ECOLE NATIONALE SUPÉRIEUR D'INFORMATIQUE POUR L'INDUSTRIE ET L'ENTREPRISE

Rapport du Projet : PROC-LASF 2020-2021

Akarioh Samir

25 mai 2021

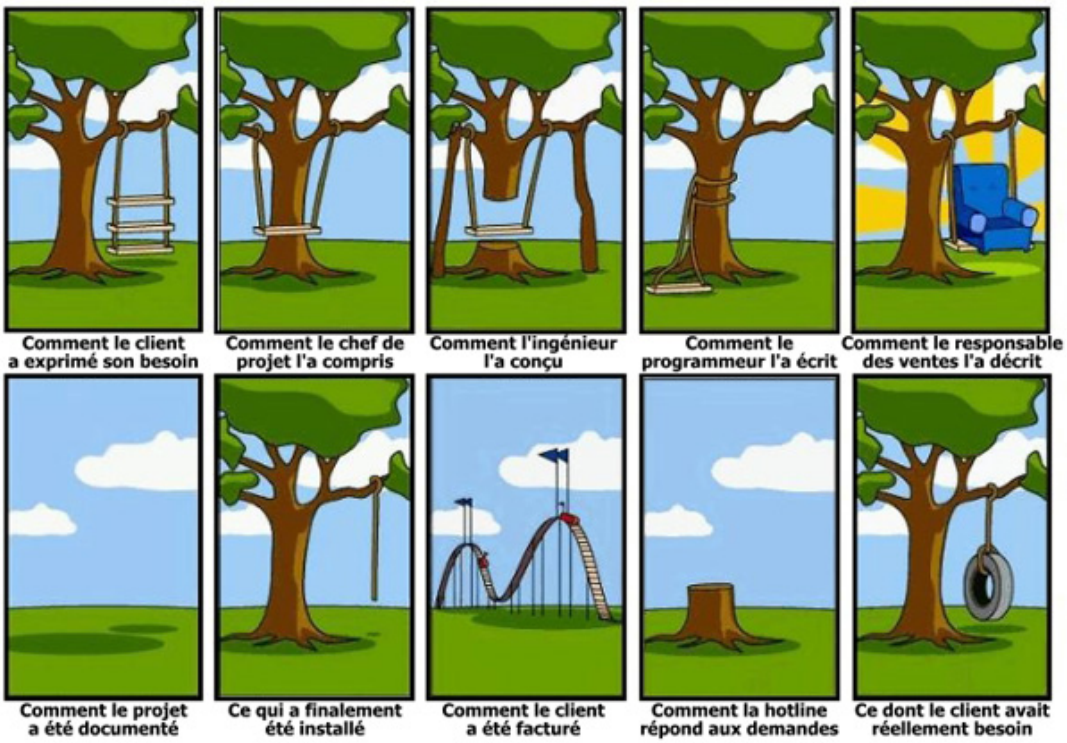


FIGURE 0.1 – Il faut avoir un cahier des charges

TABLE DES MATIÈRES

1 INSTALLATION

Pour compiler mon projet, il faut avoir la dernière version d'ocaml et celle d'ocamllex et ocamllyacc.

Pour compiler le programme, il faut se placer dans le dossier ProjetOcaml et lancer le script make.sh. Il va vous générer un exécutable qui s'appelle rendu.

Pour lancer les tests placer vous dans le dossier ProjetOcaml et lancer le script test.sh

Pour lancer les scripts, il faut lui donner les permissions via un chmod.

2 CHOIX TECHNOLOGIQUES

2.1 LANGAGE DE PROGRAMMATION

J'ai choisi de faire ce projet via le langage de programmation Ocaml, en utilisant Ocamllex et Ocamllay. La raison de ce choix est qu'Ocaml permet facilement de représenter des automates et je peux me concentrer sur la grammaire et mon code sans me soucier de la gestion de mémoire.

Je peux ainsi parcourir facilement ma liste de nœud sans me soucier des pointeurs par exemple si je dois supprimer un nœud, cela me permet d'éviter de me soucier des erreurs de segmentation fault.

2.2 TYPES UTILISÉS POUR LE PROJET

— arbre :

```
1 type 'a arbre =
2   |Noeud of string * string *  string * string
3   |Edge of string * string * string * string ;;
```

Un Noeud est composé de 4 arguments qui sont dans l'ordre :
— L'ID
— La position X
— La position Y
— Les arguments associés

Un Edge est aussi composé de 4 arguments qui sont :
— L'ID de départ
— l'ID d'arrivé
— Le label qui est composé d'une seule lettre

— Les arguments associés

‘

Par la suite, un automate est composé d’une liste de nœuds et une liste de transition séparé. Pour toutes les commandes l’id est tronqué à 15 caractères dans le fichier functionannexe. Même si l’utilisateur rentre un id de plus de 15 caractères, on récupère que les 15 premiers.

3 PRÉCISION SUR CERTAINES COMMANDES

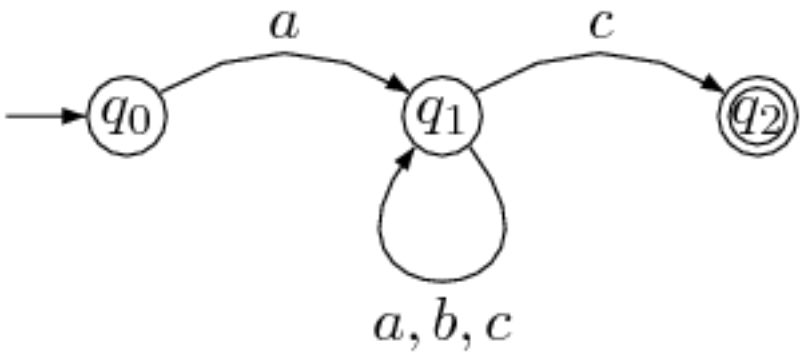
3.1 CREATE

3.1.1 ETAT INITIAL ET FINAL

Si l'utilisateur ne donne pas de direction, j'ai mis en place une méthode très classique qui n'est pas la meilleure. La méthode est de mettre les flèches initiales à gauche et celle final à droite.

3.1.2 LABEL EDGE

Une transition est égale à un label à une lettre par exemple si nous avons le schéma suivant :



Pour le sommet q1 afin d'avoir une flèche composée de a b c, il faut créer une transition par lettre. La liste de transition pour juste le sommet q1 sera :

— arbre :

```
1  [ Edge("q1","q1","a",argument);
2    Edge("q1","q1","b",argument);
3    Edge("q1","q1","c",argument) ]
```

3.1.3 ARGUMENTS OBLIGATOIRE

Afin de gérer les cas des arguments obligatoires, j'ai mis en place dans ma grammaire 4 règles avec les 4 positionnements possible de label/label at mais aussi de la commande AT cela me permet de facilement gérer les arguments obligatoires.

3.2 REMOVE

Lorsque qu'on supprime une transition, mon programme supprime toutes les transitions possibles entre les deux nœuds qu'importe le label.

3.3 MOVE

3.3.1 MOVE LISTE

Afin de récupérer les éléments de la liste, j'ai mis en place une regex qui récupérer la liste sous format str voici la regex

```
['['][a-z A-Z 0-9 _ ,]*[']']
```

Pour transformer mon str en liste,dans le fichier y, je supprime le [et le] puis je crée une liste via la fonction python-split.

3.4 EDIT

3.4.1 EDGE

Si l'utilisateur utilise la commande de base, notre programme lui spécifie qu'il doit nous donner un label. Maintenant pour edit un EDGE il faut lui préciser le label associé à la transition via la commande.

```
EDITEDGE ID TO ID LABEL vrailabel WITH attributet
```

3.5 DUMP SVG

3.5.1 TRANSITION

Pour faire mes paths svg, j'ai d'abord utilisé la lettre A, cependant lorsque les nœuds étaient assez éloignés les transitions était assez compliqué à comprendre, c'est pourquoi je suis passé à la lettre Q.

Pour les directions NE-NO-SE-SO, les transitions et la position des labels ne sont pas optimales mais suffisantes pour avoir un rendu de qualité.

Afin de mettre en place les transitions, je calcule de quel point la transition doit partir et arrivé via les coordonnées des deux points afin de connaître la direction de la transition.

3.6 Mode Non interactif

3.6.1 Méthode

Maintenant, les commandes sont lues via un fichier, afin de pouvoir le lire, je fais un prétraitement sur le fichier. Cependant, je n'affiche pas encore le numéro de la ligne.

— Je donne le fichier à une fonction
— La fonction lit chaque ligne et split les lignes contenant un ;
— On rajoute les lignes dans une liste
— Je crée un fichier temporaire sortie.txt avec les ; changé en saut de ligne.
— je lance le parseur avec le fichier sortie.txt
— Lorsque le programme est fini ou en cas d'erreur je supprime le fichier sortie.txt

3.6.2 Erreur

En cas d'erreur, id qui n'existe pas, transition/nœud déjà existante, on fait comme pour le mode non-interactif, on stoppe le programme.

3.7 Autre amélioration

3.7.1 Rendu des transitions

Voir la partie DUMP SVG, j'ai essayé de base d'avoir des transitions assez propres si jamais je n'avais pas l'attribut PATH.

3.7.2 Ajout de Commande

Dans le programme, j'ai rajouté la commande DELETE ALL qui permet de supprimer tous les éléments de la liste de nœuds et transitions.

J'ai aussi mis en place la commande transpose qui permet à partir d'un automate A d'avoir l'automate transposé de A et donc qui reconnaît les miroirs des mots reconnu par A.

La première commande me permet de faire plusieurs tests dans le même fichier txt. Pour la deuxième commande, je l'ai mis en place pour la question 5 de LASF cependant, je n'ai pas réussi à finir la fonction de minimisation.

4 LASF

4.1 IS COMPLETE

La fonction que j'ai mise en place fonctionne ainsi :

j'itère sur ma liste de nœud, pour chaque nœud je regarde si il existe une transition pour chaque lettre de ma grammaire, si cela est faux pour un nœud alors je retourne faux. À la fin de ma fonction si la liste de nœud devient vide alors je renvoie vrai.

Les deux commandes spécifié dans la note de la question ont été mise en place dans le fichier l'et y.

4.2 COMPLETE

La fonction que j'ai mise en place fonctionne ainsi :

pour chaque nœud, je vérifie s'il existe pour chaque lettre de ma grammaire une transition si cela n'est pas le cas alors je crée une transition qui va du nœud au nouveau nœud crée (état puits) puis je termine par rajouter les transitions allant de l'état puit à lui-même pour toutes les lettres. Pour finir, je rajoute l'état puit à ma liste de nœuds.

Si l'automate est déjà complet, je ne le modifie pas. L'automate avec un nœud et sans transition, n'est pas complet pour moi.

La commande spécifiée dans la note de la question a été mise en place dans le fichier l et y.

4.3 IS DETERMINISTIC

La fonction que j'ai mise en place fonctionne ainsi :

on vérifie d'abord le nombre d'états initiaux, s'il est égal à 1, on continue la fonction sinon on renvoie false. J'itère ensuite sur ma liste de nœuds, pour chaque nœud, je regarde s'il existe une et une seule transition pour chaque lettre de ma grammaire, si cela est faux pour un nœud alors je retourne false. À la fin de ma fonction si la liste de nœuds devient vide alors je renvoie true.

Les deux commandes spécifié dans la note de la question ont été mise en place dans le fichier l et y.

4.4 IS ACCEPTED

Je me mets au sommet initial de mon automate, par la suite, je me déplace de nœud en nœud en lisant une lettre par une lettre mon mot. Si à la fin de mon mot, je suis sur un état final alors je retourne vrai sinon je retourne faux.

Les deux commandes spécifié dans la note de la question ont été mise en place dans le fichier l et y.

Pour SHOW, string, j'ai fait une fonction annexe qui se base sur le même pseudo-code sauf que je sauvegarde l'id des nœuds dans un str et à la fin de la fonction, je retourne ce str.

Pour DUMP WITH string, j'utilise la fonction chemin afin de créer petit à petit mes animations SVG.

4.5 MINIMIZE

Je n'ai pas mis en place cette question dans mon code final, mais j'avais essayé de la faire.

Pour l'algorithme de Moore, j'ai réussi à initialiser mon dictionnaire avant l'algo en séparant les états finaux et le reste cependant, je n'ai pas réussi à mettre en place la suite de l'algorithme.

Pour l'algorithme de Brzozowski, j'ai réussi à mettre en place une fonction qui transpose mon automate cependant, je n'ai pas réussi à mettre en place une fonction qui détermisme un automate, je n'ai donc pas pu terminer cette fonction.

5 PROBLÉMES RECONTRÉS

Au début du projet, j'avais un bug, mes listes de nœuds et de transitions était vidé en chaque commande. La réponse m'a été donnée par Monsieur Mouilleron qui était de mettre ces variables en mode ref.

Pour l'algorithme de minimisation, j'ai eu les problèmes spécifié dans MINIMIZE cependant, je n'ai pas pu le résoudre.

Pour les transitions en utilisant la lettre A dans path j'avais des problèmes si jamais les nœuds étaient trop éloignés. Pour régler ce problème, j'ai utilisé la lettre Q pour les paths.