TP 1:
Opérations sur les automates

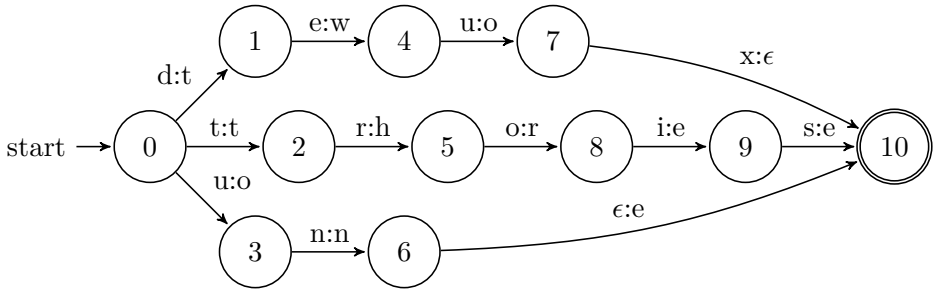
Introduction

Les transducteurs

Nous allons introduire brièvement les transducteurs (cf. Cours n° 2). Un transducteur est un dispositif recevant un message sous une certaine forme et le transformant en une autre.

C'est transducteur est un automate fini qui non seulement **reconnaît** un ensemble régulier de chaîne mais encore la **traduit** dans une autre chaîne appartenant à un autre langage régulier. Tout comme un automate accepteur, on peut le représenter par un graphe orienté étiqueté où chaque transition est étiquetée par un (ou aucun) symbole à reconnaître et un (ou aucun) symbole utilisé pour la traduction.

Un transducteur est défini sur deux alphabets: un alphabet d'entrée Σ_i et un alphabet de sortie Σ_o.



Que ce soit pour un automate ou un transducteur, chacun des arcs peut être pondéré, mais nous verrons cela au TP 2.

OpenFST

OpenFST est une librairie C++ qui permet de gérer des automates à états finis. Cet outil permet de travailler de manière équivalente sur des automates non-pondérés (FSA), pondérés (WFA), des transducteurs non-pondérés (FST) ou pondérés (WFST).

La représentation des automates avec OpenFST nécessite la définition d'un certain nombre de fichiers.

(a) Définition de l'automate: `A.txt`

```
A.txt
fromState toState inSymbol [outSymbol] [weight]
terminalState
```

- `fromState`, `toState` et `terminalState` sont des nombres entiers correspondant aux états de l'automate.
- `inSymbol` et `outSymbol` (pour les FST) sont des chaînes de caractères correspondantes au nom des symboles des alphabets d'entrée Σ_i et de sortie Σ_o.
- `weight` est un nombre décimal qui correspond au poids de la transition (WSA, WFST) (cf. TP 2).

(b) Table de symboles d'entrée: `A.isyms` et, pour les FST et WFST uniquement, table de symboles de sortie `A.osyms` .

```
A.isyms
symbol integer
```

- `symbol` est un symbole de l'alphabet correspondant
- `integer` est un entier positif tel que chaque symbole est associé à un unique entier. C'est-à-dire que chaque entier ne doit apparaître qu'une seule fois dans ce fichier.

(c) La représentation binaire de l'automate `A.fsa` ou `A.fst` est obtenue à l'aide de la commande `fstcompile`.

```
FSA: fstcompile --acceptor --isymbols=A.isyms [--keep_isymbols] A.txt A.fsa
FST: fstcompile --isymbols=A.isyms --osymbols=A.osyms [--keep_isymbols] [--keep_osymbols]
    A.txt A.fst
```

L'option `keep_isymbols` permet d'intégrer les symboles directement dans l'automate binaire et ainsi d'éviter de les respécifier à chaque opération (comme par exemple, si on souhaite représenter graphiquement l'automate).

(d) La représentation graphique de l'automate `A.dot` obtenue à l'aide de la commande `fstdraw`.

```
FSA: fstdraw --acceptor --portait [--isymbols=A.isyms] A.fsa
FST: fstdraw --portait [--isymbols=A.isyms] [--osymbols=A.osyms] A.fst
```

Puis par exemple, rediriger vers `dot` si l'outil est installé.

```
cmd | dot -Tpng > eg.png
```

Exercice 1. Prise en main

1. Pour vérifier que OpenFST est bien installé, taper dans un terminal la commande `fstcompile --help`. Lire la documentation affichée.
2. Construire les fichiers `A.txt` et `A.isyms` correspondant à l'automate $\mathcal{A}$ suivant:

$$0 \xrightarrow{a,b} 0 \quad 0 \xrightarrow{a} 1 \quad 1 \xrightarrow{a,b} 1 \quad 1 \xrightarrow{a} 3$$
$$0 \xrightarrow{b} 2 \quad 2 \xrightarrow{a,b} 2 \quad 2 \xrightarrow{b} 3$$

   De même pour l'automate $\mathcal{B}$ suivant:

$$0 \xrightarrow{a} 0 \quad 0 \xrightarrow{b} 1 \quad 1 \xrightarrow{a} 2$$
$$1 \xrightarrow{a} 0 \quad 2 \xrightarrow{b} 1 \quad 2 \xrightarrow{b} 2$$

3. Compiler ces automates et les représenter graphiquement.
4. Déterminer 8 mots du langage reconnus soit par $\mathcal{A}$ soit par $\mathcal{B}$, ou bien les deux. Ces mots pourront être utilisés par la suite pour vérifier vos résultats.
5. A partir des fonctions que vous pouvez trouver (celles qui commencent par `fst...`), déterminer et représenter graphiquement:
   - L'union: $C = A \cup B$
   - L'intersection: $D = A \cap B$
   - La concaténation: $E = A + B$
   - Le complémentaire $F = \bar{A}$

- Les fermetures de Kleene: $G = A^\star$ et $H = B^\star$

6. Parmi l'ensemble des automates obtenus, choisir un automate qui contient des ϵ-transitions. Supprimer ces transitions vides avec la commande appropriée. Vérifier le résultat obtenu.

7. Aucun des deux automates n'est déterministe. Les déterminiser. Vérifier vos résultats. Peut-on déterminiser directement un automate qui possède des transitions vides sans passer par la suppression de ces transitions ?

8. Pour chacun des automates obtenus, les minimiser avec la commande adéquate. Pour rappel, l'opération de minimisation consiste à supprimer les états équivalents: cela permet d'éviter les redondances. Quels sont les automates qui ont été minimisés ?

9. OpenFST ne permet pas de tester simplement si un mot est reconnu par un automate ou non. Une solution est d'utiliser la commande `fstshortestpath [--nshortest=n --unique]`. Elle permet d'obtenir, sous forme d'automate, les n mots reconnus par l'automate (dstincts ou non) qui ont le plus court chemin dans l'automate. Evidemment plus on augmente n, plus on a de mots reconnus. Donner par exemple, les 6 mots reconnus par l'automate $\mathcal{A}$ qui ont le plus court chemin.

Exercice 2. Reconnaissance de mots

On souhaite reconnaître des mots appelés "tokens" (constitués d'une chaîne de caractère) dans une séquence de caractères ASCII contenant éventuellement des ponctuations et des espaces. Pour cela, nous allons construire des automates reconnaissant les mots `Mars`, `homme` et `Martien`

1. Construire trois automates reconnaissant chacun un des mots `Mars`, `homme` et `Martien` dans une séquence de caractère. Vous n'utiliserez qu'un seul fichier de symboles `ascii.isyms`.

2. Si vous deviez ajouter un grand nombre de séquences de caractères ASCII, comment feriez-vous pour modifier le fichier de symboles ? Proposer un script python permettant d'écrire un fichier de symboles utilisable pour reconnaître n'importe quelle séquence de caractère ASCII. Générer ce fichier.

3. Construire un automate fini minimal et déterministe permettant de reconnaître si au moins un des trois mots est présent dans une chaîne de caractère ASCII. L'automate doit également reconnaître des enchaînements de ces trois mots.

4. Vérifier avec la commande `fstshortestpath` que l'automate reconnaît bien les mots qu'il doit reconnaître.

Exercice 3. Tokenization

À présent, on souhaite construire non plus des automates accepteurs mais des automates transducteurs. C'est-à-dire que l'automate va non seulement **accepter** certains mots mais également les **traduire**. Ces transducteurs vont reconnaître une chaîne de caractères ASCII et renvoyer le mot correspondant.

1. Reprendre les questions 1 et 3 de l'exercice 2, mais cette fois en renvoyant le mot correspondant à la chaîne de caractères reconnue. On utilisera en entrée l'alphabet généré par le script python et en sortie un alphabet que vous devez définir.

   Pour savoir si un mot est correctement reconnu et traduit par le transducteur, on va utiliser une opération très courante pour les transducteurs, à savoir la composition. Cette opération binaire permet de combiner l'entrée d'un transducteur R_1 avec la sortie d'un transducteur R_2 tel que l'alphabet de sortie de R_1 est identique à l'alphabet d'entrée de R_2 ($\Sigma_o^1 = \Sigma_i^2$) :

   $$R_2 \circ R_1 = \{(u, v) | \exists z : (u, z) \in R_1, (z, v) \in R_2\}$$

   Attention dans ce cas, les alphabets d'entrée et de sortie sont également combinés.

2. Nous allons vérifier si la chaîne de caractère `hommeMars` est bien reconnue et traduite par l'automate obtenu à la question précédente. Construire un automate (accepteur) `hommeMars.fsa` reconnaissant le mot. Composer ce nouvel automate (transducteur) avec celui obtenu à la question précédente à l'aide de la commande suivante:
   `fstcompose R1.fst R2.fst | fstproject --project_output > R.fst` .

3. D'après la documentation, à quoi sert la commande `fstproject --project\_output` ? Pourquoi en a-t-on besoin ici ? Êtes-vous satisfait du résultat obtenu ?

4. Essayer maintenant avec les chaînes de caractères suivantes

   - `homme␣Mars`
   - `hommemars`

5. En pratique, on aimerait bien pouvoir tokéniser la chaîne de caractères `homme␣Mars`. Plus généralement, on souhaite regrouper une séquence de caractère en "token" (ou mot) lorsqu'on rencontre une ponctuation ou un espace. Proposer une solution. Est-ce que cette fois la chaîne `homme␣Mars` est bien reconnue ? Quels sont les avantages et inconvénients de la solution proposée ?