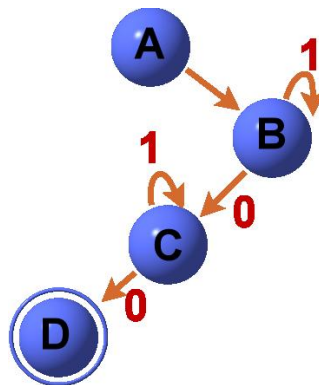




Université de Franche-Comté  
Licence 3 Informatique

# Animation d'algorithmes sur les automates d'états finis



Alexandre BAILLY  
Sylvain RELANGE

Responsable de projet: Olga KOUCHNARENKO

Année 2009-2010

# Sommaire

Introduction	1
1 Présentation du sujet	2
2 Analyse de l'existant	3
3 Mise en œuvre	5
4 Problèmes rencontrés	12
Conclusion	13
Bibliographie	15
Annexe	16

---

## Introduction

Dans le cadre de notre 3ème année de licence, nous sommes amenés à réaliser un projet tuteuré en binôme. Ce projet a pour but de développer un logiciel permettant d'animer des algorithmes sur les automates d'états finis.

En effet, il existe de multiples opérations sur les automates, allant de la détermination à l'union, en passant par le produit. Leur complexité n'est généralement pas ardue, mais un exemple est souvent le meilleur moyen de comprendre.

De ce fait, dans le cadre de l'enseignement à distance, et même présentiel, un outil détaillant ces algorithmes pourrait donner un coup de pouce à leur compréhension.

Afin de rendre compte de notre travail sur ce sujet, ce rapport comporte :

- Une étude des différents outils existants déjà.
- La présentation détaillée ainsi que les objectifs fixés.
- La description de la mise en œuvre.
- Les problèmes rencontrés durant notre travail.
- Le bilan de ce projet ainsi que les limites de notre application.

## 1 Présentation du sujet

Nous devons donc réaliser un outil capable d'animer de façon détaillée différents algorithmes d'automates d'états finis.

De plus, cet outil devait permettre une manipulation conviviale par le biais d'une interface simple et pratique. Il devait également être conçu et développé dans l'optique d'une évolution et d'ajout de fonctionnalités aisés.

Ensuite, il nous était demandé de rechercher et étudier les logiciels et outils de manipulation d'automates déjà à la disposition du public afin de déterminer si ceux-ci pouvaient nous être utiles. Cette étude avait pour but également d'éviter de recréer quelque chose d'existant si cela n'apportait rien.

Enfin, au niveau algorithme de manipulation, notre responsable de projet a souhaité que l'outil à développer puisse détailler pas à pas au minimum deux algorithmes :

- L'un prenant en entrée deux automates d'états finis.
- L'autre prenant un seul automate d'états finis.

L'outil devait en outre permettre à l'utilisateur d'aller en arrière et en avant à volonté.

En résumé, au départ de notre projet, nous avons les objectifs suivants, fixés par notre responsable et nous-même :

- Réaliser une interface graphique de création d'automate simple.
- Réaliser une interface graphique de manipulation des algorithmes intuitives.
- Réaliser la sauvegarde d'automates créés par l'outil.
- Charger des automates à partir d'un fichier.
- Concevoir un programme dans l'optique qu'il puisse évoluer après notre contribution.
- Implémenter l'animation pas à pas et détaillée de l'algorithme du produit d'automate d'états finis.
- Implémenter l'animation pas à pas et détaillée de l'algorithme de détermination d'automate d'états finis.
- Avoir un outil pouvant fonctionner à la fois sur Windows et GNU/Linux

## 2 Analyse de l'existant

Nous avons effectué des recherches pour voir quels logiciels permettant de manipuler des automates d'états finis existaient déjà.

En recherchant séparément, nous sommes arrivés à la même conclusion : le seul logiciel facile d'accès, disponible sur Internet, est JFLAP. Il existe également le logiciel AnimAlgo, créé par le binôme ayant travaillé sur le même sujet que nous l'année précédente.

Nous nous sommes donc intéressés aux fonctionnalités de ces deux logiciels.

### 2.1 AnimAlgo

AnimAlgo est le nom de projet choisi par les étudiants ayant eu le même sujet que nous l'année précédente. Nous avons eu le choix, au commencement de notre projet, entre reprendre leurs travaux, ou recommencer depuis le début avec nos méthodes.

AnimAlgo est donc développé dans le langage Java, en utilisant la bibliothèque graphique Swing pour réaliser l'interface utilisateur. Le projet se sert également du package Grappa, utilisant le software Graphviz, pour générer automatiquement une image d'automate à partir d'un fichier .DOT. Cette image est ensuite exploitée grâce à la bibliothèque Swing.

AnimAlgo possède une interface graphique simple et efficace, et permet certaines opérations de base sur les automates :

- La création en ligne de commande d'automate.
- la visualisation d'automate.
- la visualisation étape par étape d'une détermination d'un automate d'états finis.
- Sauvegarder l'automate en cours.

Cependant, le projet connaît certaines limites au niveau des sauvegardes et de la portabilité.

En effet, AnimAlgo sauvegarde toujours l'automate créé dans le dossier par défaut, quelque soit le chemin indiqué.

Au niveau de la portabilité, le logiciel semble connaître des problèmes pour trouver le chemin d'installation de Graphviz nécessaire au fonctionnement de l'application. De plus, sous GNU/Linux, il arrive que le programme rencontre une erreur nuisant au bon fonctionnement d'AnimAlgo.

## 2.2 JFLAP

JFLAP est un software développé par l'université de Duke, aux Etats-Unis. Son développement est toujours assuré, et la dernière version du logiciel date du 28 septembre 2009. Ce logiciel est également développé en Java.

C'est un outil très performant, qui permet de manipuler différents types d'automates et de grammaires, allant de l'automate d'états finis à la machine de Turing :



FIGURE 1 – Exemple de différents automates et grammaires manipulables avec JFLAP

Ce logiciel a également l'avantage d'être très facile d'utilisation , puisqu'il suffit de cliquer sur un unique fichier "JFLAP.jar" pour l'exécuter. Il est également portable sous Linux et Windows, pour peu que la personne ait Java d'installé sur son ordinateur.

Au niveau de la manipulation d'automate d'états finis, JFLAP permet de :

- Créer des automates par interface graphique.
- Déterminiser, étape par étape si souhaité.
- Minimiser, étape par étape si souhaité.
- Réaliser la complétion de l'automate.
- Générer la grammaire définis par l'automate.
- Sauvegarder des automates.

Cependant, JFLAP n'est pas forcément adapté en tant qu'outil pédagogique à un public francophone néophyte dans le domaine des automates et de leurs algorithmes.

En effet, JFLAP, bien que développant pas à pas certains algorithmes, ne commente pas la façon dont il les applique. De plus, la multiplicité des options peut perdre l'utilisateur débutant.

Pour finir, JFLAP n'implémente que très peu d'algorithmes de manipulation prenant en entrée deux automates ou grammaires. Par exemple, il ne permet pas de réaliser le produit de deux automates d'états finis, ni l'intersection de deux grammaires (algorithme correspondant au produit).

## 3 Mise en œuvre

### 3.1 Organisation et partage des tâches

Afin de réaliser ce projet en binôme, nous avons dû nous organiser afin d'être le plus efficace possible, et pour ne pas se gêner lors du développement.

Une fois le langage choisi et le diagramme UML réalisé, l'un s'est chargé de développer la partie graphique, tandis que l'autre s'est chargé de l'implémentation des algorithmes de manipulation d'automates. Nous restions tout de même en contact lors du développement via différents logiciels de communication, afin de s'entraider et de discuter de certaines idées à réaliser.

Au niveau de la méthode de travail, au début du codage de l'application, nous ne travaillions pas en même temps, donc on s'échangeait les fichiers uniquement par e-mail. Mais nous avons rapidement mis en place ensuite un dépôt SVN (cf 4.3 Outils utilisés) lorsque nous avons commencé à travailler simultanément.

Pour finir nous avons développé simultanément sous GNU/Linux et Windows, afin de nous assurer du bon fonctionnement de notre projet sous ces deux systèmes d'exploitation.

### 3.2 Conception

Lors de la conception de notre projet, il nous est vite apparu qu'afin de faciliter la manipulation des automates et de pouvoir développer des algorithmes les concernant, nous devons impérativement séparer le fond ( les automates ) de la forme ( l'interface graphique QT ).

Afin de garder une limite claire entre ces deux parties, et également de permettre l'éventuelle réutilisation de nos algorithmes, nous nous sommes imposés une règle simple : nos classes concernant les automates ne doivent pas implémenter de classes QT. Elles doivent être facilement transportables d'un programme à un autre si nécessaire.

Afin de représenter des automates, nous avons choisi de créer deux classes : **Automate** et **Etat** :

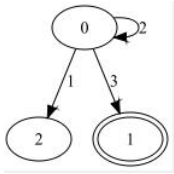
- La classe **Automate** : contient comme attribut une liste d'états, ainsi que des méthodes implémentant les algorithmes de manipulation d'automates.
- La classe **Etat** : contient comme attribut un entier correspondant au numéro de l'état dans l'automate, ainsi que les transitions de l'état. Les transitions sont représentées par une **multimap** (**etat**,**entier**), où chaque couple **etat/entier** correspond à l'état pointé par une transition par l'entier. La classe possède également deux booléens, pour savoir si l'état est initial, ou final. La classe contient aussi un chaîne de caractères, pour indiquer si l'état possède un nom différent que son entier à afficher.

```
private:
    string name;
    bool initial;
    bool final;
    multimap<int,etat> transition;
};
```

Nous souhaitions également une interface graphique la plus simple et intuitive possible. Afin de réaliser cette interface, nous avons essayé de limiter le nombre d'informations, de boutons et d'options affichés à l'écran.

Puisque la création d'un jeu d'automate est la première étape de tout utilisateur, et qu'il peut être nécessaire d'en créer plusieurs pour mieux cerner les algorithmes, l'outil de création devait être spécialement intuitif pour ne pas fatiguer l'utilisateur avant même que celui-ci ne commence la compréhension des manipulations.

Nous avons choisi de commencer notre projet par le développement d'un créateur d'automate, car celui-ci permettrait de tester les fonctions bas-niveau (ajout d'états, ajout de transitions, suppression d'états...) de nos automates rapidement. De plus il permettrait par la suite de créer facilement des automates pour tester nos fonctions de plus haut niveau (déterminisation, produit d'automates).



Un autre enjeu important de la conception était de déterminer comment afficher une représentation de nos automates à l'écran. Pour cela, nous avons suivi les conseils de notre responsable de projet, et adopté **Graphviz**. Ce software permet, entre autre, de créer une image d'un automate enregistré sous forme d'un fichier .dot . L'image créée est par défaut une image svg.

FIGURE 2 – Un automate dessiné par Graphviz

### 3.3 Langages et Outils utilisés

#### 3.3.1 C++

Nous avons choisi de programmer en C++, afin d'avoir une exécution la plus rapide possible lors de l'utilisation des algorithmes de notre programme.

De plus, nous avons tout les deux déjà eu une formation sur ce langage lors de notre DUT.

Il nous a également permis de mettre en application directe nos cours du moment, notamment avec l'utilisation de la bibliothèque STL et de ses vecteurs, multimaps...

#### 3.3.2 Framework Qt

Nous avons choisi d'utiliser QT4 comme bibliothèque graphique, car QT contient énormément de classes spécifiques, dont une classe `QSvgWidget` permettant d'afficher une image svg, et une classe `QProcess` permettant de lancer un processus externe.



Dans notre programme, nous utilisons cette classe pour exécuter Graphviz.

De plus, nous avons déjà eu des cours d'initiation à QT, en python et en C++ lors de notre DUT. Grâce à la documentation de QT, nous nous assurons de trouver rapidement des réponses fiables à nos éventuels problèmes d'interface.

#### 3.3.3 QtCréator



Nous avons choisi de programmer avec l'EDI (Environnement de développement intégré) Qt-Creator.

QtCreator est un EDI regroupant plusieurs outils :

- **Qt designer** : Qt Designer permet de créer facilement des interfaces QT standards, qui sont ensuite personnalisables dans le code.
- **Un éditeur de code** : Qt Creator contient un puissant éditeur de code, qui possède toutes les fonctionnalités que l'on peut souhaiter : indentation, complétion du code très puissante, navigation aisée entre les headers et les fichiers cpp...
- **La documentation QT** : La documentation Qt est incluse, et est accessible très facilement, avec accès direct aux classes concernées.
- **Un debugger** : un debugger qui permet de placer des balises de pause n'importe où dans le code et d'avoir accès aux valeurs des différentes variables à un moment précis...

#### 3.3.4 Subversion

Afin de mettre en commun nos codes rapidement, nous avons utilisé l'outil **Subversion**, en créant un dépôt subversion grâce au service google `googlecode.com`.



La mise en place de ce dépôt avait une certaine importance. En effet, nous avons besoin de mettre nos codes en commun souvent, pour vérifier que le code fait par l'un fonctionnait bien sous le système d'exploitation de l'autre (Windows/Linux) et ainsi se rendre rapidement compte des erreurs le cas échéant.



### 3.3.5 Graphviz



Graphviz est un software sous licence publique disponible sur les plateformes Unix, Windows et Mac. Il permet de créer une représentation graphique de toutes sortes d'automates, schémas, graphiques...

Nous utilisons Graphviz pour générer les images de nos automates à partir de fichier dot. Cet outil peut facilement être trouvé sur le net. Néanmoins, nous joindrons au fichier d'installation de notre programme (pour la version Windows) l'exécutable permettant de l'installer.

## 3.4 Développement de l'interface

Pour coder l'interface graphique, nous avons donc utilisé QtCreator, qui permet de faciliter le design d'une interface. L'utilisation de cet outil a, dans un premier temps, nécessité un temps de compréhension du logiciel, mais nous a vraiment fait gagner du temps par la suite.

Cependant, si cet outil permet de faire des interfaces statiques personnalisées facilement, la gestion de la dynamique de la fenêtre doit toujours être gérée dans le code, c'est pourquoi l'affichage, la disparition et l'ajustage des fenêtres à donc été à faire dans le code.

```
maVue1 = new QSvgWidget(ui->vue1);
maVue2 = new QSvgWidget(ui->vue2);
maVue = new QSvgWidget(ui->vueAutomate);

ui->boutonSuiv->hide();
ui->boutonPrec->hide();
ui->scrollArea_2->hide();
ui->scrollArea_3->hide();
ui->label->hide();
connect(ui->boutonPrec,SIGNAL(clicked()),this, SLOT(getPrecedent()));
connect(ui->boutonSuiv,SIGNAL(clicked()),this, SLOT(getSuivant()));
```

De plus, l'utilisation de classes bien spécifiques doit toujours être réalisée "à la main". Par exemple la classe `QSvgWidget`, qui permet l'affichage d'images svg, n'est pas incluse dans l'éditeur d'interface de **Qtcreator**.

### 3.4.1 Manipulation d'automates par l'interface

Pour manipuler les automates, certaines classes d'interface contiennent des automates. Par exemple, la classe de la fenêtre principale du créateur d'automate contient un automate, qui se construit au fur et à mesure. Les classes filles contiennent des pointeurs vers cet automate pour pouvoir le modifier.

De même, la classe `etatRight` (qui permet l'affichage à droite d'un état sélectionné à gauche) contient une fonction qui permet d'effacer des transitions directement dans l'automate de la fenêtre globale. Cette fonction envoie ensuite un signal à la classe mère, indiquant que l'affichage de l'automate doit être mis à jour.

```
void etatRight::eraseTransition(int to,int vocab){
    a->getEtat(numero)->supprimeTransition(*a->getEtat(to),vocab);
    refreshNeeded(numero);
}
```

FIGURE 4 – Suppression d'une transition

### 3.4.2 Affichage d'un automate

Certaines classes d'interface graphique créées contiennent des fonctions permettant l'affichage d'automate.

Ces fonctions appellent la fonction `toDot()` d'un automate, qui retourne un `string` correspondant au fichier `.dot` d'un automate. Pour créer une image svg, Graphviz a besoin de s'exécuter sur un fichier, ce `string` de sortie est donc sauvegardé dans un fichier `.dot`.

```
QFile tmp("./tmp.dot");
tmp.open(QFile::WriteOnly);

QTextStream out(&tmp);
out << QString().fromStdString(a.toDot());
tmp.close();
```

FIGURE 5 – Sauvegarde d'un automate dans un fichier `.dot`

```

procargs.push_back("-Tsvg");
procargs.push_back("./tmp.dot");

ProcessI.start("dot",procargs);
ProcessI.waitForFinished();

maVue1->load(ProcessI.readAll());

```

FIGURE 6 – Utilisation de Graphviz

La fonction exécute le software Graphviz sur le fichier dot créé. La sortie de cette exécution n'est pas sauvegardée dans un fichier .svg, elle est directement chargée par les `QSvgWidget`, grâce à la fonction `load`. Une fois ces images chargées, la taille des différents items doit être modifiée, pour coller à la taille de l'image chargée, ou pour afficher les ascenseurs verticaux ou horizontaux si l'image est trop grande.

À noter également que la classe `MainWindow`, correspondant à la fenêtre principale, contient un vecteur d'automates, éventuellement vide, représentant chaque étape d'un produit d'automates.

La classe contient aussi un vecteur d'automate et d'état, correspondant à chaque étape d'une détermination, et au commentaire associé à cette étape.

Lorsque l'on a choisi de déterminer un automate, des boutons "suivant" et "précédent" apparaissent et permettent de naviguer d'étape en étape.

```

vector<Automate> monVector;
vector< pair< Automate , string > > monDeterminisme;

```

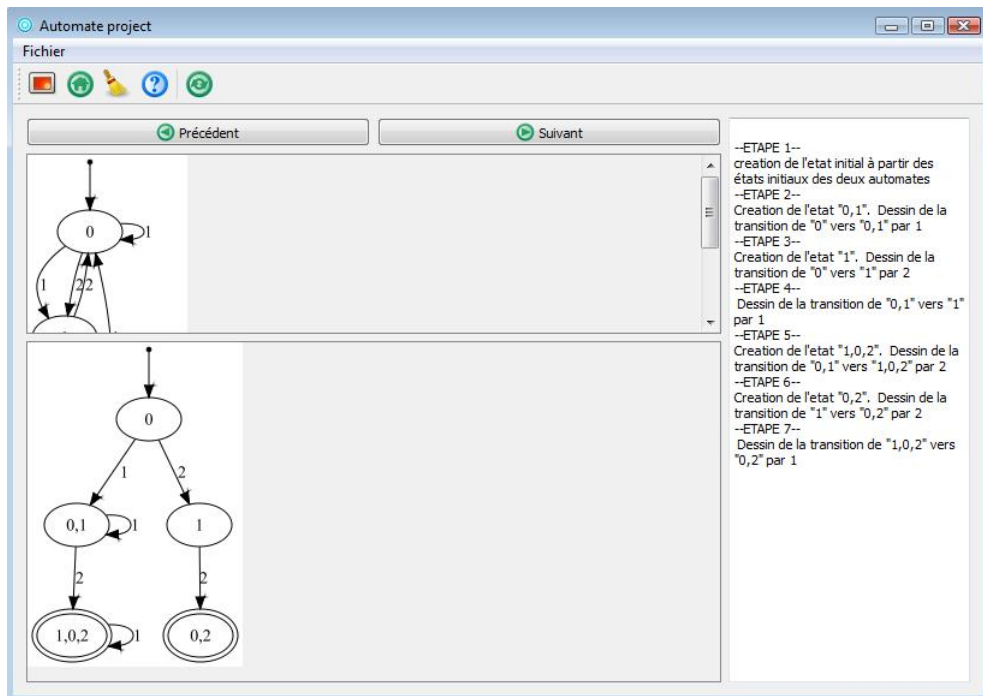

FIGURE 7 – Un extrait de `MainWindow.h`

FIGURE 8 – Interface de la détermination

### 3.4.3 L'utilisation de l'interface

Dans l'interface graphique, les différentes actions disponibles par l'utilisateur suivent la norme la plus souvent utilisée par les logiciels.

Elles sont disponibles de trois manières :

- **Sur la barre d'outils située en haut** : La barre d'outils () est repositionnable à volonté, si l'utilisateur souhaite la déplacer.

Si l'on a chargé un automate, de nouvelles options apparaissent sur la barre d'outils :

Si l'on a choisi de déterminer un automate, le bouton détermination devient donc inutile, et s'efface automatiquement. Le même système fonctionne si l'on choisit de faire le produit.



- **depuis le menu en haut de la fenêtre** :



- **depuis les raccourcis indiqués dans le menu** : les boutons "suivant" et "précédent" fonctionnent également avec le clavier, avec les flèches de droite et gauche.

Si l'on survole les icônes de la barre d'outils, une information les concernant, et permettant leur compréhension apparaît.

Pour faciliter la compréhension du logiciel, un bouton d'aide est disponible : il s'agit de l'icône  de la barre d'outils. Si l'utilisateur clique dessus, une aide apparaît. Elle disparaît ensuite si l'utilisateur re-clique sur l'icône .

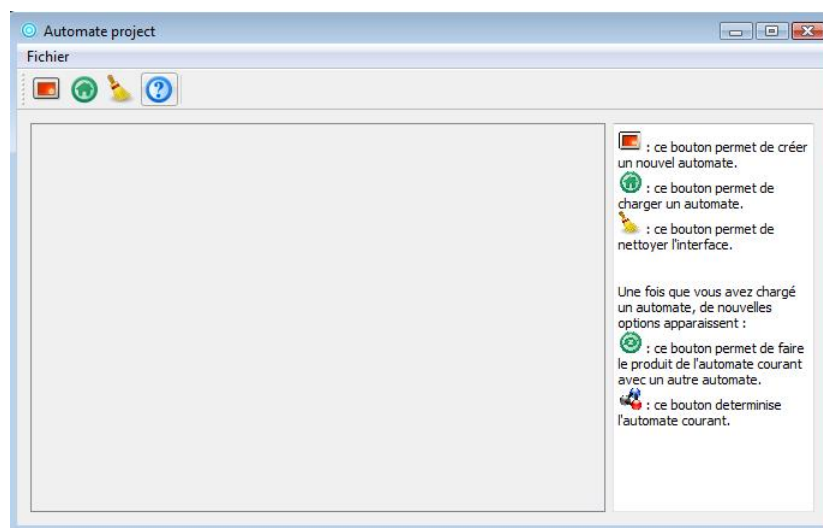


FIGURE 9 – Affichage de l'aide à l'écran

### 3.5 Implémentation des algorithmes de manipulation d'automate

#### 3.5.1 Produit d'automates

L'un des principaux objectifs du projet était d'avoir la possibilité de réaliser pas à pas le produit de deux automates donnés en entrée : c'est ce que réalise la fonction `produit` de la classe `Automate`.

Dans ce but, l'algorithme implémenté va, à partir de deux objets `Automate`, renvoyer un vecteur d'automate. Chaque automate va représenter une étape du processus d'exécution du produit. La structure de donnée dynamique choisi pour stocker ces étapes est un `vector` issu de la bibliothèque **STL**. Cette structure va permettre de parcourir la liste d'étapes dans un sens ou dans l'autre, et également de pouvoir accéder directement au  $i$ ème élément de cette liste.

Ensuite, on garde en mémoire de quels états des automates de départ sont issus les états de l'automate résultat. On utilise alors une `map` constituée d'un entier (servant de clé), et d'une `pair` d'états. L'entier correspond à l'état de l'automate résultat, et la `pair` aux deux états de l'automate de départ.

Pour finir, voici le fonctionnement de l'algorithme :

1. On parcourt chaque état du premier automate à l'aide d'une boucle.
2. Dans cette boucle, on parcourt ceux du second automate.
3. Si c'est la première étape, on crée un nouvel automate, sinon on recopie le précédent dans un nouvel automate représentant cette étape.
4. On crée un nouvel état  $p * m + q$  ( $p$  est l'état en cours du premier automate,  $q$  celui du second, et  $m$  est le produit des tailles des deux automates initiaux). Lors de la création de cet état, on indique si il doit être final et/ou initial.
5. On le nomme de façon explicite, par exemple l'état 1 sera nommé "0,1". Puis on l'ajoute à l'automate de l'étape en cours.
6. On ajoute également dans la `map` de correspondance la `pair` d'état constituant l'état créé.
7. On recherche si l'état nouvellement créé doit avoir des transitions conduisant à d'autres états déjà créés de l'automate-étape, auquel cas on ajoute ces transitions.
8. On recherche si il existe des transitions dans l'automate-étape qui conduisent à l'état nouvellement créé, auquel cas on ajoute ces transitions.
9. On ajoute l'automate en queue du vecteur d'automate-étape et on retourne au point 3 en passant à l'état suivant du second automate.
10. Quand tous les états du second automate ont été traités pour chacun des états du premier automate, on renvoie notre vecteur d'automate-étape.

#### 3.5.2 Déterminisation

Un autre objectif de notre projet était de réaliser la déterminisation pas à pas d'un automate d'états finis.

Pour cela, nous avons implémenté une fonction `determinise` qui renvoie un `vector` de `pair`, constitué d'un `Automate` et d'une chaîne de caractères. Chaque automate va correspondre à une étape, et la chaîne de caractères à un commentaire sur cette étape. Nous avons choisi la structure de données `vector` pour les mêmes raisons que précédemment.

Ensuite, pour stocker la correspondance entre un état de l'automate déterminisé et les états de l'automate de départ le constituant, on se sert d'une `map` constituée d'un entier (servant de clé), et d'une `list` d'états. En effet, un état déterminisé peut très bien contenir tous les états de l'automate initial ou seulement un seul, d'où l'utilisation de la liste d'états.

De plus, il a fallu développer certaines fonctions spécifiques, uniquement utilisées pour réaliser la déterminisation. Par exemple, l'alphabet d'un automate n'est pas stocké dans la classe `Automate`. Or il nous était nécessaire de parcourir l'alphabet dans l'algorithme de déterminisation, donc de le connaître. Pour remédier à cela, on a codé une fonction `getAlpha` renvoyant un vecteur contenant l'alphabet.

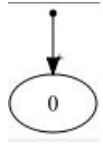
L'algorithme de détermination fonctionne de la façon suivante :

1. On récupère l'alphabet de l'automate à déterminer.
2. On recherche les états initiaux de l'automate de départ afin de créer l'état initial déterminé. On indique en même temps si cet état est final ou non.
3. On renomme l'état initial correctement (ex : "0,2,3") avant de l'insérer dans l'automate vide nouvellement créé. On ajoute dans la map, à 0, la liste de correspondance entre les états de l'automate initial et celui de l'automate déterminé.
4. On stocke une copie de cette automate dans le vecteur à renvoyer.
5. On parcourt chaque état de l'automate déterminé à l'aide la map de correspondance et d'une boucle.
6. Dans cette boucle, on parcourt l'alphabet à l'aide d'une autre boucle.
7. Pour chaque état figurant dans la liste de correspondance en cours, on cherche si il possède une transition par l'élément de l'alphabet en cours. Si c'est le cas, on stocke dans une liste l'état pointé en évitant les doublons.
8. On vérifie si la liste créée correspond à un état déterminé existant déjà, à l'aide la map de correspondance.
  - Si oui, alors on ajoute une transition depuis l'état courant vers l'état correspondant à cette liste, par la lettre courante.
  - Si non, on crée un nouvel état, on ajoute en queue la liste dans la map de correspondance, et on ajoute la transition qui pointe sur cette état depuis l'état courant, par l'élément courant de l'alphabet.
9. On stocke une copie de l'automate en cours de détermination dans le vecteur à renvoyer.
10. Quand toute la liste de correspondance a été parcourue pour l'élément de l'alphabet , on passe à l'élément suivant.
11. Quand tout l'alphabet a été parcouru pour l'état déterminé, on passe au suivant stocké dans la map.
12. Quand on est arrivé au bout de la map de correspondance, on renvoie le vecteur contenant chaque étape de la détermination.

## 4 Problèmes rencontrés

### 4.1 Respect de la convention de dessin d'un automate d'états finis

Un des problèmes rencontrés a été la création d'une fonction dans la classe automate, permettant de traduire l'automate en fichier dot.



La principale difficulté est due au fait que Graphviz ne contient pas d'option permettant de dessiner un état initial selon la convention habituelle. Pour palier à ce problème, les fichiers dot que nous utilisons pour représenter nos automates doivent créer un état fictif, nommé par convention `F[numero-de-l'état-initial]`.

Nous donnons à cet état la forme d'un point, et lui attribuons une transition vers l'état initial. De cette manière, l'état initial a une flèche pointant sur lui.

Un problème, induit par cette manipulation, est que la fonction de lecture d'automate, permettant de charger un automate à partir d'un fichier dot, doit comprendre que l'état `F0` n'existe pas, et que l'état `0` est initial.

```
F0[shape=point] ;
F0->0 ;
```

FIGURE 10 – Un état 0 initial, dans un fichier dot

### 4.2 L'ajustement des tailles des automates

Ce problème est causé par la taille des automates, puisque l'on ne connaît jamais la taille de l'image qui est chargée par le `QSvgWidget`.

La mise en forme dans un premier temps n'était pas pratique : les images d'automates trop grands étaient coupées en plein milieu ou les ascenseurs ne s'affichaient pas...

Ce problème a été résolu après une meilleure compréhension de QT, en modifiant la structure globale de la fenêtre. Nous aurions souhaité faire un affichage plus agréable des automates, en affichant les transitions de gauche à droite, au lieu de haut en bas. Cependant Graphviz ne contient pas d'option permettant de faire cette manipulation efficacement, seulement une option permettant de pivoter l'image final, ce qui ne facilite pas la lecture du texte à l'intérieur des états.

### 4.3 Double parcours de vecteur

Nous avons eu des problèmes, notamment lors de la création des algorithmes de produit et de détermination, lors de double parcours sur vecteur.

Nous utilisions deux itérateurs distincts

sur un même vecteur, et cela nous provoquait des erreurs de parcours. Après avoir cherché un moment une erreur de notre part, nous avons fini par éviter le problème en faisant simplement une copie de vecteur avant de le parcourir. En effet, sans cette copie, lorsque l'on modifiait un itérateur, l'autre était modifié également et prenait la valeur de l'autre.

```
transitions = (*it_etat).getTransitions();
//Pour chaque transition de cet état
for(it_trans=transitions.begin();it_trans!=transitions.end();it_trans++){
```

FIGURE 11 – Une copie de vecteur, avant de parcourir ce vecteur

---

## Conclusion

Ce projet nous a intéressé dès le début pour différentes raisons.

Tout d'abord, il nous a permis de revoir, et d'approfondir des notions que nous avons acquies précédemment, en particulier avec l'utilisation du framework QT. Il nous a également permis de découvrir un nouvel outil : Qt-Creator, et à travers lui de voir ce qu'il se fait de récent en matière d'outils de programmation.

Ensuite, nous avons également pu mettre en application nos cours de programmation du semestre, tout en travaillant en équipe, discutant ainsi de nos points de vue pendant le développement.

Le fait que notre projet soit une application concrète a été également un sérieux plus afin de nous motiver à réaliser un outil utile et intuitif.

De plus, nous avons la satisfaction d'avoir atteint les objectifs fixés, tout en ayant un logiciel multi-plateformes. Malgré tout, notre application possède quelques limites. En effet, il permet de créer uniquement des automates limités à 10 états, les automates ayant de trop nombreux états et/ou transitions sont illisibles à l'écran du fait de leur taille, et l'affichage d'un automate issu du produit ou de la détermination devient moins rapide lorsqu'il atteint environ 80 états à cause du temps de chargement de l'image (environ 0,5s à 1s).

Pour finir, l'application pourrait évoluer dans le futur, par l'ajout de nouveaux algorithmes, comme l'union d'automates (court terme), la minimisation ou encore l'autorisation d'un alphabet de lettres (moyen terme), voir même la génération du langage ou de la grammaire issue de l'automate (long terme).

---

# Table des matières

<b>Introduction</b>	<b>1</b>
<b>1 Présentation du sujet</b>	<b>2</b>
<b>2 Analyse de l'existant</b>	<b>3</b>
2.1 AnimAlgo . . . . .	3
2.2 JFLAP . . . . .	4
<b>3 Mise en œuvre</b>	<b>5</b>
3.1 Organisation et partage des tâches . . . . .	5
3.2 Conception . . . . .	5
3.3 Langages et Outils utilisés . . . . .	6
3.3.1 C++ . . . . .	6
3.3.2 Framework Qt . . . . .	6
3.3.3 QTCréator . . . . .	6
3.3.4 Subversion . . . . .	6
3.3.5 Graphviz . . . . .	7
3.4 Développement de l'interface . . . . .	7
3.4.1 Manipulation d'automates par l'interface . . . . .	7
3.4.2 Affichage d'un automate . . . . .	7
3.4.3 L'utilisation de l'interface . . . . .	9
3.5 Implémentation des algorithmes de manipulation d'automate . . . . .	10
3.5.1 Produit d'automates . . . . .	10
3.5.2 Détermination . . . . .	10
<b>4 Problèmes rencontrés</b>	<b>12</b>
4.1 Respect de la convention de dessin d'un automate d'états finis . . . . .	12
4.2 L'ajustement des tailles des automates . . . . .	12
4.3 Double parcours de vecteur . . . . .	12
<b>Conclusion</b>	<b>13</b>
<b>Bibliographie</b>	<b>15</b>
<b>Annexe</b>	<b>16</b>
Diagramme UML . . . . .	16
Installation de l'application sous Windows . . . . .	17



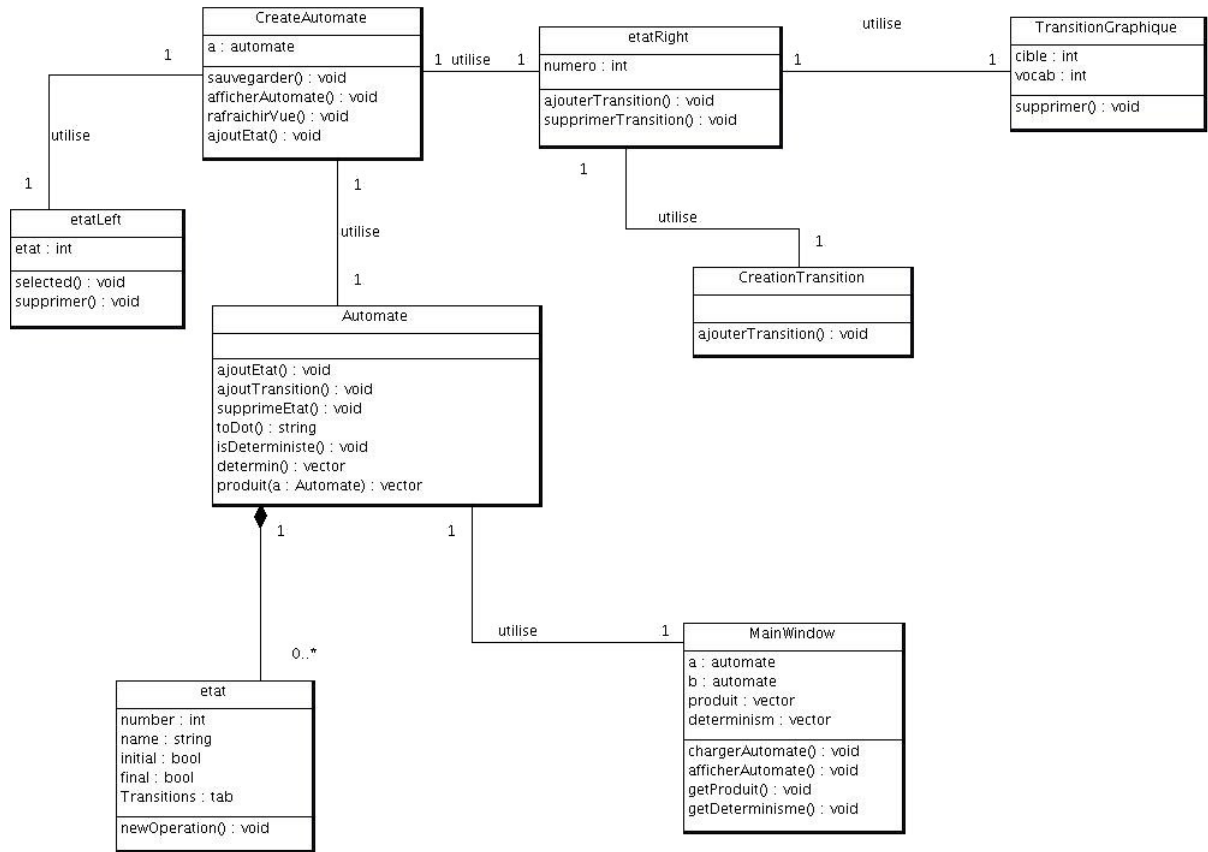
---

## Bibliographie

- [1] [http ://www.cplusplus.com/reference/stl/](http://www.cplusplus.com/reference/stl/). Site anglophone de documentation sur la bibliothèque STL et le C++ en général.
- [2] Leo FACCHINI, Jocelyn et RIZZON. **Animation d’algorithmes sur les automates d’états finis**. 2009. Rapport de projet.
- [3] [http ://www.graphviz.org/Documentation.php](http://www.graphviz.org/Documentation.php). Documentation officiel de GraphViz.
- [4] [http ://www.siteduzero.com/tutoriel-3-14171-creer-une-installation.html](http://www.siteduzero.com/tutoriel-3-14171-creer-une-installation.html). Tutoriel pour apprendre à créer une installation.
- [5] [http ://qt.nokia.com/products/developer-tools](http://qt.nokia.com/products/developer-tools). Site officiel de téléchargements du Framework QT et de l’EDI QTcreator.
- [6] [http ://doc.trolltech.com/](http://doc.trolltech.com/). Documentation officiel de QT pour le langage C++.

# Annexe

## Diagramme UML



## Installation de l'application sous Windows

