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

Chapitre 4
Formalisation des analyses LL(1)

Xavier.Nicollin@grenoble-inp.fr

thanks Sylvain Boulmé *et* Lionel Rieg

Objectifs du chapitre

On formalise les intuitions données au chapitre 3

- Définition des directeurs et BNF LL(1)
- Algo pour décider si une BNF donnée est LL(1)
  Cet algo nécessite des calculs de + petit point fixe ou d'utiliser le lemme d'Arden
- Algo simple pour traduire BNF LL(1) en programme

Ici, on ne considère que des BNF réduites, i.e :

- sans non-terminal A improductifs : $\forall A, L(A) \neq \emptyset$
- sans non-terminal A inaccessibles :
  $\forall A, \exists \alpha_1, \alpha_2 \ in V^* : S \Rightarrow^* \alpha_1 \ A \ \alpha_2$

Ces non-terminaux sont faciles à détecter et éliminer (cf. TL1)

Chapitre 4 Formalisation des analyses LL(1)

Rappels du chapitre 3

On a vu comment implémenter l'analyse de la BNF G_1 ci-dessous

(1)	$S\uparrow w$ ::=	$O\uparrow w_1$ c	$w := w_1.\mathrm{a}$
(2)	$O\uparrow w$ ::=	$P\uparrow w_1$	$w := w_1.\mathrm{c}$
(3)	|	ε	$w := \varepsilon$
(4)	$P\uparrow w$ ::=	b	$w := \varepsilon$
(5)		a $P\uparrow w_1$ c $P\uparrow w_2$	$w := \mathrm{b}.w_1.\mathrm{c}.\mathrm{a}.w_2$

via l'interface

```
current # variable globale de pré-vision
def init_parser(...): # initialise current
def parse_token(attendu):  # vérif current == attendu
                           # et avance current
```

Mais si on remplace la règle (1) par « $S \to O$ a », alors l'analyse du mot "a" ne marche pas en suivant ce principe

Retour sur la notion de directeur LL(1)

Soit \$ $\notin V$: sentinelle de fin d'entrée (token `END`)
On pose $V_{T_\$} \stackrel{def}{=} V_T \cup \{\$\}$

Intuition : $\mathrm{Dir}(X \to \alpha)$ est le sous-ensemble de $V_{T_\$}$ auquel doit appartenir `current` pour que `parse_X` puisse appliquer $X \to \alpha$ dans une dérivation depuis « S \$ » (avec S axiome)

Exemple des directeurs LL(1) de G_1

{a, b, c}	S → O c	{a, b}	O → P	{b}	P → b
		{c}	O → ε	{a}	P → a P c P

Si on remplace (1) par S → O a, alors $\mathrm{Dir}(\mathsf{O} \to \varepsilon) = \{\mathtt{a}\}$
La grammaire n'est plus LL(1), car `parse_O` ne peut plus choisir entre O → P et O → ε si `current` == a

Exo 1 Donner les directeurs LL(1) de la grammaire S ::= a S b | ε

Définition des grammaires LL(1) (Lewis/Stearns 1968)

Soit G une GHC **réduite** quelconque

Définition (Directeur $\mathrm{Dir}(\rho)$ d'une règle $\rho = \mathsf{A} \to \alpha$)

$\mathrm{Dir}(\rho)$ est défini comme l'ensemble des $a \in V_{T_\$}$ pour lesquels il existe $w_1 \in {V_T}^*$, w_2 et $w_3 \in {V_{T_\$}}^*$ tels que

$$S\ \$ \Rightarrow^* w_1\ \mathsf{A}\ w_2 \quad \text{et} \quad \alpha\ w_2 \Rightarrow^* a\ w_3$$

Intuition : Dans une dérivation depuis « S \$ », une règle $A \to \alpha$ ne peut dériver « $a\ w$ » que si $a \in \mathrm{Dir}(A \to \alpha)$

NB : Dans la déf ci-dessus, on peut avoir $\alpha \Rightarrow^* \varepsilon$
G réduite implique $\mathrm{Dir}(\rho) \neq \emptyset$

Definition (Grammaire LL(1))

G est dite LL(1) ssi $\forall A \to \alpha$ et $A \to \beta$ **distinctes** (c.-à-d. $\alpha \neq \beta$) et **de même membre gauche** A, $\mathrm{Dir}(A \to \alpha) \cap \mathrm{Dir}(A \to \beta) = \emptyset$

NB : G LL(1) implique G non ambiguë

Calcul du directeur des règles

On décompose le calcul en :

$$\mathrm{Dir}(A \to \alpha) \stackrel{def}{=} \mathrm{Prem}(\alpha) \cup \mathcal{E}(\alpha).\mathrm{Suiv}(A)$$

où

1. $\mathrm{Prem}(\alpha) = \{a \in V_T \mid$ il existe $w \in {V_T}^*$ tq $\alpha \Rightarrow^* a\ w\}$
2. $\mathcal{E}(\alpha) =$ « si $\alpha \Rightarrow^* \varepsilon$ alors $\{\varepsilon\}$ sinon $\emptyset$ »
   $= \mathcal{L}_G(\alpha) \cap \{\varepsilon\}$
   Donc : si α **peut** dériver vers ε ($\mathcal{E}(\alpha) = \{\varepsilon\}$)
   alors on doit s'intéresser à $\mathrm{Suiv}(A)$
   sinon ce n'est pas la peine (car $\emptyset.\mathrm{Suiv}(A) = \emptyset$!) !
3. $\mathrm{Suiv}(A)$ est l'ensemble des $a \in {V_T}_\$$ tels qu'il existe $w_1 \in {V_T}^*$ et $w_2 \in {V_T}_\$^*$ avec $S\ \$ \Rightarrow^* w_1\ A\ a\ w_2$

À venir calcul de $\mathcal{E}$, Prem et Suiv comme + petits point fixes

Chapitre 4 Formalisation des analyses LL(1)

Introduction aux + petits points fixes

Déf. Soit E un ensemble et f une application de $\mathcal{P}(E) \to \mathcal{P}(E)$
Un point fixe de f est un $X \subseteq E$ tq $X = f(X)$
X + petit point fixe $\Leftrightarrow$ X point fixe de f
pour tout point fixe Y de f, on a $X \subseteq Y$

Exo 2 Soit $V \stackrel{def}{=} \{a, b\}$. Pour chacune des équations suivantes, quel est le + petit $X \subseteq V^*$ qui la vérifie ?

- $X = \{a\}.X \cup \{b\}$ ...
- $X = \{a\}.X \cup X$ ...
- $X = \{a\}.X.\{b\} \cup \{\varepsilon\}$ ...

Autres exemples :

- les définitions inductives ($f(X) = \text{constructeurs}(X) \cup \text{base}$)
- le langage d'une grammaire ($f \approx$ un pas de dérivation)

Thm des + petits points fixes de Kleene (1938)

Soit E un ensemble et f une application de $\mathcal{P}(E) \to \mathcal{P}(E)$

Définition (Fonction continue)

f est continue $\stackrel{def}{=}$ pour toute suite croissante $(A_i)_{i\in\mathbb{N}}$ de $\mathcal{P}(E)$ $(\forall i, A_i \subseteq A_{i+1})$, on a $f(\bigcup_{i\in\mathbb{N}} A_i) = \bigcup_{i\in\mathbb{N}} f(A_i)$

NB : f continue implique f croissante : $X \subseteq Y \Rightarrow f(X) \subseteq f(Y)$
Réciproquement, toutes les fonctions croissantes « usuelles » sont continues La composée de deux fonctions continues est continue

Théorème (Kleene, 1938)

Si f est continue, alors le + petit point fixe de f est $\bigcup_{i\in\mathbb{N}} f^i(\emptyset)$

NB :

- f croissante donne $f^i(\emptyset) \subseteq f^{i+1}(\emptyset)$ et $\bigcup_{0\leqslant i\leqslant k} f^i(\emptyset) = f^k(\emptyset)$
- S'il existe k tq $f^k(\emptyset) = f^{k+1}(\emptyset)$, alors $\forall i \geqslant k, f^i(\emptyset) = f^k(\emptyset)$ et $\bigcup_{i\in\mathbb{N}} f^i(\emptyset) = f^k(\emptyset)$
  En particulier, si E est de cardinal fini n, alors $k \leqslant n$

technique de calcul

Exemples de points fixes

- **Exemple 1 :** pour $f : X \mapsto \{a\}.X.\{b\} \cup \{\varepsilon\}$

  $$f^i(\emptyset) = \{a^k.b^k \mid k < i\}$$

  et $$\bigcup_{i \in \mathbb{N}} f^i(\emptyset) = \{a^k.b^k \mid k \in \mathbb{N}\}$$

- **Exemple 2 :** les langages hors-contexte

  f = « premier pas de dérivation »
  = « le premier étage de l'arbre d'analyse »

  $f^i(\emptyset)$ = mots dérivables par un arbre de hauteur $\leqslant i$

  $\bigcup_{i \in \mathbb{N}} f^i(\emptyset)$ = mots dérivables par un arbre quelconque = $\mathcal{L}(G)$

  Si $X \to w_1 \mid \ldots \mid w_n$,

  $$f : \mathcal{P}(V_T^*) \to \mathcal{P}(V_T^*)$$
  $$X \mapsto \mathcal{L}(w_1) \cup \cdots \cup \mathcal{L}(w_n)$$

  Et pour plusieurs non-terminaux ?

Application aux systèmes d'équations ensemblistes

Applicable à f de $\mathcal{P}(E_1) \times \ldots \times \mathcal{P}(E_n) \rightarrow \mathcal{P}(E_1) \times \ldots \times \mathcal{P}(E_n)$
car $\mathcal{P}(E_1) \times \ldots \times \mathcal{P}(E_n) \simeq \mathcal{P}(\{1\} \times E_1 \cup \ldots \cup \{n\} \times E_n)$

Extension de $\subseteq$ et $\cup$: composante par composante
Pour $(X_1, \ldots, X_n)$ et $(Y_1, \ldots, Y_n)$ de $\mathcal{P}(E_1) \times \ldots \times \mathcal{P}(E_n)$, on a

$$(X_1, \ldots, X_n) \subseteq (Y_1, \ldots, Y_n) \stackrel{def}{=} \forall i, X_i \subseteq Y_i$$

$$(X_1, \ldots, X_n) \cup (Y_1, \ldots, Y_n) \stackrel{def}{=} (X_1 \cup Y_1, \ldots, X_n \cup Y_n)$$

Exemple Pour $V_T \stackrel{def}{=} \{\mathtt{a}, \mathtt{b}\}$,

la BNF

$S ::= \mathtt{b} \mid D\,D$
$D ::= \mathtt{a}\,S$

correspond au système

$$\begin{pmatrix} S \\ D \end{pmatrix} = \begin{pmatrix} \{\mathtt{b}\} \cup D.D \\ \{\mathtt{a}\}.S \end{pmatrix}$$

Le + petit point fixe de ce système sur $\mathcal{P}(V_T{}^*) \times \mathcal{P}(V_T{}^*)$ est (L_S, L_D) où L_S et L_D sont les langages engendrés par S et D

Le langage d'une grammaire HC s'exprime comme un point fixe

Calcul de l'image d'un + petit point fixe par commutation

Lemme de commutation

Pour $k \in \{1, 2\}$, soit f_k applications continues de $\mathcal{P}(E_k) \to \mathcal{P}(E_k)$
Soit g application *continue* de $\mathcal{P}(E_1) \to \mathcal{P}(E_2)$ avec

$$g(\emptyset) = \emptyset \qquad \text{et} \qquad g \circ f_1 = f_2 \circ g$$

Alors, on a :

$$g(\bigcup_{i \in \mathbb{N}} f_1^i(\emptyset)) = \bigcup_{i \in \mathbb{N}} f_2^i(\emptyset)$$

Application 1 : une autre démonstration du **lemme d'Arden**

Pour $A, B \subseteq V^*$, soient $\begin{cases} f_1(X) \stackrel{def}{=} A.X \cup \{\varepsilon\} \\ f_2(Y) \stackrel{def}{=} A.Y \cup B \\ g(X) \stackrel{def}{=} X.B \end{cases}$

On a $g(\emptyset) = \emptyset \qquad \text{et} \qquad g(f_1(X)) = f_2(g(X))$

Par définition, $A^* = \bigcup_{i \in \mathbb{N}} f_1^i(\emptyset)$

Par le lemme, $\bigcup_{i \in \mathbb{N}} f_2^i(\emptyset) = g\left(\bigcup_{i \in \mathbb{N}} f_1^i(\emptyset)\right) = g(A^*) = A^*.B$

Application 2 : décider $\varepsilon \in L$ avec L hors-contexte

On calcule $\mathcal{E}(L) \stackrel{def}{=} L \cap \{\varepsilon\}$ par commutation de + petit point fixe

Commençons par un exemple :

$S ::= \mathtt{b} \mid D\,D$
$D ::= \mathtt{a}\,S$

$$f_1 : \mathcal{P}(V^*) \times \mathcal{P}(V^*) \to \mathcal{P}(V^*) \times \mathcal{P}(V^*)$$
$$\begin{pmatrix} S \\ D \end{pmatrix} \mapsto \begin{pmatrix} \{\mathtt{b}\} \cup D.D \\ \{\mathtt{a}\}.S \end{pmatrix}$$

On veut calculer $\mathcal{E}(\mathcal{L}(S))$ et $\mathcal{E}(\mathcal{L}(D))$.

On pose $g = \mathcal{E} : \mathcal{P}(V^*) \to \mathcal{P}(\{\varepsilon\})$ On vérifie que $\mathcal{E}(\emptyset) = \emptyset$.

On cherche $f_2 : \mathcal{P}(\{\varepsilon\}) \to \mathcal{P}(\{\varepsilon\})$ tq

$$f_2(\mathcal{E}\begin{pmatrix} S \\ D \end{pmatrix}) = \mathcal{E}\begin{pmatrix} \{\mathtt{b}\} \cup D.D \\ \{\mathtt{a}\}.S \end{pmatrix} = \begin{pmatrix} \mathcal{E}(\{\mathtt{b}\}) \cup \mathcal{E}(D.D) \\ \mathcal{E}(\{\mathtt{a}\}) \cap \mathcal{E}(S) \end{pmatrix}$$

puis on calcule $\bigcup_{i \in \mathbb{N}} f_2^i(\emptyset)$ $\{\varepsilon\}$ est fini donc le calcul termine !

Application 2 : décider $\varepsilon \in L$ avec L hors-contexte

On calcule $\mathcal{E}(L) \stackrel{def}{=} L \cap \{\varepsilon\}$ par commutation de + petit point fixe

Système à résoudre obtenu en transformant chaque équation « $X_k ::= e_k$ » de la BNF de L en équation « $\mathcal{E}(X_k) = \mathcal{E}(e_k)$ »
où $\mathcal{E}(X_k)$ est vue comme une variable
et $\mathcal{E}(e_k)$ défini par induction structurelle sur e_k pour s'exprimer en fonction de $\mathcal{E}(X_1), \ldots, \mathcal{E}(X_n)$:

- $\mathcal{E}(\varepsilon) = \{\varepsilon\}$
- pour tout terminal a, $\mathcal{E}(a) = \emptyset$
- $\mathcal{E}(\alpha.\beta) = \mathcal{E}(\alpha) \cap \mathcal{E}(\beta)$
- $\mathcal{E}(\alpha \mid \beta) = \mathcal{E}(\alpha) \cup \mathcal{E}(\beta)$

Calcul du + petit point fixe en au plus $|V_N|$ itérations

ou par éliminations successives en exploitant la propriété suivante :

la + petite solution de $X = (X \cap \alpha) \cup \beta$ *est* β

(variante du lemme d'Arden !)

Application de cet exemple

Exo 3 Appliquer cette méthode sur la BNF

```
S ::= K P L
K ::= P L | P a L | K L P
P ::= L b | K L
L ::= L c P K | ε
```

...

Exo 4 Idem en remplaçant l'équation de P ci-dessus par

```
P ::= L b | K L | L L
```

...

Chapitre 4 Formalisation des analyses LL(1)

Calcul de Prem

Même méthode que $\mathcal{E}$ en ramenant le calcul à

$$\mathrm{Prem}(L) = \{a \in V_T \mid \exists w \in V_T{}^*,\ a\,w \in L\}$$

Système d'équations à résoudre

Transformation de chaque équation « $X_k ::= e_k$ » de la BNF en équation « $\mathrm{Prem}(X_k) = \mathrm{Prem}(e_k)$ » où $\mathrm{Prem}(e_k)$ est calculé par induction structurelle sur la syntaxe de e_k :

- Pour tout $a \in V_T$, $\mathrm{Prem}(a) = \{a\}$
- $\mathrm{Prem}(\varepsilon) = \emptyset$
- $\mathrm{Prem}(\alpha.\beta) = \mathrm{Prem}(\alpha) \cup \mathcal{E}(\alpha).\mathrm{Prem}(\beta)$
- $\mathrm{Prem}(\alpha \mid \beta) = \mathrm{Prem}(\alpha) \cup \mathrm{Prem}(\beta)$

Calcul de Suiv

Système d'équations variables $(\mathrm{Suiv}(X))_{X \in V_N}$ d'équation

$$\mathrm{Suiv}(X) = \begin{array}{l} \text{si } X \text{ axiome alors } \{\$\} \text{ sinon } \emptyset \\ \cup \\ \bigcup_{Y \to \alpha.X.\beta \in \mathcal{R}} \mathrm{Prem}(\beta) \cup \mathcal{E}(\beta).\mathrm{Suiv}(Y) \end{array}$$

NB : Dans l'union ci-dessus des règles de la forme $Y \to \alpha.X.\beta$, une même règle de $\mathcal{R}$ est utilisée autant de fois que X apparaît dans le membre droit

Exemple Pour "$X = \mathsf{C}$", la règle $\mathsf{Z} \to \mathtt{a}.\mathsf{C}.\mathtt{d}.\mathsf{C}$ compte avec

$$Y = \mathsf{Z} \quad \alpha = \mathtt{a} \quad \beta = \mathtt{d}.\mathsf{C}$$
$$\text{et } Y = \mathsf{Z} \quad \alpha = \mathtt{a}.\mathsf{C}.\mathtt{d} \quad \beta = \varepsilon.$$

Exemple

Exo 5 La BNF G_3 ci-dessous est-elle LL(1) ?

...

... $S ::= X$

... $\quad | \; Y\ \mathtt{c}$

... $X ::= \mathtt{a}\ X\ \mathtt{b}$

... $\quad | \; \varepsilon$

... $Y ::= \mathtt{b}\ Y$

... $\quad | \; \varepsilon$

...

Chapitre 4 Formalisation des analyses LL(1)

Cadre de la présentation (généralisant un peu le chapitre 3)

Petite généralisation : les tokens synthétisent un attribut (cf. « NAT↑n » en TD) éventuellement None

Machine à états de l'analyseur

```
current # variable globale du token de pré-vision

def init_parser(stream):  # init de 'current'

def parse_token(expected):
# vérifie 'current==expected'
# avance 'current' sur token suivant
# et retourne l'attribut synthétisé par
#    consommation de 'expected'
```

Principe de l'analyseur récursif

Analyseur donné par procédures *mutuellement récursives*
Intuition : arbre d'analyse reconnu = arbre des appels récursifs

Ainsi, pour tout non-terminal d'équation $X{\downarrow}h_1 \ldots {\downarrow}h_n{\uparrow}s_1 \ldots {\uparrow}s_m$,

```
def parse_X(h1, ..., hn):
```

reconnaît le +**long préfixe** de l'entrée qui correspond à X et retourne un m-uplet d'attributs synthétisés $(s_1, \ldots, s_m)$

Attention :

- `parse_X` lit le 1er token dans `current`
  En sortie, `current` sur le 1er token qui suit le préfixe reconnu
- Échec de `parse_X` $\Rightarrow$ soit pas de préfixe de X dans l'entrée, soit +long préfixe pas suivi d'un token de $\mathrm{Suiv}(X)$

Codage "automatique" des équations

Équation annotée avec directeurs

$\{a_1, \ldots, a_n\}$ $X{\downarrow}h{\uparrow}s ::= A{\uparrow}s$ c

$\{b_1, \ldots, b_m\}$ | $C{\downarrow}h{\uparrow}s_0$ e$\uparrow s_1$ $D{\downarrow}s_0{\downarrow}f(h, s_1){\uparrow}s_2$ $s := g(s_1, s_2)$

traduite en

```
def parse_X(h):
    if current in [a1, ..., an]:
        s = parse_A()
        parse_token(c)
        return s
    elif current in [b1, ... ,bm]:
        s0 = parse_C(h);
        s1 = parse_token(e);
        s2 = parse_D(s0, f(h, s1));
        return g(s1, s2)
    else:
        raise Error(...)
```

Incompatible avec dépendance droite/gauche dans calcul d'attributs

Restriction correspondant aux "*grammaires L-attribuées*"

Exemple avec variation sur le message d'erreur

Pour l'équation

$$\begin{array}{lll} \{a\} & X ::= & a\ X\ b \\ \{b, \$\} & \mid & \varepsilon \end{array}$$

3 choix d'implémen. qui changent juste les messages d'erreur

```
def parse_X():
  if current in [b,END]:
    return
  else:
    parse_token(a)
    parse_X()
    parse_token(b)
```

```
def parse_X():
  if current in [a]:
    parse_token(a)
    parse_X()
    parse_token(b)
  else:
    return
```

```
def parse_X():
  if current in [a]:
    parse_token(a)
    parse_X()
    parse_token(b)
  elif current in [b,END]:
    return
  else:
    raise Error("a b END")
```

Exemple d'erreur

aac
↑ attend a

aac
↑ attend b

aac
↑ attend a b END

Conclusion du chapitre

- Exemple complet de G_1 dans `chap3.py` sur Chamilo
Les sélections coûteuses "`if ... elif ... else: ...`" sont remplaçables par un accès efficace dans un tableau de règles (créé via le `mk_rules` du chapitre 3)

- Généralisation possible à une analyse qui continue malgré erreurs de syntaxe
Mais hors du cadre de TL2

- Prochain chapitre : comment construire des BNF LL(1) ?