CAHIER DES CHARGES - PROJET S4

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

Janvier 2023

Contents

1 Introduction

L'équipe de MatchMakers a la joie et l'honneur de vous présenter aujourd'hui et pour la toute première fois le projet Underperl !

1.1 A propos de l'équipe

Né dans le plus grand des hasards, ce groupe est composé de Djibril Traoré qui est également le chef de ce projet, Amine Mike El Maalouf et Attilio Levieils.

C'est la première fois que nous allons tous les trois travailler ensemble et nous voyons en ce projet l'opportunité de se découvrir un peu plus chaque jour.
Le fait de ne pas se connaître s'est révélé d'être en réalité un avantage quand le temps de trouver une idée est arrivée puisque nous avons pu échanger sans a priori sur la personne et donc se concentrer exclusivement sur la qualité des possibles projets.
Des idées variées ont été mises sur la table, reflétant la richesse de points de vu des esprits composant ce groupe.

De part le nombre réduit de membres (trois au lieu de quatre), nous voulions pas déterminer le sujet par un vote puisque nous pensions que l'avantage de ce petit nombre est que nous pouvons trouver une idée de projet qui plaise réellement à tout le monde et ainsi que personne ne se sente obligé de travailler pour un projet qui ne les passionne pas et donc de ne pas être complètement motivé lors de ce semestre.

Ce principe, bien que noble dans l'idée, s'est avéré bien plus compliqué qu'imaginé et un consensus a eu du mal à émerger.
Cependant après discussion nous avons réussi à obtenir un projet qui motivait et intriguait tout le monde et nous allons vous le présenter ci-dessous.

1.2 A propos 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.

Un avantage à ce projet est qu'il sera composé d'une base algorithmique puis de programmes tous indépendants les uns des autres et donc pourra être sujet à d'améliorations et d'ajouts de fonctionnalités sans remettre en cause celles pré existantes.

2 État de l'art

Il existe quelques logiciels se consacrant uniquement à l'exploitation des expressions régulières même si ceux-ci sont dans un nombre moindre à ce que l'on pourrait s'imaginer.

Le logiciel "grep" reste la référence dans le domaine grâce notamment à son efficacité mais son utilisation peut paraître compliqué pour les utilisateurs lambda puisqu'il passe par lignes de commande.

"regExBuddy" se rapproche plus de l'idée de notre projet par son interface qui a pour priorité la simplicité d'utilisation. Le site est tourné autour de l'expérience utilisateur, faisant tout pour l'aider à comprendre grâce à différents "feedbacks" visuels (affichage en direct de l'effet de modification de l'expression régulière sur un text).

Ainsi Underperl peut s'inspirer de ces différents logiciels tout en gardant ces spécificités propres : de par sa relative simplicité par rapport aux logiciels pré existants, une utilisation intuitive sera primordiale et l'interface devra être soigneusement implémenter pour assurer la compréhension la plus optimale de l'utilisateur.

3 Construction d'un automate

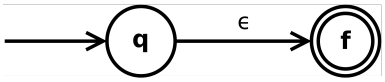
Afin de vérifier qu'un mot appartient au langage décrit par la regEx, il nous faut construire l'automate qui représente cette regEx.

3.1 L'algorithme de Thompson

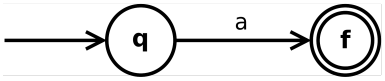
La première étape pour construire l'automate est d'utiliser l'algorithme de Thompson. Cet algorithme va générer l'automate non déterministe, fini, contenant des epsilon-transition qui représentent l'expression régulière. Une epsilon transition est une transition de l'automate qui ne contient pas de lettres, c'est-à-dire qu'elle permet de se déplacer dans l'automate sans avoir à choisir une lettre.

Le fonctionnement de l'algorithme de Thompson est assez simple, nous construisons notre regEx grâce à des cas de base et des cas inductifs.
Il existe deux cas de base:

Expression ϵ

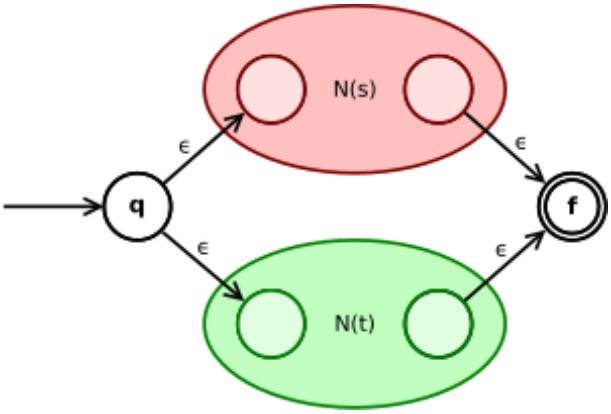


Expression a où a est une lettre

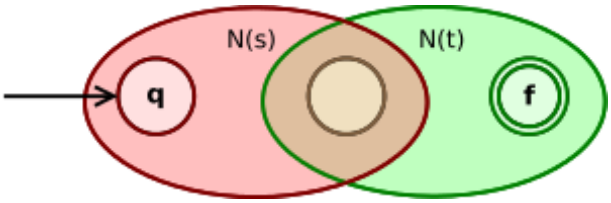


Il existe trois cas inductif:

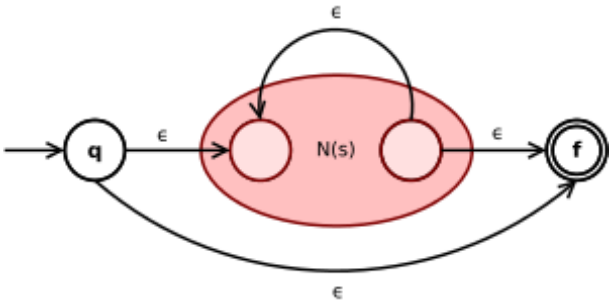
L'union



La concaténation



L'étoile de Kleene



Après avoir effectué cet algorithme, nous nous retrouverons avec un automate contenant **2n** états, n étant le nombre de symbole de l'expression régulière en excluant la concaténation et les parenthèses.

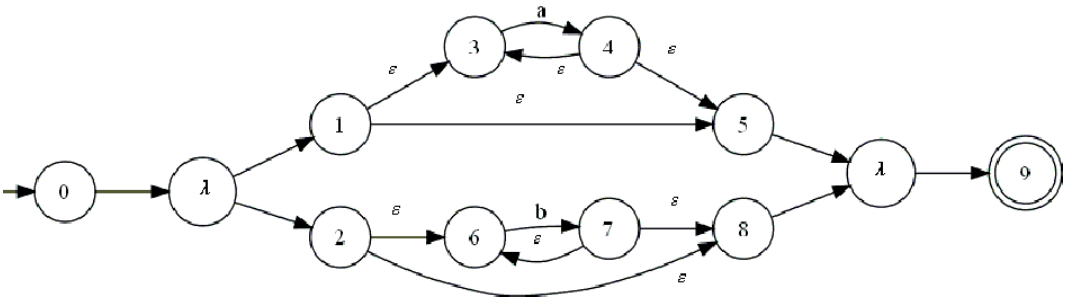


Figure 1: Résultat de l'algorithme pour l'expression $a^* + b^*$

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 O(n).

3.2 Élimination des ϵ-transition

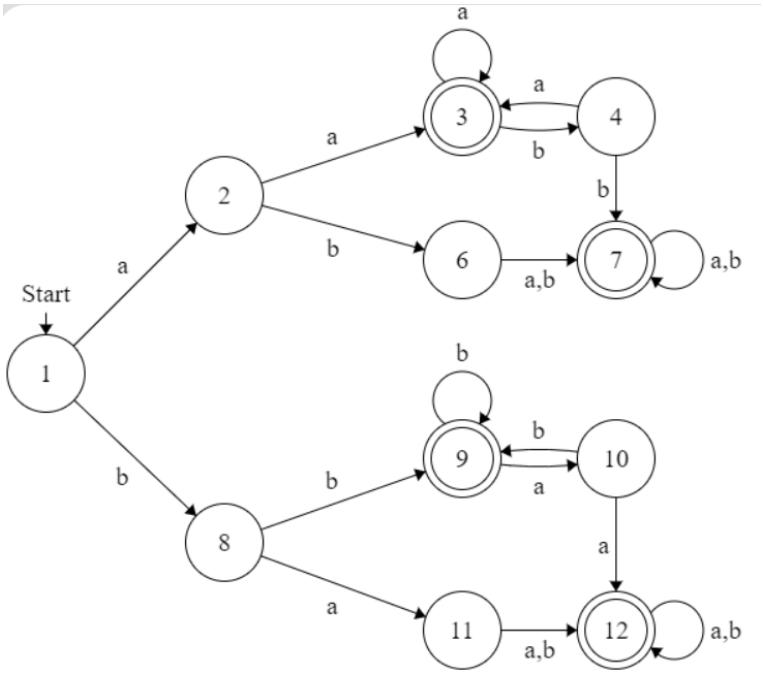
Après avoir appliqué l'algorithme de Thompson, nous obtenons un automate non déterministe, fini, contenant des ϵ-transition. La prochaine étape est de simplifier cet automate. Pour cela nous allons dans un premier temps procéder à une élimination arrière des ϵ-transitions. Cette élimination fait en sorte que le chemin emprunté par l'automate ait la même taille que l'entrée.

L'algorithme est basé sur le fait que : si on a 3 états de

l'automate q1, q2, q3 et qu'il existe un chemin de q1 à q2 ayant pour valeur une epsilon transition et s'il existe un chemin entre q2 et q3 ayant pour valeur une lettre **x**, alors il existe un chemin de valeur **x** entre q1 et q3.

Afin d'implémenter cet algorithme, nous devons commencer par calculer l'ϵ-fermeture avant de chaque état q de l'automate, à savoir l'ensemble des états accessibles depuis q en utilisant uniquement des ϵ-transitions. La manière la plus simple est d'effectuer une recherche d'accessibilité qui ne prend en compte que les ϵ-transitions.

L'automate non déterministe A' équivalent à l'automate A non déterministe à ϵ-transitions originel est alors défini de la manière suivante : ses états initiaux et ses états finaux sont égaux à ceux de A ; pour tout état q, si dans l'automate originel q1 $\in$ ϵ-transitions de q et il existe une arête de valeur a entre q1 et q2, alors on ajoute une arête de valeur a entre q et q2; enfin, si q1 $\in$ ϵ-transitions de q et q1 est un état final de A, alors q doit devenir un état final de A'.



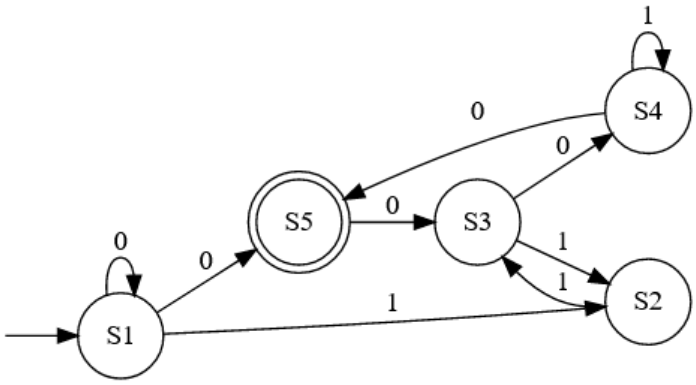
3.3 Élagage de l'automate

Une fois notre automate dépourvu de ses ϵ-transitions, il nous est possible de simplifier encore plus l'automate en supprimant les états jugés comme non utiles.

Un état q est dit **accessible** dans un automate donné s'il est accessible depuis un état initial ; il est dit **co-accessible** s'il existe un état final accessible depuis q ; il est **utile** s'il est à la fois **accessible** et **co-accessible**.

Nous déterminons si un état est utile grâce à un parcours profondeur.

Retirer les états inutiles d'un automate ne change pas le langage accepté.



3.4 Déterminisation de l'automate

Un automate est dit déterministe si:

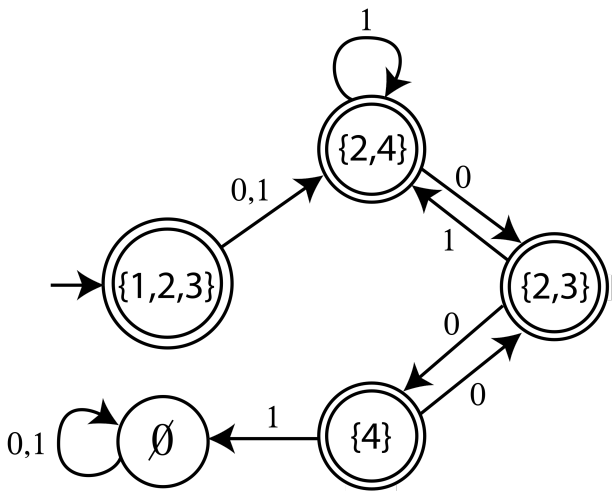
- Il ne contient pas de ϵ-transitions
- Possède un seul état initial
- Ne possède pas d'arête sortante avec la même étiquette dans le même état

Le déterminisme est efficace car est linéaire dans la taille de l'entrée. Le non-déterminisme pose aussi problème car il est non prévisible: il y a un nombre arbitraire de chemin à tester.

Pour un automate non déterministe de n états, son automate déterminisé aura au plus 2^n états. Nous prennons le risque d'une potentielle monté exponentielle de la taille de l'automate pour avoir un temps d'exécution linéaire en retour: on échange de la mémoire pour de la vitesse.

L'algorithme de déterminisation commence par explorer l'ensemble des états initiaux I $\subseteq$ Q de l'automate originel A, c'est-à-dire détermine les successeurs de I selon les différentes lettres de l'alphabet, puis explore ces successeurs à leur tour jusqu'à ne plus avoir de nouvel ensemble à explorer. Les états de l'automate déterminisé

sont alors isomorphes à ces sous-ensembles explorés ; l'unique état initial de A' correspond à I ; sont finaux les états de Q' qui correspondent à un ensemble d'états dans 2^Q contenant au moins un état final de l'automate originel A.



3.5 Minimisation avec l'algorithme de Moore

La déterminisation de l'automate pourrait pondéré un automate plus grand que nécessaire, c'est pour cela qu'il est possible de le minimiser.

Deux états q, q' sont dits indistinguables si pour tout mot w, q accepte w si et seulement si q' accepte w. Il nous est alors possible de fusionner les états indistaguables entre eux afin de réduire la taille de l'automate.

Afin de réaliser cet minimisation, nous pouvons utiliser l'algorithme de Moore qui procède de la manière suivante:

Pour commencer, nous supposons que les états finaux sont indistinguables entre eux et que les autres états sont aussi indistinguables entre eux. Tant que nous trouvons des contradictions dans les ensembles, nous détectons l'erreur et nous l'arrangeons en

créant un nouvel ensemble contenant cet état qui cause problème dans cet ensemble.

4 Appartenance de mots au langage

Après avoir construit l'automate qui représente le langage, nous passons le mot dans l'automate, si après son parcours de l'automate le mot fini sur un état final, alors le mot fait partit du langage, dans le cas contraire. Le mot n'appartient pas au langage.

5 Utilisation des regEx

Pouvant traduire une regEx en un automate et déterminer si un mot appartient au langage, nous allons maintenant vous présenter les éventuelles logicielles/fonctionnalité que nous allons implémenter.

5.1 Une recherche dans un fichier

Nous pouvons utiliser notre interpréteur de regEx afin de rechercher les éventuels mots qui vérifient le langage dans un fichier et d'afficher la ligne sur laquelle le mot est présent. Nous pourrons aussi potentiellement implémenté une fonctionnalité qui permet de remplacer les mots vérifiant le langage par un mot choisi par l'utilisateur.

5.2 Utilisation dans une API

Nous envisageons aussi, en se basant sur nos future cours magistraux de programmation sur le Networking en C, de trouver une utilisation à notre interpréteur de regEx dans une API si l'occasion de présente.

6 Répartition des charges

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 sem-

ble 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.

7 Planning d'organisation

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

7.1 Implémentation algorithmique

La tâche qui semble la plus cruciale sera l'implémentation des différents algorithmes car cela représente les murs porteurs de notre projet.

En effet, tout le bon fonctionnement des expressions régulières en dépendent, son implémentation de manière sure sera donc primordiale à la pérennité du projet. La sous-partie consacrée à l'utilisation des expressions régulière présume et se fonde sur la possibilité de reconnaître des paternes grâce aux automates conçus dans cette sous-partie.

Ainsi, cette sous-partie sera la priorité numéro 1 au début de ce projet et devra logiquement être celle la plus aboutie le jour de la première soutenance puisque, même si chaque partie du projet pourra avancer à son rythme, cela sera grâce aux algorithmes implémentés dans cette partie que l'utilisation correcte des expression régulière pourra être testée.

7.2 Site web

De part la nécessité de présenter un site web à chaque soutenance, la mise en place du notre sera une priorité et les fonctionnalités principales seront implémentées dès la première soutenance.
Il restera néanmoins quelques améliorations qui seront apportées tout au long du semestre à chaque soutenance. Notre site sera en premier lieu opérationnel, une attention au style y sera apportée dans un second temps et l'ajout de certaines fonctionnalités dépendront de l'avancement des autres parties du projet (comme le téléchargement du logiciel).

7.3 Logiciel: design/UI

L'avancement de l'interface de notre logiciel devra être bien entamé mais cette sous-partie n'est pas considérée comme une priorité car au commencement d'un projet l'équipe se focalise sur les bases ou parties fondamentales et non pas à ce dont doit ressembler le projet dans sa phase finale. Nous nous attendons donc à une interface rudimentaire, servant le rôle de ligne directrice et permettant notamment à tester les premier avancement des programmes. Il va de sois que le produit fini ne ressemblera ni de près ni de loin à ce premier jet.

7.4 Utilisation des RegEx

Enfin, le programme utilisant les expressions régulières sera encore dans sa phase la moins évoluée. Nous prédisons et voulons cette avancée puisque nous jugeons d'une plus grande priorité d'autres sous-parties précédemment nommées (site web/algorithme) et nous avons bien pris en compte le temps mis à notre disposi-

tion : nous voulons rester réalistes.
De plus, comme discuté plus haut, cette sous-partie va bénéficier des avancées des algorithmes en rapport avec les automates puisque ces derniers permettrons de vérifier le bon fonctionnement des programmes implémentés, il est donc logique que cette partie avance moins vite que celle sur les algorithmes.

8 Conclusion

L'équipe de MatchMakers est impatiente de vous présenter 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 voyons le projet de S4 comme, en premier lieu, l'opportunité de mieux se connaître et réussir à s'organiser et d'améliorer nos capacités de travail en groupe, que ce soit la répartition du travail ou la communication tout au long du semestre.

De plus, à travers ce projet nous allons assurément nous renforcer dans notre maîtrise du langage C puisque les algorithmes que nous allons devoir manipuler reposent sur des concepts nouveaux avec notamment une structure de données, les automates, que nous devrons implémenter de toutes pièces et donc devoir réfléchir à la façon la plus judicieuse et utile de les implémenter.

Enfin, les parties qui sont plus orientées utilisateurs, l'interface graphique du logiciel et le site web, représenteront un véritable défi pour nous puisqu'aucun membre du groupe est familier avec les outils appropriés, c'est-a-dire GTK pour l'interface graphique et HTML, CSS et JavaScript pour le site internet du projet. Cela ne nous fait pas peur et nous sommes même motivés à l'idée d'assimiler de nouveaux outils qui nous seront forcement utiles tout au long de notre cursus.

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

9 Annexe

9.1 Passer de la regEx à l'automate

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