Contrôle continu - Compilation (L3 Info)
Durée : 2h - documents interdits

16 et 17 février 2015

Conventions

— Axiome : symbole non terminal à gauche de la première production de la grammaire
— Symboles non terminaux : lettres *MAJUSCULES ITALIQUES*
— Symboles terminaux : lettres `minuscules true-type` ou caractères spéciaux simples

Question 1 (4 pt) - Questions de cours Pour chacune des affirmations suivantes dites si elle est vraie ou fausse en justifiant brièvement et de façon convaincante votre réponse (seules les réponses accompagnées de leurs justification seront prises en compte)

1. Toute grammaire hors-contexte récursive à gauche est ambiguë.
2. Tout langage hors-contexte peut être reconnu par un automate à pile.
3. Tout langage hors-contexte peut être reconnu par un automate à pile déterministe.
4. Les grammaires LL ne sont pas ambiguës.

Question 2 (5 pt) - Écriture de grammaire
Le langage JSON (JavaScript Object Notation) est un sous-ensemble de JavaScript, utilisé essentiellement pour l'échange et la transmission de données dans les applications web. Facile à lire et à traiter, il constitue une alternative légère à XML. La plupart des langages de programmation possèdent des bibliothèques pour lire et écrire ce format. Voici un exemple de *STRUCTURE* de données au format JSON, décrivant une personne et ses enfants :

```
{ "prenoms": ["Jason", "Philippe", "Alexandre"],
  "nom": "SILVA", "age": 47, "salarié": true,
  "enfants": [
    {"prenoms": ["Aurélie", "Jeanne", "Antoinette"], "nom": "SILVA", "age":12},
    {"prenoms": ["Kévin","Nicolas"], "nom": "SILVA", "age":7}
  ]
}
```

Une *STRUCTURE* JSON peut prendre une des formes ci-dessous :
— *LISTE* : Les listes sont délimitées par des crochets : `[` et `]`. Elles contiennent une liste de *STRUCTURE*s séparées les unes des autres par des virgules `,`. Une liste peut être vide, elle est alors représentée par une paire de crochets `[]`. Cependant, une *STRUCTURE* ne peut pas être vide. Ainsi, des constructions comme `[,"a"]`, `[1,]` ou `["a",,1]` sont incorrectes.
— *OBJET* : Les objets sont délimités par des accolades : `{` et `}`. Ils contiennent une liste de paires *clé*`:`*STRUCTURE* séparées les unes des autres par des virgules `,`. La clé est obligatoirement une `chaîne` de caractères (définie ci-dessous). La clé est suivie de deux points `:`, et ceux-ci sont suivis d'une *STRUCTURE*. Un *OBJET* peut être vide, représenté par une paire d'accolades `{}`. Cependant, les paires *clé* :*STRUCTURE* ne peuvent pas être vides. Ainsi, des constructions comme `{,"a":"b"}` ou `["a":1,]` ou `["c1":"v1",,"c2":2]` sont incorrectes.
— `chaîne` de caractères : une suite de caractères délimitée par des guillemets `"`
— `nombre` : n'importe quel nombre entier (`-5`), fractionnaire (`3.02`) ou en format exposant (`4.1e-2`)
— *CONSTANTE* : `true` représente le booléen *vrai*, `false` représente le booléen *faux* et `null` est la valeur vide

1. Écrivez une grammaire hors contexte non ambiguë pour décrire les *STRUCTURE*s de données en format JSON. Pour cela, vous pouvez supposer qu'un analyseur lexical renvoie des unités lexicales pour les nombres, chaînes de caractères et constantes. Vous pouvez utiliser dans votre grammaire les terminaux `nombre`, `chaîne`, `true`, `false` et `null`.
2. Dessinez les arbres de dérivation des *STRUCTURE*s `[true]` et `{chaîne:[nombre,nombre]}` (qui correspond par exemple à `{"centre":[10,5]}`).

Question 3 (4 pt) - Récursivité gauche Soit la grammaire G_1 :

$$G_1 : \begin{array}{lcl} S & \rightarrow & A \mid B \mid \varepsilon \\ A & \rightarrow & B \ \texttt{a} \mid \texttt{a} \\ B & \rightarrow & A \ \texttt{b} \mid \texttt{b} \end{array}$$

1. Décrivez le langage $L(G_1)$ engendré par la grammaire G_1. Quels types de mots sont acceptés/rejetés par G_1 ?
2. La grammaire G_1 est récursive à gauche. Écrivez une grammaire G_1' sans récursivité gauche directe ni indirecte, équivalente à G_1, c'est-à-dire telle que $L(G_1) = L(G_1')$.
3. Si l'on rajoute la règle $A \rightarrow S$ à G_1, la grammaire devient ambiguë. Démontrez-le.

Question 4 (7 pt) - Analyse $LL(1)$ La grammaire G_2 ci-dessous décrit un nombre *N*.

$$G_2 : \begin{array}{lllll} (1) & N & \rightarrow & OM \ \ LC \ \ OF \ \ OE \\ (2) & OM & \rightarrow & \texttt{-} \\ (3) & OM & \rightarrow & \varepsilon \\ (4) & LC & \rightarrow & \texttt{c} \ \ LCB \\ (5) & LCB & \rightarrow & \texttt{c} \ \ LCB \\ (6) & LC & \rightarrow & \varepsilon \\ (7) & OF & \rightarrow & \texttt{,} \ \ LC \\ (8) & OF & \rightarrow & \varepsilon \\ (9) & OE & \rightarrow & \texttt{E} \ \ OM \ \ LC \\ (10) & OE & \rightarrow & \varepsilon \end{array}$$

1. Calculez les ensembles PREMIER pour les symboles non terminaux de la grammaire G_2.
2. Calculez les ensembles SUIVANT pour les symboles non terminaux de la grammaire G_2.
3. Construisez la table d'analyse $LL(1)$ de la grammaire G_2.
4. Simulez l'analyse LL du mot `c,cE-c` (qui correspond par exemple à `3,4E-5`). Les configurations sont représentées par le triplet (w, α, y), où w est la partie de la bande de lecture qui commence au caractère se trouvant sous la tête de lecture (ce qui reste à analyser), α est la pile et y est la séquence de symboles de sortie (numéros des productions appliquées).