CAHIER DES CHARGES - PROJET S4

Djibril Traore (Chef de projet)
Amine Mike EL MAALOUF
Attilio LEVIEILS
Classe: B1

Février 2023

Contents

1 Introduction

L'équipe de MatchMakers a la joie et l'honneur de vous présenter aujourd'hui les premières avancées du projet Underperl !

Vous découvrirez ci-dessous une explication détaillée du fruit de notre travail de ces dernières semaine ainsi que ce dont vous pourrez vous attendre pour la prochaine soutenance mais tout d'abord laissez nous vous aider à vous remémorer ce qu'est Underperl !

2 Reprise du cahier des charges

2.1 Rappel du projet

Underperl est donc le fruit d'échanges approfondis à l'intérieur du groupe sur ce qu'est un logiciel utile ainsi que des discussions sur quel types d'algorithmes nous trouvons intéressants et voulons implémenter une version qui nous est propre.

Un point qui nous a tous rapproché a été notre intérêt commun pour les cours de THLR suivis au semestre dernier. Nous avons donc évoqué la piste de faire un projet en rapport avec cette matière et donc logiquement quelque chose qui utilise les expressions régulières.

Un second point d'entente a été notre appréciation du projet Abacus réalisé comme TP de programmation au deuxième semestre. Plus particulièrement, pouvoir construire quelque chose de A à Z sans se reposer sur des modules préconçus nous avait tous plu.

Ainsi, de ces deux observations le projet semblait dorénavant évident, un interpréteur d'expressions régulières qui prend en input une entrée de l'utilisateur. De cette input, nous devons

en récupérer les token formant l'expression régulière (lettres ou symbole de concaténation ou autre...) puis les passer dans le "pipeline" vu en THLR qui, d'une expression régulière forme un automate qui l'accepte.

Ce programme sera mis à la disposition de l'utilisateur à travers une interface où il pourra l'utiliser pour effectuer des traitements sur des fichiers texte tel que de la recherche, de la suppression et du remplacement de paternes.

2.2 Répartition des tâches

Tâches	**Djibril**	**Amine**	**Attilio**
Site web	-	X	X
Logiciel: design/UI	X	-	-
Utilisation des RegEx	X	-	X
Implémentation algorithmique	-	X	X

Table 1: Répartition des tâches

Nous avons décidé de scinder le projets en quatre sous tâches :

- la création du site web : il faudra prendre en compte son style et les fonctionnalités présentées.

- le design de notre logiciel : ce qui comporte l'interface utilisateur avec notamment la partie inscription de l'expression régulière ainsi qu'une zone pour y ajouter des fichiers par exemple et/ou une autre pour choisir l'action qui va être exécutée (recherche, tri, remplacement,...).

- l'implémentation des programmes effectuant les actions citées précédemment, c'est-à-dire l'interprétation de l'entrée donnée par l'utilisateur en tant qu'expression régulière et l'application dans un cadre concret de ces expressions.

- l'implémentation des algorithmes permettant de créer un automate qui reconnaît une expression régulière donnée. Nous aurons besoin, comme vu dans la partie précédente, de plusieurs algorithmes qui sont interdépendants et font office de "pipeline" puisqu'avec notre implémentation, chaque output d'un algorithme sera utilisé comme input du suivant.

Cette répartition nous semble logique, proposant des tâches ayant besoin de qualités différentes:
de la création et un esprit artistique pour la partie design du site web et du logiciel ainsi qu'un mode de pensée plus abstrait lors de l'implémentation des algorithme en comparaison avec celui nécessaire à la mise en place des programmes utilisant ces algorithmes. Nous trouvons ceci adapté aux points forts dont chaque membre de l'équipe dispose.

De plus, aucunes de ces tâches ne dépendent les unes des autres, assurant une autonomie de travail pour chaque personne dans le groupe à tout moment de la réalisation du projet. Cette indépendance est primordiale à la bonne réalisation du projet. Une mise en commun sera cependant obligatoire lors de la connexion des différentes parties entre elles.

2.3 Rappel des objectifs

Tâches	**Sout 1**	**Sout 2**	**Sout 3**
Site web	30%	70%	100%
Logiciel: design/UI	30%	70%	100%
Utilisation des RegEx	20%	70%	100%
Implémentation algorithmique	50%	90%	100%

Table 2: Planning d'organisation

Nous avions prévu une avancée homogène dans les différentes parties du projet, avec cependant une nette avance de l'implémentation algorithmique par rapport au reste.

Ce choix s'est avéré réaliste puisque nous avons vite remarqué l'étendu du travail de fond qu'il fallait réaliser.
En effet, nous nous sommes rendu compte du nombre d'outil, avec en tête de proue les structures de données, dont nous avons besoin et allons utiliser tout au long du projet et donc devoir implémenter en tout premier lieu.

3 Avancement du projet

3.1 Interpréteur

Les bases de l'interpréteur sont posées. Le lexer est le parser sont foctionnel avec des regex simple. L'ojectif pour la suite serait d'implementer plus de fonctionnionalités.

3.2 Implémentation algorithmique

Nous avions fixé la complétion de l'implémentation algorithmique à 50% pour cette soutenance et nous sommes fiers du fait que nous avons pu atteindre notre objectif.
En effet, nous avons actuellement un arsenal de structures de données que seront utiles à toutes les parties de notre projet, ainsi qu'un programme pour construire un automate fonctionnel non optimisé (epsilon transition nondeterministic automaton) à paritr d'un AST.

3.3 Site Web

Les bases du site web ont été mis en place pour cette premières soutenance. Nous pouvons être satisfait du travail présenté et optimiste pour les améliorations à venir.

En effet, nous avons un site web en ligne qui présente plusieurs pages liées entre elles à l'aide de boutons et une ligne directrice en terme de design qui a été dégagée.
De plus, le plus important lors de cette partie a été la prise de connaissance d'un bon nombre d'outils (langages, applications, ...) par l'équipe qui se charge de cette partie qui était novice dans le domaine.

4 Les structures

4.1 Les ensembles et dictionnaires

Les différents algorithmes qui manipulent des automates que nous voulons implémenter (Thompson pour la soutenance 1), ont recours de nombreuses fois aux ensembles et dictionnaires.
En effet, il sera pratique d'avoir des structures pouvant représenter l'ensemble des sommets de départ ou l'ensemble des arcs qui composent un chemin par exemple...

4.1.1 Le hashage

L'implémentation que nous avons privilégiée pour les ensembles et les dictionnaires se repose sur celle vu lors d'un TP de programmation utilisant un table de hashage.

Il faut donc générer une clé de hashage à chaque valeur de données que nous voulons ajouter à notre ensemble.
Pour se faire nous utilisons l'algorithme de Jenkins ("Jenkins one at a time hash") que nous calculons modulo la capacité de la table de hashage pour s'assurer que le résultat peut être un index de cette table.

De plus, nous avons décidé de gérer d'éventuelles collisions primaires par liste chaînée. Ce qui nous a motivé à faire ce choix est la facilité de mise en place de ce système.

4.1.2 Les ensembles

Les ensembles sont donc représentés par une table de hashage de capacité donnée, qui double à chaque fois qu'elle atteint un seuil de remplissage supérieur ou égal à 75%, comportant un tableau de données. Ces données sont définies par leur clé de hashage, une autre clé contenant leur valeur et possède un poin-

teur qui, en cas de collision primaire, pointe vers une éventuelle autre donnée ayant le même index dans la table.

```c
typedef struct data
{
    uint32_t hkey;
    void *key;
    struct data *next;
}data;

typedef struct set
{
    size_t len;
    size_t size;
    size_t capacity;
    struct data **elements;
}set;
```

Avec cette structure, s'accompagnent les diverses fonctions effectuant des opérations basiques sur les ensembles tel que la création d'un nouvel ensemble vide, la recherche d'une clé, l'insertion d'une donnée, sa suppression ainsi que la suppression de l'ensemble dans son intégralité mais aussi l'union entre deux ensembles.

4.1.3 Les dictionnaires

La structure des dictionnaires s'apparente grandement à celle des ensembles.
En effet, cette structure de données se repose sur les mêmes principes : une table de hashage et des données disposant d'une clé de hashage qui s'insèrent dedans. Nous utilisons également les listes chainées pour résoudre les collisions primaires.

La seule différence importante est dans la structure des données qui composent la table de hashage. Ils disposent en plus de leur clé de hashage et d'un pointeur vers une autre données s'il y a eu collision d'un champ clé et d'un champ valeur de tout type.
Ces deux champs sont ceux qui font de cette structure un dictionnaire, puisqu'ils permettent à l'utilisateur d'accéder à la valeur grâce à la clé.

```c
typedef struct pair
{
    uint32_t hkey;
    char* key;
    void* value;
    struct pair* next;
}pair;
```

Des fonctions évidentes viennent s'ajouter : la création d'un nouveau dictionnaire, la recherche par rapport à la clé, l'insertion, la suppression et la possibilité d'accéder à la valeur de la donnée grâce à la clé.

4.2 Structures de données linéaires

4.2.1 File

En informatique, une file *en anglais* ***queue***) est un type abstrait basé sur le principe " premier entré, premier sorti " ou PEPS, désigné en anglais par l'acronyme FIFO (" first in, first out ") : les premiers éléments ajoutés à la file seront les premiers à en être retirés.

Notre implémentation des files repose sur celle vue en cours de programmation. Elle est accompagnée de fonctions permettant de manipuler les files : ajouter/retirer un élément,...

4.2.2 Pile

En informatique, une pile (*en anglais* ***stack***) est une structure de données fondée sur le principe " dernier arrivé, premier sorti " (en anglais LIFO pour last in, first out), ce qui veut dire qu'en général, le dernier élément ajouté à la pile est le premier à en sortir.

De même, notre implémentation repose sur celle vue en cours de programmation ainsi que ces fonctions.

4.2.3 Listes chaînée

Une liste chaînée ou liste liée (en anglais linked list) désigne en informatique une structure de données représentant une collection ordonnée et de taille arbitraire d'éléments de même type, dont la représentation en mémoire de l'ordinateur est une succession de cellules faites d'un contenu et d'un pointeur vers une autre cellule. De façon imagée, l'ensemble des cellules ressemble à une chaîne dont les maillons seraient les cellules.

L'accès aux éléments d'une liste se fait de manière séquentielle : chaque élément permet l'accès au suivant (contrairement au tableau dans lequel l'accès se fait de manière directe, par adressage de chaque cellule dudit tableau).

Le principe de la liste chaînée est que chaque élément possède, en plus de la donnée, un pointeur vers un élément qui lui est contigu dans la liste. L'usage d'une liste est souvent préconisé pour des raisons de rapidité de traitement, lorsque l'ordre des éléments est important et que les insertions et les suppressions d'éléments quelconques sont plus fréquentes que les accès.

En effet, les insertions en début ou fin de liste et les suppressions se font en temps constant car elles ne demandent au maximum que deux écritures. En revanche, l'accès à un élément quelconque nécessite le parcours de la liste depuis le début jusqu'à l'index de l'élément choisi.

Ces listes nous serons utiles lors de l'implémentation de notre structure d'automate.

4.3 Les arbres binaires

Dans la pipeline permettant de passer d'une chaîne de caractères représentant une expression régulière à la création d'un automate à epsilon transition, nous devons transformer cette ex-

pression en un AST.

Les AST sont, dans notre cas, des arbres binaires de syntaxes où les feuilles sont les opérandes (les lettres de notre alphabet) et les noeuds internes les opérateurs (. + * par exemple).

Nous devons donc implémenter des arbres binaires qui seront définis par leur clé, c'est-à-dire, la valeur que contient sa racine et ses deux fils respectivement child1 et child2 éventuellement nuls. Nous avons donc choisi l'implémentation la plus naïve, s'appuyant sur une définition récursive des arbres.

4.4 Automates

Un automate est un dispositif reproduisant en autonomie une séquence d'actions prédéterminées sans l'intervention humaine, le système fait toujours la même chose.

Dans le domaine de l'informatique, on nomme automate une machine à traiter de l'information. Par opposition à la notion de fonction continue, cette information est de nature discrète : nombres entiers, par exemple 0 ou 1, caractères " a, b, c... "

La notion d'automate a émergé des besoins de programmation relatifs à l'analyse syntaxique : elle permettait de remplacer par des données — faciles à modifier — et un programme de cheminement unique ce qui aurait demandé un programme bien plus complexe et surtout bien plus délicat à maintenir par la suite (ce principe a été ensuite celui des systèmes experts).

Dans le cadre de notre projet, un automate est un graphe orienté valué par les mots d'un langages. Nous avons décider d'utiliser l'implémentaion par listes d'adjacences pour notre automate, la jugeant étant la plus adaptée à notre projet. Notre Structure de donnée possède alors comme variables:

- **order**: qui représente le nombre de sommets du graphe.
- **initial_states**: qui est un ensemble contenant les états initiaux de l'automate.
- **final_states**: qui est un ensemble contenant les états finaux de l'automate.
- **alphabet**: qui est un ensemble contenant l'alphabet de l'automate.
- **adjlists**: qui contient les listes d'adjacence de l'automate.

5 Interpreteur

5.1 Lexer

Le lexer qui est la première étape dans l'***analyse lexicale*** convertis une chaîne de caractères, dans notre cas une expression régulière, en une liste de symboles (*tokens en anglais).*
Dans le cadre de underpearl, l'implementation ci-dessous a été retenue.
L'objectif étant de reconnaître les différents caractères importants à l'aide de l'**enum tokentype** afin de les transformer en **Token**.
Le Lexer retourne finalement un **Array** contenant un pointeur de pointeur de Token représentant une liste de Token et la taille de cette dernière. Cette implementation peut être amené à évoluer.

```
enum tokentype
{
    backslash,
    dot,
    interogation_mark,
    star,
    open_bracket,
    close_bracket,
    pipe,
    open_parentheses,
    close_parentheses,
    add,
    space,
    other
};

typedef struct token
{
    size_t tokentype;
    size_t prioroty;
    size_t parity;
    char   symbole;
} Token;

typedef struct array
{
    size_t len;
    Token** start;
} Array;
```

5.2 Parser

Le parser, qui la seconde étape dans l'***analyse lexicale*** est un processus permettant de structurer une représentation linéaire en accord avec une grammaire donnée. Il s'applique à des domaines aussi variés que le langage naturel, le langage informatique ou aux structures de données. Dans le cas d'underpearl, le parser va prendre en entrée la sortie du lexer pour retourner un arbre de syntaxe abstraire ou plus communément appelé **AST**. Pour une meilleur visibilité du code mais également pour faciliter le debuggage nous avons pour l'instant diviser le parser en deux parties:

- Algorithme de Shunting-Yard
- Formation de l'AST

5.2.1 Shunting-Yard

L'algorithme Shunting-yard est une méthode d'analyse syntaxique (*parser*) d'une expression mathématique, logique ou les deux exprimée en notation algébrique parenthésée inventé par Edsger Dijkstra. Il peut être utilisé pour traduire l'expression en notation polonaise inverse ou **RPN**, ou en arbre syntaxique abstrait. Pour les raisons citées précédemment nous préférons passer par RPN avant d'obtenir l'AST. La RPN permet d'écrire de façon non ambiguë les formules arithmétiques sans utiliser de parenthèses.

Input	Output
3 + 4 × 2 ÷(1 - 5) ^2 ^3	3 4 2 × 1 5 - 2 3 ^^÷+

5.2.2 AST

Du résultat obtenu par l'algorithme de Shunting-Yard, nous voulons avoir un arbre de syntaxe, comme dis précédement nous voulons avoir uns structure de donnée qui est un arbre binaire avec comme feuilles les opérandes et dans les noeuds internes les

opérateurs.

Nous pouvons être sûrs qu'il s'agira d'un arbre binaire puisque tous nos opérateurs nécessites au plus deux opérandes (seul l'étoile de Kleene n'en nécessite que d'une), ce qui nous facilite grandement le travail.

Ce passage de la pile de résultat de Shunting-Yard à un arbre de syntaxe est crucial puisque l'algorithme de Thompson, dans la forme retenue pour le projet qui est celles s'apparentant le plus à celle implémentée en Python lors de la THLR au S3, peut construire un automate qu'à partir d'un AST

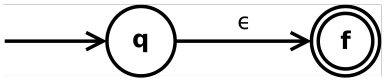
6 Algorithme de Thompson

6.1 Principe

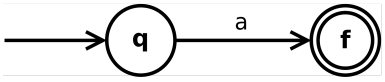
Le théorème de Kleene affirme que l'ensemble des langages rationnels sur un alphabet **A** est exactement l'ensemble des langages sur **A** reconnaissables par automate fini. Il existe des algorithmes pour passer de l'un à l'autre. L'algorithme de Thompson permet d'aller de l'expression à l'automate.

L'algorithme consiste à construire l'automate petit à petit, en utilisant des constructions pour l'union, l'étoile et la concaténation. Ces constructions font apparaître des epsilon transitions qui sont ensuite éliminées. À chaque expression rationnelle **s** est associé un automate fini **N(s)**. Cet automate est construit par induction sur la structure de l'expression.
Il existe deux cas de base:

Expression ϵ

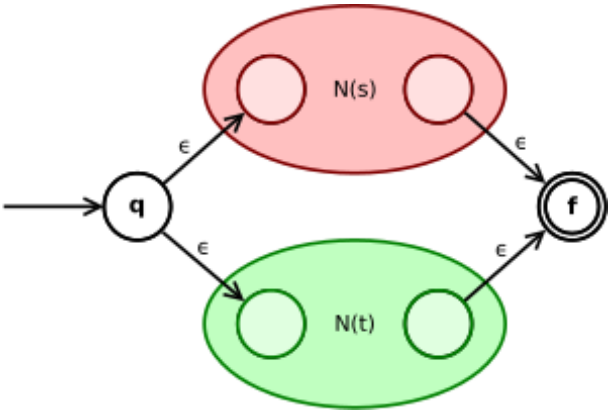


Expression a où a est une lettre

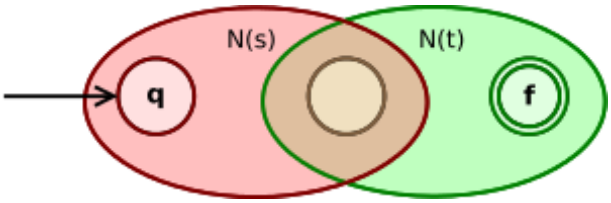


Il existe trois cas inductif:

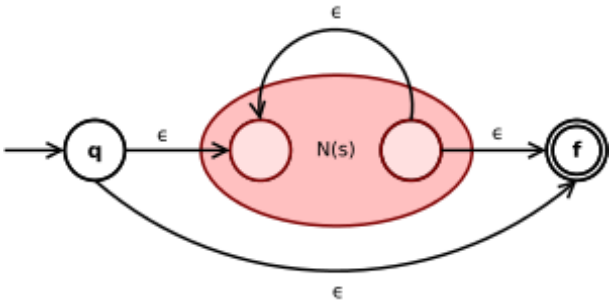
L'union



La concaténation



L'étoile de Kleene



6.2 Taille de l'automate résultant

On peut donner une majoration de la taille de l'automate en fonction de la taille de l'expression. La taille —e— d'une l'expression e est mesurée par le nombre de symboles qui y figurent, à l'exception des parenthèses. Donc

$$|\emptyset| = |\varepsilon| = |a| = 1,\ |s+t| = |s \cdot t| = |s| + |t| + 1,\ |s^*| = 1 + |s|.$$

Notons n(s) le nombre d'états de l'automate N(s). Alors

$$n(\emptyset) = n(\varepsilon) = n(a) = 2,$$

$$n(s+t) = n(s) + n(t) + 2,\ n(s \cdot t) = n(s) + n(t) - 1,$$

$$n(s^*) = 2 + n(s).$$

Dans tous les cas, on a donc

$$n(s) \leq 2|s|,$$

en d'autres termes, le nombre d'états est au plus deux fois la taille de l'expression.

Pour le nombre de transitions, un argument encore plus simple s'applique : de chaque état sortent au plus deux flèches, donc le nombre de transitions est au plus le double du nombre d'états.

6.3 Complexité en temps

En supposant connus les automates finis N(s) et N(t) associés à deux expressions régulières s et t, la construction de l'automate associé à l'union (s+t) ou la concaténation (s.t) s'effectue en temps constant (c'est-à-dire qui ne dépend pas de la taille des expressions régulières, ni des automates). De même pour la construction de l'automate associé à l'étoile de Kleene (s*).

Par conséquent, la construction de l'automate associé à une expression de taille n s'effectue en $\Theta(n)$.

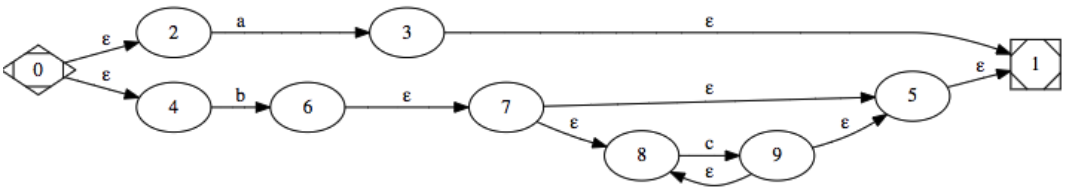


Figure 1: Automate générer par notre programme pour la regex $a + bc^*$

7 Les interfaces

7.1 L'interface utilisateur

7.1.1 Outils utilisés

Afin de présenter notre projet avec une interface graphique intuitive et facile d'utilisation, nous avons décidé d'utiliser **GTK** qui est un ensemble de bibliothèques logicielles pour interfaces graphiques. De plus, dans le but de nous faciliter le travail mais aussi de mieux maitriser le rendu final, nous avons utilisé **Glade Interface Designer** qui est un outil interactif de conception d'interface graphique GTK+ générant un fichier *XML* qui servira de base dans l'utilisation de GTK.

7.1.2 Les étapes

La production de l'interface utilisateur aka UI, peut être divisée en 3 majeures:

1. Création d'un visuel comme dit ci-dessus, intuitif et facile d'utilisation ainsi que la liste des des fonctionnalites nécessaires en prenant en compte notre avancement.
2. Implementation d'une version basique fonctionnelle.
3. Amélioration du produit et fusion de toutes les parties dans l'interface graphique.

Pour la premiere soutenance, uniquement la premiere etape a est finit finalise comme vous pouvez le voir ci-dessous.

Figure 2: Page d'acceuil

Figure 3: Page principale

7.2 Le site web

Comme nous l'avions préciser dans notre cahier des charges, le développement d'un site web est tâche inédite pour les membres de notre groupe. Une fois les pieds dans le bain, nous pouvons vous assurez que nous saurons à la hauteur de la tâches. Cette première période de projet fut une période expérimentale qui s'est avéré fructueuse, en effet, notre site est peut-être loin d'être terminé, mais nous avons pu découvrir des outils tels que VS Code et son extension Live Server qui nous faciliterons la tâches dans le future, nous rendant alors encore plus productif. Live Server permet de voir en temps réel les changements apportés à notre

site web grâce à n'importe quel navigateur. Cet outil nous permet de réaliser un gain de temps non négligeable.

Actuellement, sur une branche dans notre répo sur Github, nous possédons les premières pages de notre site web. Avoir notre index.html sur Github nous permet d'utiliser leur service d'hébergement qui est gratuit, et c'est comme ça que nous avons pu mettre en ligne notre site web.

8 Point de vu de l'équipe

8.1 Nos impressions

8.1.1 TRAORE Djibril

Pour une fois, je suis majoritairement satisfait de l'avancement du projet mais plus précisément de mon avance personnel. Je me suis occupé de la partie interpréteur en notamment fondant la base de celui-ci. J'ai pu finaliser ce qui était prévue pour cette soutenance. De plus ayant une bonne vision sur ce qu'il reste a faire je pourrais bien m'organiser en conséquence.

Concernant le prestation du groupe je suis également satisfait. J'appréhendais beaucoup ce projet notamment parce que je ne connaissais pas les membres de mon groupe cependant il y a une bonne cohésion et nous sommes généralement sur la meme longueur d'onde ce qui permet d'avancer plus rapidement et dans les meilleurs conditions. Moi qui voulais prendre ma revanche sur les anciens projet je peux dire que cela commence bien.

8.1.2 Attilio Levieils

Cette première période de travail s'est avérée intense mais je réalise déjà les compétences qu'elle m'a permis d'améliorer.

Elle a été intense car en un temps court, j'ai dû travaillé sur plusieurs terrains tous différents, que ce soit l'implémentation de certaines structures de données (set, dict et btree). Ce qui reste plutôt abstrait : il n'y avait pas une directe application de ce travail car ils servent à être intégrés plus tard à l'intérieur notamment des algorithmes manipulant les automates.
Puis j'ai dû m'occuper du développement Web pour le site du projet ce qui demande un tout autre arsenal de qualité, que je dois assimiler car c'était la première fois que je m'attaquais à ça.

Je suis content de la qualité de notre premier rendu, nous avançons bien et cela est de bon augures quant à la qualité du projet final.
De plus cela m'a permis une bonne fois pour toute de bien maitriser les pointeurs et, grâce à la création du site web, m'apprend de nouveaux langages et outils.

8.1.3 Amine Mike El Maalouf

M'étant chargé principalement de l'implémentation des automates et de sa création, j'ai eut l'occasion de beaucoup apprendre sur les graphes ainsi qu'aux différentes manières de les implémenter et je suis fière du travail que j'ai pu fournir. J'ai d'ailleurs hâte de m'attaquer à la prochaine pati du projet où je devrais manier encore plus les automates (donc les graphes).

J'ai eut aussi l'occasion de me plonger dans la création d'un site web, un monde qui m'était inconnue auparavant, et je suis assez satisfait des nouvelles compétences que j'ai pu acquérir dans ce domaine.

8.2 Nos objectifs

Nous prévoyons pour la prochaine soutenance qui aura lieu en avril une amélioration constante du projet, en particulier en terme de ce que l'utilisateur va voir.
Si pour cette première soutenance nous avons mis en place les fondations du projet, nous aurons, à temps pour le deuxième rendu, fixé la structure du projet.

En effet, vous pouvez vous attendre à un enrichissement des deux interfaces utilisateurs, le site web et l'application.
Tout d'abord une amélioration esthétique du site web grâce à nos connaissances en CSS et java script qui ne vont que progresser mais également et surtout en terme de contenu avec une explication poussée de ce qu'est une expression régulière, les opérateurs qui la compose et des exemples à disposition.
Ces deux axes d'amélioration seront attendus également pour l'application qui sera grandement enrichi des avancées dans les parties algorithmiques et programmation puisque de premières fonctionnalités seront à la disposition de l'utilisateur.

Pour la prochaine soutenance, nous envisageons aussi d'avancer sur l'élimination des ϵ-transitions, la determinisation, et la minimisation de notre automate.

9 Conclusion

L'équipe de MatchMakers espère que vous avez aimé la première version de Underperl !
Chaque membre du groupe est impliqué, motivé et croit en la qualité de ce projet qui pourra être un outil réellement utile.

Nous nous sommes renforcés dans notre maîtrise du langage C grâce à nos implémentations de structures de données qui doivent être sûres car tout le reste du projet repose sur elles.
De plus, nous avons appris à utiliser de nouveaux outils avec toujours comme priorité l'efficacité

Pour conclure, l'équipe de MatchMakers pense vous proposer un projet cohérent, de qualité et nous sommes certains de ne pas vous décevoir.