RAPPORT DE SOUTENANCE 2 - PROJET S4

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

Avril 2023

Contents

1 Introduction **4**

2 Reprise du cahier des charges **4**
2.1 Rappel du projet 4
2.2 Répartition des tâches 6
2.3 Rappel des objectifs 8

3 Avancement du projet **10**
3.1 Interpréteur 10
3.2 Implémentation algorithmique 10
3.3 Site Web 11
3.4 UI 11

4 Les structures de données **12**
4.1 Les ensembles et dictionnaires 12
4.1.1 Les ensembles 12
4.1.2 Les dictionnaires 13
4.2 Structures de données linéaires 15
4.2.1 File 15

5 Interpreteur **16**
5.1 Lexer 16
5.1.1 Le sucre syntaxique 17
5.2 Parser 19
5.2.1 Shunting-Yard 19
5.2.2 AST 20

6 Les algorithmes sur les automates **21**
6.1 Suppression des ϵ-transition 21
6.2 Suppression des états inutiles: Élagage 22
6.3 Déterminisation de l'automate 23

7 Vérification de l'appartenance d'un mot au langage de la regex **25**

8 Les interfaces **27**
8.1 L'interface utilisateur 27
8.1.1 Outils utilisés 27
8.1.2 Les étapes 27
8.1.3 Implémentation 28
8.2 Le site web 30

9 Point de vu de l'équipe **31**
9.1 Nos impressions 31
9.1.1 TRAORE Djibril 31
9.1.2 Attilio Levieils 31
9.1.3 Amine Mike El Maalouf 32
9.2 Nos objectifs 33

10 Conclusion **34**

11 Annexe **35**

1 Introduction

L'équipe de MatchMakers a la joie et l'honneur de faire un point avec vous des 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

Taches	Sout 1		Sout 2		Sout 3
	projection	constaté	projection	constaté	
Site web	30%	25%	70%	65%	100%
Logiciel: design/UI	30%	?	70%	?	100%
Utilisation des RegEx	20%	20%	70%	70%	100%
Implémentation algorithmique	50%	55%	90%	95%	100%

Table 2: Avancement constaté

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.

Dans un second temps, il a fallu implémenter la pipeline permettant de passer d'une expression régulière à un automate déterministe en priorité.
En effet, c'est seulement après s'être assuré que cette partie était opérationnelle que l'on a pu témoigner du bon fonctionnement de la partie utilisation des RegEx.

Notre seconde priorité pour cette soutenance était de régler la partie lexer et parser de notre projet.
Après les différentes difficultés rencontrées lors de la première soutenance, nous devions nous assurer de ne pas devoir perdre du temps dessus après cette soutenance et ainsi pouvoir nous concentrer sur l'exploitation des expressions régulières.

Le site internet ainsi que l'interface restent pour cette soutenance des tâches moins importantes avec un intérêt qui reste principalement esthétique.
Il faut cependant noter que nous avions comme objectif de pouvoir utiliser l'interface pour s'assurer du bon fonctionnement de chaque partie et que ces parties soient bien liées entre elles.

3 Avancement du projet

3.1 Interpréteur

Pour la première soutenance, seule une version rudimentaire de l'interpréteur était opérationnelle : le lexer ne prennait pas en compte le "sucre syntaxique", il tokenisait simplement les caractères donnés en entrée, et le parser ne gérait ni les priorités ni les parenthèses.

Ce sont ces deux points qui ont été améliorés pour cette deuxième soutenance permettant actuellement de gérer des expressions régulières plus complexes et donc portant un plus grand intérêt d'utilisation.

Cette avancée était celle espérée et prévue dans le cahier des charges, nous en sommes satisfait !

3.2 Implémentation algorithmique

Alors que plus de la moitié du travail sur l'implémentation algorithmique à été effectué en vu de la première soutenance, une partie non négligeable était attendue et crutiale tant le reste du projet en dépendait.

A l'aide des différentes structures de données implémentées précédemment il fallait désormais réaliser la partie concrète : implémenter les algorithmes optimisant l'automate obtenu en supprimant notament les epsilons transition ainsi que les états inutiles.

Au vu de ce programme chargé, arriver à tenir nos promesses se présentait compliqué mais nous avons réussi à atteindre ces objectifs et plus puisque la détermination de l'automate a été également implémenté.

3.3 Site Web

Les bases du site web avaient déjà été mises en places pour la soutenance précédente, nous n'avions plus qu'à ajouter du contenu sur les différentes pages créées.

Alors que le plus gros et les aspects plus techniques étaient réglés, le site web est depuis la première soutenance en ligne et les pages sont correctements liées entre elles, ce sont les aspects esthétique qui sont à l'honneur pour cette deuxième soutenance ! Malheureusement pour l'équipe qui en avait la responsabilité, nous n'avons pas réussi à obtenur l'amélioration stylistique espéré, ceci malgré plusieurs maquettes de site envisagés. La raison principale pour ce retard est la difficulté à maîtriser et bien comprendre CSS.

Une réelle avancée à cependant eu lieu, plus sur le contenu que la forme c'est vrai, et une remise à niveau des attentes est peut-être à envisager pour la dernière soutenance.

3.4 UI

Pour la première soutenance, seulement un visuel de l'interface utilisateur a été fait et conformément a l'avancement prévue pour la seconde soutenance l'objectif était de rendre moins le tout fonctionnel afin de montrer ce à quoi le projet final pourrais ressembler.

Nous avons réussi à faire tout ce qui était prévue pour cette soutenance et même plus car nous au final la fusion de toute les parties a pu être faite ce qui nous permettra de nous concentrer sur l'ajout de features pour la soutenance final afin de rendre le tout facilement utilisable

4 Les structures de données

Les structures ont été le coeur de nos efforts lors de la première soutenance, nous allons vous présenter les différentes modifications éventuelles et les problèmes rencontrés dû à leur utilisation.

4.1 Les ensembles et dictionnaires

Nous pensions à raison que les ensembles allaient nous être particulièrement utiles lors de la manipulation des automates
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...

Cependant, nous avons rencontrés certains problèmes en les manipulant excessivement.

4.1.1 Les ensembles

Nous vous rappelons la structure de nos ensembles qui est calquée sur celle présentée lors d'un TP de programmation que nous avons eu sur les dictionnaires.

```
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.

Nous avons vite remarqué un problème quant à l'utilisation de la clé pour localiser un élément dans l'ensemble. Nous avons noté que l'élément est bien inséré à la place correspondante au hashage de sa clé, cependant, lorsque l'ensemble est agrandi, l'élément n'est plus à sa place logique correspondante au nouveau hashage de sa clé.

Cette nouvelle a été problématique, les ensembles sont très souvent agrandi (lorsqu'ils sont occupés à 75%) et donc cette relation d'ordre si pratique et faisant la force de cette structure de données ne pouvait pas être exploité.

Le temps nécessaire à comprendre que le problème venait des ensembles le membre chargé de leur implémentation en premier lieu s'occupant désormais d'une autre partie du projet, une réécriture complète de la structure n'a pas pu être entreprise et les solutions retenues ont été mises en places en gardant en tête leurs côtés provisoires. Nous avons donc choisi parcourir séquentiellement l'ensemble lorsqu'il fallait trouver un élément, passant donc d'une complexité constante (ou linéaire en fonction du nombre d'élément ayant le même hash en cas de collision primaire) à une complexité linéaire en fonction du nombre d'éléments dans l'ensemble...

4.1.2 Les dictionnaires

La structure des dictionnaires s'apparente grandement à celle des ensembles.
Leur implémentation a été automatique et plus faite "au cas où" que pour répondre à un réel besoin prévu.

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.

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

Nous avons réalisé que nous avions pas de raison d'utiliser de dictionnaire, rendant toute cette implémentation futile...

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.

Nous avons revu notre implémentation des files qui se reposaient sur un tableau et donc d'une taille qui devait être connue à l'avance. Les files nous sont utiles dans le lexer puisque les tokens obtenus après lecture de la string sont mis dans une file. Avec le sucre syntaxique, la taille de la file de sortie n'est pas forcement la même que celle de la chaîne de caractère d'entrée, nous devons donc utiliser une liste chaînée et non plus un tableau.

Pour se faire nous nous sommes appuyés de l'implémentation proposée lors de nos cours magistraux de programmation.

```
typedef struct queue
{
    struct queue* next;
    Token* data;
} Queue;
```

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 une file de Token et la taille de cette dernière.

```
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;
```

5.1.1 Le sucre syntaxique

Le lexer, dans sa forme la plus simple a donc été implémenté dès la première soutenance. La partie plus complexe qu'il restait à faire était d'implémenter le sucre syntaxique, c'est-à-dire des simplifications de notations permettant d'écrire des expressions régulières plus courte, facile à lire et à comprendre ainsi qu'exprimer des cas plus complexes.

Concrètement, deux améliorations étaient attendues, la concaténation implicite ainsi que l'implémentation des "brackets".

- Concaténation implicite :

Alors qu'il fallait précédemment noter toutes les concaténations de caractères à l'aide du point '.', nous voulions faire en sorte de nous en passer et la rendre implicite dans plusieurs cas :

- entre deux caractères : "a.b" peut s'écrire "ab"
- entre deux parenthèses ou brackets : "(ab).[cd]" peut s'écrire "(ab)[cd]"
- entre des parenthèses/brackets et des caractères : "(ab).c.[de]" peut s'écrire "(ab)c[de]"

Comme vous pouvez le constater cette petite modification simplifie énormément l'écriture et la rend plus lisible.

Pour l'implémenter, nous faisons attention au token précédemment lu dans et dans le cas où celui courant est de même type que ceux dans l'un des trois cas énuméré au-dessus un token '.' point est inséré entre le précédent et le courant.

- Les brackets :

Un autre outil très puissant des expression régulières est l'utilisation des brackets '[]'. Ces dernières permettent de définir un ensemble

de caractère à l'intérieur et l'expression est validée si l'un de ces caractère est présent.
Ainsi, par exemple [abc] veut dire soit a soit b soit c.

La deuxième propriété des brackets qui les rend très interessantes à utiliser est la possibilité d'exprimier des intervalles de caractères.
Ainsi, par exemple [a-z] veut dire une lettre minuscule (de a à z).

Pour l'implémenter nous parcourons le contenu des brackets et nous ajoutons des '+' entre chaque caractères dans le premier cas ou des '+' ainsi que les caractères compris dans l'interval dans le second cas.

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 ^^÷+

Table 3: expression arithmétique en notation polonaise inversée

Notre version de l'algorithme de Shunting-Yard lors de la première soutenance ne traitait pas les parenthèses, limitant donc les expressions régulières que nous pouvions manipuler.

Cette modification facile mais essentielle a été faite pour cette deuxième soutenance avec également la détection d'erreur en cas de mauvais parenthésage de l'expression régulière.

5.2.2 AST

L'arbre syntaxique de la première soutenance étant toujours fonctionnel, nous n'avons pas modifié cette partie.

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 Les algorithmes sur les automates

Pour la soutenance précédente, nous avions réussi à construire un automate fini non deterministe à ϵ-transition (ENFA) à partir d'un AST grâce à l'algorithme de Thompson.

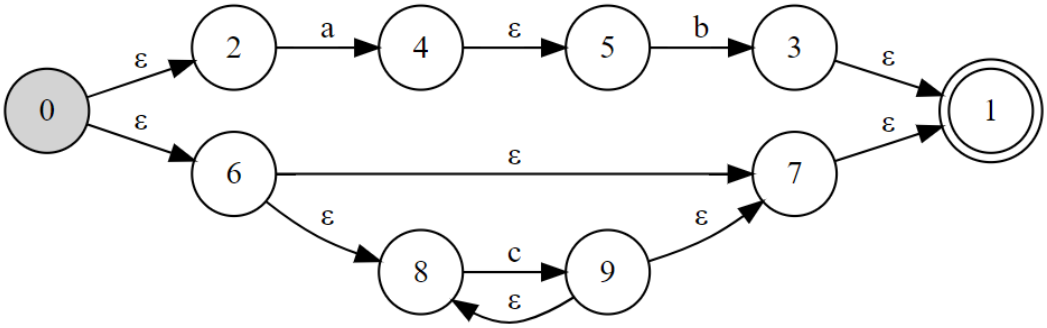

Figure 1: ENFA de la regex a.b+c*

Malheureusement, cet automate n'est pas encore utilisable afin de determiner si un mot appartient au langage de l'expression régulière, il nous faut encore retirer les ϵ-transitions, l'élaguer, et le rendre déterministe.

6.1 Suppression des ϵ-transition

Bien que l'algorithme de Thompson soit très efficace, il possède un point négatif, il crée des ϵ-transitions: un court mot pourrait emprunter un long chemin, sans ϵ-transition le chemin ferra exactement la taille de l'entrée.

Afin de résoudre ce problème, nous allons procéder à l'élimination arrière des ϵ-transition de l'automate:
Soit trois états de l'automate q1, q2 et q3, s'il existe un chemin de q1 à q2 en empruntant uniquement des ϵ-transitions, et qu'il

existe un arcs de q2 à q3 de valeur a, alors nous ajoutons une arrête de q1 à q3 de valeur a.
De plus, si à partir d'un état q nous pouvons atteindre un état final en empruntant uniquement des ϵ-transition, alors l'état q devient un état final.
Une fois cela fait, nous retirons toutes les ϵ-transition de l'automate et nous obtenons alors un automate fini non derministe (NFA).

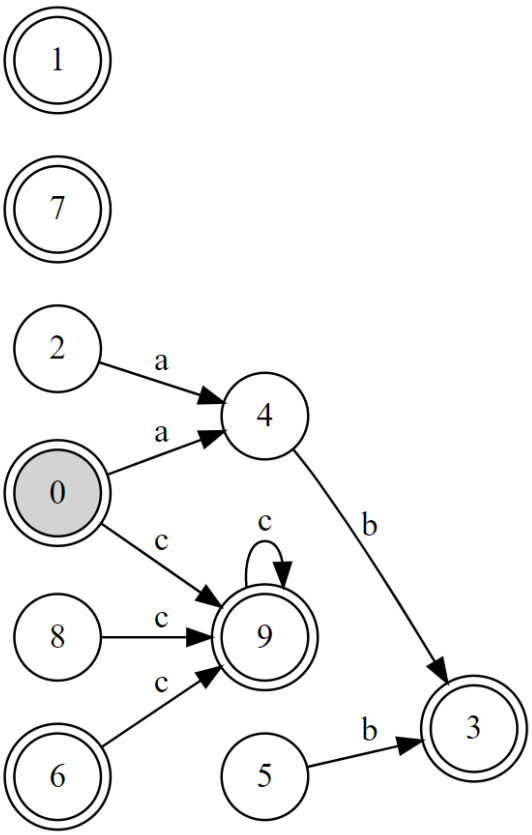

Figure 2: NFA de la regex a.b+c*

6.2 Suppression des états inutiles: Élagage

Un état est dit utile s'il est à la fois accessible et co-accessible. Un état est dit accessible s'il existe un chemin depuis un état

initial vers lui.
Un état est dit co-accessible s'il existe un chemin depuis cet état vers un état final.
Pour chaque état de l'automate, nous allons vérifier s'il est utile, s'il ne l'est pas, nous le supprimons de l'automate.

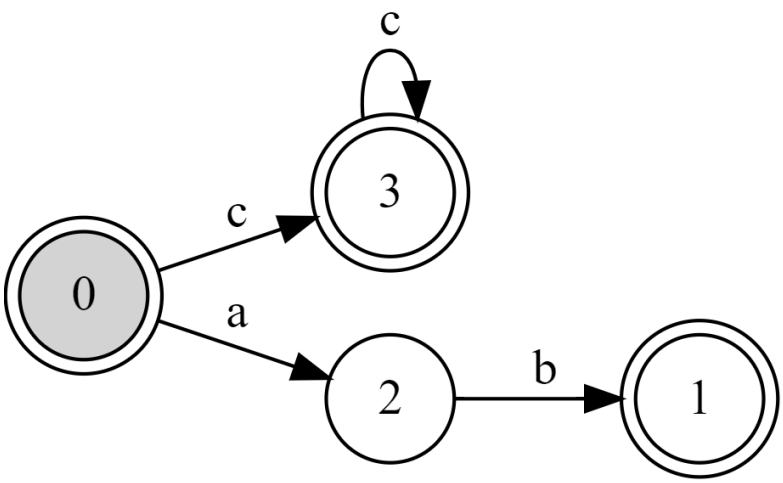

Figure 3: NFA de la regex a.b+c* après élagage

6.3 Déterminisation de l'automate

Notre automate ayant déjà vécu un long voyage, il ne lui reste plus qu'un arrêt avant le terminus: il faut le rendre déterministe. Le problème que pose notre automate actuellement est qu'il peut exister plusieurs arcs avec la même lettre qui sortent depuis un même état. Le réel problème que cela pose est que le temps d'exécution ne sera pas linéaire: au cas où n arcs ont la même lettres, il faudra au pire parcourir les n états afin de savoir si le mot appartient au langage.

Afin de rendre le temps d'exécution linéaire, nous allons prendre le risque de potentiellement avoir une explosion exponentielle

de la taille de l'automate: nous échangeons de la mémoire pour de la vitesse.

Pour l'algorithme nous allons travailler avec des ensembles d'états.
-Premièrement créons une file et nous enfilons un ensemble contenant les états initiaux de l'automate et nous créons un automate de taille 1.
-Tant que la file n'est pas vide nous la défilons.
-Pour chaque lettre de l'alphabet de l'automate, nous allons cherchez les états accessible depuis chacun des états de l'ensemble défilé, nous allons ensuite mettre ces états accessibles dans un ensemble qu'on enfilera dans la file, si l'ensemble ne possède pas déjà un état dans l'automate, nous lui en créons un.

Enfin nous ajoutons un arcs de valeur la lettre de la recherche depuis l'état enfiler en début de boucle vers l'état que nous venons d'enfiler.

Après l'exécution de cet algorithme, nous obtenons enfin un automate fini déterministe (DFA).

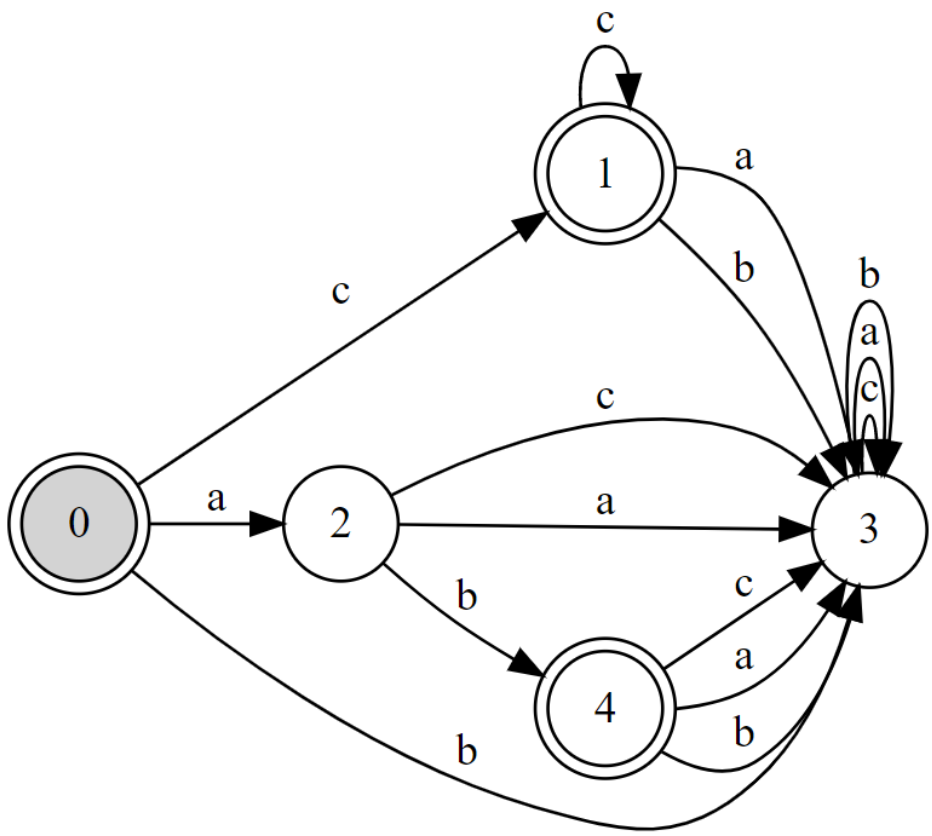

Figure 4: DFA de la regex a.b+c*

7 Vérification de l'appartenance d'un mot au langage de la regex

Arrive enfin l'étape tant attendue: comment savoir si un mot appartient au langage de la regex. Pour commencer, nous allons créer un AST à partir de la regex et nous allons appliquer notre pipeline pour construire un automate fini déterministe (DFA): Construction de l'automate grâce à Thompson –> Suppression des ϵ-transition –> Élagage -> Determinisation.

Une fois cela fait, nous allons parcourir le mot lettre par lettre. En commençant depuis l'état initial de l'automate, nous allons pour chaque lettre se diriger vers l'état où nous mène l'arc valué de cette lettre ainsi que passer à la lettre suivante du mot.
S'il n'existe pas d'arc sortant avec une lettre du mot, cela signifie que la lettre n'appartient pas au langage de la regex, donc le mot non plus.
Une fois le mot terminé si nous nous situons dans l'automate sur un état final, alors le mot appartient au langage de la regex, sinon, le mot n'appartient pas au langage de la regex.

8 Les interfaces

8.1 L'interface utilisateur

8.1.1 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.

8.1.2 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 seconde soutenance, comme précisé précedement, toutes les étapes ont été réalisé. Pour la soutenance finale, nous devrons juste rendre l'UI plus facile d'utilisation.

Figure 5: Page d'acceuil

Figure 6: Page principale

8.1.3 Implémentation

L'implementation peut toujours évoluer au fur et à mesure de l'avancement du projet. Cependant, afin d'avoir un code pas trop complexe à comprendre et surout afin de ne pas se perdre en le lisant, nous avons instantenement pensé à créer des struct et c'est principalement ces derniers qui risque d'évoluer. Etant donnée que nous avons deux pages diffrentes nous avons donc creér 2 struct. Le premier struct Page qui contient les elements d'une page, ces derniers varient d'une page à l'autre, et le second correspond à l'interface utilisateur dans l'ensemble et contient notement les struct de chaques pages. L'objectif étant d'avoir

accés à toute données utile facilement.

```
typedef struct Page
{
    GtkWindow *window;
    GtkButton *start_button;
    GtkButton *close_button;
    GtkImage *image_area;
    GtkTextView *fileView_area;
    GtkButton *execute_button;
    GtkEntry *command_entry;
} Page;

typedef struct UserInterface
{
    GtkWindow *current_window;
    Page *first_page;
    Page *second_page;
    Page *third_page;
    GtkImage *image_area;
    GdkPixbuf *pixbuf;
    GtkEntry *command_entry;
    GtkButton *execute_button;
    GtkTextView *fileView_area;
    guint event;

} UserInterface;
```

Un point positif et que nous avons fortement apprécié avec l'utilisation de Glade est que nous n'avions pas gérer la création de boutton ou de ***GtkWidget*** plus généralement mais uniquement les signaux associés avec la fonction ***g_signal_connect()***. En effet, nous pouvons récupérer les *GtkWidget* via un ***GtkBuilder*** qui est initalisé à partir du fichier *XML* généré par Glade. Ceci a grandement simplifié le code aussi bien pour nous que pour les potentiels personnes qui le liront. Dans la même optique, nous avons décidé de travaillé sur plusieurs fichiers. Les trois principaux sont les fichiers *utils.c*, *utils.h* et le *main.c* qui contiennent respectivement:

- La majorité des fonctions utilisées, notement celles executées par l'activation d'un signal avec ***G_CALLBACK()***.
- La déclaration des struct et des fonctions utilisées.

- La fonction *main* qui contient l'initialisation du *GtkBuilder* énoncé précédememt et par conséquent des *GtkWidget* et des signaux.

8.2 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 et la tâche s'est avérée être plus dure que prevue...

Alors que nous avions plutôt bien commencé nos avancées sur le site web en arrivant à le mettre en ligne rapidement et en trouvant des outils comme Live Server qui s'annoncaient comme ête très utiles pour la suite, nous avons pris du retard par rapport aux projections de début de projet.

Ce qui est d'autant plus frustrant est que ce retard ne peut pas s'expliquer par un manque de travail. En effet, l'équipe chargé du site a passé un nombre non négligeable d'heures à effectuer différentes maquettes pour le rendu esthétique du site web mais aucunes ne nous a convaincues.
Nous sommes arrivés à la réalisation que CSS et HTML sont plus durs à prendre en main et la complexité de ces langages a peut-être été sous estimé au début du projet.

Des changements notables sont cependant à noter : la mise en place d'un bandeau pour cacher le reste du temps les boutons qui n'étaient pas au goût de l'équipe design ainsi que l'apparition de contenus sur les différentes pages du site.
Les utilisateurs pourront désormais s'informer sur ce qu'est UnderPerl, connaître un peu plus l'équipe derrière le projet ainsi qu'apprendre ce qu'est une regex et comment la manipuler.

9 Point de vu de l'équipe

9.1 Nos impressions

9.1.1 TRAORE Djibril

Dans la continuité de mes ressentits de la premiere soutenance, je suis satisfait de l'avancement du projet. Pour la premiere depuis mes débuts a EPITA on envisage d'implémenter des "bonus" dans un projet.

Concernant le prestation du groupe je suis tout aussi satisfait que pour la premiere soutenance. La cohésion de groupe est bonne et l'avancement du projet est bonne.

9.1.2 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 et assez frustrante puisqu'une grande partie du travail a été d'essayer de régl
Puis j'ai dû m'occuper les différents bugs hérités de l.a Le travaisoutenance l sur le paprécédenter du développement Web pour le seite du projet ce qui demande un tout autre arsenal de qualité, que je dois assimiler et que je réalise plus dur à maîtriser que ce que je pensais.

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 ou les structs et, grâce à la création du site web, m'apprend de nouveaux langages et outils.

9.1.3 Amine Mike El Maalouf

M'étant chargé principalement de l'implémentation algorithmique sur les automates, j'ai eu encore plus l'occasion d'apprendre sur les graphes et je suis fier du travail que j'ai pu fournir.

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

9.2 Nos objectifs

Nous prévoyons pour la prochaine et dernière soutenance qui aura lieu en mai la réalisation complète du projet.
Lors de ces deux premières soutenances nous avons mis en place tous les éléments nécessaire à la finalisation du projet.

Nos yeux sont maintenant rivés sur l'implémentation à proprement parlé de notre logiciel de traitement de texte !

Vous pouvez vous attendre à de grands changement sur la partie interface utilisateur de notre projet. En effet, elle sera la vitrine de notre projet et devra donc être sujet à une attention toute particulière.
Les deux grands axes d'amélioration seront sur l'esthétisme de l'application et l'ajout de nouvelles fonctionnalités permettant dorénavant de manipuler des fichiers en effectuant des recherches et des modifications/remplacement dessus

Pour se faire, la partie utilisation des expressions régulières sera amplifiée avec l'ajout des fonctions permettant de manipuler les fichiers comme dit précédemment.

Du côté algorithmique, la priorité sera de fixer les différents bugs sur les ensembles qui ont été révélé pour cette soutenance.
Les autres structures et fonctions étant corrects peu d'autre modifications seront attendues.

Le principal ajout à faire au site web est l'ajout d'un download du projet.

10 Conclusion

L'équipe de MatchMakers espère que vous avez aimé cette seconde 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 différentes implémentations de fonctions manipulant des structures plus ou moins abstraites tel des automates ou des listes de Tokens. 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.

11 Annexe

L'algorithme de Thompson
https://en.wikipedia.org/wiki/Thompson%27s_construction

Élimination des ϵ-transition
https://en.wikipedia.org/wiki/Nondeterministic_finite_automaton

Élagage de l'automate
https://en.wikipedia.org/wiki/DFA_minimization

Déterminisation de l'automate
https://en.wikipedia.org/wiki/Deterministic_finite_automaton

Minimisation avec l'algorithme de Moore
https://en.wikipedia.org/wiki/DFA_minimization