Analyse de Données Structurées - Cours 6

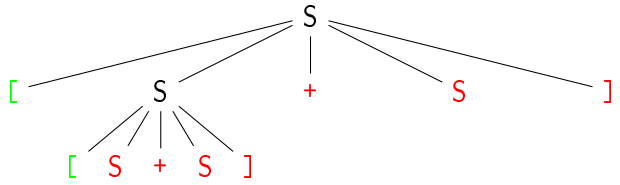
Ralf Treinen

Université Paris Diderot
UFR Informatique
Laboratoire Preuves, Programmes et Systèmes

treinen@pps.univ-paris-diderot.fr

11 mars 2015

Construction d'un arbre de dérivation



[	[	i	+	i	]	+	[	i	+	i	]	]

Choisir règle (1) : c'est la seule qui peut produire à partir de S un mot qui commence sur `i`.

Grammaires LL(1)

- Intuition derrière les grammaires LL(1) : dans la construction d'une dérivation gauche, le symbole suivant de l'entrée nous indique quelle règle de la grammaire appliquer au non-terminal le plus à gauche de l'arbre de dérivation.
- Conséquence : toute grammaire LL(1) (ou même LL(k)) est non-ambiguë (oublié de préciser au dernier cours).
- Un critère très simple : Si tous les côtés droites de la grammaire pour le *même* non-terminal commencent sur des terminaux différents, alors la grammaire est LL(1).

Une caractérisation des grammaires LL(k)

Théorème

Une grammaire $G = (V_T, V_N, S, R)$ est LL(k) ssi

- si $A \to \beta$ et $A \to \gamma$ sont deux règles différentes
- et $S \to^* wA\alpha$ une dérivation gauche, $w \in V_T^*$
- alors $\mathrm{FIRST}_k(\beta\alpha) \cap \mathrm{FIRST}_k(\gamma\alpha) = \emptyset$

- Preuve omise (conséquence immédiate de la définition)
- Attention, ce critère général prend en compte le "contexte" dans lequel on peut obtenir A.

Un meilleur critère pour être LL(1)

- Définition de $FIRST_1(\alpha)$: l'ensemble des symboles avec lesquelles un mot terminal dérivé à partir de α peut commencer (plus ϵ dans le cas où $\alpha \rightarrow^* \epsilon$).
- Vu au dernier cours : calcul de $FIRST_1$ dans le cas où aucun côté droite est ϵ.
- Meilleur critère : Si tous les côtés droites de la grammaire pour le *même* non-terminal ont des ensembles $FIRST_1$ disjointes, alors la grammaire est LL(1).

Exemple vu au dernier cours

- Grammaire $G = (\{\mathtt{a}, (,), +\}, \{\mathsf{F}, \mathsf{S}\}, \mathsf{S}, R)$ où R est

$$\begin{aligned} \mathsf{F} &\rightarrow \mathtt{a} \\ \mathsf{S} &\rightarrow (\mathsf{F}+\mathsf{S}) \\ \mathsf{S} &\rightarrow \mathsf{F} \end{aligned}$$

- Le première critère simple ne s'applique pas.
- On obtient pour les côtés droites des règles :

$$\begin{aligned} \mathrm{FIRST}_1(\mathtt{a}) &= \{\mathtt{a}\} \\ \mathrm{FIRST}_1((\mathsf{F}+\mathsf{S})) &= \{(\} \\ \mathrm{FIRST}_1(\mathsf{F}) &= \{\mathtt{a}\} \end{aligned}$$

Reconnaître la fin de l'entrée

- Le programme vu au dernier cours a un défaut : il accepte aussi des expressions correctes, avec un texte quelconque ajouté à la fin : `(a+(a+a))%$#@`
- Solution :
  - l'analyse lexicale envoie un jeton qui signale la fin de l'entrée (par exemple, EOF)
  - remplacer l'axiome par S′, avec une règle

$$\mathsf{S}' \rightarrow \mathsf{S}\ \mathtt{EOF}$$

où S est l'ancien axiome de la grammaire.

Le même exemple avec reconnaissance de la fin

- Grammaire $G = (\{\mathtt{a}, (,), +, \mathtt{EOF}\}, \{\mathsf{F}, \mathsf{S}, \mathsf{S}'\}, \mathsf{S}', R)$ où R est

$$\begin{aligned} \mathsf{F} &\rightarrow \mathtt{a} \\ \mathsf{S} &\rightarrow (\mathsf{F}+\mathsf{S}) \\ \mathsf{S} &\rightarrow \mathsf{F} \\ \mathsf{S}' &\rightarrow \mathsf{S}\ \mathtt{EOF} \end{aligned}$$

- On obtient pour les côtés droites des règles :

$$\begin{aligned} \mathrm{FIRST}_1(\mathtt{a}) &= \{\mathtt{a}\} \\ \mathrm{FIRST}_1((\mathsf{F}+\mathsf{S})) &= \{(\} \\ \mathrm{FIRST}_1(\mathsf{F}) &= \{\mathtt{a}\} \\ \mathrm{FIRST}_1(\mathsf{S}\ \mathtt{EOF}) &= \{\mathtt{a}, (\} \end{aligned}$$

Un exemple un peu plus avancé

- Une grammaire pour les expressions arithmétiques partiellement parenthésées.
- Terminaux : $\{\texttt{i}, \texttt{+}, \texttt{*}, \texttt{(}, \texttt{)}, \texttt{EOF}\}$
- Règles :

$$\begin{aligned} \mathsf{S} &\to \mathsf{E}\ \texttt{EOF} \\ \mathsf{E} &\to \mathsf{E}\texttt{+}\mathsf{T} \mid \mathsf{T} \\ \mathsf{T} &\to \mathsf{T}\texttt{*}\mathsf{F} \mid \mathsf{F} \\ \mathsf{F} &\to \texttt{(}\mathsf{E}\texttt{)} \mid \texttt{i} \end{aligned}$$

- Axiome : S

L'exemple des expressions partiellement parenthésées

- Cette grammaire est non-ambiguë ☺
- Intuition : Les "+" peuvent être produites seulement à partir du E. Tout mot engendré par E est protégé par des parenthèses (et).
- (Il y a aussi une preuve formelle mais je vous en fait grâce.)
- Cette grammaire, est-elle aussi LL(1) ? Ou au moins LL(k) pour quelque $k \in \mathbb{N}$?
- Elle n'est pas LL(k), pour aucun k ! ☹

Un critère pour ne pas *pas* être LL(k)

Définition

Une grammaire $G = (V_T, V_N, S, R)$ est *récursive à gauche* s'il y a un non-terminal $N \in V_N$ tel que $N \to^+ N\alpha$ pour un $\alpha \in (V_T \cup V_N)^*$.

- $\to^+$: dérivation en au moins une étape.
- Exemple : notre grammaire pour les expressions partiellement parenthésées, car $\mathsf{E} \to \mathsf{E}\texttt{+}\mathsf{T}$

Lemme

Si la grammaire G est récursive à gauche, alors G n'est pas *pas* LL(k), pour aucun $k \in \mathbb{N}$.

Preuve

- Supposons pour l'absurde que G est LL(k) et récursive à gauche.
- Il y a donc une dérivation gauche $S \to^* wX\gamma$ (sinon X n'est pas atteignable).
- Il y a aussi une règle $X \to X\alpha$ (hypothèse simplificatrice) , et une règle différente $X \to \beta$ (sinon X est non-productif).
- Il existe donc une dérivation gauche

$$S \to^* wX\gamma \to^* wX\alpha^k\gamma$$

- Par le théorème du début du cours :

$$\mathrm{FIRST}_k(X\alpha^{k+1}\gamma) \cap \mathrm{FIRST}_k(\beta\alpha^k\gamma) = \emptyset$$

Preuve (2)

- On a :

$$\mathrm{FIRST}_k(X\alpha^{k+1}\gamma) \cap \mathrm{FIRST}_k(\beta\alpha^k\gamma) = \emptyset$$

- Donc, grâce à la règle $X \to \beta$:

$$\mathrm{FIRST}_k(\beta\alpha^{k+1}\gamma) \cap \mathrm{FIRST}_k(\beta\alpha^k\gamma) = \emptyset$$

- Contradiction (deux cas : $\alpha \to^* \epsilon$ ou pas).

Quoi faire ?

- On peut transformer la grammaire en une grammaire équivalente (qui définit le même langage), et qui est LL(1).
- C'est toujours possible, mais il y a deux inconvénients :
  - la grammaire résultante peux être plus grande ;
  - la structure de l'arbre de dérivation peut changer.
- Il y a un troisième inconvénient : la transformation peut introduire des règles $N \to \epsilon$, il faut donc adapter la technique à ce cas.

La grammaire transformée

- Une grammaire pour les expressions arithmétiques partiellement parenthésées.
- Terminaux : $\{\texttt{i}, \texttt{+}, \texttt{*}, \texttt{(}, \texttt{)}, \texttt{EOF}\}$
- Règles :

$$\begin{aligned} \mathsf{S} &\to \mathsf{E}\ \texttt{EOF} \\ \mathsf{E} &\to \mathsf{T}\ \mathsf{E}' \\ \mathsf{E}' &\to \epsilon \mid \texttt{+}\mathsf{E} \\ \mathsf{T} &\to \mathsf{F}\ \mathsf{T}' \\ \mathsf{T}' &\to \epsilon \mid \texttt{*}\mathsf{T} \\ \mathsf{F} &\to \texttt{(}\mathsf{E}\texttt{)} \mid \texttt{i} \end{aligned}$$

- Axiome : S

Explication de la transformation

- Les deux règles originales pour le non-terminal E :

$$\begin{aligned} \mathsf{E} &\to \mathsf{T} \\ \mathsf{E} &\to \mathsf{E}\texttt{+}\mathsf{T} \end{aligned}$$

- Dans la grammaire d'origine, le non-terminal E engendre une séquence non-vide de non-terminaux T, séparés par des +.
- Dans la nouvelle grammaire, le non-terminal E′ engendre la suite de cette séquence après un T :

$$\begin{aligned} \mathsf{E} &\to \mathsf{T}\ \mathsf{E}' \\ \mathsf{E}' &\to \epsilon \mid \texttt{+}\mathsf{E} \end{aligned}$$

- Pareil pour le non-terminal T.

But d'une analyse d'une grammaire

- Détecter des propriétés des non-terminaux dans une grammaires, et des grammaires.
- Exemple : non-terminaux atteignables ou pas, productifs ou pas, pouvant engendrer le mot vide ou pas ? Grammaires LL(1), LL(k) ou pas ?
- Défis : récurrence entre des les non-terminaux d'une grammaire.
- Dans certains cas on maîtrise une descente récursive *sur un mot donné* pour la construction d'une dérivation - cas des grammaires LL(k).
- Dans une analyse de la grammaire, on a pas de mot d'entrée fixe donné mais on s'intéresse à une propriété générale de la grammaire.

Illustration du problème d'une descente récursive

- Un non-terminal N est *productif* s'il y a une règle $N \to \alpha$ telle que tous les non-terminaux dans α sont productifs. (C'est en particulier vrai quand α ne contient que des terminaux.)
- Imaginez les règles suivantes de la grammaire :

$$\begin{aligned} A &\to BC \mid a \\ B &\to Cb \\ C &\to Ac \end{aligned}$$

- Il faut éviter une descente récursive qui mène dans une boucle : A productif ? $\to$ B productif ? $\to$ C productif ?
  $\to$ A productif ? $\to \ldots$

Calcul d'un point fixe

- Notre définition de productivité est correcte, mais il faut organiser le calcul différemment :
  - Au début, on pose que tout non-terminal N pour lequel existe une règle $N \to w$, où $w \in V_T^*$, est productif.
  - Puis, s'il y a une règle $N \to \alpha$, où on a déjà reconnu tous les non-terminaux dans α comme étant productifs, alors on pose que N est aussi productif.
  - On s'arrête si on ne peu plus ajouter d'information de cette façon.
- Il s'agit d'un point fixe !

Points fixes

- En maths : un *point fixe* d'une fonction $f : D \to D$ est une valeur $x \in D$ telle que $f(x) = x$.
- Application à l'algorithmique :
  - le domaine D est l'ensemble de toutes les affectations possibles à des variables d'intérêt.
    Dans l'exemple : toutes les affectations possibles de P du type $V_N \to \texttt{boolean}$.
  - la fonction f est une mise à jour des variables.
    Dans l'exemple : propagation de l'information de productivité d'un non-terminal à un autre.
  - on ne cherche pas un point fixe quelconque, mais on commence avec une valeur initiale, puis on applique f jusqu'à un point fixe.

Calcul d'un point fixe

Nous utiliserons dans nos algorithmes la construction "do ... until X1,..., Xn fix". Exemple productivité :

```
for all N ∈ V_N :
  if exists (N → α) ∈ R tel que α ∈ V_T*
  then P(N) = true
  else P(N) = false
do
  for all (N → α) ∈ R :
    if P(M) for all non-terminals M in α
    then P(N) = true
until P fix
```


La construction do ... until ... fix

Le code

```
// code initialisation de X
do
  // code mise a jour de X
until X fix
```

peut être traduit vers :

```
// code initialisation de X
do
  Xold = X
  // code mise a jour de X
while (X != Xold)
```


Autre exemple symboles atteignables

- Un non-terminal N est *atteignable* (à partir de l'axiome S) s'il existe une dérivation $S \rightarrow^* \alpha N \beta$.
- Calcul des non-terminaux atteignables : transparent suivant.
- Une grammaire est *réduite* si
  - tous ses non-terminaux sont productifs et
  - tous ses non-terminaux sont atteignables.
- Dans la suite nous supposons que toutes les grammaires sont réduites.

Calcul des non-terminaux atteignables

Algorithme

Donnée une grammaire $G = (V_T, V_N, S, R)$.

```
for all N ∈ V_N :
  A(N) = false
A(S) = true
do
  for all (N → α) ∈ R :
    if A(N)
    then for all non-terminals M in α :
      A(M) = true
until A fix
```

Lemma

Pour tout $N \in V_N$: $A(N) == \texttt{true}$ ssi N est atteignable.

Non-terminaux qui peuvent produire ϵ

Algorithme

Donnée une grammaire $G = (V_T, V_N, S, R)$.

$$\begin{array}{l}\text{EPS} = \{N \in V_N \mid (N \to \epsilon) \in R\}\\ \texttt{do}\\ \quad \text{EPS} = \text{EPS} \cup \{N \in V_N \mid (N \to N_1 \dots N_n) \in R,\\ \qquad\qquad\qquad\qquad N_1, \dots, N_n \in \text{EPS}\}\\ \texttt{until EPS fix}\end{array}$$

Lemme

Pour tout $N \in V_N$: $N \in \text{EPS}$ ssi $N \to^* \epsilon$.

Calcul de EPS sur l'exemple

$$\begin{array}{lcllcl}\text{E} & \to & \text{T E}' & \text{E}' & \to & \epsilon \mid \texttt{+E}\\ \text{T} & \to & \text{F T}' & \text{T}' & \to & \epsilon \mid \texttt{*T}\\ \text{F} & \to & \texttt{(E)} \mid \texttt{i} & \text{S} & \to & \text{E } \texttt{EOF}\end{array}$$

- Initialisation : $EPS = \{\text{E}', \text{T}'\}$
- C'est déjà un point fixe !

Comment calculer $FIRST_1$ dans le cas général ?

- Imaginez une règle A $\to$ B C d E
- Tous les non-terminaux peuvent a priori engendrer ϵ.
- Si B $\notin EPS$: dans cette règle, seulement $FIRST_1$(B) peut contribuer à $FIRST_1$(A).
- Si B $\in EPS$: $FIRST_1$(C) peut aussi contribuer à $FIRST_1$(A).
- Si B $\in EPS$ et C $\in EPS$: d doit être dans $FIRST_1$(A).
- Dans aucun des cas, $FIRST_1$(E) ne peut contribuer car il se trouve derrière le terminal d.

Le calcul de $FIRST_1$ dans le cas général

- Pour des raisons techniques, il est plus facile de calculer d'abord une variante de $FIRST_1$ sans ϵ :

$$Fi(N) = \{a \in V_T \mid N \to^* aw, w \in V_T^*\}$$

- Puis on en obtient $FIRST_1(N)$ en utilisant EPS que nous avons déjà calculé.

Calcul de $FIRST_1$ dans le cas général

Algorithme

Donnée une grammaire $G = (V_T, V_N, S, R)$.

pour tout $N \in V_N$:
  Fi(N) = $\{a \in V_T \mid (N \to N_1 \dots N_n a\alpha) \in R, N_1, \dots, N_n \in \text{EPS}\}$
do
  pour tout $(N \to N_1 \dots N_n M\alpha) \in R$
  tel que $M \in V_N, N_1, \dots, N_n \in \text{EPS}$:
    Fi(N) = Fi(N) $\cup$ Fi(M)
until F fix

Lemme

Pour tout $N \in V_N$: $Fi(N) = FIRST_1(\text{N}) - \{\epsilon\}$

Calcul de Fi sur l'exemple

E	$\to$	T E′	E′	$\to$	$\epsilon \mid$ +E
T	$\to$	F T′	T′	$\to$	$\epsilon \mid$ *T
F	$\to$	(E) $\mid$ i	S	$\to$	E EOF

$EPS = \{\text{E}', \text{T}'\}$

Calcul

	Initial	*Iter*1	*Iter*2	*Iter*3
S	$\emptyset$	$\emptyset$	$\emptyset$	{i, (}
E	$\emptyset$	$\emptyset$	{i, (}	{i, (}
E′	{+}	{+}	{+}	{+}
T	$\emptyset$	{i, (}	{i, (}	{i, (}
T′	{*}	{*}	{*}	{*}
F	{i, (}	{i, (}	{i, (}	{i, (}

Calcul de $FIRST_1$ dans le cas général

Pour tout N $\in V_N$:

$$FIRST_i(\text{N}) = \begin{cases} Fi(N) \cup \{\epsilon\} & \text{si N} \in EPS \\ Fi(N) & \text{sinon} \end{cases}$$

Calcul de $FIRST_1$ sur l'exemple

E	$\to$	T E′	E′	$\to$	$\epsilon \mid$ +E
T	$\to$	F T′	T′	$\to$	$\epsilon \mid$ *T
F	$\to$	(E) $\mid$ i	S	$\to$	E EOF

$EPS = \{\text{E}', \text{T}'\}$

Calcul

	Fi	$FIRST_1$
S	{i, (}	{i, (}
E	{i, (}	{i, (}
E′	{+}	{+, ϵ}
T	{i, (}	{i, (}
T′	{*}	{*, ϵ}
F	{i, (}	{i, (}

Calcul de $FIRST_1$ dans le cas général

On étend maintenant $FIRST_1$ à des mots de terminaux et non-terminaux :

- $FIRST_1(\epsilon) = \{\epsilon\}$
- pour tout $a \in V_T$: $FIRST_1(a\alpha) = \{a\}$
- pour tout $N \in V_N$:

$$FIRST_1(N\alpha) = \begin{cases} FIRST_1(N) & \text{si } N \notin EPS \\ (FIRST_1(N) \setminus \{\epsilon\}) \cup FIRST_1(\alpha) & \text{si } N \in EPS \end{cases}$$

Nous avons besoin de plus d'information !

- Le calcul de $FIRST_1$ n'est plus suffisant pour savoir quelle production appliquer !
- Exemple : $E' \to \epsilon \mid +E$
  Si nous voyons + alors il faut utiliser la deuxième alternative pour réécrire E'. Mais quand faut-il appliquer la première ?
- Il nous manque une information : quel sont les symboles terminaux qui peuvent *suivre* à un mot produit par un non-terminal ?

La fonction $FOLLOW_k$

Définition

Soit $G = (V_T, V_N, S, R)$ une grammaire, $k \in \mathbb{N}$. La fonction $FOLLOW_k : V_N \to 2^{V_T^*}$ est définie par

$$FOLLOW_k(N) = \{w \mid S \to^* \beta N \gamma, \beta, \gamma \in (V_T \cup V_N)^*, w \in FIRST_k(\gamma)\}$$

Remarques

- $FOLLOW_k(N)$ est l'ensemble de tous les mots de terminaux de longueur k qui peuvent, dans des mots de $\mathcal{L}(G)$, suivre à un mot dérivé de N.

Calcul de $FOLLOW_1$

Algorithme

Donnée une grammaire $G = (V_T, V_N, S, R)$.

```
for all N ∈ V_N :
  Fo(N)={a ∈ V_T | (M → ... N N_1 ... N_k a ...) ∈ R, N_1, ..., N_n ∈ EPS}
for all (M → ... N N_1 ... N_k N' ...) ∈ R
with N_1, ..., N_k ∈ EPS :
  Fo(N) = Fo(N) ∪ Fi(N')
do
  for all (M → ... N N_1 ... N_k) ∈ R
  with N_1, ..., N_n ∈ EPS :
    Fo(N) = Fo(N) ∪ Fo(M)
until Fo fix
```

Lemme

Si $(S \to N\ \texttt{EOF}) \in R$, alors $\forall N \in V_N : Fo(N) = FOLLOW_1(N)$

Calcul de $FOLLOW_1$ sur l'exemple

$$\begin{array}{lcllcl} \mathrm{E} & \to & \mathrm{T\ E'} & \mathrm{E'} & \to & \epsilon \mid \mathtt{+E} \\ \mathrm{T} & \to & \mathrm{F\ T'} & \mathrm{T'} & \to & \epsilon \mid \mathtt{*T} \\ \mathrm{F} & \to & \mathtt{(E)} \mid \mathtt{i} & \mathrm{S} & \to & \mathrm{E\ EOF} \end{array}$$

$$EPS = \{\mathrm{E'}, \mathrm{T'}\} \quad Fi(\mathrm{E'}) = \{\mathtt{+}\} \quad Fi(\mathrm{T'}) = \{\mathtt{*}\}$$

Calcul

	Initial	*Iter*1	*Iter*2
S	$\emptyset$	$\emptyset$	$\emptyset$
E	{EOF,)}	{EOF,)}	{EOF,)}
E′	$\emptyset$	{EOF,)}	{EOF,)}
T	{+}	{+, EOF,)}	{+, EOF,)}
T′	$\emptyset$	{+}	{EOF,), +}
F	{*}	{*, EOF,), +}	{*, EOF,), +}

Comment choisir la règle dans l'analyse lexicale

Soit $A \to \alpha_1 \mid \ldots \mid \alpha_n$ une alternative. Il y deux cas :

1. Soit, aucun des $FIRST_1(\alpha_i)$ ne contient ϵ :
   comme avant :
   - On choisit la règle $A \to \alpha_i$ quand le symbole suivant est dans $FIRST_1(\alpha_i)$ (ils sont tous disjoints).
   - Erreur si aucun tel i existe
2. Soit α_i avec $\epsilon \in FIRST_i(\alpha_i)$:
   - si le symbole suivant est dans $FIRST_1(\alpha_j)$: choisir $A \to \alpha_j$, pour $1 \leq j \leq n$.
   - si le symbole suivant est dans $FOLLOW_1(A)$: choisir $A \to \alpha_i$.
   - sinon Erreur.

Finalement (!) : le critère pour être LL(1)

Théorème

La grammaire $G = (V_T, V_N, S, R)$ est LL(1) ssi pour toutes les alternatives $A \to \alpha_1 \mid \ldots \mid \alpha_n$:

- $FIRST_1(\alpha_1), \ldots, FIRST(\alpha_n)$ sont disjoints entr'eux ;
- Si $\epsilon \in FIRST_1(\alpha_i)$, alors pour tous $j \neq i$:

$$FIRST_1(\alpha_j) \cap FOLLOW_1(A) = \emptyset$$

Remarque

Condition (1) implique qu'au plus un des ensembles $FIRST_1(\alpha_i)$ contient ϵ.