

Analyse ascendante 1

Mirabelle Nebut

Bureau 223 - extension M3
`mirabelle.nebut at lifl.fr`

2009-2010

Introduction

Analyseurs LR(0)

Principes

Construction de l'automate LR-AFD

Tables d'analyse LR(0)

Analyseurs SLR(1)

Analyseurs LR(1)

Analyseur ascendant

- ▶ Effectue des **lectures** et des **réductions** ;
- ▶ construit un arbre en ordre **postfixe** ;
- ▶ en partant du mot à reconnaître ;
- ▶ construction d'une **dérivation droite** ;
- ▶ analyseurs LR(k) (from **L**eft to **R**igth, **R**igth derivation).

On parle aussi :

- ▶ d'analyse par **décalage et réduction**
- ▶ ***shift/reduce analysis***.

Exemple

$$S \rightarrow AD \mid B$$

$$A \rightarrow aAb \mid b$$

$$B \rightarrow aB \mid c$$

$$D \rightarrow e$$

Cette grammaire n'est pas LL(k) (pourquoi ?)

On va la traiter en LR(k), avec $k=0$ pour commencer.

Exemple

Analyseur LR(0) basé sur une variante de l'**automate des items**.

(vous vous rappelez?)

⇒ nouvel axiome S'

⇒ production $S' \rightarrow S$

$$S' \rightarrow S$$

$$S \rightarrow AD \mid B$$

$$A \rightarrow aAb \mid b$$

$$B \rightarrow aB \mid c$$

$$D \rightarrow e$$

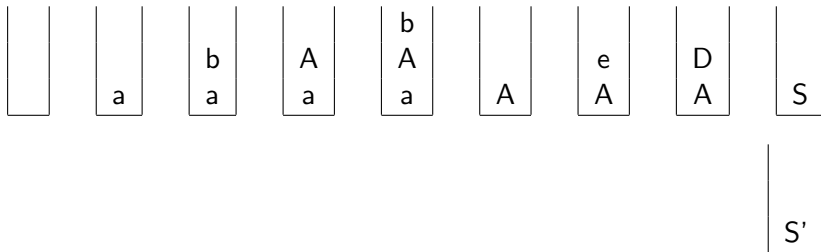
Exemple de reconnaissance

$$\begin{aligned}S' &\rightarrow S \\S &\rightarrow AD \mid B \\A &\rightarrow aAb \mid b \\B &\rightarrow aB \mid c \\D &\rightarrow e\end{aligned}$$

Reconnaître le mot *abbe#* ?

On essaie **intuitivement**, avec une pile contenant des mots de $(V_T \cup V_N)^*$.

Exemple de reconnaissance



- ▶ Construction arbre ordre **postfixe** : lectures et réductions ;
- ▶ Dérivation **droite**.

Analyse ascendante : défis

Contenu de la pile :

- ▶ mot de $(V_N \cup V_T)^*$;
- ▶ début de la dérivation droite construite ;
- ▶ **préfixe viable**.

Comment choisir de manière déterministe :

- ▶ entre lecture et réduction ;
- ▶ quelle partie du sommet de pile réduire ? (= le **manche**)
- ▶ par quelle production réduire.

Comment faire ?

Analyse ascendante : solutions

On repart de l'automate des items.

On explicite l'automate fini sous-jacent :

- ▶ **automate caractéristique** (un état = un item) ;
- ▶ on comprend comment l'analyse ascendante fonctionne avec ;
- ▶ mais cet automate est non déterministe.

On le détermine.

Et c'est gagné ! On a un **analyseur LR(0)**.

Introduction

Analyseurs LR(0)

Principes

Construction de l'automate LR-AFD

Tables d'analyse LR(0)

Analyseurs SLR(1)

Analyseurs LR(1)

Retour à l'automate des items

Trois types de transitions :

- ▶ lecture de a :

$$([X \rightarrow \alpha \bullet a \beta] , a) \rightarrow [X \rightarrow \alpha a \bullet \beta]$$

- ▶ expansion par $Y \rightarrow \gamma$:

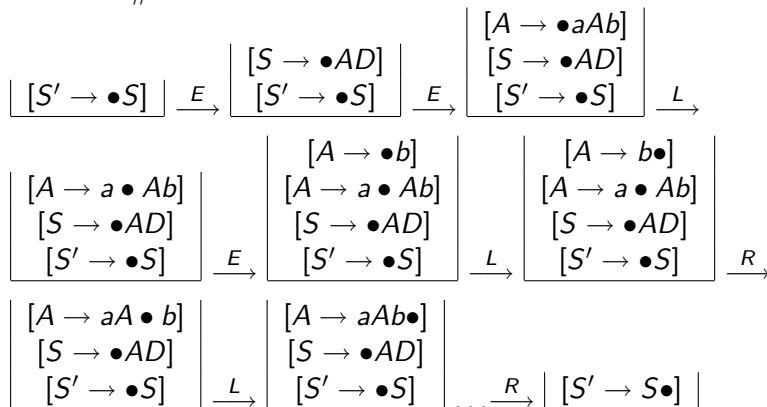
$$([X \rightarrow \alpha \bullet Y \beta] , \epsilon) \rightarrow [X \rightarrow \alpha \bullet Y \beta] [Y \rightarrow \bullet \gamma]$$

- ▶ réduction par $Y \rightarrow \gamma$:

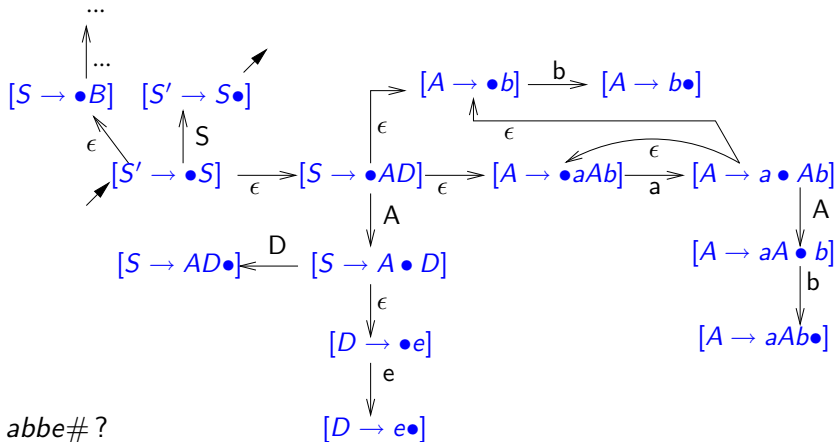
$$([X \rightarrow \alpha \bullet Y \beta] [Y \rightarrow \gamma \bullet] , \epsilon) \rightarrow [X \rightarrow \alpha Y \bullet \beta]$$

Retour à l'automate des items - exemple

Ex : *abbe#* ?



Automate caractéristique - exemple



Automate caractéristique - généralités

Automate à nombre fini d'états :

- ▶ sous-jacent à l'automate des items ;
- ▶ indique son état courant ;
- ▶ = l'item en sommet de pile.

Pour chaque transition de l'aut des items, l'aut caractéristique :

- ▶ effectue une transition ;
- ▶ ou, depuis un état puit, «revient en arrière».

Comment se comporte l'automate caractéristique ?

Automate caractéristique et lecture

Idem automate des items.

V_T -transition sur le terminal lu :

$$[X \rightarrow \alpha \bullet a \beta] \xrightarrow{a} [X \rightarrow \alpha a \bullet \beta]$$

Ex lecture de a :

$$[A \rightarrow \bullet a Ab] \xrightarrow{a} [A \rightarrow a \bullet Ab]$$

Automate caractéristique et expansion

Idem automate des items.

Expansion par $Y \rightarrow \gamma$: ϵ -transition

$$[X \rightarrow \alpha \bullet Y \beta] \xrightarrow{\epsilon} [Y \rightarrow \bullet \gamma]$$

Ex expansion par $A \rightarrow b$:

$$[A \rightarrow a \bullet Ab] \xrightarrow{\epsilon} [A \rightarrow \bullet b]$$

Automate caractéristique et réduction

Différent de l'automate des items (ϵ -production).

Conséquence d'une **réduction par** $A \in V_N$: **V_N -transition** sur A

Ex : on réduit par $A \rightarrow aAb$:

- ▶ quand on est dans l'**état puit** $[A \rightarrow aAb\bullet]$;
- ▶ alors on **rebrousse chemin** des 4 transitions qui ont amené dans cet état :
 - ▶ les 3 transitions qui correspondent à aAb ;
 - ▶ l' ϵ -transition qui correspond à l'expansion par $A \rightarrow aAb$;
- ▶ et on **transite sur** A (A -transition, on a reconnu un A).

Automate caractéristique et réduction par une production vide

Cas particulier, on réduit par $X \rightarrow \epsilon$:

- ▶ dans l'état puit $X \rightarrow \bullet$;
- ▶ on rebrousse chemin d'une transition ($|\epsilon| + 1 = 1$) ;
- ▶ et on transite sur X .

Déterminiser l'automate caractéristique

L'automate caractéristique :

- ▶ est non déterministe (des ϵ -transitions) ;
- ▶ contient des expansions (justement les ϵ -transitions).

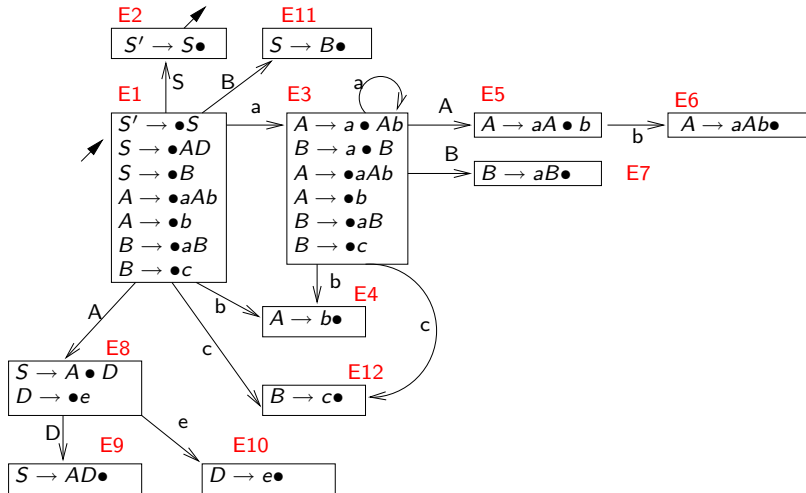
On veut un analyseur ascendant :

- ▶ déterministe ;
- ▶ sans expansions explicites (lectures et réductions).

⇒ on détermine l'automate caractéristique.

⇒ on obtient un automate dit **LR-AFD**.

Automate LR-AFD, exemple



Automate LR(0)

L'automate LR-AFD décrit un **automate à pile déterministe** appelé automate **LR(0)** effectuant 2 types d'actions :

- ▶ lecture
- ▶ réduction

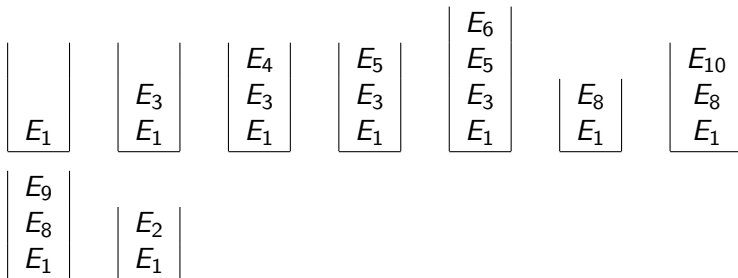
Lecture de a : dans un **état contenant** $X \rightarrow \dots \bullet a \dots$

Réduction par $X \rightarrow \alpha \bullet$: dans un **état contenant** $X \rightarrow \alpha \bullet$.

La pile permet de mémoriser les états parcourus lors des lectures et des réductions.

Exemple de fonctionnement

abbe# ?



On a ce qu'on voulait :

- ▶ l'arbre en ordre postfixe, et la dérivation droite ;
- ▶ avec des lectures et des réductions.

Reste à formaliser.

Définition de l'automate LR(0)

Un état est un ensemble d'item : si Q est l'ensemble des états

$$Q \subseteq \mathcal{P}(It_G)$$

L'alphabet de pile est Q .

L'état initial q_0 :

- ▶ contient l'item initial de la forme $[S' \rightarrow \bullet S]$;
- ▶ sert à initialiser la pile.

L'état final q_f contient l'item final, de la forme $[S' \rightarrow S \bullet]$.

Définition de l'automate LR(0) : relation de transition

On note δ la relation de transition de l'AF LR-AFD.

$\delta(q, X) = q'$ signifie :

- ▶ si l'état courant est q ;
- ▶ et que $X \in V_T \cup V_N$ est le symbole courant à traiter ;
- ▶ alors l'état courant devient q' .

Exemple :

- ▶ $\delta(E_1, S) = E_2$ ou $E_1 \xrightarrow{S} E_2$
- ▶ $\delta(E_1, b) = E_4$ ou $E_1 \xrightarrow{b} E_2$

Définition de l'automate LR(0) : relation de transition

Lecture

Relation de transition de l'automate LR(0) pour une **lecture** :

$$(q, a) \rightarrow q\delta(q, a)$$

- ▶ si q est en sommet de pile
- ▶ si a est sous la tête de lecture
- ▶ et l'un des items de q est de la forme $[X \rightarrow \dots \bullet a \dots]$;
- ▶ alors on empile l'état successeur de q pour a dans δ .

Définition de l'automate LR(0) : relation de transition

Réduction

Relation de transition de l'automate LR(0) pour une **réduction** :

$$(q q_1 \dots q_n, \epsilon) \rightarrow q \delta(q, X)$$

- ▶ si q_n est en sommet de pile ;
- ▶ si l'un des items de q_n est de la forme $[X \rightarrow \alpha \bullet]$, $|\alpha| = n$;
- ▶ alors on dépile n états ;
- ▶ puis on empile $\delta(q, X)$ le successeur par X de l'état q en sommet de pile.

Et les expansions ?

Les ϵ -transition d'expansion ont disparu avec la détermination.
 Elles se font implicitement à l'intérieur des états.

E1
$[S' \rightarrow \bullet S]$
$[S \rightarrow \bullet AD]$
$[S \rightarrow \bullet B]$
$[A \rightarrow \bullet aAb]$
$[A \rightarrow \bullet b]$
$[B \rightarrow \bullet aB]$
$[B \rightarrow \bullet c]$

\xrightarrow{c} **E12**

lecture de c possible après expansions successives par :

- ▶ $S' \rightarrow S \rightsquigarrow [S' \rightarrow \bullet S] \in E1$
- ▶ $S \rightarrow B \rightsquigarrow [S \rightarrow \bullet B] \in E1$
- ▶ $B \rightarrow c \rightsquigarrow [B \rightarrow \bullet c] \in E1$

Introduction

Analyseurs LR(0)

Principes

Construction de l'automate LR-AFD

Tables d'analyse LR(0)

Analyseurs SLR(1)

Analyseurs LR(1)

Construction de LR-AFD - en première approche

On construit Q (les états) et δ (les transitions) de l'automate caractéristique à partir de la grammaire.

On le détermine, on obtient LR-AFD.

En fait, on peut construire **directement** LR-AFD (ouf!).

Construction algorithmique directe de LR-AFD

Principe :

- ▶ on **sature** les états par **expansion** ;
- ▶ on transite sur chaque symbole Y tel que $[\dots \rightarrow \dots \bullet Y \dots]$.

Saturation des états par expansion

Un ensemble d'items E est **saturé** si :

- ▶ pour tout item $[X \rightarrow \alpha \bullet Y\beta]$ de E , $Y \in V_N$;
- ▶ pour toute production $Y \rightarrow \gamma$ de G de membre gauche Y ;
- ▶ l'item $[Y \rightarrow \bullet\gamma]$ appartient aussi à E .

On en déduit la fonction Saturation pour une grammaire G :

fonction Saturation (s : ensemble d'items) :

```
    ensemble d'items
// calcule tous les items d'un état de LR-AFD( $G$ )
// contenant au moins les items de  $s$ 
// retourne cet état.
```

Algorithme de construction de Q et δ

L'état initial est $\text{Saturation}([S' \rightarrow \bullet S])$.

Ensuite, pour chaque état saturé E et chaque symbole $Y \in V_T \cup V_N$ (lecture pour V_T , réduction pour V_N) :

- ▶ si E contient un ensemble de n items de la forme « $\bullet Y$ » :

$$\{ [X \rightarrow \alpha_i \bullet Y \beta_i] \mid 1 \leq i \leq n \}$$

- ▶ alors on calcule

$$E' = \text{Saturation}(\{ [X \rightarrow \alpha_i Y \bullet \beta_i] \mid 1 \leq i \leq n \})$$

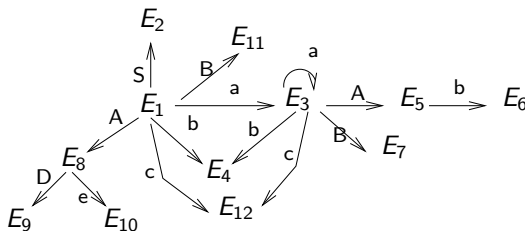
- ▶ si cet état E' n'existe pas, on l'ajoute à Q ;
- ▶ et on définit $\delta(E, Y) = E'$.

Exemple et remarque

Pour ne pas manquer de place sur sa feuille : séparer le contenu des états et la relation de transition.

E1	$[S' \rightarrow \bullet S]$
	$[S \rightarrow \bullet AD]$
	$[S \rightarrow \bullet B]$
	$[A \rightarrow \bullet aAb]$
	$[A \rightarrow \bullet b]$
	$[B \rightarrow \bullet aB]$
	$[B \rightarrow \bullet c]$

E3	$[A \rightarrow a \bullet Ab]$
	$[B \rightarrow a \bullet B]$
	$[A \rightarrow \bullet aAb]$
	$[A \rightarrow \bullet b]$
	$[B \rightarrow \bullet aB]$
	$[B \rightarrow \bullet c]$



Conflits LR(0), grammaire LR(0)

L'automate LR(0) construit peut ne pas être déterministe (2 cas).

État autorisant 2 réductions (ou plus) :

conflit LR(0) **reduce/reduce**

Ex :

$[A \rightarrow b\bullet]$
$[B \rightarrow b\bullet]$

État autorisant 1 réduction et 1 lecture (ou plus) :

conflit LR(0) **shift/reduce**

Ex :

$[A \rightarrow \bullet b]$
$[B \rightarrow c\bullet]$

Conflits LR(0), grammaire LR(0)

Une grammaire est dite **LR(0)** si aucun de ses états ne contient de conflit LR(0) :

- ▶ ni shift-reduce
- ▶ ni reduce-reduce

Les conflits shift/shift n'existent pas (aucun sens).

Remarque - CUP

On comprend mieux les messages d'erreurs de CUP, notamment en cas de grammaire ambiguë.

```
[java] Warning : *** Shift/Reduce conflict found
[java] in state #60
[java]   between expr ::= expr MOINS expr (*)
[java]   and      expr ::= expr (*) MOINS expr
[java]   under symbol MOINS
[java]   Resolved in favor of shifting.
```

Introduction

Analyseurs LR(0)

Principes

Construction de l'automate LR-AFD

Tables d'analyse LR(0)

Analyseurs SLR(1)

Analyseurs LR(1)

Tables d'un analyseur LR(0)

Un analyseur LR(0) est défini par 2 tables :

- ▶ la **table des successeurs** ;
- ▶ la **table des actions**.

Table des successeurs LR(0)

Encode la relation de transition δ de LR-AFD :

$$Q \times (V_T \cup V_N) \rightarrow Q$$

Exemple, table des successeurs

Pour tout $q \in Q$ et $X \in V_T \cup V_N$:

si $\delta(q, X) = q'$ alors mettre q' dans la case (q, X)

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	E_{11}	E_{12}
a	E_3		E_3									
b	E_4		E_4		E_6							
c	E_{12}		E_{12}									
e								E_{10}				
S'												
S	E_2											
A	E_8		E_5									
B	E_{11}		E_7									
D								E_9				

Table des actions LR(0)

Indique quelle action effectuer :

- ▶ dans un état $q \in Q$;
- ▶ si $x \in V_T \cup \{\#\}$ est sous la tête de lecture.

$Q \times (V_T \cup \{\#\}) \rightarrow$ ensemble d'actions

Une action peut être :

- ▶ la lecture du terminal x (**decale**) ;
- ▶ la réduction par une production p (**red** par p) ;
- ▶ l'acceptation (**acc**).

Exemple, table des actions

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	E_{11}	E_{12}
a	d	e	d	red $A \rightarrow b$	e	red $A \rightarrow aBb$	red $B \rightarrow aB$	e	red $S \rightarrow AD$	red $D \rightarrow e$	red $S \rightarrow B$	red $B \rightarrow c$
b	d	e	d	red $A \rightarrow b$	d	red $A \rightarrow aBb$	red $B \rightarrow aB$	e	red $S \rightarrow AD$	red $D \rightarrow e$	red $S \rightarrow B$	red $B \rightarrow c$
c	d	e	d	red $A \rightarrow b$	e	red $A \rightarrow aBb$	red $B \rightarrow aB$	e	red $S \rightarrow AD$	red $D \rightarrow e$	red $S \rightarrow B$	red $B \rightarrow c$
e	e	e	e	red $A \rightarrow b$	e	red $A \rightarrow aBb$	red $B \rightarrow aB$	d	red $S \rightarrow AD$	red $D \rightarrow e$	red $S \rightarrow B$	red $B \rightarrow c$
#	e	a	e	red $A \rightarrow b$	e	red $A \rightarrow aBb$	red $B \rightarrow aB$	e	red $S \rightarrow AD$	red $D \rightarrow e$	red $S \rightarrow B$	red $B \rightarrow c$

a : acceptation, d : décale, e : erreur, red : réduction par p

Table des actions, remplissage

Pour tout $a \in V_T$ et $q \in Q$:

si q contient un item de la forme $[X \rightarrow \dots \bullet a \dots]$
alors mettre **decale** dans la case (q, a)

Pour tout $q \in Q$, $Q \neq q_f$:

- ▶ si q contient un item terminal de la forme $[X \rightarrow \alpha \bullet]$;
- ▶ alors, pour tout $a \in V_T \cup \{\#\}$, mettre **réduction** $X \rightarrow \alpha$ dans la case (q, a) .

Mettre **acceptation** dans la case $(q_f, \#)$.

Mettre **erreur** dans les cases encore vides.

Table des actions, remarque

Pour un automate LR(0), cas dégénéré pour le remplissage de la table par une réduction.

$k=0$: aucun symbole de prédiction (pas de *Premier*, *Suivant*).

Une réduction est effectuée quelque soit la tête de lecture.

⇒ colonnes remplies de la même réduction.

Le cas général est : pour tout $a \in V_T \cup \{\#\}$ et $q \in Q$:

- ▶ si q contient un item terminal de la forme $X \rightarrow \alpha \bullet$ et que la réduction peut se faire **avec a sous la tête de lecture** ;
- ▶ alors, mettre **réduction $X \rightarrow \alpha$** dans la case (q, a) .

Caractérisation d'une grammaire LR(0)

Une grammaire est LR(0) si sa table des actions contient pour chaque case :

- ▶ une seule action
- ▶ ou erreur.

Exemple, table des actions et conflits

E

$[A \rightarrow b\bullet]$
$[B \rightarrow b\bullet]$

	E	...
c	red $A \rightarrow b$ red $B \rightarrow b$	
b	red $A \rightarrow b$ red $B \rightarrow b$	
...		

E

$[A \rightarrow \bullet b]$
$[B \rightarrow c\bullet]$

	E	...
c	red $B \rightarrow c$	
b	red $B \rightarrow c$ decale	
...		

Quand une grammaire n'est pas LR(0)

C'est peut-être à cause du 0.

On peut essayer une analyse LR(1) : beaucoup plus puissante.

C'est plus facile d'expliquer d'abord les grammaires SLR(1) :
Simple LR(1).

Exemple

Soit la grammaire $S' \rightarrow S, S \rightarrow a \mid \epsilon$.

Conflit shift/reduce dans l'état initial (lire a , réduire par $S \rightarrow \epsilon$) :

$S' \rightarrow \bullet S$
$S \rightarrow \bullet a$
$S \rightarrow \bullet$

Mais si la tête de lecture est :

- ▶ dans $\{a\}$, alors lire a ;
- ▶ dans $\{\#\} = \text{Suivant}(S)$ alors réduire par $S \rightarrow \epsilon$.

Un automate SLR(1) exploite cette information.

Introduction

Analyseurs LR(0)

Principes

Construction de l'automate LR-AFD

Tables d'analyse LR(0)

Analyseurs SLR(1)

Analyseurs LR(1)

Principe : $k=1$ et exploitation des *Suivant*

Un analyseur SLR(1) prend en compte le symbole sous la tête de lecture ($k=1$, cf LL(1)) pour **décider d'une réduction** :

Réduction par $X \rightarrow \alpha$ seulement si **tête lecture** \in *Suivant*(X)

Repose comme l'analyse LR(0) sur l'automate LR-AFD.

Permet d'arbitrer certains conflits LR(0) S/R et R/R.

Conflicts shift/reduce au sens SLR(1)

Un état de LR-AFD provoque un **conflit S/R** au sens **SLR(1)** s'il contient à la fois :

- ▶ un item de la forme $[Y \rightarrow \dots \bullet a \dots]$
- ▶ un item de la forme $[X \rightarrow \alpha \bullet]$ avec $a \in \text{Suivant}(X)$

Comparer avec LR(0) : conflit S/R au sens LR(0) si l'état contient les items $[Y \rightarrow \dots \bullet a \dots]$ et $[X \rightarrow \alpha \bullet]$

Conflits reduce/reduce au sens SLR(1)

Un état de LR-AFD provoque un **conflit R/R** au sens **SLR(1)** s'il contient à la fois :

- ▶ un item de la forme $[Y \rightarrow \beta \bullet]$
- ▶ un item de la forme $[X \rightarrow \alpha \bullet]$
- ▶ avec $Suivant(X) \cap Suivant(Y) \neq \emptyset$

Comparer avec LR(0) : conflit R/R au sens LR(0) si l'état contient les items $[Y \rightarrow \beta \bullet]$ et $[X \rightarrow \alpha \bullet]$

Grammaire SLR(1)

Une grammaire est dite **SLR(1)** si l'automate LR-AFD ne contient pas de conflits au sens SLR(1).

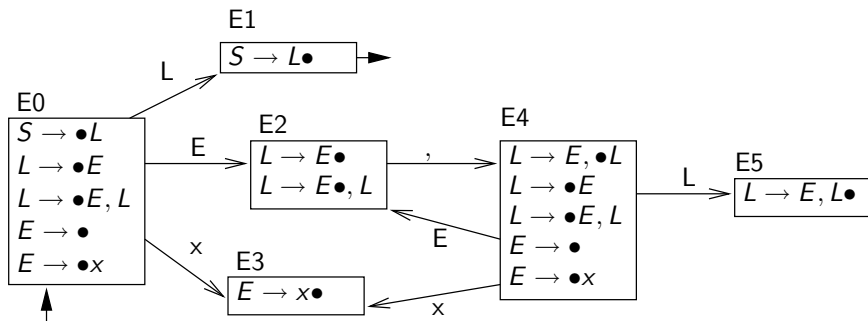
Grammaire SLR(1), exemple

Listes de x séparés par $,$ et à trou

$$\begin{aligned}L &\rightarrow E \mid E, L \\ E &\rightarrow \epsilon \mid x\end{aligned}$$

$$S \rightarrow L$$

Exemple : automate LR-AFD



Exemple : conflits au sens LR(0)

- ▶ $E0$: conflit S/R entre lire x et réduire par $E \rightarrow \epsilon$;
- ▶ $E2$: conflit S/R entre lire $" , "$ et réduire par $L \rightarrow E$;
- ▶ $E4$: conflit S/R entre lire x et réduire par $E \rightarrow \epsilon$.

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

Pour savoir si ce sont des conflits au sens SLR(1) : calcul des *Suivant*.

- ▶ $Suivant(S) = \{\#\}$;
- ▶ $Suivant(L) = Suivant(S) = \{\#\}$;
- ▶ $Suivant(E) = Suivant(L) \cup \{", " \} = \{", " , \#\}$.

Exemple : conflits au sens SLR(1)

- ▶ $E0 : x \notin \text{Suivant}(E)$ donc pas de conflit entre lire x et réduire par $E \rightarrow \epsilon$;
- ▶ $E2 : ", " \notin \text{Suivant}(L)$ donc pas de conflit entre lire $,$ et réduire par $L \rightarrow E$;
- ▶ $E4$: idem $E0$.

La grammaire est donc SLR(1).

Construction de la table des actions SLR(1)

Pour tout $a \in V_T$ et $q \in Q$:

si q contient un item de la forme $[X \rightarrow \dots \bullet a \dots]$
alors mettre **decale** dans la case (q, a)

Pour tout $q \in Q$, $q \neq q_f$ et tout $a \in V_T \cup \{\#\}$:

- ▶ si q contient un item terminal de la forme $X \rightarrow \alpha \bullet$;
- ▶ alors, si $a \in \text{Suivant}(X)$, mettre **réduction** $X \rightarrow \alpha$ dans la case (q, a) .

Mettre **acceptation** dans la case $(q_f, \#)$.

Mettre **erreur** dans les cases encore vides.

Exemple : table des actions SLR(1)

$Suivant(S) = \{\#\}$ $Suivant(L) = \{\#\}$

$Suivant(E) = \{",", \#\}$

	$E0$	$E1$	$E2$	$E3$	$E4$	$E5$
x	decale	erreur	erreur	erreur	decale	erreur
$,$	red $E \rightarrow \epsilon$	erreur	decale	red $E \rightarrow x$	red $E \rightarrow \epsilon$	erreur
$\#$	red $E \rightarrow \epsilon$	accepte	red $L \rightarrow E$	red $E \rightarrow x$	red $E \rightarrow \epsilon$	red $L \rightarrow E, L$

Caractérisation d'une grammaire SLR(1)

La grammaire est **SLR(1)** si sa table des actions contient pour chaque case :

- ▶ une seule action
- ▶ ou erreur.

Remarques

Une grammaire LR(0) ou SLR(1) n'est pas ambiguë.

Une grammaire ambiguë n'est ni LR(0) ni SLR(1).

Comparaison SLR(1) - LR(0)

Méthode SLR(1) basée comme LR(0) sur l'automate LR-AFD :

- ▶ les tables des successeurs LR(0) et SLR(1) sont identiques ;
- ▶ les tables LR(0) et SLR(1) ont le même encombrement mémoire.

Comparaison SLR(1) - LR(0)

Grâce au $k = 1$:

- ▶ l'analyse SLR(1) est strictement plus puissante que l'analyse LR(0) ;
- ▶ = elle engendre moins de conflits.

$$\text{LR}(0) \subset \text{SLR}(1)$$

Néanmoins beaucoup de grammaires (non ambiguës) ne sont pas SLR(1).

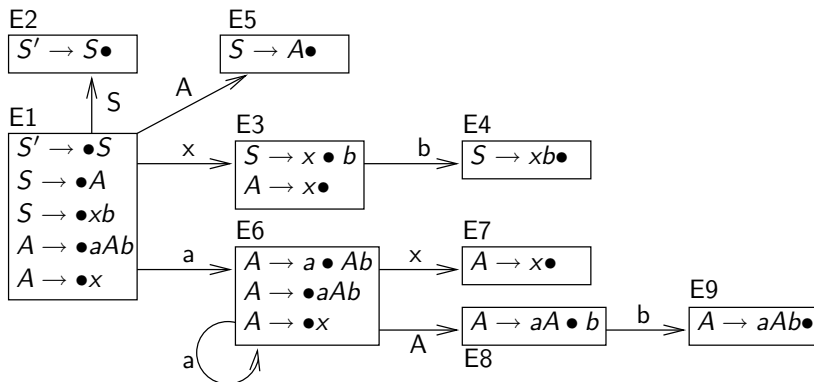
Exemple 1 : G_1

$$\begin{aligned} S &\rightarrow A \mid xb \\ A &\rightarrow aAb \mid x \end{aligned}$$

G_1 grammaire non ambiguë (mais non LL(1)) :

- ▶ si $xb : S \Rightarrow xb$;
- ▶ si $a^nxb^n : S \Rightarrow A \Rightarrow^n a^nAb^n \Rightarrow a^nxb^n$.

Automate LR-AFD de G_1



Conflit pour G_1

L'automate LR-AFD contient un conflit S/R au sens LR(0) dans l'état E3 :

$$\begin{aligned} [S \rightarrow x \bullet b] \\ [A \rightarrow x \bullet] \end{aligned}$$

Pour savoir si c'est un conflit au sens SLR(1), calcul des *Suivant* :

- ▶ $Suivant(S') = Suivant(S) = \{\#\}$;
- ▶ $Suivant(A) = Suivant(S) \cup \{b\} = \{\#, b\}$;

$b \in Suivant(A)$ donc E3 contient un **conflit S/R au sens SLR(1)**.

Conflit SLR(1) pour G_1 : origine

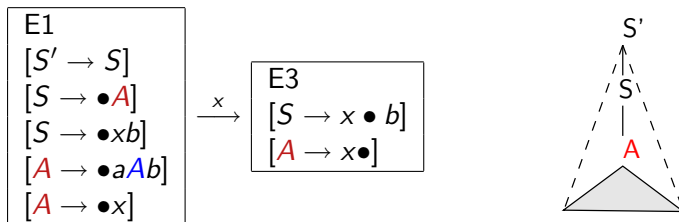
Conflit dans E3 car $b \in \text{Suivant}(A)$. Et pourtant...

... la lecture de b impose la dérivation $S' \Rightarrow S \Rightarrow xb$.

... mais *Suivant* trop imprécis pour le voir.

Comment être plus précis ?

Conflit SLR(1) pour G_1 : solution



Les A de $E1$ et $E3$ ne peuvent être suivis que d'un $\#$, pas d'un b .
 Ce A (suivi par b) n'est pas expansé dans $E1$ et $E3$, mais dans $E6$.
 Si on considère les symboles de $V_T \cup \{\#\}$ qui peuvent suivre A
 dans $E3$, on fait sauter le conflit.

Restriction des symboles de look-ahead

L'analyse LR(1) ne considère pas tous l'ensemble $Suivant(X)$ pour réduire par $X \rightarrow \dots$.

Elle calcule :

- ▶ pour chaque item $[X \rightarrow \alpha]$ d'un état E ;
- ▶ un ensemble $L \subseteq Suivant(X)$;
- ▶ contenant les symboles qui peuvent suivre X dans E .

L peut parfois être égal à $Suivant(X)$.

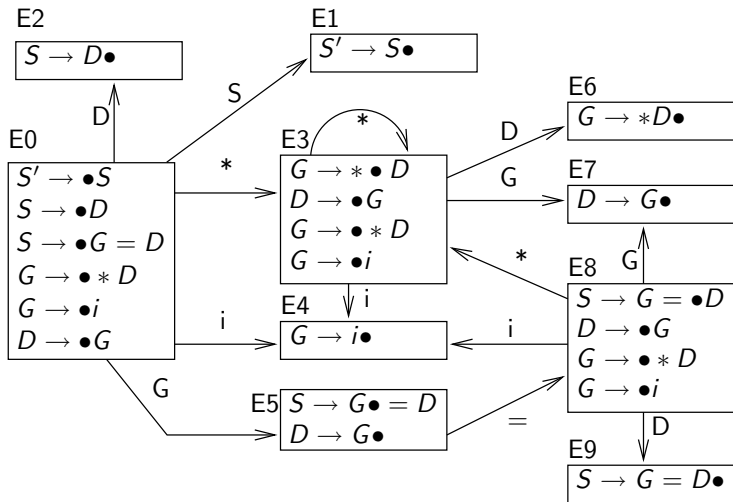
Exemple 2 : G_2

$$\begin{aligned} S &\rightarrow G = D \mid D \\ G &\rightarrow *D \mid i \\ D &\rightarrow G \end{aligned}$$

Grammaire G_2 non ambiguë :

- ▶ la présence ou l'absence du $=$ indique s'il faut choisir $S \rightarrow G = D$ ou $S \rightarrow D$;
- ▶ la grammaire de productions $\{ G \rightarrow *D \mid i, D \rightarrow G \}$ est LL(1).

Automate LR-AFD pour G_2



Conflit pour G_2

L'automate LR-AFD contient un conflit S/R au sens LR(0) dans l'état 5 :

$$\begin{array}{l} E5 \\ [S \rightarrow G\bullet = D] \\ [D \rightarrow G\bullet] \end{array}$$

Pour savoir si c'est un conflit au sens SLR(1), calcul des *Suivant* :

- ▶ $Suivant(S') = Suivant(S) = \{\#\}$;
- ▶ $Suivant(G) = \{=\} \cup Suivant(D)$;
- ▶ $Suivant(D) = Suivant(S) \cup Suivant(G)$;

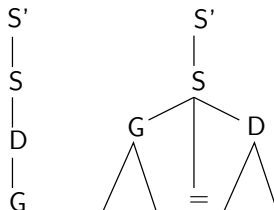
D'où $Suivant(G) = Suivant(D) = \{\#, =\}$.

$\Rightarrow \in Suivant(D)$ donc E_5 contient un **conflit S/R au sens SLR(1)**.

Conflit SLR(1) pour G_2 : origine

Conflit car " $=$ " $\in \text{Suivant}(D)$.

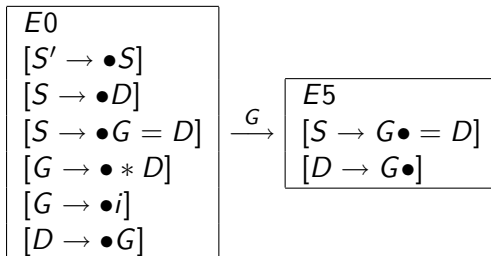
Pourtant il n'existe pas de dérivation t.q. $S \Rightarrow^* w_1 D = w_2$



$\text{Suivant}(D)$ contient ici un " $=$ " jamais rencontré comme look-ahead dans une analyse effective.

Restriction des symboles de look-ahead

Si on particularise les symboles de look-ahead aux états $E0$ et $E5$:



En $E0$ et $E5$, D ne peut être suivi que par $\#$: levée du conflit.

Introduction

Analyseurs LR(0)

Principes

Construction de l'automate LR-AFD

Tables d'analyse LR(0)

Analyseurs SLR(1)

Analyseurs LR(1)

Principe

Enrichissement des items : **items généralisés** de la forme

$$[X \rightarrow \alpha \bullet, L], \text{ avec } L \subseteq V_T \cup \{\#\}$$

Dans $[X \rightarrow \alpha_1 \bullet \alpha_2, L]$, L contient les symboles qui **peuvent suivre** X à ce stade de l'analyse.

Un analyseur LR(1) réduit par $X \rightarrow \alpha$ dans un état E contenant $[X \rightarrow \alpha \bullet, L]$ seulement si le symbole sous la tête de lecture appartient à L .

Remarque : pour $[X \rightarrow \alpha \bullet, L]$, $L \subseteq \text{Suivant}(X)$

Automate LR(1)

La méthode LR(1) ne repose **pas** sur l'automate LR-AFD.

Deux items $[X \rightarrow \alpha \bullet \beta, L]$ et $[X \rightarrow \alpha \bullet \beta, L']$ sont considérés comme différents si $L \neq L'$.

L'automate fini caractéristique d'un analyseur LR(1) (dit **automate LR(1)**) est donc beaucoup plus gros que l'automate LR-AFD, ce qui explique sa plus grande puissance.

Algorithme de construction de l'automate LR(1)

On procède comme pour l'automate LR-AFD :

- ▶ on **sature** les états par **expansion** ;
- ▶ on transite sur chaque symbole Y tel que $[\dots \rightarrow \dots \bullet Y \dots]$

Mais on **modifie** la **saturation** pour calculer L .

Plus facile à expliquer si on décompose $[X \rightarrow \alpha, \{x_1, \dots, x_n\}]$ en un ensemble d'items **généralisés unitaires** :

$$[X \rightarrow \alpha, x_1], \dots, [X \rightarrow \alpha, x_n]$$

Saturation des états LR(1) : intuition

On considère l'item généralisé unitaire $[X \rightarrow \alpha \bullet Y\beta, a]$;

- ▶ on cherche à saturer pour Y : qui peut suivre Y ?
- ▶ au moins les $Premier(\beta)$;
- ▶ mais si $\beta \Rightarrow^* \epsilon$, alors a , qui peut suivre X , peut aussi suivre Y .
- ▶ Donc Y peut être suivi par $Premier(\beta a)$.

Saturation des états LR(1) : définition

Un ensemble d'items généralisés unitaires E est **saturé** si :

- ▶ s'il contient l'item généralisé unitaire $[X \rightarrow \alpha \bullet Y\beta, a]$;
- ▶ alors pour toutes les productions $Y \rightarrow \gamma \in P$,
- ▶ et pour tout $b \in \text{Premier}(\beta a)$,
- ▶ on a $[Y \rightarrow \bullet \gamma, b] \in E$.

En fin de saturation on reconstruit les items généralisés.

Algorithme de construction de Q et δ

L'état initial est $\text{Saturation}([S' \rightarrow \bullet S, \{\# \}])$.

Ensuite, pour chaque état saturé E et chaque symbole $Y \in V_T \cup V_N$ (lecture pour V_T , réduction pour V_N) :

- ▶ si E contient un ensemble de n items enrichis de la forme « $\bullet Y$ » :

$$\{ [X \rightarrow \alpha_i \bullet Y \beta_i, L_i] \mid 1 \leq i \leq n \}$$

- ▶ alors on calcule

$$E' = \text{Saturation}(\{ [X \rightarrow \alpha_i Y \bullet \beta_i, L_i] \mid 1 \leq i \leq n \})$$

- ▶ si cet état E' n'existe pas, on l'ajoute à Q ;
- ▶ et on définit $\delta(E, Y) = E'$.

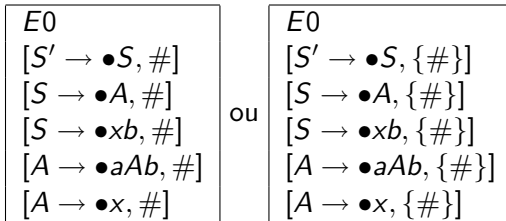
Exemple de G_1

$$\begin{aligned}S' &\rightarrow S \\S &\rightarrow A \mid xb \\A &\rightarrow aAb \mid x\end{aligned}$$

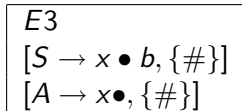
$$\text{Premier}(S) = \text{Premier}(A) \cup \{x\} = \{a, x\}$$

$$\text{Premier}(A) = \{a, x\}$$

État initial de G_1



Transition par x vers $E3 = \text{Saturation}(\begin{array}{|l} [S \rightarrow x \bullet b, \{\#\}] \\ [A \rightarrow x \bullet, \{\#\}] \end{array})$



Conflit au sens LR(1) ?

Conflits au sens LR(1)

Un ensemble d'items généralisés provoque un **conflit S/R** s'il contient à la fois :

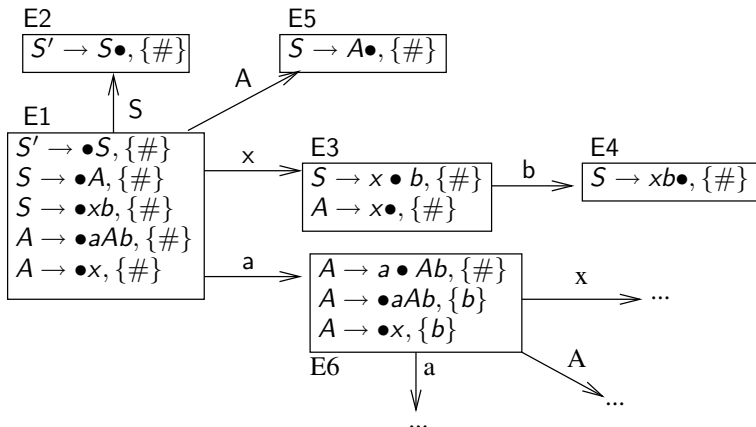
- ▶ un item de la forme $[Y \rightarrow \dots \bullet a \dots, L]$, avec $a \in V_T$;
- ▶ un item de la forme $[X \rightarrow \alpha \bullet, L']$ avec $a \in L'$

Un ensemble d'items généralisés provoque un **conflit R/R** s'il contient à la fois :

- ▶ un item de la forme $[X \rightarrow \alpha \bullet, L]$;
- ▶ un item de la forme $[Y \rightarrow \beta \bullet, L']$ avec $L \cap L' \neq \emptyset$.

\Rightarrow pas de conflit au sens LR(1) en $E3$: G_1 est LR(1).

Automate LR(1) pour G_1 , suite



Automate LR(1) pour G_1 , remarque

L'état $E6$ de LR-AFD :

$[A \rightarrow \bullet Ab]$
$[A \rightarrow \bullet aAb]$
$[A \rightarrow \bullet x]$

a éclaté en deux états LR(1) :

$\begin{array}{l} [A \rightarrow \bullet Ab, \{\#\}] \\ [A \rightarrow \bullet aAb, \{b\}] \\ [A \rightarrow \bullet x, \{b\}] \end{array}$	et	$\begin{array}{l} [A \rightarrow \bullet Ab, \{b\}] \\ [A \rightarrow \bullet aAb, \{b\}] \\ [A \rightarrow \bullet x, \{b\}] \end{array}$
---	----	--

\Rightarrow automate LR(1) plus gros que LR-AFD.

Exemple de G_2

$$\begin{aligned}S' &\rightarrow S \\S &\rightarrow G = D \mid D \\G &\rightarrow *D \mid i \\D &\rightarrow G\end{aligned}$$

État initial LR(1) pour G_2

$[S' \rightarrow \bullet S, \#]$
 $[S \rightarrow \bullet G = D, \#]$
 $[S \rightarrow \bullet D, \#]$
 $[G \rightarrow \bullet * D, =]$
 $[G \rightarrow \bullet i, =]$
 $[D \rightarrow \bullet G, \#]$
 $[G \rightarrow \bullet * D, \#]$
 $[G \rightarrow \bullet i, \#]$

ou

$[S' \rightarrow \bullet S, \{\#\}]$
 $[S \rightarrow \bullet G = D, \{\#\}]$
 $[S \rightarrow \bullet D, \{\#\}]$
 $[G \rightarrow \bullet * D, \{=, \#\}]$
 $[G \rightarrow \bullet i, \{=, \#\}]$
 $[D \rightarrow \bullet G, \{\#\}]$

Automate LR(1) pour G_2

Transition $E0 \xrightarrow{G} E5$:

$E5$ $[S \rightarrow G\bullet = D, \{\#\}]$ $[D \rightarrow G\bullet, \{\#\}]$
--

Conflit S/R levé au sens LR(1) : G_2 est LR(1).

L'automate LR(1) comporte 14 états, contre 10 pour l'automate LR-AFD.

Construction de la table des actions LR(1)

Pour tout $a \in V_T$ et $q \in Q$:

si q contient un item de la forme $[X \rightarrow \dots \bullet a \dots]$
alors mettre **decale** dans la case (q, a)

Pour tout $q \in Q$, $q \neq q_f$ et tout $a \in V_T \cup \{\#\}$:

- ▶ si q contient un item terminal de la forme $[X \rightarrow \alpha \bullet, L]$;
- ▶ alors, si $a \in L$, mettre **réduction** $X \rightarrow \alpha$ dans la case (q, a) .

Mettre **acceptation** dans la case $(q_f, \#)$.

Mettre **erreur** dans les cases encore vides.

Caractérisation d'une grammaire LR(1)

Une grammaire est LR(1) si sa table des actions contient pour chaque case :

- ▶ une seule action
- ▶ ou erreur.

Au delà des grammaires LR(1)

Beaucoup de grammaires sont LR(1).

Mais les tables sont rapidement trop grosses pour tenir en mémoire.

L'analyse utilisée en pratique est l'analyse LALR(1) (Look-Ahead LR(1)), avec :

$$\text{LR}(0) \subseteq \text{SLR}(1) \subseteq \text{LALR}(1) \subseteq \text{LR}(1)$$

L'analyse LALR(1) est un bon compromis entre puissance et encombrement mémoire.

CUP est un analyseur LALR(1).