Théorie des Langages de Programmation

Nom Chinois: Wang Fu
Nom Francais: Francois
No. 516261910014

2020 年 1 月 1 日

Contents

1 Introduction

Tous les codes sont écrit dans C, et j'ai les exécutésur Ubuntu 18.04, comme présentédans la Figure 1.

Dans les trois analyseurs, je présente une raison brutale si l'analyse ne peut pas réussir, la lange de cette raison est possible anglais. Mais tous les commentaires sont écrit en francais. Et vous pouvez choisir est-ce que vous voulez entrer dans la mode **debug**. Les fichers *.c* sont les fichers avec *main.c* pour tester, et les fichers *.h* sont les fichers des functions.

2 Analyse Lexicale

2.1 Idée

Dans cette partie, j'ai utiliséun automate pour réaliser l'analyseur lexicale. On peut considérer que chaque lettre dans l'entrée suit une transition àun autre état. Pour supprimer les commentaires(de format //... ou /*...*/), il est évident que par entrer le sign "/" ou "*", l'automate arrive aux états de commentaires s'il y a un "/" avant. La sortie d'un des ces états est le fin de ligne, d'autre état est "*/". Alos j'ai désigne un automate qui correspond àcette situation.

Ensuite, pour examiner est-ce que le ficher d'automate est correct lexicalement, j'ai construit un autre automate. L'état de cet automate change aussi selon le lettre d'entrée. Pour vérifier les charactères chinois(Seulement "零, 一, 二, 三, 四, 五, 六, 七, 八, 九, 十") et la flèche, je détecte le ASCII code de chaque lettre lit, si le code est inférieur àzéro, le logiciel lit ensuite deux lettres et justifier est-ce que ces trois lettres est la représentation d'un charactère chinois. Pour les mot réservé("Automate", "etats", "initial", "final", "transitions"), on peut les considérer comme une liste d'états. Si le logiciel lit les premiers lettres de ces mots en dehors d'un pair de guilletmets, l'automate entre le premier état de cette liste. Alors c'est facile de faire l'analyse lexicale. Et après ces processus, un ficher *tmp.txt*, c'est la lexème du ficher d'automate qui est utilisédans l'analyse suivante.

Vous pouvez compiler *AnalyseurLexical.c* pour tester cette analyse. Et tous les functions sauf le *main()* function sont écrit dans *AnalyseurLexical.h*.

2.2 Functions

- *checkNum(int a)*

Cette function examiner est-ce qu'un lettre de ASCII code a est un chiffre. Si c'est un chiffre, elle rentre 1, sinon, elle rentre 0.

• *checkAlphabet(int a)*

Cette function examiner est-ce qu'un lettre de ASCII code a est un lettre latin. Si c'est un lettre latin, elle rentre 1, sinon, elle rentre 0.

• *checkSign(int a)*

Cette function examiner est-ce qu'un lettre de ASCII code a est un sign légal dans le lexème. Si c'est un sign légal, elle rentre 1, sinon, elle rentre 0.

• *checkSpe(int a[])*

Cette function examiner est-ce qu'un caractère de UTF-8 code (a[0],a[1],a[2]) est un charactère chinois ou la flèche. Si c'est oui, elle rentre 1, sinon, elle rentre 0.

• *AnalyzeLexical(char dir[])*

Cette function fait l'analyze lexicale d'automate de ficher de qui le nom est enregistrer dans *dir[]*. Elle rentre 0 si l'analyze réussi et 0 s'il y a une erreur lexicale. Et elle donne une raison brutale de cette erreur.

3 Analyse Syntaxique

3.1 Idée

Dans cette partie, j'ai utiliséun automate pour faire l'analyze syntaxique. L'analyseur prend *tmp.txt* générépar l'analyseur lexical comme l'entrée. Comme la format du ficher d'automate est défini strictement, le logiciel peut changer l'état en considérant le lettre d'entrée. Je vérifie les problèmes des parenthèses, des crochets et des accolades dans cette partie. Vous devez d'abord donner les noms d'états. Vous pouvez donner l'état initial, l'état final et les transitions dans un ordre quelconque. Et vous pouvez omettre les parties des piles dans la description d'une transition s'il n'y a pas de manipulation. Pour les autres grammaires, vous devez respecter strictement la format des fichers donnés comme *Dpile.txt*.

Cet analyseur construit deux autres fichers temporel. Le premier est *info.txt* qui contient l'arbre syntaxique de cet automate. L'autre est *tmp1.txt* qui peut faciliter l'entrée d'analyseur sementique.

Vous pouvez compiler *AnalyseurSyntaxique.c* pour tester cette analyse. Et tous les functions sauf le *main()* function sont écrit dans *AnalyseurSyntaxique.h*.

3.2 BNF

La grammaire BNF de mon automate est définie comme:

• <lettre>::=“a” | “b” | ... | “z” | “A” | “B” | ... | “Z”
• <chiffrenn>::=“1” | “2” | “3” | ... | “9”
• <chiffre>::=“0” | <chiffrenn>
• <chinois>::=“零” | “一” | “二” | “三” | “四” | “五” | “六” | “七” | “八” | “九” | “十”
• <special>::=<chinois>| “→”
• <naturel>::=<chiffre>|<chiffrenn>{<chiffre>}
• <naturels>::=<naturel>|<naturels>“,”<naturel>
• <guillemet>::=“ ‘ ” | “ ” ”
• <numpile>::=“Automate=(” “0”|“1”|“2” “)”
• <initial>::=“initial=”<naturel>
• <chartrans>::=<guillemet>(<chiffre>|<lettre>)<guillemet>
• <translettre>::=
<naturel>“→”<naturel>“,”<guillemet><chartrans><guillemet>
• <pilein>::=“ (→, ” <chartrans>“)”
• <pileout>::=“(”<chartrans>“,→)”
• <manipulation>::=“,”(<pilein>| <pileout>|“()”)
• <transition>::=“(”(<translettre>| <translettre><manipulation>|
<translettre><manipulation><manipulation>)“)”
• <transitions>::=<transition>|<transitions>“,”<transition>
• <mot>::=<lettre>|<lettre><mot>
• <etat>::=<guillemet>(<mot>|<naturel>| <chinois>)<guillemet>
• <etats>::=<etat>| <etats>“,”<etat>
• <expression>::=((“etats”|“final”|“transitions”)“ =[”(<naturels>|
<etats>|<transitions>)) | ” initial= ”<naturel>
• <expressions>::=<expression>|<expressions><expression>
• <automate>::=<numpile>“{”<expressions>“}”

3.3 Arbre Syntaxique

3.4 Functions

4 Analyse Sementique

5 Execution