Université de Lille 1 SCIENCES ET TECHNOLOGIES — UFR IEEA Informatique, Electronique Electrotechnique, Automatique

Informatique

L3S5 Miage

Analyse Syntaxique – CTD - Partie 4

Analyse syntaxique ascendante

L'analyse ascendante correspond à la seconde manière de construire systématiquement un automate à pile partant d'une grammaire donnée (voir Cours-TD 4 page 3). Dans cet automate, les transitions empilent les symboles terminaux de la grammaire, et dépilent les non terminaux en ce sens qu'un membre droit de règle situé en sommet de pile est remplacé par le membre gauche correspondant. Plus prosaiquement :

- on part du mot à analyser,
- on remplace itérativement des fragments du mot courant qui correspondent à des membres droits de règle de production par le membre gauche de cette règle,
- l'analyse réussit si le mot courant final est l'axiome de la grammaire

Analyseur ascendant prédictif LR(k)

- Left-to-right scanning : on lit l'entrée de gauche à droite.
- Rightmost derivation in reverse : on construit une dérivation droite en partant du mot à analyser.
- on utilise k lettres du mot d'entrée pour faire la prédiction.

Famille d'algorithmes (LR(0), SLR(1), LR(1), LALR(1)) basés sur les opérations de décalage/réduction.

- décalage : on lit des terminaux dans le mot d'entrée
- réduction : on remplace une partie droite par une partie gauche de règle de production.

On souhaite analyser le mot $w = a_1 \dots a_n$ pour une grammaire G. A chaque instant, on dispose :

- d'un suffixe w' du mot d'entrée
- d'une pile contenant des terminaux et des non-terminaux

Initialement, le suffixe est le mot à analyser w et la pile est vide. A chaque étape de l'algorithme, si le suffixe est $a_m \dots a_n$ et le contenu de la pile est $\alpha_1 \dots \alpha_l$ (où α_l est le sommet de pile), on a $\alpha_1 \dots \alpha_l \rightarrow_G^* a_1 \dots a_{m-1}$. L'analyse réussit si le mot w a été entièrement lu et si la pile contient l'axiome.

Prenons pour exemple la grammaire suivante :

$$S' \rightarrow S \qquad \text{Augmentation de la grammaire}$$
$$S \rightarrow Ac$$
$$A \rightarrow AaAb \mid d$$

- Décalage : le premier terminal du mot d'entrée est effacé et est placé au sommet de la pile.

$$\boxed{} \quad dadbc \quad \Rightarrow \quad \boxed{d} \quad adbc$$

- Réduction : les n premiers symboles sur la pile forme un membre droit de règle de production, on les dépile et on empile le membre gauche correspondant.

$$\boxed{d} \quad adbc \quad \Rightarrow \quad \boxed{A} \quad adbc$$

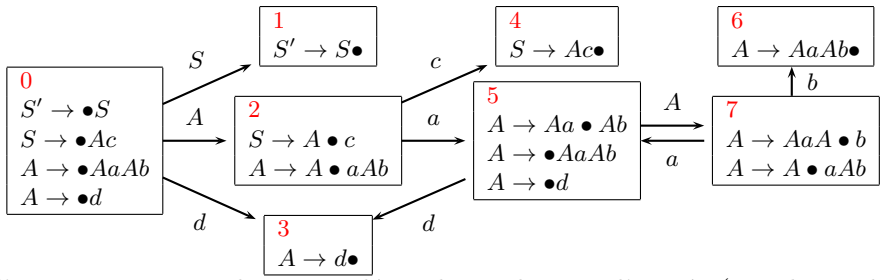
En continuant le processus : on ne peut pas réduire, donc on décale
$\boxed{Aa}$ $dbc\$$ $dadbc \leftarrow Aadbc$
on ne peut pas réduire, donc on décale
$\boxed{Aad}$ $bc\$$ $dadbc \leftarrow Aadbc$
Ni Aad, ni ad, ni d ne sont préfixe strict d'un membre droit de règle, donc un décalage ne peut être utile ; donc on réduit
$\boxed{AaA}$ $bc\$$ $dadbc \leftarrow Aadbc \leftarrow AaAbc$
on ne peut pas réduire, donc on décale
$\boxed{AaAb}$ $c\$$ $dadbc \leftarrow Aadbc \leftarrow AaAbc$
Ni $AaAb$, ni aAb, ni Ab, ni b ne sont préfixe strict d'un membre droit de règle, donc un décalage ne peut être utile ; donc on réduit
$\boxed{A}$ $c\$$ $dadbc \leftarrow Aadbc \leftarrow AaAbc \leftarrow Ac$
on ne peut pas réduire, donc on décale
$\boxed{Ac}$ $\$$ $dadbc \leftarrow Aadbc \leftarrow AaAbc \leftarrow Ac$
on ne peut plus décaler ; on réduit
$\boxed{S}$ $\$$ $dadbc \leftarrow Aadbc \leftarrow AaAbc \leftarrow Ac \leftarrow S$
on réduit à nouveau
$\boxed{S'}$ $\$$ $dadbc \leftarrow Aadbc \leftarrow AaAbc \leftarrow Ac \leftarrow S \leftarrow S'$

Le mot *dadbc* est bien engendré par la grammaire.
Comment par un algorithme décider à chaque étape si on doit réduire (et par quelle règle) ou décaler ? On construit un automate déterministe appelé *automate LR(0)* qui dira quoi faire en fonction du contenu de la pile et la première lettre de l'entrée.

Automates LR(0)

Règles pointées Pour toute règle de la grammaire $X \to \beta\gamma$ (avec $\beta, \gamma \in (\Sigma \cup V)^*$,
$X \to \beta \bullet \gamma$ est une *règle pointée.*
Une règle pointée $X \to \beta \bullet \gamma$ est dite *complète* si $\gamma = \epsilon$.
Exemple Pour les règles $S' \to S,\ S \to Ac,\ A \to AaAb \mid d$, les règles pointées sont : $S' \to \bullet S,\ S' \to S\bullet,\ S \to \bullet A$
$S \to Ac\bullet,\ A \to \bullet AaAb,\ A \to A \bullet aAb,\ A \to Aa \bullet Ab,$
$A \to AaA \bullet b,\ A \to AaAb\bullet,\ A \to \bullet d,\ A \to d\bullet.$
Saturation Un ensemble de règles pointées E est dit *saturé* si $X \to \beta \bullet Y\gamma$ appartient à E et $Y \in V$ alors pour toute règle $Y \to \alpha$ de la grammaire, $Y \to \bullet\alpha$ appartient à E.
Pour un ensemble de règles pointées E, on note $Saturation(E)$ le plus petit ensemble saturé E' contenant E.
Etats de l'automate : Les *états* Q de l'automate LR(0) seront des ensembles saturés de règles pointées. Ils sont souvent appelés *collections LR(0)* (ou *LR(0)-items*). L' *état initial* de l'automate est $Saturation(\{S' \to \bullet S\})$.
Transitions de l'automate : Pour un ensemble saturé de règles pointées E et $x \in \Sigma \cup V$, on définit l'état $\delta(E, x)$ comme $Saturation(\{X \to \beta x \bullet \gamma \mid X \to \beta \bullet x\gamma \in E\})$ Lors de la construction de l'automate, on ne considèrera bien-sûr que les états accessibles, en partant de l'état initial $Saturation(\{S' \to \bullet S\})$.
Exemple

$S' \to S$
$S \to Ac$
$A \to AaAb \mid d$



La construction de l'automate est totalement indépendante du mot d'entrée (pas de symbole de prévision), d'où son nom : LR(0). On utilise l'automate pour "lire" le mot dans la pile. Si la pile

contient $\alpha_1 \dots \alpha_m$, alors l'état $q = \delta(0, \alpha_1 \dots \alpha_m)$ (où 0 est l'état initial) nous dit ce qu'il faut faire :

- si l'état q contient une règle pointée complète $X \to \beta\bullet$, alors on réduit en utilisant la règle $X \to \beta$.
- si le mot courant débute par le terminal a de Σ et s'il existe une transition $\delta(q, a) = q'$, alors on décale.

Si pour tout état q de l'automate,

- q ne contient au plus qu'une seule règle pointée complète (*pas de conflit réduction/réduction*) et
- si q contient une règle pointée complète alors pour aucun a de Σ il n'existe de transition $q' = \delta(q, a)$ (*pas de conflit décalage/réduction*),

alors l'automate est dit LR(0)-*consistant* et la grammaire est alors LR(0).

Plutot que d'évaluer systèmatiquement $\delta(0, \alpha_1 \dots \alpha_m)$ où $\alpha_1 \dots \alpha_m$ est le contenu de la pile (lu du bas vers le haut), on garde trace des états directement dans la pile. La pile est donc une suite $0\,\alpha_1\,q_1\,\alpha_2\,\dots\,q_{m-1}\,\alpha_m\,q_m$ où

- les q_i sont des états de l'automate,
- $q_i = \delta(q_{i-1}, \alpha_i) = \delta(0, \alpha_1 \dots \alpha_i)$ pour tout $1 \leq i \leq m$.

Initialement, la pile contient l'état initial 0. Lors d'une étape du calcul,

- si on décale un terminal a, on empile a puis l'état $q = \delta(q_m, a)$.
- si on réduit par une règle $X \to \alpha_r \dots \alpha_m$, on dépile q_m, α_m, q_{m-1}, …, q_r et α_r. on empile X puis l'état $\delta(q_{r-1}, X)$.

Tables LR(0)

L'automate LR(0) est en fait codé dans deux tables, une table *Action* et une table *Successeur* :

- *Action* : $Q \times (\Sigma \cup \{\$\}) \to$ un ensemble d'actions. Les actions :
  - décaler q : $d\,q$ avec $q \in Q$.
  - réduire p : $r\,p$ avec p une production
  - accepter
  - erreur
- *Successeur* : $Q \times V \to Q$ est la restriction de la fonction de transition aux non-terminaux.

On remplit ces tables à partir de l'automate LR(0).

Table *Action* :

- pour tout $a \in \Sigma$, $q \in Q$, si $\delta(q, a) = q'$ alors mettre "$d\,q'$" dans la case (q, a)
- pour tout $a \in (\Sigma \cup \{\$\})$, $q \in Q$, si q contient une règle pointée complète $X \to \beta\bullet$ avec $X \neq S'$, alors mettre "$r\,X \to \beta$" dans la case (q, a)
- mettre "accepter" dans la case $(q, \$)$ où q est l'état contenant $S' \to S\bullet$.
- mettre "erreur" dans toutes les cases encore vide.

Si l'automate est LR(0)-consistant alors les cases de la table *Action* contiennent exactement une action.

Table *Successeur* : pour tout $q \in Q$ et $X \in V$, si $\delta(q, X) = q'$ alors mettre q' dans la case (q, X).

Exemple

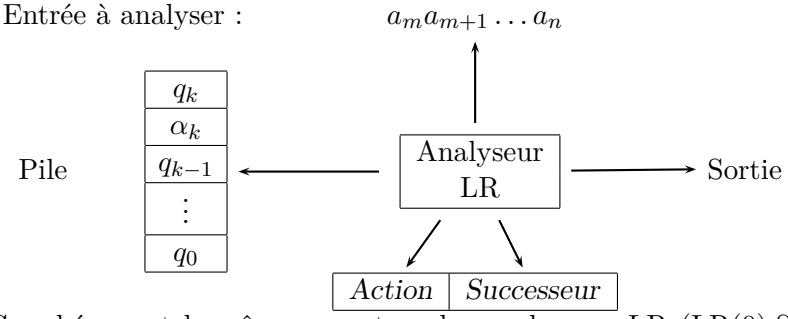
Table *Action*

	a	b	c	d	\$
0	erreur	erreur	erreur	$d\,3$	erreur
1	erreur	erreur	erreur	erreur	accepter
2	$d\,5$	erreur	$d\,4$	erreur	erreur
3	$r\,A \to d$	$r\,A \to d$	$r\,A \to d$	$r\,A \to d$	$r\,A \to d$
4	$r\,S \to Ac$	$r\,S \to Ac$	$r\,S \to Ac$	$r\,S \to Ac$	$r\,S \to Ac$
5	erreur	erreur	erreur	$d\,3$	erreur
6	$r\,A \to AaAb$	$r\,A \to AaAb$	$r\,A \to AaAb$	$r\,A \to AaAb$	$r\,A \to AaAb$
7	$d\,5$	$d\,6$	erreur	erreur	erreur

Table *Successeur*

	S'	S	A
0		1	2
1			
2			
3			
4			
5			7
6			
7			

Fonctionnement d'un analyseur LR



Ce schéma est le même pour tous les analyseurs LR (LR(0),SLR(1),LALR(1), LR(1),LR(k)) ; seules changent les tables *Action* et *Successeur*.

La pile contient

$$0\,\alpha_1\,q_1\,\alpha_2\,\dots\,q_{m-1}\,\alpha_m\,q_m$$

et le mot à analyser est $\alpha_{m+1} \dots \alpha_n$.

Si la case (q_m, α_{m+1}) de la table *Action* contient

- accepter alors l'analyse réussit,
- erreur alors l'analyse échoue,
- $d\,q'$, on empile α_{m+1}, puis q',
- $r\,X \to \alpha_r \dots \alpha_m$, on dépile $\alpha_m, \dots, \alpha_r$ et les états correspondants $q_m, \dots, q_r$; on empile X, puis le contenu de la case (q_{r-1}, X) de la table *Successeur*.

Exemple

$\boxed{0}$ *dadbc*\$

Action contient $d\,3$ en $(0, d)$

$\boxed{0d3}$ *adbc*\$

Action contient $r\,A \to d$ en $(3, a)$ et *Successeur* 2 en $(0, A)$

$\boxed{0A2}$ *adbc*\$

Action contient $d\,5$ en $(2, a)$

$\boxed{0A2a5}$ *dbc*\$

Action contient $d\,3$ en $(5, d)$

$\boxed{0A2a5d3}$ *bc*\$

Action contient $r\,A \to d$ en $(3, b)$ et *Successeur* 7 en $(5, A)$

$\boxed{0A2a5A7}$ *bc*\$

Action contient $d\,6$ en $(7, b)$

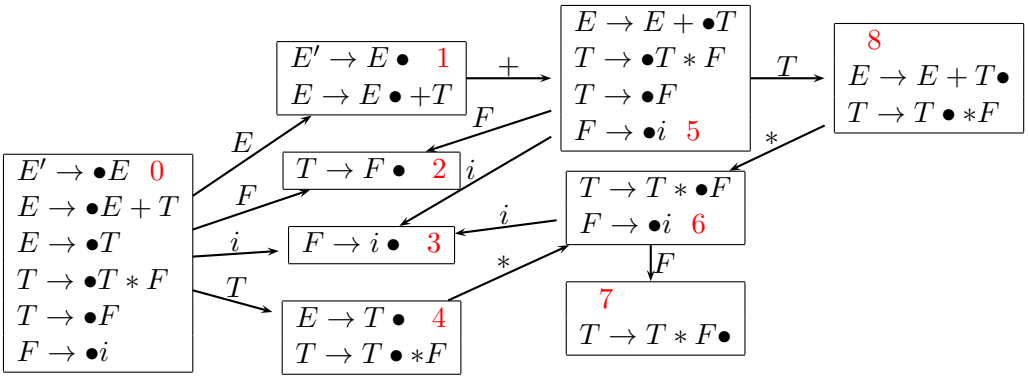
$\boxed{0A2a5A7b6}$ *c*\$

Action contient $r\,A \to AaAb$ en $(6, c)$ et *Successeur* 2 en $(0, A)$

$\boxed{0A2}$ $c\$$
Action contient $d\,4$ en $(2, c)$
$\boxed{0A2c4}$ $\$$
Action contient $r\,S \to Ac$ en $(4, \$)$ et *Successeur* 1 en $(0, S)$
$\boxed{0S1}$ $\$$
Action contient accepter en $(1, \$)$

Exercice 1. Faire l'automate LR(0) pour la grammaire des expressions suivante :

$$E' \to E,\ E \to E + T \mid T,\ T \to T * F \mid T,\ F \to i$$



Conflits dans cet automate Cet automate n'est pas LR(0)-consistant : conflit décalage/réduction pour l'état 1 par exemple.
$\boxed{0}$ $i + i * i\$$ on décale vers l'état 3
$\boxed{0i3}$ $+ i * i\$$ on réduit par $F \to i$ et on va dans l'état 2
$\boxed{0F2}$ $+ i * i\$$ on réduit par $T \to F$ et on va dans l'état 4
$\boxed{0T4}$ $+ i * i\$$ on réduit par $E \to T$ et on va dans l'état 1
$\boxed{0E1}$ $+ i * i\$$
Conflit entre réduire par $E' \to E$ et décaler $+$!! Mais si on réduit alors pour que le mot soit accepté, il faudrait qu'il existe une dérivation (droite) $E' \to^*_G E' + i * i$; ce n'est bien-sûr pas le cas puisque *Suivant*(E')=$\{\$\}$. Donc on décale et on va dans l'état 5.
Quels sont les autres conflits possibles ?

Analyse SLR(1)

Comme on vient de le voir dans l'exercice, on peut résoudre les conflits décalage/réduction en regardant si le terminal en tête du mot restant à analyser est dans l'ensemble des *Suivants* du non-terminal en membre gauche de la règle utilisable pour la réduction.Si c'est le cas, on effectue la réduction, sinon on effectue un décalage. Ceci est formalisé par la méthode SLR(1) dont voici la manière de construire la table *Action* à partir de l'automate LR(0) : Si l'automate LR(0) a une ensemble d'états Q et un ensemble de transitions δ,
Table *Action* :

- pour tout $a \in \Sigma$, $q \in Q$, si $\delta(q, a) = q'$ alors mettre "$d\,q'$" dans la case (q, a)
- pour toute règle pointée complète $X \to \beta\bullet$ de q avec $X \neq S'$, mettre "$r\,X \to \beta$" dans la case (q, a) pour tout $a \in Suivant(X)$.
- mettre "accepter" dans la case $(q, \$)$ où q est l'état contenant $S' \to S\bullet$.
- mettre "erreur" dans toutes les cases encore vide.

Table *Successeur* : identique à la table *Successeur* LR(0).

Si la table *Action* contient exactement une action par case, alors la grammaire est SLR(1).

Application à l'exercice précédent

$Suivant(E') = \{\$\}$ $Suivant(E) = \{+, \$\}$

$Suivant(T) = \{+, *, \$\}$ $Suivant(F) = \{+, *, \$\}$

Table *Action*

	$+$	$*$	i	$\$$
0	erreur	erreur	$d\,3$	erreur
1	$d\,5$	erreur	erreur	accepter
2	$r\,T \to F$	$r\,T \to F$	erreur	$r\,T \to F$
3	$r\,F \to i$	$r\,F \to i$	erreur	$r\,F \to i$
4	$r\,E \to T$	$d\,6$	erreur	$r\,E \to T$
5	erreur	erreur	$d\,3$	erreur
6	erreur	erreur	$d\,3$	erreur
7	$r\,T \to T * F$	$r\,T \to T * F$	erreur	$r\,T \to T * F$
8	$r\,E \to E + T$	$d\,6$	erreur	$r\,E \to E + T$

Table *Successeur*

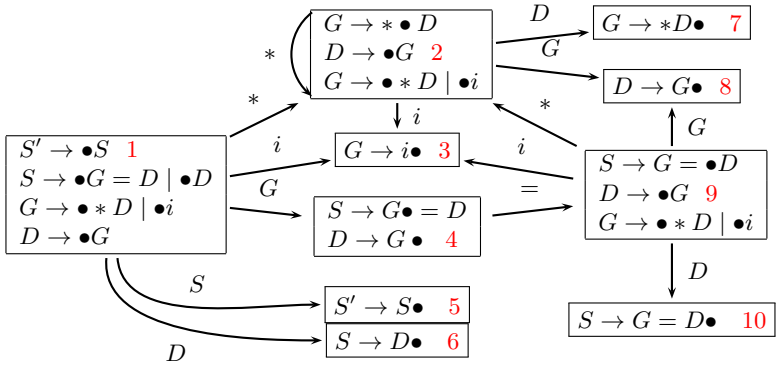
	E'	E	T	F
0		1	4	2
1				
2				
3				
4				
5			8	2
6				7
7				
8				

Une seule action par case dans la table *Action* ⇒ la grammaire est SLR(1).

- toute grammaire LR(0) est SLR(1)
- toute grammaire SLR(1) est non-ambigüe
- bcp de grammaires non ambigües ne sont pas SLR(1)

Exercice 2. Construire les tables SLR(1) pour la grammaire $\left\{ \begin{array}{l} S \to G = D \mid D \\ G \to *D \mid i \\ D \to G \end{array} \right\}$

Pour cela, on construit l'automate LR(0).



En appliquant la méthode SLR(1) décrite ci-dessus, il va subsister un conflit pour l'état 4 et le terminal = : on peut en effet décaler ($d\,9$), mais aussi réduire ($r\,D \to G$) car le terminal = appartient à $Suivant(D)$ ($\{=, \$\}$). Cette grammaire n'est donc pas SLR(1).

Analyse LR(1)

On enrichit les règles pointées avec des terminaux qui permettent de prédire quand effectuer une réduction plutôt qu'un décalage.

Automates LR(1)

Les états d'un automate LR(1) sont des ensembles de règles pointées *étendues*. Une règle pointée étendue est une "règle pointée associée à un symbole terminal a" ($a \in \Sigma \cup \{\$\}$) et se note $X \to \alpha \bullet \beta, a$. Si dans un même état, on a $X \to \alpha \bullet \beta, a$ et $X \to \alpha \bullet \beta, b$, on notera de manière plus concise $X \to \alpha \bullet \beta, a \mid b$.

Remarque : si un état contient une règle pointée complète $X \to \gamma\bullet, a \mid b \mid c$, alors $\{a, b, c\} \subseteq \mathit{Suivant}(X)$, mais il est possible que ce ne soit qu'un sous-ensemble strict.

Saturation d'un ensemble de règles pointées étendues : un ensemble de règles pointées étendues E est dit *saturé* si $X \to \beta \bullet Y\gamma, a$ appartient à E et $Y \in V$ alors pour toute règle $Y \to \alpha$ de la grammaire, $Y \to \bullet\alpha, b$ avec $b \in \mathit{Premier}(\gamma a)$ appartient à E.
Pour un ensemble de règles pointées étendues E, on note $\mathit{Saturation}(E)$ le plus petit ensemble saturé E' contenant E.

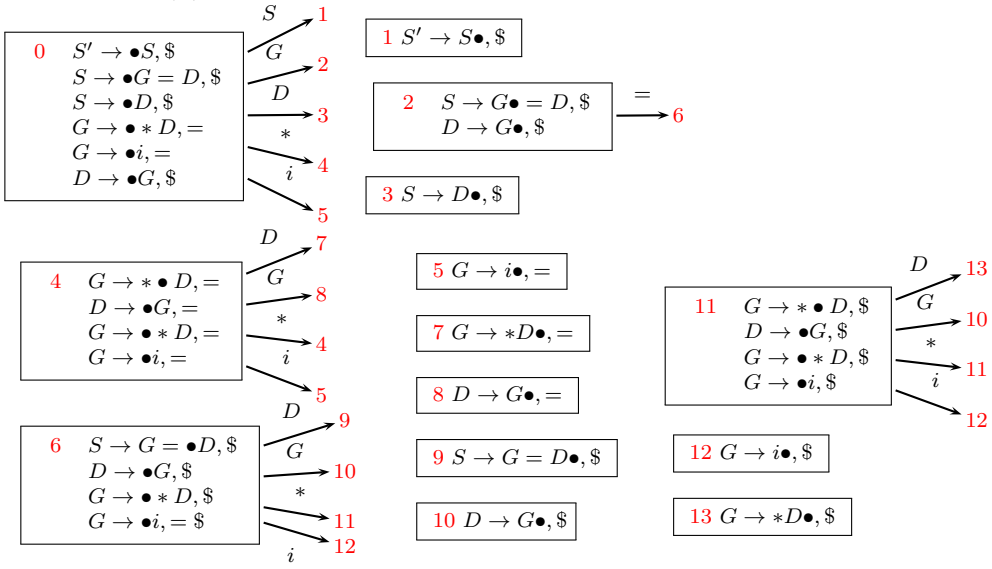
Exercice 3. Calculer $\mathit{Saturation}(\{S' \to \bullet S, \$\})$ pour la grammaire de l'exercice 2.

Etats de l'automate LR(1) :
Les *états* Q de l'automate LR(1) seront des ensembles saturés de règles pointées étendues. Ils sont souvent appelés *collections LR(1)* (ou *LR(1)-items*). L' *état initial* de l'automate est $\mathit{Saturation}(\{S' \to \bullet S, \$\})$.

Transitions de l'automate LR(1)
Pour un ensemble saturé de règles pointées étendues E et $x \in \Sigma \cup V$, on définit l'état $\delta(E, x)$ comme

$$\mathit{Saturation}(\{X \to \beta x \bullet \gamma, a \mid X \to \beta \bullet x\gamma, a \in E\})$$

Automate LR(1) pour la grammaire de l'exercice 2



Remplissage des tables à partir de l'automate LR(1)
Table *Successeur* : pour tout $q \in Q$ et $X \in V$, si $\delta(q, X) = q'$ alors mettre q' dans la case (q, X) (identique aux méthodes LR(0) et SLR(1)).

Table *Action* :

- pour tout $a \in \Sigma$, $q \in Q$, si $\delta(q, a) = q'$ alors mettre "$d\,q'$" dans la case (q, a)
- pour tout $a \in (\Sigma \cup \{\$\})$, $q \in Q$, si q contient une règle pointée complète $X \to \beta\bullet, a$ avec $X \neq S'$, alors mettre "$r\,X \to \beta$" dans la case (q, a)
- mettre "accepter" dans la case $(q, \$)$ où q est l'état contenant $S' \to S\bullet, \$$.
- mettre "erreur" dans toutes les cases encore vide.

Table Action pour la grammaire de l'exercice 2

	$*$	i	$=$	\$
0	$d\,4$	$d\,5$		
1				accepter
2			$d\,6$	$r\,D \to G$
3				$r\,S \to D$
4	$d\,4$	$d\,5$		
5			$r\,G \to i$	
6	$d\,11$	$d\,12$		

	$*$	i	$=$	\$
7			$r\,G \to *D$	
8			$r\,D \to G$	
9				$r\,S \to G = D$
10				$r\,D \to G$
11	$d\,11$	$d\,12$		
12				$r\,G \to i$
13				$r\,G \to *D$

L'automate LR(0) comportait 10 états, l'automate LR(1) comporte 14 états. De manière générale, les automates LR(1) peuvent comporter des milliers d'états (produit cartésien) pour des langages de programmation classiques.

Automates LALR(1)

Dans l'automate LR(1) décrit précédemment, il existe des états qui ne diffèrent d'autres états que par les symboles de prévision associés aux règles pointées. Lorsque deux états de l'automate LR(1) ne diffèrent que sur les caractères de prévision, on dit qu'ils ont le même *cœur LR(0)*. L'idée de la méthode LALR(1) est de fusionner les états LR(1) ayant le même cœur LR(0) en créant un état dont les règles sont l'union des règles des deux états fusionnés. Par exemple,

4 $G \to * \bullet D, =\mid \$$; $D \to \bullet G, =\mid \$$; $G \to \bullet * D, =\mid \$$; $G \to \bullet i, =\mid \$$

et

11 $G \to * \bullet D, \$$; $D \to \bullet G, \$$; $G \to \bullet * D, \$$; $G \to \bullet i, \$$

fusionnent en

4/11 $G \to * \bullet D, =\mid \$$; $D \to \bullet G, =\mid \$$; $G \to \bullet * D, =\mid \$$; $G \to \bullet i, =\mid \$$

On construit l'automate LALR(1) à partir de l'automate LR(1) en fusionnant les états ayant le même cœur LR(0). Les transitions de l'automate sont simplement "mises à jour" en fonction des fusions effectuées. Remarquez que dans l'automate LR(1), les cœurs LR(0) des états sont précisément les états de l'automates LR(0). Donc l'automate LALR(1) a exactement le même nombre d'états que l'automate LR(0).

Le remplissage des tables *Action* et *Suivant* LALR(1) se fait comme le remplissage des tables LR(1) mais à partir de l'automate LALR(1). Si la table *Action* ne comporte qu'une action par case, alors la grammaire est LALR(1). Voici quelques remarques à noter :

- toute grammaire LALR(1) est LR(1).
- la fusion des états peut créer des conflits qui n'existaient pas dans l'automate LR(1)
  $\Rightarrow$ Il existe des grammaires LR(1) qui ne sont pas LALR(1).
- les grammaires LR(1) et donc, LALR(1) ne sont pas ambigües.

Automate LALR(1) pour la grammaire de l'exercice 2

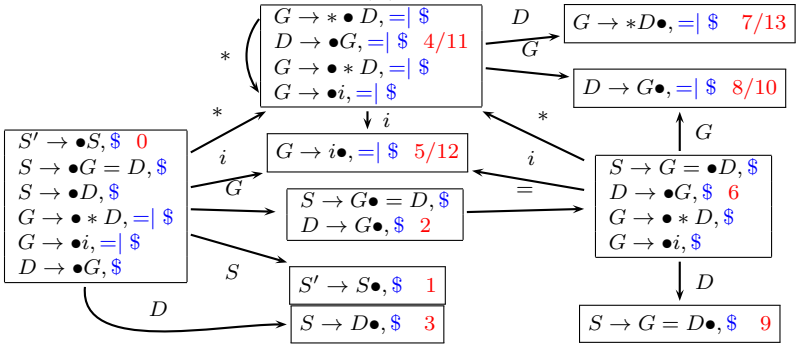


Table Action LALR(1)

	$*$	i	$=$	\$
0	$d\,4/11$	$d\,5/12$		
1				*accepter*
2			$d\,6$	$r\,D \to G$
3				$r\,S \to D$
4/11	$d\,4/11$	$d\,5/12$		
5/12			$r\,G \to i$	$r\,G \to i$
6	$d\,4/11$	$d\,5/12$		
7/13			$r\,G \to *D$	$r\,G \to *D$
8/10			$r\,D \to G$	$r\,D \to G$
9				$r\,S \to G = D$