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

ZEMRIS

Dinko Korunić
0036355514

Anđelko Iharoš
0036355096

Laboratorijske vježbe

Dokumentacija za drugu vježbu

U ovom dokumentu se nalazi dokumentacija za drugu laboratorijsku vježbu iz predmeta **Automati, formalni jezici i jezični procesori I**, koja se bavi minimizacijom deterministčkih konačnih automata i njihovim simuliranjem:

```
* 4. Minimizirati DKA iz prethodne vjezbe. Ispisati
zadani DKA i dobiveni minimalni DKA. Za zadanu ulaznu
datoteku provjeriti ekvivalentnost rada automata.
```

U sklopu 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. Kao DKA simulator su korišteni programi iz prve vježbe.

Uvod

Za programski jezik smo odabrali C++ zbog odličnog *template* (STL) interfejsa koji nam je bitno olakšao korištenje memorije. Posebno moramo spomenuti 2D dinamički rastezljiva polja i metode implementirane nad njima - dinamičko izbacivanje elemenata, dodavanje, *random access*, overloadani operatori kopiranja, usporedbe, itd.

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 posebno 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 osnovnu literaturu smo koristili skripte prof. Srbljića.

Izvedba
(programska implementacija i opis rješenja)

```
typedef vector<int> Vector; // 1d
typedef vector<Vector> Vector2; // 2d
```

U samoj implementaciji algoritma je korišten STL zbog mogućnosti korištenja dvodimenzionalnih vektora, koji su nam na izuzetno jednostavan i portabilan način omogućili dinamičko širenje i skupljanje potrebnih struktura tijekom rada.

```
class Table
{
```

```
       Vector2 DFTable; // tablica stanja
       void  UcitajPosebneZnakove(istream  &);  //  ucitavanje
opisa tablice
      public:
       int OcistiNedohvatljiva(void);
       int Minimiziraj(void);
       int IsteGrupe(Vector2, int, int);
       friend ostream& operator<< (ostream &, Table &);
       friend istream& operator>> (istream &, Table &);
     }; // class Table
```

Kao što je očito, u klasi "Table" su enkapsulirane sve potrebne metode zajedno sa podacima. Za sam ispis klase *overloadan* je operatori ispisa "<<" radi pojednostavljenja i skrivanja metoda u duhu objektnih jezika, kao što je overloadan i operator ">>" radi unosa podataka bilo iz *cin* bilo iz datoteke. "DFTable" je dvodimenzionalni vektor naslijeđen iz STL klasa. Funkcije "OcistiNedohvatljiva()" i "Minimiziraj()" rade očite ekvivalente formalnom zapisu algoritma. Funkcija "IsteGrupe()" nam služi za uspoređivanje grupa što je također zasebna implementacija dijela algoritma.

Algoritmi vježbi

Pseudokod za "`OcistiNedohvatljiva()`"

napravi vektor koji sadrži kao dohvatljivo stanje nulto stanje
dok možeš dodavati u listu dohvatljivih stanja
    za svako stanje iz liste dodaj svako novo stanje u listu za bilo koji znak
sortiraj listu dohvatljivih uzlazno
za svako stanje (redak) iz tabele stanja
    ako stanje nije u listi dohvatljivih
        izbaci ga
kraj

Pseudokod za "`IsteGrupe()`"

za sve ulazne znakove stanja a
    nadji u koje grupe vodi to stanje
    ako vodi u različitu grupu od koje vodi stanje b
        vrati da je u različitim grupama
vrati da je u istim grupama
kraj

Pseudokod za "`Minimiziraj()`"

podijeli stanja na prihvatljiva i neprihvatljiva
dok možeš podijeliti
za svaku podjelu grupa
    za svako stanje pocevsi od drugog do kraja
        ako su prvo stanje i to iterirano stanje nemaju prijelaze u iste grupe
            izbaci to stanje u novu grupu zajedno sa svima izbacenima iz te grupe
za sve grupe u zadnjoj dobivenoj podjeli

ako ima vise od jednog elementa u listi ekvivalentnih
za svaki element u listi ekviv. osim za prvog
dodaj to stanje u listu za izbaciti
zamijeni sve reference u stanjima na nulti element
obrni poredak liste za izbaciti
za sve elemente iz liste za izbaciti
ako je indeks u tekućem retku veci od od onog za izbacivanje smanji indeks za jedan
za sve elemente iz liste za izbaciti
izbaci to stanje
kraj

Primjer ponašanja:

```
Originalna tablica
6; 5; 2; 1;
7; 5; 2; 0;
7; 7; 2; 1;
6; 5; 2; 1;
6; 5; 2; 1;
7; 5; 2; 1;
6; 5; 2; 1;
7; 7; 7; 0;

Tablica bez nedohvatljivih stanja
6; 5; 2; 1;
7; 5; 2; 0;
7; 7; 2; 1;
7; 5; 2; 1;
6; 5; 2; 1;
7; 7; 7; 0;

Minimizirana tablica
4; 3; 2; 1;
5; 3; 2; 0;
5; 5; 2; 1;
5; 5; 5; 0;
```