

Analyse lexicale et syntaxique

Text $\rightarrow$ Arbre de dérivation

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

Grammaire G :

$$E \rightarrow \mathbf{int} \mid E + E \mid E * E \mid (E)$$

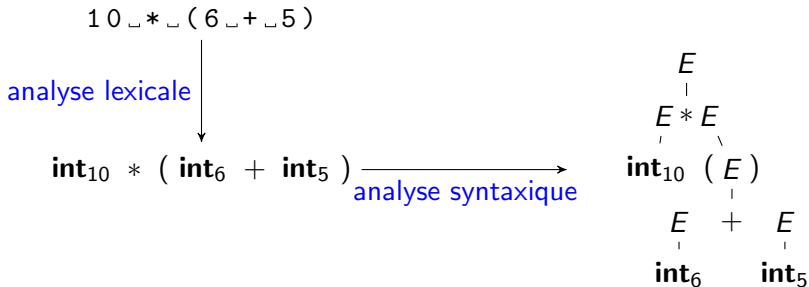
Tester si la séquence $10 * (6 + 5)$ est engendré par la grammaire.

Exemple

Grammaire G :

$$E \rightarrow \text{int} \mid E + E \mid E * E \mid (E)$$

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



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 situation 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 est représenté par une expression régulière.
- ▶ P.ex. $R_{\text{int}} = \{0, \dots, 9\}^+ + 0x \cdot \{0, \dots, 9, a, \dots, f\}^+$

Résolution d'ambiguïté

Longueur d'un terminal

Une séquence 10 peut correspondre à une ou deux instances de R_{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 $P_0 := S' \rightarrow S$.

Connaissances préalables :

- ▶ dérivation (de gauche / de droite)
- ▶ arbre de dérivation
- ▶ grammaire non-ambigue
- ▶ algorithme de *pre* / Cooke-Younger-Kasami

Objectif

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

Notre objectif

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

Objectif

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

Notre objectif

Identifier une classe 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 $X \rightarrow \alpha$ 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 *pre* / CYK : temps cubique (mais fonctionne pour toutes les grammaires)

Notre objectif

Identifier une classe 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 $X \rightarrow \alpha$, 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

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

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 $\rightarrow$ 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:

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

Rappel : 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 $i = 0, \dots, n-1$.

Analyse Shift-Reduce par AAP régulier

Items

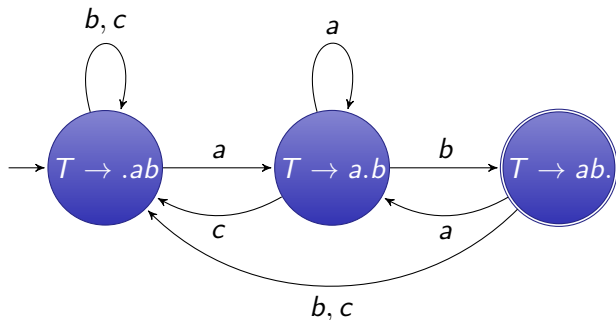
Soit $G = \langle \Sigma, V, P, S' \rangle$ une grammaire.

- ▶ Les *items* d'une production $X \rightarrow \alpha$ sont

$$\text{Items}(X \rightarrow \alpha) = \{ X \rightarrow \beta.\gamma \mid \alpha = \beta\gamma \}$$

- ▶ Les *items* de G est l'union des items de ses productions.
- ▶ Soit $\mathcal{I}_G := 2^{\text{Items}(G)}$; 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 $\langle Q, \Sigma, Z, T, q_0, F \rangle$:

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

où T est composé comme suit:

- ▶ $T_{\text{shift}} = \{ \langle Z^*, \perp, a, a, \perp \rangle \mid a \in \Sigma \};$
- ▶ $T_{\text{reduce}} = \{ \langle Z^* \alpha, \perp, \varepsilon, X \rightarrow \alpha. \rangle \mid X \rightarrow \alpha \in P \}$
 $\cup \{ \langle Z^*, X \rightarrow \alpha z. \beta, \varepsilon, X \rightarrow \alpha. z \beta \rangle \mid X \rightarrow \alpha \beta \in P \}$
 $\cup \{ \langle Z^*, X \rightarrow . \alpha, \varepsilon, X, \perp \rangle \mid X \rightarrow \alpha \in P \setminus \{P_0\} \};$
- ▶ $T_{\text{accept}} = \{ \langle \{S'\}, \perp, \varepsilon, \top \rangle \}.$

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 $\mathcal{I} \times Z$
- ▶ 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

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

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

Action	reste de w	configuration	états ($l_j := I'_j \cup \{S' \rightarrow .S\}$ pour $j = 0, \dots, 7$)
	aabbc	l_0	$I'_0 := \{S \rightarrow .TU, T \rightarrow .aTb, T \rightarrow .ab, U \rightarrow .c\}$
shift a	abbc	$l_0 \xrightarrow{a} l_1$	$I'_1 := \{S \rightarrow .TU, T \rightarrow a.Tb, T \rightarrow a.b, U \rightarrow .c\}$
shift a	bbc	$l_0 \xrightarrow{a} l_1 \xrightarrow{a} l_1$	
shift b	bc	$l_0 \xrightarrow{a} l_1 \xrightarrow{a} l_1 \xrightarrow{b} l_2$	$I'_2 := \{S \rightarrow .TU, T \rightarrow .aTb, T \rightarrow ab., U \rightarrow .c\}$
reduce P_3	bc	$l_0 \xrightarrow{a} l_1 \xrightarrow{T} l_3$	$I'_3 := \{S \rightarrow T.U, T \rightarrow aT.b, T \rightarrow .ab, U \rightarrow .c\}$
shift b	c	$l_0 \xrightarrow{a} l_1 \xrightarrow{T} l_3 \xrightarrow{b} l_4$	$I'_4 := \{S \rightarrow .TU, T \rightarrow aTb., T \rightarrow .ab, U \rightarrow .c\}$
reduce P_2	c	$l_0 \xrightarrow{T} l_5$	$I'_5 := \{S \rightarrow T.U, T \rightarrow .aTb, T \rightarrow .ab, U \rightarrow .c\}$
shift c	ε	$l_0 \xrightarrow{T} l_5 \xrightarrow{c} l_6$	$I'_6 := \{S \rightarrow .TU, T \rightarrow .aTb, T \rightarrow .ab, U \rightarrow .c\}$
reduce P_4	ε	$l_0 \xrightarrow{T} l_5 \xrightarrow{U} l_7$	$I'_7 := \{S \rightarrow TU., T \rightarrow .aTb, T \rightarrow .ab, U \rightarrow .c\}$
reduce P_1	ε	$l_0 \xrightarrow{S} l_8$	$l_8 := \{S' \rightarrow S.\}$
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 $P_3 := T \rightarrow \varepsilon$, 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: $E \rightarrow \mathbf{int} \mid E + E \mid E * E \mid (E)$

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

Définition de Follow

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

$$\text{Follow}_k(X) := \{ w \in \Sigma^* \mid \exists S' \rightarrow^* \gamma X \delta \wedge w \in \text{First}_k(\delta) \}$$

Intuitivement, $\text{Follow}_k(X)$ contient tous les mots terminaux jusqu'à longueur k qui peuvent suivre une occurrence de variable X dans une dérivation.

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

$$\text{Follow}_1(E) := \{ \varepsilon,), +, * \}$$

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 $S' \rightarrow .S$, puis clôturer:

Clôture

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

$$\frac{X \rightarrow \alpha.Y\beta \in J \quad Y \in V \quad Y \rightarrow \gamma \in P}{Y \rightarrow .\gamma \in J}$$

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

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

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 $I \subseteq \text{Items}(G)$ et $z \in \Sigma \cup V$

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

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

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

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

Construction de l'analyseur SLR

Tableau d'actions

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

Actions

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

Conditions d'inclusion

- ▶ $s \in \text{actions}(q, u)$ si $u \in \Sigma$ et q contient un item $X \rightarrow \alpha.u\beta$;
- ▶ $r_i \in \text{actions}(q, u)$ si $P_i = X \rightarrow \alpha$ et q contient $X \rightarrow \alpha.$,
 $u \in \text{Follow}_1(X)$ et $X \neq S'$;
- ▶ $a \in \text{actions}(q, \varepsilon)$ si q contient $S' \rightarrow S$.

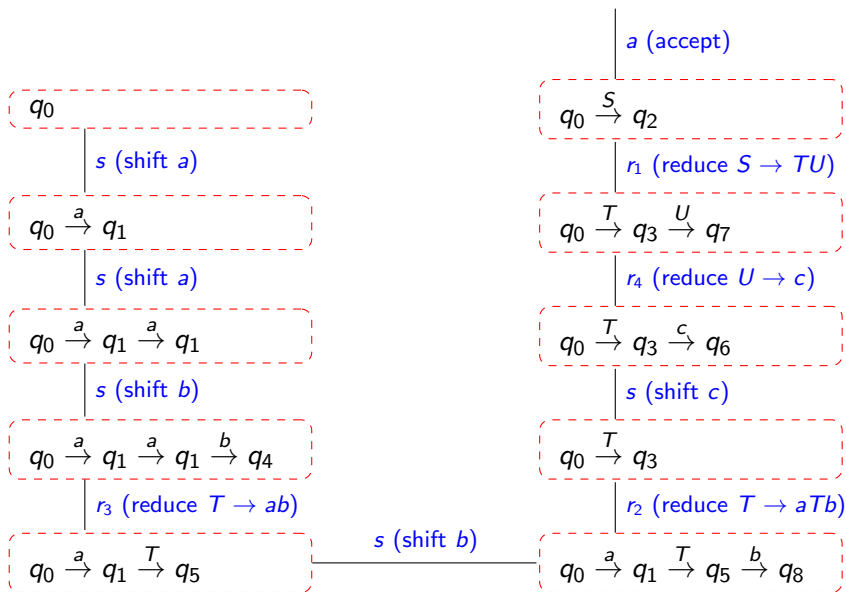
Exemple de SLR

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

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

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

Calcul de l'analyseur pour *aabbcc*



Remarques sur SLR

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

$$r_i \in \text{actions}(q, u) \text{ si } P_i = X \rightarrow \alpha \text{ et } q \text{ contient } X \rightarrow \alpha., \\ u \in \text{Follow}_1(X) \text{ et } X \neq S';$$

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

- ▶ $\text{Follow}_1(T) = \{a, b\}$ et $\text{Follow}_1(S) = \text{Follow}_1(U) = \{\varepsilon\}$
- ▶ $q_0 := \{S' \rightarrow .S, S \rightarrow .TTb, S \rightarrow .U, T \rightarrow .a, U \rightarrow .ab\}$
- ▶ $\text{goto}(q_0, a) = \{T \rightarrow a., U \rightarrow a.b\} =: q_1$
- ▶ $s \in \text{actions}(q_1, b)$ car q_1 contient $U \rightarrow a.b$
- ▶ $r_3 \in \text{actions}(q_1, b)$ car q_1 contient $T \rightarrow a.$ et $b \in \text{Follow}_1(T)$

Cette grammaire contient un conflit shift/reduce inutile:

- ▶ r_3 y est car le second T dans $P_1 = S \rightarrow TTb$ 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 $X \rightarrow \cdot\alpha$ 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 $[X \rightarrow \beta.\gamma, u]$ t.q. $X \rightarrow \beta\gamma \in P$ 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:

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

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

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

$$\{[S' \rightarrow \cdot S, \varepsilon], [S \rightarrow \cdot TTb, \varepsilon], [S \rightarrow \cdot U, \varepsilon], [T \rightarrow \cdot a, a], [U \rightarrow \cdot ab, \varepsilon]\}$$

Construction d'un analyseur LR(1)

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

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

Actions d'un LR(1)

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

Grammaire LR(1)

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

Exemple de LR(1)

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

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

Remarques sur les grammaires LR

Analyseur LR(k)

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

Caractérisation des LR(k)

Une grammaire est LR(k) (ne possède aucun conflit shift/reduce ou reduce/reduce) si et seulement si

$$\left. \begin{array}{l} S \rightarrow_d^* \delta X w \rightarrow_d \delta \alpha w \\ S \rightarrow_d^* \gamma \rightarrow_d \delta \alpha w' \\ First_k(w) = First_k(w') \end{array} \right\} \implies \gamma = \delta X w'$$

pour toute production $X \rightarrow \alpha$, $\delta, \gamma \in (\Sigma \cup V)^*$ et $w, w' \in \Sigma^*$.

Hiérarchie des grammaires

Hiérarchie stricte des $LR(k)$

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

$$S \rightarrow ab^k c \mid Ab^k d, \quad A \rightarrow a$$

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

$$LR(0) \subseteq SLR \subseteq LALR \subseteq LR(1) \subseteq LR(2) \subseteq \dots$$

Les $LR(1)$ acceptent tous langages déterministes

Soit L un langage déterministe. Alors L est engendré par une grammaire $LR(1)$.

Idée: L est accepté par un AAP déterministe tel que tout mot de L possède un seul calcul acceptant (voir la construction pour le complément). La grammaire obtenue par la traduction d'un tel AAPD est $LR(1)$.