Thème Théorie des langages
Modélisation d'un langage

Lecture d'expressions régulières

Exercice 1 Soit l'alphabet ASCII standard, quels sont les langages représentés par les expressions régulières suivantes (pour cela, distinguer les caractères faisant parti des mots du langage des caractères correspondant à des opérateurs dans les expressions régulières) ?

1. `[a-zA-Z_][a-zA-Z0-9_]*`
2. `(0|(-?[1-9][0-9]*(/[1-9][0-9]*)?))`

Écriture d'expressions régulières

Exercice 2 Donner une expression régulière représentant chaque langage suivant :

1. Le langage des identificateurs en forme *camel case* : il s'agit d'identificateurs composés d'une suite de mots collés l'un à l'autre. Chaque mot commence par une majuscule et est ensuite composé de minuscules. Par exemple : `SciencesDuNumerique`
2. Le langage des chaînes de caractères : celles-ci commencent et se terminent par un guillemet `"` puis contiennent une suite de caractères autres que des guillemets. Elles peuvent contenir le caractère guillemet s'il est préfixé par le caractère *back-slash* \ (contre-oblique en Français). Elles peuvent contenir le caractère contre-oblique s'il est préfixé par un autre caractère contre-oblique. Par exemple : `"Bonjour,\\n\"Marc\""`
3. Le langage des entiers naturels représentés en Décimal Codé Binaire (DCB ou Binary Coded Decimal – BCD) consiste à utiliser 4 bits pour coder chaque chiffre du nombre (il y a alors une perte de 6 valeurs parmi les 16 valeurs possibles pour 4 bits, les valeurs supprimées sont celles correspondant aux chiffres A, B, C, D, E et F en hexadécimal). Un nombre codé en DCB est donc composé d'une suite non vide de mots de 4 bits. Un bit est représenté par les caractères `0` et `1`.
4. Le langage précédent peut faire apparaitre des chiffres `0` inutiles à gauche du codage du nombre. Proposer une modification de l'expression régulière qui ne permette pas leur apparition. Le nombre de bits utilisés pourra alors ne plus être un multiple de 4.
5. Le langage des nombres entiers complexes :

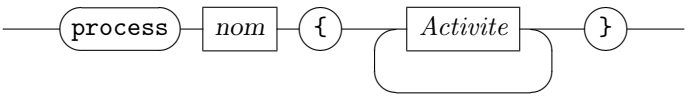
   - Un nombre entier complexe est composé d'une partie réelle et d'une partie imaginaire dont l'une des deux est optionnelle (il existe des nombres réels purs et des nombres imaginaires purs).
   - La partie réelle est un nombre entier relatif.
   - La partie imaginaire est un nombre entier naturel suivi du caractère `i`.
   - Si les deux parties du nombre complexe sont présentes, elles sont séparées d'un signe `+` si la partie imaginaire est positive et d'un signe `-` sinon.
   - Un nombre entier naturel est une suite de chiffres non vide.
   - Un nombre entier relatif est un nombre entier naturel éventuellement précédée du signe `-`

   Par exemple : `-1+2i`, `-3i`, `-2`

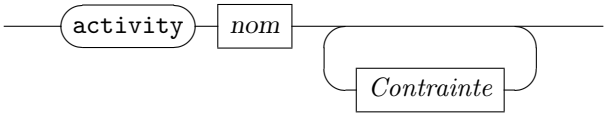
Lecture de grammaires

Exercice 3 Soit le langage de modélisation de processus définie par le diagramme de Conway et la grammaire EBNF équivalente suivant :

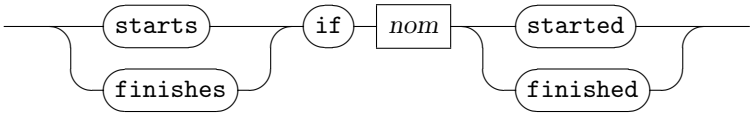
Processus



Activite



Contrainte




```
<Processus> ::= process nom \{ <Activite> { <Activite> } \}
<Activite> ::= activity nom { <Contrainte> }
<Contrainte> ::= ( starts | finishes ) if nom ( started | finished )
```

Définir un processus comportant 3 activités A, B et C; telles que B peut démarrer si A a démarré et C peut s'arrêter si A s'est arrêtée.

Écriture de grammaires

Exercice 4

1. Décrire le langage des nombres entiers complexes de l'exercice 2 sous la forme :
   (a) Diagramme de Conway
   (b) Règles EBNF
   (c) Règles de production
2. Donner un arbre de dérivation pour le nombre `-10+21i`

Exercice 5 L'objectif de cet exercice est de décrire la syntaxe du format d'échange d'objets JSON (*JavaScript Object Notation*). Voici un exemple de document JSON :

```
{
    "prénom" : "Marc",
    "nom" : "Pantel",
    "naissance" : {
       "jour" : 4,
       "mois" : "mai",
       "année" : 1966
    },
    "matière" : [ "OMI", "TOB", "GLS", "STL", "SCP", "CL" ]
}
```

Ce format respecte les contraintes suivantes :

- les terminaux, qui ne seront pas détaillés dans cet exercice, sont les chaînes de caractères `string`, les nombres `number`, les valeurs booléennes `true` et `false` et l'objet non initialisé `null` ;
- un document est un objet ;
- un objet est composé d'une liste entre accolades (`{` et `}`) d'attributs séparés par des virgules. Cette liste peut être vide ;
- un attribut est composée d'un nom représenté par une chaîne de caractères et d'une valeur séparé par `:` ;
- une valeur est soit une chaîne de caractères, soit un nombre, soit un objet, soit un tableau, soit une valeur booléenne, soit l'objet non initialisé ;
- un tableau est composé d'une liste entre crochets (`[` et `]`) de valeurs séparées par des virgules. Cette liste peut être vide.

1. Donner une grammaire pour le format JSON sous la forme de :
   (a) Diagramme de CONWAY;
   (b) Règles de production.
2. Donner un arbre de dérivation pour l'exemple suivant en utilisant les règles de production que vous avez proposées.

```
{
    "nom" : "Pantel",
    "naissance" : {
       "année" : 1966
    },
    "matière" : [ "MP" ]
}
```

Dérivations et Arbres syntaxiques

Construction de dérivations

Exercice 6 Soit une grammaire possible des expressions :

1. $E \rightarrow E$ `+` T
2. $E \rightarrow T$
3. $T \rightarrow T$ `*` F
4. $T \rightarrow F$
5. $F \rightarrow$ `entier`
6. $F \rightarrow$ `ident`
7. $F \rightarrow$ `(` E `)`
8. $F \rightarrow$ `-` F

Soit l'expression `-1+4*x+y` :

1. Construire la dérivation la plus à gauche et l'arbre de dérivation associé.
2. Construire la dérivation la plus à droite et l'arbre de dérivation associé.

Grammaire ambiguë : l'instruction conditionnelle

Exercice 7

1. Soit la grammaire (des instructions d'un langage) définie partiellement par les règles de production suivantes :

   ```
   Instruction -> IF Expression THEN Instruction ELSE Instruction
   Instruction -> IF Expression THEN Instruction
   Inst -> ...
   ```

   Soit l'expression :

   ```
   IF expr1 THEN IF expr2 THEN inst1 ELSE inst2
   ```

   (a) Construire les différentes dérivations possibles avec la stratégie la plus à gauche et les arbres de dérivation associés

   (b) Que peut-on conclure ?

   (c) Proposer des modifications du langage et/ou de la grammaire en conséquence.

Rappels de cours distribués lors de l'examen écrit.

1 Expressions régulières : Equivalence sémantique

L'opérateur de concaténation/juxtaposition . est implicite pour ne pas surcharger l'écriture des expressions : $e_1.e_2$ est notée $e_1\,e_2$.

$$\begin{array}{ll}
\emptyset\, e = e\, \emptyset = \emptyset & \Lambda\, e = e\, \Lambda = e \\
e \mid \emptyset = \emptyset \mid e = e & e \mid e = e \\
e_1\,(e_2\,e_3) = (e_1\,e_2)\,e_3 & e_1 \mid (e_2 \mid e_3) = (e_1 \mid e_2) \mid e_3 \\
e_1\,(e_2 \mid e_3) = (e_1\,e_2) \mid (e_1\,e_3) & (e_1 \mid e_2)\,e_3 = (e_1\,e_2) \mid (e_1\,e_3) \\
e_1 \mid e_2 = e_2 \mid e_1 & \emptyset^\star = \Lambda^\star = \Lambda \\
e^\star = \Lambda \mid e^+ & e^+ = e\,e^\star = e^\star\,e \\
e^\star\,e^\star = e^\star & e^{\star\star} = e^\star \\
e = e^\star \Leftrightarrow e = e\,e \wedge e \neq \emptyset & e\,e^\star = e^\star \Leftrightarrow \Lambda \in L(e) \\
(e_1^\star\,e_2^\star)^\star = (e_1 \mid e_2)^\star = (e_1^\star \mid e_2^\star)^\star & \\
(e_1^\star\,e_2)^\star\,(e_1^\star) = (e_1 \mid e_2)^\star = e_1^\star\,(e_2\,(e_1^\star))^\star &
\end{array}$$