Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Ivan Vragović

Seminarski rad iz predmeta Prevođenje programskih jezika

Zadatak broj 186.

Zagreb, siječanj 2009.

Seminarski rad iz predmeta Prevođenje programskih jezika

Student: Ivan Vragović

Matični broj studenta: 0036428045

Zadatak broj 186: Pomoću programa *LEX* ostvariti **leksičku analizu jezike WSDL.** Potrebno je podržati osnovne konstrukte, te riješiti nejednoznačnosti koje se pojavljuju u leksičkoj analizi. Izlaz iz programa su četiri tablice leksičke analize.

Uvod

Leksički analizator

Leksička analiza je prvi korak rada u jezičkom procesoru. Njegov zadatak je pripremiti ulazni tekst za obradu u sintaksnom analizatoru.

Leksički analizator Leksički analizator odbacuje sve znakove koji se ne koriste u daljnjim fazama rada jezičnog procesora. Iz izvornog programa izbacuju se komentari te svi znakovi koji određuju tekstualnu strukturu programa - znak bjeline, tabulatori, znak novog reda (LF), znak vraćanja na početak reda (CR) itd.

Algoritam određivanja klase leksičkih jedinki jest središnji algoritam leksičkog analizatora. Pripadnost pojedine leksičke jedinke nekoj klasi određena su pravilima koja se definiraju ovisno o načinu izvedbe (više o tome u podpoglavlju "LEX"). Na primjer u većini slučajeva su identifikatori definirani nizom znakova i slova ukoliko je krajnje lijevi znak u nizu slovo.

Još jedan od zadataka leksičkog analizatora jest određivanje mjesta pojave grešaka te prijava istih. Način na koji sam to ostvario biti će opisan u poglavlju "Ostvarenje". Postupak otkrivanja greške izravno je povezan sa oporavkom od iste.

Leksički analizator generira niz leksičkih jedinki koje čita sintaksni analizator kao niz kodnih znakova koji se nazivaju uniformni znakovi. Leksičke jedinke grupiraju se u klase te, najčešće, prema nazivlju klasa bivaju pohranjeni u tablicu uniformnih znakova uz kazaljku. Kazaljka koja je pohranjena uz oznaku za pripadajuću klasu ukazuje na poziciju leksičke jedinke u pojedinoj tablici određenoj klasom.

Primjer:

tablica KROS
ako
onda

tablica uniformnih znakova	
KROS	2
KROS	1

Ove tablice upućuju da je ulaz u leksički analizator bio "onda ako" sa eventualnim komentarima i bjelinama. Osim tablica KROS (ključne riječi, operatori i specijalni znakovi)

postoje još tablice klasa identifikatora i konstanti za koje vrijede ista pravila pri leksičkoj analizi osim što se dinamički stvaraju prilikom čitanja s ulaza i analize.

Na nejednoznačnost u leksičkoj analizi se nailazi u, na primjer, programskim jezicima koji ne zahtijevaju odvajanje leksičkih jedinki bjelinama ili nekim drugim posebnim znakovima. WSDL nema nejednoznačnosti pri leksičkoj analizi, barem ne nekih koje je moguće pronaći u literaturi ili pomoću moćnijih pretraživača interneta.

LEX - *Lexical Analyser Generator*

Kao što i podnaslov predlaže, za poznavatelje engleskog jezika, LEX je program koji omogućuje automatiziranu izgradnju leksičkog analizatora. LEX, unatoč tome što značajno olakšava izradu leksičkog analizatora i generiranje koda za isti, ima nedostatke od kojih mogu spomenuti loš indikator na pogreške u samom kodu LEX-a.

LEX koristi programski jezik C za izgradnju leksičkog analizatora te se i u samom LEX kodu u odgovarajućim blokovima smiju koristiti C naredbe i priključiti C biblioteke.

Format LEX datoteke:

Regularne definicije

%{

Deklaracije

%}

Regularne definicije

%%

Pravila prevođenja

%%

Pomoćne procedure

U dijelu *Deklaracije* potrebno je deklarirati glavni program `main()` (ukoliko nije deklariran u nekoj drugoj datoteci) koji poziva funkcijski potprogram `yylex()`, tj. leksički analizator.

Primjer potpune LEX datoteke:

```
slovo      [a-zA-Z]
%{
#include <stdio.h>
int main()
{
     yylex();
}
int br=1;
%}
%%
slovo      { printf ("Slovo "); }
[0-9]      { printf ("%d", br); }
[ \n]      { br++;}
%%
```

Ovo je jednostavan leksički analizator bez ijedne pomoćne procedure. Kada na ulazu pročita slovo ovaj leksički analizator će ispisati riječ "Slovo" i razmak, a kada pročita znamenku ispisat će broj reda u kojem se ta znamenka nalazi. Dakako ovaj leksički analizator nema neku praktičnu primjenu, ali za demonstraciju je dovoljan.

WSDL

WSDL dokument je formatiran u XML specifikacijama. Drugim riječima WSDL dokument je XML koji odgovara određenim specifikacijama. XML elementi definirani u WSDL-u su

- `<wdsl:types>`
- `<wsdl:message>`
- `<wsdl:operation>`
- `<wsdl:portType>`
- `<wsdl:service>`
- `<wsdl:port>`
- `<wsdl:binding>`

Uz njih također se može naići na SOAP poruke i XML schema definicije.

Ovo mi je glavni vodič gdje i kako tražiti ključne riječi za WSDL budući da nisam dobio popis istih niti sam ga uspio naći te zato ne mogu garantirati da sam našao sve ključne riječi.

Ostvarenje

Zadatak je ostvariti leksičku analizu jezika WSDL pomoću programa LEX. Sam program LEX i njegove osnovne značajke opisane su u poglavlju "Uvod", podpoglavlje "LEX". Format datoteke LEX je preuzet iz primjera iz knjige "Prevođenje programskih jezika". Sam kod LEX datoteke je ostvaren samostalno jer knjiga niti ne sadrži neke kompliciranije primjere.

ULAZ

Ulaz se ostvaruje preko standardnog ulaza (stdin). Ulaz za izvršnu datoteku je dakako WSDL dokument odnosno kod pisan WSDL jezikom. Program je testiran samo na operacijskom sustavu Windows, ali je preveden pomoću open-source programa (flex i gcc) čiji ekvivalenti postoje i za Linux, dapače u originalu su i razvijeni za Linux, te nema razloga da ne radi i pod Linux-om čime bi se mogla datoteka s WSDL kodom proslijediti na standardni ulaz umjesto upisivanja WSDL koda na standardni ulaz. Unos se u komandnoj liniji prekida prekidanjem izvršavanja programa (Ctrl + C).
Primjer ulaza (nije potrebna sintaksna točnost pa koristim kraći ulazni primjer, koji ne mora nužno biti i sintaksno ispravan):

```
<?xml version="1.0" encoding="UTF-8"?>
```

ALGORITAM

Algoritam nije potrebno smišljati budući da je LEX napravljen tako da olakša ovu vrstu posla. Dakle ono što je bilo potrebno je:

1. naći ključne riječi WSDL-a
2. osmisliti oporavka od greške i prijave greške

3. osmisliti način ispisa imajući na umu da kazaljke u tablici uniformnih znakova trebaju pokazivati na odgovarajuće elemente u tablici pojedine klase (eventualno izuzev konstante)
4. osmisliti način ignoriranja komentara

1. KLJUČNE RIJEČI - Kao što sam već ranije napomenuo, nije lako naći bazu ključnih riječi za WSDL te sam na kraju bio primoran tražiti različite primjere kodova WSDL-a te iz njih vaditi ključne riječi. Ne moram niti reći koliko je to neučinkovit te nadasve zatupljujuć način, ali nisam našao drugi.

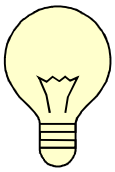
2. GREŠKE - Iako je pod brojem 2, ovo sam posljednje riješio, ne zbog težine već zbog logičnog slijeda rješavanja. Način je bio taj da uzimam znakove za koje nije pronađen niti jedan odgovarajući regularni izraz te prijaviti linije koda u kojoj se greške nalaze te ispisati znakove koji se nalaze u greškama.

3. ISPIS - Ovaj dio mi je zadavao najveće muke. Na razmišljanje o zapisu u polja me naveo primjer korišten u knjizi "Prevođenje programskih jezika" na stanici 69. Problem nastupa u tome što je u knjizi korišteno statičko polje, no ja bih morao upotrijebiti dinamičko, ali i tada bi kod većih programa moglo doći do nedostatka memorije, pogotovo ukoliko određeni operativni sustav ili određeni prevodioc ima malo ograničenje na memoriju koju smije koristiti. Još jedan od problema kod polja je konkretan ispis. Pretpostavljam ispis na standardni izlaz (stdout) što nikako nije dobra ideja pogotovo s obzirom na preglednost.

Slijedeća ideja se odmah nameće, a to je ispis u datoteku. Budući da nisam ograničen brojem datoteka koje smijem koristiti, zašto ne upotrijebiti više datoteka? Točnije, upotrijebit ću po jednu datoteku za svaku od tablica. Ova ideja mi savršeno odgovara jer ne samo da dobijem na preglednosti izlaza, već mi je znatno olakšano i samo kodiranje LEX programa.

4. KOMENTARI - Ovaj dio nije bio pretežak. Bilo je potrebno neko vrijeme da se snađem u LEX-u da bih mogao to riješiti, ali od tog trenutka je krenulo puno lakše. Kada regularni izraz se poklapa s znakovima za početak komentara ("<!--") uđe se u pomoćnu proceduru `comment()` u kojoj se uzimaju tri znaka ukoliko prvi odgovara znaku "-" koji je prvi od znakova za kraj komentara ("-->"). Ukoliko se ne poklapaju sva tri znaka sa znakovima "-->" tim redoslijedom, znakovi se vraćaju u ulazni niz. Taj dio je bio kompliciran dok nisam saznao za naredbe kojima uzimam po jedan znak ili isti vraćam na ulaz.

Dakle riješio sam sve očigledne probleme no daljnjim kodiranjem sjetio sam se da moram riješiti problem kazaljki u listi uniformnih znakova, barem za identifikatore, budući da su KROS unaprijed definirane s njima nema problema jer sam siguran na kojoj će se poziciji nalaziti u KROS tablici te mogu odmah taj broj zapisati kao kazaljku u listu uniformnih. Za konstante nije nužno da riješim taj problem jer znam da se ne inzistira na pamćenju konstanti.



Ideju sam dobio kada sam malo proučavao načine otvaranja datoteka u C-u. Primijetio sam da ako otvorim datoteku s opcijom "a+" da ću moći dodavati na kraj datoteke bez obzira gdje sam prije toga čitao u datoteci čime ću si mnogo olakšati posao. Također sam našao funkciju `rewind()` koja pokazivač u datoteci vraća na početak. Dakle ideja je da datoteku "iden.txt", u koju spremam identifikatore, pri svakom pronalasku identifikatora postavim na početak za čitanje. Pretražim postoji li već takav identifikator. Ako postoji zapamtim redak u kojem se nalazi u datoteci. Ako ne postoji dodam ga na kraj datoteke i zapišem pod kazaljku u tablici uniformnih znakova broj redaka datoteke. Kad sam riješio problem s pamćenjem kazaljke za identifikatore jednostavno sam napravio istu funkciju i za konstante te time dodao podršku za pamćenje konstanti.

IZLAZ

Izlaz je u četiri datoteke:

- iden.txt → tablica identifikatora
- kons.txt → tablica konstanti
- kros.txt → KROS tablica
- unif.txt → tablica uniformnih znakova

Svaka od datoteka sadrži što jedinke koje odgovaraju klasi te tablice. Jedinke su odvojene znakom novog reda. Jedina drugačija datoteka je unif.txt jer se kod nje u redu nalazi i kazaljka koja ukazuje na red u datoteci te klase u kojem se nalazi leksička jedinka.

PRIMJER

Ulaz:

```
<?xml version="1.0" encoding="UTF-8"?>
^C
```

Izlaz:

unif.txt - tablica uniformnih znakova

```
KROS 71
KROS 6
IDN 1
KROS 73
KON 1
IDN 2
KROS 73
KON 2
KROS 75
```

kons.txt - tablica konstanti

"1.0"
"UTF-8"

iden.txt - tablica identifikatora

version
encoding

kros.txt - KROS tablica - zbog preglednosti u stupcima

description	definitions	=
service	address	/>
endpoint	xsd	?>
binding	xs	>
operation	xsi	:
xml	pattern	/
schema	whiteSpace	
element	length	
complexType	fractionDigits	
choice	maxExclusive	
any	maxInclusive	
anyAttribute	maxLength	
sequence	minExclusive	
annotation	minInclusive	
documentation	minLength	
key	totalDigits	
selector	simpleContent	
field	maxOccurs	
extension	minOccurs	
attribute	name	
unique	attributeGroup	
complexContent	substitutionGroup	
group	base	
port	appInfo	
types	import	
message	include	
portType	keyref	
simpleType	list	
enumeration	notation	
restriction	redifine	
all	union	
wsdl	body	
input	type	
output	</	
fault	<?	
part	<	

Zaključak

Što se činilo kao prilično jednostavan zadatak, zakompliciralo se raznim problemima. Dakako naviše je pomoglo definiranje konačnog oblika ispisa tih tablica. Naravno problematično je i programiranje u LEX-u u početku. Učinkovito ostvarenje leksičkog analizatora moguće je postići isključivo dobrim poznavanjem rada LEX-a, što, budimo realni, nije slučaj kod studenta koji je tek ovaj semestar prvi put vidio LEX.

Ispravnost rezultata, nažalost, nikako ne može biti zagarantirana zbog već prije spomenutog problema nedostatka popisa ključnih riječi programskog jezika WSDL što ne, blago rečeno, zagorčalo ovaj zadatak u potpuno nepotrebnom smjeru i vjerujem da to nikako nije bio cilj ovog samostalnog projekta.

Olakšavajuće je što WSDL nije imao nejednoznačnosti na razini leksičke analize, stoga je dodatno vrijeme utrošeno pronalaskom ključnih riječi barem malo umanjeno budući da nije bilo potrebno smišljati ili tražiti algoritam rješavanja nejednoznačnosti.