TME3 : Grammaires réduites

(définitions, exemples, programmation en Python)

– version 2.0 (Janvier 2022) –

Mathieu.Jaume@lip6.fr

1 Grammaires hors-contexte

Le code présenté dans cette section se trouve dans le fichier `ghc.py`.

1.1 Définition et représentation

Une *grammaire hors-contexte* (GHC) G est un quadruplet $G = (V, \Sigma, R, S)$ où V est un ensemble de *symboles non-terminaux*, Σ est un ensemble de *symboles terminaux*, R est un ensemble de *productions* de la forme $X \to u$ où $X \in V$ et $u \in (V \cup \Sigma)^\star$ et $S \in V$ est l'*axiome* (i.e. le symbole de départ) de G.

Ensemble des productions de G On choisit ici de représenter l'ensemble R des productions d'une GHC G par une liste contenant pour chaque $X \in V$ la paire (X, L) où L est une liste contenant pour chaque production $X \to u$ la liste $[u]$. Par exemple, si X est associé aux 3 productions $X \to Yb \mid ab \mid \varepsilon$, la liste représentant R contiendra la paire $(X, [[Y, b], [a, b], []])$. On impose ici que la liste représentant R contienne au plus une paire (X, L) pour chaque $X \in V$.

Fonction d'égalité sur les symboles non-terminaux de G La définition d'une GHC G contient la définition d'une fonction d'égalité sur les symboles de V. Ceci est utile lorsque les symboles de V ne sont pas de type atomique (c'est par exemple le cas lorsque la GHC est obtenue à partir d'un automate à pile en suivant la construction présentée dans le cours). En revanche, nous supposons ici que l'égalité sur les symboles de Σ peut être testée avec la fonction d'égalité sur les types atomiques.

Représentation d'une GHC G Une GHC `G` est représentée par un tuple `(nt,t,r,si,eqnt)` où `nt` est une liste représentant un ensemble de symboles non-terminaux, `t` est une liste représentant un ensemble de symboles terminaux, `r` une liste représentant l'ensemble des productions, `si` est l'axiome et `eqnt` est la fonction d'égalité sur les symboles non-terminaux.

Exemple 1.1 La grammaire $G_1 = (V_1, \Sigma_1, R_1, S)$ où $V_1 = \{S, A_1, \cdots, A_9\}$, $\Sigma_1 = \{a, b, c\}$ et :

$$R_1 = \left\{ \begin{array}{lcl} S \to A_1 \mid A_2 \mid A_3 & , & A_1 \to aA_1A_1 \mid aA_2A_4 \mid aA_3A_7 \mid A_4 \\ A_2 \to aA_1A_2 \mid aA_2A_5 \mid aA_3A_8 \mid A_5 & , & A_3 \to aA_1A_3 \mid aA_2A_6 \mid aA_3A_9 \mid A_6 \\ A_4 \to bA_4 & , & A_5 \to bA_5 \\ A_6 \to bA_6 \mid \varepsilon & , & A_9 \to c \end{array} \right\} \quad (1)$$

est représentée comme suit :

exemple : GHC G_1

```
g1_nt = ["S","A1","A2","A3","A4""A5","A6","A7","A8","A9"]
g1_t = ["a","b","c"]
g1_r = [("S",[["A1"],["A2"],["A3"]]),\
        ("A1",[["a","A1","A1"],["a","A2","A4"],["a","A3","A7"],["A4"]]),\
        ("A2",[["a","A1","A2"],["a","A2","A5"],["a","A3","A8"],["A5"]]),\
        ("A3",[["a","A1","A3"],["a","A2","A6"],["a","A3","A9"],["A6"]]),\
        ("A4",[["b","A4"]]),\
        ("A5",[["b","A5"]]),\
        ("A6",[["b","A6"],[]]),\
        ("A9",[["c"]])]
g1_s = "S"
g1_g = (g1_nt,g1_t,g1_r,g1_s,eq_atom)
```

Egalité sur les parties droites de productions La fonction `make_eq_prod` permet de construire une fonction d'égalité sur les parties droites des productions d'une grammaire dont les symboles non-terminaux de `nt` sont comparables avec la fonction d'égalité `eqnt`.

égalité sur les parties droites de productions

```
def make_eq_prod(nt,eqnt):
    def _eq_prod(p1,p2):
        if p1==[] or p2==[]:
            return p1==[] and p2==[]
        else:
            b1 = is_in(eqnt,p1[0],nt)
            b2 = is_in(eqnt,p2[0],nt)
            if b1 and b2:
                return eqnt(p1[0],p2[0]) and _eq_prod(p1[1:],p2[1:])
            else:
                if ((not b1) and b2) or (b1 and (not b2)):
                    return False
                else:
                    return eq_atom(p1[0],p2[0]) and _eq_prod(p1[1:],p2[1:])
    return _eq_prod
```

On définit également une fonction `add_prod` qui permet d'ajouter une production à un ensemble de productions.

ajout d'une production à un ensemble de production

```
def add_prod(ns,nl,nt,r,eqnt):
    # ns : symbole non terminal
    # nl : partie droite de production
    # nt : symboles non terminaux
    # r : liste de productions
    # eqnt : egalite sur les non terminaux
    new_nr = []
    r_aux = r
    while not(r_aux == []):
        s,ls = r_aux[0]
        if eqnt(s,ns):
            ls = ajout(make_eq_prod(nt,eqnt),nl,ls)
            new_nr = new_nr + [(s,ls)] + r_aux[1:]
            return new_nr
        else:
            new_nr = new_nr + [(s,ls)]
            r_aux = r_aux[1:]
    new_nr = new_nr + [(ns,[nl])]
    return new_nr
```

Enfin, on définit une fonction `prods_s` qui permet de construite la liste des parties droites des productions d'un symbole non-terminal.

parties droites des productions d'un symbole non-terminal

```
def prods_s(r,eqnt,x):
    # r : ensemble de production
    # eqnt : egalite sur les non terminaux
    # x : symbole non terminal
    for s,ls in r:
        if eqnt(x,s):
            return ls
    return []
```

2 Grammaires hors-contexte réduites

Le code présenté dans cette section se trouve dans le fichier `reduced_ghc.py`.

Soit $G = (V, \Sigma, R, S)$ une grammaire hors-contexte engendrant un langage non vide. Un symbole $X \in V$ est *utile* s'il existe un mot $w \in \Sigma^\star$ tel que $S \Mapsto^* \alpha X \beta \Mapsto^* w$ avec $\alpha, \beta \in (\Sigma \cup V)^\star$. Une grammaire hors-contexte *réduite* est une grammaire ne contenant pas de symbole non-terminal inutile (elle peut être obtenue en supprimant les productions contenant des symboles *non-productifs* ou des symboles *non-accessibles*).

2.1 Elimination des symboles non-productifs

Un symbole $X \in V$ est *productif* s'il existe un mot $w \in \Sigma^\star$ tel que $X \Mapsto^* w$. L'ensemble $\mathbb{P}(G) \subseteq V$ des symboles productifs de G est obtenu en construisant la suite croissante et bornée (car V est fini) :

$$\begin{array}{l} P_0(G) = \{X \mid X \to w \in R \text{ avec } w \in \Sigma^\star\} \\ P_{n+1}(G) = P_n(G) \cup \{X \mid X \to u \in R \text{ et } u \in (\Sigma \cup P_n(G))^\star\} \end{array} \quad \mathbb{P}(G) = \bigcup_{n \geq 0} P_n(G)$$

Il s'agit donc de la construction d'un point fixe qui permet de définir la grammaire hors-contexte $\mathcal{P}(G) = (\mathbb{P}(G), \Sigma, R', S)$ où R' est obtenue en supprimant de R les productions contenant des symboles non-productifs. La grammaire $\mathcal{P}(G)$ engendre le même langage que G et on a donc $\mathcal{L}(G) = \mathcal{L}(\mathcal{P}(G))$. Lorsque $S \notin \mathbb{P}(G)$, le langage engendré par G est vide (i.e. $\mathcal{L}(G) = \emptyset$).

Exemple Considérons la grammaire hors-contexte G_1 de l'exemple 1.1. La suite $(P_n(G_1))_{n \in \mathbb{N}}$ est construite comme suit :

$$P_0(G_1) = \{A_6, A_9\} \quad P_1(G_1) = \{A_3, A_6, A_9\} \quad P_2(G_1) = \{S, A_3, A_6, A_9\} = P_3(G_1) = \mathbb{P}(G_1)$$

En éliminant les productions contenant des symboles de $V_1 \setminus \mathbb{P}(G_1) = \{A_1, A_2, A_4, A_5, A_7, A_8\}$, on obtient la GHC $G_3 = \mathcal{P}(G_1) = (V_3, \Sigma, R_3, S)$ où $V_3 = \{S, A_3, A_6, A_9\}$ et

$$R_3 = \{S \to A_3 \ , \quad A_3 \to aA_3A_9 \mid A_6 \ , \quad A_6 \to bA_6 \mid \varepsilon \ , \quad A_9 \to c\} \tag{2}$$

Implantation Pour construire la grammaire $\mathcal{P}(G)$, on calcule tout d'abord l'ensemble $\mathbb{P}(G)$. Ce calcul est effectué à partir de l'ensemble $P_0(G)$ en appliquant une fonction `next_prod` permettant de construire $P_{n+1}(G)$ à partir de $P_n(G)$ jusqu'à obtenir un point fixe, c-à-d un ensemble $P_{n+1}(G)$ égal à $P_n(G)$.

Calcul de $P_0(G)$ Ce calcul est effectué à partir de la liste `t` représentant Σ et de la liste `r` des productions de G.

à compléter : calcul de $P_0(G)$

```
def prod0(t,r):
    # t : symboles terminaux
    # r : liste de productions
```

exemple : $P_0(G_1)$

```
>>> prod0(g1_t,g1_r)
['A6', 'A9']
```

Calcul de $P_{n+1}(G)$ à partir de $P_n(G)$ La fonction `next_prod` effectue ce calcul. Outre l'ensemble `prev` à partir duquel est effectué ce calcul, cette fonction utilise la liste `t` représentant Σ, la liste `r` des productions de G, et la fonction d'égalité `eqnt` sur V.

à compléter : calcul de $P_{n+1}(G)$ à partir de $P_n(G)$

```
def next_prod(t,r,eqnt,prev):
    # t : symboles terminaux
    # r : liste de productions
    # eqnt : egalite sur les non terminaux
    # prev : liste de non terminaux de depart
```

exemple : $P_1(G_1)$ et $P_2(G_1)$

```
>>> next_prod(g1_t,g1_r,eq_atom,prod0(g1_t,g1_r))
['A3', 'A6', 'A9']
>>> next_prod(g1_t,g1_r,eq_atom,next_prod(g1_t,g1_r,eq_atom,prod0(g1_t,g1_r)))
['S', 'A3', 'A6', 'A9']
```

Calcul de $\mathbb{P}(G)$ Ce calcul correspond à la construction itérative d'un point fixe (et utilise donc la fonction `fixpoint_from` définie dans la section « Construction itérative d'un point fixe » de l'annexe sur la représentation des ensembles) à partir des fonctions `prod0` et `next_prod`. Détecter si un point fixe a été obtenu nécessite ici de tester l'égalité entre deux ensembles. La fonction `make_eq_set` (définie dans la section « Inclusion et égalité sur les ensembles » de l'annexe sur la représentation des ensembles) permet de définir une fonction d'égalité sur des ensembles qui dépend de l'égalité sur les éléments de ces ensembles.

à compléter : calcul de $\mathbb{P}(G)$

```
def prod(t,r,eqnt):
    # t : symboles terminaux
    # r : liste de productions
    # eqnt : egalite sur les non terminaux
```

exemple : $\mathbb{P}(G_1)$

```
>>> prod(g1_t,g1_r,eq_atom)
['S', 'A3', 'A6', 'A9']
```

Calcul de $\mathcal{P}(G)$ Ce calcul s'effectue en éliminant les symboles non-productifs des productions de G. On définit donc tout d'abord une fonction permettant d'éliminer d'une grammaire les symboles non-terminaux (et les productions qui les contiennent) n'appartenant pas à un ensemble donné.

Suppression de non-terminaux La fonction `remove_nt` permet de supprimer de l'ensemble des non-terminaux d'une grammaire `g` tous les symboles n'appartenant pas à l'ensemble `ent`. Les productions de `g` contenant des symboles n'appartenant pas à `ent` sont également supprimées.

suppression de non-terminaux

```
def remove_nt(g,ent):
    # g : ghc
    # ent : symboles non terminaux
    nt,t,r,si,eqnt = g
    def is_terminal(x):
        return is_in(eq_atom,x,t)
    def is_in_ent(x):
        return is_in(eqnt,x,ent)
    def is_in_ent_or_terminal(x):
        return is_in_ent(x) or is_terminal(x)
    def all_in_ent_or_terminal(l):
        return forall_such_that(l,is_in_ent_or_terminal)
    new_r = [(s,[l for l in ls if all_in_ent_or_terminal(l)])
             for s,ls in r if is_in_ent(s)]
    if is_in_ent(si):
        return (ent,t,new_r,si,eqnt)
    else:
        return (ent,t,new_r,None,eqnt)
```

à compléter : calcul de $\mathcal{P}(G)$

```
def remove_non_prod(g):
    # g : ghc
```

exemple : $\mathcal{P}(G_1)$

```
>>> g3_g = remove_non_prod(g1_g)
>>> g3_nt,g3_t,g3_r,g3_si,g3_eqnt = g3_g
>>> g3_nt
['S', 'A3', 'A6', 'A9']
>>> g3_r
[('S', [['A3']]), ('A3', [['a', 'A3', 'A9'], ['A6']]),
 ('A6', [['b', 'A6'], []]), ('A9', [['c']])]
```

2.2 Elimination des symboles non-accessibles

Un symbole $X \in V$ est *accessible* s'il existe une dérivation $S \mapsto^* \alpha X \beta$ ($\alpha, \beta \in (\Sigma \cup V)^\star$). L'ensemble $\mathbb{R}(G) \subseteq V$ des symboles accessibles de G est obtenu en construisant la suite croissante et bornée (car V est fini) :

$$\begin{array}{l} R_0(G) = \{S\} \\ R_{n+1}(G) = R_n(G) \cup \{X \mid Y \to u \in R \text{ et } Y \in R_n(G) \text{ et } X \in u\} \end{array} \quad \mathbb{R}(G) = \bigcup_{n \geq 0} R_n(G)$$

Il s'agit donc de la construction d'un point fixe qui permet de définir la grammaire hors-contexte $\mathcal{R}(G) = (\mathbb{R}(G), \Sigma, R', S)$ où R' est obtenue en supprimant de R les productions contenant des symboles non-accessibles. La grammaire $\mathcal{R}(G)$ engendre le même langage que G et on a donc $\mathcal{L}(G) = \mathcal{L}(\mathcal{R}(G))$.

Exemple Considérons la grammaire hors-contexte G_3 définie en (2). La suite $(R_n(G_3))_{n \in \mathbb{N}}$ est construite comme suit :

$$R_0(G_3) = \{S\} \quad R_1(G_3) = \{S, A_3\} \quad R_2(G_3) = \{S, A_3, A_6, A_9\} = R_3(G_3) = \mathbb{R}(G_3)$$

On a donc $V_3 \setminus \mathbb{R}(G_3) = \emptyset$ et donc $\mathcal{R}(G_3) = G_3$.

Implantation Pour construire la grammaire $\mathcal{R}(G)$, on calcule tout d'abord l'ensemble $\mathbb{R}(G)$. Ce calcul est effectué à partir de l'ensemble $R_0(G)$ en appliquant une fonction `next_reach` permettant de construire $R_{n+1}(G)$ à partir de $R_n(G)$ jusqu'à obtenir un point fixe, c-à-d un ensemble $R_{n+1}(G)$ égal à $R_n(G)$.

Calcul de $R_{n+1}(G)$ à partir de $R_n(G)$ La fonction `next_reach` effectue ce calcul. Outre l'ensemble `prev` à partir duquel est effectué ce calcul, cette fonction utilise la liste `nt` représentant V, la liste `r` des productions de G, et la fonction d'égalité `eqnt` sur V.

à compléter : calcul de $R_{n+1}(G)$ à partir de $R_n(G)$

```
def next_reach(nt,r,eqnt,prev):
    # nt : symboles non terminaux
    # r : liste de productions
    # eqnt : egalite sur les non terminaux
    # prev : liste de non terminaux de depart
```

exemples : $R_1(G_3)$, $R_2(G_3)$

```
>>> next_reach(g3_nt,g3_r,g3_eqnt,['S'])
['S', 'A3']
>>> next_reach(g3_nt,g3_r,g3_eqnt,['S','A3'])
['S', 'A6', 'A9', 'A3']
```

Calcul de $\mathbb{R}(G)$ Ce calcul correspond à la construction itérative d'un point fixe (et utilise donc la fonction `fixpoint_from` de l'annexe sur la représentation des ensembles) à partir de la fonction `next_reach`. Ici encore, cette construction utilise la fonction `make_eq_set` (définie dans l'annexe sur la représentation des ensembles) permettant de définir une fonction d'égalité sur des ensembles qui dépend de l'égalité sur les éléments de ces ensembles.

à compléter : calcul de $\mathbb{R}(G)$

```
def reach(nt,r,si,eqnt):
    # nt : symboles non terminaux
    # r : liste de productions
    # si : symbole de depart
    # eqnt : egalite sur les non terminaux
```

exemples : $\mathbb{R}(G_3)$

```
>>> reach(g3_nt,g3_r,g3_si,g3_eqnt)
['S', 'A6', 'A9', 'A3']
```

Calcul de $\mathcal{R}(G)$ Ce calcul s'effectue en utilisant la fonction `remove_nt` définie page 5 pour éliminer les symboles non-accessibles de G.

à compléter : calcul de $\mathcal{R}(G)$

```
def remove_non_reach(g):
    # g : ghc
```

2.3 Réduction d'une GHC

La construction d'une grammaire G' en forme réduite engendrant le même langage qu'une grammaire G s'obtient simplement en éliminant ses symboles non-productifs et ses symboles non-accessibles :

$$G' = \mathcal{R}(\mathcal{P}(G))$$

à compléter : réduction d'une GHC

```
def reduce_grammar(g):
    # g : ghc
```

exemple : réduction de G_1

```
>>> g3_g_bis = reduce_grammar(g1_g)
>>> g3_g_bis
(['S', 'A6', 'A9', 'A3'], ['a', 'b', 'c'],
 [('S', [['A3']]), ('A3', [['a', 'A3', 'A9'], ['A6']]), ('A6', [['b', 'A6'], []]),
 ('A9', [['c']])],
 'S', <function eq_atom ...>)
```