Bureau d'études
Automates et Théorie des Langages
Documents autorisés
1h45

1 Prélude

- Télécharger depuis moodle l'archive `source.tgz`
- Désarchiver son contenu avec la commande : `tar xzvf source.tgz`
- Vous obtenez un répertoire nommé `source`
- Renommer ce répertoire sous la forme `source_Nom1_Nom2` (en remplaçant Nom1 et Nom2 par le nom des deux membres du binôme). Par exemple, si les membres sont Xavier Crégut et Marc Pantel, vous utiliserez la commande : `mv source source_Cregut_Pantel`

2 Postlude

Lorsque la séance se termine à 9h45, vous devrez :

- Vérifier que les résultats de vos travaux sont bien compilables
- Créer une archive avec la commande : `tar xzvf source_Xxx_Yyy.tgz source_Xxx_Yyy`
- Déposer cette archive sur moodle

3 Le format JSON

L'objectif du bureau d'étude est de construire un analyseur lexical pour le format JSON en exploitant l'outil `ocamlex` pour générer l'analyseur lexical et deux analyseurs syntaxiques pour le format JSON en exploitant d'une part l'outil `menhir` pour générer l'analyseur syntaxique, et d'autre part la technique d'analyse descendante récursive programmée en `ocaml` en utilisant la structure de monade.

Voici un exemple de document JSON (*JavaScript Object Notation*) :

```
{
   "prenom" : "Marc",
   "nom" : "Pantel",
   "naissance" : {
      "jour" : 4,
      "mois" : "mai",
      "annee" : 1966
   },
   "matiere" : [ "OMI", "TOB", "GLS", "STL", "SCP", "CL" ]
}
```

Ce format respecte les contraintes suivantes :

- les terminaux sont les chaînes de caractères `chaine`, les nombres `nombre`, les valeurs booléennes `true` et `false`, l'objet non initialisé `null`, les accolades ouvrante `{` et fermante `}`, les crochets ouvrant `[` et fermant `]`, la virgule `,` et le deux points `:` ;
- 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 ;

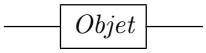
- un attribut est composée d'un nom représenté par une chaîne de caractères et d'une valeur séparé par `:` ;
- une valeur est soit une chaîne de caractères, soit un nombre, soit un objet, soit un tableau, soit une valeur booléenne, soit l'objet non initialisé ;
- un tableau est composé d'une liste entre crochets (`[` et `]`) de valeurs séparées par des virgules `,` . Cette liste peut être vide.

Voici les expressions régulières pour les terminaux :

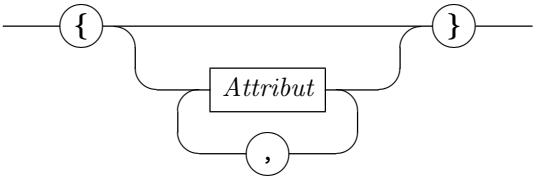
- `chaine` : "$[a-zA-Z]^{\star}$"
- `nombre` : $(\backslash + \mid \backslash -)?[0-9]^{+}$

Voici la grammaire au format graphique de Conway :

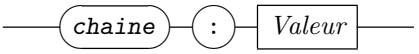
Document



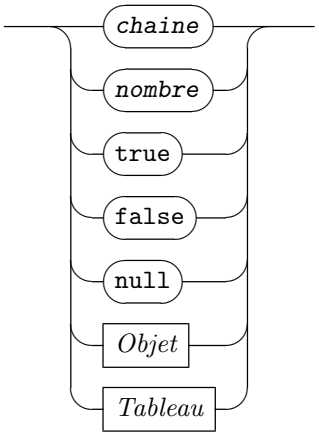
Objet



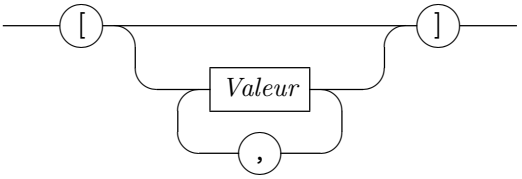
Attribut



Valeur



Tableau



Voici la grammaire sous la forme de règles de production et les symboles directeurs de chaque règle de production :

1.	$D \rightarrow$ **{** L_A **}**	**{**
2.	$L_A \rightarrow \Lambda$	**}**
3.	$L_A \rightarrow A\ S_A$	*chaine*
4.	$S_A \rightarrow \Lambda$	**}**
5.	$S_A \rightarrow$ **,** $A\ S_A$	**,**
6.	$A \rightarrow$ *chaine* **:** V	*chaine*
7.	$V \rightarrow$ *chaine*	*chaine*
8.	$V \rightarrow$ *nombre*	*nombre*
9.	$V \rightarrow$ `true`	`true`
10.	$V \rightarrow$ `false`	`false`
11.	$V \rightarrow$ `null`	`null`
12.	$V \rightarrow$ **{** L_A **}**	**{**
13.	$V \rightarrow$ [L_V]	[
14.	$L_V \rightarrow \Lambda$	]
15.	$L_V \rightarrow V\ S_V$	*chaine nombre* `true false null` **{** [
16.	$S_V \rightarrow \Lambda$	]
17.	$S_V \rightarrow$ **,** $V\ S_V$	**,**

4 Analyseur syntaxique ascendant

Vous devez travailler dans le répertoire `ascendant`.

Vous compilerez régulièrement les modifications réalisées pour détecter les erreurs au plus tôt.

Vous testerez régulièrement votre travail en ajoutant des tests de difficulté croissante dans le répertoire `tests` à la racine de l'archive.

La sémantique de l'analyseur syntaxique consiste à afficher les règles appliquées pour l'analyse.

Complétez les fichiers `Lexer.mll` (analyseur lexical) puis `Parser.mly` (analyseur syntaxique). Le programme principal est contenu dans le fichier `MainJSON.ml`. La commande `jbuilder build MainJSON.exe` produit l'exécutable `_build/default/MainJSON.exe` qui prend comme paramètre le fichier à analyser. L'exemple de ce sujet est disponible dans le répertoire `tests`.

5 Analyseur syntaxique par descente récursive

Vous devez travailler dans le répertoire `descendant`.

Vous compilerez régulièrement les modifications réalisées pour détecter les erreurs au plus tôt.

Vous testerez régulièrement votre travail en ajoutant des tests de difficulté croissante dans le répertoire `tests` à la racine de l'archive.

L'analyseur syntaxique devra afficher les règles appliquées au fur et à mesure de l'analyse. Les éléments nécessaires sont disponibles en commentaires dans le fichier.

Complétez les fichiers `Lexer.mll` (analyseur lexical) puis `Parser.mly` (analyseur syntaxique). Le programme principal est contenu dans le fichier `MainJSON.ml`. La commande `jbuilder build MainJSON.exe` produit l'exécutable `_build/default/MainJSON.exe` qui prend comme paramètre le fichier à analyser. L'exemple de ce sujet est disponible dans le répertoire `tests`.