TD analyse ascendante

Exercice 1 (Analyse ascendante) On prend la grammaire suivante pour les expressions arithmétiques, et les règles d'analyse ascendante détaillées dans les notes de cours (tableau p.80).

$$
\begin{array}{lcl}
S & \to & E\,\# \\
E & \to & \mathsf{n} \\
 & \mid & E + E \\
 & \mid & E * E \\
 & \mid & (\ E\ )
\end{array}
$$

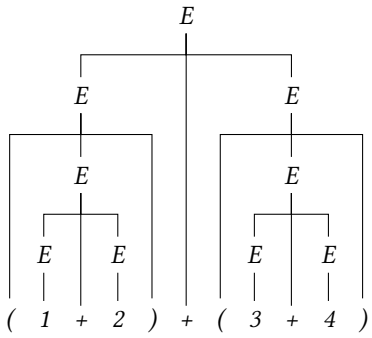
1. Détailler les étapes de l'analyse ascendante de l'entrée (1 + 2) + (3 + 4) et donner l'arbre de dérivation correspondant.
2. Détailler les étapes de l'analyse ascendante de l'entrée (1 + 2) (3). Que se passe-t-il ?

Correction :

1. *On alterne étapes de progression et de réduction jusqu'à consommer toute l'entrée et réduire la pile à une unique expression E : l'analyse a réussi.*

Pile	*Entrée*	*Action*
ε	*(1+2)+(3+4)*	*S*
(	*1+2)+(3+4)*	*S*
(1	*+2)+(3+4)*	*R [E → n]*
(E	*+2)+(3+4)*	*S*
(E+	*2)+(3+4)*	*S*
(E+2	*)+(3+4)*	*R [E → n]*
(E+E	*)+(3+4)*	*R [E → E+E]*
(E	*)+(3+4)*	*S*
(E)	*+(3+4)*	*R [E → (E)]*
E	*+(3+4)*	*S*
E+	*(3+4)*	*S*
E+(	*3+4)*	*S*
E+(3	*+4)*	*R [E → n]*
E+(E	*+4)*	*S*
E+(E+	*4)*	*S*
E+(E+4	*)*	*R [E → n]*
E+(E+E	*)*	*R [E → E+E]*
E+(E	*)*	*S*
E+(E)	*∅*	*R [E → (E)]*
E+E	*∅*	*R [E → E+E]*
E	*∅*	*succès*

L'arbre de dérivation correspond est le suivant.



2. *La reconnaissance de cette nouvelle expression commence similairement à la précédente. En revanche, on arrive au bout d'un moment à une situation où plus aucune règle ne peut s'appliquer : l'analyse échoue, indiquant une phrase mal formée.*

Pile	*Entrée*	*Action*
ε	(1+2)(3)	*S*
(	1+2)(3)	*S*
(1	+2)(3)	*R* $[E \to n]$
(E	+2)(3)	*S*
(E+	2)(3)	*S*
(E+2	)(3)	*R* $[E \to n]$
(E+E	)(3)	*R* $[E \to E+E]$
(E	)(3)	*S*
(E)	(3)	*R* $[E \to (E)]$
E	(3)	*échec*

□

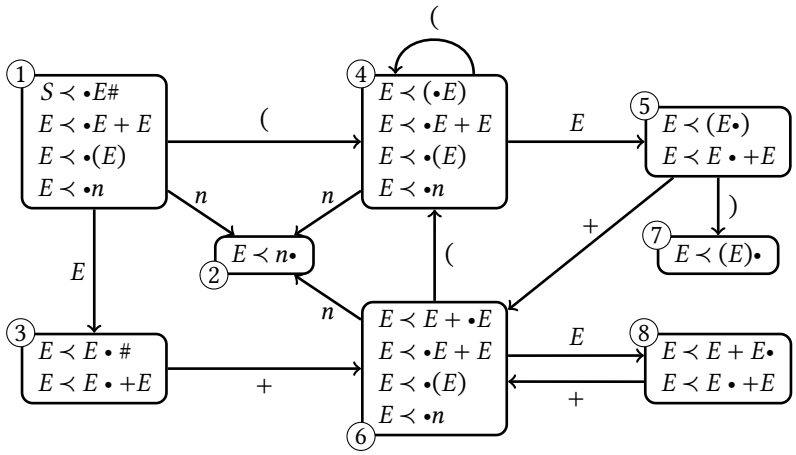
Exercice 2 (Construction d'un automate LR(0)) On considère la grammaire naïve suivante pour un fragment de l'ensemble des expressions arithmétiques.

$$
\begin{array}{lcl}
S & ::= & E\ \# \\
E & ::= & n \\
 & | & E\ +\ E \\
 & | & (\ E\ )
\end{array}
$$

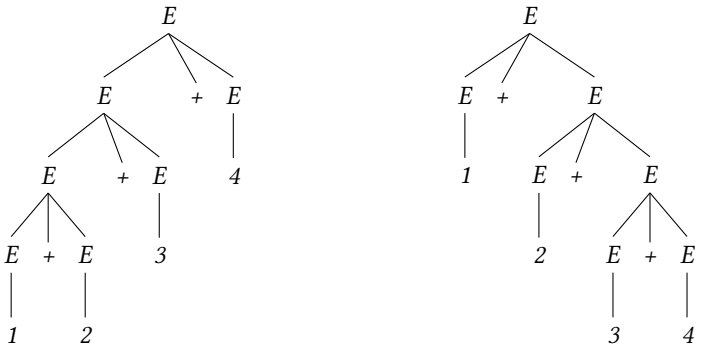
1. Construire l'automate LR(0) associé à cette grammaire.
2. Décrire le conflit obtenu. Est-il lié à une ambiguïté de la grammaire ?
3. Pour chacun des deux choix possibles d'action au niveau du conflit, détailler les étapes de l'analyse ascendante de l'entrée `1 + 2 + 3 + 4` et donner les arbres de dérivation correspondants.

Correction :

1. *Automate LR(0) déterministe.*

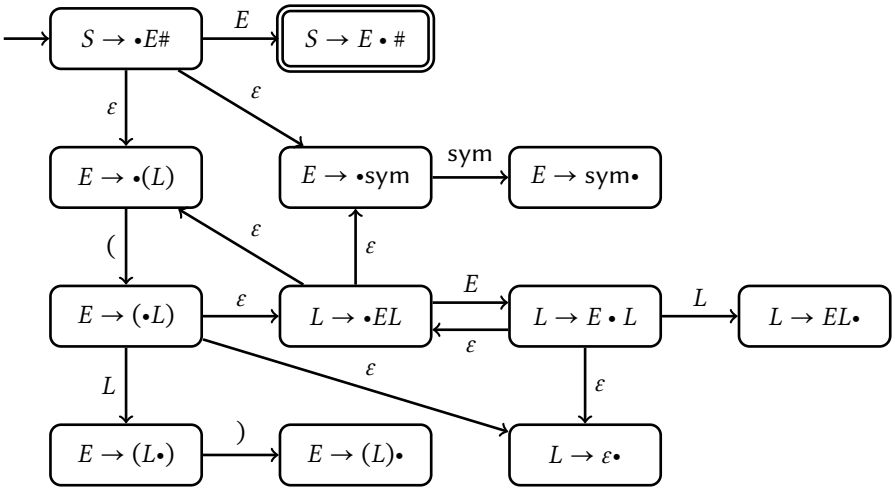


2. *Conflit* shift/reduce *dans l'état* 8 *: on peut réduire la règle* $E \prec E + E$ *ou progresser en lisant un nouveau symbole* $+$*. Ce conflit apparaît par exemple sur l'entrée* $1 + 2 + 3$*, après lecture de* $1 + 2$ *: on a sur la pile* $E + E$ *et il reste à lire* $+3$*. Il correspond à une ambiguïté de la grammaire (associativité de l'addition).*
3. *À gauche : arbre de dérivation obtenu en favorisant la réduction de* $E \prec E + E$*. À droite : arbre obtenu en favorisant la progression sur un nouveau symbole* $+$*.*



□

Exercice 3 (Interprétation d'un automate LR(0)) Voici l'automate LR(0) non déterministe d'une certaine grammaire G.



Questions

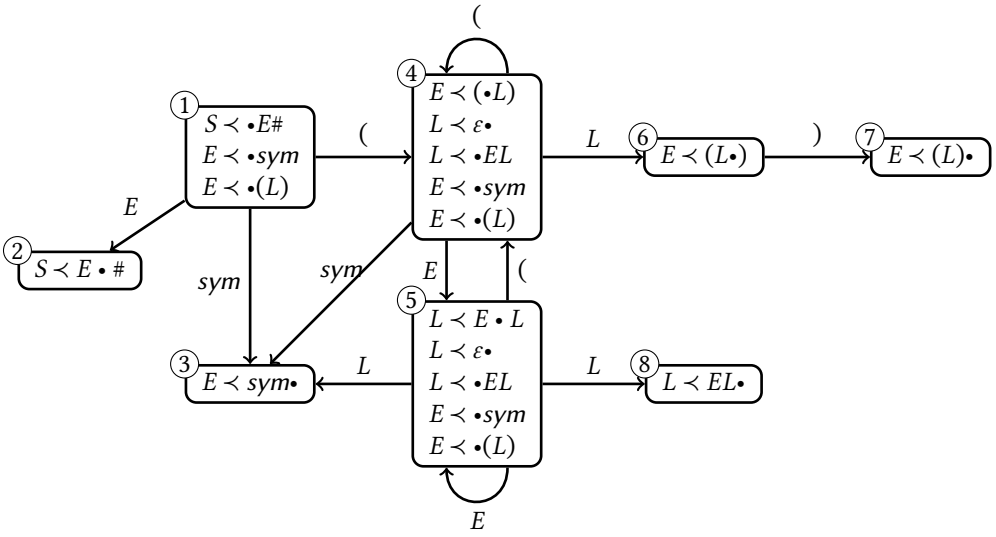
1. Quelle est cette grammaire G?
2. Déterminiser l'automate.
3. Donner les tables d'action et de déplacement associées à cet automate.
4. Décrivez les conflits LR(0) de cette grammaire et des entrées menant à ces conflits.
5. Pouvez-vous retoucher la table d'action pour éliminer les conflits d'une manière satisfaisante ?

Correction :

1. *Grammaire :*

$$
\begin{array}{lll}
S & ::= & E\,\# \\
E & ::= & sym \\
 & | & (\ L\ ) \\
L & ::= & \varepsilon \\
 & | & E\,L
\end{array}
$$

2. *Automate déterminisé*



3. *Tables*

	Actions				***Sauts***	
	sym	(	)	#	*E*	*L*
q_1	q_3	q_4			q_2	
q_2				*ok*		
q_3	$E \prec sym$					
q_4	q_3	q_4			q_5	q_6
	$L \prec \varepsilon$					
q_5	q_3	q_4			q_5	q_8
	$L \prec \varepsilon$					
q_6			q_7			
q_7	$E \prec (L)$					
q_8	$L \prec EL$					

4. *Les états 4 et 5 permettent une réduction de la règle $L \prec \varepsilon$, mais également de progresser avec les symboles sym ou (: on a des conflits* shift/reduce. *Un conflit de l'état 4 apparaît par exemple avec l'entrée* `(s)`, *sur le symbole* `s` *après lecture de la première parenthèse. Un conflit de l'état 5 apparaît par exemple avec l'entrée* `(s())`, *sur la deuxième parenthèse (la pile contient alors* $(E$ *et le prochain symbole est* $)$*).*
5. *La règle $L \prec \varepsilon$ sert à terminer une liste L. Il ne faut l'appliquer que lorsque le prochain symbole est la parenthèse fermante. On obtient alors la table corrigée suivante :*

	Actions				***Sauts***	
	sym	(	)	#	*E*	*L*
q_1	q_3	q_4			q_2	
q_2				*ok*		
q_3	$E \prec sym$					
q_4	q_3	q_4	$L \prec \varepsilon$		q_5	q_6
q_5	q_3	q_4	$L \prec \varepsilon$		q_5	q_8
q_6			q_7			
q_7	$E \prec (L)$					
q_8	$L \prec EL$					

Note : cette grammaire est SLR(1).

□

Exercice 4 (Grammaires et ambiguïtés) On s'intéresse à des grammaires pour des expressions contenant des conditionnelles avec ou sans **`else`**. On considère les symboles terminaux `x` (pour une variable) et **`if`**,

then, **else**. E et E' sont les symboles non terminaux, et E est le symbole de départ.

G_1

$$
\begin{array}{rcl}
E & \to & \texttt{x} \\
 & | & \texttt{if } E \texttt{ then } E \\
 & | & \texttt{if } E \texttt{ then } E \texttt{ else } E
\end{array}
$$

G_2

$$
\begin{array}{rcl}
E & \to & \texttt{x} \\
 & | & \texttt{if } E \texttt{ then } E\,E' \\
E' & \to & \varepsilon \\
 & | & \texttt{else } E
\end{array}
$$

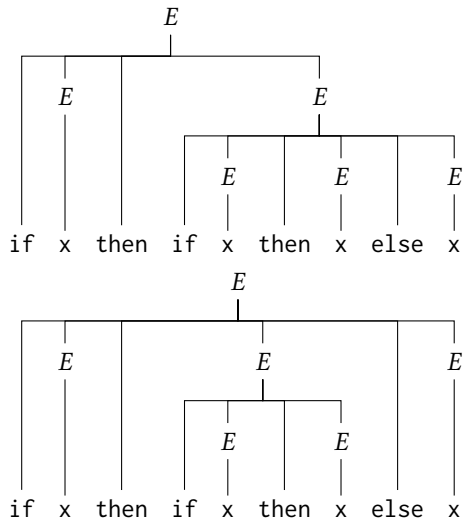
G_3

$$
\begin{array}{rcl}
E & \to & \texttt{if } E \texttt{ then } E \\
 & | & E' \\
E' & \to & \texttt{x} \\
 & | & \texttt{if } E \texttt{ then } E' \texttt{ else } E'
\end{array}
$$

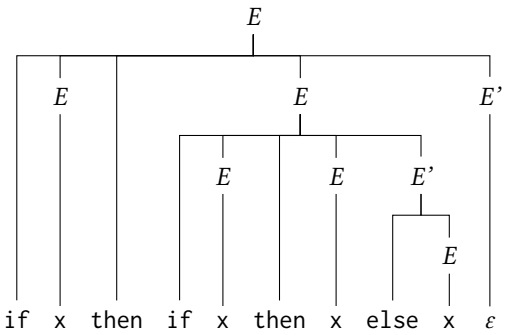
1. Donner deux dérivations distinctes pour la phrase `if x then if x then x else x` dans la grammaire G_1, de manière à déduire que cette grammaire est ambiguë.
2. Montrer que la grammaire G_2 ne corrige pas le problème.
3. Montrer que la grammaire G_3 est incomplète, c'est-à-dire qu'elle ne permet pas de dériver certaines phrases qui sont dérivables par G_1.
4. Proposer une manière de corriger G_3 de sorte à ce qu'elle permette de dériver toute phrase dérivable avec G_1, sans être ambiguë.
5. Ce problème d'ambiguïté des conditionnelles a été écrit en utilisant une syntaxe « à la caml ». Existe-t-il aussi en Python ? En C ou en Java ?

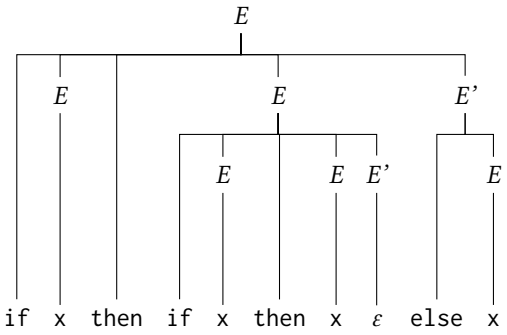
Correction :

1. *Deux arbres de dérivation distincts, qui se distinguent par le* **if** *auquel se rapporte l'unique* **else**.



2. *On a les deux mêmes interprétations possibles, avec le* **else** *se rapportant à l'un ou l'autre des* **if**.





3. *La phrase* `if x then x else if x then x` *n'est pas dérivable dans* G_3. *En effet, la seule règle de* G_3 *permettant de dériver un* `else` *impose que celui-ci soit suivi par une phrase* E', *et la phrase* `if x then x` *n'est pas dérivable à partir de* E'. *Note : cette solution associe nécessairement chaque nouveau* `else` *au* `if` *encore ouvert le plus proche.*
4. *Il suffit d'ajouter la troisième règle* $E \to$ `if` E `then` E' `else` E *pour le symbole non terminal E. Interprétation de ces règles : E désigne une phrase conditionnelle quelconque, et* E' *une phrase conditionnelle qui ne peut plus être complétée par un* `else`.
5. *Ce problème n'existe pas lorsque les blocs d'instructions sont délimités de manière explicite. C'est nécessairement le cas en python puisque les blocs sont définis par l'indentation. En C ou en java, on ne voit pas le problème si l'on utilise systématiquement les accolades pour délimiter les blocs, mais il existe néanmoins car ces langages permettent d'omettre les accolades lorsqu'un bloc ne contient qu'une seule instruction.*

```
if (x) if (x) { ... } else { ... }
```

Note : en ocaml, on peut délimiter explicitement des blocs avec les parenthèses ou avec les mots-clés **`begin`** *et* **`end`**.

□