# Rattrapage THL — Théorie des Langages
# Aucun document autorisé

EPITA – Promo 2007

Juillet 2005 (1h30)

Ce sujet est l'exacte copie de l'épreuve THL originelle que vous avez échouée.

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.

Dans cette épreuve, les non-terminaux sont écrits en majuscules, les terminaux en minuscules.

## 1 Hiérarchie de Chomsky

Pour chacun des langages suivants,

- préciser son type dans la hiérarchie de Chomsky (son rang et son nom, e.g., 3, langage rationnel). On demande **évidemment** le type le plus précis.
- proposer une grammaire *non ambiguë*, de même type de Chomsky, qui l'engendre. Les grammaires longues peuvent être esquissées et se terminer par « ... ».
- proposer un outil engendrant un analyseur adéquat pour saisir *la structure* du langage auquel il est fait référence. On ne demande pas l'outil minimum correspondant au rang du langage, mais celui qui permette de retrouver ce que l'on cherche dans chacun des cas.

1. Les mots du français.
2. Les nombres entiers en base 3 non signés.
3. Les expressions mathématiques composées de puissances (^), de parenthèses (« ( », « ) »), et de nombres entiers en base 3 non signés.

## 2 Parsage LL(1)

Soit le langage de la logique (dite propositionnelle) composée de deux symboles `t` (vrai) et `f` (faux), de l'opération unaire ¬ (non), des opérations binaires ∨ (ou) et ∧ (et), et des parenthèses. Ce langage inclut des mots tels que $f \wedge f$, $t \vee f$ et $\neg\neg(t \wedge t) \vee (f \wedge f)$.

1. Qu'avez-vous à dire de la grammaire suivante ?

   `S` → `S ∧ S | S ∨ S | ¬ S | ( S ) | t | f`

2. Qu'avez-vous à dire de la grammaire suivante ?

   `S` → `S ∨ T | T`
   `T` → `T ∧ F | F`
   `F` → `¬ F | ( S ) | t | f`

3. En déduire les priorités et associativités des opérateurs.
4. Donner deux raisons évidentes pour lesquelles cette grammaire n'est pas LL(1).

5. Calculer NULLABLE, FIRST et FOLLOW pour S, T et T dans la grammaire 2.
6. Qu'avez-vous à dire de la grammaire suivante ?

```
S  → T S'
S' → ∨ T S' | ε
T  → F T'
T' → ∧ F T' | ε
F  → ¬ F | ( S ) | t | f
```

7. Qu'avez-vous à dire de la grammaire suivante ?

```
S  → T (∨ T)*
T  → F (∧ F)*
F  → ¬ F | ( S ) | t | f
```

8. Écrire en pseudo code la routine de parsage LL(1) de T dans la grammaire 7.

# 3 Opérateurs n-aires

Soit le fragment suivant d'une grammaire d'un langage concernant l'opérateur `?:` :

```
exp: "exp" | exp "?" exp ":" exp
```

1. Montrer que cette grammaire est ambiguë en exhibant tous les arbres de dérivation engendrant le plus petit mot ambigu.
2. Cette ambiguïté est tout à fait comparable à une classe bien connue d'ambiguïté d'opérateurs, laquelle ?
3. On résoudra cette ambiguïté en ne conservant que l'une de ces possibilités. Discuter tous les choix possibles et leurs vertus pour un langage de programmation.
4. Écrire une grammaire non-ambiguë (sans annotation `%foo`) engendrant le même langage ne retenant qu'une interprétation.
5. Dessiner la partie de l'automate LR(1) conduisant à l'état (aux états) exhibant le conflit (les conflits) résultant de l'ambiguïté de la grammaire originale.
6. On rappelle qu'en Yacc, une règle hérite de la priorité/associativité de son terminal le plus à droite. Quelles sont les trois possibilités les plus simples d'annotation (`%foo`) de cette grammaire pour qu'il n'y ait plus de conflit ?
7. Comparer les avantages du parsage de ce langage par, d'une part une implémentation de la grammaire désambiguïsée de la question 4 en Yacc, et, d'autre part, celle exploitant les directives de Yacc comme dans la question 6.
8. L'une des possibilités de la question 6 implique que la grammaire annotée ainsi définie n'engendre pas le même langage. Pourquoi ?
9. Parmi les possibilités de la question 6 conduisant à une grammaire annotée en Yacc, quel est le choix exerçant la plus faible pression sur le parseur ?
10. Informellement, mais de façon convaincante, traiter l'extension suivante de cet opérateur :

```
exp: "exp"
   | exp "?" ":" exp
   | exp "?" exp ":" exp
   | exp "?" exp "," exp ":" exp
```

    On attend en particulier une description de l'ambiguïté de cette grammaire, et un mode d'implémentation Yacc.
11. Critiquer l'extension suivante :

```
exp: "exp"
   | exp "?" exp ":" exp
   | exp "?" exp ":" exp ":" exp
```