Compilateur d'automates

1 Sujet

Le but de ce mini-projet est de réaliser un compilateur d'automate fini appelé **scc** (state chart compiler). Ce compilateur permettra de fournir quelques informations sur l'automate et de faire quelques vérifications sur la bonne écriture de celui-ci.

Le langage SCXML

Votre compilateur devra reconnaître un sous-ensemble du langage à balises `SCXML` (State Chart XML) qui permet de décrire une machine d'états et donc en particulier un automate :

`http://en.wikipedia.org/wiki/SCXML`
`http://commons.apache.org/scxml/guide/scxml-documents.html`

Vous trouverez dans le cours `Langages et Traducteurs - GIS3` de `Moodle Lille 1` le fichier `exemple.xml` ci-dessous décrivant un automate à partir du langage `SCXML` :

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <scxml xmlns="http://www.w3.org/2005/07/scxml" initial="s1">
3   <state id="s1">
4     <transition event="a" target="s1" />
5     <transition event="b" target="s2" />
6     <transition event="a" target="s4" />
7   </state>
8   <state id="s2">
9     <transition event="c" target="s3" />
10    <transition event="e" target="s5" />
11  </state>
12  <state id="s3">
13    <transition event="d" target="s4" />
14  </state>
15  <state id="s4" final="true">
16  </state>
17  <state id="s5">
18    <transition event="a" target="s6" />
19  </state>
20  <state id="s6" final="true">
21    <transition event="a" target="s4" />
22  </state>
23 </scxml>
```

Un document SCXML dispose d'un prologue (ligne 1) (toujours le même) suivi d'une balise `scxml` contenant deux attributs (ligne 2). Le premier attribut `xmlns` indique la référence officielle du document XML (dans le cadre du projet, ce sera toujours la même valeur), le second attribut indique le nom de l'état initial de l'automate. La balise `scxml` englobe l'ensemble des états de l'automate décrit à l'aide des balises `state` (lignes 3-22).

La balise `state` contient obligatoirement un attribut `id` qui indique le nom de l'état (exemple ligne 8) suivi éventuellement de l'attribut `final` (exemple ligne 15) qui permet de préciser les états finaux de l'automate (état final si la valeur est égale à `true`).

Les éventuels arcs sortants d'un état de l'automate sont décrits à l'aide de la balise `transition` (exemple lignes 4-6). La balise `transition` dispose obligatoirement de deux attributs : l'attribut `event` qui précise l'étiquette de l'arc suivi de l'attribut `target` qui indique le nom de l'état de destination de l'arc.

La figure 1 est une représentation graphique conforme à cet exemple SCXML. Dans le cadre de ce projet, nous n'exploiterons pas d'autres balises du langage SCXML que celles décrites pour cet exemple.

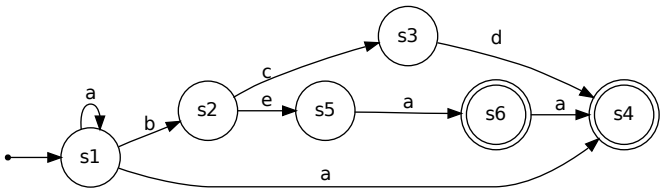

FIGURE 1 – Représentation graphique de l'exemple

2 Découpage du projet

Le projet est décliné en plusieurs versions. Vous devez implémenter chaque version à l'aide des outils `Lex` et `Accent`.

Attention, la grammaire définie dans la première version ne doit pas être modifiée dans les versions suivantes. Seuls des attributs et des actions sémantiques pourront être ajoutés à cette version de base.

2.1 Première version

Proposez une grammaire permettant de vérifier si un document `XML` définit un automate tel que décrit précédemment.

Remarques :

- Le prologue du document doit toujours être :

  <?**xml version**="1.0" encoding="UTF–8"?>

- `xmlns`, le premier attribut de la balise `scxml`, doit contenir la valeur :

  "http://www.w3.org/2005/07/scxml"

- Le nom d'un état ou de l'étiquette d'un arc est constitué d'une lettre suivi d'une suite pouvant être vide de chiffres, de lettres et des caractères _ et -.
- Il n'y a pas d'espace entre les valeurs des attributs et leurs guillements.

2.2 Deuxième version

A partir de la grammaire définie dans la **première version**, proposez un schéma de traduction dirigé par la syntaxe permettant d'afficher pour chaque automate reconnu :

- son état initial,
- la liste de ses états en précisant pour chacun d'eux son nom, ses arcs, son nombre d'arcs et si c'est un état final,
- son nombre total d'états,
- son nombre total d'états finaux et
- son nombre total d'arcs.

Pour le fichier `exemple.xml`, l'affichage doit être de la forme suivante :

```
Etat initial : s1

   Etat s1
      Arc : Etiquette a - Etat s1
      Arc : Etiquette b - Etat s2
      Arc : Etiquette a - Etat s4
   Nombre d arc(s) : 3

   Etat s2
      Arc : Etiquette c - Etat s3
      Arc : Etiquette e - Etat s5
   Nombre d arc(s) : 2
```

```
   Etat s3
      Arc : Etiquette d - Etat s4
   Nombre d arc(s) : 1

   Etat s4 : Etat final
   Nombre d arc(s) : 0

   Etat s5
      Arc : Etiquette a - Etat s6
   Nombre d arc(s) : 1

   Etat s6 : Etat final
      Arc : Etiquette a - Etat s4
   Nombre d arc(s) : 1

Nombre total d etats : 6
Nombre d etats finaux : 2
Nombre total d'arcs : 8
```

Remarques :

- Dans cette partie, aucune vérification n'est à effectuer sur les états et leurs arcs : un état ou un arc peut être déclaré plusieurs fois.
- Pour effectuer les traitements demandés, vous devez utiliser uniquement des attributs et vous devez ajouter à la grammaire de la **première version** les actions qui permettent de faire le calcul de ces attributs et les affichages demandés.
- Vous devez utiliser la partie 2 du TP n°4 (Accent : Gestion des attributs des terminaux) pour récupérer les noms des états et des arcs à partir de votre lexer.

2.3 Troisième version

Dans cette version, vous allez repartir de la **première version** pour proposer un schéma de traduction dirigé par la syntaxe permettant d'effectuer les vérifications suivantes :

- deux états ne doivent pas avoir le même nom,
- l'attribut `initial` de la balise `scxml` doit référencer un attribut `id` d'une balise `state`,
- l'attribut `final` de la balise `state` doit avoir pour valeur soit `true`, soit `false`,
- deux arcs ne doivent pas avoir les mêmes état de départ, état d'arrivée et étiquette.

Pour l'implémentation de cette version :

- Nous considérons que nous gérons des automates d'au plus 500 états, que chaque état a au plus 100 arcs et que les chaînes de caractères des attributs ne dépassent pas 50 caractères.
- Dès qu'un des points à vérifier n'est pas satisfait, l'analyse doit se terminer en indiquant le plus précisément possible le problème rencontré.
- Vous allez utiliser les fichiers `etats.h` et `etats.c` qui se trouvent dans le cours `Langages et Traducteurs - GIS3` de la plateforme `Moodle Lille 1` :
  - Le fichier `etats.c` doit être ajouté au niveau de la compilation C dans le fichier `makefile`.
  - Dans le fichier `etats.h` sont définis :
    - le type `chaineC` correspondant à un vecteur de 50 caractères,
    - la structure `tabEtats` contenant un champ `nbEtats` qui sera être égal au nombre total d'états de l'automate et un vecteur `vecEtats` contenant les informations de chaque état de l'automate,
    - la structure `etatInfos` indiquant pour chaque état son nom (champ `nomEtat`), son nombre d'arcs (champ `nbArcs`), ses arcs (champ `vecArcs`), si cet étal est final (champ `etatInitial` à `VRAI`) et si cet état est final (champ `etatFinal` à `VRAI`),
    - la structure `arcInfos` indiquant pour chaque arc le nom de son état d'arrivée et le nom de son étiquette.

Remarques :

- Vous devez ajouter à la grammaire de la **première version**, des actions sémantiques effectuant la gestion des états de l'automate en déclarant une variable de type `tabEtats` dans votre parser et en utilisant les traitements définis dans le fichier `etats.c`.
- Dans le cas où l'analyse d'un document `XML` se termine avec succès, vous devez utiliser l'action `afficherTable` du fichier `etats.c` pour afficher l'ensemble des informations sur l'automate défini dans ce document.

3 Travaux à rendre

3.1 Travaux intermédiaires

A la fin de chaque séance, vous devez déposer sur la plateforme `Moodle Lille 1`, l'ensemble des travaux que vous avez réalisé depuis le début du projet en suivant les consignes suivantes :

1. Créez une archive compressée des documents à déposer avec la commande :

   ```
   tar -czvf nom1_nom2.tgz Dossier
   ```

   où `nom1` et `nom2` sont les noms des étudiants composant votre groupe et `Dossier` est le répertoire contenant les documents que vous devez rendre.
2. Sélectionnez dans le cours `Langages et Traducteurs - GIS3` la séance appropriée.
3. Déposez votre archive compressée.

3.2 Rendu final

Vous devez déposer pour le **jeudi 12 avril 17h** au plus tard :

- Sur la plateforme `Moodle Lille 1`, une archive de vos versions contenant des fichiers de test pour chaque version (les fichiers de vos différentes versions doivent être rangés dans des répertoires de noms `V1`, `V2` et `V3`).
- Un rapport imprimé dans le casier courrier de votre tuteur.

  Votre rapport doit être décomposé en fonction des versions à réaliser. Vous devez décrire dans ce rapport :
  - les grammaires et schémas de traduction dirigée par la syntaxe utilisés,
  - les structures et types que vous avez définis,
  - les attributs et variables utilisés, et leurs objectifs,
  - les tests que vous avez réalisés,
  - ...

  La conclusion de votre rapport doit comporter une critique du travail réalisé et mentionner des perspectives d'amélioration de votre travail.

  Votre rapport doit contenir également une impression de vos différents fichiers.