Analyse syntaxique

Cyril Rabat
cyril.rabat@univ-reims.fr

Licence 3 Informatique - Info0602 - Langages et compilation

2021-2022

UFR SEN
Sciences Exactes et Naturelles

Université de Reims Champagne-Ardenne

Cours n°3

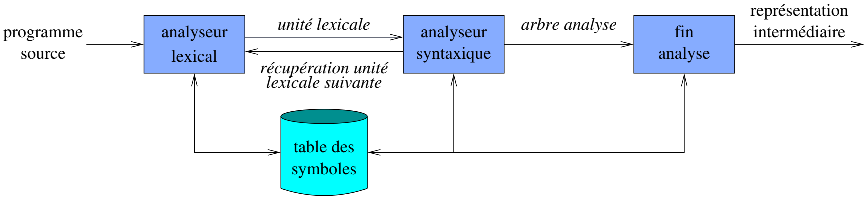
Qu'est-ce qu'un analyseur syntaxique ?
Grammaires

Version 31 janvier 2022

Table des matières

Un analyseur syntaxique

- Récupération des unités lexicales depuis l'analyseur lexical
- Vérification de la structuration
  ↪ Utilisation d'une **grammaire**
- Gestion des erreurs :
  ↪ Signalement + support des erreurs courantes



Une règle de production

- Exemple de la conditionnelle en C :

$$\textbf{if } (\textit{expression}) \textit{ instruction } \textbf{else } \textit{instruction}$$

$\hookrightarrow$ Concaténation du mot-clé **if**, d'une parenthèse ouvrante, d'une expression, d'une parenthèse fermante, *etc.*

- Ce qui peut s'exprimer sous la forme d'une règle :

$$\textit{inst} \rightarrow \textbf{if } (\textit{expr}) \textit{ inst } \textbf{else } \textit{inst}$$

$\hookrightarrow$ C'est une **règle de production**

  - if, (,) sont des unités lexicales appelées **symboles terminaux**
  - *expr*, *inst* sont des suites d'unités lexicales appelées **symboles non terminaux**

Définition d'un langage à l'aide de règles

- Exemple : soit le langage $L = \{a^n b^n \mid n \geq 0\}$
  $\hookrightarrow \epsilon \in L$
  $\hookrightarrow \{a\}L\{b\} \subseteq L$
- Nous pouvons le décrire sous forme de règles : $\begin{array}{lcr} L & \rightarrow & \epsilon \\ L & \rightarrow & aLb \end{array}$

  $\hookrightarrow$ Ce qui est équivalent à $L \rightarrow \epsilon \mid aLb$
- Interprétation :
  $\hookrightarrow L \rightarrow \epsilon \Leftrightarrow \epsilon \in L$
  $\hookrightarrow L \rightarrow aLb \Leftrightarrow \forall w \in L, awb \in L$
- Ces règles forment une **grammaire** et permettent d'engendrer les mots d'un langage

Une grammaire

Définition : grammaire

*Une **grammaire** est un quadruplet $G = (T, N, R, S)$ où :*

- *T est un ensemble de symboles terminaux*
- *N est un ensemble de symboles non terminaux*
- *$R \subseteq ((T \cup N)^+ \times (T \cup N)^*)$ est un ensemble de règles de réécriture ou de production*
- *$S \in N$ est le symbole de départ ou **axiome***

- Les symboles de N n'apparaissent pas dans les mots générés
- Les règles sont de la forme

$$u1 \rightarrow u2 \text{ avec } u1 \in (N \cup T)^+ \text{ et } u2 \in (N \cup T)^*$$

  $\hookrightarrow$ Si $u2 \in T^*$, $u1 \rightarrow u2$ est une **règle terminale**
- C'est à partir de S que la génération de mots commence

Remarques

- Langage défini par une grammaire = ensemble des mots obtenus (dérivés) à partir de l'axiome par application des règles de la grammaire
- Règles de notation :
  - Les symboles terminaux sont représentés par des minuscules
  - Les symboles non-terminaux par des majuscules

Mot reconnu par une grammaire

- Soit G la grammaire suivante :

$$G = (T = \{S\}, N = \{a, b\}, R = \{S \rightarrow \epsilon, S \rightarrow aSb\}, S)$$

- G définit le langage $L = \{a^n b^n \mid n \geq 0\}$
- Le mot *aabb* fait partie du langage L car :
  1. $\underline{S}$
  2. $a\underline{S}b$ ($S \rightarrow aSb$)
  3. $aa\underline{S}bb$ ($S \rightarrow aSb$)
  4. $aabb$ ($S \rightarrow \epsilon$)

Dérivation en une étape

Définition : dérivation (en une étape)

Soit une grammaire $G = (T, N, R, S)$ *et* $u \in (T \cup N)^+$, $v \in (T \cup N)^*$). G *permet de dériver* v *de* u *en une étape (noté* $u \Rightarrow v$*) si et seulement si :*

- $u = xu'y$ *(*u *peut être décomposé en* x, u', y *;* x *et* y *éventuellement vides)*
- $v = xv'y$ *(*v *peut être décomposé en* x, v', y *éventuellement vides)*
- $u' \rightarrow v' \in R$

Dérivation et langage généré

Définition : dérivation

$G = (T, N, R, S)$ permet de dériver v de u (noté $u \Rightarrow^* v$) si et seulement si $\exists k \geq 0$ et $v_0, \ldots, v_{k-1} \in (T \cup N)^+$ et $v_k \in (T \cup N)^*$ tel que :

- $u = v_0$
- $v = v_k$
- $v_i \Rightarrow v_{i+1}$ *pour* $0 \leq i < k$

Définition : langage généré

Le langage $L(G)$ généré par une grammaire $G = (T, N, R, S)$ est l'ensemble des mots qui peuvent être générés par G :

$$L(G) = \{v \in N^* \mid S \Rightarrow^* v\}$$

Classification des grammaires (1/2)

- Classification en fonction de la forme des règles
  $\hookrightarrow$ Définie en 1957 par Noam Chomsky
- **Type 0** : pas de restriction
  $\hookrightarrow$ Règles de la forme $w \rightarrow v$
- **Type 1** : grammaires sensibles au contexte (dites contextuelles)
  $\hookrightarrow$ Règles de la forme $uAv \rightarrow uwv$
- **Type 2** : grammaires hors-contexte
  $\hookrightarrow$ Règles de la forme $A \rightarrow u$
- **Type 3** : grammaires régulières
  - À droite
    $\hookrightarrow$ Règles de la forme $A \rightarrow aB$ ou $A \rightarrow a$
  - À gauche
    $\hookrightarrow$ Règles de la forme $A \rightarrow Ba$ ou $A \rightarrow a$

Avec $A, B \in N, u, v \in (N \cup T)^*, w \in (N \cup T)^+, a \in T^*$

Classification des grammaires (2/2)

- Les grammaires de type 3 génèrent les langages réguliers
- Les grammaires de type 2 génèrent les langages hors-contexte
- Les grammaires de type 1 génèrent les langages contextuels
- Les grammaires de type 0 permettent de générer tout langage "**décidables**"
  $\hookrightarrow$ Langages reconnus en temps fini par une machine
- Les langages qui ne peuvent être générés par une grammaire de type 0 sont dits "**indécidables**"
- L'ensemble des langages générés par des grammaires de type n est strictement inclus dans celui des grammaires de type $n-1$ (avec $n \in \{1,2,3\}$)

Grammaires et reconnaissance par des automates

- Chaque type de grammaire est reconnu par un type spécifique d'automate qui reconnaît les langages générés
  - Langages réguliers : automates finis
  - Langages hors-contexte : automates finis à pile
  - Autres langages : machines de Turing

Grammaires hors-contexte et régulière

Définition : grammaire hors-contexte

Une grammaire est hors-contexte si le membre de gauche de toute règle est un non terminal :

$$A \rightarrow w, \text{ avec } A \in N \text{ et } w \in (N \cup T)^*$$

Définition : grammaire régulière droite

Une grammaire est régulière droite (ou linéaire droite) si toutes ses productions vérifient une des deux formes :

$$A \rightarrow aB \text{ ou } A \rightarrow a \text{ avec } A, B \in N \text{ et } a \in T^*$$

Grammaires linéaires droite et gauche (1/2)

- La majorité des langages de programmation peuvent être décrits par un langage régulier
- Lors de la lecture des symboles d'un mot à analyser de la gauche vers la droite :
  - Grammaire régulière droite : analyse descendante
    $\hookrightarrow$ De l'axiome vers le mot
  - Grammaire régulière gauche : analyse ascendante
    $\hookrightarrow$ Du mot vers l'axiome

Grammaires linéaires droite et gauche (2/2)

Exemple : $G_1 = \{T, N_1, S_1, R_1\}$ et $G_2 = \{T, N_2, S_2, R_2\}$ avec $T = \{a, b\}$ engendrent le même langage $L = \{a^n b^m / n > 0 \wedge m > 0\}$ où :

$$N_1 = \{S_1, U_1\} \qquad N_2 = \{S_2, U_2\}$$

$$R_1 = \left\{ \begin{array}{l} S_1 \rightarrow aS_1 \mid aU_1 \\ U_1 \rightarrow bU_1 \mid b \end{array} \right\} \quad R_2 = \left\{ \begin{array}{l} S_2 \rightarrow S_2 b \mid U_2 b \\ U_2 \rightarrow U_2 a \mid a \end{array} \right\}$$

L'analyse du mot *aaabb* avec G_1 :

$$S_1 \Rightarrow aS_1 \Rightarrow aaS_1 \Rightarrow aaaU_1 \Rightarrow aaabU_1 \Rightarrow aaabb$$

L'analyse du mot *aaabb* avec G_2 :

$$S_2 \Rightarrow S_2 b \Rightarrow U_2 bb \Rightarrow U_2 abb \Rightarrow U_2 aabb \Rightarrow aaabb$$

Langage régulier et automate régulier

Théorème

Tout langage accepté par un automate fini est régulier.

Théorème

Tout langage régulier est accepté par un automate fini.

Remarque

À démontrer en TD !

Grammaire contextuelle

- Certains langages ne sont pas engendrés par une grammaire hors-contexte
- Exemple : $L = \{w \mid w = a^n b^n c^n, n \geq 0\}$
  $\hookrightarrow$ Possible de définir une grammaire sous contexte pour ce langage

Définition : grammaire sous contexte

Une grammaire sous contexte (ou contextuelle) est définie par $G = \{T, N, R, S\}$ où les règles de R sont de la forme suivante :

$$uAv \rightarrow uwv \text{ où } A \in N, u, v \in (T \cup N)^*, w \in (T \cup N)^+$$

Exemple de grammaire sous contexte

Soit la grammaire sous contexte :

$$\begin{array}{lcl} S & \rightarrow & aAb \\ aA & \rightarrow & aaAb \\ A & \rightarrow & \epsilon \end{array}$$

Le langage est $L(G) = \{a^n b^n, n \geq 1\}$.

Exemple de dérivation avec le mot *aaabbb* :

$S \Rightarrow aAb \Rightarrow aaAbb \Rightarrow aaaAbbb \Rightarrow aaabbb$

Arbre d'analyse / syntaxique

Définition : arbre d'analyse

Un arbre d'analyse (ou syntaxique) pour $G = \{T, N, S, R\}$ *est un arbre dont chaque nœud est étiqueté par un élément de* $(T \cup N \cup \epsilon)$ *et qui satisfait les conditions suivantes :*

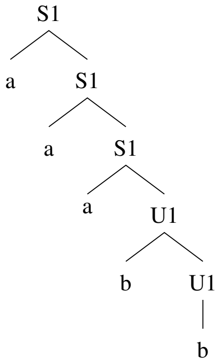
- *La racine est étiquetée par l'axiome* S
- *Chaque nœud inférieur est étiqueté par un non-terminal* $(\in N)$ *et chaque feuille par un symbole terminal* $(\in T)$ *ou par* ϵ
- *Pour tout nœud intérieur, si son étiquette est* $A \in N$ *et si ses fils sont les nœuds ayant pour étiquettes* $X_1, \ldots, X_k$ *alors :* $A \rightarrow X_1 X_2 \ldots X_k \in R$
- *Si un nœud est étiqueté par* ϵ*, il est le seul fils de son père*

Exemple

Pour la dérivation :

$$S_1 \Rightarrow aS_1 \Rightarrow aaS_1 \Rightarrow aaaU_1 \Rightarrow aaabU_1 \Rightarrow aaabb$$

L'arbre syntaxique correspondant est :



Mot généré par un arbre d'analyse

Définition : mot généré par un arbre d'analyse

Le mot généré par un arbre d'analyse est celui obtenu par la concaténation des feuilles de l'arbre de gauche à droite.

Théorème

Étant donnée une grammaire hors-contexte G, un mot w est généré par G ($S \Rightarrow^* w$) si et seulement s'il existe un arbre d'analyse de la grammaire G qui génère w.

Exemple avec les expressions arithmétiques (1/2)

Soit la grammaire $G = \{T, N, R, E\}$ avec R :

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

Première dérivation possible du mot $2 * x + y$:

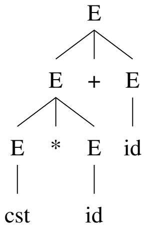
$$E \Rightarrow E + E \Rightarrow E * E + E \Rightarrow E \text{ * id + } E \Rightarrow \text{cst} * \text{id } + E \Rightarrow \text{cst * id + id}$$

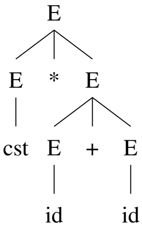
Deuxième dérivation possible :

$$E \Rightarrow E * E \Rightarrow E * E + E \Rightarrow \text{cst * } E + E \Rightarrow \text{cst * id + } E \Rightarrow \text{cst * id + id}$$

Exemple avec les expressions arithmétiques (2/2)

Les deux arbres syntaxiques correspondant aux dérivations :





- Deux arbres syntaxiques pour le même mot : ambiguïté !
  ↪ Mathématiquement, seul celui de gauche est "correct"
- Cela nécessite l'utilisation de parenthèses ou la définition de priorité

Grammaires ambiguës

Définition : grammaire ambiguë

Une grammaire est ambiguë si plus d'un arbre syntaxique est généré pour un même mot.

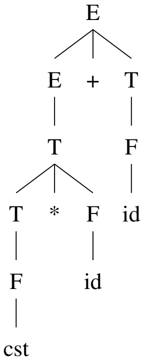
- Il n'existe pas d'algorithme permettant de vérifier si une grammaire est ambiguë
  $\hookrightarrow$ C'est un problème indécidable
- Cependant, il existe des familles de grammaires démontrées non ambiguës

Les expressions sans ambiguïté

Soit la grammaire $G = \{T, N, R, E\}$ avec R :

$$\begin{array}{lcl} E \rightarrow E + E \mid E * E & & E \rightarrow E + T \mid T \\ E \rightarrow (E) & \text{devient} & T \rightarrow T * F \mid F \\ E \rightarrow \text{cst} \mid id & & F \rightarrow \text{cst} \mid \text{id} \mid (E) \end{array}$$

Dérivation du mot $2 * x + y$:



Autre exemple : les conditionnelles (1/3)

Voici la grammaire ambiguë qui reconnaît une conditionnelle :

$$\begin{array}{lcl} \mathit{inst} & \rightarrow & \text{Si } \mathit{expr} \text{ Alors } \mathit{inst} \\ & \rightarrow & \text{Si } \mathit{expr} \text{ Alors } \mathit{inst} \text{ Sinon } \mathit{inst} \\ & \rightarrow & \text{autres instructions} \end{array}$$

Avec $T = \{Si, Alors, Sinon, \ldots\}$ et $N = \{inst, expr, \ldots\}$

Comment reconnaître le mot suivant ?

Si cond1 Alors Si cond2 Alors inst1 Sinon inst2

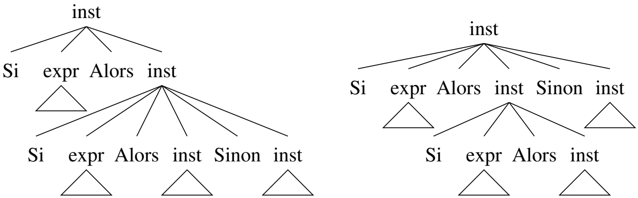
Autre exemple : les conditionnelles (2/3)

Deux dérivations possibles (celle de gauche est préférée habituellement) :

```
Si cond1 Alors
  Si cond2 Alors
    inst1
  Sinon
    inst2
```

```
Si cond1 Alors
  Si cond2 Alors
    inst1
Sinon
  inst2
```

Les arbres syntaxiques correspondants :



Autre exemple : les conditionnelles (3/3)

Voici une grammaire qui lève l'ambiguïté :

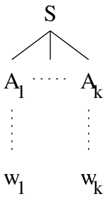
$$
\begin{array}{lcl}
inst & \rightarrow & inst\_close \\
 & | & inst\_non\_close \\
inst\_close & \rightarrow & \text{Si } expr \text{ Alors } inst\_close \text{ Sinon } inst\_close \\
 & | & \text{autre} \\
inst\_non\_close & \rightarrow & \text{Si } expr \text{ Alors } inst \\
 & | & \text{Si } expr \text{ Alors } inst\_close \text{ Sinon } inst\_non\_close
\end{array}
$$

- Permet de faire correspondre le *sinon* avec le *alors* précédent
  ↪ Comme la majorité des langages de programmation actuels
- Instruction close :
  - Instruction si/alors/sinon sans instruction non close
  - Instruction non conditionnelle

Dérivation gauche et droite

- À partir d'une grammaire, il est possible de construire automatiquement un analyseur syntaxique
- Lorsque plusieurs dérivations sont possibles :
  - Analyseur gauche : prend la dérivation la plus à gauche possible
  - Analyseur droit : prend la dérivation la plus à droite possible

Exemple



- Soit la dérivation :
  $S \Rightarrow A_1 \ldots A_k \Rightarrow^* w_1 \ldots w_k$
- À l'étape $A_1 \ldots A_k$:
  $\hookrightarrow$ A_1 : dérivation la plus à gauche
  $\hookrightarrow$ A_k : dérivation la plus à droite

Comparaison entre dérivation gauche et droite

Soit la grammaire des expressions :

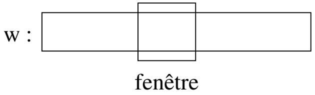
$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to \text{cst} \mid \text{id} \mid (E)$$

Mot à analyser : $id + id * id$

Dérivation gauche	**Dérivation droite**
E	E
E + T	E + T
T + T	E + T * F
F + T	E + T * id
id + T	E + F * id
id + T * F	E + id * id
id + F * F	T + id * id
id + id * F	F + id * id
id + id * id	id + id * id

Analyse syntaxique descendante

- Vue comme une tentative pour déterminer une dérivation gauche d'un mot
- Construction de l'arbre :
  - On part de la racine
  - On construit les nœuds en préordre
- Idée : on avance le plus possible dans le mot



Problème du rebroussement

Soit la grammaire suivante :

$$S \rightarrow cAd$$
$$A \rightarrow ab \mid a$$

- Que faire lorsqu'un a est rencontré ?
- Si la "mauvaise règle" est utilisée, un **rebroussement** est nécessaire
  $\hookrightarrow$ L'analyse descendante récursive
- Généralement, peu fréquents car peu efficaces
  - Possible de l'éviter avec la plupart des langages
  - Réécriture de la grammaire si possible
- Un analyseur sans rebroussement est appelé **analyseur prédictif**

Problème de la récursivité (gauche)

Définition : grammaire récursive gauche

Une grammaire est récursive gauche si elle contient $A \in N$ tel que :

$$A \Rightarrow^{+} A\alpha \text{ avec } \alpha \in (T \cup N)^{*}$$

- La construction d'un analyseur récursif est impossible
  - La procédure de reconnaissance de A s'appelle indéfiniment
  - La fenêtre ne sera jamais modifiée

$\hookrightarrow$ Exécution infinie

$\Rightarrow$ Il faut supprimer la récursivité gauche !

Élimination de la récursivité gauche (immédiate)

- Soit G une grammaire récursive gauche dont les règles sont de la forme :

$$\begin{array}{rcl} A & \rightarrow & A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_n \\ A & \rightarrow & \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m \end{array}$$

- Pour éliminer la récursivité, nous réécrivons les règles comme suit :

$$\begin{array}{rcl} A & \rightarrow & \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_m A' \\ A' & \rightarrow & \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_n A' \mid \epsilon \end{array}$$

Exemple (1/2)

Soit la grammaire des expressions :

$$
\begin{array}{lcl}
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
F & \rightarrow & \text{cst} \mid \text{id} \mid (E)
\end{array}
$$

Nous devons supprimer la récursivité pour E, puis pour T.

- Si $A = E$, $\alpha_1 = +T$, $\beta_1 = T$
  - $A \rightarrow \beta_1 A'$ devient $E \rightarrow TE'$
  - $A' \rightarrow \alpha_1 A' | \epsilon$ devient $E' \rightarrow +TE' | \epsilon$
- Si $A = T$, $\alpha_1 = *F$, $\beta_1 = F$
  - $A \rightarrow \beta_1 A'$ devient $T \rightarrow FT'$
  - $A' \rightarrow \alpha_1 A' | \epsilon$ devient $T' \rightarrow *FT' | \epsilon$

Exemple (2/2)

La grammaire des expressions :

$$
\begin{array}{ll}
E & \rightarrow E + T \mid T \\
T & \rightarrow T * F \mid F \\
F & \rightarrow \text{cst} \mid \text{id} \mid (E)
\end{array}
$$

Devient :

$$
\begin{array}{lll}
E & \rightarrow & TE' \\
E' & \rightarrow & +TE'|\epsilon \\
T & \rightarrow & FT' \\
T' & \rightarrow & *FT'|\epsilon \\
F & \rightarrow & \text{cst} \mid \text{id} \mid (E)
\end{array}
$$

Problème de la récursivité profonde

- L'absence de récursivité immédiate n'est pas suffisante
- Exemple :

$$\begin{aligned} S &\rightarrow Aa \mid b \\ A &\rightarrow Ac \mid Sd \mid \epsilon \end{aligned}$$

- Possible d'avoir les dérivations suivantes : $S \Rightarrow Aa \Rightarrow Sda$
  $\hookrightarrow$ La grammaire est récursive
- Solution : utiliser un algorithme pour l'éliminer

Grammaire sans production vide

- $v \rightarrow \epsilon$ est appelée une **production vide**

Définition : grammaire sans production vide

Une grammaire est sans production vide si :

- *Elle n'a pas de production vide*
- *Elle a une production vide $S \rightarrow \epsilon$ où S est l'axiome et n'apparaît pas dans les parties droites des productions*

Remarque

Il est possible de transformer une grammaire avec productions vides en une grammaire sans production vide.

Algorithme d'élimination de la récursivité gauche

Algorithme pour une grammaire sans cycle et sans production vide

Déterminer un ordre pour les non terminaux notés $A_1 \ldots A_n$
Pour i **allant de** 1 **à** n **Faire**
  Pour j **allant de** 1 **à** $i - 1$ **Faire**
    Remplacer productions de la forme $A_i \rightarrow A_j\alpha$ par
    $A_i \rightarrow \beta_1\alpha| \ldots |\beta_k\alpha$ où $A_j \rightarrow \beta_1 \ldots \beta_k$
  Fin Pour
  Éliminer la récursivité gauche immédiate des A_i productions
Fin Pour
Éliminer les symboles grammaticaux qui ne servent plus

Remarque

Dans certains cas, il est possible d'utiliser cet algorithme pour des grammaires ayant des productions vides.

Exemple d'élimination de la récursivité gauche

Soit :

$$\begin{aligned} S &\rightarrow Aa \mid b \\ A &\rightarrow Ac \mid Sd \mid \epsilon \end{aligned}$$

- Choix d'un ordre ; exemple : S, A donc $A_1 = S$ et $A_2 = A$
- Pour $i = 1$:
  - On ne rentre pas dans la deuxième boucle
  - Pas de récursivité gauche immédiate pour S
- Pour $i = 2$:
  - Remplacement de $A \rightarrow Sd$ par des $A \rightarrow Aad|bd$
  - La grammaire devient :
  $$\begin{aligned} S &\rightarrow Aa \mid b \\ A &\rightarrow Ac \mid Aad \mid bd \mid \epsilon \end{aligned}$$
  - Suppression de la récursivité gauche immédiate de A
  $$\begin{aligned} S &\rightarrow Aa \mid b \\ A &\rightarrow bdA' \mid A' \\ A' &\rightarrow cA' \mid adA' \mid \epsilon \end{aligned}$$

Factorisation gauche : présentation de la problématique

Soit :

$$\begin{array}{rcl} inst & \rightarrow & \text{Si } expr \text{ Alors } inst \\ & \rightarrow & \text{Si } expr \text{ Alors } inst \text{ Sinon } inst \\ & \rightarrow & \text{autres instructions} \end{array}$$

Avec $T = \{Si, Alors, Sinon, \ldots\}$ et $N = \{inst, expr, \ldots\}$

- Quelle règle de production choisir lorsqu'un 'Si' est rencontré ?
- Choix une fois le 'Sinon' rencontré ou pas
- Solution : factoriser la grammaire

Factorisation gauche

Si une grammaire contient des règles de la forme :

$$\begin{aligned} A &\rightarrow \alpha\beta_1 \mid \ldots \mid \alpha\beta_n \\ &\rightarrow \gamma_1 \mid \ldots \mid \gamma_m \end{aligned}$$

Nous pouvons factoriser à gauche comme suit :

$$\begin{aligned} A &\rightarrow \alpha A' \mid \gamma_1 \mid \ldots \mid \gamma_m \\ A' &\rightarrow \beta_1 \mid \ldots \mid \beta_n \end{aligned}$$

Exemple de factorisation à gauche

Soit :

$$\begin{array}{lcl} S & \rightarrow & iEtS \mid iEtSeS \mid a \\ E & \rightarrow & b \end{array}$$

- Correspond à la grammaire des conditionnelles :
  - i correspond à 'Si', t à 'Alors' et e à 'Sinon'
  - E est une expression, S une instruction
- Après la factorisation :

$$\begin{array}{lcl} S & \rightarrow & iEtSS' \mid a \\ S' & \rightarrow & eS \mid \epsilon \\ E & \rightarrow & b \end{array}$$

- Sur un 'Si', on peut maintenant développer $iEtSS'$
- Une fois $iEtSS'$ reconnu : on développe eS ou ϵ

Analyseur prédictif

- Soit une grammaire :
  - Élimination des récursivités à gauche
  - Factorisation à gauche

$\Rightarrow$ Grammaire analysée par descente récursive sans rebroussement

$\hookrightarrow$ Construction d'un analyseur prédictif

- Problématique :
  - Soit un symbole terminal a en entrée
  - Soit le symbole A à développer

$\hookrightarrow$ Quelle production parmi $A \rightarrow \alpha_1 \mid \ldots \mid \alpha_n$ doit-on développer ?

Fonctions *Premier* et *Suivant*

Définition : fonction *Premier*

Soit $G = (T, N, R, S)$. *Si* $\alpha \in (T \cup N)^*$, *Premier(*α*) est l'ensemble des symboles terminaux qui commencent les chaînes qui se dérivent de* α *:*

$$Premier(\alpha) = \{x \in T \mid \exists \beta \in (T \cup N)^*, \alpha \Rightarrow^* x\beta\} \cup \{\epsilon \text{ si } \alpha \Rightarrow^* \epsilon\}$$

Définition : fonction *Suivant*

Soit $G = (T, N, R, S)$. *Si* $A \in N$, *Suivant(*A*) est l'ensemble des symboles terminaux qui peuvent apparaître directement à droite de* A *dans une protophrase :*

$$Suivant(A) = \{x \in T \mid \exists \alpha, \beta \in (T \cup N)^*, S \Rightarrow^* \alpha A x \beta\} \cup \{\epsilon \text{ si } S \Rightarrow^* \alpha A\}$$

Algorithme de calcul pour *Premier*

Initialisation :

- $\forall X \in T$, $Premier(X) = \{X\}$
- $\forall X \in N$, $Premier(X) = \emptyset$

Pour tout $X \in N$ et $X \rightarrow Y_1 Y_2 \ldots Y_n$, appliquer ces règles jusqu'à ce qu'aucun terminal, ni ϵ ne puisse être ajouté aux ensembles *Premier* :

- Si $\exists i \in [1, n]$ / $a \neq \epsilon \in Premier(Y_i)$ et $\forall j \in [1, i-1]$ $\epsilon \in Premier(Y_j)$ alors $a \in Premier(X)$;
- Si $\forall i \in [1, n]$, $\epsilon \in Premier(Y_i)$ alors $\epsilon \in Premier(X)$

Remarques

- Si $X \rightarrow \epsilon$, alors $\epsilon \in Premier(X)$
- Si $X \rightarrow aY$ avec $a \in T$, alors $a \in Premier(X)$

Algorithme de calcul pour *Suivant*

Initialisation :

- $\forall A \in N$, $Suivant(A) = \emptyset$

On applique les règles suivantes :

- $\epsilon \in Suivant(S)$ si S est l'axiome
- Si $A \rightarrow \alpha B \beta$ où $A, B \in N$ et $\alpha, \beta \in (T \cup N)^*$:
  $Suivant(B) = Suivant(B) \cup Premier(\beta) \setminus \epsilon$
- Si $A \rightarrow \alpha B$ ou $A \rightarrow \alpha B \beta$ avec $\epsilon \in Premier(\beta)$:
  $Suivant(B) = Suivant(B) \cup Suivant(A)$

Exemples de calcul de *Premier* et de *Suivant*

Soit :

$$\begin{array}{lcl} S & \rightarrow & X\ Y \mid d \\ Y & \rightarrow & c \mid \epsilon \\ X & \rightarrow & Y \mid a \end{array}$$

- $Premier(S) = \{a, c, d, \epsilon\}$
- $Premier(X) = \{a, c, \epsilon\}$
- $Premier(Y) = \{c, \epsilon\}$
- $Suivant(S) = \{\epsilon\}$
- $Suivant(X) = \{c, \epsilon\}$
- $Suivant(Y) = \{c, \epsilon\}$

Grammaires LL(1) : caractérisation

- Soit une grammaire $G = (T, N, R, S)$ hors-contexte, possédant des règles de la forme $A \rightarrow \alpha_1 \mid \ldots \mid \alpha_n$
- Pour construire un analyseur prédictif descendant sans rebroussement, il est nécessaire de pouvoir choisir la règle de production ($A \rightarrow \alpha_i$, $1 \leq i \leq n$) en fonction de la fenêtre (le prochain lexème à analyser)
  $\hookrightarrow$ Une seule règle possible
- Plusieurs conditions :
  - Ensembles *Premier*(α_i) disjoints 2 à 2
    $\hookrightarrow$ Le symbole actuel nous permet de choisir une seule règle
  - Si $\epsilon \in$ *Premier(A)*, la fenêtre est élément de *Suivant(A)* ; chaque *Premier*(α_i) contenant ϵ est disjoint de *Suivant(A)*
    $\hookrightarrow$ Pas de confusion entre la réduction de A et de α_i

Grammaires LL(1)

Définition : grammaire LL(1)

$G = (T, N, R, S)$ *est une grammaire LL(1) si et seulement si* $\forall A \in N / A \rightarrow \alpha \in R \wedge A \rightarrow \beta \in R$ *:*

$$(A \rightarrow \alpha \neq A \rightarrow \beta) \Rightarrow$$
$$(Premier(\alpha) \cap Premier(\beta) = \emptyset$$
$$et$$
$$\epsilon \in Suivant(\beta) \Rightarrow Premier(\alpha) \cap Suivant(A) = \emptyset)$$

Remarques

- Il y a au plus une seule règle de A qui dérive sur ϵ
- Si $\beta \Rightarrow^* \epsilon$, α ne dérive pas sur une chaîne commençant par *Suivant(A)*

Analyseur syntaxique récursif descendant (1/3)

Soit $G = (T, N, R, S)$, une grammaire LL(1)

- Augmentation de la grammaire par un $S' \in N$ qui est le nouvel axiome :
$$G' = (T, N \cup \{S'\}, R \cup \{S' \rightarrow S\}, S')$$
- La procédure pour S' est la suivante :

Procédure S'()

S()
Si symbole $\neq \dashv$ **Alors**
Erreur

Analyseur syntaxique récursif descendant (2/3)

- Pour chaque règle $A \rightarrow \alpha_1 \mid \ldots \mid \alpha_n$ avec $\epsilon \notin$ *Premier(A)*, on construit une procédure
- Si $\epsilon \in$ *Premier(A)*, il faut réduire A à ϵ pour tout $a \in T$ tel que $\forall 1 \leq i \leq n, a \notin$ *Premier*(α_i)

Procédure A()

  Si symbole $\in$ *Premier*(α_1) **Alors**
    $\alpha_1()$
  Sinon Si symbole $\in$ *Premier*(α_2) **Alors**
    $\alpha_2()$
  Sinon
    Si $\epsilon \notin$ *Premier(A)* **Alors**
      Erreur
    Sinon
      / A est réduit à* ϵ **/*

Analyseur syntaxique récursif descendant (3/3)

- Pour chaque α_i, on construit une procédure
- Si $\alpha_i \rightarrow a\beta$ et $a \in T$, déplacement de la fenêtre avant de reconnaître β

Procédure α_i**()**

/ symbole = a, avec a* $\in$ *Premier(A) */*
avance
β()

- Si $\alpha_i \rightarrow AB$ avec $A \in N$, la fenêtre n'est pas modifiée

Procédure α_i**()**

A()
B()

- Si $\alpha_i \rightarrow \epsilon$ alors la procédure est vide

Exemple (1/3)

Soit la grammaire :

$$
\begin{array}{lcl}
S & \rightarrow & (\ \mathrm{A} \\
A & \rightarrow & \mathrm{B}\ ) \\
 & \rightarrow &) \\
B & \rightarrow & \mathrm{entier}\ \mathrm{C} \\
 & \rightarrow & \mathrm{S}\ \mathrm{C} \\
C & \rightarrow & ,\ \mathrm{B} \\
 & \rightarrow & \epsilon
\end{array}
$$

$S' \rightarrow S$

Procédure S'()

S()

Si symbole $\neq \dashv$ **Alors**

Erreur

$S \rightarrow (A$

Procédure S()

Si symbole = '(' **Alors**

avance ; A()

Sinon

erreur

Exemple (2/3)

$A \rightarrow B) \mid)$

Procédure A()

  Si symbole $\in$ *Premier(B)* $= \{$ *entier*, $'('\}$ **Alors**

    B()

    Si symbole = ')' **Alors**

      avance

    Sinon

      erreur

  Sinon

    Si symbole = ')' **Alors**

      avance

    Sinon

      erreur

Exemple (3/3)

$B \rightarrow entier\ \ C \mid SC$

Procédure B()

Si symbole = entier **Alors**
avance
C()
Sinon Si symbole = '(' **Alors**
S()
C()
Sinon
erreur

Avec $Premier(S) = \{\ '('\ \}$

$C \rightarrow\ ,\ B \mid \epsilon$

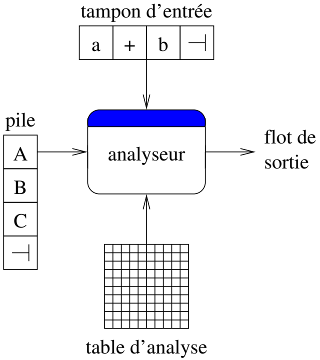
Procédure C()

Si symbole = ',' **Alors**
avance
B()
Sinon
/ Rien */*

Optimisations

- Remplacer les procédures appelées une fois par leur corps
- Éliminer des textes de programme communs
- Éliminer les appels récursifs terminaux à l'aide de boucles TantQue

Analyseur LL



- Tampon d'entrée : chaîne à analyser terminée par ⊣
- Pile : contient des symboles grammaticaux avec ⊣ au fond
  ↪ Initialement : axiome et ⊣
- Table d'analyse : tableau à deux dimensions $M[A, a]$ avec $A \in N$, $a \in T \cup \{\dashv\}$

↪ En fonction du sommet de la pile et du prochain lexème, action déterminée par la table

Exécution d'un analyseur LL

En fonction du symbole X de la pile et de a le symbole d'entrée :

- $X = a = \dashv$: analyse réussie
  $\hookrightarrow$ Action **ACC** (ACCepter)
- $X = a \neq \dashv$: X dépilé, pointeur d'entrée avancé
  $\hookrightarrow$ Action **AVCTS** (AVancer dans la Cible et dans le Texte Source)
- Si $X \in N$, consultation de la table $M[X, a]$, deux possibilités :
  - Erreur
  - Production $X \rightarrow UVW$, X remplacé par W, V et U (U au sommet de la pile)
    $\hookrightarrow$ Si production vide : action **AVC** (AVancer dans la Cible)

La construction de la table a besoin de *Premier* et *Suivant*

Construction de la table d'analyse

Nous distinguons les 4 règles suivantes :

1. $M[\dashv, \dashv] = \text{ACC}$
   $\hookrightarrow$ Le mot a été reconnu
2. $\forall a \in T$, $M[a, a] = \text{AVCTS}$
   $\hookrightarrow$ Une unité lexicale a été reconnue, il passe à la suite
3. $\forall A \in N$, s'il existe une règle $A \rightarrow \alpha$ (avec $\alpha \neq \epsilon$), $M[A, a] = A \rightarrow \alpha$, $\forall a \in Premier(\alpha) \cup Suivant(A)$ si $\alpha \Rightarrow^* \epsilon$
   $\hookrightarrow$ Une production est développable
4. $\forall A \in N$, s'il existe une règle $A \rightarrow \epsilon$, $M[A, a] = AVC$, $\forall a \in Suivant(A)$
   $\hookrightarrow$ A doit être réduit par ϵ si on trouve un suivant de A

Construction de la table d'analyse : exemple

$$
\begin{array}{lll}
S & \rightarrow & \text{(A} \\
A & \rightarrow & \text{B)} \\
 & \rightarrow & \text{)} \\
B & \rightarrow & \text{entier C} \\
 & \rightarrow & \text{S C} \\
C & \rightarrow & \text{, B} \\
 & \rightarrow & \epsilon
\end{array}
$$

- *Premier* :
  - *Premier(S)* = { '(' }
  - *Premier(A)* = { *entier*, '(', ')' }
  - *Premier(B)* = { *entier*, '(' }
  - *Premier(C)* = { ',', ϵ }
- *Suivant* :
  - *Suivant(S)* = { ',', ')', ϵ }
  - *Suivant(A)* = { ',', ')', ϵ }
  - *Suivant(B)* = { ')' }
  - *Suivant(C)* = { ')' }

Construction de la table

	'('	')'	*entier*	','	$\dashv$
S	$S \rightarrow (A$			AVC	
A	$A \rightarrow B)$	$A \rightarrow)$	$A \rightarrow L)$		
B	$B \rightarrow SC$		$B \rightarrow aC$		
C		$C \rightarrow \epsilon$		$C \rightarrow , B$	
'('	AVCTS				
')'		AVCTS			
a			AVCTS		
','				AVCTS	
$\dashv$					ACC

Exemple d'exécution

- Analyse de : $(1, 2)$

Cible	Texte source	Action
$S \dashv$	$(1, 2) \dashv$	$S \rightarrow (A$
$(A \dashv$	$(1, 2) \dashv$	AVCTS
$A \dashv$	$1, 2) \dashv$	$A \rightarrow B)$
$B) \dashv$	$1, 2) \dashv$	$B \rightarrow entier\ C$
$entier\ C) \dashv$	$1, 2) \dashv$	AVCTS
$C) \dashv$	$, 2) \dashv$	$C \rightarrow , B$
$, B) \dashv$	$, 2) \dashv$	AVCTS
$B) \dashv$	$2) \dashv$	$B \rightarrow entier\ C$
$entier\ C) \dashv$	$2) \dashv$	AVCTS
$C) \dashv$	$) \dashv$	AVC
$) \dashv$	$) \dashv$	AVCTS
$\dashv$	$\dashv$	ACC