Corrigé TD 3

Grammaires

Analyse ascendante

1 Grammaires simples

1.1 Grammaire G1

La grammaire est `S → a S | x` que l'on transforme en

```
0   S' → S $
1   S  → a S
2   S  → x
```

On peut calculer les PREMIERs et les SUIVANTs

```
PREMIER(S)  = { a, x }    SUIVANT(S) = { $ }
```

On peut donc construire la table

	a	x	$
S	1	2	

La grammaire est donc bien LL(1).

Construisons maintenant les ensembles d'items pour G1:

```
I0: S' -> . S $       I1
    S  -> . a S       I2
    S  -> . x         I3
```

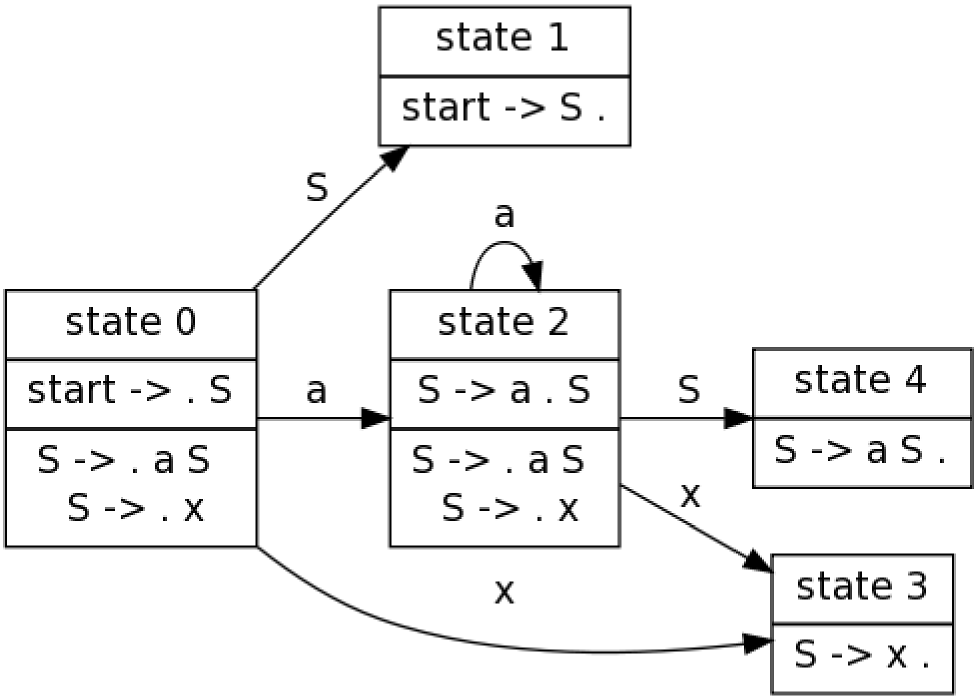
```
I1: S' -> S . $      accept
```

```
I2: S -> a . S        I4
    S -> . a S        I2
    S -> . x          I3
```

```
I3: S -> x .          reduce 2
```

```
I4: S -> a S .        reduce 1
```

Nous avons donc la table de transitions suivante:



La table LR(0) de cette grammaire est donc:

	a	x	$	S
0	s2	s3		1
1			accept	
2	s2	s3		4
3	r2	r2	r2	
4	r1	r1	r1	

Pas de conflit ⇒ la grammaire est LR(0) (et donc aussi SLR(1))

Un exemple d'analyse

Prenons maintenant la table précédente pour analyser la phrase `aax`. On obtient

Pile	Entrée	Commentaire
$0	aax$	shift puis état 2
$0 a2	ax$	shift puis état 2
$0 a2 a2	x$	shift puis état 3
$0 a2 a2 x3	$	reduce suivant règle 2, passage en état 4
$0 a2 a2 S4	$	reduce suivant règle 1, passage en état 4
$0 a2 S4	$	reduce suivant règle 1, passage en état 1
$0 S1	$	**accept**

1.2 Grammaire G2

La grammaire est `S → a S | ε` que l'on transforme en

```
0   S' → S $
1   S  → a S
2   S  → ε
```

On peut calculer les PREMIERs et les SUIVANTs

```
PREMIER(S)  = { a, ε }    SUIVANT(S) = { $ }
```

On peut donc construire la table

	a	$
S	1	2

La grammaire est donc bien LL(1).

Construisons maintenant les ensembles d'items pour G2:

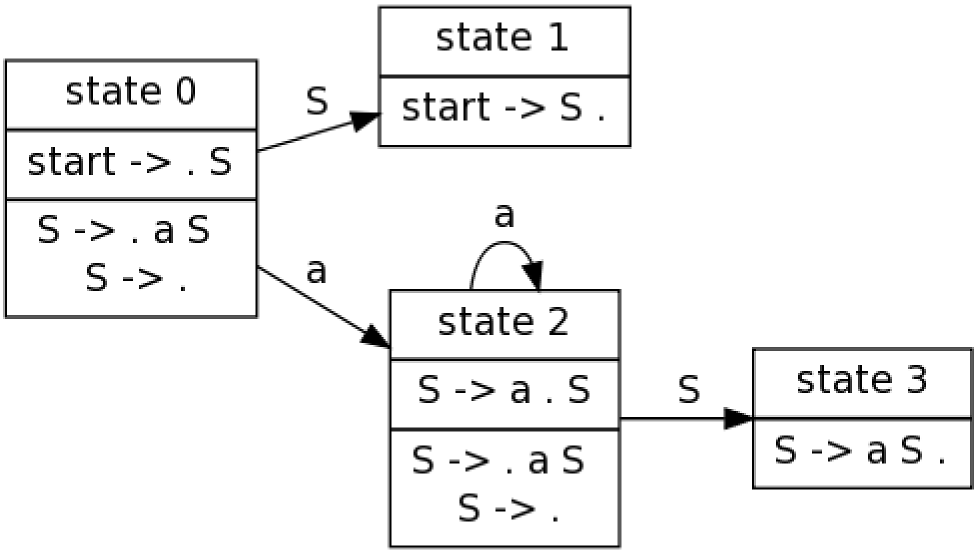
```
I0: S' -> . S $       I1
    S  -> . a S       I2
    S  -> .           reduce 2
```

```
I1: S' -> S . $      accept
```

```
I2: S -> a . S        I3
    S -> . a S        I2
    S -> .            reduce 2
```

```
I3: S -> a S .        reduce 1
```

Nous avons donc la table de transitions suivante:



La table LR(0) de cette grammaire est donc:

	a	$	S
0	**s2** **r2**	r2	1
1		accept	
2	**s2** **r2**	r2	3
3	r1	r1	

Nous avons deux conflits. Par conséquent, G2 n'est pas LR(0).

Voyons maintenant si G2 est SLR(1). Pour cela on construit la table SLR. On rappelle que pour la construction de la table SLR(1), on ne remplit plus systématiquement la table en mettant des réductions pour **tous** les terminaux. En effet, dans ce cas, si *N* est le terminal synthétisé, il ne faut indiquer une réduction que sur les *SUIVANTS(N)*.

Ici, dans l'état 0, quand on réduit suivant la règle r2, on ne le fait que pour *SUIVANTS(S)* (puisque on synthétise S avec la règle `S → ε`). Même chose, pour lorsqu'on réduit suivant la règle r1.

La table obtenue est donc:

	a	$	S

	a	$	S
0	s2	r2	1
1		accept	
2	s2	r2	3
3		r1	

On voit que G2 est donc bien SLR(1).

Finalement, G2 est LL(1) et SLR(1) mais pas LR(0).

2 Grammaire des s-expressions

2.1 Grammaire initiale

On a la grammaire:

```
S → atom | ( M )
M → M S | ε
```

On ajoute la règle

```
0  S' → S $
```

Les ensembles d'items de cette grammaire sont:

```
I0: S' → • S $          I1
    S  → • atom         I2
    S  → • ( M )        I3
```

```
I1: S' → S • $          accept
```

```
I2: S → atom •          reduce 1
```

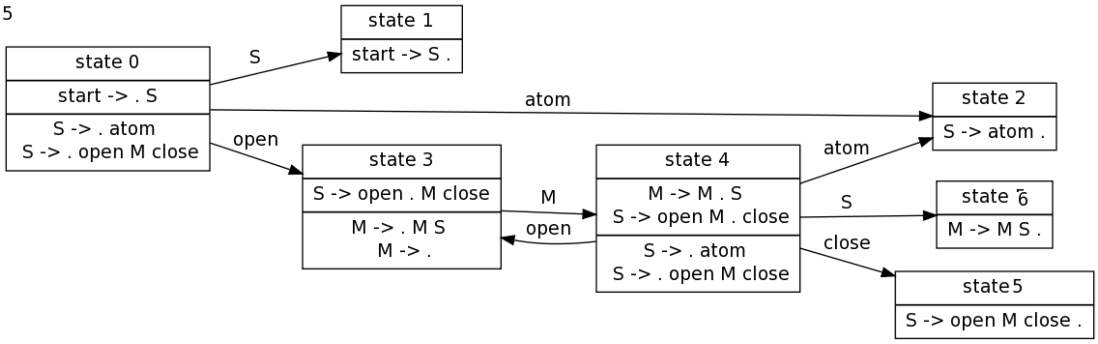
```
I3: S → ( • M )         I4
    M → • M S           I4
    M → •               reduce 4
```

```
I4: S → ( M • )         I5
    M → M • S           I6
    S → • atom          I2
    S → • ( M )         I3
```

```
I5: S → ( M ) •         reduce 2
```

```
I6: M → M S •           reduce 3
```

Ce qui correspond au diagramme de transitions suivant (ici `'('` et `')'` sont dénommés `open` et `close`)



La table LR(0) associée à cette grammaire est donc:

	atom	**(**	**)**	**$**	**S**	**M**
0	s2	s3			1	
1				accept		
2	r1	r1	r1	r1		
3	r4	r4	r4	r4		4
4	s2	s3	s5		6	
5	r2	r2	r2	r2		
6	r3	r3	r3	r3		

Comme il n'y a pas de conflit, cette grammaire est LR(0).

2.2 Grammaire modifiée

Avec la grammaire

```
S → atom | ( M )
M → S M | ε
```

On obtient les ensembles d'items suivants (partiellement)

```
I0: S' → • S $         I1
    S  → • atom        I2
    S  → • ( M )       I3
```

```
I1: S' → S • $         accept
```

```
I2: S → atom •         reduce 1
```

```
I3: S → ( • M )        I4
    M → • S M          I5
    M → •              reduce 4
    S → • atom         I2
    S → • ( M )        I3
```

Il est inutile de continuer plus avant. On voit ici que la troisième règle provoque une réduction alors que les deux suivantes provoquent des décalages. On aura donc la ligne suivante dans la la table LR(0):

	atom	**(**	**)**	**$**	**S**	**M**
….						
3	**r4** **s2**	**r4** **s3**	r4	r4	5	4
…						

La grammaire modifiée n'est donc pas LR(0).

La grammaire modifiée est-elle SLR(1)? Pour voir cette grammaire est SLR(1), on a besoin des PREMIERs et des SUIVANTs. Ici on a:

```
PREMIER(S) = {atom, '('}        SUIVANT(S) = { atom, '(', ')', '$'}
PREMIER(M) = {atom, '(', ε}     SUIVANT(M) = { ')' }
```

Pour construire la table SLR(1), à l'état 3, on ne doit faire la réduction suivant la règle 4 (M → ε) que pour SUIVANT(M), c'est à dire ')'. La table pour cet état devient donc:

	atom	**(**	**)**	**$**	**S**	**M**
….						
3	s2	s3	r4		5	4
…						

On voit donc que ce conflit a a disparu.

Le reste de la construction de la table est laissé en exercice (il y a encore une ligne avec des conflits *shift/reduce* qui se réduisent). Vous devriez trouver que la grammaire modifiée est finalement SLR(1).

3 Affectations

Soit la grammaire **Ga**:

```
1  S  → S ; A
2  S  → A
3  A  → E
4  A  → id := E
5  E  → E + id
6  E  → id
```

On ajoute la règle

```
0  S' → S
```

Les ensembles d'items de cette grammaire sont:

```
I0: S'→ • S                I1
    S → • S ; A            I1
    S → • A                I2
    A → • E                I3
    A → • id := E          I4
    E → • E + id           I3
    E → • id               I4
```

```
I1: S' → S •               accept
    S  → S • ; A           I5
```

```
I2: S  -> A •              reduce 2
```

```
I3: A → E •                reduce 3
    E → E • + id           I6
```

```
I4: A → id • := E          I7
    E → id •               reduce 6
```

```
I5: S → S ; • A              I8
    A → • E                  I3
    A → • id := E            I4
    E → • E + id             I3
    E → • id                 I4
```

```
I6: E → E + • id             I9
```

```
I7: A → id := • E            I10
    E → • E + id             I10
    E → • id                 I11          !! ATTENTION !! Voir note
```

```
I8: S → S ; A •              reduce 1
```

```
I9: E → E + id •             reduce 5
```

```
I10: A → id := E •           reduce 4
     E → E • + id            I6
```

```
I11: E →  id •               reduce 6
```

Note:

Dans I7, sur la règle `E → • id`, nous ne devons pas réutiliser I4. En effet, on voit que dans I0 et dans I5, on peut se trouver à gauche de `id` de **deux** façons:

- soit parce qu'on a la règle `A → • id := E` ou
- soit parce que on a `E → • id`.

Donc on passe dans l'état 4 lorsqu'on essaie de synthétiser un `A` **ou bien** un `E`.

Dans I7, il n'y a qu'**une** seule façon de se retrouver devant un `id` : lorsqu'on essaie de sysnthétiser un `E`. On doit donc créer un nouvel état I11.

Nous avons donc la table de transitions suivante:

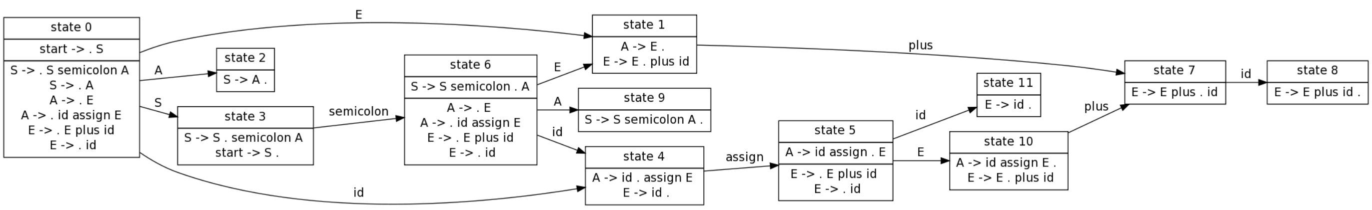


Table LR(0) de Ga

	id	**;**	**+**	**:=**	**$**	**S**	**A**	**E**
0	s4					1	2	3
1		s5			accept			
2	r2	r2	r2	r2	r2			
3	r3	r3	**r3** **s6**	r3	r3			
4	r6	r6	r6	**r6** **s7**	r6			
5	s4						8	3
6	s9							
7	s11							10
8	r1	r1	r1	r1	r1			
9	r5	r5	r5	r5	r5			

	id	;	+	:=	$	S	A	E
10	r4	r4	**r4** **s6**	r4	r4			
11	r6	r6	r6	r6	r6			

On voit dans cette table que l'on a 3 conflits **shift/reduce**. La grammaire n'est donc pas LR(0)

Table SLR(1) de G1

Pour construire la table SLR(1) nous devons chercher les SUIVANTs des non-terminaux de la grammaire. Nous avons:

```
PREMIER(S) =  { id }        SUIVANT(S) = { ';', '$' }
PREMIER(A) =  { id }        SUIVANT(A) = { ';', '$' }
PREMIER(E) =  { id }        SUIVANT(E) = { ';', '+', '$'}
```

Pour construire la table, seules les lignes où il y des réductions sont modifiées. On obtient:

	id	;	+	:=	$	S	A	E
0	s4					1	2	3
1		s5			accept			
2		r2			r2			
3		r3	s6		r3			
4		r6	r6	s7	r6			
5	s4						8	3
6	s9							
7	s11							10
8		r1			r1			
9		r5	r5		r5			
10		r4	s6		r4			
11		r6	r6		r6			

Ga est donc SLR(1) alors qu'elle n'est pas LR(0).
Cela veut dire qu'elle est analysable (sans modification) avec un analyseur SLR(1),

Note:

> On voit bien dans cette table que, dans la configuration I7, il ne fallait pas confondre les états 4 et 11 (cf note précédente).
> En effet, ces deux états ne sont pas équivalents:
>
> - dans l'état 4, si on a le symbole `:=`, c'est qu'on était en train d'analyser un `A` et donc l'action que l'on fait alors est: `shift 7`. Par contre, si on a `;`, `+` ou `$`, c'est qu'on était dans l'analyse d'un `E`.
> - dans l'état 11, si on a un `:=`, nous déclencherons une erreur (et toujours réduction si on rencontre `;`, `+` ou `$`).