TD2 - Induction structurelle 08/09/25

Exercice 1 :

Q1 longueur([]) = 0 (cas de base)
longueur(x::ℓ) = 1+longueur ℓ (hérédité)

Q2 : Par récurrence sur ℓ_1 :
concat([], ℓ_2) = ℓ_2
concat(x::ℓ, ℓ_2) = x::concat(ℓ, ℓ_2)

---

3) $\forall \ell_1, \ell_2 \in$ liste, longueur(concat(ℓ_1, ℓ_2)) = longueur(ℓ_1)+longueur(ℓ_2)
Par récurrence sur ℓ_1 :

- longueur(concat([], ℓ_2])
  = longueur(ℓ_2)
  = longueur(ℓ_2) + longueur([])
  "
  0

- longueur(concat(x::ℓ, ℓ_2))
  = longueur(x :: concat(ℓ, ℓ_2)) [définition de concat]
  = 1+ longueur(concat(ℓ, ℓ_2)) [définition de longueur]
  = 1 + longueur(ℓ) + longueur(ℓ_2) [par hyp de rec]
  = (1+long(ℓ)) + long(ℓ_2)
  = long(x::ℓ) + long(ℓ_2)

---

4) $\forall \ell_1, \ell_2, \ell_3 \in$ liste, concat(ℓ_1, concat(ℓ_2, ℓ_3)) = concat(concat(ℓ_1, ℓ_2), ℓ_3)
Par récurrence sur ℓ_1 :
concat([], concat(ℓ_2, ℓ_3))
= concat(ℓ_2, ℓ_3)
= concat(concat([], ℓ_2), ℓ_3))
"

concat(x::ℓ, concat(ℓ_2, ℓ_3))
= x:: concat(ℓ, concat(ℓ_2, ℓ_3))
= x:: concat(concat(ℓ, ℓ_2), ℓ_3) [par HR]
= concat(x:: concat(ℓ, ℓ_2), ℓ_3)
= concat(concat(x::ℓ, ℓ_2), ℓ_3) [définition deconcat]
"

Exercice 2 : rev([]) = [], rev(x::ℓ) = concat(rev(ℓ), x::[])

1) $\forall \ell \in$ liste, longueur(rev(ℓ)) = longueur(ℓ)
Par récurrence sur ℓ_1 :
longueur(rev([])) = longueur([])
[par définition de rev]

longueur(rev(x::ℓ))
= longueur(concat(rev(ℓ)), x::[])
= longueur(rev(ℓ)) + longueur(x::[])
= longueur(ℓ) + longueur(x::[]) par HR
= longueur(ℓ) + longueur(x) + longueur([])
= longueur(ℓ) + 1 = longueur(ℓ_1)

---

2) $\forall \ell_1, \ell_2 \in$ liste, rev(concat(ℓ_1, ℓ_2)) = concat(rev(ℓ_2), rev(ℓ_1)) (*)
Par récurrence sur ℓ_1 :

- cas []:
rev(concat([], ℓ_2)) = rev(ℓ_2)
= concat(rev(ℓ_2), [])
= concat(rev(ℓ_2), rev([]))

- Cas x::ℓ
rev(concat(x::ℓ_1, ℓ_2))
= rev(x:: concat(ℓ, ℓ_2)) [def de concat]
= concat(rev(concat(ℓ, ℓ_2)), x::[]) [def de rev]
= concat(concat(rev(ℓ_2), rev(ℓ)), x::[]) par HR
= concat(rev(ℓ_2), concat(rev(ℓ), x::[])) associativité de Concat
= concat(rev(ℓ_2), rev(x::ℓ))

3) $\forall \ell$ liste, $rev(rev(\ell)) = \ell$

Par récurrence sur ℓ :

- cas $[]$: $rev(rev([])) \overset{def}{=} rev([]) \overset{def}{=} []$

- cas $x::\ell$:

$rev(rev(x::\ell)) = rev(concat(rev(\ell)), x::[])$

$= concat(rev(x::[]), rev(rev(\ell)))$ [Q. précédente]

$= concat(x::[], \ell)$ [par HR]

$= x::\ell$

Exercice 3

$rev\_rt(\ell) = aux(\ell, [])$

$aux([], a) = a$

$aux(x::\ell, a) = aux(\ell, x::a)$

Mq $\forall \ell$ liste, $rev\_rt(\ell) = rev(\ell)$

- Montrons par récurrence sur ℓ que $\forall a$, $aux(\ell, a) = concat(rev(\ell), a)$
  - cas $[]$: $aux([], a) = a$
    $= concat([], a)$
    $= concat(rev([]), a)$
  - cas $x::\ell$: $aux(x::\ell, a) = aux(\ell, x::a)$
    $= concat(rev(\ell), x::a)$ par HR, ℓ sous-liste
    $= concat(rev(\ell), concat(x, a))$
    $= concat(concat(rev(\ell), x::[]), a)$ associativité de concat
    $= concat(rev(x::\ell), a)$

Exercice 4 : E : arbre vide

$N(t_1, n, t_2)$: t_1, t_2 sag, sad, n : étiquette de la racine

1) infixe : arbre → liste

$infixe(E) = []$

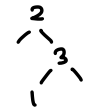
$infixe(N(t_1, n, t_2)) = [infixe(t_1) :: n :: infixe(t_2)]$

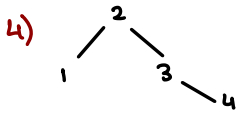
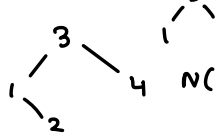
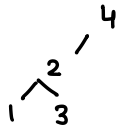
2) appartient : $\mathbb{N}$ x arbre → $\mathbb{B}$

$appartient(n, E) = false$

$appartient(n, N(t_1, m, t_2)) = (n = m) \vee (appartient(n, t_1)) \vee (appartient(n, t_2))$

3) a) $N(N(F,1,F),2,N(F,3,F))$  oui

b) $N(F,2,N(N(F,1,F),3,F))$  non car 4 est dans le sous-arbre droit du noeud portant l'étiquette 2

4)   $N(N(F,1,F),2,N(F,3,F)),4,F)$ 

5) Si t ABR, alors infix(t) est trié [HYP]

Par récurrence sur t :

cas E : infix(E) = []. La liste vide est triée. = Nil

cas $N(t_1, n, t_2)$: Cet arbre étant un ABR, ses sous-arbres t_1, t_2 sont des ABR également.

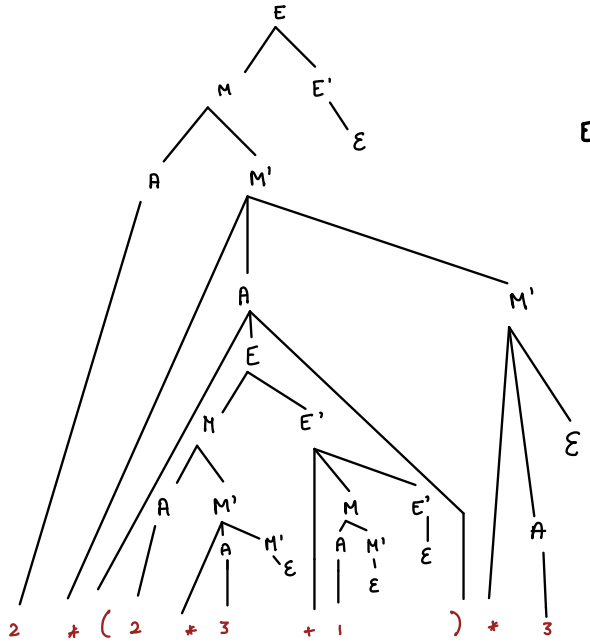
Par hypothèse de récurrence, $infix(t_1)$ et $infix(t_2)$ sont triées. En outre, tous les elts de $infix\ t_1$ (resp $infix\ t_2$) sont $<=$ (resp $>=$) à n.

Donc $n :: infix(t_2)$ est triée, et donc $concat(t_1, n::t_2) = infix(t_1, n, t_2)$ est triée également.

TD 3 - grammaires

Ex1

2*(2*3+1)*3



Etapes de dérivation:

E ⇒ ME'
⇒ AM'E'
⇒ nM'E'
⇒ n*AM'E'
⇒ n*(E)M'E'

⇒ n*(ME')M'E'
⇒ n*(AM'E')M'E'
⇒ n*(nM'E')M'E'
⇒ n*(n*AM'E')M'E'
⇒ n*(n*nM'E')M'E'

⇒ n*(n*nE')M'E'
⇒ n*(n*n+ME')M'E'
⇒ n*(n*n+AM'E')M'E'
⇒ n*(n*n+nM'E')M'E'
⇒ n*(n*n+n)*n

Ex2

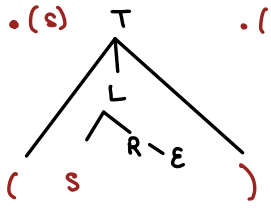
1) T → (L)
L → SR
R → ,S | ε

2) T → (L)
L → SR | ε
R → ,SR | ε

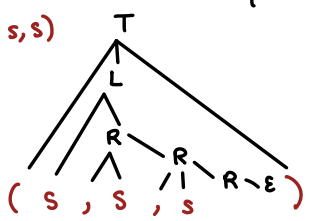
3) Les deux grammaires n'autorisent pas "," comme symbole terminal et ne peuvent donc pas dériver (s, s,).

- une seule règle permet d'introduire le symbole virgule et cette règle introduit nécessairement un symbole s immédiatement après. On ne peut donc pas avoir une virgule suivie immédiatement par une parenthèse.

- (s)



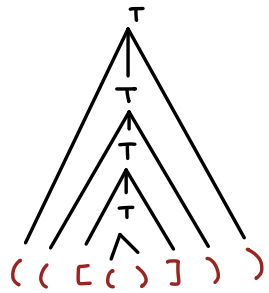
- (s, s, s)



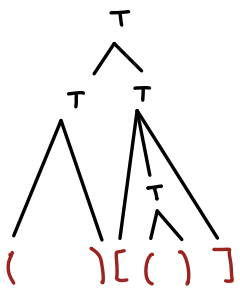
Ex3

Grammaire: T → ()
(+) | []
| TT
| [T]
| (T)

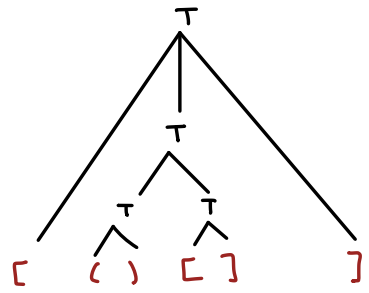
- ((([])))



- ()[()]



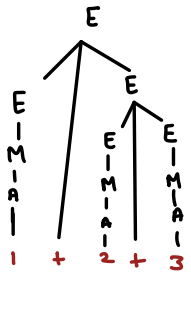
- [()[]]



Ex4

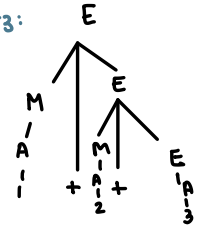
1) (1+2+3)

- G1:

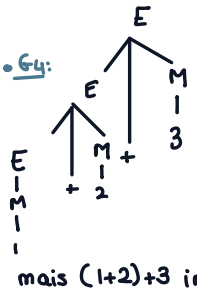


- G2 : impossible
force l'utilisation de parenthèses à chaque enchaînement de 2+ ou *
Elle n'accepte pas 1+2+3 mais accepte (1+2)+3

- G3:

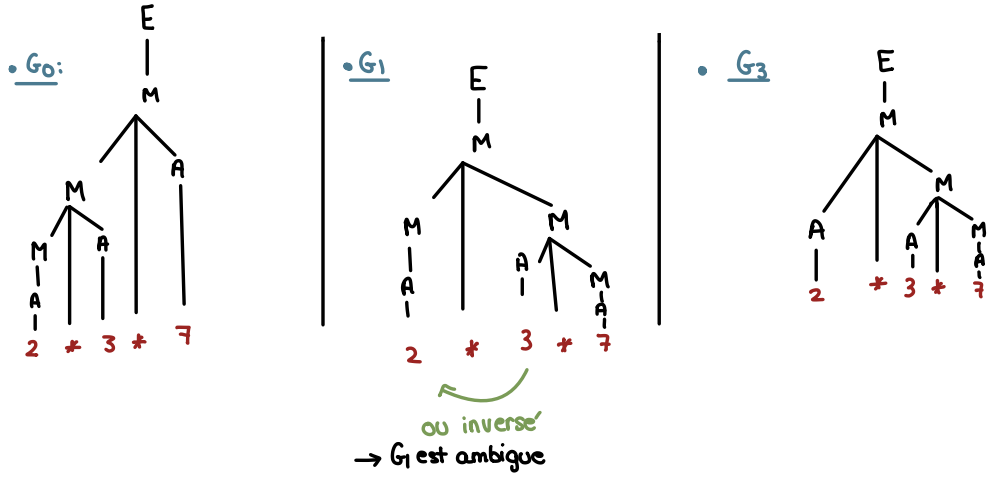


- G4:



mais (1+2)+3 impossible car on ne peut pas avoir (.)+. mais que (.)*.

2) 2*3*7 G_0, G_1 et G_3



---

3) Grammaire: G_0 : E → E+M | E-M | M

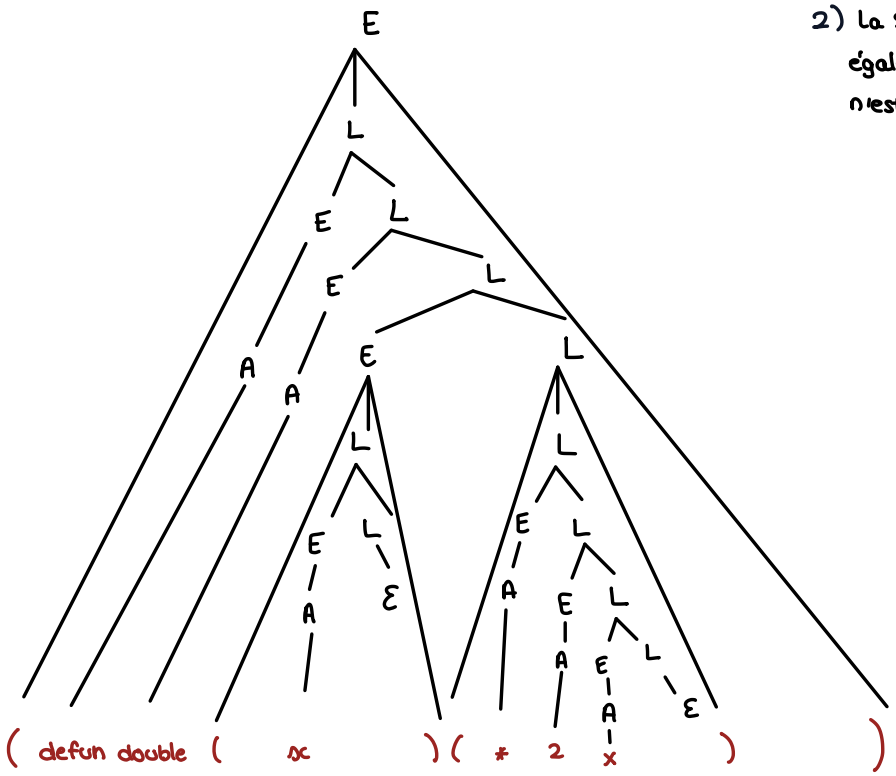
G_1 : E → E+E | E-M | M

reste ambigüe

G_3 : E-M ne fonctionne pas, on utilise plutôt M-E

E → M+E | M-E | M (E-E ne convient pas pour G_0)

Ex5



2) La seule règle introduisant une parenthèse introduit également une parenthèse fermante. Donc la phrase n'est pas possible

TD 5 - Automates 29/09/25

Ex 1:

1) $a(a|b)^*$
Mots commençant par a

2) $a^* | b^*$
Mots uniquement faits de a ou vide ou de b

3) $(b|ab)^*(a|\varepsilon)$
Mots n'ayant pas de a consécutifs

4) $(aa|b)^*$
Mots avec des blocs de a de longueur paire

5) $(ab^*a|b)^*$
Mots ayant un nombre pair de a.

Ex.2:

1) mots de longueur au plus deux
$(a|b|\varepsilon)(a|b|\varepsilon)$
↑ taille au plus 2

2) mots de longueur paire
$((a|b)(a|b))^*$

3) mots contenant la séquence ab mais pas la séquence ba
$(a^*)(ab)(b)^*$ ou: $a^+ b^+$

4) mots contenant au plus l'une des deux séquences ab ou ba
$(a^*b^*)|(b^*a^*)$

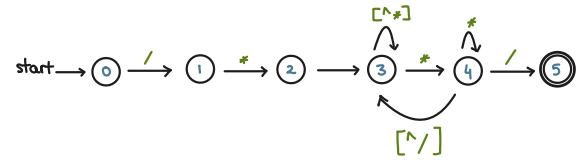
5) mots contenant la séquence ab mais pas la séquence aa
$b^*(ab^+)^+a$?
ou: $b^* ab(ab|b)^*(a|\varepsilon)$

Ex3: /* aa ... */

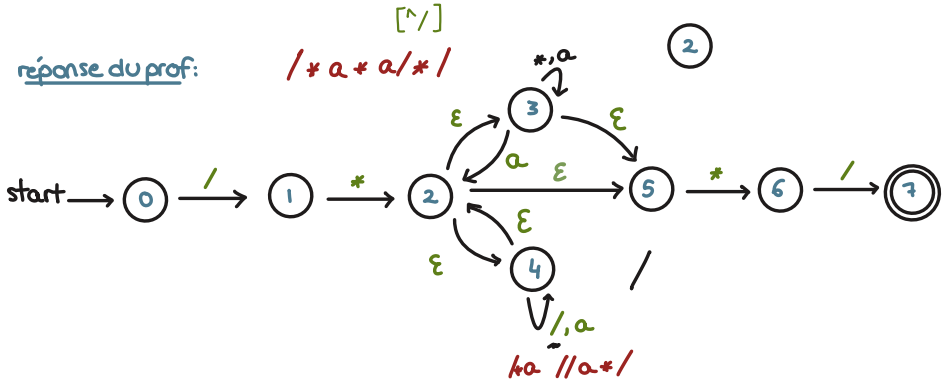
Δ! pas d'étoiles suivies d'un \ ds le commentaire

/**/ /* ab ***/ arrêter l'automate /*/*/ /**/*/ ↑ ne marche pas

1)

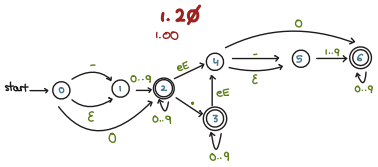


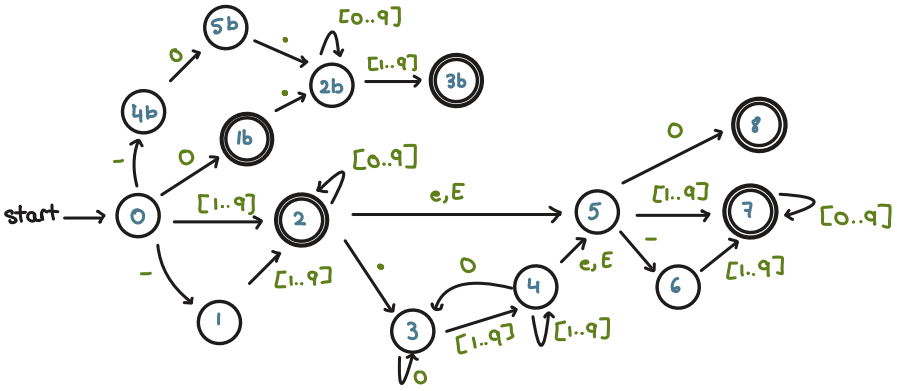
réponse du prof: /* a * a/*/



/a //a*/

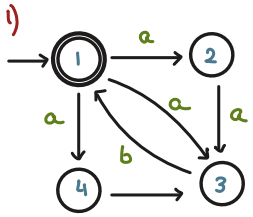
2) nombres en écriture décimale





Ex4 [determinisation]

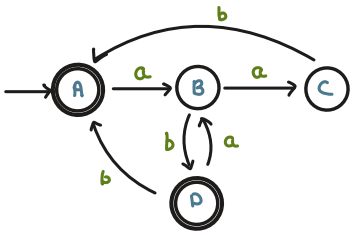
1)



		a	b
A=	1	2,3,4	
	2	3	
C=	3		1
	4		3

B= 2,3,4 3 1,3

D= 1,3 2,3,4 1



A = {1}

B = {2,3,4}

C = {3}

D = {1,3}

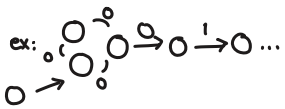
2)

Language correspondant: $(ab|aab|abb)^*$

Ex5: $\underbrace{0\dots01}_{N+1}\underbrace{0\dots0}_{N+1}$

1) Ces N premières transitions visitent N+1 états. Comme il n'y a que N états dans l'automate, au moins 1 état est visité 2 fois.

La portion de chemin entre les 2 1ères visites de cet état forme un cycle de longueur $k \geq 1$.

2) ex: 

On obtient encore 1 chemin en retirant 1 passage dans le cycle identifié. Ce chemin raccourci est étiqueté par le mot $0^{N+1-k}10^{N+1}$ qui est alors reconnu mais sans être un palindrome.

3) Structure complète: preuve par l'absurde.

On suppose que P est reconnaissable, et par les 2 points précédents mentionnés en 1) et 2), on en déduit la contradiction (la déclaration que $0^{N+1-k}10^{N+1}$ appartient à P). Donc P ne pouvait pas être reconnaissable. $P \equiv \neg P \Rightarrow \bot \equiv \neg\neg P \vee \bot \equiv P \vee \bot \equiv P$ (par l'absurde)

4) Lemme de l'étoile:

$m = m_1 m_2 m_3$, $m_2 \neq \varepsilon$ $L(m_1 m_2^* m_3) \in L$

Soit p tel que

On prend $m = \alpha^{2^p}$ et $N < 2^p$

Soit $m_2 = \alpha^x$ avec $1 \leq x \leq N < 2^p$

On aurait $\alpha^{2^p+x} \in L$ (on a repeté m_2 1 fois de +) → pas reconnaissable

impossible car $2^p < 2^p + x < 2^p + 2^p$ (2^{p+1})

pas une puissance de 2

TD 6 - Analyse ascendante

Ex1:

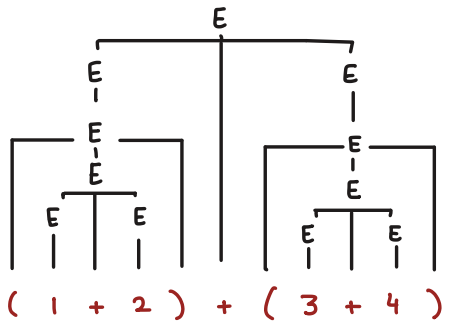
$S \rightarrow E\#$

$E \rightarrow n$
$\quad | E+E$
$\quad | E*E$
$\quad | (E)$

1) étapes de l'analyse ascendante de (1+2)+(3+4)

pile	entrée	action
ε	(1+2)+(3+4) #	S(shift) (
(	1+2)+(3+4)#	shift n→1
(n	+2)+(3+4) #	reduce n→E
(E	+2)+(3+4) #	shift +
(E+	2)+(3+4) #	shift 2→1
(E+n	)+(3+4)#	reduce E<n
(E+E	)+(3+4) #	reduce E<E+E
(E	)+(3+4) #	shift)
(E)	+(3+4)#	reduce E<(E)
E	+(3+4)#	shift +
E+	(3+4) #	shift (
E+(	3+4) #	shift 3→n
E+(n	+4)#	reduce E<n
E+(E	+4)#	shift +
E+(E+	4)#	shift 4→n
E+(E+n	)#	reduce E<n
E+(E+E	)#	reduce E<E+E
E+(E	)#	shift)
E+(E)	#	reduce E<(E)
E+E	#	reduce E<E+E
E	#	Shift #
E#	∅	reduce S<E#

arbre de dérivation:



2) (1+2)(3)

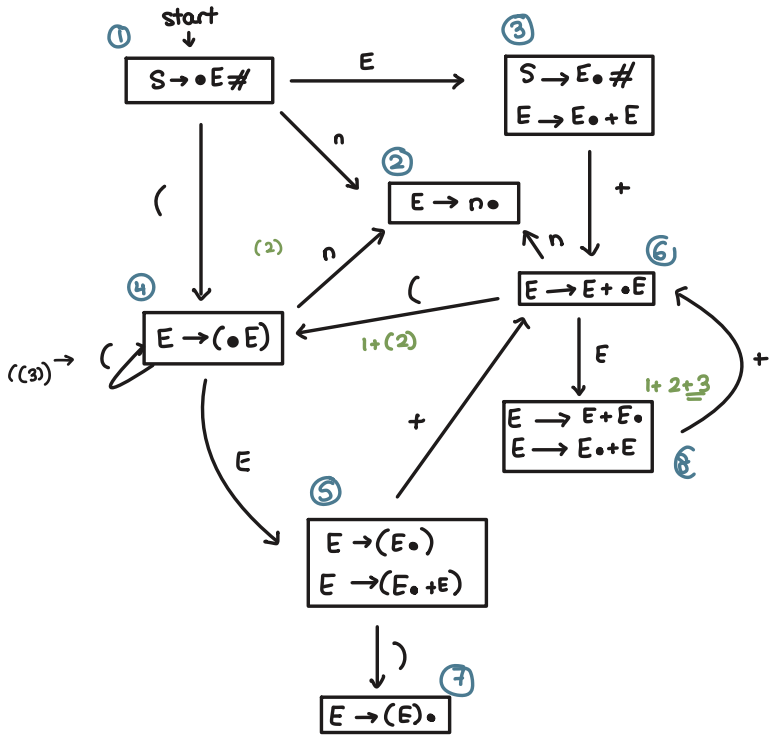
pile	entrée	action
ε	(1+2)(3)#	S
(	1+2)(3)#	S
(1	+2)(3)#	R[E→n]
(E	+2)(3)#	S
(E+	2)(3)#	S
(E+2	)(3)#	R[E→n]
(E+E	)(3)#	R[E→E+E]
(E	)(3)#	S
(E)	(3)#	R[E→(E)]
E	(3)#	échec

Ex 2

S = E#

E = n
| E+E
| (E)

Automate LR(0)

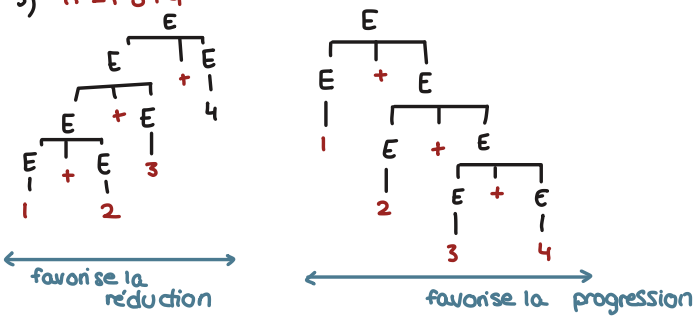


CONFLIT (8): (associativité de l'addition)

progression ou réduction possible

(lié à une ambiguïté de la grammaire
↝ 2 arbres différents pour une même expression)

3) 1+2+3+4

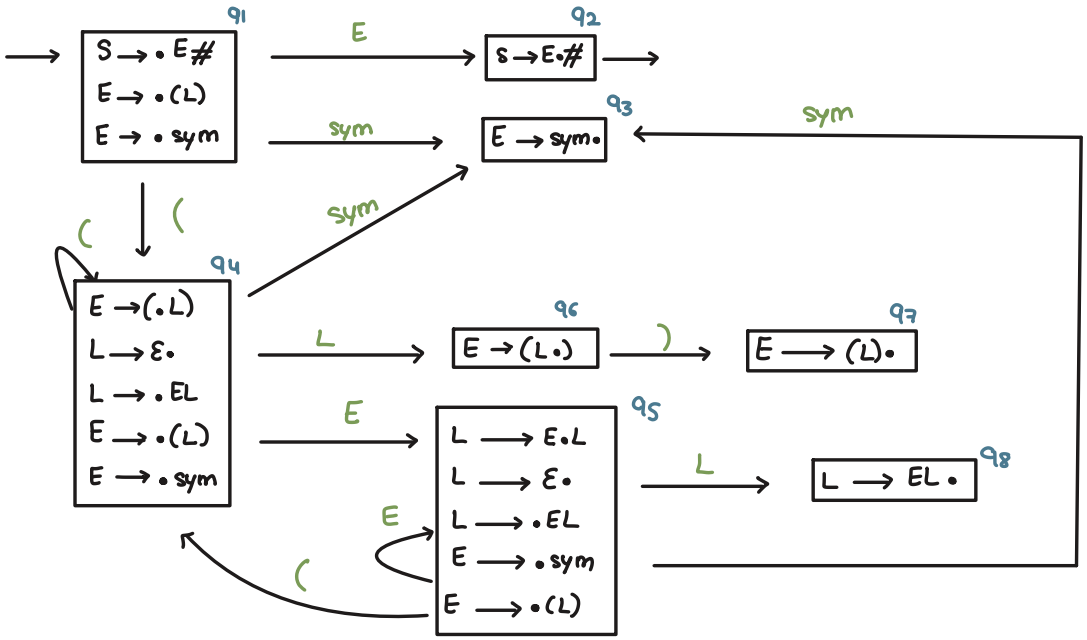


Ex 3

1) Grammaire G:

S ::= E#
E ::= sym
| (L)
L ::= EL
| ε

2) déterminiser l'automate :



3) tables d'action et de déplacement

	Actions				Sauts	
	sym	(	)	#	E	L
q1	q3	q4			q2	
q2				OK.		
q3	E ← sym					
q4	q3	q4	L ← ε (conflit)		q5	q6
q5	q3	q4	L ← ε		q5	q8
q6			q7			
q7	E ← (L)					
q8	L ← EL					

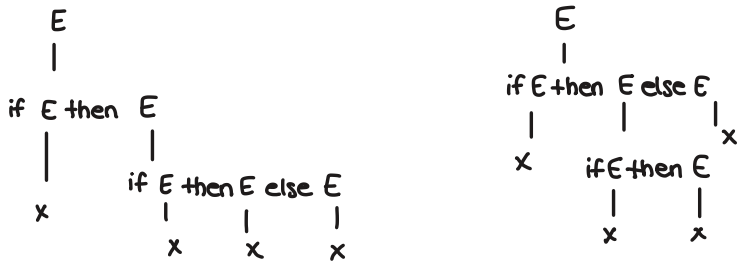
Grammaire SLR(1)

↕ les états 4 et 5 permettent une réduction de la règle $L \leftarrow \varepsilon$, mais également de progresser avec les symboles sym et (
c'est un conflit shift-/reduce

4) conflit état 4 : (• S) sur S après avoir lu la 1ère parenthèse
conflit état 5 : (S( ))
conflit sur la 2e parenthèse
↝ la pile contient E et le prochain symbole est (

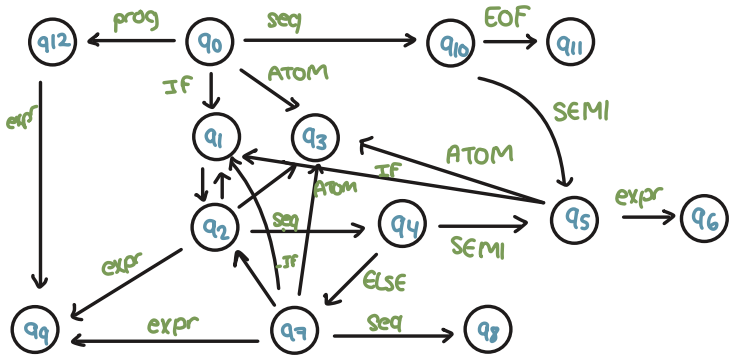
5) $L \leftarrow \varepsilon$: sert à terminer la liste
↳ ne l'appliquer que lorsque le prochain symbole est).

Ex 4 :
if x then if x then x else x



TP 7

1) if. automaton



q0: S → • prog

q1: expr → IF • COND seq
IF • COND seq ELSE seq

q2: expr → IF COND • seq
IF COND • seq ELSE seq

q3: expr → ATOM •

q4: expr → IF COND seq •
IF COND seq • ELSE seq

q5: seq → seq SEMI • expr

q6: seq → seq SEMI expr •

q7: expr → IF COND seq ELSE • seq

q8: expr → IF COND seq ELSE seq •
seq • SEMI expr

q9: seq → expr •

q10: prog → seq • EOF
seq → seq • SEMI expr

q11: prog → seq EOF

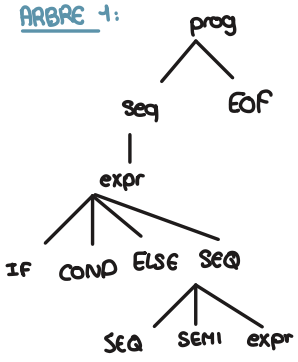
q12: S → prog •

1er conflit: état 8, shift/reduce sur SEMI
IF COND seq ELSE seq ; exp EOF

2e conflit: similaire

3e conflit: sur IF COND IF COND seq ELSE seq

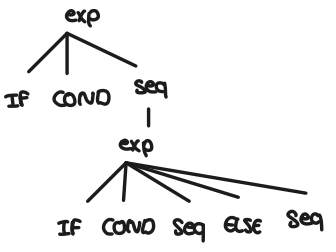
ARBRE 1:



ARBRE 2:

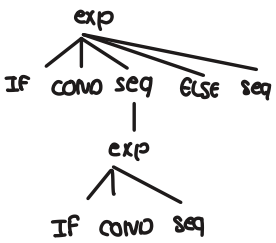
→ favoriser la réduction
→ % nonassoc SEMI

ARBRE 1:



favorise la progression
% nonassoc ELSE

ARBRE 2:



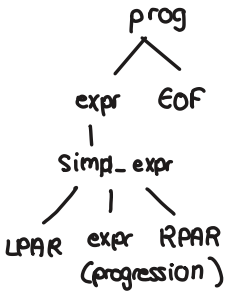
Δ! Fonctions:

curryfiée : let f x y = $x \longmapsto (y \longmapsto x+y)$ f 3 : $y \longrightarrow 3+y$

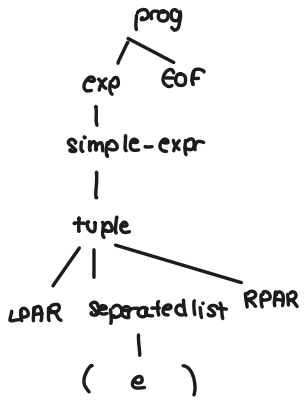
décurryfiée : let f(x, y) = x+y f 3 5 : 3+5 = 8

Exercice 2:

conflit 1:



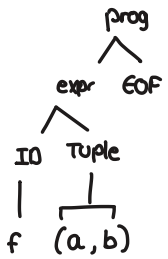
or



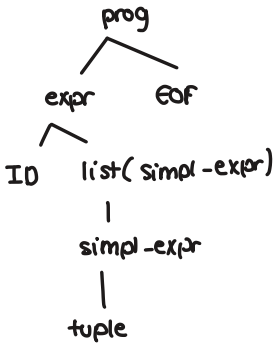
ambiguité entre
(e) vu comme une expr entre parenthèse
et (e) vu comme un n-uplet à 1 élément

→ favorise la réduction

conflit2: sur ID tuple EOF



Ambiguité entre:
- f(a,b) vu comme un appel à la C (décurrifié)



- f(a,b) comme un appel curryfié à la caml dont le 1er serait un n-uplet

f(a,b)
un seul argument (a,b)

TD8

$$\frac{}{\Gamma \vdash n : int} \qquad \frac{}{\Gamma \vdash x : \Gamma(x)} \qquad \frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int}{\Gamma \vdash e_1 - e_2 : int}$$

$$\frac{\Gamma \vdash e_1 : \tau \quad \dots \quad \Gamma \vdash e_k : \tau}{\Gamma \vdash [e_1, \dots, e_k] : \tau[]} \qquad \frac{\Gamma \vdash e_1 : \tau[] \quad \Gamma \vdash e_2 : int}{\Gamma \vdash e_1[e_2] : \tau}$$

Ex 1:

1) t[1]+2 : ok avec type int[] pour t et résultat de type int
2) t[t[3]] : ↑
3) t[3][4] : ok avec type τ[][] pour t et resultat de type τ
4) t[3][t[4]] : non car le type de τ devrait à la fois avoir la forme τ[][] et la forme int[]
5) [t[1], t[3]] : ok avec type τ[] pour t et résultat de type τ[] (4ème règle de typage)
6) [t, t[0]] : non typable car t de type τ = τ'[]

$$\underbrace{[t}_{\tau}, \underbrace{t[0]}_{\tau'}]$$ avec $\tau = \tau'$

Ex 2:

Notion de paire :
- (e_1, e_2) construction d'une paire
- fst(e) extraction de la première composante
- snd(e) _____ 2e _____

$\tau_1 \times \tau_2$ type des paires dont la 1e composante a le type τ_1 et la 2e τ_2

Règles de typage :

- $$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}$$
- $$\frac{\Gamma \vdash e = (e_1, e_2) : \tau_1 \times \tau_2}{\Gamma \vdash fst(e) : \tau_1} \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash snd(e) : \tau_2}$$

Ex 3:

- constantes true et false
- op binaires $e_1 < e_2$ $e_1 = e_2$ $e_1 \&\& e_2$
- exp conditionnelle $e_0 ? e_1 : e_2$

Règles de typage :

- $$\frac{}{\Gamma \vdash true : bool} \qquad \frac{}{\Gamma \vdash false : bool}$$
- $$\frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int}{\Gamma \vdash e_1 < e_2 : bool} \qquad \frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 = e_2 : bool} \qquad \frac{\Gamma \vdash e_1 : bool \quad \Gamma \vdash e_2 : bool}{\Gamma \vdash e_1 \,\&\&\, e_2 : bool}$$
- $$\frac{\Gamma \vdash e_0 : bool \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_0 ? e_1 : e_2 : \tau}$$

Ex 4:

Definir une fonction F qui transforme une expression e en éliminant l'opérateur &&
($e_1 \&\& e_2$ doit être remplacée par une expression conditionnelle)

Demontrer que si $\Gamma \vdash e : \tau$ alors $\Gamma \vdash F(e) : \tau$

$F(e_1 \&\& e_2) = F(e_1) ? F(e_2) : False$

On veut mq pour tout expr. e de type τ alors F(e) est de type τ

Pour les cas sans && : F est un morphisme

Pour $F(e_1 \&\& e_2)$: on démontre la propriété par récurrence sur la dérivation $\Gamma \vdash e : \tau$

Seul cas intéressant: $\Gamma \vdash e_1 \&\& e_2 : bool$ avec les prémisses $\Gamma \vdash e_1 : bool$ et $\Gamma \vdash e_2 : bool$

Par HR, on a $\Gamma \vdash F(e_1) : bool$ et $\Gamma \vdash F(e_2) : bool$

$\Gamma \vdash false : bool$ avec un type qui est bien égal au type de e_2,

→ on peut appliquer la règle de typage du ? pour obtenir:

$\Gamma \vdash F(e_1) ? F(e_2) : false : bool$

Ex 5:

$n^{\{x \leftarrow e'\}} = n$

$(e_1 - e_2)^{\{x \leftarrow e'\}} = e_1^{\{x \leftarrow e'\}} - e_2^{\{x \leftarrow e'\}}$

$y^{\{x \leftarrow e'\}} = \begin{cases} e' & \text{si } x = y \\ y & \text{si } x \neq y \end{cases}$

$(e_1[e_2])^{\{x \leftarrow e'\}} = e_1^{\{x \leftarrow e'\}}[e_2^{\{x \leftarrow e'\}}]$

$[e_1, \_\_\_, e_k]^{\{x \leftarrow e'\}} = [e_1^{\{x \leftarrow e'\}}, \_\_\_, e_k^{\{x \leftarrow e'\}}]$

Mq si $\Gamma \vdash e' : \tau'$ et $\Gamma, x : \tau' \vdash e : \tau$ alors $\Gamma \vdash e^{\{x \leftarrow e'\}} : \tau$

Récurrence sur la dérivation de $\Gamma, x : \tau' \vdash e : \tau$

- cas $\Gamma, x : \tau' \vdash n : int$: (immédiat car $n^{\{x \leftarrow e\}} = n$)
- cas $\Gamma, x : \tau' \vdash e_1 - e_2 : int$: avec $\Gamma, x : \tau' \vdash e_1 : int$ et $\Gamma, x \vdash e_2 : int$ par règle de typage

  $(e_1 - e_2)^{\{x \leftarrow e'\}} = e_1^{\{x \leftarrow e'\}} - e_2^{\{x \leftarrow e'\}}$

  Par HR on a: $\Gamma \vdash e_1^{\{x \leftarrow e'\}} : int$ et $\Gamma \vdash e_2^{\{x \leftarrow e'\}} : int$

  On en déduit $\Gamma \vdash e_1^{\{x \leftarrow e'\}} - e_2^{\{x \leftarrow e'\}} : int$

- cas $\Gamma, x : \tau' \vdash y : \tau$ avec τ le type associé à y par l'environnement étendu (Γ, x, τ')

  deux sous cas:

  - si $x = y$ alors $\tau = \tau'$, et $y^{\{x \in e'\}} = e'$

    on conclus par hypothèse, $\Gamma : e' \vdash \tau'$
  - si $x \neq y$ alors $\tau = \Gamma(y)$ et $y^{\{x \in e'\}} = y$

    Conclusion par la règle de typage des variables.

  Les 2 cas des tableaux sont similaires au cas de la soustraction.

TP9: Projet

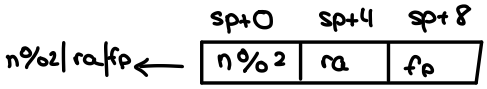
TD 10

Ex1: [Traduire en MIPS]

convention: $a0 arg1
$a1 arg2
$v0 resultat

1)

```
let rec compte n =
  if n = 0
  then 0
  else n mod 2 + compte (n/2)
```




```
main: li $a0 ,-81
      jal compte
      move $a0, $v0
      li $v0,1
      syscall
      li $v0, 10
      syscall
```

```
compte:
  subi $sp, $sp, 8  # sauvegarde fp et ra
  sw   $fp, 8($sp)
  sw   $ra, 4($sp)
  addi $fp, $sp, 8   # init nouveau fp
  bnez $a0, compte_rec  # cas de base: 0
  li $v0,0
  b compte_fin
```

```
compte_rec: #cas récursif (else)
  rem $t0, $a0, 2   # n mod 2 dans t0
  sw $t0, 0($sp)  # sauvegarde dans la pile
  subi $sp, $sp, 4
  sra $a0, $a0, 1
  jal compte          #appel récursif sur n/2
  addi $sp, $sp, 4 # restauration de t0
  lw $t0, 0($sp)
  add $v0, $t0, $v0 # rés: n mod 2 + résultat appel recursif
```

```
compte_fin: # nettoyage du tableau d'activation
  lw $ra, 4($sp) #restauration ra et fp
  lw $fp, 8($sp)
  addi $sp, $sp, 8
  jr $ra  #fin
```

VERSION SANS FP:

```
compte:
  subi $sp, $sp, 8
  sw   $ra, 4($sp)
  bnez $a0, compte_rec
  li $v0,0
  b compte_fin
```

```
compte_fin:
  lw $ra, 4($sp)
  lw $fp, 8($sp)
  jr $ra
```

2)

```
let rec aux n acc =
  if n = 0
  then acc
  else compte (n/2) (acc + n mod 2)

let compte n = aux n 0
```

↳ ici, pas besoin d'un tableau

```
compte_tr:
  li $a1,0
  j aux
```

```
aux:
  bnez $a0, aux_rec
  move $v0, $a1
  jr $ra
```

```
aux_rec:
  rem $t0, $a0,2
  add $a1, $a1, $t0
  sra $a0, $a0, 1
  j aux
```

3)

```
let rec f91 n =
  if n > 100
  then n - 10
  else f91(f91(n+11))
```

```
f91:
  subi $sp, $sp, 8
  sw $ra, 4($sp)

  li $t0, 100 # cas de base: n>100
  ble $a0, $t0, f91_rec # n≤100 ⇝ else

  subi $v0, $v0, 10 # v0 = n-10
  b f91_fin
```

```
f91_rec:
  addi $a0, $a0, 11 # n = n+11
  jal f91 # f91(n+11)
  move $a0, $v0
  jal f91 # f91(f91(n+11))
```

↑ 3 blocs dans le tableau

Δ ! a0 c'est l'argument pour le 2e appel

```
f91_fin:
  lw $ra, 4($sp)
  addi $sp, $sp, 8
  jr $ra
```

VERSION AVEC FP:

```
f91:
  subi $sp, $sp, 8
  sw $fp, 4($sp)
  sw $ra, 4($sp)
  li $t0, 100
  ble $a0, $t0, f91_rec
  subi $v0, $v0, 10
  b f91_fin
```

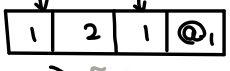
```
f91_fin:
  lw $ra, 4($sp)
  lw $fp, 8($sp)
  addi $sp, $sp, 8
  jr $ra
```

Ex 2 : Some e : valeur optionnelle calculée pare représentée par un pointeur vers un bloc alloué dans le tas
entête : nb elts ds le bloc , un champs par elt du bloc (0 n'est pas un pointeur valide)
None : valeur absente représentée représentée comme l'entier 0
⇝ ?e:d

1) Décrire l'état des registres et du tas après l'exécution du code + valeur dans $v0

```
li   $a0, 8
li   $v0, 9
syscall
li   $t0, 1
sw   $t0, 0($v0)
li   $t0, 2
sw   $t0, 4($v0)
move $t0, $v0
li   $v0, 9
syscall
sw   $t0, 4($v0)
li   $t0, 1
sw   $t0, 0($v0)
```

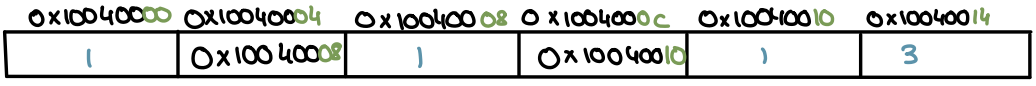
$a0 = 8
$t0 = 1
$v0 = @2 @a1 @a2
le tas =



(2 blocs consécutifs de 8 octets)

⇝ Some(Some 2)

2) Some(Some(3))



3) Some(Some n)

```
lw $v0, 4($a0)
lw $v0, 4($v0)
```

($a0 contient un pointeur vers un bloc B_1, dont le 2e champs contient un pointeur vers un bloc B_2, dont le 2e champs contient la valeur n recherchée. on la stocke temporairement dans $v0 puis on va lire le 2e champs de B_2)

4) f(None) = -1

f(Some n) = n+1

si le paramètre vaut 0, c.à.d. None, on saute à L1 et on charge -1 dans le resultat.

sinon, on va lire le contenu du 1e champs et on l'incrémente de 1. ⇝ L2 dans tous les cas puis on revient à l'appelant

```
beqz $a0, L1        L1:
lw   $v0, 4($a0)      li $v0, -1
addi $v0, $v0, +1   L2:
b    L2               jr $ra
```