Predmet: Automati, formalni jezici i jezični procesori I
Školska godina: 2000/2001.

ZEMRIS

Dinko Korunić
0036355514

Anđelko Iharoš
0036355096

Laboratorijske vježbe

Dokumentacija za prvu vježbu

U ovom dokumentu se nalazi dokumentacija za prvu laboratorijsku vježbu iz predmeta **Automati, formalni jezici i jezični procesori I**, koja se bavi konstrukcijom nedeterminističkih i determinističkih konačnih automata i njihovim simuliranjem:

** 1. Konstruirati NKA (nedeterministički konačni automat) koji će prihvaćati sve moguće zapise cijelih i realnih brojeva te ga programski realizirati. Program iz datoteke čita pojedine zapise koji mogu biti odvojeni prazninama, tabulatorima te znakovima za novi red. Za svaki zapis program treba provjeriti pripada li u zadanu klasu te to ispisati u posebnu datoteku.*

PRIMJER:

Cijeli brojevi:	*22*	*0*	*8*	*-112*	*+0*
Realni brojevi:	*2.5*	*.5*	*0.0*	*2e-8*	*+3.E+4* *55.*
Ne prihvaća:	*.*	*023*	*.e4*	*2e-8.2*	

** 2. Konstruirati generator NKA koji će prepoznavati riječi jezika koje su navedene u ulaznoj datoteci. Za zadanu ispitnu datoteku generirani NKA određuje koje jedinke pripadaju zadanom jeziku, a koje ne pripadaju.*

U sklopu svakog zadatka bilo je potrebno i pomoću test datoteke provjeriti ispravnost funkcioniranja automata te usporediti sa predviđenim ponašanjem. Dokumentaciju u općem smislu čini ovaj dotični dokument te komentari unutar samog izvornog koda. Zahtjev prvog dijela prvog zadatka bio je ostvariti NKA simulator, dok je zahtjev u drugom dijelu prvog zadatka bio ostvariti NKA generator.

Uvod

Za programski jezik smo odabrali C isključivo radi subjektivnog preferiranja (i godina provedenih uz *gcc* i *gdb*), iako bi vjerojatno prilagođeniji bio ObjC ili pak C++ zbog objektne orijentiranosti i raznih prednosti (iako je to diskutabilno). U izvedbi programa nismo obraćali previše pažnje potrošnji memorijskog prostora radi ostvarivanje veće brzine rada programa. Optimizacija po brzini nije tražila, no nekakvo bazično promišljanje o brzini izvođenja i obrade podataka je nužno. U današnje vrijeme računala raspolažu većim količinama radne memorije, tako da je logičnije za različite tablice stanja i obrade koristiti memoriju nego kakve privremene datoteke (što zapravo bi značilo neizbježno usporenje rada). Također nismo eksplicitno oslobađali memorijski prostor koristeći se pretpostavkom da dotični program radi na nekom OS-u koji dobije nazad memorijski prostor zauzet programom nakon kraja izvođenja dotičnog. Dodali smo i tzv. *garbage collector* radi boljeg korištenja memorijskog prostora.

Za debugiranja i obradu koristili smo se slijedećim standardnim Unix alatima: *gdb* (debugger), *gcc* (compiler), *ccmalloc* (malloc debugger), *gccchecker* (wrapper za niz funkcija – malloc i memory access debugger), *libefence* (malloc i memory access debugger), te *gprof* (program profiler); na *Debian 2.3 GNU/Linux* platformi kao i na *Sun Sparc Solaris 2.7* platformi. Za unos podataka i izvornog koda koristili smo *vim* (*vi* klon) i *XEmacs* koji se pokazao pogotovo dobrim za RCS projekte. Za poravnavanje i poljepšavanje izvornog koda koristili smo *astyle* program sa ANSI-C stilom te *XEmacs* mogućnosti poravnanja sa GNU stilom. Za upravljanje revizijama programa i njihovim logiranjem koristili smo standardni *GNU RCS* paket.

Kao dodatnu literaturu smo koristili niz dokumenata koje je moguće naći na Internetu (bilo da je riječ o W3 ili pak *Usenet*-u). Također su vrlo iscrpne bili radovi studenata na MIT-u glede konačnih automata, kao i knjiga *R. Hall: Art of Assembly Programming* (poglavlje 16), te FAQ *comp.compilers* grupe kao i arhive članaka koje su se pojavljivale na dotičnoj grupi.

Izvedba
(programska implementacija i opis rješenja)

Promislivši o implementaciji dotičnog NKA automata odlučili smo se za eksplicitno zadan automat sa funkcijom prijelaza koja je ostvarena vektorskim pristupom. Time smo dobili na brzini izvođenja programa, no program je stoga poprilično "gladan" za memorijskim prostorom. Dakle, tablica stanja automata je vektor po kojem se može proizvoljno pozicionirati znajući koji je po redu željeni znak u nekom vektoru dozvoljenih znakova. Pri tome je sa 'x' (*#define BLANK 'x'*) definiran prazan skup stanja (dakle da daljnji slijed stanja ne postoji bez obzira na ulazne znakove), kao i sa '*' (*#define EPSI '*'*) definiran epsilon prijelaz (dakle prijelaz bez ulaznih znakova). Kao što je već rečeno, tablica stanja automata je vektor struktura:

```
typedef struct state_struct
{
        char *nextstate; /* Niz znakova odijeljenih ',', NFA! */
        int isfinal; /* Doticno stanje je prihvatljivo ili ne, 0/!0 Takodjer,
                        1 oznacava prvu klasu, 10 drugu, 100 trecu.. radi
                        koristenja OR */
}
state_type;
```

pri čemu je *char *nexstate* pokazivač tipa *char* na niz znakova koji su niz stanja odijeljenih zarezom. Važno je primijetiti da smo u tablicama stanja u datotekama (koje oba programa koriste – jedan kao generator i jedan kao simulator) više dozvoljenih stanja za jedno polje također odvajali sa ',' a polja međusobno sa ';' što nam se činio razuman izbor odjeljujućih znakova. No, korištenje dotičnih delimitera stavilo nas je pred dvojbu: koristiti li *strsep()* ili *strtok()* C-funkcije? Naime, *strsep()* radi upravo ono odjeljivanje koje nam treba, i može raditi čak i ako naiđe na polja koja su recimo prazna (NULL), no dotična funkcija nije prema ANSI-C standardu. A kao alternativa tu je *strtok()* koji nije preporučljivo koristiti jer mijenja vrijednost prvog argumenta, ne može se koristiti sa praznim poljima i ima niz dodatnih nedostataka. Stoga smo implementirali vlastitu funkciju za odjeljivanje delimitera i vraćanje tzv. tokena:

```
char *strtoken (char **save, char *str, char *fs);
```

Osim toga, za stvaranju datoteka dozvoljeno je da korisnik može koristiti delimitere tipa tab, space i nextline. Dotični ne utiču na rad programa, no povećavaju preglednost ulazno/izlaznih datoteka programa, tako da je moguća i interakcija korisnika za tim datotekama. Čišćenje nizova ulaznih znakova koje programi pročitaju iz takvih datoteka ostvarili smo funkcijom

```
char *purify (char *string);
```

koja čim naiđe na neki delimiter od gore navedenih kopira niz znakova počevši od dotičnog do kraja niza preko tog delimitera i time pomiče kraj niza za jedan 'ulijevo'. Naravno da se dotična funkcija dala inteligentnije i spretnije napisati, no ovdje nije bio naglasak i cilj vježbe na njoj.

Također je bitno primijetiti da je za NKA simulator preporučljivo, ako ne i nužno izvršiti transliteraciju odnosno pojednostavljivanje ulaznih nizova čime se dobiva efikasniji zapis tablice stanja i samim time puno brži rad NKA simulatora. Za izvršavanje dotičnog smo koristili slijedeću funkciju:

```
void trans_string(char *oldstring);
```

Algoritmi vježbi

Pseudokod za NKA simulator:

```
inicijaliziraj početne vrijednosti varijabli
ako trenutni učitani znak ne označava kraj
        ako učitani znak nije u tablici dozvoljenih izadji sa 0
        nadji niz slijedećih stanja i pridruži ga varijabli
        parsiraj niz stanja sve do kraja niza i za svako dobiveno stanje
                ako stanje vodi dalje pozovi simuliranje sa slijedećim ulaznim znakom
inače saznaj da li je trenutno stanje prihvatljivo i vrati tu vrijednost
```

Pseudokod za ε-NKA je nešto složeniji:

```
inicijaliziraj početne vrijednosti varijabli
ako postoji epsilon prijelaz u tablici dozvoljenih znakova
        nadji niz slijedećih stanja i pridruži ga varijabli
        parsiraj niz stanja sve do kraja niza i za svako dobiveno stanje
                ako stanje ne vodi dalje
                        saznaj da li je trenutno stanje prihvaljivo
                                ako je stanje prihvatljivo i nije kraj ulaznog niza
                                        vrati 0
                        inače
                                pozovi simuliranje sa istim ulaznim znakom
ako trenutni učitani znak ne označava kraj
        ako učitani znak nije u tablici dozvoljenih
                izadji sa 0
        nadji niz slijedećih stanja i pridruži ga varijabli
        parsiraj niz stanja sve do kraja niza i za svako dobiveno stanje
                ako stanje vodi dalje
                        pozovi simuliranje sa slijedećim ulaznim znakom
inače
        saznaj da li je trenutno stanje prihvatljivo i vrati tu vrijednost
```

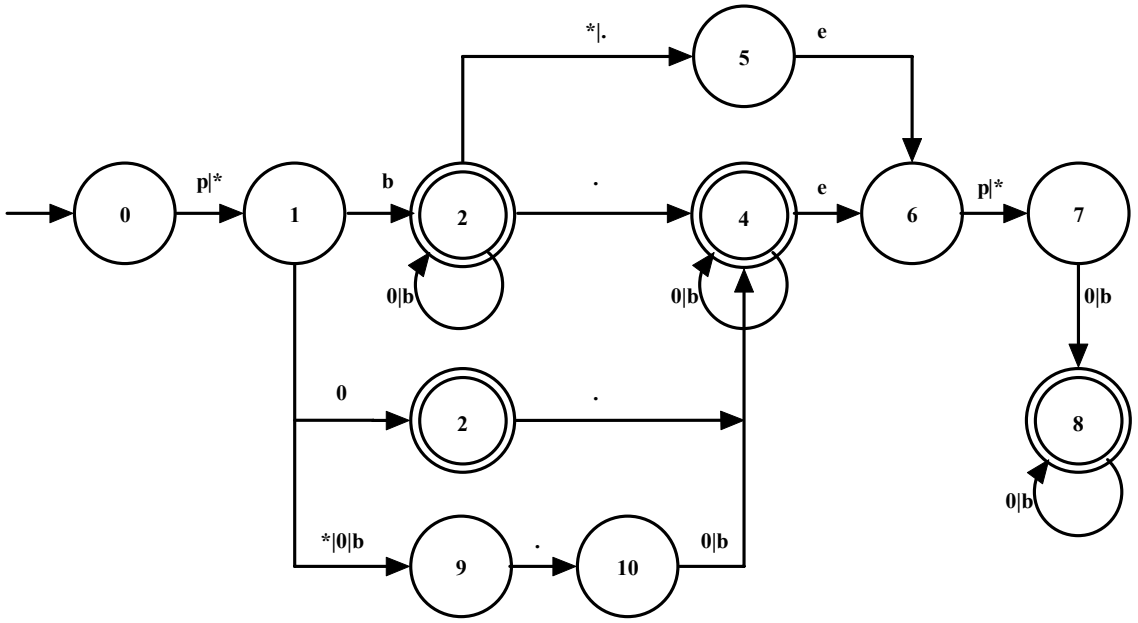
Pseudokod za NKA generatora:

```
inicijaliziraj početne vrijednosti varijabli
dok ima ulaznih znakova
        nadji niz slijedećih stanja pomoću trenutnog stanja
        ako niz slijedećih stanja označava da nema slijedećih
                obriši taj znak
        inače
                dodaj na niz slijedećih stanja delimiter za slijedeće stanje
        povećaj maksimalno do sada dobiveno stanje
        pridruži tu vrijednost trenutnom stanju
        dobiveno stanje dodaj na niz slijedećih stanja
označi trenutno stanje uvećano za jedan finalnim stanjem
```

Tablica stanja za NKA simulator – automat koji prepoznaje cijele i realne brojeve te ih razlikuje što je i traženo u tekstu zadatka:

stanje	**p**	*****	**0**	**b**	**e**	**.**	prihv.
0	{1}	{1}	∅	∅	∅	∅	0
1	∅	{9}	{3,9}	{2,9}	∅	∅	0
2	∅	{5}	{2}	{2}	∅	{4,5}	1
3	∅	∅	∅	∅	∅	{4}	1
4	∅	∅	{4}	{4}	{6}	∅	10
5	∅	∅	∅	∅	{6}	∅	0
6	{7}	{7}	∅	∅	∅	∅	0
7	∅	∅	{8}	{8}	∅	∅	0
8	∅	∅	{8}	{8}	∅	∅	10
9	∅	∅	∅	∅	∅	{10}	0
10	∅	∅	{4}	{4}	∅	∅	0

Dijagram stanja dotičnog NKA:



Pri čemu je bitno primijetiti da su 0, p, * i . zadani znakovi dobiveni transliteracijom po navedenom pravilu:

p+;
b123456789;
eE;

što zapravo znači: zamijeni sve znakove u ulaznom nizu tako da se uzme prvi znak u svakom polju kao relevantni znak te da se zamijene oni znakovi koji slijede iza prvog sa sa prvim znakom, i tako za sva polja. Svako polje transliteriranja je odvojeno delimiterom (';').

Dakle, naveni program za ulazne podatke:

0 8 -112 +0
2.5 .5 0.0 2e-8 +3.E+4 55.
. 023 .e4 2e-8.2

daje slijedeći izlaz:

22 bb se prihvaca sa 1
0 0 se prihvaca sa 1
8 b se prihvaca sa 1
-112 pbbb se prihvaca sa 1
+0 p0 se prihvaca sa 1
2.5 b.b se prihvaca sa 10
.5 .b se prihvaca sa 10
0.0 0.0 se prihvaca sa 10
2e-8 bepb se prihvaca sa 10
+3.E+4 pb.epb se prihvaca sa 10
55. bb. se prihvaca sa 10
. . se ne prihvaca
023 0bb se ne prihvaca
.e4 .eb se ne prihvaca
2e-8.2 bepb.b se ne prihvaca

Što odgovara očekivanom rezultatu.

Ako sad promotrimo generator NKA na slijedećoj ulaznoj datoteci:

amid
abid
amidy

dobivamo slijedeću tablicu stanja:

1,5,9;	*x;*	*x;*	*x;*	*x;*	*x;*	*0;*
x;	*2;*	*x;*	*x;*	*x;*	*x;*	*0;*
x;	*x;*	*3;*	*x;*	*x;*	*x;*	*0;*
x;	*x;*	*x;*	*4;*	*x;*	*x;*	*0;*
x;	*x;*	*x;*	*x;*	*x;*	*x;*	*1;*
x;	*x;*	*x;*	*x;*	*6;*	*x;*	*0;*
x;	*x;*	*7;*	*x;*	*x;*	*x;*	*0;*
x;	*x;*	*x;*	*8;*	*x;*	*x;*	*0;*
x;	*x;*	*x;*	*x;*	*x;*	*x;*	*1;*
x;	*10;*	*x;*	*x;*	*x;*	*x;*	*0;*
x;	*x;*	*11;*	*x;*	*x;*	*x;*	*0;*
x;	*x;*	*x;*	*12;*	*x;*	*x;*	*0;*
x;	*x;*	*x;*	*x;*	*x;*	*13;*	*0;*
x;	*x;*	*x;*	*x;*	*x;*	*x;*	*1;*

sa ovim oznakama stanja:

amidby

Program možemo provjeriti uvrštavanjem dotične tablice stanja (koja je stoga namjerno kompatibilna) u prvi program definiran prvim dijelom ove laboratorijske vježbe.