Analyse syntaxique ascendante

Cyril Rabat
cyril.rabat@univ-reims.fr

Licence 3 Informatique - Info0602 - Langages et compilation

2021-2022

UFR SEN
Sciences Exactes et Naturelles

Université de Reims Champagne-Ardenne

Cours n°4

Analyseurs SLR, LR(1) et LALR(1)

Version 22 février 2022

Table des matières

L'analyse syntaxique ascendante

- Connue également sous le nom d'**analyse par décalage-réduction**
- But : construire un arbre syntaxique en partant des feuilles vers la racine
- Principe : réduction de la chaîne vers l'axiome de la grammaire
- À chaque étape :
  - Réduction d'une sous-chaîne correspondant à la partie droite d'une règle
  - Remplacée par le symbole de la partie gauche
- Plusieurs méthodes existent :
  - La méthode d'analyse par précédence d'opérateurs
  - La méthode LR, plus générale
    $\hookrightarrow$ Utilisée dans de nombreuses constructions d'analyseurs

Exemple d'analyse ascendante

Soit la grammaire suivante :

$$\begin{array}{lcl} S & \rightarrow & \text{a A B e} \\ A & \rightarrow & \text{A b c} \mid \text{b} \\ B & \rightarrow & \text{d} \end{array}$$

Analysons le mot *abbcde* :

- $\text{a}\underline{\text{b}}\text{bcde}$ avec $A \rightarrow \text{b}$
- $\text{a}\underline{\text{Abc}}\text{de}$ avec $A \rightarrow \text{A b c}$
- $\text{aA}\underline{\text{d}}\text{e}$ avec $B \rightarrow \text{d}$
- $\underline{\text{aABe}}$ avec $S \rightarrow \text{a A B e}$
- S

Nous obtenons la dérivation droite suivante :

$$\text{S} \Rightarrow \text{aABe} \Rightarrow \text{aAde} \Rightarrow \text{aAbcde} \Rightarrow \text{abbcde}$$

Implémentation à l'aide d'une pile

- Deux problèmes à résoudre :
  1. Repérer la sous-chaîne à réduire
  2. Choisir la règle si plusieurs sont possibles
- Utilisation d'une pile pour conserver les symboles grammaticaux
- Augmentation de la grammaire $G = (T, N, R, S)$:
  $\hookrightarrow G' = (T \cup \{\dashv\}, N \cup \{S'\}, R \cup \{S' \rightarrow S \dashv\}, S')$
- **Décalage** : avancée dans la chaîne de caractères
- **Réduction** : si α est au sommet, il peut être remplacé par A si $A \rightarrow \alpha \in R$
- **Acceptation** : la pile contient $\dashv S$ et l'entrée $\dashv$
- **Erreur** : une erreur de syntaxe est détectée ; récupération sur erreur

Exemple d'analyse avec une pile

Soit la grammaire :
$$\begin{array}{lcl} E & \rightarrow & E + E \\ E & \rightarrow & E * E \\ E & \rightarrow & (E) \\ E & \rightarrow & \text{id} \end{array}$$

Analyse du mot $id_1 + id_2 * id_3$:

Pile	**Entrée**	**Action**
$\dashv$	$id_1 + id_2 * id_3 \dashv$	décaler
$\dashv id_1$	$+id_2 * id_3 \dashv$	réduire par $E \rightarrow id$
$\dashv E$	$+id_2 * id_3 \dashv$	décaler
$\dashv E+$	$id_2 * id_3 \dashv$	décaler
$\dashv E + id_2$	$*id_3 \dashv$	réduire par $E \rightarrow id$
$\dashv E + E$	$*id_3 \dashv$	décaler
$\dashv E + E*$	$id_3 \dashv$	décaler
$\dashv E + E * id_3$	$\dashv$	réduire par $E \rightarrow id$
$\dashv E + E * E$	$\dashv$	réduire par $E \rightarrow E * E$
$\dashv E + E$	$\dashv$	réduire par $E \rightarrow E + E$
$\dashv E$	$\dashv$	accepter

Conflits décalage/réduction

- Conflits rencontrés pour certaines grammaires non contextuelles
- Dans une configuration donnée, choix à faire :
  - Entre décalage et réduction
  - Entre plusieurs réductions
- Possible de trouver des solutions :
  - Faire un choix de décalage par défaut
  - Utilisation d'une table des symboles

Analyse LR (1/2)

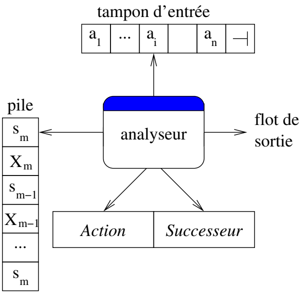
- Utilisable pour une large "gamme" de grammaires non contextuelles
- Notée LR(k) :
  - L pour *Left to right scanning of the input*
    $\hookrightarrow$ Parcours de l'entrée de gauche à droite
  - R pour *Rightmost derivation in reverse*
    $\hookrightarrow$ En construisant une dérivation droite inverse
  - k le nombre de symboles de pré-vision utilisés pour les décisions
- LR(1) est également notée LR
  $\hookrightarrow$ Utilisée pour la plupart des langages en compilation
- La classe des grammaires traitées par des analyseurs prédictifs est inclue dans la classe des grammaires qui peuvent être analysées avec l'analyse LR

Analyse LR (2/2)

- Méthode la plus connue
- Cependant, trop gourmande à réaliser manuellement
- Nécessite des outils spécialisés
  $\hookrightarrow$ Exemple : *Yacc*
- Grammaire passée en entrée et génération d'un analyseur :
  $\hookrightarrow$ Ambiguïtés et conflits détectés
  $\hookrightarrow$ L'utilisateur peut ensuite corriger

Analyseur LR (1/2)

- Un analyseur LR est composé de :
  - Un tampon d'entrée
  - Une pile
  - Une table d'analyse composée de 2 parties : **actions** et **successeurs**
  - Un flot de sortie
- L'analyseur lit les unités lexicales les unes après les autres dans le tampon d'entrée



Analyseur LR (2/2)

- La pile contient des chaînes $s_0 X_1 s_1 X_2 s_2 \ldots X_n s_n$ où
  - s_n est le sommet
  - X_i est un symbole de la grammaire ($X_i \in T \cup N$)
  - s_i est un symbole nommé **état**
- En fonction de l'état au sommet de la pile, du symbole d'entrée, la table permet de déterminer l'action à réaliser et le prochain état
- 4 actions différentes :
  - Décaler s où s est un état, noté d, s
  - Réduire par une règle
  - Accepter
  - Erreur

Techniques de construction de la table

- Plusieurs techniques de construction de la table d'analyse :
  - SLR pour *simple LR* : facile à implémenter, mais la moins "puissante"
    $\hookrightarrow$ La technique de construction peut échouer pour certaines grammaires
  - LR (canonique) la plus "puissante" mais la plus coûteuse
    $\hookrightarrow$ Elle produit de nombreux états
  - LALR pour *LookAhead LR* : utilisable pour un grand nombre de grammaires de langages de programmation
    $\hookrightarrow$ Elle vise à réduire le nombre d'états

Fonctionnement de la table

- Prend comme arguments un état et un symbole non terminal et retourne le prochain état
- Avec :
  - La pile contient $s_0X_1s_1X_2s_2\dots X_ms_m$
  - L'entrée contient $a_ia_{i+1}\dots a_n \dashv$
- Si $Action[s_m, a_i] =$ décaler s
  - La pile devient $s_0X_1s_1X_2s_2\dots X_ms_ma_is$
  - L'entrée devient $a_{i+1}\dots a_n \dashv$
- Si $Action[s_m, a_i] =$ réduire par $A \rightarrow \beta$, avec $Successeur[s_{m-r}, A] = s$, r est la longueur de β
  - La pile devient $s_0X_1s_1X_2s_2\dots X_{m-r}s_{m-r}As$
  - L'entrée reste $a_i\dots a_n \dashv$

  $\hookrightarrow$ $2 \times r$ symboles dépilés, A et $Successeur[s_{m-r}, A] = s$ empilés
- Si $Action[s_m, a_i] =$ accepter, l'analyse est terminée
- Si $Action[s_m, a_i] =$ erreur, une erreur est détectée ; l'analyse s'arrête

Exemple : la grammaire et la table d'analyse

Soit la grammaire :

$$
\begin{array}{lcl}
P & \rightarrow & \text{a P b } \textit{(règle n°1)} \\
 & \rightarrow & \text{a b } \textit{(règle n°2)}
\end{array}
$$

Nous obtenons la table suivante :

États	*Action*			*Successeur*
	a	b	$\dashv$	P
0	d,2			1
1			acc.	
2	d,2	d,4		3
3		d,5		
4		r,2	r,2	
5		r,1	r,1	

Exemple : exécution

Pile	Entrée	Action
0	aaabbb ⊣	d,2
0a2	aabbb ⊣	d,2
0a2a2	abbb ⊣	d,2
0a2a2a2	bbb ⊣	d,4
0a2a2a2b4	bb ⊣	r,2
0a2a2P3	bb ⊣	d,5
0a2a2P3b5	b ⊣	r,1
0a2P3	b ⊣	d,5
0a2P3b5	⊣	r,1
0P1	⊣	acc

Grammaire LR(k)

Définition : grammaire LR(k)

Une grammaire est dire LR(k) s'il est possible de construire la table d'analyse syntaxique. k désigne le nombre de symboles à utiliser pour prendre une décision. En pratique, $k = 0$ ou $k = 1$.

Définition : item LR(0)

Un item LR(0) est une règle de la grammaire avec un point (.) dans la partie droite. Le point représente la position de la tête de lecture (fenêtre) dans l'analyse

- Exemple : $A \rightarrow XYZ$ donne les 4 items $A \rightarrow .XYZ$, $A \rightarrow X.YZ$, $A \rightarrow XY.Z$ et $A \rightarrow XYZ.$

Fermeture d'un ensemble d'items LR(0)

Définition : fermeture d'un ensemble d'items LR(0)

La fermeture de I (notée Fermeture(I)), un ensemble d'items LR(0), est définie par :

- $I \subseteq$ *Fermeture(I)*
- *Si* $A \rightarrow \alpha.B\beta \in$ *Fermeture(I) et* $B \rightarrow \gamma \in R$*,*
  alors $B \rightarrow .\gamma \in$ *Fermeture(I)*

La dernière règle est appliquée jusqu'à ce qu'aucun item ne puisse plus être ajouté.

Exemple de fermeture d'un ensemble d'items LR(0)

Soit la grammaire des expressions augmentée :

E' → E
E → E + T | T
T → T * F | F
F → id | (E)

$Fermeture(\{E' \rightarrow .E\})$ contient :

- $E' \rightarrow .E$ car $I \subseteq$ *Fermeture(I)*
- $E \rightarrow .E + T$ et $E \rightarrow .T$: ajout des règles avec E à gauche
- $T \rightarrow .T * F$ et $T \rightarrow .F$: ajout des règles avec T à gauche
- $F \rightarrow .id$ et $F \rightarrow .(E)$: ajout des règles avec F à gauche

Fermeture$(\{E' \rightarrow .E\}) = \{E' \rightarrow .E,\ E \rightarrow .E + T,\ E \rightarrow .T,\ T \rightarrow .T * F,$
$T \rightarrow .F,\ F \rightarrow .id,\ F \rightarrow .(E)\}$

L'opérateur *Transition*

Définition : transition d'un item

La transition d'un ensemble d'items LR(0) I sur un symbole X (notée Transition(I,X)) est égale à Fermeture($\{A \rightarrow \alpha X.\beta / A \rightarrow \alpha.X\beta \in I\}$*).*

Exemple

Avec $Transition(\{E' \rightarrow E., E \rightarrow E. + T\}, +)$:

- $E \rightarrow E + .T$: seul le deuxième item est conservé
- $T \rightarrow .T * F$, $T \rightarrow .F$: ajout des règles avec T à gauche
- $F \rightarrow .id$, $F \rightarrow .(E)$: ajout des règles avec F à gauche

$Transition(\{E' \rightarrow E., E \rightarrow E. + T\}, +) = \{E \rightarrow E + .T,\ T \rightarrow .T * F,\ T \rightarrow .F,\ F \rightarrow .id,\ F \rightarrow .(E)\}$

Construction des ensembles d'items d'une grammaire

Procédure items(G')

C $\leftarrow$ { $\mathit{Fermeture}(\{S' \rightarrow .S\})$ }

Répète

modif $\leftarrow$ faux

Pour tout $I \in C$ **Faire**

Pour tout $X \in T \cup N$ / $\mathit{Transition}$(I,X) $\neq \emptyset \wedge$ $\mathit{Transition}$(I,X) $\notin C$ **Faire**

$C \leftarrow C \cup \mathit{Transition}$(I,X)

modif $\leftarrow$ vrai

Fin Pour

Fin Pour

Jusqu'à modif = faux

Remarques

- Chaque élément de C est un ensemble d'items LR(0)
- Chaque élément de C est un état

Exemple de construction des items (1/2)

Soit la grammaire des expressions augmentée :

$$\begin{array}{ll} E' & \rightarrow E \\ E & \rightarrow E + T \mid T \\ T & \rightarrow T * F \mid F \\ F & \rightarrow id \mid (E) \end{array}$$

- $I_0 = Fermeture(\{E' \rightarrow .E\}) = \{E' \rightarrow .E, E \rightarrow .E + T, E \rightarrow .T, T \rightarrow .T * F, T \rightarrow .F, F \rightarrow .id, F \rightarrow .(E)\}$
- $I_1 = Transition(I_0,\ E) = \{E' \rightarrow E., E \rightarrow E. + T\}$
- $I_2 = Transition(I_0,\ T) = \{E \rightarrow T., E \rightarrow T. * F\}$
- $I_3 = Transition(I_0,\ F) = \{T \rightarrow F.\}$
- $I_4 = Transition(I_0,\ id) = \{F \rightarrow id.\}$
- $I_5 = Transition(I_0,\ '(') = \{F \rightarrow (.E), E \rightarrow .E + T, E \rightarrow .T, T \rightarrow .T * F, T \rightarrow .F, F \rightarrow .id, F \rightarrow .(E)\}$
- $I_6 = Transition(I_1,\ +) =$ $\{E \rightarrow E + .T, T \rightarrow .T * F, T \rightarrow .F, F \rightarrow .id, F \rightarrow .(E)\}$

Exemple de construction des items (2/2)

Soit la grammaire des expressions augmentée :

E'	$\rightarrow$ E
E	$\rightarrow$ E + T | T
T	$\rightarrow$ T * F | F
F	$\rightarrow$ id | (E)

- $I_7 = Transition(I_2, *) = \{E \rightarrow T * .F, F \rightarrow .id, F \rightarrow .(E)\}$
- $I_8 = Transition(I_5, E) = \{F \rightarrow (E.), E \rightarrow E. + T\}$
- $Transition(I_5, T) = \{E \rightarrow T., E \rightarrow T. * F\} = I_2$
- $Transition(I_5, F) = I_3$, $Transition(I_5, id) = I_4$, $Transition(I_5, '(') = I_5$
- $I_9 = Transition(I_6, T) = \{E \rightarrow E + T., T \rightarrow T. * F\}$
- $Transition(I_6, F) = I_3$, $Transition(I_6, id) = I_4$, $Transition(I_6, '(') = I_5$
- $I_{10} = Transition(I_7, F) = \{T \rightarrow T * F.\}$
- $Transition(I_7, id) = I_4$, $Transition(I_7, '(') = I_5$
- $I_{11} = Transition(I_8, ')') = \{F \rightarrow (E).\}$
- $Transition(I_8, +) = I_6$,
- $Transition(I_9, *) = I_7$

Construction de la table d'analyse SLR(1)

- On part de la grammaire augmentée G' en numérotant chaque règle
- Calcul de C = $\{I_0, I_1, \ldots, I_n\}$, l'ensemble des ensembles d'items LR(0) pour G'
- Construction de l'état i à partir de I_i
- Actions :
  - Si $A \to \alpha.a\beta \in I_i$ avec $a \in T$, $Transition(I_i,\ a){=}I_j$ alors $Action[i, a] =$ décaler j
  - Si $A \to \alpha. \in I_i$, alors $Action[i, b] =$ réduire $A \to \alpha$ avec $b \in$ *Suivant(A)*, avec $A \neq S'$
  - Si $S' \to S. \in I_i$, alors $Action[i, \dashv] =$ accepter
- Successeurs :
  - Si $Transition(I_i,\ A) = I_j$ avec $A \in N$, alors $Successeur[i, A] = j$
- Toutes les autres entrées sont fixées à *erreur*
- L'état initial de l'analyseur est celui construit à partir de l'ensemble contenant $S' \to .S$

Construction de la table d'analyse

Nous partons sur la grammaire augmentée :

$$
\begin{array}{lll}
E' & \rightarrow E & (1) \\
E & \rightarrow E + T & (2) \\
 & \rightarrow T & (3) \\
T & \rightarrow T * F & (4) \\
 & \rightarrow F & (5) \\
F & \rightarrow id & (6) \\
 & \rightarrow (E) & (7)
\end{array}
$$

- Premier(E') $= \{id, '('\}$
- Premier(E) $= \{id, '('\}$
- Premier(T) $= \{id, '('\}$
- Premier(F) $= \{id, '('\}$

- Suivant(E') $= \{\epsilon\}$
- Suivant(E) $= \{+, ')', \epsilon\}$
- Suivant(T) $= \{*, +, ')', \epsilon\}$
- Suivant(F) $= \{*, +, ')', \epsilon\}$

Construction : *voir au tableau*

Construction de la table d'analyse : résultat

États	*Action*						*Successeur*		
	id	+	*	(	)	⊣	E	T	F
0	d,4			d,5			1	2	3
1		d,6				Acc			
2		r,3	d,7		r,3	r,3			
3		r,5	r,5		r,5	r,5			
4		r,6	r,6		r,6	r,6			
5	d,4			d,5			8	2	3
6	d,4			d,5				9	3
7	d,4			d,5					10
8		d,6			d,11				
9		r,2	d,7		r,2	r,2			
10		r,4	r,4		r,4	r,4			
11		r,7	r,7		r,7	r,7			

Exemple d'analyse

Pile	Entrée	Action
0	id + id * id ⊣	d,4
0id4	+ id * id ⊣	r,6
0F3	+ id * id ⊣	r,5
0T2	+ id * id ⊣	r,3
0E1	+ id * id ⊣	d,6
0E1+6	id * id ⊣	d,4
0E1+6id4	* id ⊣	r,6
0E1+6F3	* id ⊣	r,5
0E1+6T9	* id ⊣	d,7
0E1+6T9*7	id ⊣	d,4
0E1+6T9*7id4	⊣	r,6
0E1+6T9*7F10	⊣	r,4
0E1+6T9	⊣	r,2
0E1	⊣	Acc

Grammaire SLR(1)

Définition : grammaire SLR(1)

Une grammaire est dite SLR(1) s'il est possible de construire la table SLR(1).

- Toute grammaire SLR(1) est non ambiguë
- Beaucoup de grammaires non ambiguës ne sont pas SLR(1)

Conflit décalage/réduction

Soit la grammaire suivante : $\begin{array}{lcl} S & \rightarrow & G = D \mid D \\ G & \rightarrow & *D \mid id \\ D & \rightarrow & G \end{array}$

- $I_0 = \{S' \rightarrow .S, S \rightarrow .G = D, S \rightarrow .D, D \rightarrow .G, G \rightarrow . * D, G \rightarrow .id\}$
- $I_1 = Transition(I_0,\ S) = \{S' \rightarrow S.\}$
- $I_2 = Transition(I_0,\ G) = \{S \rightarrow G. = D, D \rightarrow G.\} \ldots$

- $Action[2, =]$ = décaler X (ici X est un état quelconque)
- Or, $Suivant$(D) contient '=', donc $Action[2, =]$ = réduire 5
- Conflit décalage/réduction lié à un manque d'informations :
  $\hookrightarrow$ Réduction dès que possible sur le *Suivant*
- Idée : proposer une réduction uniquement avec certains symboles

Symbole de prévision et item LR(1)

Définition : item LR(1)

Un item LR(1) est un item LR(0) suivi d'un symbole terminal (ou de $\dashv$) appelé ***symbole de prévision****.*

- Symbole de prévision utile uniquement quand une partie droite de règle est au sommet de la pile
- Permet de décider s'il faut réduire ou pas
- Si $a \in T \cup \{\dashv\}$, $A \in N$ et $\alpha_1, \alpha_2 \in \{T \cup N\}^*$, $[A \rightarrow \alpha_1.\alpha_2, a]$, le symbole a signifie :
  - $\alpha_2 \neq \epsilon$: aucune
  - $\alpha_2 = \epsilon$: la réduction de $A \rightarrow \alpha_1\alpha_2$ est effectuée sur a uniquement
- Les deux items $[A \rightarrow \alpha_1.\alpha_2, a]$ et $[A \rightarrow \alpha_1.\alpha_2, b]$ peuvent s'écrire $[A \rightarrow \alpha_1.\alpha_2, \{a, b\}]$

Fermeture d'un ensemble d'items LR(1)

Définition : fermeture d'un ensemble d'items LR(1)

La fermeture F(I) d'un ensemble I d'items LR(1) est définie par :

- $I \subseteq Fermeture(I)$
- *Si* $[A \rightarrow \alpha.B\beta, a] \in$ *Fermeture(I) et* $B \rightarrow \gamma \in R$*, alors* $[B \rightarrow .\gamma, b] \in$ *Fermeture(I) pour tout* $(b \in N) \in Premier(\beta a)$

Procédure Fermeture(I)

Répète

modif $\leftarrow$ faux

Pour tout $[A \rightarrow \alpha.B\beta, a] \in I$, $B \rightarrow \gamma \in R$ et $b \in$ *Premier*(βa) / $[B \rightarrow .\gamma, b] \notin I$ **Faire**

$I \leftarrow I \cup [B \rightarrow .\gamma, b]$

modif $\leftarrow$ vrai

Fin Pour

Jusqu'à modif = faux

Construction de l'automate LR(1)

Procédure items(G')

C $\leftarrow$ { $Fermeture([S' \rightarrow .S, \dashv])$ }

Répète

modif $\leftarrow$ faux

Pour tout $I \in C$ **Faire**

Pour tout $X \in T \cup N$ / $Transition$(I,X) $\neq \emptyset \wedge$ $Transition$(I,X) $\notin C$ **Faire**

$C \leftarrow C \cup Transition$(I,X)

modif $\leftarrow$ vrai

Fin Pour

Fin Pour

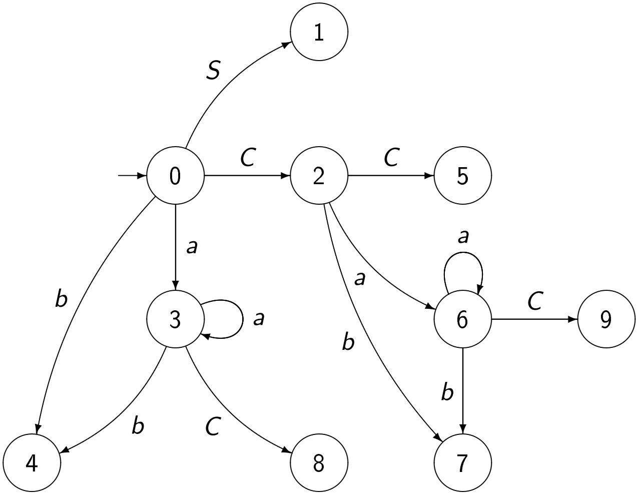
Jusqu'à modif = faux

Exemple de construction des items LR(1)

$$
\begin{aligned}
S' &\to S\\
S &\to CC\\
C &\to aC \mid b
\end{aligned}
$$

- $I_0 = \{[S' \to .S, \dashv], [S \to .CC, \dashv], [C \to .aC, \{a,b\}], [C \to .b, \{a,b\}]\}$
- $I_1 = Transition(I_0,\ S) = \{[S' \to S., \dashv]\}$
- $I_2 = Transition(I_0,\ C) = \{[S \to C.C, \dashv], [C \to .aC, \dashv], [C \to .b, \dashv]\}$
- $I_3 = Transition(I_0,\ a) =$
  $\{[C \to a.C, \{a,b\}], [C \to .aC, \{a,b\}], [C \to .b, \{a,b\}]\}$
- $I_4 = Transition(I_0,\ b) = \{[C \to b., \{a,b\}]\}$
- $I_5 = Transition(I_2,\ C) = \{[S \to CC., \dashv]\}$
- $I_6 = Transition(I_2,\ a) = \{[C \to a.C, \dashv], [C \to .aC, \dashv], [C \to .b, \dashv]\}$
- $I_7 = Transition(I_2,\ b) = \{[C \to b., \dashv]\}$
- $I_8 = Transition(I_3,\ C) = \{[C \to aC., \{a,b\}]\}$
- $Transition(I_3,\ a) = I_3,\ Transition(I_3,\ b) = I_4$
- $I_9 = Transition(I_6,\ C) = \{C \to aC., \dashv]\}$
- $Transition(I_6,\ a) = I_6,\ Transition(I_6,\ b) = I_7$

Graphe de la fonction de transition



Construction de la table LR(1)

1. Construire l'automate LR(1)
   $\hookrightarrow$ États = ensemble d'items LR(1)
   $\hookrightarrow$ Transitions
2. Remplir la table :
   Pour les actions :
   - Si $[A \rightarrow \alpha.a\beta, b] \in I_i$ et $a \in T$, *Transition*(I_i, a) = I_j alors *Action*[i, a] = décaler j
   - Si $[A \rightarrow \alpha., b] \in I_i$ alors *Action*[i, b] = réduire $A \rightarrow \alpha$
   - Si $[S' \rightarrow S., \dashv] \in I_i$ alors *Action*[i, $\dashv$] = accepter

   Pour la fonction successeur (idem à SLR)
   - Si *Transition*[I_i, A] = I_j alors *Successeur*[i, A] = j

Table obtenue avec la construction précédente

États	*Action*			*Successeur*	
	a	b	⊣	S	C
0	d,3	d,4		1	2
1			acc.		
2	d,6	d,7			5
3	d,3	d,4			8
4	r,3	r,3			
5			r,1		
6	d,6	d,7			9
7			r,3		
8	r,2	r,2			
9			r,2		

Grammaire LR(1)

Définition : grammaire LR(1)

Une grammaire est dite LR(1) s'il est possible de construire la table LR(1).

- Toute grammaire SLR(1) est LR(1) mais le nombre d'états de l'analyseur LR(1) peut avoir plus d'états que celui SLR(1)
- Les tables LR(1) sont généralement beaucoup plus grandes que les tables SLR(1) mais elles permettent l'analyse d'un plus grand nombre de langages

Construction de l'automate LALR(1)

- LALR pour *Look Ahead LR*
- Objectif : réduire le nombre d'états de l'automate LR(1)
  $\hookrightarrow$ En essayant de reconnaître presqu'autant de grammaires

Exemple

Avec la grammaire précédente :

$$\begin{aligned} S &\rightarrow \mathrm{CC} \\ C &\rightarrow \mathrm{aC} \\ C &\rightarrow \mathrm{b} \end{aligned}$$

- $I_4 = \{[C \rightarrow b., \{a, b\}]\}$ et $I_7 = \{[C \rightarrow b., \dashv]\}$ sont semblables
- Le cœur (item LR(0)) est identique : $C \rightarrow b.$

Comportement des états 4 et 7

- Le langage engendré par la grammaire est a^*ba^*b
- Quand l'analyseur lit $aa \dots aba \dots ab$, il décale les premiers a et le premier b puis se retrouve dans l'état 4
- Il réduit le symbole b par la règle $C \rightarrow b$ si la fenêtre est égale à a ou b
  $\hookrightarrow$ Correct car les mots a^*b ne font pas partie de ce langage
  $\hookrightarrow$ Donc, pas de réduction sur $\dashv$
- Après avoir décalé le deuxième b, il est normal de réduire b à C uniquement sur $\dashv$

Construction de l'automate LALR(1) (1/2)

- Les états 4 et 7 peuvent être regroupés dans un nouvel état I_{4-7} :
$$I_{4-7} = \{[C \rightarrow b., \{a, b, \dashv\}]\}$$
- Les transitions de I_0, I_2, I_3, I_6 sur b arrivent sur I_{4-7}
- L'action de l'état I_{4-7} est de réduire $C \rightarrow b$ pour toute entrée
- L'automate modifié a un comportement très proche de l'original :
    - Il réduit $C \rightarrow b$ sur le deuxième b même si le symbole de la fenêtre est a ou b
    - Dans ce cas, l'erreur sera détectée plus tard (avant le prochain décalage)

Construction de l'automate LALR(1) (2/2)

- De façon générale, il est possible de regrouper dans un seul ensemble, tous les états qui possèdent le même cœur (seuls les symboles de prévision diffèrent)
- Il suffit d'appliquer l'algorithme de construction de la table LR(1) à l'automate LALR(1)
- Exemple :
  - I_3 et I_6 : $\{[C \rightarrow a.C, \{a, b, \dashv\}], [C \rightarrow .aC, \{a, b, \dashv\}], [C \rightarrow .b, \{a, b, \dashv\}]\}$
  - I_4 et I_7 : $\{[C \rightarrow b., \{a, b, \dashv\}]\}$
  - I_8 et I_9 : $\{[C \rightarrow aC., \{a, b, \dashv\}]\}$

La table de l'automate LALR(1)

États	*Action*			*Successeur*	
	a	b	⊣	S	C
0	d,3-6	d,4-7		1	2
1			acc.		
2	d,3-6	d,4-7			5
3-6	d,3-6	d,4-7			8-9
4-7	r,3	r,3	r,3		
5			r,1		
8-9	r,2	r,2	r,2		

Remarques

- L'analyseur LALR(1) comporte le même nombre d'états que l'analyseur SLR(1)
- Il ne peut y avoir de conflit décaler/réduire en construisant l'analyseur LALR(1) car le symbole de prévision n'est utile qu'en cas de réduction
- Il peut y avoir des conflits réduire/réduire
- La technique de construction proposée pour construire la table LALR n'est pas très efficace :
  $\hookrightarrow$ Elle utile l'automate LR(1) qui possède un grand nombre d'états

Construction efficace des tables d'analyse LALR (1/2)

- Un ensemble d'items peut être représenté par son noyau :
  - Soit l'item correspondant à l'axiome ($S' \rightarrow .S$)
  - Soit les items où le repère n'est pas au début

  $\hookrightarrow$ Cela revient à prendre les items LR(0) et à ignorer ceux obtenus par fermeture
- Construction des noyaux à partir de I_0 qui contient uniquement $S' \rightarrow .S$
- Pour construire les états :
  - Si $\beta \rightarrow \gamma.X\delta \in I$, alors $\beta \rightarrow \gamma X.\delta \in$ *Transition(I, X)*
  - Si $\beta \rightarrow \gamma.C\delta \in I$ et $C \stackrel{*}{\Rightarrow} A\eta$, s'il existe une règle $A \rightarrow X\beta$, alors $A \rightarrow X.\beta \in$ *Transition(I, X)*

Construction efficace des tables d'analyse LALR (2/2)

- À partir des noyaux, il reste à calculer les symboles de prévision
  - Ils sont propagés d'un état à l'autre
  - Ils peuvent également être générés spontanément
- Pour chaque noyau, en partant de I_0, il faut calculer la fermeture de l'item avec un contexte fictif (ici $\#$)

Exemple

$$
\begin{array}{l|ll}
S' \rightarrow S & F(I_0) = & \{[S' \rightarrow .S, \#] \\
S \rightarrow L = R & & [S \rightarrow .L = R, \#] \\
S \rightarrow R & & [S \rightarrow .R, \#] \\
L \rightarrow *R & & [L \rightarrow .*R, \#/=] \\
L \rightarrow id & & [L \rightarrow .id, \#/=] \\
R \rightarrow L & & [R \rightarrow .L, \#]\}
\end{array}
$$

Présentation

- Une grammaire ambiguë n'est pas LR
  $\hookrightarrow$ Impossible de construire un analyseur SLR, LR ou LALR
- Or, ces grammaires possèdent des avantages :
  - Plus lisibles, plus naturelles
  - Plus concises (moins de règles et de symboles)

Rappels avec la grammaire des expressions

$$\begin{aligned} E &\rightarrow E + E \\ E &\rightarrow E * E \\ E &\rightarrow (E) \\ E &\rightarrow \text{id} \end{aligned} \qquad \begin{aligned} \text{E} &\rightarrow \text{E} + \text{T} \mid \text{T} \\ \text{T} &\rightarrow \text{T} * \text{F} \mid \text{F} \\ \text{F} &\rightarrow (\text{E}) \mid \text{id} \end{aligned}$$

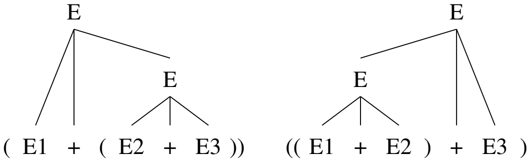
- Comme vu précédemment, l'analyseur réduit souvent $E \rightarrow T$ et $T \rightarrow F$ qui ne sont utiles que pour donner la priorité à '*' par rapport au '+'

Items LR(0) pour la grammaire des expressions augmentée

- $I_0 = \{E' \rightarrow .E, E \rightarrow .E + E, E \rightarrow .E * E, E \rightarrow .(E), E \rightarrow .id\}$
- $I_1 = Transition(I_0, E) = \{E' \rightarrow E., E \rightarrow E. + E, E \rightarrow E. * E\}$
- $I_2 = Transition(I_0, () = \{E \rightarrow (.E), \ldots\}$
- $I_3 = Transition(I_0, id) = \{E \rightarrow id.\}$
- $I_4 = Transition(I_1, +) = \{E \rightarrow E + .E, \ldots\}$
- $I_5 = Transition(I_1, *) = \{E \rightarrow E * .E, \ldots\}$
- $I_6 = Transition(I_2, E) = \{E \rightarrow (E.), E \rightarrow E. + E, E \rightarrow E * .E\}$
- $Transition(I_2, ()) = I_2$, $Transition(I_2, id) = I_3$
- $I_7 = Transition(I_4, E) = \{E \rightarrow E + E., E \rightarrow E. + E, E \rightarrow E. * E\}$
- $Transition(I_4, () = I_2$, $Transition(I_4, id) = I_3$
- $I_8 = Transition(I_5, E) = \{E \rightarrow E * E., E \rightarrow E. + E, E \rightarrow E. * E\}$
- $Transition(I_5, () = I_2$, $Transition(I_5, id) = I_3$
- $I_9 = Transition(I_6,)) = \{E \rightarrow (E).\}$
- $Transition(I_6, +) = I_4$, $Transition(I_6, *) = I_5$
- $Transition(I_7, +) = I_4$, $Transition(I_7, *) = I_5$
- $Transition(I_8, +) = I_4$, $Transition(I_8, *) = I_5$

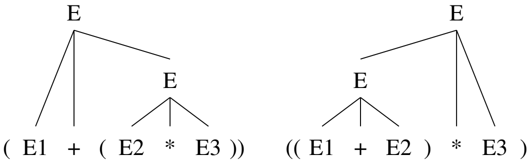
Conflits décalage/réduction à l'état 7

- $I_7 = Transition(I_4, E) = \{E \rightarrow E + E., E \rightarrow E. + E, E \rightarrow E. * E\}$
- Sur un '+' :



Décaler état 4 *Réduire règle 1*

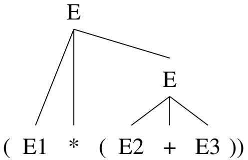
- Sur un '*' :



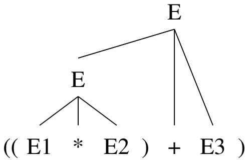
Décaler état 5 *Réduire règle 1*

Conflits décalage/réduction à l'état 8

- $I_8 = Transition(I_4, E) = \{E \rightarrow E * E., E \rightarrow E. + E, E \rightarrow E. * E\}$
- Sur un '+' :

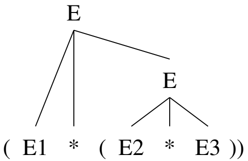


Décaler état 4

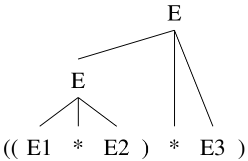


Réduire règle 2

- Sur un '*' :



Décaler état 5



Réduire règle 2

Priorités et associativités

- En considérant le mot $id + id * id$, après avoir traité $id + id$, l'analyseur est dans l'état suivant :

Pile	Entrée
0 E 1 + 4 E 7	* id ⊣

- En ajoutant une priorité sur '*' par rapport à '+', l'analyseur doit décaler et non réduire
- En considérant le mot $id + id + id$, après avoir traité $id + id$, l'analyseur est dans l'état suivant :

Pile	Entrée
0 E 1 + 4 E 7	+ id ⊣

- En ajoutant une associativité gauche sur '+', l'action correcte à la réduction de $E \rightarrow E + E$

Récupération sur erreur (1/2)

- Un analyseur retourne une erreur lorsqu'il consulte la table d'analyse *Action* et trouve une entrée *erreur* (vide)
- Solution 1 : isoler la phrase contenant une erreur
- Dès qu'une erreur est détectée :
  - Dépiler jusqu'à trouver un état où la fonction *Successeur* est définie pour un $A \in N$
  - Avancer jusqu'à trouver un $a \in Suivant(A)$
  - Empiler $Successeur[s, A]$ (l'état s est dans la pile)
- A doit être déterminé stratégiquement :
  $\hookrightarrow$ Expression, instruction, bloc

Récupération sur erreur (2/2)

- Solution 2 : récupération au niveau du syntagme (au milieu de la phrase)
- Pour chaque entrée erreur, l'erreur est corrigée manuellement
  $\hookrightarrow$ Suivant le langage, on détermine l'erreur la plus vraisemblable
- Procédure de récupération pour chaque cas
- Exemple avec la grammaire des expressions :
  - $id+$ (états 0, 2, 4 et 5) : il manque un identificateur
    $\hookrightarrow$ Ajout d'un id imaginaire et de l'état 3 (successeur des états 0, 2, 4 et 5)
  - $id + id)$ (états 0, 1, 2, 4 et 5) : il y a une parenthèse en trop
    $\hookrightarrow$ Décalage de la fenêtre sans en tenir compte
  - $id(id)$ (états 1 et 6) : il manque un opérateur
    $\hookrightarrow$ Ajout d'un + et de l'état 4
  - $id + (id * id \dashv$: il manque une parenthèse
    $\hookrightarrow$ Ajout d'une parenthèse fermante et de l'état 9