UNIVERSITÉ BORDEAUX I — U.F.R. Mathématiques et Informatique
Session de Septembre 1996 — J.Bétréma

Maîtrise Informatique
Compilation (tronc commun)
(Durée: 2 heures)

Notes de cours et de TD autorisées. Autres documents interdits.

Le sujet comporte 3 parties, sur 5 pages (pas de panique!). Des réponses non justifiées ne rapportent aucun point. Le barème est donné à titre indicatif (la somme des points dépasse volontairement 20).

1 Analyse lexicale

(5 points) Voici un programme écrit dans le langage reconnu par *lex* :

```
%{
 int banniere = 0;
%}

%%
.*"<BODY".*             {printf ("<BODY>"); banniere = 1;}
.*"</BANNER>".*         banniere = 0;
^"<IMG".*               ;
.*"cgi-bin".*           ;
.*"</HTMLPLUS>".*       printf("</BODY></HTMLPLUS>");
.*                      if (!banniere) printf ("%s", yytext);
\n                      if (!banniere) printf ("\n");
%%
```

Décrire exactement les transformations subies par un texte qu'on soumet à l'action de ce programme (une fois celui-ci compilé, bien entendu).

NB : paraphraser le programme ne rapportera aucun point; les transformations doivent être décrites sous une forme claire, compréhensible par quelqu'un ignorant tout de l'analyse lexicale. Une interprétation raisonnable de l'utilité de ce programme sera bienvenue, mais n'est pas exigée.

2 Grammaire

Voici une des règles de la grammaire du langage Ada :

corps ⟶

prototype **is**

begin

suite_d_instructions

[**exception**

traite_exception

{ traite_exception }]

end [identificateur]

Les mots écrits en gras sont des mots-clefs, les autres (sauf *identificateur*) sont des symboles non terminaux, qu'on remplacera dans la suite par leurs initiales (resp. C, P, S et T). La particularité de cette grammaire est d'utiliser les crochets (ie [et]) pour indiquer une partie optionnelle, et les accolades pour indiquer une partie qui peut être répétée un nombre quelconque (y compris nul) de fois.

1. (4 points) Remplacer cette règle par un ensemble de règles équivalent, aussi simple que possible, écrites dans le formalisme normal d'une grammaire algébrique (c'est à dire sans crochet ni accolade).

2. (2 points) Pensez-vous qu'il existe des langages non algébriques, et reconnus par une grammaire du type de celle du langage Ada (ie. comportant des parties optionnelles ou répétitives) ?

   NB : ici comme ailleurs, une réponse non justifiée ne rapporte aucun point.

3 Analyse syntaxique

Voici un fichier produit par *yacc* (en fait *bison*), en utilisant l'option de compilation -v :

```
Grammar
rule 1    expr -> expr '|' terme
rule 2    expr -> terme
rule 3    terme -> terme facteur
rule 4    terme -> facteur
rule 5    facteur -> facteur '*'
rule 6    facteur -> facteur '+'
rule 7    facteur -> facteur '?'
rule 8    facteur -> '(' expr ')'
rule 9    facteur -> ID

state 0

    ID   shift, and go to state 1
    '('  shift, and go to state 2
```

```
    expr go to state 3
    terme go to state 4
    facteur go to state 5

state 1

    facteur  ->  ID .   (rule 9)

    $default reduce using rule 9 (facteur)

state 2

    facteur  ->  '(' . expr ')'   (rule 8)

    ID   shift, and go to state 1
    '('  shift, and go to state 2

    expr go to state 6
    terme go to state 4
    facteur go to state 5

state 3

    expr  ->  expr . '|' terme   (rule 1)

    $    go to state 14
    '|'  shift, and go to state 7

state 4

    expr  ->  terme .   (rule 2)
    terme  ->  terme . facteur   (rule 3)

    ID   shift, and go to state 1
    '('  shift, and go to state 2

    $default reduce using rule 2 (expr)

    facteur go to state 8

state 5

    terme  ->  facteur .   (rule 4)
    facteur  ->  facteur . '*'   (rule 5)
    facteur  ->  facteur . '+'   (rule 6)
    facteur  ->  facteur . '?'   (rule 7)

    '*'  shift, and go to state 9
```

```
    '+'  shift, and go to state 10
    '?'  shift, and go to state 11

    $default reduce using rule 4 (terme)

state 6

    expr  ->  expr . '|' terme   (rule 1)
    facteur  ->  '(' expr . ')'   (rule 8)

    '|'  shift, and go to state 7
    ')'  shift, and go to state 12

state 7

    expr  ->  expr '|' . terme   (rule 1)

    ID   shift, and go to state 1
    '('  shift, and go to state 2

    terme go to state 13
    facteur go to state 5

state 8

    terme  ->  terme facteur .   (rule 3)
    facteur  ->  facteur . '*'   (rule 5)
    facteur  ->  facteur . '+'   (rule 6)
    facteur  ->  facteur . '?'   (rule 7)

    '*'  shift, and go to state 9
    '+'  shift, and go to state 10
    '?'  shift, and go to state 11

    $default reduce using rule 3 (terme)

state 9

    facteur  ->  facteur '*' .   (rule 5)

    $default reduce using rule 5 (facteur)

state 10

    facteur  ->  facteur '+' .   (rule 6)

    $default reduce using rule 6 (facteur)
```

```
state 11

    facteur  ->  facteur '?' .   (rule 7)

    $default reduce using rule 7 (facteur)

state 12

    facteur  ->  '(' expr ')' .   (rule 8)

    $default reduce using rule 8 (facteur)

state 13

    expr  ->  expr '|' terme .   (rule 1)
    terme  ->  terme . facteur   (rule 3)

    ID   shift, and go to state 1
    '('  shift, and go to state 2

    $default reduce using rule 1 (expr)

    facteur go to state 8

state 14

    $    go to state 15

state 15

    $default accept
```

1. (2 points) Quel est le langage reconnu par cette grammaire ?
2. (3 points) Donner un exemple simple, mais non trivial, de mot reconnu par cette grammaire, avec son arbre de dérivation.
3. (4 points) Expliquer les états 0, 4 et 8, en refaisant les calculs à la main, sur votre copie.
4. (6 points) Quels sont les états avec conflits ? Expliquer la façon dont ils sont résolus, et prouver que la grammaire est SLR(1).