Licence d'informatique – 2003/2004

**ASC**

mardi 7 septembre 2004

Durée 3h. Documents autorisés

---

# Examen 2ème session

## 1 Grammaires algébriques

On considère la grammaire $G_1 = (X_1, V_1, S, P_1)$ où $X_1 = \{ a , b \}$, $V_1 = \{ S , S_1 \}$, et $P_1 = \{$

$$\begin{array}{lcl} S & \longrightarrow & S_1 \ \mathrm{S} \ S_1 \mid b \\ S_1 & \longrightarrow & a \ S_1 \ a \mid \varepsilon \end{array}$$

$\}$.

Question 1.1 : La grammaire $G_1$ est-elle $LL(1)$, $SLR(1)$, ambiguë ? Justifier.

Question 1.2 : Trouver une grammaire régulière engendrant le même langage que la grammaire $G_1$.

## 2 Composants logiciels

Dans le cadre du développement d'applications réparties à base de composants logiciels réutilisables, on souhaite décrire l'architecture de l'application à l'aide d'un langage permettant de construire de nouveaux composants par assemblage d'autres composants[1].

Un composant logiciel est caractérisé par un nombre de connexions d'entrée et un nombre de connexions de sortie. Une représentation graphique d'un composant élémentaire est donnée figure 1. Textuellement, on représentera un composant élémentaire par un couple <*nombre de connexions d'entrée*; *nombre de connexions de sortie*>. Par exemple, le composant de la figure 1 s'écrit < 3 ; 2 >.

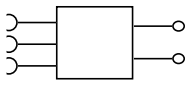


FIG. 1 – Un composant logiciel à 3 entrées et 2 sorties

Pour assembler les composants, on dispose de 2 opérations élémentaires :

**La mise en série :** La mise en série d'un composant $c_1$ et d'un composant $c_2$ correspont à la connexion des sorties de $c_1$ aux entrées de $c_2$. Cette mise en série n'est possible que si le nombre de sorties de $c_1$ est égal au nombre de sorties de $c_2$. On obtient alors un composant dont le nombre d'entrées est le nombre d'entrées de $c_1$ et le nombre de sorties est le nombre de sorties de $c_2$. Par exemple < 3 ; 2 > < 2 ; 2 > est une représentation textuelle de la mise en série de la figure 2. En revanche, < 3 ; 2 > < 3 ; 2 > n'est pas une représentation de mise en série valide.

[1] Une application peut alors être vue comme un composant obtenu par assemblage.

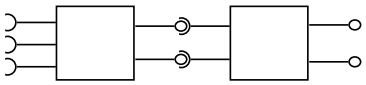


FIG. 2 – Une mise en série de 2 composants

**La mise en parallèle :** La mise en parallèle d'un composant $c_1$ et d'un composant $c_2$ correspont à la construction d'un nouveau composant comportant un nombre d'entrées égal à la somme du nombre d'entrées de $c_1$ et de $c_2$ et un nombre de sorties égal à la somme du nombre de sorties de $c_1$ et de $c_2$. On représentera textuellement la mise en parallèle de deux composants sous la forme d'un opérateur binaire associatif noté `&`. Par exemple `< 3 ; 2 > & < 2 ; 2 >` est une représentation textuelle de la mise en parallèle de la figure 3.

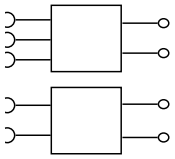


FIG. 3 – La mise en parallèle de 2 composants

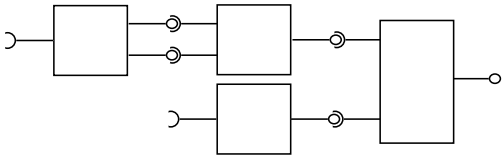
Pour décrire la syntaxe du langage d'assemblage de composants, on se donne la grammaire $G_2 = (X_2, V_2, C, P_2)$ où $X_2 = \{ <, >, ;, \&, (, ), \texttt{int} \}$, $V_2 = \{ C, C', P, P', E \}$, et $P_2 = \{$

$$
\begin{array}{lcl}
C & \longrightarrow & P\ C' \\
C' & \longrightarrow & \texttt{\&}\ P\ C' \mid \varepsilon \\
P & \longrightarrow & E\ P' \\
P' & \longrightarrow & E\ P' \mid \varepsilon \\
E & \longrightarrow & \texttt{<}\ \texttt{int}\ \texttt{;}\ \texttt{int}\ \texttt{>} \mid \texttt{(}\ C\ \texttt{)}
\end{array}
$$

$\}$.

Question 2.1 : D'après la grammaire $G_2$, quelle opération est la plus prioritaire : la mise en série ou la mise en parallèle ?

Question 2.2 : Donner une représentation textuelle pour l'assemblage de composants suivant :



Question 2.3 : Trouver les ensembles *premiers* et *suivants* associés aux variables de la grammaire $G_2$.

Question 2.4 : Construire la table d'analyse $LL(1)$ de la grammaire $G_2$.

Une représentation textuelle d'un assemblage peut être syntaxiquement correcte (c'est à dire engendrable par la grammaire $G_2$), mais sémantiquement incorrecte par rapport aux conditions à respecter pour la mise en série. C'est le cas par exemple pour l'expression suivante :

`< 2 ; 3 > ( < 1 ; 2 > < 2 ; 1 > & < 1 ; 3 > )`

Question 2.5 : Décorer la grammaire $G_2$ d'actions sémantiques vérifiant la validité d'une représentation textuelle d'assemblage de composants, avec declenchement d'une exception en cas d'expression invalide. Pour chaque variable, indiquer les attributs qui leur sont associés en donnant leur type et en précisant s'il s'agit d'attributs synthétisés ou hérités.

Question 2.6 : Écrire en `java` les méthodes associées aux variables `P'` et `E` dans l'analyseur récursif descendant correspondant à la grammaire attribuée précédente.

Pour cette dernière question, on déclenchera une exception de nom `SyntaxException` en cas d'erreur syntaxique et une exception de nom `SemanticException` en cas d'erreur sémantique. On pourra également supposer que l'on dispose de l'analyseur lexical suivant :

```
public class Token {

  public static final Token INT = ... ; // terminal INT
  public static final Token OPENING = ... ; // terminal '('
  public static final Token CLOSING = ... ; // terminal ')'
  public static final Token SEMI_COLON = ... ; // terminal ';'
  public static final Token AMPERSAND = ... ; // terminal '&'
  public static final Token LESS_THAN = ... ; // terminal '<'
  public static final Token MORE_THAN = ... ; // terminal '>'
  public static final Token END = ... ; // pseudo terminal '$'


  // passe au token suivant
  public static void goToNextToken() {...

  // donne la valeur du token courant
  public static Token currentToken() {...

  // retourne la valeur entière du token courant. N'a de sens
  // que quand le token courant vaut INT
  public int intValue() {...
}
```

# 3 Si...alors...sinon

On considère la grammaire suivante décrivant la syntaxe d'un *si ...alors ...sinon* :

| | | |
|---|---|---|
| *instruction* | $\longrightarrow$ | *instruction_si* |
| *instruction_si* | $\longrightarrow$ | `si_b_alors` *instruction_si* $\mid$ *si_complet* |
| *si_complet* | $\longrightarrow$ | `si_b_alors` *si_complet* `sinon` *instruction_si* $\mid$ `a` |

Pour alléger l'écriture, on considère $G_3 = (X_3, V_3, P, P_3)$ où $X_3 = \{$ `i` , `e` , `a` $\}$, $V_3 = \{ P , S , C \}$, et $P_3 = \{$

| | | |
|---|---|---|
| $P$ | $\longrightarrow$ | $S$ |
| $S$ | $\longrightarrow$ | `i` $S \mid C$ |
| $C$ | $\longrightarrow$ | `i` $C$ `e` $S \mid$ `a` |

$\}$.

Question 3.1 : Construire l'automate $LR(0)$ de la grammaire $G_3$.

Question 3.2 : Identifier dans l'automate $LR(0)$ un conflit *réduction-décalage* montrant que la grammaire n'est pas $SLR(1)$.

Question 3.3 : Dans ce conflit, faut-il choisir le décalage ou la réduction pour que l'expression soit analysée selon la règle : "*Tout* `sinon` *se rattache au* `si` *sans* `sinon` *le plus proche*" ?

Question 3.4 : Montrer que la grammaire $G_3$ est ambiguë.