Maîtrise d'Informatique

COMPILATION

Durée 3 heures, documents manuscrits autorisés

Session de Sepetmbre 1999

Problème 1 : Traductions en Lex *(8 pts)*

Question 1.
On utilise le programme produit par le texte Lex :

```
%option noyywrap

%%
a       printf("00");
ab      printf("01");
bba     printf("100");
bbc     printf("101");
cc      printf("110");

%%
main() {yylex(); }
```

Chacun des cinq mots suivants sont donnés en entrée au programme. Indiquer pour chacun d'eux quelle est la sortie obtenue :

$$accbbcbba, abbba, abacc, ba, abbc.$$

Question 2. On modifie le texte Lex en :

```
%option noyywrap

%%
a/a       printf("00");
a/c       printf("00");
a/bbc     printf("00");
a/bba     printf("00");
ab/a      printf("01");
ab/bb     printf("01");
ab/c      printf("01");
bba       printf("100");
bbc       printf("101");
cc        printf("110");

%%

main() {yylex();}
```

Quelle est cette fois la sortie pour chacun des 5 mots précédents ?

Question 3. Décrire de façon précise la transformation réalisée par le programme Lex de la question 2.

Question 4. On considère la transformation ϕ réalisée sur les mots de l'alphabet $A = \{a, b, c\}$ vers des mots sur l'alphabet $\{0, 1\}$ donnée par :

$$\phi(a) = 00, \quad \phi(ab) = 01, \quad \phi(bba) = 100$$

$$\phi(bbc) = 101, \quad \phi(cc) = 110, \quad \phi(ccb) = 111$$

On admettra que tout mot de A^* ou bien n'a pas d'image par la transformation ϕ ou bien en a une et une seule. Ainsi par exemple $abbb$ n'admet pas d'image par contre l'image de $abbba$ est 01100.

Donner un automate avec sortie qui effectue cette transformation.

Question 5. Ecrire un texte Lex qui réalise la transformation de la Question 4.

Problème 2: Analyse LR *(8pts)*

On donne la grammaire $\mathcal{G} = (A, \Sigma = \{S, T, U\}, \mathcal{R}, S)$ où les notations sont celles du cours, S est l'axiome, l'alphabet terminal est $\{a, +, *, (,)\}$.

Les règles $\mathcal{R}$ sont :

$$S \to T, \quad T \to T + U, \quad T \to U$$
$$U \to (T), \quad U \to a, \quad U \to a * (T)$$

Question 1. Donner un mot engendré par la grammaire E, T, F du cours qui n'est pas engendré par la grammaire $\mathcal{G}$ ci-dessus.

Question 2. Montrer (par récurrence sur la longueur) que tout mot engendré par la grammaire $\mathcal{G}$ (que l'axiome soit S ou T ou U) est aussi engendré par la grammaire E, T, F du cours.

Question 3. Calculer les 0-items pour cette grammaire.

Question 4. Montrer que cette grammaire n'est pas LR(0) en utilisant ces items.

Question 5. Cette grammaire est-elle $LR(1)$, $SLR(1)$?

Problème 3: Grammaire de langage de programmation*(4 pts)*

On utilise la partie de la grammaire du langage `Java` donnée en Annexe, dans laquelle le symbole : signifie $\to$, et où les non-terminaux sont représentés par des mots.

Question 1.

Comment se retrouvent dans cette grammaire de Java les 6 règles de la grammaire E, T, F du cours?

Question 2. Donner l'arbre de dérivation de la partie de programme suivante :

```
x = a + b;
```

On utilisera comme axiome le non-terminal `StatementExpression` et on considèrera `Name` comme un symbole pouvant engendrer n'importe quelle suite de lettres de l'alphabet ordinaire.

Question 3. Donner les règles de priorité entre opérateurs de Java que l'on peut déduire de la grammaire (par exemple `*` est prioritaire par rapport à `+`).

Annexe: Partie de Grammaire Java

```
StatementExpression :
        Assignment
        PreIncrementExpression
        PreDecrementExpression
        PostIncrementExpression
        PostDecrementExpression

PostfixExpression :
        Primary
        Name
        PostIncrementExpression
        PostDecrementExpression

PostIncrementExpression :
        PostfixExpression  ++

PostDecrementExpression :
        PostfixExpression  --

UnaryExpression :
        PreIncrementExpression
        PreDecrementExpression
      +   UnaryExpression
      -   UnaryExpression
        UnaryExpressionNotPlusMinus


UnaryExpressionNotPlusMinus :
        PostfixExpression
      ~   UnaryExpression
      !   UnaryExpression
        CastExpression

MultExpression :
        UnaryExpression
        MultExpression  *   UnaryExpression
        MultExpression  /   UnaryExpression
        MultExpression  %   UnaryExpression

AdditiveExpression :
        MultExpression
        AddExpression  +   MultExpression
        AddExpression  -   MultExpression

ShiftExpression :
        AddExpression
        ShiftExpression  <<   AddExpression
        ShiftExpression  >>   AddExpression
        ShiftExpression  >>>   AddExpression
```

```
RelExpression :
        ShiftExpression
        RelExpression  <   ShiftExpression
        RelExpression  >   ShiftExpression
        RelExpression  <=   ShiftExpression
        RelExpression  >=   ShiftExpression
        RelExpression  instanceof   RefType

EqualityExpression :
        RelExpression
        EqualityExpression  ==   RelExpression
        EqualityExpression  !=   RelExpression

AndExpression :
        EqualityExpression
        AndExpression  &   EqualityExpression

ExcOrExpression :
        AndExpression
        ExcOrExpression  ^   AndExpression

IncOrExpression :
        ExcOrExpression
        IncOrExpression  |   ExcOrExpression

CondAndExpression :
        IncOrExpression
        CondAndExpression  &&   IncOrExpression

CondOrExpression :
        CondAndExpression
        CondOrExpression  ||   CondAndExpression

CondExpression :
        CondOrExpression
        CondOrExpression  ?   Expression  :   CondExpression

AssignmentExpression :
        CondExpression
        Assignment

Assignment :
        LeftHandSide    AssignmentOp    AssignmentExpression

LeftHandSide :
        Name
        FieldAccess
        ArrayAccess

AssignmentOp :  one of
       = *= /= %= += -= <<= >>= >>>= &= ^= |=

Expression :
        AssignmentExpression

ConstantExpression :
        Expression
```