Ensimag 1ère année 26 avril 2022

Théorie des Langages 2

Durée : 3h. Documents : tous documents autorisés.

Les exercices sont indépendants. Barème indicatif

Exercice 1 (10 points)

▷ Question 1. (3 points)

Soit la grammaire G suivante :

(1)	SI	→	I	(2)	SI	→	I SI
(3)	I	→	affect	(4)	I	→	**if** exp **then** SI X **end ;**
(5)	X	→	ϵ	(6)	X	→	**else** SI
(7)	affect	→	**idf :=** exp **;**				

Le non-terminal SI est l'axiome et les éléments du vocabulaire terminal sont notés en gras. Le non-terminal exp n'est pas défini ici. Expliquer pourquoi la grammaire G n'est pas LL(1). Proposer une grammaire G' équivalente à G et LL(1). On prouvera le caractère LL(1) de la grammaire proposée.

▷ Question 2. (2 points)

Certaines normes de développement imposent des règles de programmation permettant de maîtriser la complexité du code. Une règle classique est de limiter la profondeur d'imbrication des conditionnelles. Par exemple la profondeur maximale d'imbrication des conditionnelles pour l'exemple suivant est 2 :

```
if e1 then
   if e2 then x:=3 ; end ;
else x:=1 ; end ;
if e3 then  x:=4 ; else x:=0 ; end ;
```

Ajouter un calcul d'attributs sur la grammaire G' de la question 1 permettant de calculer la profondeur maximale d'imbrication des conditionnelles dans une suite d'instructions (SI). On expliquera précisément la signification des attributs utilisés.

▷ Question 3. (3 points)

A partir de la grammaire G' écrire (en pseudo-Python) un analyseur LL(1) qui reconnait une instruction (non-terminal I) et calcule (attribut synthétisé) la profondeur maximale d'imbrication des conditionnelles. On supposera écrites les procédures d'analyse `parse_SI`, `parse_exp` et `parse_affect` ainsi que la fonction `get_current()` qui retourne le token courant dans l'ensemble des symboles du vocabulaire terminal augmenté du symbole, et la fonction `parse_token(t)` qui vérifie que le token courant est bien `t` et avance au token suivant... Le token de fin est noté `END`. On écrira toutes autres procédures d'analyse nécessaires.

▷ **Question 4. (2 points)**.

On désire maintenant étendre la définition de l'instruction conditionnelle pour ajouter une partie optionnelle `elif ... elif ...`. En partant de la question 1 proposer une grammaire LL(1) prenant en compte cette extension. On prouvera le caractère LL(1) de la nouvelle grammaire proposée (attention l'extension proposée peut modifier les réponses de la question 1). Il n'est pas demandé d'étendre le calcul d'attributs de la question 2.

Exercice 2 (4 points)

On s'intéresse dans cet exercice à étendre la méthode d'analyse LL(1) vue en cours au cas LL(2). Soit $G = (V_T, V_N, S, R)$ une grammaire hors-contexte. On définit Dir_2, l'ensemble des directeurs de taille 2, de la manière suivante :

$$(1) \quad Dir_2(A \rightarrow\ w) = \{\mathbf{xy} \in {V'_T}^2 \mid \exists u, v, t \in (V'_T \cup V_N)^* \ (S\$\$ \Rightarrow^* uAv \Rightarrow uwv \Rightarrow^* u\mathbf{xy}t)\}$$

Avec \$ un symbole de fin de mot tel que $\$ \notin V_T$ et $V'_T = V_T \cup \{\$\}$. On rappelle que V^2 désigne les mots de longueur 2 sur le vocabulaire V. La grammaire G est dite LL(2) [1] si et seulement si :

$$Dir_2(A \rightarrow\ w_1) \cap\ Dir_2(A \rightarrow\ w_2) = \emptyset$$

pour tout couple de règles $A \rightarrow\ w_1$ et $A \rightarrow\ w_2$ avec $w_1 \neq w_2$.

▷ **Question 1. (2 points)**

Donner les ensembles Dir_2 pour chacune des règles de la grammaire suivante, A étant l'axiome. En déduire que la grammaire est LL(2).

(1)	A	$\rightarrow$	$aBaa$	(2)	A	$\rightarrow$	$bCba$
(3)	B	$\rightarrow$	b	(4)	B	$\rightarrow$	ϵ
(5)	C	$\rightarrow$	b	(6)	C	$\rightarrow$	ϵ

▷ **Question 2. (2 points)**

On s'intéresse ici à étendre les définitions vues en cours permettant de calculer les ensembles Premier, Suivant et Directeur pour le cas LL(2) (ensembles notés ici $Prem_2$, $Suiv_2$ et Dir_2). En plus de la définition (1) on pose :

$$\begin{array}{lll} (2) \quad Prem_2(w) & = & \{\mathbf{xy} \in {V''_T}^2 \mid \exists u \in (V''_T \cup V_N)^* \ (w!! \Rightarrow^* \mathbf{xy}u)\} \\ (3) \quad Suiv_2(A) & = & \{\mathbf{xy} \in {V'_T}^2 \mid \exists u, v \in (V'_T \cup V_N)^* \ (S\$\$ \Rightarrow^* uA\mathbf{xy}v)\} \end{array}$$

avec w un mot sur $V_T \cup\ V_N$, A un élément de V_N, ! un nouveau symbole tel que $! \notin V_T$ et $V'_T = V_T \cup \{\$\}$ et $V''_T = V_T \cup \{!\}$.

Le symbole ! permet de détecter les premiers de taille 0 ou 1. Par exemple si $!! \in Prem_2(w)$ ceci signifie que w peut dériver vers ϵ et si $x! \in Prem_2(w)$ avec $x \in V_T$ ceci signifie que w peut dériver vers un mot de taille 1 (ici x). Enfin on définit la fonction $conc_2(e_1, e_2)$ avec e_1 et e_2 deux ensembles de mots appartenant à $(V_T \cup \{\$, !\})^2$ de la manière suivante :

$$\begin{array}{lll} conc_2(e_1, e_2) & = & \{xy \mid xy \in e_1 \ et \ x \in V_T \ et \ y \in V_T\} \\ & & \cup \{xy \mid xy \in e_2 \ et \ !! \in e_1\} \\ & & \cup \{xy \mid x! \in e_1 \ et \ x \in V_T \ et \ \exists z(yz \in e_2)\} \end{array}$$

Étendre les définitions vues en cours pour calculer $Prem_2(w)$, $Suiv_2(A)$ et $Dir_2(A \rightarrow\ w)$.

1. dans la réalité appelée Strong LL(2)

Exercice 3 (6 points)
(1) On a défini en cours les *ensembles Récursivement Énumérables* comme les sous-ensembles de $\mathbb{N}$ *acceptés* par une Machine de Turing : E est RE s'il existe une MT m_E qui accepte E. Pour simplifier : m_E s'arrête si et seulement si son entrée est dans E.
On peut aussi dire qu'il existe une fonction Python (dont le paramètre est dans $\mathbb{N}$)
`def Accepte_E(n): ...` qui s'arrête si et seulement si n est dans E

(2) Il existe une autre définition des ensembles RE : E est RE si et seulement si il est vide ou c'est *l'image d'une fonction f totale calculable* :
$E = \{n \mid \exists p \in \mathbb{N} : f(p) = n\}$
$f(0)$ est dans E, $f(1)$ est dans E, $f(2)$ est dans E... Noter qu'il est possible que $f(p) = f(q)$...
Autrement dit : il existe une MT m_f qui calcule f (et s'arrête toujours car f est totale), et dont le résultat après calcul sur un entier p quelconque est toujours un élément de E.
On peut aussi dire qu'il existe une fonction Python
`Calcule_f(p): ...` qui s'arrête toujours et retourne un élément de E.

Le but de l'exercice est de montrer l'équivalence des deux définitions.
Attention : On ne s'intéresse ici qu'aux E *infinis*

▷ **Question 1. (3 points)**
Montrer que **(2)** $\Rightarrow$ **(1)** : soit expliquer précisément (mais sans la donner) comment construire m_E à partir de m_f (avec autant de rubans qu'on veut), soit écrire `Accepte_E` en utilisant `Calcule_f`.

▷ **Question 2. (3 points)**
(Plus difficile) Montrer que **(1)** $\Rightarrow$ **(2)** : soit expliquer précisément (mais sans la donner) comment construire m_f à partir de m_E (avec autant de rubans qu'on veut, un nombre fini mais non borné...), soit écrire `Calcule_f` en utilisant `Accepte_E`. On pourra (devra) utiliser la fonction
`def Accepte_borne_E(n, t):...` qui s'arrête toujours, et retourne `True` si `Accepte_E(n)` termine avant `t` millisecondes, `False` sinon.