LOG3210
Cours 4

Parseurs LR (suite)

Retour sur la table d'items

- Comment déterminer le symbole associé à un état?

- Quel est le symbole associé à l'état 0 (l'état initial)?

- Un symbole peut-il être associé à plus d'un état?

- S'il y a une transition entre un état *i* et *j*, qu'est-ce que cela implique par rapport aux GOTO?

Algorithme de parsage LR

- Le programme de parsage LR (l'algorithme) est le même pour tous les parseurs LR
- C'est la table de parsage qui change

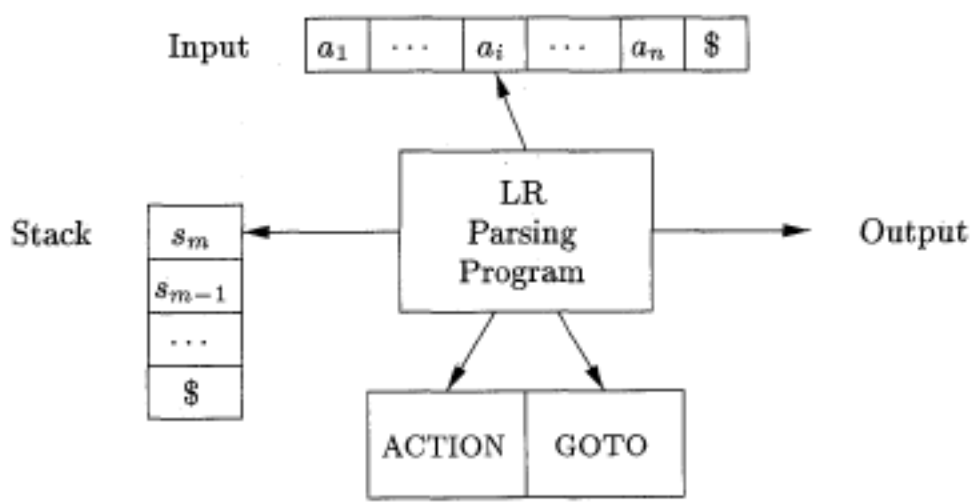

Figure 4.35: Model of an LR parser

Parseur LR vs *shift-reduce*

- Un parseur *shift-reduce* décale les symboles d'entrée sur la pile.

- Un parseur LR décale des états.
  - Parseur SLR: les états viennent de l'automate LR(0)
  - Chaque état a un symbole qui lui est associé

Structure de la table de parsage LR

- Deux parties: ACTION et GOTO

- Fonction d'action de parsage ACTION
  - Arguments
    - État *i*
    - Terminal *a* (ou $, la fin de l'entrée)
  - On effectue l'action située à ACTION[*i*, *a*]

- Fonction GOTO
  - On l'étend à des états
  - Si GOTO[I_i, *A*] = I_j , alors GOTO associe un état *i* et un non terminal *A* à l'état *j*

Fonction ACTION

- La valeur de ACTION[*l*, *a*] peut avoir quatre formes:
  - **Décaler l'état *j* (*shift*)**
    - Met l'état *j* sur la pile, pour représenter que le symbole *a* est sur la pile
  - **Réduire *A* → β**
    - Remplacer β du dessus de la pile par *A*
  - **Accepter**
    - Accepter l'entrée et terminer l'analyse syntaxique
  - **Erreur**
    - Détection d'une erreur syntaxique et mise en marche de routine de correction d'erreurs

Configuration d'un parseur LR

- La configuration d'un parseur LR est une paire (pile, entrée): $(s_0 s_1 \cdots s_m,\ a_i a_{i+1} \cdots a_n \$)$
- **Configuration initiale:**
  - $(<S_0>, <a_1,a_2,\ldots,a_n,\$>)$
- **Configuration intermédiaire:**
  - $(<S_0,S_1,S_2,\ldots, S_m>, <a_i,a_{i+1},\ldots,a_n,\$>)$
- **Configuration finale:**
  - $(<S_0,S_{acc}>, <\$>)$
- Chaque état, sauf S_0, est relié à un symbole de la grammaire

Changement de configuration d'un parseur LR

- Les changements de configuration sont déterminés par:
  - Le symbole courant a_i
  - l'état sur le dessus de la pile s_m
  - l'action ACTION[s_m, a_i] de la table de parsage

- Si ACTION[s_m, a_i] = *shift* et GOTO[s_m, a_i] = s , alors la configuration résultante est:
  - $(<S_0,S_1,S_2,\ldots,S_m,s>, <a_{i+1},a_{i+2},\ldots,a_n,\$>)$
  - Le symbole a_i n'est pas sur la pile, mais il peut être récupéré via l'état s, si nécessaire
  - Le symbole courant est maintenant a_{i+1}

Changement de configuration d'un parseur LR

- Si ACTION[s_m, a_i] = réduire par $A \rightarrow \beta$, $|\beta| = r$ et GOTO[s_{m-r}, A] = s alors la configuration résultante est:
  - ($<S_0, S_1, S_2, \ldots, S_{m-r}, s>$, $<a_i, a_{i+1}, \ldots, a_n, \$>$)
  - Le symbole courant est encore a_i
  - **NOTE:** les symboles associés à $S_{m-r+1} \ldots S_m$ correspondent à β
  - **NOTE2:** on a ôté r états de la pile, puis ajouté GOTO[S_{m-r}, A] sur la pile

- Si ACTION[s_m, a_i] = acceptation, le parsage est complété

- Si ACTION[s_m, a_i] = erreur, on lance une routine de correction d'erreur

Changement de configuration d’un parseur LR

- Notes:
  - Pour générer la sortie d’un parseur, des actions sémantiques peuvent être associées aux décalages et aux réductions
  - Nous y reviendrons au prochain cours (Traduction dirigée par la syntaxe)
  - Pour l’instant, on peut assumer que chaque réduction est associée à une action qui imprime la production utilisée.

Parseur LR
Algorithme

```
let a be the first symbol of w$;
while(1) { /* repeat forever */
        let s be the state on top of the stack;
        if ( ACTION[s, a] = shift t ) {
                push t onto the stack;
                let a be the next input symbol;
        } else if ( ACTION[s, a] = reduce A → β ) {
                pop |β| symbols off the stack;
                let state t now be on top of the stack;
                push GOTO[t, A] onto the stack;
                output the production A → β;
        } else if ( ACTION[s, a] = accept ) break; /* parsing is done */
        else call error-recovery routine;
}
```

Figure 4.36: LR-parsing program

Table de parsage LR
Représentation

- Dans le cadre du cours, on utilisera les codes d'actions suivants dans les tables de parsage
  - ***si*** : décaler l'entrée et empile l'**état** *i*
  - ***rj*** : réduire par la **production** numérotée *j*
  - **acc** : accepter
  - **Rien (*blank*)** : erreur

Table de parsage LR
Représentation

(1) $E \to E + T$
(2) $E \to T$
(3) $T \to T * F$
(4) $T \to F$
(5) $F \to (E)$
(6) $F \to \mathbf{id}$

STATE	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Figure 4.37: Parsing table for expression grammar

Table de parsage LR
Exemple d'utilisation

- Utilisons la table de la diapositive précédente avec l'entrée: **id * id + id**

- Tableau de base:

Étape	Pile	Symboles	Entrée	Action
(1)	0	$	**id * id + id $**	
(2)				
(3)				
(4)				
(5)				
(6)				
…				

Construction de la table SLR

- La construction de la table SLR nécessite:
  - La grammaire augmentée *G'*
  - Les items LR(0) (l'ensemble canonique)
  - L'automate LR(0)
  - Les FOLLOW de chaque non terminal de la grammaire

Construction de la table SLR Algorithme

1. Contruire C = {I_0, I_1, …, I_n}, l'ensemble d'ensemble d'items LR(0) pour *G'*

2. L'état *i* est construit à partir de I_i. Les actions sont déterminées comme suit:
   A. Si [$A \rightarrow \alpha \cdot a\beta$] est dans I_i et GOTO(I_i ,a) = I_j, alors mettre « *shift j* » dans ACTION[*i, a*]. **Ici, *a* doit être un terminal.**
   B. Si [$A \rightarrow \alpha \cdot$] est dans I_i, ***A* ≠ S'**, alors mettre « *reduce* $A \rightarrow \alpha$ » dans ACTION[*i, a*] pour tous les *a* dans FOLLOW(*A*).
   C. Si [$S' \rightarrow S\cdot$] est dans I_i, alors mettre « *accept* » dans ACTION[*i*, $].
   D. Si les étapes ci-haut résultent en un conflit d'actions, la grammaire n'est pas SLR(1). L'algorithme échoue alors à produire un parseur.

Construction de la table SLR
Algorithme

3. Les transitions *GOTO* pour l'état *i* sont construites pour tous les non terminaux *A* en utilisant la règle:
   - Si GOTO(I_i,*A*) = I_j, alors GOTO[*i*,*A*] = *j*

4. Toutes les entrées non définies par les règles (2) et (3) sont des erreurs

5. L'état initial du parseur est celui construit à partir de l'ensemble d'items contenant [$S' \rightarrow \cdot S$]

Table SLR

- La table construite est la table SLR(1) pour *G*.
- Un parseur LR qui utilise la table SLR(1) pour *G* est appelé parseur SLR(1) pour *G*.
- Une grammaire qui a une table de parsage SLR(1) est dite SLR(1).
- On omet habituellement le « (1) » après SLR

Construction de la table SLR
Exemple

- Soit les règles suivantes et l'ensemble d'ensemble canonique LR(0) suivant, construire la table de parsage SLR.

(1)	$E \to E + T$	(4)	$T \to F$
(2)	$E \to T$	(5)	$F \to (E)$
(3)	$T \to T * F$	(6)	$F \to \mathbf{id}$

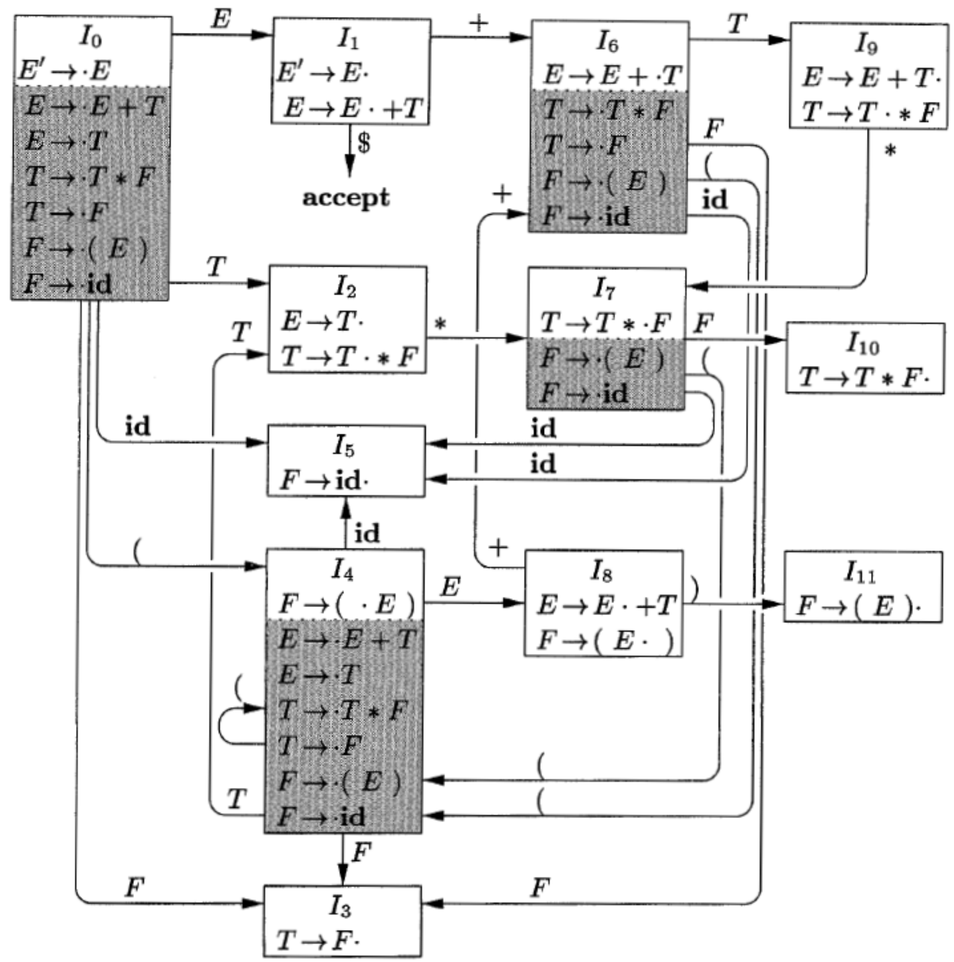

Figure 4.31: LR(0) automaton for the expression grammar (4.1)

Grammaires SLR(1)
Ambiguïté

- Toutes les grammaires SLR(1) sont non-ambiguës

- Il y a des grammaires non-ambiguës qui ne sont pas SLR(1)

- Une grammaire n'est pas SLR(1) si deux actions se retrouvent dans la même case de la table SLR

- Démontrez que la grammaire suivante n'est pas SLR(1) sur l'ensemble d'items fourni:

$$\begin{aligned} S &\rightarrow L = R \mid R \\ L &\rightarrow *R \mid \mathbf{id} \\ R &\rightarrow L \end{aligned}$$

$$I_2: \quad \begin{aligned} &S \rightarrow L \cdot = R \\ &R \rightarrow L \cdot \end{aligned}$$

Exercice 1

- Soit la grammaire S → *S S* + | *S S* * | *a*
  1. Si ce n'est déjà fait, construire l'ensemble d'items SLR pour la grammaire augmentée et calculer la fonction GOTO pour chacun des ensembles (fait au dernier cours)
  2. Faire la table de parsage SLR
  3. Faire la table des actions (avec l'état de la pile, des symboles et de l'entrée) pour l'entrée suivante: *aa*a*+

Exercice 1 (réponse)

- $I_0 = \{ S' \rightarrow \cdot S ,\ S \rightarrow \cdot SS+ ,\ S \rightarrow \cdot SS* ,\ S \rightarrow \cdot a\}$
- $I_1 = GOTO(I_0, S) =$
  $\{ S' \rightarrow S\cdot ,\ S \rightarrow S\cdot S+ ,\ S \rightarrow S\cdot S* ,\ S \rightarrow \cdot SS+ ,\ S \rightarrow \cdot SS* ,$
  $S \rightarrow \cdot a\}$
- $I_2 = GOTO(I_0, a) = \{ S \rightarrow a\cdot \}$
- $I_3 = GOTO(I_1, S) =$
  $\{ S \rightarrow SS\cdot + ,\ S \rightarrow SS\cdot * ,\ S \rightarrow S\cdot S+ ,\ S \rightarrow S\cdot S* ,\ S \rightarrow \cdot SS+ ,$
  $S \rightarrow \cdot SS*, S \rightarrow \cdot a \}$
- $I_4 = GOTO(I_3, +) = \{ S \rightarrow SS+\cdot \}$
- $I_5 = GOTO(I_3, *) = \{ S \rightarrow SS*\cdot \}$
- $GOTO(I_3, S) = GOTO(I_1, S) = I_3$
- $GOTO(I_1, a) = GOTO(I_3, a) = GOTO(I_0, a) = I_2$

Exercice 1 (réponse)

- FOLLOW(S) = { a ,+, *, $ }

State	ACTION				GOTO
	+	*	a	$	S
0			s2		1
1			s2	acc	3
2	r3	r3	r3	r3	
3	s4	s5	s2		3
4	r1	r1	r1	r1	
5	r2	r2	r2	r2	

Exercice 1 (réponse)

Étape	Pile	Symboles	Entrée	Action
(1)	0		aa*a+$	s2
(2)	0 2	a	a*a+$	r3
(3)	0 1	S	a*a+$	s2
(4)	0 1 2	Sa	*a+$	r3
(5)	0 1 3	SS	*a+$	s5
(6)	0 1 3 5	SS*	a+$	r2
(7)	0 1	S	a+$	s2
(8)	0 1 2	Sa	+$	r3
(9)	0 1 3	SS	+$	s4
(10)	0 1 3 4	SS+	$	r1
(11)	0 1	S	$	accept

Parseurs LR avancés

- Les items calculés jusqu'à maintenant étaient des items LR(0) : ils ne tiennent pas compte du symbole suivant dans l'entrée.

- Dans certaines situations, il est possible qu'une réduction $A \rightarrow \alpha$ soit invalide selon le symbole suivant dans l'entrée.

- Voir exemple 4.51 du livre

$$\begin{array}{lcl} S & \rightarrow & L = R \mid R \\ L & \rightarrow & *R \mid \mathbf{id} \\ R & \rightarrow & L \end{array} \qquad I_2: \quad \begin{array}{l} S \rightarrow L \cdot = R \\ R \rightarrow L \cdot \end{array}$$

- Comment régler ce problème?

Parseurs LR avancés

- On peut séparer les états en plusieurs états afin de traiter les réductions invalides.
- On redéfinit les items pour qu'ils contiennent également un symbole terminal (ou $) qui doit être le symbole d'entrée suivant
  - $[A \rightarrow \alpha \cdot \beta, a]$
- Un tel item est un item LR(1)
- Le chiffre 1 représente la longueur du deuxième paramètre. On l'appelle le *lookahead* de l'item.

Items LR(1) Exemple

- Construisons l'ensemble d'items LR(1) pour la grammaire suivante:

$$
\begin{array}{rcl}
S' & \rightarrow & S \\
S & \rightarrow & C\ C \\
C & \rightarrow & c\ C \ \mid \ d
\end{array}
$$

Items LR(1) Exemple

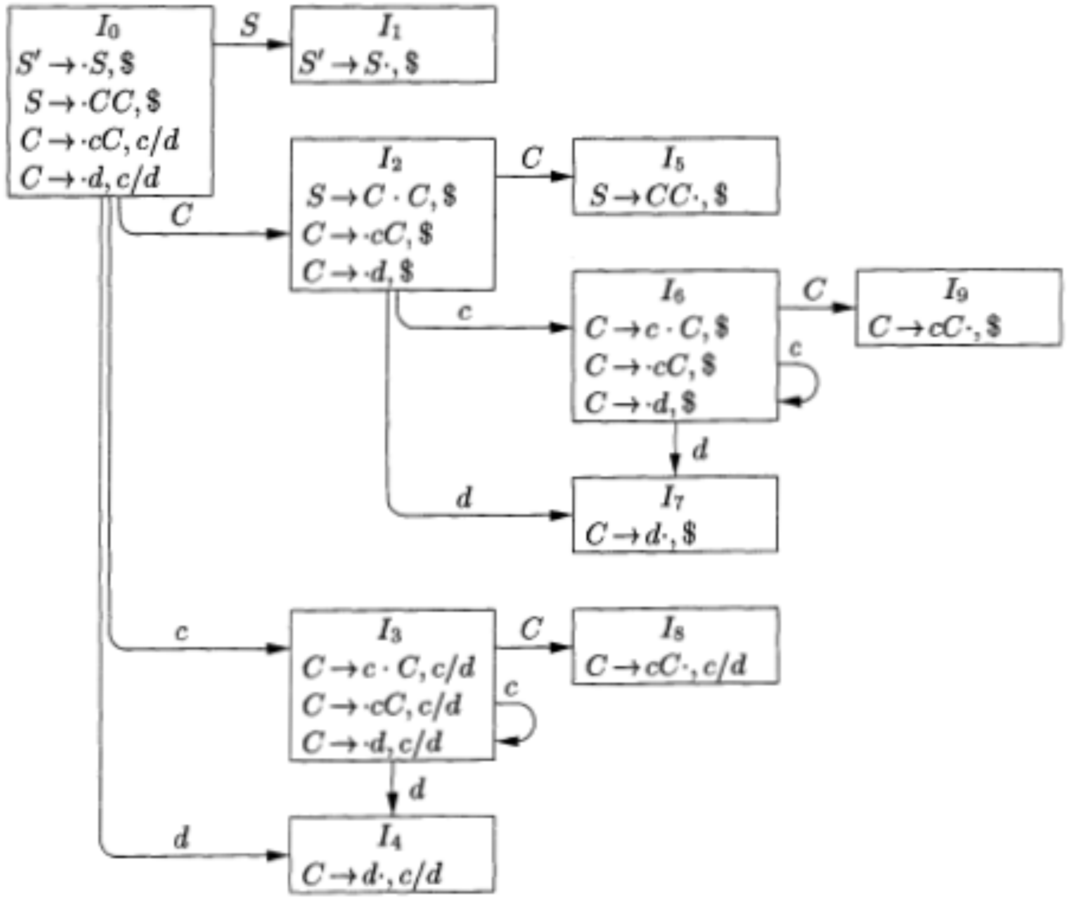

Figure 4.41: The GOTO graph for grammar (4.55)

Table de parsage canonique LR(1)

- Construction des ensembles d'items LR(1)
- Algorithme de construction de la table de parsage canonique LR(1)

- Section 4.7 du livre

Table de parsage canonique LR(1)

- La table construite est la table canonique LR(1) pour *G*.
- Un parseur LR qui utilise cette table est un parseur canonique LR(1)
- Une grammaire qui a une table de parsage canonique LR(1) est dite grammaire LR(1).
- On omet habituellement le « (1) » après LR
- Toutes les grammaires SLR(1) sont LR(1)
  - Pour une même grammaire, le parseur LR a généralement beaucoup plus d'états que le parseur SLR
  - Par exemple, l'exemple précédent n'aurait que 7 états en SLR

Construction de la table LR(1) Exemple

- Construisons la table canonique LR pour la grammaire suivante:

$$
\begin{array}{lcl}
S' & \rightarrow & S \\
S & \rightarrow & C\ C \\
C & \rightarrow & c\ C \mid d
\end{array}
$$

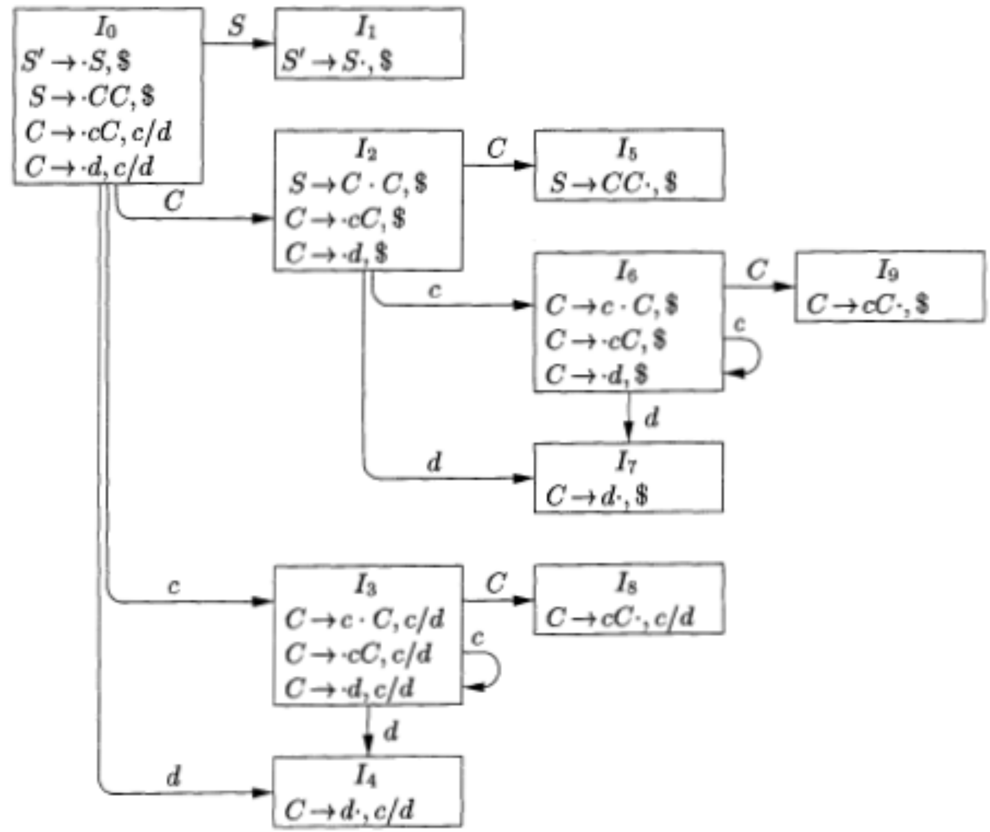

Figure 4.41: The GOTO graph for grammar (4.55)

Construction de la table LR(1) Exemple

STATE	ACTION			GOTO	
	c	d	$\$$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

Figure 4.42: Canonical parsing table for grammar (4.55)

Tables LALR

- Un troisième type de tables existe pour les parseurs LR: les tables LALR (*lookahead-LR*)
- LALR est souvent utilisé en pratique, car les tables obtenues sont plus petites et permettent de couvrir la plupart des structures retrouvées en programmation.
- Elles sont plus puissantes que SLR (mais moins que LR) et couvrent certaines structures que SLR ne supporte pas.
- Pas matière au cours, mais documentées dans la section 4.7.4

Exercices supplémentaires

- Soit la grammaire $S \rightarrow S\ S + \mid S\ S * \mid a$
  1. Construire l'ensemble d'items LR(1) pour la grammaire augmentée et calculer la fonction GOTO pour chacun des ensembles
  2. Construire la table de parsage canonique LR

1. Montrez que la grammaire suivante est LL(1), mais pas SLR(1)

$$\begin{array}{lcl} S & \rightarrow & A\,a\,A\,b \mid B\,b\,B\,a \\ A & \rightarrow & \epsilon \\ B & \rightarrow & \epsilon \end{array}$$

2. Montrez que la grammaire suivante est SLR(1), mais pas LL(1)

$$\begin{array}{lcl} S & \rightarrow & S\,A \mid A \\ A & \rightarrow & a \end{array}$$

Hiérarchie de grammaires

- LL(1) $\subset$ LR(1)

- SLR(1) $\subset$ LALR(1) $\subset$ LR(1)

- LL(1) $\not\subset$ SLR(1)
  SLR(1) $\not\subset$ LL(1)