THL – Théorie des Langages

EPITA – Promo 2008 – **Tous documents autorisés**

Novembre 2005 (1h30)

Une copie synthétique, bien orthographiée, avec un affichage clair des résultats, sera toujours mieux notée qu'une autre demandant une quelconque forme d'effort de la part du correcteur. Une argumentation informelle mais convaincante, sera souvent suffisante.

1 Incontournables

Il n'est pas admissible d'échouer sur une des questions suivantes : **chacune représente 25% de la note finale**. En d'autres termes, deux erreurs dans cette partie réduisent la note de la suite de moitié.

1. Le langage engendré par $S \to aX \quad S \to b \quad X \to Sc$ est rationnel. vrai/faux ?
2. Un sous-langage d'un langage rationnel (i.e., un sous-ensemble) est rationnel. vrai/faux ?
3. LR(k) est plus puissant que LL(k). vrai/faux ?
4. Si un analyseur GLR a des conflits « reduce/reduce » alors il est incorrect. vrai/faux ?

2 Langages rationnels

On pourra admettre le résultat principal de cette section dans la suite de l'épreuve.

1. Comment montre-t-on que l'intersection de deux langages rationnels est rationnelle ?
2. Comment montre-t-on que si un langage est rationnel, alors son « renversé » (tous les mots écrits à l'envers) l'est aussi ?
3. En déduire que $\{a^n b^m | m \leq n\}$ n'est pas rationnel.

3 Si alors sinon

3.1 Grammaires

Considérons l'extrait de grammaire suivant :

```
<stm>   : :=   if <exp> then <stm> else <stm>
           |   if <exp> then <stm>
           |   ...
```

1. Quelle est sa catégorie de Chomsky ? Justifier.
2. En considérant que `<exp>` et `...` sont des terminaux, quelle est la catégorie de Chomsky du langage engendré ? Justifier.
3. Cette grammaire est-elle ambiguë ? Justifier.
4. Produire deux arbres de dérivation différents d'une phrase la plus courte possible.
5. Pourquoi la syntaxe du shell comprend-elle un terminal `fi` ?

3.2 Désambiguïsation

À partir de ce point on ne retiendra que l'interprétation la plus répandue dans les langages de programmation : on attache un `else` au `if` le plus proche.

1. On rencontre fréquemment des macros comparables aux suivantes.

```
#define LOG(Msg)                                        \
  if (using_syslog)                                     \
    vsyslog (LOG_INFO, "%s\n", Msg);                    \
  else                                                  \
    fprintf (stderr, "%s: %s\n", program_name, Msg)

#define TRACE_INT(Int)                                  \
  do {                                                  \
    if (trace)                                          \
      fprintf (stderr, "Trace: %s = %d\n", #Int, Int);  \
  } while (0)
```

   Comment expliquez-vous cette différence de style ?

2. En introduisant <`stm-if-then`> et <`stm-if-then-else`>, proposer une grammaire non ambiguë équivalente à celle-ci :

   <stm-if> : := if <exp> then <stm-if> else <stm-if>
   | if <exp> then <stm-if>

3. Dans la réalité la grammaire originelle est plutôt la suivante. Quel problème allons-nous rencontrer pour adapter le travail précédent ?

   <stm> : := if <exp> then <stm> else <stm>
   | if <exp> then <stm>
   | while <exp> do <stm>
   | ...

3.3 Analyse LL

1. La grammaire suivante est-elle LL(k) ? Justifier.

   <stm> : := if <exp> then <stm> else <stm>
   | if <exp> then <stm>
   | while <exp> do <stm>
   | ...

2. Proposer en BNF une grammaire LL(1) engendrant le même langage.
3. Proposer en EBNF une grammaire LL(1) engendrant le même langage.
4. Déduire de la grammaire de l'item 3 une implémentation de `parse_stm`. Toutes les routines d'analyse retournent un arbre de type `ast_t *` et utilisent `token_if` ... pour les terminaux. Les routines telles que `ast_t *ast_if (ast_t *c, ast_t* s1, ast_t* s2)` construisent ces arbres.

3.4 Analyse LR

1. Proposer une implémentation LALR(1) de la grammaire suivante qui tire parti des fonctionnalités de Yacc.

   <stm> : := if <exp> then <stm> else <stm>
   | if <exp> then <stm>
   | while <exp> do <stm>
   | ...

2. Pourquoi préférer cette solution plutôt que celle retenue dans la section 3.2 ou dans la section 3.3 ?