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

ZEMRIS

Dinko Korunić
0036355514

Laboratorijske vježbe

Dokumentacija za treću vježbu

U ovom dokumentu se nalazi dokumentacija za treću laboratorijsku vježbu iz predmeta **Automati, formalni jezici i jezični procesori I**, koja se bavi konstrukcijom nedeterminističkih (i determinističkih) potisnih automata i njihovim simuliranjem, te izgradnjoj pripadajuće gramatike:

Metodom potisnog automata izgraditi program koji •e prihva•ati sve ispravno napisane aritmeti•ke izraze u infiks notaciji korištenjem operatora **+, -, *, /, (,), [,], {, }** te operanada (hijerarhija zagrada: **{,[,(**). U izlaznoj datoteci pokazati konfiguraciju potisnog automata za zadani ulazni niz. Na temelju izgra•enog potisnog automata izgradite odgovaraju•u gramatiku.

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 sam odabrao C++ koji koristi neke dijelove koda iz prve vježbe (vlastitu implementaciju `strtoken()` koja je ekvivalentna `strsep()` BSD funkciji). Pokazala se nadasve korisnim i upotreba STLa (Standard Template Library) koji je omogućio dinamičko alociranje memorije, objekte tipa *string* i *vector*.

Za debugiranja i obradu koristili smo se slijedećim standardnim Unix alatima: *gdb* (debugger), *g++* (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.

Izvedba
(programska implementacija i opis rješenja)

```
struct PAData
{
  int State,
    NewState;
  char InputChar,
    Stack,
    NewStack;
};
```

Struktura `PAData` sadrži opis prijelaza - stanje, ulazni znak i trenutni stog (koristi se za lociranje referentnog stanja), kao i ono što se odnosi na sami prijelaz - a to se odnosi na izmjene stoga i prijelaz u novo stanje.

```
typedef struct PAData PAData_t;
typedef vector<PAData_t> PAData_Vector;
typedef vector<char> char_Vector
```

Dotično nam služi za deklariranje vlastitih tipova podataka, radi razumljivosti i jednostavnosti kao i preglednosti izvornog koda.

```
class PA
{
  char_Vector Stack;
  PAData_Vector PATable;
  string TransData;
  int CurrState;
  void ParseString(string);
  string TransliterateString(string);
  PAData_Vector::iterator FindTable(char **);
public:
  void InputTransliterator(istream &);
  int ParseFile(istream &);
  friend ostream& operator<< (ostream &, PA &);
  friend istream& operator>> (istream &, PA &);
};
```

Ova klasa sadržava relevantne podatke i metode. `Stack` je vlastita implementacija stoga, i to je tipa STL vektor. Nadalje, `PATable` je također izvedenica STL vektora, s time da su ćelije strukture `PAData`. Nadalje, `TransData` je STL string koji sadržava podatke za transliterator. `CurrState` je flag trenutnog stanja. Što se samih metoda tiče, one su `ParseString()` za parsiranje individualnih stringova kroz PA; `TransliterateString()` koji vrši samo transliteriranje (koristeći neke vlastite C-oidne funkcije iz prve vježbe). Javno dostupne metode su `InputTransliterator()` koji puni transliterator, te `ParseFile()` koji parsira red po red ulaznu datoteku i poziva `ParseString()` metodu.
Očito je i da su operatori ulaza i izlaza preopterećeni te mogu raditi na PA objektima.

Algoritmi vježbi

PA je zadan slijedećom tablicom prijelaza, koja je definirana ovako (x u oznaci ulaznog znaka označava ε prijelaz, a x kod znaka za novi stog označava da nema promjene na stogu):

stanje	ulazni znak	znak na stogu	novo stanje	novi stog
0	-	K	0	x
0	-	{	0	x
0	-	[	0	x
0	-	(	0	x
0	{	K	0	{
0	[	K	0	[
0	(	K	0	(
0	[	{	0	[
0	(	[	0	(
0	a	K	1	x
0	a	{	1	x
0	a	[	1	x
0	a	(	1	x
0	x	K	2	b
1	+	{	0	x
1	+	[	0	x
1	+	(	0	x
1	+	K	0	x
1	-	{	0	x
1	-	[	0	x
1	-	(	0	x
1	-	K	0	x
1	}	{	1	b
1	]	[	1	b
1	)	(	1	b
1	x	K	2	b

Dotični automat samo dodaje ili skida ili ne dira znak na stogu, stoga su promjene jednoslovne (lako je pokazati da je ekvivalentan onima koji vrše promjene sa više znakova stoga odjednom).

Sama gramatika je PA definirana ovako:

```
S --> [q0,K,q0] | [q0,K,q1] | *
[q0,K,q0] --> - [q0,K,q0] | - [q0,K,q0] | { [q0,{,q0] [q0,K,q0] | {
[q0,{,q1] [q1,K,q0] | [ [q0,[,q0] [q0,K,q0] | [ [q0,[,q1] [q1,K,q0] | (
[q0,(,q0] [q0,K,q0] | ( [q0,(,q1] [q1,K,q0] | a [q0,K,q0] | a [q0,K,q0] | x
| *
[q0,K,q1] --> - [q0,K,q1] | - [q0,K,q1] | { [q0,{,q0] [q0,K,q1] | {
[q0,{,q1] [q1,K,q1] | [ [q0,[,q0] [q0,K,q1] | [ [q0,[,q1] [q1,K,q1] | (
[q0,(,q0] [q0,K,q1] | ( [q0,(,q1] [q1,K,q1] | a [q0,K,q1] | a [q0,K,q1] | x
| *
[q0,{,q0] --> - [q0,{,q0] | - [q0,{,q0] | [ [q0,[,q0] [q0,{,q0] | [
[q0,[,q1] [q1,{,q0] | a [q0,{,q0] | a [q0,{,q0] | *
[q0,{,q1] --> - [q0,{,q1] | - [q0,{,q1] | [ [q0,[,q0] [q0,{,q1] | [
[q0,[,q1] [q1,{,q1] | a [q0,{,q1] | a [q0,{,q1] | *
[q0,[,q0] --> - [q0,[,q0] | - [q0,[,q0] | ( [q0,(,q0] [q0,[,q0] | (
[q0,(,q1] [q1,[,q0] | a [q0,[,q0] | a [q0,[,q0] | *
```

```
[q0,[,q1] --> - [q0,[,q1] | - [q0,[,q1] | ( [q0,(,q0] [q0,[,q1] | (
[q0,(,q1] [q1,[,q1] | a [q0,[,q1] | a [q0,[,q1] | *
[q0,(,q0] --> - [q0,(,q0] | - [q0,(,q0] | a [q0,(,q0] | a [q0,(,q0] | *
[q0,(,q1] --> - [q0,(,q1] | - [q0,(,q1] | a [q0,(,q1] | a [q0,(,q1] | *
[q1,K,q0] --> + [q0,K,q0] | + [q0,K,q0] | - [q0,K,q0] | - [q0,K,q0] | x | *
[q1,K,q1] --> + [q0,K,q1] | + [q0,K,q1] | - [q0,K,q1] | - [q0,K,q1] | x | *
[q1,{,q0] --> + [q0,{,q0] | + [q0,{,q0] | - [q0,K,q0] | - [q0,K,q0] | } | *
[q1,{,q1] --> + [q0,{,q1] | + [q0,{,q1] | - [q0,K,q1] | - [q0,K,q1] | } | *
[q1,[,q0] --> + [q0,[,q0] | + [q0,[,q0] | - [q0,K,q0] | - [q0,K,q0] | ] | *
[q1,[,q1] --> + [q0,[,q1] | + [q0,[,q1] | - [q0,K,q1] | - [q0,K,q1] | ] | *
[q1,(,q0] --> + [q0,(,q0] | + [q0,(,q0] | - [q0,K,q0] | - [q0,K,q0] | ) | *
[q1,(,q1] --> + [q0,(,q1] | + [q0,(,q1] | - [q0,K,q1] | - [q0,K,q1] | ) | *
```

Pseudokod emulatora:

```
početno stanje je nula
stog je prazan
stavi na stog 0
dok postoji prijelaz za trenutno stanje ili trenutno stanje i znak
      prebaci stanje u novo stanje
      za oznaku novog stoga
            ako je b
                  obriši jedan znak sa stoga
            ako je x
                  nemoj ništa učiniti
            za sve ostalo
                  stavi novu oznaku stoga na stog
      ako je nije ε prijelaz povećaj pokazivač na slijedeći ulazni znak
ako je stog prazan i postojali su svi prijelazi niz se prihvaća
```

Primjer ulazne datoteke:

```
a+(a*a)+(a+a)/a
-a/a+{a*[a-a]*[a-(a/a)/a]/a}
a*{a-[-a*(a-a)]*a}
(a/a)
{[()]}
```

Primjer izlaznih podataka (za tu istu ulaznu datoteku):

```
./afjjp1-3 input trans parse
Interni zapis tablice PA:
```

<kraćeno!>

```
Rezultat parsiranja datoteke:
Niz a+(a+a)+(a+a)+a se prihvaca
Prazan stack
Niz -a+a+{a+[a-a]+[a-(a+a)+a]+a} se prihvaca
Prazan stack
Niz a+{a-[-a+(a-a)]+a} se prihvaca
Prazan stack
Niz (a+a) se prihvaca
Prazan stack
Niz {[()]} se ne prihvaca
Na stacku je: K { [ (
```