TP 2
Vaucanson – 1

Version du 16 septembre 2013

Dans le cadre de ce TP, nous allons utiliser une partie de Vaucanson[1]. Ce projet est une bibliothèque de manipulation d'automates développée au LRDE en collaboration avec Télécom ParisTech. Un ensemble de logiciels nommé TAF-Kit utilisant cette bibliothèque est dédié à l'utilisation en ligne de commande d'automates.

Nous utiliserons plus particulièrement la commande `vcsn-char-b` qui permet de manipuler des automates « classiques ». Il est possible de l'utiliser pour définir des automates interactivement, d'en exporter vers divers formats, d'évaluer des mots, etc.

'`vcsn-char-b -l`' permet de connaître l'ensemble des commandes disponibles avec cet outil (nous n'en utiliserons qu'une petite partie).

De manière générale, '`vcsn-char-b`' s'utilise de la manière suivante : '`vcsn-char-b` *`commande arguments`*'. Tapez '`vcsn-char-b --help`' pour obtenir une aide générale.

Pour travailler avec cet outil, vous devez d'abord entrer dans votre terminal les commandes suivantes[2] :

```
% cd
% wget www.lrde.epita.fr/dload/vaucanson/thlr/vcsn139rack.tgz
% tar zxvf vcsn139rack.tgz
% rm vcsn139rack.tgz
% export PATH=$HOME/usr/bin:$PATH
% export LD_LIBRARY_PATH=$HOME/usr/lib
% export VCSN_DATA_PATH=$HOME/usr/share/vaucanson
```

Vous devrez retaper ces trois dernières lignes demain ou dans chaque nouveau terminal que vous ouvrez.

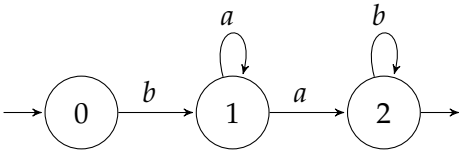
Exercice 1

1. Plusieurs automates sont prédéfinis par TAF-Kit. Leur liste est affichée par '`vcsn-char-b --list-automata`'. Pour les utiliser, passez leur nom en argument des commandes de TAF-Kit qui attendent un automate. Par exemple pour envoyer le fichier sur la sortie standard, vous pouvez utiliser la commande '`vcsn-char-b cat` *`automate.xml`*'[3], qui, comme son nom l'indique, reproduit un automate à l'identique.

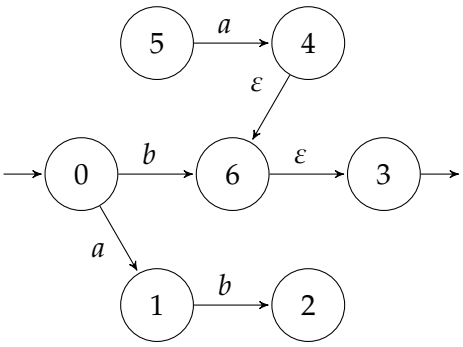
   Copiez les automates prédéfinis dans votre répertoire en exécutant la commande '`vcsn-char-b cat` *`automate.xml`* `>` *`automate.xml`*' pour tous les automates.
2. Il est possible de créer (ou éditer) interactivement des automates en tapant '`vcsn-char-b edit -a abc` *`automate.xml`*'. Il apparaît alors un menu textuel dans lequel vous pourrez effectuer plusieurs opérations de création ou modification. L'option '`-a abc`' permet de spécifier l'alphabet sur lequel on travaille, ici $\{a, b, c\}$. Lorsque vous quittez le programme, le fichier *`automate.xml`* est sauvegardé.

   Créez l'automate 1.
3. Pour afficher un automate rapidement et directement à partir de son fichier XML, vous pouvez utiliser '`vcsn-char-b display` *`automate.xml`*'. Essayez sur plusieurs automates.
4. Vous pouvez émonder un automate (ce n'est pas sale) en tapant '`vcsn-char-b trim` *`automate.xml`* `>` *`automate_emonde.xml`*'.

1. http://vaucanson.lrde.epita.fr
2. Remplacez '`export A=B`' par '`setenv A B`' si votre shell est `csh` ou `tcsh`.
3. Un texte en italique, comme '*`automate.xml`*' ici, signifie qu'il s'agit d'un paramètre qu'il vous faudra substituer par la valeur adéquate ; cela restera vrai pour le reste de l'énoncé.



Automate 1: Un automate à saisir



Automate 2: Un automate à émonder.

Créez donc l'automate 2, puis émondez-le à l'aide de la commande '`trim`'. Vérifiez que le résultat correspond bien à celui que vous attendez (c'est-à-dire l'automate privé des états qui ne sont pas co-accessibles ou pas accessibles).

Essayez aussi les commandes '`accessible`' (pour ne garder que les états accessibles) et '`coaccessible`' (pour ne garder que les états co-accessibles).

5. La commande '`vcsn-char-b proper automate.xml > automate_sans_eps.xml`' permet de supprimer les transitions spontanées. Appliquez-la à l'automate précédent pour vérifier sont fonctionnement.
6. La commande '`vcsn-char-b -a abc thompson 'expression_rationnelle' > exp.xml`' permet de générer l'automate correspondant à une expression rationnelle à l'aide de l'algorithme de Thompson, avec l'alphabet $\{a, b, c\}$. Attention, Vaucanson ne gère pas les expressions rationnelles avec sucre syntaxique ! La syntaxe est la suivante :

   — '`a+b`' : '`a`' ou '`b`',
   — '`()`' : groupement,
   — '`a.b`' : '`a`' suivi de '`b`',
   — '`a*`' : '`a`' répété $n \geq 0$ fois,
   — '`1`' : le mot vide, ε.

   D'autre part, pour Vaucanson la concaténation implicite ('`ab`') a une priorité plus forte que tous les autres opérateurs, en particulier l'étoile. Ainsi '`ab*`' est équivalant pour Vaucanson à '`(ab)*`' et non '`a(b*)`' comme on pourrait s'y attendre... Le mieux est encore de toujours utiliser la concaténation explicite ('`a.b`') pour éviter les soucis : '`a.b*`' correspond bien à '`a.(b*)`'.

   Générez les représentations graphiques des automates correspondant aux expressions rationnelles suivantes exprimées avec la syntaxe de Perl (il faut donc les réécrire pour Vaucanson). On supposera que l'alphabet est $\{a, b, c\}$.

   — '`ab+`'
   — '`ab*`'
   — '`ab+|ab*`'
   — '`abc|(bac)*|(cab)+`'
   — `[a-c]*[^a]`
   — '`a?bc|ab?c`'

7. Construisez un automate reconnaissant les nombres décimaux qui sont divisibles par 3. Vous utiliserez '`vcsn-char-b -adigits edit div3.xml`' pour créer l'automate en utilisant un alphabet de chiffres. Vous considérerez que le mot vide (noté '`e`' car '`1`' désigne maintenant une lettre de l'alphabet) est équivalent au nombre 0 et fait donc partie du langage reconnu par l'automate.

   N'hésitez pas à vous inspirer de l'automate '`div3base2.xml`', que vous avez récupéré à la première question et qui fait la même chose sur les nombres binaires.

   Pour vérifier votre automate, utilisez la commande '`vcsn-char-b enumerate div3.xml 5`'. Elle va énumérer tous les mots de taille ≤ 5 reconnus par votre automate.