Projet : Création et rendu d'automates en SVG

Consignes :

- Lisez bien tout le sujet avant de commencer à coder.
- Déposez avant le **mercredi 26 mai à 23h59** une archive au format `.tar.gz`, contenant votre code pour les trois premières parties, vos fichiers de test éventuels et un rapport au **format PDF**, dans le dépôt `proc_2020` sur `http://exam.ensiie.fr`.
- Déposez avant le **vendredi 4 juin à 23h59** une archive au format `.tar.gz`, contenant votre code pour la dernière partie dans le dépôt `lasf_2020` sur `http://exam.ensiie.fr`.

L'objectif de ce projet est de développer un outil permettant de définir et de manipuler des automates finis, et d'en faire le rendu au format SVG.

Le sujet est composé de quatre parties :

1. la description du langage de commandes que nous utiliserons pour définir un automate,
2. le rendu en SVG,
3. une liste de plusieurs améliorations possibles,
4. **[LASF]** quelques traitements sur les automates à implanter.

Vous aurez besoin d'avoir une structure de données pour représenter les automates finis afin de faire la dernière partie, qui servira d'évaluation pour le cours LASF. Il est vivement recommandé d'utiliser cette structure dès la première partie.

1 Un langage dédié à la définition d'un automate fini

L'objectif de cette partie est de poser les bases du projet, c'est-à-dire de produire :
- un analyseur lexical (fichier `.l`)
- un analyseur syntaxique (fichier `.y`)
- un programme principal pour lier le tout.

Afin de définir un automate fini, nous allons utiliser une suite de commandes :

CREATE pour créer un nouveau noeud ou une nouvelle transition,

REMOVE pour supprimer un objet créé par un précédent appel à `CREATE`,

MOVE pour déplacer un ou plusieurs noeuds,

DUMP pour déclencher un rendu,

RENAME pour renommer un noeud,

EDIT pour modifier les attributs d'un objet.

1.1 Définitions préliminaires

Dans toute la suite, un identifiant sera une chaîne de 15 caractères maximum, servant à identifier de manière unique un noeud. Cette chaîne devra commencer par une lettre, et ne contenir que des lettres, des chiffres et le caractère `_`.

Un label sera une chaîne de caractères quelconque [1], qui correspondra en général à du texte à afficher (nom donné à un noeud, étiquette d'une flèche). La chaîne sera entourée par des guillemets doubles. Pour faciliter les choses, on supposera en plus que les caractères \, <, > et " sont interdits.

1. On pourra imposer une taille maximale si nécessaire, mais cette limitation n'est pas recommandée.

Les constantes numériques seront des nombres flottants, sauf mention contraire.

Les directions seront notées soit par un point cardinal (`NORTH`, `WEST`, `SOUTH` ou `EAST`), soit par un couple `<dir1>-<dir2>`. Dans ce cas, `<dir1>` sera forcément `NORTH` ou `SOUTH`, `<dir2>` sera `WEST` ou `EAST` et ces deux mots clés seront séparés par un tiret.

Enfin, les mots clés du langage seront écrits tout en majuscules dans ce document. Cependant, votre programme devra aussi accepter les commandes avec des mots clés en minuscules, ou avec un mélange de majuscules et minuscules. Ces mots clés sont :

CREATE	REMOVE	MOVE	DUMP	EDIT	RENAME
NODE	EDGE	FROM	TO	AT	WITH
LABEL	SIZE	COLOR	BGCOLOR	INITIAL	FINAL
NORTH	WEST	SOUTH	EAST		
PATH	FOREACH	DO	DONE		
IS	COMPLETE	DETERMINISTIC	MINIMIZE	SHOW	

1.2 La commande `CREATE`

Cette commande possède deux syntaxes :
- `CREATE NODE <identifiant> <attributs>`
- `CREATE EDGE FROM <identifiant> TO <identifiant> <attributs>`

La première syntaxe permet de créer un nouveau noeud avec un identifiant. Les attributs possibles sont :
- la position, donnée par `AT <num> <num>`
- un label, donné par `LABEL <label>`
- une couleur donnée par `COLOR <label>`, ainsi qu'une couleur de remplissage donnée par `BGCOLOR <label>`, où le label contiendra une couleur valide en HTML
- une taille, donnée par `SIZE <num>`
- le statut d'état initial, donné par `INITIAL <direction>`
- le statut d'état final, donné par `FINAL <direction>`.

On peut donner autant d'attributs différents que l'on veut, à condition de les séparer par un blanc (espace ou tabulation). Seule la position est obligatoire. La taille par défaut est fixée à 30. Le label par défaut est l'identifiant. Enfin, pour les attributs `INITIAL` et `FINAL`, la direction indique où doit être positionnée la flèche (entrée ou sortie) par rapport au noeud. En absence de direction, on essaiera d'optimiser le placement [2] de la flèche en fonction des arêtes présentes.

La seconde syntaxe permet de créer une transition entre deux noeuds, donnés via leurs identifiants respectifs. Les attributs possibles sont :
- un label, donné par `LABEL <label>` ou par `LABEL <label> AT <num> <num>` pour donner la position du label relativement au milieu du segment reliant les centres des deux noeuds
- une couleur donnée par `COLOR <label>`,
- un chemin donné par `PATH <string>`.

Là aussi, on peut donner plusieurs attributs en les séparants par des blancs. Seul le label est obligatoire. Pour le chemin, la chaîne permettra de donner explicitement le code à utiliser pour faire le rendu de l'arête en `.svg`.

Question 1. Écrivez un analyseur lexical et un analyseur syntaxique afin d'accepter une suite de commandes `CREATE` saisies sur l'entrée standard.

1.3 La commande `DUMP`

Sans argument, cette commande affichera sur la sortie standard un résumé des éléments constituant l'automate en cours de définition. Suivie d'une chaîne de caractères entre `"`, le programme devra produit un rendu en SVG (voir partie suivante) dans le fichier dont le nom est donné par cette chaîne.

2. Une stratégie simple est de mettre toutes les flèches d'entrée à gauche (`WEST`) et toutes les flèches de sortie à droite (`EAST`). Ceci n'est toutefois pas recommandé.

Question 2. Modifiez vos analyseurs lexical et syntaxique afin d'accepter la commande `DUMP`.

Question 3. Implantez le comportement de la commande `CREATE`, puis de la commande `DUMP` appelée sans arguments.

1.4 La commande `REMOVE`

Cette commande possède elle aussi deux syntaxes :
- `REMOVE NODE <identifiant>`
- `REMOVE EDGE FROM <identifiant> TO <identifiant>`

Question 4. Modifiez votre programme afin de gérer la commande `REMOVE`.

note : Comme nous sommes en mode interactif (commandes saisies sur l'entrée standard), on se contentera d'afficher un message d'erreur en cas de suppression invalide.

1.5 La commande `MOVE`

Cette commande possède plusieurs syntaxes :
- `MOVE <num> <num>`
- `MOVE <identifiant> <num> <num>`
- `MOVE [<identifiant>,...,<identifiant>] <num> <num>`

Dans le premier cas, on déplace tous les noeuds. Dans le deuxième cas, seul le noeud avec l'identifiant concerné est déplacé. Dans le dernier cas, tous les noeuds dont l'identifiant figure dans la liste donnée seront déplacés.

À chaque fois, le déplacement est donné sous forme d'un décalage par rapport à la position courante des noeuds. Par exemple, `MOVE 200 100` a pour effet de déplacer les noeuds concernés du 200 unités vers la droite et 100 unités vers le bas.

Question 5. Modifiez votre programme afin de gérer la commande `MOVE`.

1.6 La commande `RENAME`

La syntaxe `RENAME <identifiant> WITH <identifiant>` permet de renommer un noeud.

Question 6. Modifiez votre programme afin de gérer la commande `RENAME`.

1.7 La commande `EDIT`

Comme pour `CREATE`, la commande possède deux syntaxes suivant qu'on modifie un noeud ou une transition :
- pour les noeuds, la syntaxe est `EDIT <identifiant> WITH <attributs>`,
- pour les transition, la syntaxe est `EDIT EDGE FROM <identifiant> TO <identifiant> WITH <attributs>`.

Pour connaître la liste des attributs valides dans chaque cas, on pourra consulter la section sur la commande `CREATE`.

Question 7. Modifiez votre programme afin de gérer la commande `EDIT`.

2 Rendu d'un automate en SVG

Il s'agit en fait dans cette partie d'implanter le comportement de la commande `DUMP` lorsqu'on lui passe le nom d'un fichier. Le coeur du problème est donc de produire un fichier SVG valide à partir de l'automate défini par les commandes précédant `DUMP`.

Pour une petite introduction au format SVG, vous pouvez consulter le site `https://www.w3schools.com/graphics/svg_intro.asp` qui contient de nombreux exemples simples. Un exemple de fichier SVG contenant un automate fini est disponible sur la page `http://web4.ensiie.fr/~christophe.mouilleron/teaching/PROC/example.svg` (je vous invite d'ailleurs à aller voir le contenu du fichier).

Afin de visualiser les fichiers produits, vous pouvez tout simplement les ouvrir dans un navigateur web.

Question 8. Écrivez du code permettant de générer du SVG pour les noeuds.

Question 9. Écrivez du code permettant de générer du SVG pour les transitions.

note : Cette question peut s'avérer assez difficile. Dans un premier temps, concentrez-vous sur une approche simple et fonctionnelle (la qualité visuelle du rendu n'est pas le point important ici).

Question 10. Implantez un support complet pour la commande `DUMP`.

Lors de vos tests, vous arriverez sans doute dans des situations où les transitions s'entremêlent. Sur un automate de taille modeste, la commande `MOVE` devrait permettre de trouver un agencement judicieux des noeuds et donc de limiter ce problème.

3 Quelques améliorations possibles

3.1 Mode non-interactif

Jusqu'à présent, les commandes sont saisies sur l'entrée standard, en mode interactif. Si le programme est lancé avec un nom de fichier en argument, on traitera plutôt ce dernier en mode non-interactif. Ce mode possède les particularités suivantes :

- Les commandes sont lues depuis un fichier, et plus depuis l'entrée standard.
- On peut mettre plusieurs commandes sur une seule ligne, à condition de les séparer par un `;`.
- Désormais, toute erreur (tel la suppression d'un élément non existant) met fin au programme.
- En cas d'erreur, il faudra afficher sur la sortie d'erreur un message contenant le numéro de ligne à laquelle cela s'est produit. Autant que possible, on essaiera aussi de donner des détails sur la nature de l'erreur.

Question 11. Implantez le mode non-interactif.

3.2 Éviter les chemins en dur pour les arêtes

Afin de ne pas avoir à coder le rendu de l'arête complètement en dur lors de l'utilisation de l'attribut `PATH`, on se propose d'ajouter la possibilité d'utiliser dans la chaîne les mots clés `@sx` et `@sy` pour désigner les coordonnées du point de départ, et `@dx` et `@dy` pour désigner les coordonnées du point d'arrivée.

Question 12. Améliorez à votre programme afin que celui-ci calcule ces quatre coordonnées en fonction des centres des noeuds de départ et d'arrivée, et remplace les chaînes de la forme `@xx` par les valeurs ainsi calculées lors de génération du fichier `.svg`.

3.3 Génération par lots [⋆]

Afin de créer plusieurs éléments similaires en une fois, on se propose de mettre en place la syntaxe `FOREACH <valeurs> DO <commande> ; ... ; <commande> DONE`.

Ici, `<valeurs>` sera une suite d'entiers séparés par des virgules, entre les symboles `{` et `}`. On exécutera alors la liste des commandes pour chacun des ces entiers successivement.

Afin d'utiliser l'entier courant au sein des commandes, on va s'autoriser à écrire des calculs. Ces calculs seront entourés des symboles `{` et `}`, et pourront contenir la lettre `x` désignant l'entier courant, des constantes entières ou flottantes, les opérations classiques [3] (+, −, × et % pour le modulo), et des parenthèses pour imposer des priorités. Ces calculs pourront être utilisés à l'intérieur des identifiants (mais ceux-ci doivent toujours commencer par une lettre), à l'intérieur des chaînes (pour les labels notamment) et à la place des constantes numériques. Dans ce dernier cas, c'est le résultat du calcul qui fera office de valeur à utiliser.

Question 13. Implantez et testez cette nouvelle syntaxe.

3.4 Autres améliorations

Outre les améliorations citées ci-dessus, voici une liste (non exhaustive) de points qui nécessitent votre attention :
- la qualité des messages d'erreur,
- la qualité du rendu des transitions en l'absence d'attribut `PATH`,
- l'ajout de commandes ou de nouveaux éléments de syntaxe pour faciliter la vie de l'utilisateur,
- la factorisation d'attributs dans le fichier `.svg` grâce à l'utilisation de la balise `<g>`.

Un travail abouti sur un de ces points, ou tout autre point suffisamment motivé dans votre rapport, donnera lieu à un bonus sur votre note.

4 [LASF] Traitements sur les automates

On rappelle que cette dernière partie contient les questions qui serviront à évaluer le cours LASF. Les questions sont relativement indépendante de ce qui précède, mais il est vivement conseillé d'utiliser le rendu mis en place précédemment pour tester le code produit pour cette partie. Ainsi, des éléments de syntaxes à intégrer dans le projet compilation seront proposés. Parfois, il suffit juste d'appeler la fonction demandée pour gérer cette nouvelle syntaxe. Et parfois, cela demande un peu plus de travail, mais permet en contrepartie de tester et de déboguer beaucoup plus facilement le code produit.

Question 1. Implantez une fonction `is_complete` qui prend un automate en entrée et renvoie un booléen indiquant si cet automate est complet ou non.

note : On pourra ajouter la commande `IS COMPLETE` pour afficher sur la sortie standard `true` ou `false` suivant que l'automate courant est complet ou non, et la commande `SHOW COMPLETE <couleur>` qui change la couleur de fond des noeuds où l'automate n'est pas complet.

Question 2. Implantez une fonction `complete` qui prend en entrée un automate et un nouvel identifiant, et qui retourne l'automate complété (si besoin) avec un noeud supplémentaire ayant l'identifiant passé en argument.

note : On pourra ajouter la commande `COMPLETE WITH <identifiant> AT <num> <num>` pour compléter l'automate courant.

Question 3. Implantez une fonction `is_deterministic` qui prend un automate en entrée et renvoie un booléen indiquant si cet automate est déterministe ou non.

note : On pourra ajouter la commande `IS DETERMINISTIC` pour afficher sur la sortie standard `true` ou `false` suivant que l'automate courant est déterministe ou non, et la commande `SHOW DETERMINISTIC <couleur>` qui change la couleur des flèches qui rendent l'automate non déterministe.

Question 4. Implantez une fonction `is_accepted`, qui sur la donnée d'un automate déterministe et d'un mot (chaîne de caractères), retourne `true` si le mot et reconnu, et `false` sinon.

note : On pourra ajouter la commande `SHOW <string>` qui affiche le chemin (suite de noeuds) parcouru dans l'automate. En bonus, on pourra aussi coder la commande `DUMP <file> WITH <string>` qui produit des animations en `.svg` comme celles que vous pouvez trouver ici (*aaa* accepté) et ici (*abb* refusé).

3. La division a été volontairement omise ici.

Question 5. Implantez une fonction `minimize` qui prend un automate en entrée et renvoie l'automate minimal associé.

note : On pourra ajouter la commande `MINIMIZE` pour minimiser l'automate courant. En cas de fusion entre états, on conservera les attributs de l'état avec l'identifiant le plus petit lexicographiquement. On pourra enfin ajouter la commande `SHOW MINIMIZE` qui se contente d'attribuer des couleurs de fond aux noeuds existants en fonction de leur paquet final dans l'algorithme de minimisation.