Grammaires et Analyse Syntaxique - Cours 7
LR(0) et LR(1)

Ralf Treinen

Université Paris Cité

INSTITUT DE RECHERCHE EN INFORMATIQUE FONDAMENTALE

treinen@irif.fr

11 mars 2025

L'automate caractéristique non-déterministe

- Aussi appelé *automate LR(0)* non-déterministe.
- Chaque état de l'automate est un item (production de la grammaire plus position sur le côté droit).
- États initiaux : items pour l'axiome de la grammaire avec la position au début du côté droit.
- États acceptants : items avec position à la fin (items complets).
- Transitions :

$$\begin{array}{lcll} [A \to \alpha.x\beta] & \xrightarrow{x} & [A \to \alpha x.\beta] & (x \in \Sigma \cup N) \\ [A \to \alpha.B\beta] & \xrightarrow{\epsilon} & [B \to .\gamma] & (B \in N) \end{array}$$

où $A \to \alpha x \beta$, $B \to \gamma$ sont des règles de la grammaire.

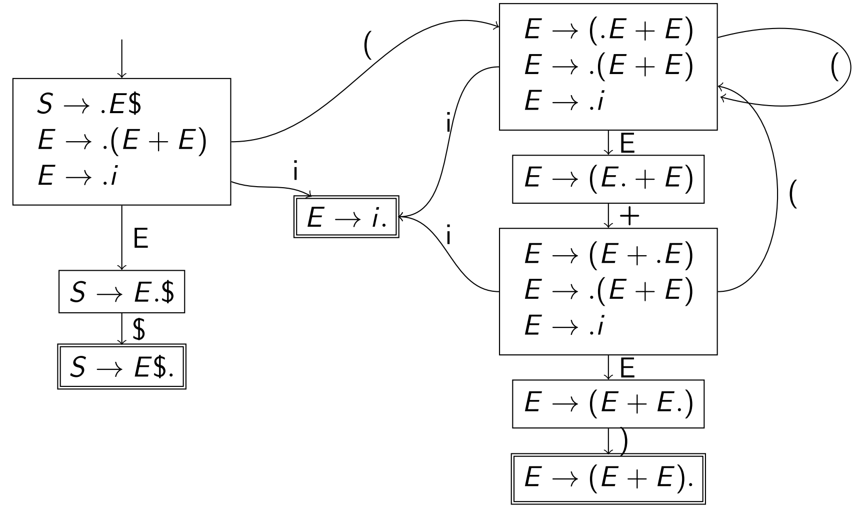
L'automate caractéristique déterministe

- Aussi appelé *automate LR(0)* déterministe.
- État initial : ϵ-clôture de tous les items pour l'axiome avec position au début.
- Transition de P_1 vers P_2 avec symbole $x \in \Sigma \cup N$ quand

$$P_2 = \epsilon\text{-clôture}(\{[A \to \alpha x.\beta] \mid A \to \alpha.x\beta] \in P_1\})$$

- P est un état acceptant quand il contient un item complet.
- P est en conflit *reduce/reduce* LR(0) quand il contient deux items complets.
- P est en conflit *shift/reduce* LR(0) quand il contient un item complet et un item $[A \to \alpha.a\beta]$ avec $a \in \Sigma$.

Sur l'exemple S → E\$ E → (E+E)   E → i



C'est donc bien une grammaire LR(0) !

Rappel : forme requise de la grammaire

- On suppose que la grammaire satisfait ces propriétés :
  - elle est réduite (tous les non-terminaux sont accessibles et productifs) ;
  - tous les côtés droits de productions pour l'axiome ont un terminal à la fin (par exemple $ ou EOF), et l'axiome ne parait pas sur le côté droit d'une règle.

Un exemple d'une grammaire qui est LR(0)

- Règles : $\mathsf{S} \rightarrow \mathsf{A}\ \mathtt{\$} \quad \mathsf{A} \rightarrow \mathtt{c} \mid \mathtt{a}\mathsf{A}\mathtt{b}$
- Axiome : S
- Langage : $\{\mathtt{a}^n\mathtt{c}\mathtt{b}^n\mathtt{\$} \mid n \geq 0\}$
- Construction de l'automate déterministe LR(0) au tableau.

L'algorithme d'analyse grammaticale LR(0)

- Hypothèse : absence de conflit LR(0).
- Initialement on met sur la pile l'état initial qui contient tous les items $[S \rightarrow .\alpha\$]$.
- Tant que le sommet de la pile ne contient pas un item $\{[S \rightarrow \alpha\$.]\}$:
  - Si l'état sur le sommet de la pile
    - contient un item complet $\{[N \rightarrow \alpha.]\}$: faire un **reduce** $\{[N \rightarrow \alpha.]\}$, mettre à jour l'état sur la pile ;
    - contient un item incomplet de la forme $[N \rightarrow \alpha.a\beta]$ avec $a \in \Sigma$: faire un **shift**, mettre à jour l'état sur la pile.
    - dans les autres cas **échec**.
  - **accepter**
- Voir l'exemple au tableau.

Un exemple d'une grammaire qui n'est pas LR(0)

- Règles : $\mathsf{S} \rightarrow \mathsf{A}\ \mathtt{\$} \quad \mathsf{A} \rightarrow \epsilon \mid \mathtt{a}\mathsf{A}\mathtt{b}$
- Axiome : S
- Langage : $\{\mathtt{a}^n\mathtt{b}^n\mathtt{\$} \mid n \geq 0\}$
- Début de la construction de l'automate déterministe LR(0) au tableau.

Les cas d'échec

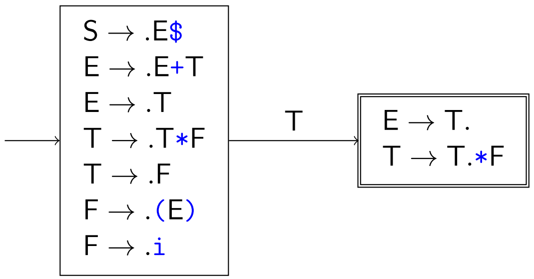
- Il ne faut pas confondre les deux types d'échec qui peuvent se produire à des moments différents :
  1. Notre construction de l'automate caractéristique peut nous amener à un conflit reduce-reduce ou shift-reduce. Dans ce cas la grammaire est rejetée car elle n'est pas LR(0).
  2. Quand notre grammaire est LR(0), l'analyse d'un texte d'entrée peut échouer, c'est le cas précisément quand l'entrée ne correspond pas à la grammaire.
- Ces deux possibilités existent aussi pour l'analyse LL(1).

Retour à l'exemple des expressions arithmétiques avec priorités

- Rappel :
  $S \to E\ \$ \quad E \to E + T \mid T \quad T \to T * F \mid F \quad F \to (\ E\ ) \mid i$
- Est elle LR(0) ?
- Essayons de construire l'automate caractéristique déterministe !
- On commence par l'état initial et on ajoute les autres états au fur et à mesure, mais nous allons pour cet exemple nous arrêter au premier problème rencontré.

Construction de l'automate déterministe

$S \to E\ \$ \quad E \to E + T \mid T \quad T \to T * F \mid F \quad F \to (\ E\ ) \mid i$



- On peut déjà s'arrêter là car on a un *conflit shift-reduce* !
- Si on continue la construction on trouve un deuxième conflit shift-reduce.

Faire un *lookahead*

- La solution dans des cas comme le dernier exemple est de permettre un regard en avant (lookahead).
- Pour cela on va ajouter à un item $[N \to \alpha.\beta]$ un ensemble de symboles qui peuvent suivre à un mot produit à partir de $\alpha\beta$.
- On fait cela en utilisant l'information comment on est arrivé à cet item.
- Le lookahead va être utilisé quand on arrive dans un état acceptant.

Les items LR(1)

- Soit $G = (\Sigma, N, S, P)$ une grammaire.
- Un item LR(1) est une expression de la forme

$$[K \to \alpha.\beta, L]$$

  où
  - $K \to \alpha\beta \in P$ est une règle de la grammaire
  - $L \subseteq \Sigma \cup \{\epsilon\}$
- $[K \to \alpha.\beta]$ est son *noyau*.
- L est son *lookahead*
- La longueur des lookahead est limité à 1, dont le nombre 1 dans "LR(1)".

L'automate non-déterministe LR(1)

- L'automate caractéristique LR(1) est construit très similaire à l'automate pour LR(0), la différence est seulement dans la gestion des lookahead.
- Dans l'automate déterministe on aura une définition de conflit plus fine que pour LR(0) car elle prend aussi en compte le lookahead.
- Ce raffinement de la détection de conflit est la raison pourquoi on passe des LR(0) aux LR(1).
- Nous allons décrire la version non déterministe de cet automate, mais sur l'exemple nous allons construire tout de suite la version déterministe.

L'automate non-déterministe LR(1)

- Grammaire (Σ, N, S, P)
- L'ensemble des états est l'ensemble des items LR(1).
- Les états initiaux sont les items $[S \to .\alpha, \{\epsilon\}]$
- Les états acceptants sont les items LR(1) avec un noyau complet, c-a-d de la forme $[K \to \alpha., L]$
- Transitions par un symbole x $[K \to \alpha.x\beta, L] \overset{x}{\Rightarrow} [K \to \alpha x.\beta, L]$
- Transitions ϵ : $[K \to \alpha.N\beta, L] \overset{\epsilon}{\Rightarrow} [N \to .\gamma, \mathrm{First}_1(\beta L)]$ quand $N \to \gamma \in P$.
- $$\mathrm{First}_1(\beta L) = \begin{cases} \mathrm{First}_1(\beta) & \text{si pas } \beta \to^* \epsilon \\ \mathrm{First}_1(\beta) \cup L & \text{si } \beta \to^* \epsilon \end{cases}$$

L'automate déterministe LR(1)

- En principe obtenu par déterminisation de l'automate non-déterministe LR(1).
- En pratique on construit souvent directement l'automate déterministe, similaire à ce qu'on a vu pour l'automate déterministe LR(0).
- Attention l'automate peut dans le pire des cas être très grand car si Σ a m éléments alors il y a en théorie 2^m possibilités pour L.
- Si on a *dans le même état* deux items $[A \to \alpha.\beta, L_1]$ et $[A \to \alpha.\beta, L_2]$ alors on a le droit de les fusionner : $[A \to \alpha.\beta, L_1 \cup L_2]$.

Exemple d'un automate LR(1)

- Début de la construction de l'automate déterministe LR(1) pour la grammaire des expressions arithmétiques :

  S → E $ E → E + T | T   T → T ∗ F | F   F → (E) | i

- Nous avons vu que cette grammaire n'est pas LR(0), à cause d'un conflit shift-reduce LR(0) :

S → .E$ E → .E+T E → .T T → .T∗F T → .F F → .(E) F → .i	$\xrightarrow{T}$	E → T. T → T.∗F

La taille de l'automate déterministe LR(1)

- S → E $ E → E + T | T   T → T ∗ F | F   F → (E) | i
- Il y 21 items LR(0), 6 symboles terminaux, ça fait
  - $2^6 = 64$ possibilités pour L,
  - en théorie $21 * 64 = 1344$ états dans l'automate LR(1) non-déterministe,
  - en théorie 2^{1344} états dans l'automate LR(1) déterministe.
- On va donc plutôt pas dessiner cet automate avec tous les états, mais on va construire tout de suite l'automate déterministe qui contient seulement les états accessibles à partir de son état initial.

Quel est l'état initial ?

- S → E $ E → E + T | T   T → T ∗ F | F   F → (E) | i
- État initial de l'automate non déterministe : $[S \to .E\$, \{\epsilon\}]$
- Quels états peut on atteindre à partir de là par des ϵ-transitions ?
- $[E \to .E{+}T, \{\$\}]$ car $\mathrm{First}_1(\$\{\epsilon\}) = \{\$\}$
- $[E \to .E{+}T, \{+\}]$ car $\mathrm{First}_1(+\{\$\}) = \{+\}$
- Dans l'automate déterministe on regroupe ces deux en $[E \to .E{+}T, \{\$, +\}]$.
- Et on continue ...

Début de la construction de l'automate déterministe LR(1)

S → .E$, $\{\epsilon\}$ E → .E+T, $\{\$, +\}$ E → .T, $\{\$, +\}$ T → .T∗F, $\{\$, +, *\}$ T → .F, $\{\$, +, *\}$ F → .(E), $\{\$, +, *\}$ F → .i, $\{\$, +, *\}$	$\xrightarrow{T}$	E → T., $\{\$, +\}$ T → T.∗F, $\{\$, +, *\}$

(la construction n'est pas encore terminée)

Conflits reduce-reduce dans le cas LR(1)

- Un ensemble d'items LR(1) a un **conflit reduce-reduce** quand il contient deux items complets

$$[N_1 \to \alpha_1., L_1]$$
$$[N_2 \to \alpha_2., L_2]$$

avec $L_1 \cap L_2 \neq \emptyset$

- Quand il n'y a pas de conflit reduce-reduce et on atteint un état avec des items complets :

$$[N_1 \to \alpha_1., L_1]$$
$$\ldots$$
$$[N_i \to \alpha_i., L_i]$$
$$\ldots$$

alors on fait une action reduce $N_i \to \alpha_i$ quand le symbole suivant de l'entrée appartient à L_i.

Conflits shift-reduce dans le cas LR(1)

- Un ensemble d'items LR(1) a un **conflit shift-reduce** quand il contient deux items

$$[N_1 \to \alpha_1., L_1]$$
$$[N_2 \to \alpha_2.c\beta_2, L_2]$$

avec $c \in L_1$.

- Quand il n'y a pas de conflit shift-reduce et on est dans un état qui contient un item

$$[N_2 \to \alpha_2.c\beta_2, L_2]$$

et le symbole suivant de l'entrée est c alors on fait un shift.

Les grammaires **LR(1)**

- Une grammaire est LR(1) quand son automate LR(1) déterministe n'a pas de conflits LR(1) (ni shift-reduce, ni reduce-reduce)

Sur l'exemple

- Dans notre exemple on a obtenu pour LR(0).

$$\{[E \to T.], [T \to T. * F]\}$$
$$\{[E \to E + T.], [T \to T. * F]\}$$

- Quand on continue la construction de l'automate pour LR(1) on obtient les états

$$\{[E \to T., \{\$, +\}], [T \to T. * F, \{\$, +, *\}]\}$$
$$\{[E \to E + T., \{\$, +\}], [T \to T. * F, \{\$, +, *\}]\}$$

et il n'y a pas de conflits LR(1) car $* \notin \{\$, +\}$.

- Cette grammaire est donc LR(1) mais elle n'est pas LR(0).

Retour à un exemple du début du cours

- Règles : S $\rightarrow$ A \$ A $\rightarrow \epsilon$ | aAb
- Axiome : S
- Langage : $\{a^n b^n \$ \mid n \geq 0\}$
- Nous avons déjà vu que cette grammaire n'est pas LR(0).
- Construction de l'automate déterministe LR(1) au tableau.

Classes de grammaires vues

- au cours 3 : les grammaires *non ambiguës* (tout mot du langage a un arbre de dérivation unique) et les *ambiguës* (les autres).
- aux cours 4 et 5 : les grammaires LL(1) : analyse *descendante* déterministe avec lookahead 1
- aux cours 6 et 7 : les grammaires LR(0) et LR(1) : analyse *ascendante* déterministe avec lookahead 0, resp. 1
- Quelle est la relation entre toutes ces classes ?

Relation entre LR(0) et LR(1)

- Toute grammaire LR(0) est évidemment aussi LR(1) : s'il n'y a pas de conflit dans l'automate LR(0) il n'y a évidement pas non plus de conflit dans l'automate LR(1).
- Nous avons déjà vu un exemple d'une grammaire qui est LR(1) mais pas LR(0) :

$$
\begin{array}{lcl}
S & \rightarrow & E\ \$ \\
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
F & \rightarrow & (\ E\ ) \mid i
\end{array}
$$

Relation entre LR(1) et les ambiguës

- Nous avons déjà vu que toutes les grammaires LL(1), LR(0), LR(1) sont non ambiguës car il y a un algorithme déterministe pour construire un unique arbre de dérivation.
- Exemple d'une grammaire qui est non ambiguë, mais pas LR(1) (et par conséquent pas LR(0)) :

$$S' \rightarrow S\$ \quad S \rightarrow a \mid a\ S\ a$$

- Le langage engendré est $\{a^{2n+1}\$ \mid n \geq 0\}$, c'est donc même un langage rationnel.

Pourquoi cette grammaire est non ambiguë

- Grammaire : S' → S\$ S → a | a S a
- Chaque mot $a^{2n+1}\$$ a une seule dérivation :

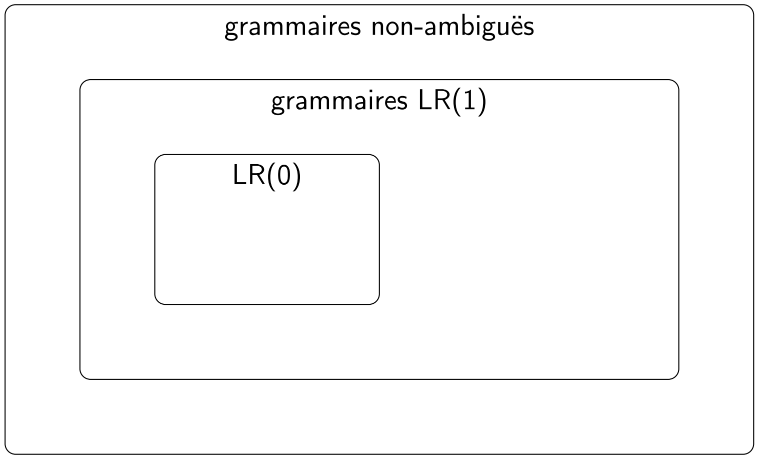
$$S' \rightarrow S\$ \rightarrow aSa\$ \rightarrow \cdots \rightarrow a^nSa^n\$ \rightarrow a^naa^n\$ = a^{2n+1}\$$$

- et par conséquent un seul arbre de dérivation.

Pourquoi cette grammaire n'est pas LR(1)

- Il faut regarder l'automate LR(1) : voir la construction au tableau jusqu'au conflit shift/reduce rencontré.
- Il y a une raison intuitive :
  - Un algorithme ascendant doit commencer par empiler les a.
  - Puis il "décide" qu'il vient de voir le a du milieu du mot, le réduit vers S, puis alterne des shift avec des reduce.
  - Or, l'algorithme n'a aucun moyen pour savoir quand il est au milieu.

Schéma : grammaires non-ambiguës, LR(0) et LR(1)



Relation entre LL(1) et LR(1)

- Il y a un théorème qui dit que toute grammaire qui est LL(1) est aussi LR(1). Nous n'allons pas prouver ce théorème içi.
- Nous avons déjà vu un exemple d'une grammaire qui est LR(1) mais pas LL(1) : la grammaire pour les expressions arithmétiques avec priorités.

Relation entre LL(1) et LR(0)

- Exemple d'une grammaire qui est LR(0) et (donc aussi LR(1)) mais pas LL(1) :

$$E \rightarrow \texttt{(}E\texttt{+}E\texttt{)} \mid \texttt{(}E\texttt{*}E\texttt{)} \mid \texttt{i}$$

- On a vu un exemple similaire au cours 6 (construction de l'automate LR(0) qui n'a pas de conflit)
- Cette grammaire n'est pas LL(1) car il y a des productions pour E qui commencent avec le même non terminal (.

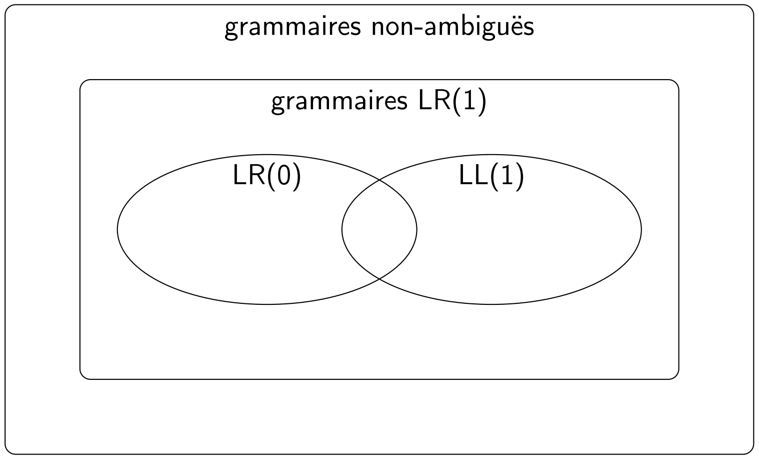
Relation entre LL(1) et LR(0)

- Exemple d'une grammaire qui est LL(1) mais pas LR(0) :
- Axiome : S

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

- Il y a un conflit reduce/reduce dans l'état initial de l'automate LR(0) (voir au tableau).

Schéma : grammaires non-ambiguës, LL(1), LR(0) et LR(1)



Autres approches

- On trouve dans la littérature (en particulier quand elle date un peu) souvent des autres approches qui sont plus fortes que LR(0) mais plus faibles que LR(1) : SLR(1) et LALR(1).
- Dans SLR(1) et LALR(1) on garde un seul item LR(1) pour chaque noyau, il y a donc autant d'états que dans l'automate LR(0).
- L'inconvénient est qu'on perd, comparé à LR(1) en précision car il y a une analyse moins fine des lookahead. Il est donc possible que SLR(1) ou LALR(1) n'arrivent pas à résoudre un conflit là où LR(1) réussit.
- Aujourd'hui, les ordinateurs sont bien capables de construire des automates LR(1) même si c'est un peu pénible à la main, les restrictions SLR(1) et LALR(1) ont donc un peu perdu l'intérêt.

Conclusion sur LR(0) et LR(1)

- Nous avons vu la construction d'analyseurs grammaticales LR(1).
- La construction de l'automate caractéristique est, pour des grammaires réalistes, fastidieuse quand on la fait à la main.
- Il nous faut donc un *générateur* qui prend une grammaire en entrée et qui produit le module qui fait l'analyse grammaticale, similaire à ce qu'on a vu en L2 pour l'analyse lexicale.
- Le générateur échoue quand la grammaire n'est pas LR(1). Dans ce cas il faut comprendre les conflits indiqués par le générateur, et les résoudre.
- Présentation du générateur d'analyse grammaticale `menhir` la semaine prochaine.