Retour aux grammaires. . .

INFO010 – Théorie des langages – Partie pratique

S. COLLETTE et G. GEERAERTS

Les grammaires

Une grammaire est un quadruplet $G = \langle V, T, P, S \rangle$ où

- V est l'ensemble des variables ;
- T est l'ensemble des terminaux ;
- P est l'ensemble des règles de production.
  $P \subseteq (V \cup T)^* V (V \cup T)^* \times (V \cup T)^*$;
- $S \in V$ est le symbole de départ.

Suppresion des symboles *inutiles*

- Un symbole $X \in V \cup T$ est inutile s'il n'existe pas de dérivation de la forme

$$\underbrace{S \overset{*}{\Rightarrow}}_{\text{accessible}} wXy \underbrace{\overset{*}{\Rightarrow} wxy}_{\text{productif}}$$

- Pour supprimer les symboles inutiles :
  - On supprime les symboles qui ne produisent rien.
  - On supprime les symboles inaccessibles.
- L'ordre est important ! !

Symboles inutiles – Exemples

- Grammaire dont le langage est vide

$$\begin{aligned} S &\rightarrow aAb \\ A &\rightarrow aA \\ &\quad\ bC \\ C &\rightarrow Ab \end{aligned}$$

- Grammaire avec une variable inaccessible (C)

$$\begin{aligned} S &\rightarrow aAb \\ A &\rightarrow aA \\ &\quad\ b \\ C &\rightarrow bA \end{aligned}$$

Symboles inutiles – Algorithmes – 1

Grammaire `SuppNonProd`(**Grammaire** $G = \langle V, T, P, S \rangle$)

begin

$V_0 \leftarrow \emptyset$; $i \leftarrow 0$;

répéter

$i \leftarrow i + 1$;

$V_i \leftarrow \{A | A \rightarrow \alpha \in P \wedge \alpha \in (V_{i-1} \cup T)^*\} \cup V_{i-1}$;

jusqu'à $V_i = V_{i-1}$;

$V' \leftarrow V_i$;

$P' \leftarrow$ ensemble des règles de P qui ne contiennent pas de variables dans $V \setminus V'$;

`retourner`$(G' = \langle V', T, P', S \rangle)$;

end

Symboles inutiles – Algorithmes – 2

Grammaire `SuppInacc`(**Grammaire** $G = \langle V, T, P, S \rangle$)

begin

$V_0 \leftarrow \{S\}$; $i \leftarrow 0$;

répéter

$i \leftarrow i + 1$;

$V_i \leftarrow \{X | \exists A \rightarrow \alpha X \beta \text{ dans } P \wedge A \in V_{i-1}\} \cup V_{i-1}$;

jusqu'à $V_i = V_{i-1}$;

$V' \leftarrow V_i \cap V$; $T' \leftarrow V_i \cap T$;

$P' \leftarrow$ les productions de P qui contiennent uniquement des symboles de V_i ;

`retourner`($G' = \langle V', T', P', S \rangle$) ;

end

Symboles inutiles – Algorithmes – 3

- Pour supprimer les symboles inutiles :

Grammaire `SuppInut(`**Grammaire** $G = \langle V, T, P, S \rangle$`)`

begin

  Grammaire $G_1 \leftarrow$ `SuppNonProd(`G`)` ;

  Grammaire $G_2 \leftarrow$ `SuppInacc(`G_1`)` ;

  `retourner(`G_2`)` ;

end

Exercice 1

Supprimez les symboles inutiles dans les grammaires suivantes :

$$
\begin{aligned}
S &\rightarrow a|A \\
A &\rightarrow AB \\
B &\rightarrow b
\end{aligned}
$$

$$
\begin{aligned}
S &\rightarrow A \\
&\quad\ B \\
A &\rightarrow aB \\
&\quad\ bS \\
&\quad\ b \\
B &\rightarrow AB \\
&\quad\ Ba \\
C &\rightarrow AS \\
&\quad\ b
\end{aligned}
$$

Exercice 1 – solution

- Suppression des symboles non-productifs. À la stabilisation, on trouve $V_i = \{S, B\}$ et donc $G_1 = \langle \{S, B\}, \{a, b\}, \{S \to a, B \to b\}, S \rangle$.
- Mais B n'est pas accessible à partir de S, on a donc : $G' =< \{S\}, \{a\}, \{S \to a\}, S >$.

Exercice 1 – solution – suite

- Étapes du calcul de V_i :

i	V_i
0	$\emptyset$
1	$\{C, A\}$
2	$\{C, A, S\}$
3	$\{C, A, S\}$

Exercice 1 – solution – suite

- On a donc pour P' :

$$
\begin{array}{lcl}
S & \rightarrow & A \\
A & \rightarrow & bS \\
 & & b \\
C & \rightarrow & AS \\
 & & b
\end{array}
$$

Exercice 1 – solution – suite

- Supprimons maintenant les symboles inaccessibles :

i	V_i
0	$\{S\}$
1	$\{S, A\}$
2	$\{S, A\}$

- On a donc G' avec : $V' = \{S, A\}$, $P' = \{S \to A, A \to bS|b\}$, $T' = \{b\}$ et $S' = S$.

Transformations des CFG

Lever les ambiguïtés Un grammaire G est dite ambiguë s'il existe un mot $w \in G$ tel qu'il existe au moins deux arbres de dérivation différents pour w. Par exemple (nb désigne un nombre entier) :

$$\begin{aligned} E &\rightarrow E + E \\ &\quad\; E * E \\ &\quad\; (E) \\ &\quad\; nb \end{aligned}$$

est ambiguë. Considérez par exemple le mot 5+3+2.

Transformations des CFG

- Associativité des opérateurs On peut vouloir imposer une certaine associativité aux opérateurs. Dans l'exemple précédent, l'associativité n'est pas fixée (en raison de l'ambiguïté).
- Priorité des opérateurs De façon similaire, on peut constater que le * n'a pas priorité sur le +.

Lever les ambiguïtés

On résout les deux premiers problèmes en transformant G en G' (ce qui fixe l'associativité à gauche) :

$$
\begin{array}{rcl}
E & \rightarrow & E + T \\
 & & E * T \\
 & & T \\
T & \rightarrow & (E) \\
 & & nb
\end{array}
$$

Lever les ambiguïtés – suite

On impose ensuite la priorité de * par rapport à + en transformant G' en G'' :

$$
\begin{array}{rcl}
E & \rightarrow & E + T \\
 & & T \\
T & \rightarrow & T * F \\
 & & F \\
F & \rightarrow & (E) \\
 & & nb
\end{array}
$$

Exercice 2

Soit la grammaire $\mathcal{G}$:

$$
\begin{array}{rcl}
E & \rightarrow & E\ op\ E \\
 & & I\,D[E] \\
 & & I\,D \\
op & \rightarrow & * \\
 & & / \\
 & & + \\
 & & - \\
 & & ->
\end{array}
$$

- Montrez que la grammaire est ambiguë.
- L'ordre de priorité des opérateurs est le suivant : $\{[], ->\} > \{*, /\} > \{+, -\}$. Modifiez $\mathcal{G}$ de manière à prendre en compte la priorité des opérateurs, ainsi que l'associativité à gauche.

Correction de l'exercice 2

$$
\begin{array}{lcl}
E & \rightarrow & E+T \\
 & & E-T \\
 & & T \\
T & \rightarrow & T*F \\
 & & T/F \\
 & & F \\
F & \rightarrow & F->G \\
 & & I\,D[E] \\
 & & G \\
G & \rightarrow & I\,D
\end{array}
$$

Factorisation

- La factorisation a pour but de supprimer les règles ayant un préfixe commun (qui posent des problèmes de prédiction aux analyseurs).
- Le cas échéant, la factorisation peut permettre de transformer en grammaire LL(1) une grammaire qui ne l'était pas !
- Exemple : $S \rightarrow ab|aa$ n'est pas LL(1)
- Après factorisation on a la grammaire LL(1) :

$$S \rightarrow aN$$

$$N \rightarrow a|b$$

Factorisation – algorithme

`Factorise(`**Grammaire** $G = \langle V, T, P, S \rangle$`)` **begin**

tant que G *possède au moins deux règles avec le même membre gauche et un préfixe commun* **faire**

Soit $E = \{A \to \alpha\beta, \ldots, A \to \alpha\zeta\}$ un tel ensemble de règles ;

Soit $\mathcal{V}$ une nouvelle variable;

$V \leftarrow V \cup \{\mathcal{V}\}$;

$P \leftarrow P \setminus E$;

$P \leftarrow P \cup \{A \to \alpha\mathcal{V}, \mathcal{V} \to \beta, \ldots, \mathcal{V} \to \zeta\}$;

end

Exercice 3

Appliquez ce principe à la grammaire suivante :

<stmt> → **if** <expr> **then** <stmt list> **end if**

<stmt> → **if** <expr> **then** <stmt list> **else** <stmt list> **end if**

Exercice 3 – solution

<stmt> → **if** <expr> **then** <stmt list> <iftail>

<iftail> → **end if**

<iftail> → **else** <stmt list> **end if**

Dérécursification

- Pour les mêmes raisons, on aime souvent supprimer la récursivité à gauche (ou à droite) dans une grammaire.
- Par exemple :
  - La règle suivante est récursive à gauche : $S \rightarrow S\alpha|\beta$

$$S \rightarrow \mathcal{V}\mathcal{T}$$

  - On peut la transformer en : $\mathcal{V} \rightarrow \beta$

$$\mathcal{T} \rightarrow \alpha\mathcal{T}|\varepsilon$$

Dérécursification – Algorithme

`Derecursifie(`**Grammaire** $G = \langle V, T, P, S \rangle$`)` **begin**

tant que G *contient une variable* A *récursive à gauche* **faire**

Soit $E = \{A \to A\alpha, A \to \beta, \ldots, A \to \zeta\}$ l'ensemble des productions ayant A pour membre de gauche ;

Soient $\mathcal{U}$ et $\mathcal{V}$ deux nouvelles variables ;

$V \leftarrow V \cup \{\mathcal{U}, \mathcal{V}\}$;

$P \leftarrow P \setminus E$;

$P \leftarrow P \cup \{A \to \mathcal{U}\mathcal{V}, \mathcal{U} \to \beta, \ldots, \mathcal{U} \to \zeta,$
$\mathcal{V} \to \alpha\mathcal{V}, \mathcal{V} \to \varepsilon\}$;

end

Exercice 4

Appliquez l'algorithme de dérécursification à la grammaire suivante :

$$
\begin{aligned}
E &\rightarrow E + T \\
&\quad\ T \\
T &\rightarrow T * P \\
&\quad\ P \\
P &\rightarrow I\,D
\end{aligned}
$$

Exercice 4 – solution

$$
\begin{aligned}
E &\rightarrow AB \\
A &\rightarrow T \\
B &\rightarrow +TB \\
&\quad\ \varepsilon \\
T &\rightarrow CD \\
C &\rightarrow P \\
D &\rightarrow *PD \\
&\quad\ \varepsilon \\
P &\rightarrow ID
\end{aligned}
$$

Exercice 5 – Question d'examen

Transformez la grammaire suivante pour la rendre LL(1) :

$$
\begin{aligned}
S &\rightarrow aE|bF \\
E &\rightarrow bE|\varepsilon \\
F &\rightarrow aF|aG|aHD \\
G &\rightarrow Gc|d \\
H &\rightarrow Ca \\
C &\rightarrow Hb \\
D &\rightarrow ab
\end{aligned}
$$

Exercice 5 – Correction

- Suppression des symboles non-productifs : H et C ne produisent rien car ils sont mutuellement récursifs et que rien ne permet d'arrêter cette récursivité. On enlève donc les règles $H \rightarrow Ca$, $C \rightarrow Hb$ et $F \rightarrow aHD$.
- Suppression des symboles inaccessibles : de par la suppression de $F \rightarrow aHD$, D devient inaccessible. On supprime donc $D \rightarrow ab$.

Exercice 5 – Correction – 2

On a maintenant :

$$\begin{aligned} S &\rightarrow aE|bF \\ E &\rightarrow bE|\varepsilon \\ F &\rightarrow aF|aG \\ G &\rightarrow Gc|d \end{aligned}$$

- Suppression de la récursivité : $G \rightarrow Gc$ est récursive. On remplace $G \rightarrow Gc|d$ par $G \rightarrow dG'$ et $G' \rightarrow cG'|\varepsilon$.
- Factorisation : On remplace $F \rightarrow aF|aG$ par $F \rightarrow aF'$ et $F' \rightarrow F|G$.

Exercice 5 – Correction – 3

Au final, on a bien une grammaire LL(1) :

$$
\begin{array}{rrcl}
(0) & S' & \rightarrow & S\$ \\
(1,2) & S & \rightarrow & aE|bF \\
(3,4) & E & \rightarrow & bE|\varepsilon \\
(5) & F & \rightarrow & aF' \\
(6,7) & F' & \rightarrow & F|G \\
(8) & G & \rightarrow & dG' \\
(9,10) & G' & \rightarrow & cG'|\varepsilon
\end{array}
$$

	a	b	c	d	$
S'	P0	P0	×	×	×
S	P1	P2	×	×	×
E	×	P3	×	×	P4
F	P5	×	×	×	×
F'	P6	×	×	P7	×
G	×	×	×	P8	×
G'	×	×	P9	×	P10