THL – Théorie des Langages

EPITA – Promo 2009 – **Documents autorisés**

Novembre 2006 (1h30)

Écrire court, juste, et bien. Une argumentation informelle mais convaincante, sera souvent suffisante.

1 Incontournables

Les questions suivantes sont fondamentales. Une pénalité sur la note finale sera appliquée pour les erreurs.

1. Un sous-langage d'un langage rationnel (i.e., un sous-ensemble) est rationnel. vrai/faux ?
2. Le langage engendré par $S \to 0X \quad S \to \varepsilon \quad X \to S1$ est rationnel. vrai/faux ?
3. Un automate non-déterministe est un automate qui n'est pas déterministe. vrai/faux ?

2 Culture Générale

1. Combien existe-t-il de sous-ensembles d'un ensemble de taille n ?
2. Combien existe-t-il de n-uplets (c'est-à-dire une suite de n éléments) d'un ensemble de taille m ?
3. Que vaut 2^{16} ?
4. Comment numéroter (i.e., étiqueter par un élément unique de $\mathbb{N}$) les réels ($\mathbb{R}$) ?
5. Qui a inventé les algorithmes LR(k) ?

3 Hiérarchie de Chomsky

Pour chacune des grammaires suivantes, préciser (i) son type dans la hiérarchie de Chomsky, (ii) si elle est ambiguë, (iii) le langage qu'elle engendre, (iv) le type du langage dans la hiérarchie, (v) s'il existe un automate fini déterministe qui reconnaisse ce langage.

Justifier vos réponses.

1. `P` $\to$ `P inst ';'`
   `P` $\to \varepsilon$
2. `P` $\to$ `P1`
   `P` $\to \varepsilon$
   `P1` $\to$ `P1 ';' inst`
   `P1` $\to$ `inst`
3. `P` $\to$ `P1`
   `P` $\to \varepsilon$
   `P1` $\to$ `P1 ';' P1`
   `P1` $\to$ `inst`

4 Parsage LL

On rappelle que 1/2/3 = 1/6 et non pas 1.5, et 2^{4^4} n'est pas égal à $(2^4)^4 = 65536$, mais vaut :

$2^{(4^4)} = 115792089237316195423570985008687907853269984665640564039457584007913129639936$

1. Écrire la grammaire naïve de l'arithmétique avec les terminaux « n » (les entiers), « / » la division, « ^ » la puissance, « (» et «) » les parenthèses.
2. En plusieurs étapes, en faire une grammaire adaptée au parsage LL(1).
3. Quel opérateur n'est pas lu correctement au terme de ces transformations ?
4. En s'autorisant l'utilisation de l'étoile de Kleene (*) dans les parties droites de règles, modifier la grammaire précédente pour « réparer » l'opérateur abîmé.
5. Considérant les déclarations suivantes :

```
/* The lookahead. */
token_t la;
/* Its value when it's an ``n''. */
int value;
/* Check that the lookahead is equal to ``t'', and then advance.
   Otherwise, report an error, and throw tokens until ``t'' (or
   end of file) is found. */
void eat (token_t t);
```

   écrire les routines d'analyse des non terminaux. On prendra garde à :
   – retourner la valeur de l'expression,
   – reporter les erreurs à l'utilisateur,
   – soumettre du code *lisible*,
   – ne pas se perdre dans les détails, rester abstrait
   (e.g., on peut utiliser `afficher (la)` sans en fournir d'implémentation).

5 Automate LR(1)

Soit le fichier Yacc/Bison suivant :

```
%%
l: l l | 'n';  /* These "l" are "L", not "1", of course. */
```

1. Montrer que cette grammaire est ambiguë en exhibant deux arbres de dérivation du plus petit mot ambigu.
2. Dessiner l'automate LR(1) en montrant bien les lookaheads.
3. Indiquer sur cette figure le conflit dû à l'ambiguïté.
4. Entre règles récursives à droite et à gauche, lesquelles préfèrent les parsers LR, et pourquoi.
5. Comment utiliser les directives de Yacc/Bison pour choisir cette « associativité » ?