 3') (3', a, 1') (3', b, 2). Initial 1, finals 1 2 et 3

— multiples de trois : difficile, surtout l'expression. il faut commencer par l'automate. Il y a 3 états numérotés 0,1,2. On est dans l'état i quand a lu un nombre qui vaut i modulo 3. Pas besoin de faire un cas spécial pour epsilon, son reste modulo 3 est 0. Si je suis dans l'etat i mod 3 et que la prochaine lettre est j, je vais en $(2i+j)[3]$ (0, 0, 0), (0, 1, 1), (1, 1, 0), (1, 0, 2), (2, 0, 1), (2, 1, 2). L'état final est 1. L'état initial est 0. Pour l'expression, On utlise Arden : $(0 + 1(01^*0)^*1)^*$

20.4 Resolution d'équations

L'exemple optionel correspond à $\{w \in \mathcal{A}^* \mid w$ contient exactement une occurrence de la chaîne $bb\}$. Son expression rationnelle est : $a^*b(a^+b)^*b(a^*b)^*a^*$.
Ou encore $(a^* + a^*ba)^*bb(aba^* + a)^*$.
Ou encore $(ba + a)^*bb(ab + a)^*$.

20.5 Construction d'automates

Automate reconnaissant l'ensemble des mots obtenus en effaçant un nombre pair de lettres d'un mot de L : Correction : On fait le produit synchronisé avec l'automate reconnaissant un nombre pair de lettres, qui a seulement deux états 0, et 1, ou l'état 0 reconnait les mots de longeur paire. On enléve toutes les lettres des transitions ; on rajoute les transition (q,0) $-a->$ (q',0)) et (q,1) $-a->$ (q',1)) si q, $-a->$ q' etait une transition de l'automate reconnaissant L. On obtient un automate non déterministe, on peut choisir à tout instant d' aller se balader dans les état (*,1) quand on reviendra dans les (*,0), on aura effacé deux lettres. Les états final sont donc (q,O) ou q est final.

20.6 Le barman aveugle

question 1 : Il y a 4 configurations possibles :
— T : les verres Tous retournés dans le meme sens
— U : Un verre retourné dans un sens, et les 3 autres dans l'autre
— D : 2 verres en Diagonale dans un sens et les 2 autres dans l'autre
— C : 2 verres a Coté l'un de l'autre dans un sens et les 2 autres dans l'autre

3 coups possibles (on laisse "tout retourner" et "rien retourner" qui ne font rien)
— u : retourner un verre (revient au meme qu'en retourner trois)
— d : retourner deux verres en diagonale
— c : retourner deux verres cote à cote

Etats initiaux : tous (au choix du client) Les transitions :

	u	d	c
T	U	D	C
U	T,D,C	U	U
D	U	T	C
C	U	C	T,D

question 2 : Il suffit de jouer "cd". 'c' oblige le barman à passer dans D, puis 'd', fait gagner le client.

question 3 : Le client gagne si il fait se balader dans l'automate ci-dessus sans passer par l'état T. Donc l'automate en question est celui-ci dessus dans lequel on enleve l'etat T, et où tous les états sont finaux

question 4 : Il faut reconnaître le complémentaire. Donc on déterminise l'automate ci-dessus. Attention : l'état initial sera la réunion des trois états U,C,D. On voit que l'on peut progressivement réduire l'incertitude, c'est a-dire que nos états vont être des ensembles d'états de plus en plus petits. Le non-deterministe avait tous ses états finaux, le déterminisé a donc aussi tous ses états finaux. On rajoute la poubelle, seul états non-final. Puis on intervertit les finals et non-finals, pour trouver le complémentaire. La poubelle se retrouve donc seul état final. On trouve une séquence gagnante (un mot reconnu) qui est dcdudcd. Youpi !

20.7 Minimisation

Calcul des classe d'équivalence de la demi congruences pour le langage $\{uu|u \in A^*\}$. Toutes les classes sont des singletons. Si $u \neq v$, alors il y a un mot w tq uw est carre ouexclusif vw carre : si $|u| = |v|$, prendre u, si $|u|$ pair et $|v|$ impair, idem. Si $|u| > |v|$ meme parite, considerez $\{x|\,|x| = |u|, ux$ carre$\}$ et $\{x|\,|x| = |u|, vx$ carre$\}$. Le premier est un singleton, le second contient $2^{|u|-|v|}$ mots. donc y a un w dans le second qu'est pas dans

20.8 Le lemme de la pompe

Non Pompabilité Le langage $\{a^n b^m | n \neq m\}$ n'est pas pompable. soit N quelconque, je choisi $u = a^N b^{N+N!}$,soit une décomposition quelonque $u = xyz$, $|y| > 0$, $|xy| <= N$, on a y tombe dans les a, $y = a^k$,et $a^{N-k}(a^k)^{(1+N!/k)}b^{N+N!}$ sort du langage, par ce que quand on regroupe les a, on se rends compte que l'exposant vaut celui de b.

Pompabilité Le langage suivant est pompable mais pas reconnaissable : $\{b^m a^n b^n \mid m > 0,\ n \geq 0\} \cup a(a+b)^*$. Soit $N = 2$, soit u un mot quelconque de longeuer au moins 2, s'il est dans $a(a+b)^*$, je peux pomper la deuxieme lettre, s'il est en $b^{\geq 2}a^n b^n$, je peux pomper le premier b, je reste dans $\{b^m a^n b^n\}$, s'il est en $ba^n b^n$, je peux pomper aussi le premier b, puissance 0, je suis dans $a(a+b)^*$, puissance 2 et plus, je suis dans $\{b^m a^n b^n\}$. S'il était reconnaissable, alors son intersection avec le reconnaissable $b(a+b)^*$ le serait aussi, mais c'est $\{ba^N b^N\}$ qui lui n'est pas pompable

20.9 Grammaires hors contexte

Grammaires pour les langages suivants. $\{a^n b^m c^p d^q \mid n+q = m+p\}$

Correction : $L7 = \{(a^n b^n) b^m (c^p d^p) d^m\} + \{a^n (a^m b^m) c^n (c^p d^p) | m > 0\}$ (on décompose suivant plus de a que b ou contraire) d'ou grammaire non ambiguë

$L8 = \{a^n b^m c^p d^q \mid n+p = m+q\} : S -> aSd|T|U;$

Correction : $T -> ...a^x b^{x+y} c^y; U -> b^x c^{x+y} d^y, y > 0,$ (ambigu si je met pas le $y > 0$, cas ou autant de a que de d) Donc pour $L8$ on décompose suivant plus de a que de b. (faire un dessin ou monte de 1 avec un a ou c, descent de avec b ou d) $L8 = a^n (a^p b^p)(c^m d^m) d^n + (a^n b^n)(b^m c^m)(c^p d^p)$ ambigu, on enlève l'ambiguïté en forçant $m > 0$ dans le second par exemple.

20.10 Désambiguation difficile.

Correction : La solution suivante est simple mais ambigue. $D1 -> aD1b|bD1a|D1D1|epsilon$ ambigu $D1 -> aD1bD1|bD1aD1|epsilon$ ambigu, cf abab. Faire des graphes, monte de 1 sur un a, descend sur un b. $D1$ = part de 0, finit en 0, $D2$ = part de 0, finit en 0, reste au dessus de 0, $D3$ = part de 0, finit en 0, reste au dessous de 0. $D1 = (D2+D3)^*$ On décompose un mot de $D1$ en une suite de mots, non nulls, en coupant a chaque fois que ca croise 0. On génère de facon non ambiguë un mot de $(D2+D3)^*$ Un mot de $D2$ non vide s'écrit $aD2b$, ou $D2$ est l'axiome du langage $D2$. On a aussi que un mot de $D2$ non vide se décompose comme un mot de $D2$ non vide, suivit par un mot de $D2$. un mot de D3 est soit un mot vide, soit un mot non vide : $D1 -> aD2bD1|bD3aD1|epsilon; D2 -> aD2bD2|epsilon; D3 -> bD3aD3|epsilon$

20.11 Grammaires contextuelles

Cette grammaire a été vue en cours, ca génére $a^n b^n b^n$.

20.12 Est il algébrique difficile

Le complémentaire de $\{uu \mid u \in A^*\}$ est algébrique. En effet, m n'est pas un carre ssi il est de longueur impaire, ou il s'ecrit uv avec u et v de meme longueur mais differents. On se ramene facilement aux uv. Dire que u et v ont meme longueur mais sont differents signifie qu'il existe deux lettres $a <> b$ et des mots x, y, w, t tq $|x| = |y|, |w| = |t|$ $u = xaw$ et $v = ybt$, ce qui revient a dire que m peut s'ecrire $xawybt$.

La, grosse ruse, un mot s'ecrit wy avec $|x| = |y|, |w| = |t|$ ssi il est de longeur $|t|+|y|$ ssi il est de longeur $|y|+|t|$ ssi il s'ecrit $y'w'$ avec $|x| = |y'|, |w'| = |t|$

ce qui fait qu'un mot pair n'est pas un carre ssi il s'ecrit $xay'w'bt$ avec a et b differents, x et y de meme longueur, w et t de meme longueur, ce qui revient a dire qu'il est genere pas la grammaire

$S_a -> \alpha S_a \beta$ pour toutes lettres a, α β

$S_a -> a$ pour toute lettre a

$S -> S_a S_b$ pour toutes lettres a et b avec $a \neq b$

Bien sur, la grammaire est ambigu, il y a autant de facon d'avoir uv avec $|u| = |v|$ qu'il y a de i tq la ieme lettre de u est differente de la ieme de v.

Le langage $\{u \mid |u|_a + 3|u|_b = 2|u|_c\}$ est Algebrique , faire automate pile. Morale : sur a, j'empile U sur b j'empile UUU sur C je dépile UU

Le langage $\{u \mid |u|_a = 3|u|_b = 2|u|_c\}$ n'est pas algèbrique Intersection avec $a*b*c*$ (ca donne $a^{6n} b^{2n} c^{3n}$) puis image inverse via le morphisme f défini par $f(a) = aaaaaa, f(b) = bb, f(c) = ccc$.

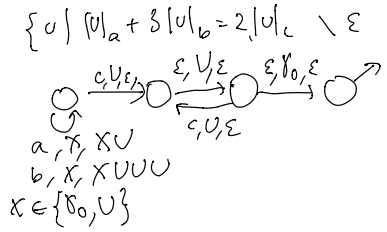


FIGURE 1 – Corrigé 7

Le langage $\{a^n b^n a^n b^n | n \geq 0\}$ n'est pas algébrique , on se ramene a un connu. Malgre son air de anbmanbm, c'est a anbncn qu'on se ramene. image inverse de $h(a) = h(c) = a$, $h(b) = h(d) = b$, puis intersection avec $a^* b^* c^* d^*$, puis morphisme efface d

Le lagnage $\{a^p b^q c^r | p \leq q \leq r\}$ n'est pas algébrique , pompage. prendre $a^N b^N c^N$. Etude des decompositions. On pompe en paralléle x et y.

1er cas : x ou y est "a cheval" sur deux paquets. $k = 2$ et on n'est plus dans $a^* b^* c^*$.

2eme cas : x ou y est nul, si le non nul est dans les a ou les b, faire $k = 2$, dans les c, faire $k = 0$.

3eme cas, x dans les a, y dans les b : $k = 2$

4eme cas, x dans les b, y dans les c : $k = 0$

Le langage $\{f(y) \mid y \in Y\}$ 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) est algèbrique. on considére le morphisme définit par $h(a) = h(\bar{a}) = a, h(b) = h(\bar{b}) = b$ $g(a) = a, g(b) = b, g(\bar{b}) = g(\bar{a}) = epsilon$ $f(X) = g(h^{-1}(X) \cap ((a+b)(\bar{a}+\bar{a}))^*(a+b+epsilon))$.

Le langage $\{a^p b^q c^r d^s e^t f^u | (p,q,r,s,t,u)$ croit ou décroit $\}$ n'est pas algèbrique. Pompage, il faut prendre le mot $a^N b^N c^N d^{N+1} e^{N+1} f^{N+1}$. Je vous laisse faire l'etude des cas. Remarque : si on ne met que 4 lettres, c'est pompable ! Avec 5 lettre, non pompable, mais il faut jouer sur le fait que $|xuy| \leq N$. Prendre $a^N b^{3N} c^{3N+1} d^{3N+1} e^{3N+1}$.

20.13 Analyse pas SLR(1).

Suivants de T et de A.

$$\begin{array}{ll} \text{SUIV}(T) & = \text{SUIV}(A) \cup \{\sharp, *\} \quad = \{\rightarrow, \sharp, *\} \\ \text{SUIV}(A) & = \{\rightarrow\} \end{array}$$

Les états de l'automate LR(0) sont

s_1 : $S \rightarrow .T\sharp$
$T \rightarrow .\texttt{int}$
$T \rightarrow .A \rightarrow \texttt{int}$
$A \rightarrow .T$
$A \rightarrow .T * A$

s_2 : $T \rightarrow \texttt{int}.$

s_3 : $S \rightarrow T.\sharp$
$A \rightarrow T.$
$A \rightarrow T. * A$

s_4 : $T \rightarrow A. \rightarrow \texttt{int}$

s_5 : $A \rightarrow T * .A$
$T \rightarrow .\texttt{int}$
$T \rightarrow .A \rightarrow \texttt{int}$
$A \rightarrow .T$
$A \rightarrow .T * A$

s_6 : $T \rightarrow A \rightarrow .\texttt{int}$

s_7 : $A \rightarrow T.$
$A \rightarrow T. * A$

s_8 : $A \rightarrow T * A.$
$T \rightarrow A. \rightarrow \texttt{int}$

s_9 : $T \rightarrow A \rightarrow \texttt{int}.$

La table de transitions est :

	int	→	*	♯	T	A
s_1	shift s_2				goto s_3	goto s_4
s_2		. . . reduce $T ::= \texttt{int}$ . . .				
s_3		reduce $A ::= T$	shift s_5	succes		
s_4		shift s_6				
s_5	shift s_2				goto s_7	goto s_8
s_6	shift s_9					
s_7		reduce $A ::= T$	shift s_5			
s_8		shift s_6 reduce $A ::= T * A$				
s_9		. . . reduce $T ::= A \rightarrow \texttt{int}$ . . .				

La grammaire n'est pas SLR(1) (état s_8) avec le symbole d'avance $\rightarrow$. Le conflit est un conflit shift/reduce dans une situation de la forme $T * A. \rightarrow \texttt{int}$ on ne sait pas s'il faut réduire pour obtenir $A \rightarrow \texttt{int}$ ou au contraire lire $T * A \rightarrow .\texttt{int}$ ce qui amènera ensuite à réduire $A \rightarrow \texttt{int}$ en T. Le problème se pose dans le cas du type

$$\texttt{int} * \texttt{int} \rightarrow \texttt{int} \rightarrow \texttt{int}$$

qui a deux interprétations possibles : une fonction à un argument fonctionnel de type $\texttt{int} * \texttt{int} \rightarrow \texttt{int}$ ou bien une fonction à deux arguments l'un de type `int` et l'autre de type $\texttt{int} \rightarrow \texttt{int}$. Cet exemple montre que la grammaire est ambiguë.

NB avec le type

$$\texttt{int} * \texttt{int} \rightarrow \texttt{int}$$

on arrive aussi à l'état ou il y a conflit. Il n'y a pas de conventions qui s'impose, le mieux est probablement de forcer un parenthésage du type des arguments lorsqu'ils sont fonctionnels.

Corrigés

Corrigé. Il faut commencer par calculer L_2^2 avant de calculer L_2^3, on remarque que $|L_2^2| \neq |L_2|^2$, parce que certain mot de L_2^2 se décompose de deux façon différentes, comme un produit de deux mot de L_2

$$\begin{aligned} L_1L_2 &= \{a, ba, ab, bab, aaa, baaa\} \\ L_2L_1 &= \{a, ba, bba, aaa, aaba\} \\ L_1^2 &= \{aa, aba, baa, baba\} \\ L_2^2 &= \{\epsilon, b, aa, bb, baa, aab, aaaa\} \end{aligned}$$

Corrigé.L_1^* est l'ensemble des mots sur $\{a, b\}$ tels qu'un b est toujours suivi d'un a, L_2^* est l'ensemble des mots sur $\{a, b\}$ tels que les a apparaissent toujours par deux.

Corrigé. Cet exercice a pour but de maîtriser la puissance des langages, l'épsilon et une deuxième confrontation à l'étoile de Kleene.

Corrigé. La concaténation peut s'écrire par simple juxtaposition L_1L_2 ou bien en utilisant la notation $L_1 \cdot L_2$. Expliquez bien comment la définition du produit de langage permet de démontrer que $L \cdot \emptyset = \emptyset$. En effet, lorsqu'on cherche en element dans $\emptyset$ pour le marier avec un autre élément de L, on n'en trouve tous simplement pas, puisque il est vide, donc on génére bien vide. $L \cup \emptyset = L$, $L \cap \emptyset = \emptyset$ et $L \cdot \emptyset = \emptyset$. $L_0 = \{\epsilon\}$ convient pour $L \cdot L_0 = L_0 \cdot L = L$.

Corrigé. La réponse est évidente pour le langage L_2 qui contient tous les mots formés à l'aide des lettres de A.

Le tableau suivant résume les dérivations qui permettent de décider lorsque le mot appartient au langage

	a	b	aa	ab	ba	aab	$abab$
L_1	a^1b^0	a^0b^1	a^2b^0	a^1b^1	Non	Oui	Non
L_3	Non	Non	Non	$(ab)^1$	Non	Non	$(ab)^2$
L_4	a^1	b^1	a^2	Non	Non	Non	Non
L_5	a^1	Non	a^2	$(ab)^1$	Non	$a \cdot ab$	$(ab)^2$

Les résultats négatifs sont les plus difficiles à justifier, on peut passer un peu rapidement la dessus pour avoir le temps de faire les deux exos de la fin. La non-appartenance de ba et $abab$ à L_1 est due au fait que dans un mot du langage L_1 un a ne peut pas apparaitre après un b. Les mots de L_3 sont une alternance stricte de lettre a et de lettre b commencant par un a. Les mots a, b, aa et ba ne répondent pas à cette définition. Enfin, les mots de L_4 ne contiennent que des a ou que des b. Les mots qui contiennent des a et des b n'appartiennent donc pas à L_4.

Corrigé.

- L_1 est l'ensemble des mots a^nb^m, $n, m \geq 0$.
- L_2 est l'ensemble de tous les mots A^*.
- L_3 est l'ensemble des mots $\{ab\}^n$, $n \geq 0$.
- L_4 est l'union des mots a^n, $n \geq 0$ et b^m, $m \geq 0$.
- L_5 est l'ensemble des mots ou chaque b est précédé d'un a.

Corrigé.À faire en mode toute la classe.

- Le langage L_2 est l'ensemble de tous les mots sur l'alphabet $\{a, b\}$. Il contient donc les langages L_1, L_3, L_4 et L_5. Nous avons donc $L_2 \supseteq L_1$, $L_2 \supseteq L_3$, $L_2 \supseteq L_4$, $L_2 \supseteq L_5$.
- L_1 et L_3. Nous avons qu'il existe au moins un mot (aa) qui appartient à L_1 et pas à L_3 et un mot $(abab)$ Qui appartient à L_3 et pas à L_1. La comparaison entre L_3 et L_4 est elle aussi vite faite à l'aide des mots a et ab. En fait les seuls mots communs à ces deux langues sont ε et ab.
- L_1 et L_4. Nous avons le mot ab qui appartient à L_1 mais pas à L_4. Par contre, tous les mots de L_4 sont aussi des mots de L_1 : pour un mot de L_4, il y a deux possibilités. Soit ce mot est de la forme a^n avec $n \geq 0$ et dans ce cas il est aussi dans le langage L_1 sous la forme $a^n.b^0$. Soit ce mot est de la forme b^n avec $n \geq 0$ et dans ce cas il est aussi dans le langage L_1 sous la forme $a^0.b^n$. Nous avons donc $L_4 \subseteq L_1$.
- $L_3 \subseteq L_5$
- L_5 et L_1 ou L_4 ne sont pas comparables : b est dans L_1 et L_4 mais pas dans L_5, aba est dans L_5 mais ni dans L_1 ni dans L_4.

Corrigé.corrigés étudiants sur TD1.png TD1.pdf

1. ATTENTION : Bien leur faire voir le cote "automatique" de ce type de preuve. Montrez aussi que voir ou une preuve echoue aide a construire un contre-exemple.
   Soit $u \in L.(M \cap N)$, on a donc $u = vw$ avec $v \in L$ et $w \in M \cap N$ ie $w \in M$ et $w \in N$. on a donc $vw \in LM$, aussi $vw \in LN$, donc $uvw \in (L.M) \cap (L.N)$. Ce sens fonctionne.
   Soit $u \in (L.M) \cap (L.N)$. On a donc $u \in LM$ et $u \in LN$, ie $u = vw$ avec $v \in L$ et $w \in M$, et aussi $u = v'w'$ avec $v' \in L$ et $w' \in N$. Ah oui mais je suis coince car je n'ai pas forcement $v = v'$. On cherche un contre-exemple sur ce schéma. Exemple de contre-exemple : $L = \{ab, a\}$, $M = \{a\}$, $N = \{ba\}$, $u = aba$ L'intersection des unions n'est pas vide, mais l'union des intersections est vide. Donc l'inclusion qu'on vient de démontrer est bien stricte.
2. On utilisera la triple inclusion :
   1 inclus dans 2 : car $u = u.\epsilon$...
   2 inclus dans 3 : $X^2 \in X^*$ applique a $X = L^*$
   3 inclus dans 1 car une séquence de mots, c'est encore une séquence de mot de L.
   Les élèves sont toujours très surpris qu'une telle réponse suffise "à avoir les points dans un examen". Une courte réponse démontre en général une bonne maîtrise. On peut faire une démonstration plus formelle, mais il faut faire une double indexation, c'est un peu lourd et on n'a pas trop le temps pour cela.

Corrigé.

1. 1.0^*
2. $1.(00)^*$
3. $0|1.(0|1)^*$
4. $(0|1)^*0$ exemple canonique où c'est plus facile en non-déterministe
5. $(0|1)^*01$ Autre exemple canonique où c'est plus facile en non-déterministe

Corrigé.

1. $a.b.(a|b)^*.b.b|a.b.b$
2. $(a|b)^*.a.a.a.(a|b)^*$ autre exemple canonique ou c'est plus facile en non déterministe
3. $\epsilon|a.(a|b)^*|b.(\epsilon|b.(a|b)^*)$

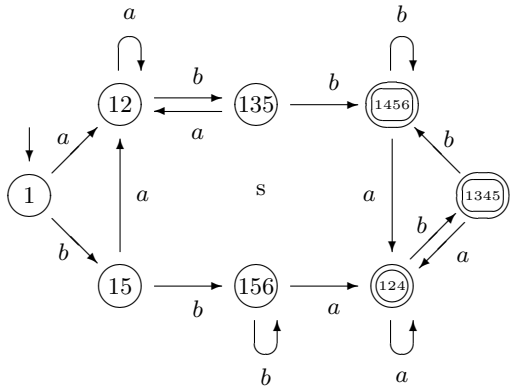
4. $\epsilon|(a|b).(\epsilon|(a|b))|(a|b)^*.(a.(a|b).(a|b)|b.(a.(a|b)|b.b))$
5. $(\epsilon|b|a.b)^*.(\epsilon|a)$
6. $((a|b)^2)^*$
7. pas de solution, car il faudrait compter les ''a' et on ne peut pas compter en état fini
8. a^*b^*

Corrigé.corrigés étudiants automate sur TD2.png TD2.pdf

1. id = lettre.(lettre|chiffre*)
2. chiffre= $0-9$
3. entier = $[1-9][0-9]*|0$
4. entier-relatif = $-?[1-9][0-9]*|0$
5. nombre à virgule = entier-relatif'.' entier

Les étudiants posent souvent la question : est-ce que 0001 est un entier ? fOui, parce que c'est plus simple, et c'est vrai en pratique, par exemple en java.

Corrigé.J'ai déjà fait un exemple en cours. Il faut faire une table avec une colonne pour les ensembles d'états, une colonne pour le résultat des transitions par a, et une autre pour b. Au fur et à mesure, on crée de nouveaux ensembles qu'on doit réintroduire pour calculer leurs transitions. Une fois la table terminée, on peut dessiner le graphe de l'automate. Amorcer le processus en commençant à remplir, les laisser faire un peu, et envoyer au tableau un étudiant qui a des difficultés.



Corrigé.corrigé étudiants sur TD2.png TD2.pdf Il y a n+1 états dans le non-déterministe, et 2^n dans le deterministe Moralité : ca explose. Le non déterministe est facile, (1,a ou b,1), (1,b,2),(2, a ou b 3) (3 a ou b 4). Pour le déterminisé, on remarque que il faut stoquer les trois derniéres lettres lues dans l'état. Ca fait au total 8 états indéxé par ce même triplet. On dessine les états et ensuite on prends conscience que c'est facile de rajouter les transitions. Les état finaux sont ceux qui ont b à la bonne place ; Faut leur faire deviner l'état initial qui est moins facile a trouver : c'est aaa, parceque cela oblige a lire au moins trois lettres.

Corrigé.correction étudiants sur TD2.png TD2.pdf Scénario dramatique : Je leur dis qu'ils sont en prison, avec un petit bout de papier et un crayon qui symbolisent les états finis. Ils vont voir passer un mot avec plusieurs milliers de lettres, et à un moment le roi dit ≪ stop ≫ et leur demande si le mot est dans le langage ou non. Bien sûr, s'ils se trompent, on leur coupe la tête. Que doivent-ils écrire sur leur papier pour ne pas se tromper ? réponse : un 'a' s'il ils on vu passer un 'a', meme chose pour les 'b' et les 'c' . Ensuite , combien d'états possibles peuvent apparaître sur le papier : 8, les sous-ensembles de a,b,c. Ensuite, comment on note les états : par le sous-ensemble associé. État initial ? ensemble vide. Final ? a,b,c. On dessine les états au tableau en mettant sur une même ligne horizontale les singletons, puis les doubletons en dessous. On mets les transitions, et ne pas oublier les boucles.

Corrigé.

Voici les ε-clotures :

$cloture(0) = \{0, 1, 2, 4, 6, 7\}$
$cloture(1) = \{1, 2, 4\}$
$cloture(3) = \{1, 2, 3, 4, 6, 7\}$
$cloture(5) = \{1, 2, 4, 5, 6, 7\}$
$cloture(6) = \{1, 2, 4, 6, 7\}$

et pour les autres états, on a $cloture(i) = \{i\}$

Pour arriver rapidement a une solution, cela demande de réfléchier sérieusement. C'est un peu l'objectif de cet exo de montrez que souvent, si on ne prends pas le temps de reflechir et prendre du recul, on peut se casser les dents bien comme il faut en examen.

D'abord on raisonne donc, juste sur les clotures comme c'est dis dans le cours. Ensuite on remarque que l'image par a d'une cloture c contient cloture(q3) et/ou cloture(q8) : cloture(q3) si c contient q2, et cloture(q8) si c contient q7. De meme l'image par b contient cloture(q5) si c contient q4. On deduit que
cloture(q0)$-a->$cloture(q8), cloture(q3)
cloture(q0)$-b->$cloture(q5)
cloture(q5)$-a->$cloture(q8), cloture(q3)
cloture(q3)$-a->$cloture(q8), cloture(q3)
cloture(q3)$-b->$cloture(q5)
Ensuite on progresse en lisant b de q8=cloture(q8) vers q9=cloture(q9) et enfin q10=cloture(q10) :

L'automate solution n'utilise pas les clôtures de q1, q2, q4, q6 car elles ne sont pas atteignables depuis l'état initial qui est la clôture de q0. Comme c'est dit dans le cours, on obtient un automate où les états sont des epsilon-clôtures, Il n'est toujours pas déterministe, mais au moins on constate bien qu'on s'est bien débarrassé des epsilon transitions. "Les epsilon sont sagement entrés dans les clotures !". Le non determinisme vient des transitions par 'a' qui mène vers deux clôtures distinctes : cloture(q8) et cloture(q3) Il faudrait donc encore déterminiser ? Ben non ; c'est pas demandé dans l'énoncé ! Ceci illustre une autre erreur courante en examen qui consiste à ne pas répondre à la question car on l'a lu trop vite.

Corrigé.correction étudiante sur TD2.png TD2.pdf Un exemple a déja été traité en cours. Rappel des règles :

- $R = \alpha R + \gamma$ a pour solution (unique si $\epsilon \notin \alpha$) $R = \alpha^*\gamma$;
- On ajoute aX s'il y a une transition a vers X
- on ajoute $+\epsilon$ aux états initiaux ;
- Il y a une inconnue par état qui représente le "Futur" de l'etat, le langage reconnus en prenant cet états pour etat initial.
- Le langage reconnu est donc l'inconnue associée a l'état initial.

$$\begin{cases} X_1 = aX_1 + bX_2 \\ X_2 = (a+b)X_2 + \epsilon \end{cases}$$

On a donc $X_2 = (a+b)^*$. D'où, en remplaçant X_2 dans X_1 :
$X_1 = aX_1 + b(a+b)* = a^*b(a+b)*$.

Corrigé. Correspond à $\{w \in \mathcal{A}^* \mid w$ ne contient aucune occurrence de la chaîne bb $\}$ On obtient $a^*(ba^+)^*(\epsilon + b)$. Ou encore $(a+ba)^*(\epsilon+b)$.

Corrigé.correction étudiants sur TD3.png Correspond à $\{w \in \mathcal{A}^* \mid w$ contient exactement une occurrence de la chaîne $bb\}$.

Expression rationnelle par dérivation : $a^*b(a^+b)^*b(a^*b)^*a^*$.
Ou encore $(a^* + a^*ba)^*bb(aba^* + a)^*$.
Ou encore $(ba+a)^*bb(ab+a)^*$.

Corrigé.

Correspond à la version minimisée de $\{w \in \mathcal{A}^* \mid$ le nombre de a et de b ont même parité dans w $\}$.

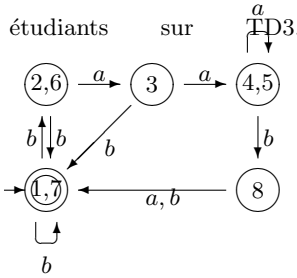
Expression rationnelle de l'automate minimal : $((a+b)^2)^*$.

Corrigé.correction étudiante sur TD3.png On s'attachera à écrire dans chaque cas, quel est le nouveau delta, le nouveau F, le nouveau q0.

1. $\delta_{\text{miroir}}(q,a) = q'/\delta(q',a) = q$
   $F_{\text{miroir}} = \{q_0\}$ $q_{0\text{miroir}}$ = nouvel état, avec epsilon transition vers les états de F
2. $\delta_{\text{SansA}}(q,a) = \emptyset$
   $\delta_{\text{SansA}}(q,\epsilon) = \delta(q,a)$
   si x différent de a, ϵ $\delta_{\text{SansA}}(q,x) = \delta(q,x)$
3. On a juste à modifier l'ensemble F des finaux. Plus précisément, permuter final et non final : $F_{\text{Comp}} = Q \setminus F$

Corrigé. Il faut faire tourner les deux automates en même temps. Il faut construire le produit synchronisé, dont les états sont des couples d'états. l'intersection : ca marche sur non déterministe ou sur non-complet l'union : le non determinisme ne dérange pas mais ça foire si pas complet. Le problème est que si u est reconnu par A1, mais que pas de chemin de q0 vers quelque chose indexé par u dans A2, alors y a pas de chemin dans l'automate produit.

Corrigé.correction étudiants sur TD3.png



Partitions de $\sim_0$: 234568 / 17

par a, 23456 vont dans le 1er ensemble, 8 dans le second, (8 s'isole), 17 va dans le second

par b, 45 va dans le premier ensemble, 2368 dans le second (45 se separe de 2368), 17 reste dans le second

Donc partition $\sim_1$: 236 / 45 / 8 / 17

par a, 23 vont dans le premier ensemble, 6 dans le second (6 s'isole), 45 vont dans le second, 17 dans le premier

par b, 236 vont dans le 4eme, 45 dans le troisieme, 17 dans le 4eme

Donc partition $\sim_2$: 23 / 6 / 45 / 8 / 17

par a, 23 vont dans le premier ensemble 45 vont dans le troisieme, 17 dans le premier

par b, 23 vont dans le 5eme, 45 dans le 4eme, 17 dans le 5eme

Donc $\sim_2 = \sim_3 = \sim$

Corrigé.correction étudiants sur TD4.png On minimise le deuxième, et on trouve trois états que l'on peut mettre en correspondance avec le premier donc c'est le même.

Corrigé.correction étudiants sur TD5.png

1. Si $L1$ était rationnel, $L1 \cap a^*b^*$ le serait aussi, or c'est $\{a^nb^n\}$, absurde
2. Si L_2 était rationnel, alors le complémentaire le serait aussi, le complémentaire c'est a^nb^n plus le complémentaire de a^*b^*. Il ne reste qu'à faire intersection avec a^*b^* pour tomber sur $\{a^nb^n\}$, ce qui est absurde.
3. Si L3 était rationnel, alors $L3 + a.L3.b = \{a^nb^n\}$ le serait aussi, absurde.
4. Si $L4$ était reconnaissable, alors $L' = \{a^nb^p | n > p\}$ le serait aussi car c'est $a.L4$, puis a^nb^n le serait aussi parce que c'est $L' - L4$, ce qui est absurde.

Corrigé.correction étudiante sur TD5.png. Il faut présenter les démonstrations, en utilisant deux couleurs : une pour les « il existe » $\exists$ et une autre pour « quelque soit » $\forall$. Pour demontrer un $\forall x$ vérifiantLa propriété P, il faut considérer un objet arbitraire, vérifiant quand même la propriété P. On utilise pour cela le subjconctif du verbe être : par exemple "soit" N un entier quelconque. En revanche, pour démontrer un $\exists x$ vérifiant la propriété P, il faut explicitement construire l'objet x et montrer que oui, il vérifie P ; j'utilise le verbe "choisir" pour bien marquer le fait que je réfléchis pour construire x. Par exemple, "je choisis" le mot $u = a^Nb^{2N}$, la longueur est $3N$ et on vérifie bien que $3N > N$.

1. $\{a^nb^{2n} | n \geq 0\}$
   s soit N quelconque, je choisis $u = a^Nb^{2N}$ la longueur est $3N > N$ Soit une décomposition quelconque $u = xyz$, $|y| > 0$, $|xy| <= N$, on a y tombe dans les a. Je choisit $k = 2$. On a bien $xyyz \notin L$, car on rajoute des a sans modifier le nombre de b, donc il y aura un déséquilibre. $k = 0$ marche aussi, on enléve que des a
2. $\{(ab)^nc^n | n \geq 0\}$
   Soit N un entier quelconque. Je choisis $u = (ab)^Nc^N$ soit une décomposition quelonque $u = xyz$, $|y| > 0$, $|xy| <= N$, Il n'y a pas de c dans y, je choisis $k = 2$ on a $m' = xyyz \notin L$, car pas assez de 'c'. En effet, on a rajouté ou bien des 'a' ou bien des 'b' mais aucun 'c'
3. $\{a^nb^m \mid n \geq m \geq 0\}$
   Soit N quelconque, Je choisis $u = a^Nb^N$, à la frontière du langage. Il suffit d'enlever un seul 'a' pour sortir du langage. Soit une décomposition quelconque $u = xyz$, $|y| > 0$, $|xy| <= N$, on a y tombe dans les a. Je choisis $k = 0$, car il faut enlever des a pour sortir du langage. $xy^0z = xz \notin L$

4. $\{a^n b^m \mid m \geq n \geq 0\}$
   idem, mais cette fois ci il faut rajouter des a, donc on prends $k = 2$ $xy^2z \notin L$.
5. $\{a^n b^n | n \geq 0\} + \{a^p b^q | p \neq q[7]\}$
   prendre $u = a^N b^N$, comme d'habitude v tombe dans les 'a' et s'écrit donc a^l . On choisit $k = 8$, on vérifie alors que $xy^8z \notin L$ car $|m'|_a = N + 7 * l$ et $|m'|_b = N$ les deux sont donc égaux modulo 7.
6. optionnel $\{a^n b^m | n \neq m\}$
   soit N quelconque, je choisi $u = a^N b^{N+N!}$, soit une décomposition quelconque $u = xyz$, $|y| > 0$, $|xy| <= N$, on a y tombe dans les a, $y = a^k$, et $a^{N-k}(a^k)^{(1+N!/k)}b^{N+N!}$ sort du langage, parce que quand on regroupe les a, on se rend compte que l'exposant vaut celui de b.

Corrigé.

1. Mettre un état final non accessible.
2. En pratique ces individus se rencontrent quand on fais des math, c'est pratique donc, de leur donner le droit donc, d'exister dans la nature.
3. y a juste l'état initial, et pas d'état finaux
4. En exam on vous possera que des probléme décidable, il suffira donc de donner un algo, et en général cet algo comporte une étape de nettoyage : Le langage d'un automate A est infini si et seulement si le nettoyé à des cycles. Donnez un exemple d'automates avec cycle dont le langage est fini, (le cycle étant suprimmé par le nettoyage).
5. oui On nettoie et minimise chaque automate et si ils ont le même nombre d'états, on essaie de mettre en correspondance les états de l'un avec ceux de l'autre. Le problème est donc décidable. Il reste a priori quand même difficile de trouver la correspondance, car il faut essayer toutes les permutations d'états.
6. Les états inutiles ont tous le meme futur qui est l'ensemble vide, il vont donc fusionner tous ensemble vers un seul état inutile, le minimisé ne sera donc pas propre, et pas minimal non plus puisque cet état inutile peut gicler aussi.

Corrigé.correction étudiante sur TD5.png

1. $S \to \epsilon \mid aaaS$
   $(aaa)^*$
2. $S \to ab \mid aSb$
   $\{a^n b^n \mid n \geq 1\}$
3. $S \to XY \mid Z$; $X \to Xa \mid a$; $Y \to aYb \mid \epsilon$; $Z \to aZb \mid W$; $W \to bW \mid b$
   On commence par calculer le langage $L(W)$ généré par W, puis $L(Z)$ etc...Les $a^n b^p$ avec $n \neq p$. Noter que l'on traite de façon non symétriques le débord de a et le débord de b, alors qu'on pourrait uniformiser
4. $S \to SS \mid \epsilon \mid (S)$
   les mots bien parentheses,
5. $S \to SS \mid \epsilon \mid (S) \mid [S]$
   itou mais sur deux types de parentheses
6. $S \to \epsilon \quad S \to a_i S a_i$ pour tout i, $1 \leq i \leq n$ ou les lettres a_i représentent n terminaux, i.e. $\Sigma_T = \{a_1, \dots a_n\}$
   les palindromes de longueur paire
7. $S \to bS \mid aT$; $T \to aT \mid bU$; $U \to aV \mid bS$; $V \to aT \mid bU \mid \epsilon$
   grammaire reguliere, donc automate avec un état pour chaque non-terminal. Il reconnait les mots qui finissent par aba

Ambiguïté : Pour se convaincre qu'une grammaire n'est pas ambigüe, il faut construire un arbre de dérivation pour un mot représentatif, et constater qu'on n'a jamais le choix ; je fais faire ce travail par les étudiants aux tableaux. Pour construire un tel arbre, on peut soit partir de la racine vers les feuilles, soit le contraire.
1,2 et 5 sont non ambigus
3 ambigu, car SSS peut s'obtenir de deux façons : $S- > SS$ puis je dérive le premier S ou le second, d'ou ambiguite sur ()()(). Aussi : je peux jouer a faire $S- > SS$ puis transformer un S en epsilon
3 en non ambigue : $S \to (S)S \mid \epsilon$
Même principe pour 4
Pour les exos du type de 7, la grammaire est non ambiguë si et seulement si l'automate est déterministe.

Corrigé.correction étudiants sur TD6.png

1. L$\cup$ M, L.M L*, ou L et M sont deux langages reconnus par des grammaires d'axiome respectivement X et Y
   On rajoute un nouvel axiome S, et la règle S $- >$ X|Y la regle S $- >$XY, les regles S$- >$ SX $|\epsilon$
2. $\{a^n b^p \mid 0 < p < n\}$
   $S- > TU; T- > aT|a; U- > aUb|ab$ $S- > aSb|X; X- > aX|aab$
3. $\{a^n b^n c^m d^m \mid n, m \in \mathcal{N}\}$
   $L3 : S- > UV; U- > aUb|\epsilon; V- > cVd|\epsilon$
4. $\{a^n b^m c^{n+m} \mid n, m \in \mathcal{N}\}$
   $L5 : S- > aSc|T; T- > bTc|eps$
5. $\{a^n b^m c^p \mid n = m$ ou $m = p\}$
   $L6 : S- > S1X|YS2; S1- > aS1b|eps; S2- > bS2c|eps; X- > cX|esp; Y- > aY|eps$ (ambigu, pour les mots de la forme $a^n b^n c^n$. On sent qu'il y a un problème pour faire non ambigu, et puis une démonstration au-dessus du niveau licence montre que ce n'est pas possible en non ambigu.)

Corrigé. Il y a deux arbres qui correspondent à $(id - id) - id$ et $id - (id - id)$ qui ne sont pas equivalents.

Comment désambiguïser proprement ? Il faut aider les étudiants, voir leur filer la solution :

F2 : E pour expression, T pour terme, F pour facteur
$E \to E + T \mid E - T \mid T$
$T \to (E) \mid id$

G2 : E pour expression, T pour terme, F pour facteur
$E \to E + T \mid E - T \mid T$
$T \to T * F \mid T/F \mid F$
$F \to G \wedge F \mid G$

Notez que F est une récursion droite car l'opérateur puissance associe à droite.
$G \to (E) \mid id$

Corrigé.correction étudiants sur TD6.png On donne le début la fin est a faire en DM

```
Prog      ::= program ident ; Corps .
Corps     ::= var DeclList ; Bloc
Bloc      ::= begin InstrList end
Bloc      ::= Instr
Decllist  ::= Decl ; DeclList  | Decl
Decl      ::= IdList : Type
IdList    ::= ident, IdList  | ident
Type      ::= integer  |
```

Corrigé.L'arbre de dérivation ne présente pas de grosse difficulté ; il faut faire apparaître les valeurs des tokens id et cte, dans une autre couleur pour plus de clarté. Vous le faire faire permet de m'assurer que vous comprenez bien ce qui se passe. Faut prévoir une ou deux pages A4.

L'arbre de syntaxe abstraite : en voici le début. C'est l'arbre de dérivation résumé : On enleve les non-terminaux tout en conservant la structure pour pouvoir ensuite travailler dessus pour compiler. Les non-terminaux "decls" et "instrs" sont représentés par des listes de déclarations et d'instructions.

```
program calcul(
decls(
:( (id "T") ,
array(cte 1,cte 10,integer) ) ,
:( (id "S", id "I"),integer )
) ;
instrs(
:=(id "S")(cte 0),
.....
)
)
```

Corrigé. Algo pour éliminer les non productifs, on augmente un ensemble de productif initialement vide en rajoutant les non-terminaux qui génèrent un mot composé des terminaux, ou de non-terminaux dont on a déjà montré qu'ils sont productifs. On n'explore que les non-terminaux pas encore visités.

Pour trouver les accessibles c'est symétrique. On part de l'axiome et on rajoute tous les non-terminaux qu'ils générent, puis tous les non-terminaux que ces non-terminaux génèrent, et ainsi de suite. On n'explore que les non-terminaux pas encore visités.

L'exemple $S \to aMN, M \to a$ montre que l'ordre compte : si on enlève les non atteignable d'abord M est atteignable, il ne sera pas enlevé par l'étape 2, mais S n'est pas productif, donc il est enlevé par l'étape 2, suite a quoi M ne devient plus atteignable. Donc un terminal productif atteignable peut devenir non atteignable après suppression des non productifs Dans l'autre sens, un terminal productif atteignable, restera productif après qu'on est enlevé les non atteignables, car s'il est atteignable, les non-terminaux dont il a besoin pour produire le seront aussi via lui-même.

Corrigé. La grammaire à nettoyer est déjà propre, mais ça met du temps à le voir

Corrigé.correction étudiants sur TD7.png La clef qui ouvre toutes les portes pour faire du déterministe : avoir un état vers lequel on va en dépilant puis rempilant le fond de pile ; cet état sert a repérer que la pile est réduite au fond de pile, de façon déterministe.

1. premier état, j'empile les a, au premier b, je passe a un Deuxième état, et je dépile les a en lisant les b. Quand j'arrive au fond de pile, je passe au troisième état final. reconnait par pile vide et état final. Pour faire le cas $n = 0$, on rajoute un nouvel état à la fois initial et final. Au premier a, on passe à l'état un en empilant, ca ne reconnait plus par pile vide, si on veut que cela reconnaisse par pile vide, ça devient non déterministe. En cours, on a vu que la bonne notion d'automate à pile déterministe, et la reconnaissance par état final. Cela permet de conserver plus de déterminisme, et donc de passer au non déterministe seulement si c'est vraiment nécessaire.
2. on empile les a sur etat 1, on passe a etat 2 pour dépiler les a en lisant les b ?, quand on trouve pile vide, on passe a etat 3 ou on empile les b, puis etat 4, on depile les b en lisant les c, quand on arrive au fond de pile, on passe a 5 final en enlevant le fond de pile. Ca reconnait par pile vide et état final. Remarques : on peut rajouter une transition de état 1 vers 3 pour gérer le fait que p puisse etre nul. Les transitions d'un état au suivant peuvent être des epsilon (c'est facile) mais alors c'est plus déterministe, Pour conserver le deterministe, il faut que les transitions qu'on rajoute lisent les lettres et donc empile / depile. Il faut faire comme dans la question précédente, rajouter un nouvel état initial. Si on veut en plus ajouter la possibilite q egal 0, alors difficulte, car on reconnait ap bp et meme epsilon. Encore une fois, on peut mettre des transitions epsilon vers l'état final pile vide, mais alors l'automate n'est plus déterministe. Par pile vide, on ne peut pas faire deterministe. Plus généralement, on ne peut pas faire en déterministe pile vide les langages L tq existe u et v reconnu et u prefixe de v. Par contre, c'est possible par état final :on rajoute un autre état final ou on va avec une epsilon transition depuis l'état 2, si on lit le fond de pile. cette transition remet le fond de pile, afin de pouvoir continuer. On repart de ce nouvel état vers l'état 3, s'il y a encore des b a lire)
3. un seul etat, on empile les parenthese, et on depile la parenthese adhoc quand on lit une fermante. Reconnaissance par pile réduite à fond de pile. Par pile vide, on rajoute une transition epsilon qui vide la pile (non determniste). Par etat final deterministe, Un deuxieme etat ou l'on est quand la pile est reduite a fond de pile. Le deuxieme etat est initial. quand je suis sur premier etat et que je vois le fond de pile, je ne peux rien lire, je suis oblige de faire une transition epsilon vers le deuxieme etat
4. On empile le surplus. si je lis aababaab, je sais qu'il y a un surplus de 2, et que ce sont des a. Le nombre de surplus est lu dans la taille de la pile. Sa nature, au choix, par utilisation de deux symboles dans la pile (sol 1) ou par un état (sol 2) sol 1 : un seul etat, quand je lis a, si top-pile est fondpile ou A, j'empile A, si c'est B, je depile. Pour B, c'est invers. reconnaisance par pile reduite a fondpile. sol 2 : deux etat qa et qb. si je lis a, dans qa, j'empile X. dans qb, si fondpile, j'empile A et je passe en qa, sinon, je depile X. inverse pour b. etat initial indifferent. reconnaissance par reduction au fond

depile.

5. le non-deterministe est indispensable, impossible en deterministe (bien que la grammaire soit facilement non ambigu, mais la grammaire peut lire en avance la fin, pas l'automate a pile. Faut laisser les étudiants deviner qu'on décide de commencer à déplier... au hasard ! C'est pas évident à trouver.
6. Impossible. On peut chercher pour voir ou est le probleme. J'empile les a, je compte les b, mais ça détruit le compte et je n'ai plus l'info pour les c.
7. Impossible pour an bm an bm, on empile les a puis les b, puis il faut acceder aux a, mais ils sont dessous et pour y acceder, il faut detruire les b au dessus, et alors on n'a plus l'info pour la fin.

   Enfin, on finit par se convaincre que ces langages ne sont pas algebriques

Corrigé.correction étudiants sur TD8.png

1. algébrique. Faire grammaire pour $m < n$, une pour $n < m < 2n$, une pour $m > 2n$. On peut faire la grammaire non ambigue : la plus difficile pour $n < m < 2n$ s'obtient comme suit :
   S→aaaUbb ; U→aaUb|V ; V→aVb|ϵ
   La règle avec S, génère le mot le plus petit dans le langage : aaabb, avec un non-terminal U intercalé pour rajouter d'autres lettres ; le non-terminal ,U, lui, rajoute l'excédent ; la partie récursive doit donc s'appeler $\#U = (n-3)-(m-2) = n-m-1$ fois. La partie récursive de V sert à finir de rajouter les b et a qui restent à rajouter en nombre égal. $\#V = m-2-(n-m-1) = 2m-n-1$. On vérifie que le nombre de 'a' vaut bien $3+2*\#U+\#V = n$ et le nombre de b, $2+*\#U+\#V = m$ l'automate à pile sera toujours Non-déterministe : au dedéterministebut, trois epsilon transitions depuis l'état initial pour les 3 cas. Dans 2 cas sur ces trois, la suite est déterministe, dans le dernier($n < m < 2n$), ce qu'on peut faire de mieux, c'est empiler un A par 'a', puis deux A par a, puis début lorsqu'on lit les 'b'. Le lmoment ou l'on passe de un A a deux A étant encore non deterministe.
2. Supposons L_2 algébrique, il doit alors vérifier le lemme de la pompe algébrique pour une certaine constante d'itération N. Soit alors $m = a^N b^N a^N b^N$. On a $m \in L$, et $|m| \geq N$, d'où d'après le Lemme de la pompe algébrique, il existe u, x, v, y, w tels que
   — $m = uxvyw$
   — $|xvy| \leq N$
   — $|xy| > 0$
   — $\forall k \in \mathcal{N}, ux^k vy^k w \in L_2$
   Note : Ici, on cherche une contradiction. Comme on veut contredire un "il existe", il faut qu'on montre qu'aucun choix de u, x, v, y, w ne peut marcher : que "pour tout" les cas, ça ne peut pas marcher, il faut donc bien faire attention à n'oublier aucun cas possible.
   Comme $|xvy| < N$ on constate que xvy ne peut contenir qu'un seul changement de lettre (car les blocs de lettres dans μ sont de taille N). On décompose alors selon les cas possibles suivants :
   — Si x ou y contient à la fois des a et des b, ux^2vy^2w contient au moins 5 changements de lettre alors que tous les mots dans W ont au plus 3 changements de lettre, donc $ux^2vy^2w \notin L_2$.
   — Sinon, x et y, sont complétement inclus dans une même plage, ou bien dans une plage de 'a' est dans la plage de 'b' qui suit, ou le contraire en échangeant 'a' et 'b' . Cette plage ou ces deux plages vont grossir lorsque l'on va pomper, mais pas les autres plages restantes, ce qui va induire un déséquilibre entre les deux plages de 'a' et/ou entre les deux plages de 'b'.

   D'où, dans tous les cas, contradiction. L n'est donc pas algébrique.
3. pas algebrique, pompage : Puisque l'alphabet contient une seule lettre 'a', un mot représente un nombre entier : le nombre de 'a'. Lorsque l'on va vouloir pomper notre mot m, à chaque nouveau coup de pompe on va rajouter un certain nombre C de 'a' toujours le même. les entiers ainsi représentés seront de la forme $M + k * C$, ou M est la longueur du mot considéré m et k est le nombre de coups de pompe. Trions tous ces entiers par ordre croissant. L'écart entre deux entiers consécutifs est constant, il vaut C. Mais d'un autre côté, dans notre langage, l'écart entre deux mots consécutifs augmente strictement. Ces deux ensembles de nombres ne peuvent donc être semblables. Il existe donc des valeurs de k pour lesquels $M + k * C$ n'est pas représenté dans notre langage. Pour ces valeurs, on sortira du langage.
4. Pas algébrique cloture, intersection avec $a * b * c*$, on obtient $\{a^n b^n c^n, n > 0\}$
5. Pas algébrique, cloture, intersection avec $a^* b^* a^* b^*$

Corrigé. Correction étudiants sur TD8.png. Le membre droit (resp. à gauche) et la chaîne qui se trouve à droite (resp. à gauche) de la flèche de réécriture de la règle. Ce qui les distingue, c'est que le membre gauche est un seul non-terminal, alors que le membre droit est une chaîne de taille arbitraire d'un mix de non-terminaux et terminaux. premier(C) = c
premier(A) =premier(C)union premier(B) union 'd' = bcd
(A et C sont tous les deux nullifiables, ils peuvent donc "découvrir" donc le d premier(S) =premier(A) union 'a'= $abcd$
Pour les suivants de X il faut regarder les occurrences de X dans les membres droits suivant(S)=vide
suivant(A) = a, d
Suivant(B) = b union suivant(S)union suivant(A)
= b, a, d
suivant(C) =premier(Bb) union premier(Ad)
=premier(B) union premier(A) union $d = bcd$

Corrigé. Elle est difficile, car elle contient un non terminal nullifiable : S. On choisira d'expandre S en ϵ si le prochain caractère est dans suivant(S)=$\{\#, b\}$.

Corrigé. Exemple avec règle dont le membre droit se nullifie.

	(	)	#
S	(S)S	ϵ	ϵ

Analyse :

S#	(()())#	$S \rightarrow (S)S$
(S)S#	(()())#	shift
S)S#	()())#	$S \rightarrow (S)S$
(S)S)S#	()())#	shift
S)S)S#	)())#	$S \rightarrow \epsilon$
)S)S#	)())#	shift
S)S)S#	())#	$S \rightarrow \epsilon$
)S)S#	))#	shift, $S \rightarrow \epsilon$
)S#	)#	shift, $S \rightarrow \epsilon$
#	#	succés

Corrigé.

1. Les trois grammaires permettent de reconnaître le langage $a(;a)^*$
2. La première grammaire est récursive gauche donc n'est pas LL(1). Le membre droit recursif et le terminal vont se retrouver dans la même case de la table de transition qu'il faut dessiner pour enfoncer le clou.
3. La seconde grammaire n'est pas LL(1) car deux productions $L ::= A; L$ et $L ::= A$ ont le même préfixe. Elles vont se retrouver dans la même case de la table de transition qu'il faut dessiner pour enfoncer le clou.
4. La troisième grammaire est ambiguë donc ni LL(1) ni rien. Une telle grammaire est pourrie dès le départ, car l'ambiguïté implique un non-déterminisme intrinsèque.

Corrigé.

Bien entendu il faut décomposer les étapes du déroulement de l'algorithme. Au final, on trouve :

	a	b	c	e	f	#
S	$S \rightarrow XaY$	$S \rightarrow XaY$				
X	$X \rightarrow T$	$X \rightarrow W$				
W		$W \rightarrow bc$				
T	$T \rightarrow ac$					
Y				$Y \rightarrow eY$	$Y \rightarrow f$	$Y \rightarrow \varepsilon$

oui c'est bien LL(1) pourquoi ? ben parcekia au plus un membre droit dans chaque case.

S#	acaebe#	$S \rightarrow XaY$
XaY#	acaebe#	$X \rightarrow T$
TaY#	acaebe#	$T \rightarrow ac$
acaY#	acaebe#	
caY#	caebe#	
aY#	aebe#	
Y#	ebe#	$Y \rightarrow eY$
eY#	ebe#	
Y#	be#	ERREUR

S#	acaeee#	$S \rightarrow XaY$
XaY#	acaeee#	$X \rightarrow T$
TaY#	acaeee#	$T \rightarrow ac$
acaY#	acaebe#	
caY#	caebe#	
aY#	aebe#	
Y#	eee#	$Y \rightarrow eY$
eY#	eee#	
Y#	ee#	$Y \rightarrow eY$
e#Y	ee#	
Y#	e#	$Y \rightarrow eY$
eY#	e#	
Y#	#	$Y \rightarrow \varepsilon$

Corrigé.

1. Les expressions arithmétiques avec plus et multiplié, on l'a déjà vu dans cette grammaire. La grammaire n'est pas ambiguë
2. Dérivation droite : on réécrit à chaque fois le plus à droite
3. Non, On se ramène au cas ou on dépile un seul symbole en stockant un nombre borné de lettres dépilées dans l'état qui reste fini, et en les rempilant derechef, quitte a les redépiler si besoin est.
4. Facile
5. Il faut transiter vers le final si $S\$$ dans la pile. Commencer la simul, les laisser terminer Normalement l'état d'un automate comprend le mot, la pile et l'état. Mais comme il n'y a qu'un État, pas besoin d'en parler. Il faut écrire le mot de pile à gauche et le mot restant à droite ; on voit que la concaténation des deux permet de reconstruire une étape de la dérivation droite. Au final, on fait une dérivation droite à l'envers. On rajoute à droite une colonne action qui indique si on doit lire et pousser sur la pile, ou bien réduire.

pile	*mot*	*action*
$\$$	$id * id + cte$	*lecture*
$\$id$	$*id + cte$	$red\ F \rightarrow d; red\ T \rightarrow F$
$\$T$	$*id + cte$	$lecture; lecture$
$\$T * id$	$+cte$	$red\ F \rightarrow id$
$\$T * F$	$+cte$	$red\ T \rightarrow T * F$
$\$T$	$+cte$	$re\ E \rightarrow T$
$\$E$	$+cte$	$lecture; lecture$
$\$E + cte$		$red\ F \rightarrow cte; red\ T \rightarrow F$
$\$E + T$		$red\ E \rightarrow E + T$
$\$E$		

6. Il y a des conflit reduction lecture pratiquement tout le temps, et aussi des conflit reduction reduction lorqu'on a le choix entre reduire par $E- > E+T$ ou par $E- > T$.
7. Super vachement gènant, on veut faire de l'analyse automatique, si c'est non deterministe, on ne peut pas l'utiliser pour reconnaitre le mot automatiquement, car contrairement au automate d'etat fini, lorsqu'il y a une pile le non détérminisme n'est plus gérable, (le nombre de chemins a explorer n'est plus a priori borner)
8. Faut le détérminiser, on utilise le mot conflit pour désigner la situation dans laquelle deux transitions sont possible, on cherche une méthode pour casser les conflits, en cherchant un moyen de décider laquelle des deux est la bonne, a partir d'info sur la configuration courante de l'automate.
9. Sur cet exemple, une stratégie simple consiste à
   1- si on le choix entre reduction et lecture faut toujours reduire, sauf pour E− >T si le prochain caractére est * (parcequ'alors faut laisser le T pour pouvoir avoir T*F sur la pile. Faut donc considérer le prochain caractére a lire
   -2 toujour reduire le plus possible. Example : si on a le choix entre $E- > E+T$ et $E- > T$ on choisi de réduire par $E- > E+T$.
   Noter que cette stratégie suppose qu'on peut "dépiler pour voir". On devra utiliser un automate a plusieurs états pour voir ce qu'il y a sur le haut de la pile, un peu plus loin que juste le sommet.
10. On rajoute systématiquement un diése à la fin des mots en rajoutant une régle $S' \Rightarrow S\#$

Corrigé.correction étudiants sur TD8.png TD9-1.png TD9-2.png

1. La première grammaire, L'automate LR(0) est :

état1 : $[S' \to .S\sharp]$ $[S \to .L]$ $[L \to .L;A]$ $[L \to .A]$ $[A \to .a]$
état2 : $[S' \to S.\sharp]$
état3 : $[S \to L.]$ $[L \to L.;A]$
état4 : $[L \to A.]$
état5 : $[A \to a.]$
état6 : $[L \to L;.A]$ $[A \to .a]$
état7 : $[L \to L;A.]$

La grammaire n'est pas LR(0) car il y a un conflit entre réduction de $S ::= L$ et une lecture de ; dans l'état 3. Le suivant de S est $\sharp$ donc il n'y a pas de conflit pour l'analyse SLR(1). Les suivants de L sont égaux aux suivants de A c'est-à-dire $\{;, \sharp\}$. La table d'actions et de déplacements est la suivante :

	;	a	$\sharp$	S	L	A
$e1$		shift $e5$		$e2$	$e3$	$e4$
$e2$			succès			
$e3$	shift $e6$		reduce $S ::= L$			
$e4$	reduce $L ::= A$		reduce $L ::= A$			
$e5$	reduce $A ::= a$		reduce $A ::= a$			
$e6$		shift $e5$				$e7$
$e7$	reduce $L ::= L;A$		reduce $L ::= L;A$			

L'automate LR(0) est :
état1 : $[S' \to .S\sharp]$ $[S \to .L]$ $[L \to .A;L]$ $[L \to .A]$ $[A \to .a]$
état2 : $[S' \to S.\sharp]$
état3 : $[S \to L.]$
état4 : $[L \to A.][L \to A.;L]$
état5 : $[A \to a.]$
état6 : $[L \to A;.L]$ $[L \to .A;L]$ $[L \to .A]$ $[A \to .a]$
état7 : $[L \to A;L.]$
La grammaire n'est pas LR(0) car il y a conflit dans l'état 4 entre une réduction de $L ::= A$ et une lecture de ;.
Le suivant de L est égal au suivant de S qui est $\sharp$ donc il n'y a pas de conflit pour l'analyse SLR(1). Les suivants de A sont $\{;, \sharp\}$. La table d'actions et de déplacements est la suivante :

	;	a	$\sharp$	S	L	A
$e1$		shift $e5$		$e2$	$e3$	$e4$
$e2$			succès			
$e3$			reduce $S ::= L$			
$e4$	shift $e6$		reduce $L ::= A$			
$e5$	reduce $A ::= a$		reduce $A ::= a$			
$e6$		shift $e5$			$e7$	$e4$
$e7$			reduce $L ::= A;L$			

La troisième grammaire est ambiguë donc ni LL(1) ni LR(0) ni SLR(1), ni que d'al.

2. Pour la première grammaire :

$e1$	$a;a;a\sharp$	shift $e5$
$e1\ a\ e5$	$;a;a\sharp$	reduce $A ::= a$
$e1\ A\ e4$	$;a;a\sharp$	reduce $L ::= A$
$e1\ L\ e3$	$;a;a\sharp$	shift $e6$
$e1\ L\ e3\ ;\ e6$	$a;a\sharp$	shift $e5$
$e1\ L\ e3\ ;\ e6\ a\ e5$	$;a\sharp$	reduce $A ::= a$
$e1\ L\ e3\ ;\ e6\ A\ e7$	$;a\sharp$	reduce $L ::= L;A$
$e1\ L\ e3$	$;a\sharp$	shift $e6$
$e1\ L\ e3;e6$	$a\sharp$	shift $e5$
$e1\ L\ e3;e6\ a\ e5$	$\sharp$	reduce $A ::= a$
$e1\ L\ e3;e6\ A\ e7$	$\sharp$	reduce $L ::= L;A$
$e1\ S\ e2$	$\sharp$	succès

Pour la seconde grammaire :

$e1$	$a;a;a\sharp$	shift $e5$
$e1\ a\ e5$	$;a;a\sharp$	reduce $A ::= a$
$e1\ A\ e4$	$;a;a\sharp$	shift $e6$
$e1\ A\ e4\ ;\ e6$	$a;a\sharp$	shift $e5$
$e1\ A\ e4\ ;\ e6\ a\ e5$	$;a\sharp$	reduce $A ::= a$
$e1\ A\ e4\ ;\ e6\ A\ e4$	$;a\sharp$	shift $e6$
$e1\ A\ e4\ ;\ e6\ A\ e4\ ;\ e6$	$a\sharp$	shift $e5$
$e1\ A\ e4\ ;\ e6\ A\ e4\ ;\ e6\ a\ e5$	$\sharp$	reduce $A ::= a$
$e1\ A\ e4\ ;\ e6\ A\ e4\ ;\ e6\ A\ e4$	$\sharp$	reduce $L ::= A$
$e1\ A\ e4\ ;\ e6\ A\ e4\ ;\ e6\ L\ e7$	$\sharp$	reduce $L ::= A;L$
$e1\ A\ e4\ ;\ e6\ L\ e7$	$\sharp$	reduce $L ::= A;L$
$e1\ L\ e3$	$\sharp$	reduce $S ::= L$
$e1\ S\ e2$	$\sharp$	succès

3. La première grammaire consiste à associer l'opérateur à gauche et la seconde à droite. Dans le premier cas c'est une recursion gauche, la taille de la pile reste bornée quelle que soit la longueur du mot à analyser. On va donc plutot faire des récursions gauche lorsqu'on spécifie des grammaires de langages de programmation. Le langage des mots de pile est qqc comme ("L ;" +epsilon)(a+A+epsilon) Dans le second cas le mot est presque complètement accumulé sur la pile avant que les réductions des règles mettant en jeu le ; ne puissent commencer. Le langage des mots de pile est ((A ;)* (a+A+L+epsilon))

   Si on utilisait la troisième grammaire avec des précédences on aurait la même distinction suivant si on choisit une associativité gauche ou droite.

Corrigé.correction étudiants : TD10g1.png il y a un conflit lecture $*$, réduction $E \to T$ résolus lorsque l'on regarde les suivants.

Corrigé.L'automate LR(0) est le suivant :
état1 : $[S \to .E\sharp]$ $[E \to .\texttt{id}]$ $[E \to .\texttt{id}(E)]$ $[E \to .E + \texttt{id}]$
état2 : $[S \to E.\sharp]$ $[E \to E. + \texttt{id}]$
état3 : $[E \to \texttt{id}.]$ $[E \to \texttt{id}.(E)]$
état4 : $[E \to E + .\texttt{id}]$
état5 : $[E \to \texttt{id}(.E)]$ $[E \to .\texttt{id}]$ $[E \to .\texttt{id}(E)]$ $[E \to .E + \texttt{id}]$
état6 : $[E \to E + \texttt{id}.]$
état7 : $[E \to \texttt{id}(E.)]$ $[E \to E. + \texttt{id}]$
état8 : $[E \to \texttt{id}(E).]$

L'automate LR(0) présente un conflit entre une lecture de (et une réduction de $E ::= \texttt{id}$) dans l'état3, mais $\text{SUIV}(E) = \{\sharp,), +\}$ donc ce n'est pas un conflit SLR(1).

La grammaire n'est donc pas LR(0), elle est par contre SLR(1) donc par conséquent LR(1)

Corrigé. Correction étudiants dans TDanalyseAscendanteAutomateLR1.pdf Pas LR(0) car après lecture de 'd', conflit entre lecture de 'c' et reduction M− >d. Conflit non résolu par SLR(1) car suivant(M)={a,c}. On construit l'automate LR(1), celui ci est sans conflit, et il corresponds a l'automate LALR(1), car tout les états ont une structure LR(0) qui est différente. Dans TDanalyseAscendanteAutomateLR1.pdf, il y a aussi l'analyse des mot "da" et "dc" qui est faite, pour bien montrer que ca marche pour LR(1) et que ca marche pas avec juste SLR(1).

Corrigé.

1. $$\begin{aligned} \text{SUIV}(T) &= \text{SUIV}(A) \cup \{\sharp, *\} &= \{\rightarrow, \sharp, *\} \\ \text{SUIV}(A) &= \{\rightarrow\} \end{aligned}$$

2. Les états sont :

$$\begin{array}{lll} s_1 : & S \rightarrow .T\sharp & \\ & T \rightarrow .\texttt{int} & \\ & T \rightarrow .A \rightarrow \texttt{int} & \\ & A \rightarrow .T & \\ & A \rightarrow .T * A & \\ s_2 : & T \rightarrow \texttt{int}. & \\ s_5 : & A \rightarrow T * .A & \\ & T \rightarrow .\texttt{int} & \\ & T \rightarrow .A \rightarrow \texttt{int} & \\ & A \rightarrow .T & \\ & A \rightarrow .T * A & \\ s_6 : & T \rightarrow A \rightarrow .\texttt{int} & \end{array} \qquad \begin{array}{lll} s_3 : & S \rightarrow T.\sharp & \\ & A \rightarrow T. & \\ & A \rightarrow T. * A & \\ s_4 : & T \rightarrow A. \rightarrow \texttt{int} & \\ & & \\ & & \\ & & \\ s_7 : & A \rightarrow T. & \\ & A \rightarrow T. * A & \\ s_8 : & A \rightarrow T * A. & \\ & T \rightarrow A. \rightarrow \texttt{int} & \\ s_9 : & T \rightarrow A \rightarrow \texttt{int}. & \end{array}$$

La table de transitions est :

	int	→	*	♯	T	A
s_1	shift s_2				s_3	s_4
s_2		. . . reduce $T ::=$ int . . .				
s_3		reduce $A ::= T$	shift s_5	end		
s_4		shift s_6				
s_5	shift s_2				s_7	s_8
s_6	shift s_9					
s_7		reduce $A ::= T$	shift s_5			
s_8		shift s_6 reduce $A ::= T * A$				
s_9		. . . reduce $T ::= A \rightarrow$ int . . .				

La grammaire n'est pas SLR(1) (état s_8) avec le symbole d'avance →.

3. Le conflit est un conflit shift/reduce dans une situation de la forme $T * A. \rightarrow \texttt{int}$ on ne sait pas s'il faut réduire pour obtenir $A \rightarrow \texttt{int}$ ou au contraire lire $T * A \rightarrow .\texttt{int}$ ce qui amènera ensuite à réduire $A \rightarrow \texttt{int}$ en T. Le problème se pose dans le cas du type

$$\texttt{int} * \texttt{int} \rightarrow \texttt{int} \rightarrow \texttt{int}$$

qui a deux interprétations possibles : une fonction à un argument fonctionnel de type $\texttt{int} * \texttt{int} \rightarrow \texttt{int}$ ou bien une fonction à deux arguments l'un de type int et l'autre de type $\texttt{int} \rightarrow \texttt{int}$. Cet exemple montre que la grammaire est ambiguë.

NB avec le type

$$\texttt{int} * \texttt{int} \rightarrow \texttt{int}$$

on arrive aussi à l'état ou il y a conflit.

4. Il n'y a pas de conventions qui s'impose, le mieux est probablement de forcer un parenthésage du type des arguments lorsqu'ils sont fonctionnels.