Analyse lexicale et syntaxique

Text $\longrightarrow$ Arbre de dérivation

- Application pratique des langages réguliers et algébriques
- En anglais: parsing
- Tester si un texte correspond à une grammaire

Grammaire G:

$$E \to \textbf{int} \mid E + E \mid E * E \mid (\ E\ )$$

Tester si la séquence `10␣*␣(6␣+␣5)` est engendré par la grammaire.

Exemple

Grammaire G:

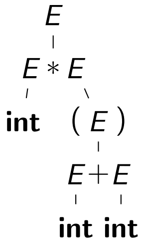
$$E \to \textbf{int} \mid E + E \mid E * E \mid (\ E\ )$$

Tester si la séquence `10␣*␣(6␣+␣5)` est engendré par la grammaire.

`10␣*␣(6␣+␣5)`

$\downarrow$ analyse lexicale

$\textbf{int}\ *\ (\ \textbf{int}\ +\ \textbf{int}\ )$ $\xrightarrow{\text{analyse syntaxique}}$



Analyse lexicale et syntaxique

Résumé

- Application pratique des langages réguliers et algébriques (parsing)
- Analyse lexicale: diviser un texte en terminaux
- Analyse syntaxique: en créer un arbre syntaxique
- Applications: compilateurs ; toute situaton où vos applications doivent prendre en compte des données complexes
- En pratique : les terminaux et sommets de l'arbre sont décorés avec des informations supplémentaires (**int** avec valeur de l'entier).
- Outils: lex/flex pour analyse lexicale, yacc/bison pour analyse syntaxique

Analyse lexicale

Rappel: Diviser texte en terminaux

Certains terminaux sont de seuls symboles ou séquences fixes:

- Opérateurs `+`, `*`, `<=`
- Mots clés: `if`, `while`

D'autres possèdent une forme variable:

- Nom de variable (**id**) : séquence alphanumérique non-vide
- Entiers (**int**) : séquence non-vide de chiffres, ou encore `0x` suivi de caractères hexadécimaux

En pratique:

- Tout terminal représenté par une expression regulière
- P.ex. $R_{\textbf{int}} = \{\mathtt{0}, \ldots, \mathtt{9}\}^+ + 0x \cdot \{\mathtt{0}, \ldots, \mathtt{9}, \mathtt{a}, \ldots, \mathtt{f}\}^+$

Résolution d'ambiguïté

Longueur d'un terminal

Une séquence `10` peut correspondre à une ou deux instances de $R_{\mathbf{int}}$. Par convention, on consomme la partie la plus longue correspondant à un terminal. Dans l'exemple, `10` devient un seul terminal **int**.

Autre exemple: `0x9e` devient un seul **int** (et non **int id int id**)

Priorité

Si `if` est un mot-clé, la séquence `if` correspond à ce mot-clé, mais aussi à un **id**. Par convention, on ordonne les expressions par priorité décroissante.

Rappels

Grammaire algébrique

$G = \langle \Sigma, V, P, S' \rangle$, avec

- Σ alphabet de *terminaux*
- V ensemble de *variables*
- P ensemble de *productions*
- S' variable de départ

On suppose que S' n'apparaît que dans une seule production $P_0 := S' \to S$.

Connaissances préalables :

- dérivation (de gauche / de droit)
- arbre de dérivation
- grammaire non-ambigue
- algorithme de Cooke-Younger-Kasami

Objectif

Algorithme de CYK : temps cubique (mais fonctionne pour toutes les grammaires)

Notre objectif

Identifier une class de grammaires non-ambigues qu'on peut analyser en temps linéaire.

Objectif

Algorithme de CYK : temps cubique (mais fonctionne pour toutes les grammaires)

Notre objectif

Identifier une class de grammaires non-ambigues qu'on peut analyser en temps linéaire.

Analyse top-down:

Données: $w \in \Sigma^*$. Initialement $\gamma := S'$. Deux actions:

- Gonfler: Si $\gamma = X\delta$ pour variable X, choisir une production on $X \to \alpha$ et mettre $\gamma := \alpha\delta$.
- Consommer: Si $\gamma = a\delta$ pour un terminal a, et a est le prochain terminal dans w, consommer a et mettre $\gamma := \delta$.

Accepter si $\gamma = \varepsilon$ à la fin de w. (Exemple: Grammaire G)

Objectif

Algorithme de CYK : temps cubique (mais fonctionne pour toutes les grammaires)

Notre objectif

Identifier une class de grammaires non-ambigues qu'on peut analyser en temps linéaire.

Analyse bottom-up:

Données: $w \in \Sigma^*$. Initialement $\gamma := \varepsilon$. Deux actions:

- Empiler (Shift): Si a est le prochain terminal, mettre $\gamma := \gamma a$.
- Réduire (Reduce): Si $\gamma = \delta\alpha$ et il existe une production $X \to \alpha$, mettre $\gamma := \delta X$.

Accepter si $\gamma = S'$ à la fin de w. (Exemple: Grammaire G)

Nous allons traiter l'approche bottom-up.

AAP régulier

L'analyse bottom-up fonctionne comme un AAP (avec alphabet de pile $\Sigma \cup V$) sauf que les réductions traitent plusieurs symbols de pile à la fois. C'est un cas spécial d'un AAP régulier.

On écrit Σ' pour $\Sigma \cup \{\varepsilon\}$.

AAP régulier

$\mathcal{A} = \langle Q, \Sigma, Z, T, q_0, F \rangle$, avec

$$T \subseteq (Rec(Z^*) \times Q \times \Sigma' \times Z \times Q) \cup (Rec(Z^*) \times Q \times \Sigma' \times Q)$$

1. $\langle w, q \rangle \xrightarrow{a} \langle wz, q' \rangle$ si $\langle L, q, a, z, q' \rangle \in T$ et $w \in L$ (push)
2. $\langle wz, q \rangle \xrightarrow{a} \langle w, q' \rangle$ si $\langle L, q, a, q' \rangle \in T$ et $wz \in L$ (pop)

Remarque: Sommet et pile à droite !

AAP régulier $\to$ AAP ordinaire

Soit $\mathcal{A} = \langle Q, \Sigma, Z, T, q_0, F \rangle$ un AAP régulier avec $k := |T|$
et $\forall i : \mathcal{A}_i = \langle Q_i, Z, \delta_i, \iota_i, F_i \rangle$ DCA acceptant les langages dans T.
Définissons:

- $\mathcal{Q} := Q_1 \times \cdots \times Q_k, \quad \iota := \langle \iota_1, \ldots, \iota_k \rangle$
- $\mathcal{F}_i := \{ \langle q_1, \ldots, q_k \rangle \in \mathcal{Q} \mid q_i \in F_i \}$
- $\delta : \mathcal{Q} \times Z \to \mathcal{Q}$ avec $\delta(\langle q_1, \ldots, q_k \rangle, z) := \langle \delta_1(q_1, z), \ldots, \delta_k(q_k, z) \rangle$.

Rappel (de TD): Construction d'un AAP ordinaire simulant $\mathcal{A}$

$\mathcal{A}' := \langle \mathcal{Q} \times Q, \Sigma, \mathcal{Q} \times Z, T', \langle \iota, q_0 \rangle, \mathcal{Q} \times F \rangle$, avec:

- (push) pour tout $\langle L_i, q, a, z, q' \rangle \in T$ et $f \in \mathcal{F}_i$, on a $\langle \langle f, q \rangle, a, \langle f, z \rangle, \langle \delta(f, z), q' \rangle \rangle \in T'$;
- (pop) pour tout $\langle L_i, q, a, q' \rangle \in T$, $z \in Z$, $q'' \in \mathcal{Q}$ et $f \in F_i$, on a $\langle \langle q'', z \rangle, \langle f, q \rangle, a, \langle q'', q' \rangle \rangle \in T'$.

Invariante: Si $\mathcal{A}$ accède à une configuration $\langle z_1 \cdots z_n, q \rangle$, alors $\mathcal{A}'$ accède à $\langle \langle q'_0, z_1 \rangle \cdots \langle q'_{n-1}, z_n \rangle, \langle q'_n, q \rangle \rangle$, avec $q'_0 = \iota$ et $q'_{i+1} = \delta(q'_i, z_{i+1})$ pour $i = 0, \ldots, n-1$.

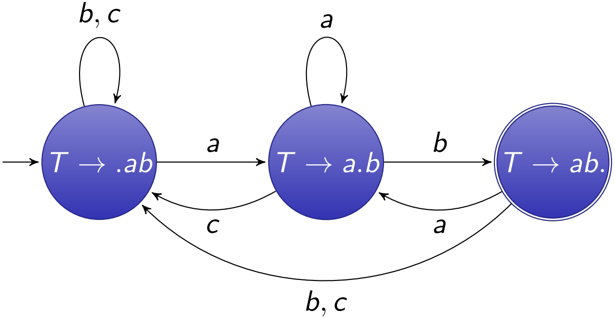
Analyse Shift-Reduce par AAP régulier

Items

Soit $G = \langle \Sigma, V, P, S' \rangle$ une grammaire.

- Les *items* d'une production $X \to \alpha$ sont
$$\mathit{Items}(X \to \alpha) = \{\, X \to \beta.\gamma \mid \alpha = \beta\gamma \,\}$$
- Les *items* de G est l'union des items de ses productions.
- Soit $\mathcal{I}_G := 2^{\mathit{Items}(G)}$; on écrit $\mathcal{I}$ si G est connu.

Automate DC pour $Z^* ab$:



Analyse Shift-Reduce par AAP régulier

Une grammaire G est reconnue par l'AAP régulier $\langle Q, \Sigma, Z, T, q_0, F\rangle$:

- $Q := \{\bot, \top\} \cup \mathit{Items}(G), \quad Z := \Sigma \cup V$
- $q_0 := \bot, \quad F := \{\top\}, \quad T := T_{shift} \cup T_{reduce} \cup T_{accept}$

où T est composé comme suit:

- $T_{shift} = \{ \langle Z^*, \bot, a, a, \bot\rangle \mid a \in \Sigma \}$;
- $$\begin{aligned} T_{reduce} = & \{ \langle Z^*\alpha, \bot, \varepsilon, X \to \alpha.\rangle \mid X \to \alpha \in P \} \\ & \cup \{ \langle Z^*, X \to \alpha z.\beta, \varepsilon, X \to \alpha.z\beta\rangle \mid X \to \alpha\beta \in P \} \\ & \cup \{ \langle Z^*, X \to .\alpha, \varepsilon, X, \bot\rangle \mid X \to \alpha \in P \setminus \{P_0\} \}; \end{aligned}$$
- $T_{accept} = \{\langle \{S\}, \bot, \varepsilon, \top\rangle\}$.

Remarques

Dans l'AAP régulier $\mathcal{A}$:

- Seul T_{reduce} utilise des conditions autre que Z^*.
- L'effet de T_{reduce} est de remplacer α par X dans la pile, seules ces transitions utilisent un état autre que $\perp$.

Dans l'AAP ordinaire $\mathcal{A}'$:

- États $\mathcal{I} \times Q$, alphabet de pile $I \times Z$
- Si on ignore les configurations avec état autre que $\perp$,
  il convient d'écrire une configuration comme un chemin entre éléments de $\mathcal{I}$, liés par lettres de Σ.

Exemple

Grammaire G

$P_0 := S' \to S \quad P_1 := S \to TU \quad P_2 := T \to aTb \quad P_3 := T \to ab \quad P_4 := U \to c$

Calcul acceptant d'un AAP ordinaire sur le mot $w = aabbc$:

Action	reste de w	configuration	états ($I_j := I'_j \cup \{S' \to .S\}$ pour $j = 0, \dots, 7$)
	aabbc	I_0	$I'_0 := \{S \to .TU, T \to .aTb, T \to .ab, U \to .c\}$
shift a	$abbc$	$I_0 \xrightarrow{a} I_1$	$I'_1 := \{S \to .TU, T \to a.Tb, T \to a.b, U \to .c\}$
shift a	bbc	$I_0 \xrightarrow{a} I_1 \xrightarrow{a} I_1$	
shift b	bc	$I_0 \xrightarrow{a} I_1 \xrightarrow{a} I_1 \xrightarrow{b} I_2$	$I'_2 := \{S \to .TU, T \to .aTb, T \to ab., U \to .c\}$
reduce P_3	bc	$I_0 \xrightarrow{a} I_1 \xrightarrow{T} I_3$	$I'_3 := \{S \to T.U, T \to aT.b, T \to .ab, U \to .c\}$
shift b	c	$I_0 \xrightarrow{a} I_1 \xrightarrow{T} I_3 \xrightarrow{b} I_4$	$I'_4 := \{S \to .TU, T \to aTb., T \to .ab, U \to .c\}$
reduce P_2	c	$I_0 \xrightarrow{T} I_5$	$I'_5 := \{S \to T.U, T \to .aTb, T \to .ab, U \to .c\}$
shift c	ε	$I_0 \xrightarrow{T} I_5 \xrightarrow{c} I_6$	$I'_6 := \{S \to .TU, T \to .aTb, T \to .ab, U \to c.\}$
reduce P_4	ε	$I_0 \xrightarrow{T} I_5 \xrightarrow{U} I_7$	$I'_7 := \{S \to TU., T \to .aTb, T \to .ab, U \to .c\}$
reduce P_1	ε	$I_0 \xrightarrow{S} I_8$	$I_8 := \{S' \to S.\}$
accept			

Problèmes

Non-déterminisme

P.ex. $\mathcal{A}'$ peut toujours empiler au lieu d'appliquer une réduction utile (conflit shift/reduce). Ou bien $\mathcal{A}'$ doit décider entre deux réductions différentes (conflit reduce/reduce). P.ex. si $P_3 := T \to \varepsilon$, on peut appliquer cette réduction partout !

$\mathcal{I}$ est trop grand

$\mathcal{I}$ traque tous les α ce qui rend l'automate très grand. Par contre, au début il semble inutile de traquer P_4, et vers la fin, P_2 et P_3 sont inutiles.

Dans la suite, on va adresser ces deux points en même temps:

- Identifier les productions intéressantes à traquer (réduire $\mathcal{I}$).
- Cela réduit les conflits potentiels en même temps !

Analyseurs du type LR

Analyse LR

L : lecture de gauche à droite (*left*)
R : production d'une dérivation de droite, ordre inverse (*right*)

- Exemples: SLR, LR, LALR
- Application du principe bottom-up avec *lookahead*:
  on connaît le(s) prochain(s) caractère(s) du mot sans l'avoir consommé
- On note la taille de lookahead entre parenthèses: p.ex. SLR(k) pour un analyseur SLR avec lookahead de k. On omet k si $k = 1$.
- Différents compromis entre taille de l'analyseur et sa puissance

Définition de First

Soit $k \geq 0$ et $G = \langle \Sigma, V, P, S \rangle$ une grammaire.

- pour $w = a_1 \cdots a_l \in \Sigma^*$, on met $First_k(w) := w$ si $l \leq k$ et $First_k(w) := a_1 \cdots a_k$ sinon.
- pour $L \subseteq \Sigma^*$, on met $First_k(L) := \{\, First_k(w) \mid w \in L \,\}$.
- pour $\alpha \in (\Sigma \cup V)^*$, on met $First_k(\alpha) := First_k(\mathcal{L}_G(\alpha))$.

Autrement dit, $First_k(\alpha)$ est l'ensemble de mots de longueur au plus k qu'on dérive depuis α.

Exemple: $E \to \textbf{int} \mid E + E \mid E * E \mid (\ E\ )$

$$First_2(E) := \{((, (\textbf{int}, \textbf{int}, \textbf{int}+, \textbf{int}*, \textbf{int}\}$$

Définition de Follow

Soit $k \geq 0$, $G = \langle \Sigma, V, P, S \rangle$ une grammaire et $X \in V$

$$Follow_k(X) := \{\, w \in \Sigma^* \mid \exists S' \rightarrow^* \gamma X \delta \wedge w \in First_k(\delta) \,\}$$

Intuitivement, $Follow_k(X)$ contient tous les mots terminaux jusqu'à longueur k qui peuvent suivre une occurrence de variable X dans une dérivation.

Exemple: $S' \rightarrow E,\ \ E \rightarrow \textbf{int} \mid E + E \mid E * E \mid (\ E\ )$

$$Follow_1(E) := \{\varepsilon,), +, *\}$$

Analyseur SLR

SLR = Simple LR, on étudie le cas SLR(1).

Proche à l'AAP précédent, mais identifie les productions 'utiles' à tout moment. Idée: démarrer avec $S' \to .S$, puis clôturer:

Clôture

Soit $I \subseteq Items(G)$. Alors $clot(I)$ est l'ensemble minimal $J \supseteq I$ satisfaisant:

$$\frac{X \to \alpha.Y\beta \in J \qquad Y \in V \qquad Y \to \gamma \in P}{Y \to .\gamma \in J}$$

Exemple: $S' \to S \quad S \to TU \quad T \to aTb \quad T \to ab \quad U \to c$

$$clot(\{S' \to .S\}) = \{S' \to .S, S \to .TU, T \to .aTb, T \to .ab\}$$

Définition de goto

Intuitivement, *goto* joue le rôle de δ dans notre AAP;
on l'applique en empilant un terminal (shift) ou une variable (reduce).

Définition: Soit $I \subseteq \mathit{Items}(G)$ et $z \in \Sigma \cup V$

$$\mathit{goto}(I, z) := \mathit{clot}(\{\, X \to \alpha z.\beta \mid X \to \alpha.z\beta \in I \,\})$$

Exemple: $S' \to S \quad S \to TU \quad T \to aTb \quad T \to ab \quad U \to c$

Soit $J = \mathit{clot}(\{S' \to .S\}) = \{S' \to .S, S \to .TU, T \to .aTb, T \to .ab\}$.

$$\mathit{goto}(J, a) = \{\, T \to a.Tb,\ T \to a.b,\ T \to .aTb,\ T \to .ab \}$$

Construction de l'analyseur SLR

Tableau d'actions

- Les états sont ceux accessible depuis $q_0 := clot(\{S' \to .S\})$ par *goto*.
- Pour tout état q et lookahead $u \in \Sigma' := \Sigma \cup \{\varepsilon\}$, on construit un ensemble $actions(q, u)$.
- G est dite *SLR* si $|actions(q, u)| \leq 1$ pour toute paire q, u.

Actions

- **shift** (s): empiler $u \in \Sigma$ et passer à $goto(q, u)$.
- **reduce** (r_i): pour $P_i = X \to \alpha$, supprimer α dans la pile, revenant sur un état q', puis empiler X et passer à $goto(q', X)$.
- **accept** (a): on a gagné !

Conditions d'inclusion

- $s \in actions(q, u)$ si $u \in \Sigma$ et q contient un item $X \to \alpha.u\beta$;
- $r_i \in actions(q, u)$ si $P_i = X \to \alpha$ et q contient $X \to \alpha.$, $u \in Follow_1(X)$ et $X \neq S'$;
- $a \in actions(q, \varepsilon)$ si q contient $S' \to S.$

Exemple de SLR

$P_0: S' \to S \quad P_1: S \to TU \quad P_2: T \to aTb \quad P_3: T \to ab \quad P_4: U \to c$

$$Follow_1(S) = \{\varepsilon\} \qquad Follow_1(T) = \{b, c\} \qquad Follow_1(U) = \{\varepsilon\}$$

état	détail	goto						actions			
		a	b	c	S	T	U	a	b	c	ε
q_0	$S' \to .S, S \to .TU, T \to .aTb, T \to .ab$	q_1			q_2	q_3		s			
q_1	$T \to a.Tb, T \to a.b, T \to .aTb, T \to .ab$	q_1	q_4			q_5		s	s		
q_2	$S' \to S.$										a
q_3	$S \to T.U, U \to .c$			q_6			q_7			s	
q_4	$T \to ab.$								r_3	r_3	
q_5	$T \to aT.b$		q_8						s		
q_6	$U \to c.$										r_4
q_7	$S \to TU.$										r_1
q_8	$T \to aTb.$								r_2	r_2	

Calcul de l'analyseur pour *aabbc*

q_0

s (shift a)

$q_0 \xrightarrow{a} q_1$

s (shift a)

$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_1$

s (shift b)

$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_1 \xrightarrow{b} q_4$

r_3 (reduce $T \to ab$)

$q_0 \xrightarrow{a} q_1 \xrightarrow{T} q_5$

s (shift b)

$q_0 \xrightarrow{a} q_1 \xrightarrow{T} q_5 \xrightarrow{b} q_8$

r_2 (reduce $T \to aTb$)

$q_0 \xrightarrow{T} q_3$

s (shift c)

$q_0 \xrightarrow{T} q_3 \xrightarrow{c} q_6$

r_4 (reduce $U \to c$)

$q_0 \xrightarrow{T} q_3 \xrightarrow{U} q_7$

r_1 (reduce $S \to TU$)

$q_0 \xrightarrow{S} q_2$

a (accept)

Remarques sur SLR

Un point faible de SLR est la règle de réduction. Rappel:

$$r_i \in actions(q,u) \text{ si } P_i = X \to \alpha \text{ et } q \text{ contient } X \to \alpha.,$$
$$u \in Follow_1(X) \text{ et } X \neq S';$$

Exemple: $P_0 : S' \to S \quad P_1 : S \to TTb \quad P_2 : S \to U \quad P_3 : T \to a \quad P_4 : U \to ab$

- $Follow_1(T) = \{a, b\}$ et $Follow_1(S) = Follow_1(U) = \{\varepsilon\}$
- $q_0 := \{S' \to .S, S \to .TTb, S \to .U, T \to .a, U \to .ab\}$
- $goto(q_0, a) = \{T \to a., U \to a.b\} =: q_1$
- $s \in actions(q_1, b)$ car q_1 contient $U \to a.b$
- $r_3 \in actions(q_1, b)$ car q_1 contient $T \to a.$ et $b \in Follow_1(T)$

Cette grammaire contient un conflit shift/reduce inutile:

- r_3 y est car le second T dans $P_1 = S \to TTb$ est suivi de b.
- Mais dans q_1, le T concerné est le premier de TTb, suivi d'un a.

Analyseur LR(1)

- Idée: éliminer les conflits en limitant les réductions inutiles
- En ajoutant $X \to .\alpha$ à un état, mémoriser une lettre qui peut suivre X.

1-item

Un *1-item* de $G = \langle \Sigma, V, P, S' \rangle$ est une paire $[X \to \beta.\gamma, u]$ t.q. $X \to \beta\gamma \in P$ et $u \in \Sigma^{\leq 1}$. $Items_1(G)$ note les 1-items de G.

Clôture

$clot(I) \subseteq Items_1(G)$ est l'ensemble minimal $J \supseteq I$ satisfaisant:

$$\frac{[X \to \alpha.Y\beta, u] \in J \qquad Y \to \gamma \in P \qquad v \in First_1(\beta u)}{[Y \to .\gamma, v] \in J}$$

Exemple: $P_0: S' \to S \quad P_1: S \to TTb \quad P_2: S \to U \quad P_3: T \to a \quad P_4: U \to ab$

L'état initial est $clot(\{[S' \to .S, \varepsilon]\})$:

$$\{[S' \to .S, \varepsilon], [S \to .TTb, \varepsilon], [S \to .U, \varepsilon], [T \to .a, a], [U \to .ab, \varepsilon]\}$$

Construction d'un analyseur LR(1)

Soit $I \subseteq Items_1(G)$ et $z \in \Sigma \cup V$

$$goto(I, z) := clot(\{\, [X \to \alpha z.\beta, u] \mid [X \to \alpha.z\beta, u] \in I \,\})$$

Actions d'un LR(1)

- $s \in actions(q, u)$ if $u \in \Sigma$ et q contient un item $[X \to \alpha.u\beta, v]$;
- $r_i \in actions(q, u)$ si $P_i = X \to \alpha$ et q contient $[X \to \alpha., u]$ et $X \neq S'$;
- $a \in actions(q, \varepsilon)$ si q contient $[S' \to S., \varepsilon]$

Grammaire LR(1)

G est dite LR(1) si $|actions(q, u)| \leq 1$ pour toute paire q, u.

Exemple de LR(1)

$P_0: S' \to S \quad P_1: S \to TTb \quad P_2: S \to U \quad P_3: T \to a \quad P_4: U \to ab$

état	détail	goto					actions		
		a	b	S	T	U	a	b	ε
q_0	$[S' \to .S, \varepsilon], [S \to .TTb, \varepsilon], [S \to .U, \varepsilon], [T \to .a, a], [U \to .ab, \varepsilon]$	q_1		q_6	q_3	q_4	s		
q_1	$[T \to a., a], [U \to a.b, \varepsilon]$		q_2				r_3	s	
q_2	$[U \to ab., \varepsilon]$								r_4
q_3	$[S \to T.Tb, \varepsilon], [T \to .a, b]$	q_5			q_7		s		
q_4	$[S \to U., \varepsilon]$								r_2
q_5	$[T \to a., b]$							r_3	
q_6	$[S' \to S., \varepsilon]$								a
q_7	$[S \to TT.b, \varepsilon]$		q_8					s	
q_8	$[S \to TTb., \varepsilon]$								r_1

Analyseur LALR

- (analyseur standard réalisé par bison)
- Idée: réduire la consommation mémoire du LR(1)
- Construire d'abord le tableau d'actions LR(1), puis rayer les lookahead dans tous les items, fusionner les états ainsi identiques.

état	détail	goto					actions		
		a	b	S	T	U	a	b	ε
q_0	$[S'\to.S],[S\to.TTb],[S\to.U],[T\to.a],[U\to.ab]$	q_1		q_6	q_3	q_4	s		
q_1	$[T\to a.],[U\to a.b]$		q_2				r_3	s	
q_2	$[U\to ab.]$								r_4
q_3	$[S\to T.Tb],[T\to .a]$	q_5			q_7		s		
q_4	$[S\to U.]$								r_2
q_5	$[T\to a.]$							r_3	
q_6	$[S'\to S.]$								a
q_7	$[S\to TT.b]$		q_8					s	
q_8	$[S\to TTb.]$								r_1

Remarques finales

Analyseur LR(k)

Fonctionne comme LR(1) mais avec des items qui mémorisent k lettres.

Hiérarchie stricte des LR(k)

Pour tout $k \geq 0$, il existe une grammaire LR($k+1$) mais non LR(k):

$$S \to ab^k c \mid Ab^k d, \qquad A \to a$$

Au total, ça nous donne l'hiérarchie suivante entre classes de grammaires :

$$\text{LR}(0) \subseteq \text{SLR} \subseteq \text{LALR} \subseteq \text{LR}(1) \subseteq \text{LR}(2) \subseteq \cdots$$

Par contre, toute grammaire LR(k) est accepté par un AAP déterministe. La traduction d'un AAP en grammaire donne une grammaire LR(1) pour un AAPD. Du coup, tout langage engendré par une LR(k) est aussi engendré par une LR(1).