SEG2506

RAPPORT
LABORATOIRE 3

PARTIE 1 :Expressions régulière et automates

Auteurs :

BOUGUENG TCHEMEUBE Renaud
4333634

Évaluateur : **Robin Tropper**

Lundi, le 28 Janvier 2008

Laboratoire 3 : barême de correction

- **Partie 1:**
  total 25/25 (2points bonus)
  - Expressions régulières
    - 6/7 (1 point chacun)
  - Automates
    - 9/10 (2 points chacun)
  - Analyseur en Java
    - 10/10
- **Partie 2:**
  total 20/25
  - Introduction (analyseur des adresses de courriel)
    3/3
  - Description de l'expression régulière retenue
    5/10
  - Programme Lex fonctionne tel que spécifié dans l'énoncé
    7/7
  - Problèmes rencontrés et leçons apprises
    5/5
  - Conclusion
    2/2

Bonus : remis à la première échéance : 5% => 2.5 points.

Déduction : pas respecté le format de soumission (seul document, seul dossier, nommenclature des fichiers) -5% => -2.5 points.

Grand total

45/50 = 90%

I- Expressions régulières:

Donnez les expressions régulières correspondantes aux langages définis ci-dessous:

1. Toutes les phrases qui commencent par ***aa***.
   aa(a | b)*

2. Toutes les phrases qui contiennent ***aa***.
   (a | b)*aa(a | b)*

3. Toutes les phrases qui contiennent un nombre pair de lettres.
   (aa | ab | ba | bb)*

4. Toutes les phrases qui contiennent un nombre pair d'occurrences de lettre ***a***
   ((aa | b) | ab*a)*

N'assure pas au moins 2 a.

5. Toutes les phrases dont la longueur est un multiple de 5.
   ((a | b)(a | b)(a | b)(a | b)(a | b))*

6. Toutes les phrases qui contiennent trois ***a*** consécutifs.
   (a | b)*aaa(a | b)*

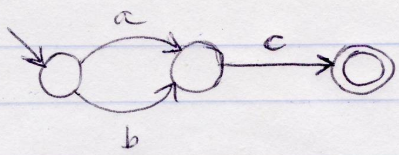
7. Toutes les phrases qui ne contiennent pas trois ***a*** consécutifs.
   (b | (ab) | (aab))*

II- Automates:

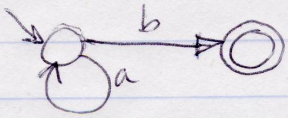
Pour chaque expression régulière ci-dessous, donnez un automate qui accepte le language défini par l'expression régulière:

1. L = (a | b) c
2. L = a* b
3. L = a* (a b c)*
4. L = (a | b)*
5. L = (a | b) c* c b b*

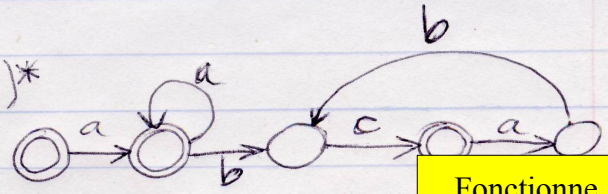
1) $L = (a|b)c$



2) $L = a^*b$



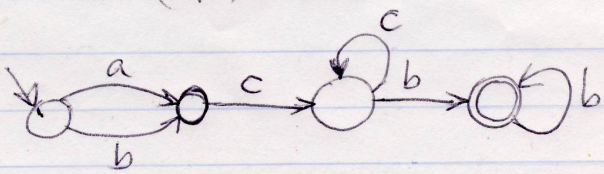
3) $L = a^*(abc)^*$



Fonctionne, mais inutilement compliqué. -1/2 voir solution

4) $L = (a|b)^*$

5) $L = (a|b)c^*cbb^*$



III- Programmer un analyseur en Java (cf. Analyseur.java dans le repertoire principal)

SEG2506

RAPPORT
LABORATOIRE 3

PARTIE 2 : l'outil Lex

Auteurs :

Boniface MBOUZAO
4275992

BOUGUENG TCHEMEUBE Renaud
4333634

Évaluateur : **Robin Tropper**

Lundi, le 28 Janvier 2008

IV- Introduction

Il s'agissait dans cette partie d'implémenter un analyseur d'adresses de courriel en utilisant l'outil LEX. Mais tout d'abord, nous devions nous familiariser avec le langage, la notation particulière, et le processus de compilation de cette puissante application. Nous nous sommes exercés à concevoir des analyseurs simples pour les cinq expressions régulières de la partie 1; histoire de s'habituer à l'environnement de développement et à cerner les caractéristiques clés de ce système qu'est LEX.

V- Description de l'expression régulière retenue

- Une adresse de courriel valide est formée d'une partie dite « **local** » et partie dite « **domaine** » toutes deux séparées par le symbole d'arobase « **@** ».
- La partie locale se constitue d'un ou de plusieurs caractères incluant les caractères alphanumériques (minuscules a-z et majuscules A-Z et chiffres 0-9), et les caractères spéciaux suivants : !, #, $, %, *, /, ?, |, {, }, `, ~, &, ‘, +, =, _, -.
- La partie domaine est formée d'un ou de plusieurs caractères incluant les caractères alphanumériques (minuscules a-z et majuscules A-Z et chiffres 0-9) et le tiret (-).
- La partie locale peut contenir au plus un seule caractère « point » (.) qui ne peut se trouver en début ou en fin de la chaine de caractères.
- La partie domaine doit contenir au moins un caractère « point » (.) qui ne peut se trouver en début ou en fin de sa chaine de caractère.
- La partie locale et la partie domaine ne peuvent être vides.

VI- Description de la spécification pour Lex

Après avoir évalué les exigences ci-dessus, nous avons obtenus la spécification pour Lex:

Votre 'local part' accepte des caractères invalides. -4

```
ALPH        [a-zA-Z0-9]
OTHER
["!""#""$""%""*""/""?""|""{""}""`""~""&""'""+""=""_"]

LOCALFORM   ({ALPH}|{OTHER}|"-")+
LOCALPART   {LOCALFORM}("."{LOCALFORM})?

DOMAINFORM   ({ALPH}|"-")+
DOMAINPART   {DOMAINFORM}("."{DOMAINFORM})+

{LOCALPART}"@"{DOMAINPART}
```

'_' devrait entrer dans LOCALFORM. Pourquoi limiter le nombre de '.{LOCALFORM}à 1 ? -2

Une addresse IP numérique ne contient que 4 nombres de 0 à 255 séparés par points. Votre logiciel ne les valide pas. Pas exigé par le devoir alors pas de pénalité.

VII- Problèmes rencontrés et leçons apprises

Pour arriver à l'élaboration de cet automate, nous a
nombreux obstacles.

Tout d'abord, le souci de spécification. En effet, il n'était pas claire quelle était la forme exacte que doit avoir une adresse courriel, et ce, malgré les informations supplémentaires apportés par l'assistant. Aussi, nous sommes-nous référés à une page web de source sûr que nous avons pris comme modèle de définition des exigences de format d'une adresse courriel: http://en.wikipedia.org/wiki/Email_address.

Ensuite, après avoir réunies les exigences, l'élaboration de l'expression régulière appropriée a plutôt été une formalité.

Enfin, lors de l'implémentation dans FLEX, nous avo
avec la notation dont la complexité, comme nous nous en son
de difficulté proportionnelle à la cardinalité de l'alphabet util
au niveau de l'anlyseur. Après quelques 30 minutes de débogages entres guillemets, parenthèses et crochets, nous avons finalement assimilés comment ca fonctionne.

30 minutes? Rien-là: ma solution doit avoir pris 4 heures.

Le dernier problème, que nous n'avons pas réussi à résoudre, est que certaines adresses de courriel invalides peuvent contenir des sous chaines de caractères d'adresses valides (Exemple : seg@seg@qc.ca est invalide mais contient seg@qc.ca qui est valide). D'après l'énoncé du devoir, l'analyseur devrait simplement les rejeter. Mais, nous n'avons pas réussi à faire cela. Car pour ca, il fallait être capable de:

- soit suivre la trace pas à pas de l'analyseur pour passer à la chaine suivante lorsqu'on se rend compte que la chaine courante est invalide
- soit d'avoir une référence vers les positions précédentes du pointeur de l'application pour savoir si l'adresse que nous prenons comme solution valide, n'est pas une sous chaine d'une chaine invalide. Bref, nous n'avons pas réussi, malgré nos recherches à implémenter cela en utilisant LEX.

VIII- Conclusion

Finalement, non seulement, nous avons appris à construire des automates à partir d'expressions régulières et vice-versa; mais nous avons aussi pu découvrir la puissance et la convenance de l'outil LEX. Il permet de générer un code C (ou C++) (puissance) simulant un automate que nous aurons préalablement défini suivant un langage plus souple et plus approprié (convenance) à l'analyse lexicale et syntaxique.