Construction d'analyseurs syntaxiques
TL2 Ensimag 1A 2022-2023

Chapitre 5
Construction de BNF LL(1)

Xavier.Nicollin@grenoble-inp.fr
thanks Sylvain Boulmé *et* Lionel Rieg

Problématique

Transformer la « spécification » d'un analyseur
via BNF attribuée + priorités
en « implémentation » via BNF LL(1) attribuée équivalente

équivalente = même syntaxe (langage reconnu)
et même sémantique associée (priorités + attributs)

Difficultés

- Il n'existe pas forcément de grammaire LL(1) équivalente (ex : palindromes)
- Le problème est indécidable : on applique des patrons à partir d'exemples types $\rightsquigarrow$ heuristiques

Chapitre 5 Construction de BNF LL(1)

Priorités dans les langages d'expressions

Transformation en grammaire LL(1)

Langages réguliers et EBNF

Exemple de mini-expressions arithmétiques $\mathrm{exp}\uparrow\mathbb{Z}$

$$
\begin{array}{rcll}
\mathrm{exp}\uparrow r & ::= & \mathtt{NAT}\uparrow r & \\
 & | & \mathtt{-}\ \mathrm{exp}\uparrow r_1 & r := -r_1 \\
 & | & \mathrm{exp}\uparrow r_1\ \mathtt{-}\ \mathrm{exp}\uparrow r_2 & r := r_1 - r_2 \\
 & | & (\mathrm{exp}\uparrow r) &
\end{array}
$$

Priorité 1 : moins binaire (associatif à gauche)
Priorité 0 : moins unaire

Plan

1. Désambiguïsation avec encodage des priorités dans la grammaire
2. Mise en forme LL(1) par transformations de grammaires

Encodage des priorités d'une grammaire d'expressions

Introduire un non-terminal $\mathsf{E_n}$ **par** niveau de priorité n via

$\mathcal{L}(\mathsf{E_n}) \stackrel{def}{=}$

ensemble des expr tq tout opérateur de priorité $> \mathsf{n}$
apparaît uniquement dans une sous-expr de forme « $\mathtt{(}e\mathtt{)}$ »

Pour n maximal, $\mathcal{L}(\mathsf{E_n}) =$ ensemble des expressions

Construction des règles

- pour tout $n > 0$, on a une règle $\mathsf{E_n} \to \mathsf{E_{n-1}}$
- pour n maximal, on a une règle $\mathsf{E_0} \to \mathtt{(}\ \mathsf{E_n}\ \mathtt{)}$
- Tout op binaire ♠ de niveau n induit une des 3 règles
  si n associatif à gauche, $\mathsf{E_n} \to \mathsf{E_n} \spadesuit \mathsf{E_{n-1}}$
  si n associatif à droite, $\mathsf{E_n} \to \mathsf{E_{n-1}} \spadesuit \mathsf{E_n}$
  si n non-associatif, $\mathsf{E_n} \to \mathsf{E_{n-1}} \spadesuit \mathsf{E_{n-1}}$

NB Associativité fixée par niveau de priorité !

Application sur l'exemple précédent

$$
\begin{array}{rcll}
\text{exp}\uparrow r & ::= & \texttt{NAT}\uparrow r & \\
& | & \texttt{-}\ \text{exp}\uparrow r_1 & r := -r_1 \\
& | & \text{exp}\uparrow r_1\ \texttt{-}\ \text{exp}\uparrow r_2 & r := r_1 - r_2 \\
& | & (\text{exp}\uparrow r) &
\end{array}
$$

Priorité 1 : moins binaire (associatif à gauche)
Priorité 0 : moins unaire

...

Exo 1 Arbres d'analyses de « $-1 - 1 - 1$ » et de « $-(1 - (1 - 1))$ » ?
(où « 1 » représente le terminal « $\texttt{NAT}\uparrow 1$ »)

Exo 2 La grammaire obtenue est-elle LL(1) ?

Chapitre 5 Construction de BNF LL(1)

Élimination des règles immédiatement récursives à gauche

Soit l'équation X ::= X α | β avec $\alpha \in \mathcal{V}^*$ et $\beta \in \{\varepsilon\} \cup (\mathcal{V} \backslash X).\mathcal{V}^*$
Exo 3 Montrer qu'une BNF LL(1) n'a pas une telle équation

...

Langage reconnu de forme « $\beta.(\alpha^*)$ »

Donc remplacement de l'équation précédente par

X ::= β X′ X′ ::= α X′ | ε

La nouvelle BNF peut être LL(1) si $\mathrm{Prem}(\alpha) \cap \mathrm{Suiv}(X) = \emptyset$

Et pour le calcul d'attributs ?

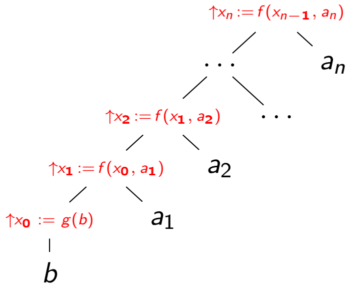
Élimination de la récursion gauche avec la sémantique

Soit BNF (1)
$$\begin{array}{lcll} X\uparrow p & ::= & X\uparrow n\ \alpha\uparrow a & p := f(n,a) \\ & | & \beta\uparrow b & p := g(b) \end{array}$$

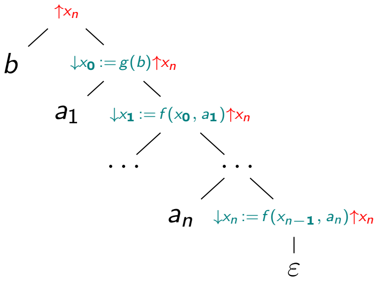
$\leadsto$ BNF (2) équivalente sans récursion gauche

$$\begin{array}{rcll} X\uparrow p & ::= & \beta\uparrow b\ X'\downarrow g(b)\uparrow p & \\ X'\downarrow n\uparrow p & ::= & \alpha\uparrow a\ X'\downarrow f(n,a)\uparrow p & \\ & | & \varepsilon & p := n \end{array}$$

Arbres d'analyses BNF (1)



Arbres d'analyses BNF (2)



Application pour l'associativé à gauche

Exo 4 Traiter l'exemple en indiquant la condition pour être LL(1)

$$\begin{array}{lcll} \mathrm{exp}_1{\uparrow}r & ::= & \mathrm{exp}_1{\uparrow}r_1 \text{ - } \mathrm{exp}_0{\uparrow}r_2 & r := r_1 - r_2 \\ & \mid & \mathrm{exp}_0{\uparrow}r & \end{array}$$

...

Le cas de l'associativité à droite

Exo 5 Traiter l'exemple en indiquant la condition pour être LL(1)

$$
\begin{array}{lcll}
\mathrm{exp}_1{\uparrow}r & ::= & \mathrm{exp}_0{\uparrow}r_1 \;{**}\; \mathrm{exp}_1{\uparrow}r_2 & r := r_1^{r_2} \\
& \mid & \mathrm{exp}_0{\uparrow}r &
\end{array}
$$

Solution = factorisation à gauche

...

Chapitre 5 Construction de BNF LL(1)

Tout langage régulier L est LL(1)

BNF LL(1) de $L =$ système d'équations (linéaires à droite) de l'automate déterministe minimal de L moins l'éventuel état puits

Exo 6 Faire la preuve que la BNF est bien LL(1)

...

Les calculs de directeurs LL(1) s'étendent aux EBNF

Définition EBNF = BNF avec des expressions régulières
en membre droit d'équations
$\Rightarrow$ expressif & efficace pour engendrer des analyseurs LL
(cf. méta-interpréteur ANTLR)

Exemple 1 $\mathsf{X} ::= \alpha_1.(\alpha_2)^*.\alpha_3$
se réécrit (pour le calcul de directeur) en

$$\mathsf{X} ::= \alpha_1.\mathsf{X}'.\alpha_3 \qquad \mathsf{X}' ::= \alpha_2.\mathsf{X}' \mid \varepsilon$$

Exemple 2 $\mathsf{X} ::= \alpha_1.(\alpha_2 \mid \alpha_3).\alpha_4$
se réécrit (pour le calcul de directeur) en

$$\mathsf{X} ::= \alpha_1.\mathsf{X}'.\alpha_4 \qquad \mathsf{X}' ::= \alpha_2 \mid \alpha_3$$

Exemple de EBNF L-attribuée

Pour associativité à gauche du '-' binaire (cf. exo 4 précédent)

$$\mathrm{exp}_1{\uparrow}r ::= \mathrm{exp}_0{\uparrow}r_1\ \{r := r_1\}\ (\text{'-'}\ \mathrm{exp}_0{\uparrow}r_2\ \{r := r - r_2\})^*$$

Si LL(1), s'implémente très efficacement en :

```
def parse_exp1():
   r1 = parse_exp0()
   r = r1
   while current == '-':
     parse_token('-')
     r2 = parse_exp0()
     r = r - r2
   return r
```

Autre exemple de EBNF L-attribuée

Pour associativité à droite du '**' binaire (cf. exo 5 précédent)

$$\mathsf{exp}_1\uparrow r ::= \mathsf{exp}_0\uparrow r_1\ (\varepsilon\ \{r := r_1\} \mid \text{'**'}\ \mathsf{exp}_1\uparrow r_2\ \{r := r_1^{r_2}\})$$

Si LL(1), s'implémente en :

```
def parse_exp1():
  r1 = parse_exp0()
  if current == '**':
    parse_token('**')
    r2 = parse_exp1()
    r = r1 ** r2
  else:
    r = r1
  return r
```

Associativité à droite nécessite récursivité (ou son codage via pile) !
Dû à lecture gauche/droite
Vrai même pour autres analyses que LL

Conclusion du cours

- Analyse LL(1) généralisable pour augmenter l'expressivité
  cf. analyse LL($\star$) du méta-interpréteur ANTLR

- Existence d'autres types d'analyse comme LALR
  (cf. méta-interpréteur Bison-Yacc)
  Analyse ascendante : l'arbre d'analyse est « construit »
  des feuilles vers la racine...

  Avantages LALR
  - récursivité à gauche et à droite
  - pas besoin de factorisation à gauche

  Inconvénients LALR
  - notion complexe de conflits donc difficile pour débogage
  - pas d'attributs hérités (que des synthétisés)

NB limites d'expressivité sur BNF souvent compensables
par des « conditions sur les attributs »