Analyse de Données Structurées - Cours 8

Ralf Treinen

Université Paris Diderot
UFR Informatique
Laboratoire Preuves, Programmes et Systèmes

treinen@pps.univ-paris-diderot.fr

25 mars 2015

Formats génériques pour des données structurées

- Permettent la représentation textuelle de données structurées
- Utiles pour :
  - Stocker des données dans un fichier : persistance
  - Transmission de données entre des applications hétérogènes (éventuellement écrites dans des langages différents)
  - Transmission de données sur le web (par exemple, AJAX)
  - Des fichiers de configuration (car facile à modifier pour des humains, et facile à interpréter pour des programmes)
- Exemples dans ce cours : JSON et XML

Le format JSON

- **J**ava**S**cript **O**bject **N**otation
- Le document de standardisation commence avec le résumé suivant :

  JavaScript Object Notation (JSON) is a lightweight, text-based, language-independent data interchange format.
  [...]
  JSON defines a small set of formatting rules for the portable representation of structured data.

Le format JSON

- Valeurs de base : `true`, `false`, `null`, valeurs numériques, et chaînes de caractères.
- Deux mécanismes pour combiner des valeurs :
  - *array* : c'est simplement une liste de valeurs entre crochet [et], séparées par des virgules
  - *object* : c'est une séquence de paires, chacune consistant en une chaîne de caractères, et une valeur. La séquence est notée entre accolades { et }, les paires sont séparées par des virgules.
- Attention à la différence entre *terminé* et *séparé* par des virgules.

Exemple

```
{
  "menu": {
    "id": "file",
    "value": "File",
    "popup": {
      "menuitem": [
        { "value": "New", "onclick": "CreateNewDoc()" },
        { "value": "Open", "onclick": "OpenDoc()" },
        { "value": "Close", "onclick": "CloseDoc()" }
      ]
    }
  }
}
```

Source : `http://fr.wikipedia.org/wiki/JavaScript_Object_Notation`

Un autre exemple

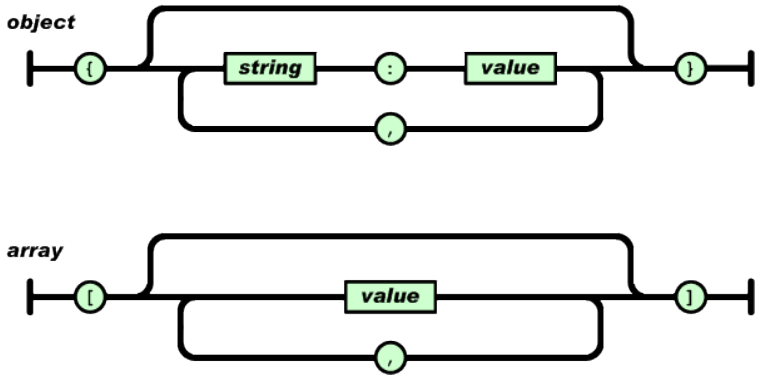
```
{
  "Image": {
    "Width":  800,
    "Height": 600,
    "Title":  "View␣from␣15th␣Floor",
    "Thumbnail": {
      "Url":    "http://www.example.com/image/481989",
      "Height": 125,
      "Width":  100
    },
    "Animated" : false,
    "IDs": [116, 943, 234, 38793]
  }
}
```

Source : `http://tools.ietf.org/html/rfc7159`

La définition officielle

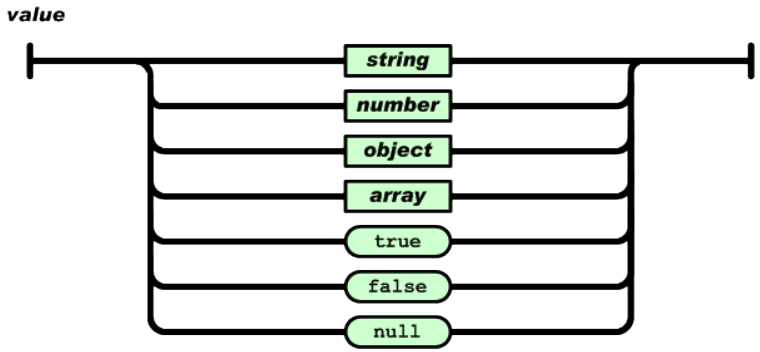
- On trouve sur le “site officiel” de JSON, `http://json.org/`, une série de diagrammes de syntaxe.
- Il y a aussi une proposition de standardisation de la part de la IETF (Internet Engineering Task Force) “RFC7159”, disponible à l'adresse `http://tools.ietf.org/html/rfc7159`.

Diagrammes de syntaxe



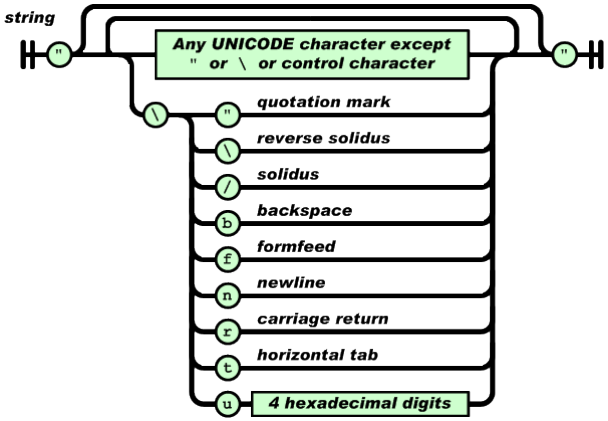
Source : `http://json.org/`

Diagrammes de syntaxe



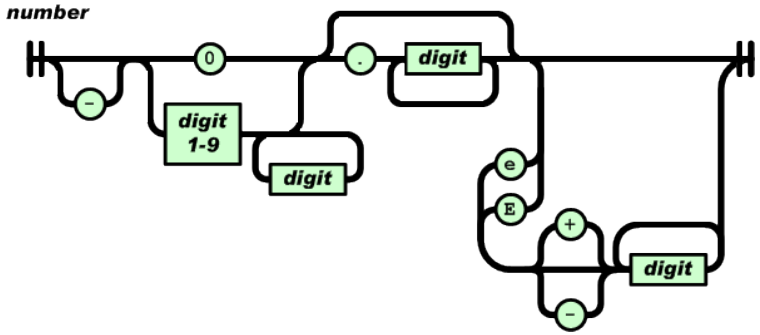
Source : http://json.org/

Diagrammes de syntaxe



Source : http://json.org/

Diagrammes de syntaxe



Source : http://json.org/

Comment faire analyse lexicale/syntaxique ?

- Expressions régulières pour les chaînes et les valeurs numériques.
- Jetons (délivrés par l'analyse lexicale) :
  - `string`, `number`, (munis de valeurs)
  - `true`, `false`, `null`,
  - `,` , `:`, `{`, `}`, `[`, `]`

La grammaire (première tentative)

- $V_N = \{\text{Value}, \text{Object}, \text{Array}, \text{Aseq}, \text{Oseq}, \text{AseqNV}, \text{OseqNV}\}$
- Axiome : value
- Règles :

$$
\begin{array}{rcl}
\text{Value} & \rightarrow & \texttt{string} \mid \texttt{number} \mid \text{Object} \mid \text{Array} \mid \texttt{true} \mid \texttt{false} \mid \texttt{null} \\
\text{Array} & \rightarrow & \texttt{[}\ \text{Aseq}\ \texttt{]} \\
\text{Aseq} & \rightarrow & \epsilon \mid \text{AseqNV} \\
\text{AseqNV} & \rightarrow & \text{Value} \mid \text{Value}\ \texttt{,}\ \text{AseqNV} \\
\text{Object} & \rightarrow & \texttt{\{}\ \text{Oseq}\ \texttt{\}} \\
\text{Oseq} & \rightarrow & \epsilon \mid \text{OseqNV} \\
\text{OseqNV} & \rightarrow & \texttt{string}\ \texttt{:}\ \text{Value} \mid \texttt{string}\ \texttt{:}\ \text{Value}\ \texttt{,}\ \text{Oseq}
\end{array}
$$

- Est-ce LL(1) ?

La grammaire (deuxième tentative)

- $V_N = \{\text{Value}, \text{Object}, \text{Array}, \text{Aseq}, \text{Oseq}, \text{Aseq}', \text{Oseq}'\}$
- Axiome : value
- Règles :

$$
\begin{array}{rcl}
\text{Value} & \rightarrow & \texttt{string} \mid \texttt{number} \mid \text{Object} \mid \text{Array} \mid \texttt{true} \mid \texttt{false} \mid \texttt{null} \\
\text{Array} & \rightarrow & \texttt{[}\ \text{Aseq}\ \texttt{]} \\
\text{Aseq} & \rightarrow & \epsilon \mid \text{Value}\ \text{Aseq}' \\
\text{Aseq}' & \rightarrow & \epsilon \mid \texttt{,}\ \text{Value}\ \text{Aseq}' \\
\text{Object} & \rightarrow & \texttt{\{}\ \text{Oseq}\ \texttt{\}} \\
\text{Oseq} & \rightarrow & \epsilon \mid \texttt{string}\ \texttt{:}\ \text{Value}\ \text{Oseq}' \\
\text{Oseq}' & \rightarrow & \epsilon \mid \texttt{,}\ \texttt{string}\ \texttt{:}\ \text{Value}\ \text{Oseq}'
\end{array}
$$

- Est-ce maintenant LL(1) ?

Analyser un document écrit en JSON

- Il est donc assez facile d'écrire un analyseur syntaxique pour le format JSON.
- Il existe aussi de nombreuses bibliothèques, pour presque toutes les langages de programmation (voir la liste sur `json.org`).

Le format XML

- **E**xtensibe **M**arkup **L**anguage
  (fr. : *langage de balisage extensible*)
- Objectifs : les mêmes que pour JSON
- Il s'agit d'une instance d'un format plus général, du nom SGML (**S**tandard **G**eneralized **M**arkup **L**anguage).
- Langage plus riche que JSON, mais aussi plus verbeux et moins élégant.
- Le langage XML est standardisé par le W3C (World Wide Web Consortium).

La structure d'un document XML

- Un document XML décrit un arbre.
- Un nœud peut avoir un nombre arbitraire d'enfants.
- Un nœud peut avoir des *attributs*.
- Les enfants d'un nœud sont appelés ses *éléments* ; un élément peut être un morceau de texte, ou un autre nœud.

Exemple

Le même exemple que dans la section sur JSON :

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()">
    </menuitem>
    <menuitem value="Open" onclick="OpenDoc()">
    </menuitem>
    <menuitem value="Close" onclick="CloseDoc()">
    </menuitem>
  </popup>
</menu>
```

(sans d'utiliser le raccourci pour les nœuds sans enfants)

Remarques

- Ça ressemble à du HTML, et pour cause : XML et HTML sont tous les deux des dérivés du langage plus général SGML.
- L'utilisation des chevrons < et > et assez caractéristique pour ces langages.
- Le XML est plus stricte que le HTML, par exemple :
  - XML exige que les valeurs des attributs sont écrites entre guillemets, contrairement aux premières versions de HTML.
  - XML exige que toutes les balises sont proprement fermées, contrairement à HTML qui est très libéral en ce regard.

Vers une définition de la syntaxe

- Un nœud commence avec une balise (angl. : *tag*) de la forme

  $$<m>$$

  et se termine avec

  $$</m>$$

  nom est mot quelconque, appelé le *nom* de cet élément.
- Le nom est obligatoirement le même dans la balise de début et de fin du même nœud. Il est permis d'utiliser le même nom pour plusieurs nœuds du même arbre.
- Une balise de début peut en plus contenir des paires de attributs avec leurs valeur.

Vers une définition de la syntaxe

- Une paire attribut/valeur est donnée dans la forme

  attr=valeur

  où *attr* est un mot (appelé l'attribut), et *valeur* est une chaîne de caractères entre guillemets (appelée la valeur).
- Il n'est pas permis définir dans la même balise deux fois le même attribut.

Raccourci pour des nœuds sans enfants

La syntaxe

<nom attr1="value1" ... attrn="valuen" />

est un raccourci pour

<nom attr1="value1" ... attrn="valuen">
</nom>

Exemple

Le même exemple avec ce raccourci :

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

Source : `http://fr.wikipedia.org/wiki/JavaScript_Object_Notation`

Exemple : Données OpenStreetMap

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6"
     copyright="OpenStreetMap and contributors"
     license="http://opendatacommons.org/licenses/odbl/1-0/">
  <way id="62378611"
       visible="true" version="8" changeset="20691652"
       timestamp="2014-02-21T11:23:38Z" user="thibdrev" uid="1279506">
    <nd ref="779143878"/>
    <nd ref="2198721646"/>
    <nd ref="2198721727"/>
    ........
    <tag k="amenity" v="university"/>
    <tag k="building" v="yes"/>
    <tag k="name" v="Halle aux Farines (Université Paris Diderot)"/>
    <tag k="source" v="cadastre-dgi-fr source : Direction Générale des Impôts
    <tag k="wheelchair" v="yes"/>
    <tag k="wikipedia" v="fr:Université Paris VII - Diderot"/>
  </way>
</osm>
```

Une grammaire pour XML ?

- Terminaux (jetons) : `mot`, `string`, `texte`, `/`, `=`, `<`, `>`
- Non-terminaux : Arbre, Start, End, Attrbs, Elements
- Axiome : Arbre
- Règles :

$$
\begin{aligned}
\text{Arbre} &\rightarrow \text{Start Elements End} \\
\text{Start} &\rightarrow \texttt{<}\ \texttt{mot}\ \text{Attrbs}\ \texttt{>} \\
\text{End} &\rightarrow \texttt{<}\ \texttt{/}\ \texttt{mot}\ \texttt{>} \\
\text{Attrbs} &\rightarrow \epsilon \mid \texttt{mot}\ \texttt{=}\ \texttt{string}\ \text{Attrbs} \\
\text{Elements} &\rightarrow \epsilon \mid \text{Arbre Elements} \mid \texttt{texte}\ \text{Elements}
\end{aligned}
$$

Une grammaire pour XML ?

- Attention : Cette grammaire ne définit pas complètement le langage XML.
- Il y a deux éléments de la définition qui ne sont pas exprimés par la grammaire :
  - Toute balise de début doit être fermée par une balise de fin *du même nom*.
  - Tous les attributs dans une balise de début doivent être *différents*.
- Peut on améliorer la grammaire pour exprimer aussi ces deux restrictions ?
- Réponse : non ! voir les transparents suivants.

Vers une preuve

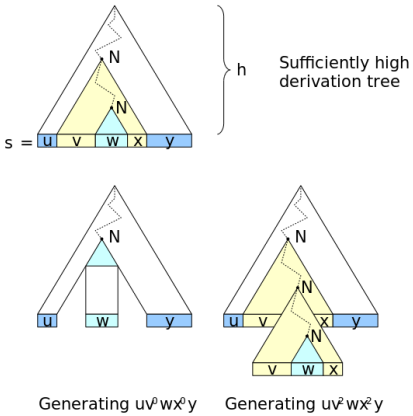
- Nous allons montrer que c'est impossible de définir le langage XML entièrement par une grammaire hors-contexte.
- Cela impliquera que c'est également impossible on utilisant une combinaison expressions régulières - grammaire hors-contexte, car toute expression régulière peut être transformée en une grammaire.

Rappel : lemme de pompage pour les langages algébriques

Soit L un langage algébrique. Alors il existe un $p \geq 0$ tel que tout mot $s \in L$ avec $|s| \geq p$ peut être décomposé en $z = uvwxy$ avec :

- $|vwx| \leq p$
- $|vx| \geq 1$
- $uv^i wx^i y \in L$ pour tout $i \geq 0$

Idée de la preuve du lemme de pompage



Source : Jochen Burghardt

Application du lemme de pompage à XML (1)

Considérons le mot suivant qui est en L_{XML} :

$$s = <\underbrace{a\cdots a}_{p}\underbrace{b\cdots b}_{p}><\!/\underbrace{a\cdots a}_{p}\underbrace{b\cdots b}_{p}>$$

- On peut écrire $s = uvwxy$, avec $|vwx| \leq p$ et $|vx| \geq 1$, tel que $uwy \in L_{XML}$.
- Cas 1 : vwx est entièrement dans la partie verte ou la partie rouge : contradiction car, après passage à uwy, les parties verte et rouge sont de longueur différente !

Application du lemme de pompage à XML (2)

$$s = <\underbrace{a\cdots a}_{p}\underbrace{b\cdots b}_{p}><\!/\underbrace{a\cdots a}_{p}\underbrace{b\cdots b}_{p}>$$

- $s = uvwxy$, $|vwx| \leq p$, $|vx| \geq 1$, $uwy \in L$.
- Cas 2 : vwx est à cheval entre vert et rouge.
- Dans ce cas, v et x ne peuvent pas couvrir la partie $><\!/$, sinon pompage fait disparaître un des symboles $<,>,/$.
- Donc, on passant à uwy, on efface des b dans le vert, ou des a dans le rouge, ou les deux : contradiction !

Conclusion : Validation du XML

- Il est inévitable que la grammaire définit seulement une *sur-approximation* du langage XML.
- Il nous faut une étape supplémentaire de validation qui vérifie les deux restrictions supplémentaires.
- On rencontre le même problème dans les langages de programmation : toute variable doit être déclarée avant son utilisation.

Pour aller plus loin

- Il existe un langage de *schéma* pour des documents XML. Un schéma restreint la forme d'un document XML. DTD (**D**ocument **T**ype **D**efinition).
- Un schéma peut par exemple définir quels attributs sont obligatoires (autorisés) pour quel type, ou quels types d'éléments sont obligatoires (autorisés) pour un nœud d'un certain type.
- Il existe aussi des langages riches pour des requêtes (extraction de données d'un document XML), par ex. XPath, XQuery.
- Pour en savoir plus : voir le cours XML du M1