SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Asinkroni produkt usmjerenih labeliranih grafova

Tomislav Bradarić

Voditelj: *Prof. dr. sc. Bruno Blašković*

Zagreb, svibanj, 2012.

Sadržaj

1. Uvod ... 1
2. Osnovni pojmovi ... 2
   2.1 Usmjereni labelirani graf ... 2
   2.2 LTS format ... 2
3. Asinkroni produkt ... 4
   3.1 Definicija ... 4
   3.2 Važnost ... 4
   3.3 Primjer ... 4
4. Implementacija ... 7
   4.1 Odabir jezika i racionalizacija ... 7
      4.1.1 Skupovi ... 7
      4.1.2 Riječnici ... 8
5. Primjene ... 9
6. Zaključak ... 10
7. Literatura ... 11
8. Sažetak ... 12
9. Dodatak ... 13
   9.1 Dodatak A – Perl aralts2lts parser ... 13
   9.2 Dodatak B – Python skripta za asinkroni produkt ... 15

1. Uvod

Za konačne automate, odnosno njihov vizualni prikaz kao usmjerene labelirane grafove postoje brojne primjene, uključujući rutiranje, teoriju kodiranja, kriptografiju, umjetnu inteligenciju, adresne sustave u komunikacijskim mrežama, kozmologiju, biologiju, financijsku analizu, kristalografsku analizu, prevođenje programskih jezika te mnoge druge.

Grafovima možemo manipulirati na brojne načine, a jedna od operacija koje možemo obavljati nad njima je asinkroni produkt.

Budući da ne postoji potpuna standardizacija zapisnih formata usmjerenih labeliranih grafova, u skriptnom jeziku Perl, inače moćnom *text processoru*, izradit ću parser koji pretvara usmjereni labelirani graf u "aralts" formatu u konačni automat, odnosno LTS(engl. *Labeled transition system*), te tako implementirati asinkroni produkt bilo grafova, bilo konačnih automata.

Nadalje, zbog mnoštva modula, metoda i sl. ostvarenih i optimiziranih u jeziku C, kao i posjedovanja iznimno prigodnih struktura podataka, kao koristan programski jezik za implementaciju samog algoritma asinkronog produkta usmjerenih labeliranih grafova nameće se skriptni jezik Python. Imajući to na umu, oblikovat ću algoritam te izraditi Python skriptu koja učinkovito izvodi operaciju asinkronog produkta proizvoljnog broja usmjerenih labeliranih grafova pretvorenih u konačne automate u LTS formatu te rezultat operacije vraća također u LTS formatu.

Osnovni pojmovi i definicije koje se ovdje koriste opisani su u poglavlju 2. Formalni opis algoritma asinkronog produkta opisan je u poglavlju 3. Površni opis same implementacije u skriptnom jeziku Python te razlozi iza iste sadržani su u poglavlju 4. Jednostavnije primjene ukratko su spomenute u poglavlju 5. Naposljetku, u dodatku A i B na kraju dokumenta nalazi se *source* kod gore navedene Perl, odnosno Python skripte.

2. Osnovni pojmovi

2.1 Usmjereni labelirani graf

Usmjereni labelirani graf definiramo kao skup čvorova povezanih labeliranim bridovima usmjerenim od jednog čvora prema drugome. Formalno ga možemo definirati uređenom trojkom:

A = (V, E, L), gdje su:

- V - Skup vrhova(engl. v*ertex)*
- *E* - Skup bridova(engl. *Edges)*
- *L* - Skup labela(engl. *labels)*

Konačni automat definiramo na sljedeći način:

A' = (s0, S, L, T, F), gdje su:

- *s0* $\in$ *S* - *Početno stanje*
- *S* - *Konačan skup stanja*
- *L* - *Konačan skup labela*
- *T* - *Funkcija prijelaza* $S \times L \rightarrow S$
- F $\subseteq$ S - Skup finalnih stanja

Kratak opis pretvorbe graf => LTS:

- Početno stanje proizvoljno definiramo(tipično prvo po redu)
- Skup stanja analogan je skupu vrhova
- Skup labela analogan je skupu labela
- Funkcije prijelaza analogne su skupu bridova.
- Finalna stanja proizvoljno definiramo

2.2 LTS format

Zbog praktičnih razloga, napose simulacije i manipulacije na računalima, potrebni su određeni formati zapisivanja definicija usmjerenih labeliranih grafova te njihovog transformata u konačni automat. Poželjno je da takvi formati budu uravnoteženi sa strane čitljivosti(engl. *Human readability*) i memorijske učinkovitosti. Jedan takav format je LTS(engl. *Labeled Transition System).*

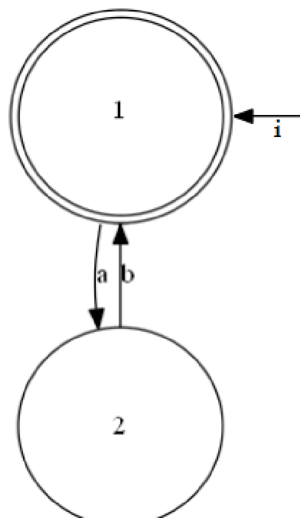
Linije koje počinju znakom "#" smatraju se komentarima i ignoriraju od strane računala. Svaki član uređene petorke formalne definicije u datoteci je najavljen ključnom riječju, a svaki podčlan članova uređene ntorke zauzima jedan redak. Ključna riječ "INIT" najavljuje početno stanje, "STATES" najavljuje stanja, "LABELS" labele, "TRANSITIONS" prijelaze te naposljetku "FINAL" najavljuje finalna stanja. Korisna je činjenica da se pri zapisu prijelaza na stanja i labele referenciramo pomoću njihovih identifikatora, a ne imena, čime u slučaju deskriptivnijih imena možemo ostvariti znatnu memorijsku uštedu.

Nekoliko napomena:

- Labela tipično rednog broja 0 te imena "i" označava prijelaz u početno stanje.
- Prijelazi su oblika "trenutno stanje" "sljedece stanje" "redni broj labele"
    - Korištenjem rednog broja labele za referiranje na labelu potencijalno ostvarujemo znatnu memorijsku uštedu pri pohrani LTS datoteke jer *string* s nazivom labele moramo pohraniti samo jedanput, umjesto proizvoljnog broja puta.

Primjerice, za usmjereni labelirani graf M prikazan na slici 2.1. pretvoren u konačni automat s proizvoljno definiranim početnim i finalnim stanjem, zapis u LTS formatu glasi:

INIT
1
STATES
1
2
LABELS
0 i
1 a
2 b
TRANSITIONS
1 2 a
2 1 b
FINAL
2



Slika 2.1: Primjer usmjerenog labeliranog grafa

3. Asinkroni produkt

3.1 Definicija

Ako imamo konačne automate $A_1,...A_n$, nad kojima obavljamo operaciju asinkronog produkta, rezultat je opet konačni automat C, i to sljedećeg oblika:

1) s0 je n-torka $\{A_1(s0),..., A_n(s0)\}$

2) S je Kartezijev produkt $A_1(S) \times \ldots \times A_n(S)$

3) L je unija $A_1(L) \cup \ldots \cup A_n(L)$

4) T je skup n-torki $((x_1,...,x_n), l, ((y_1,...,y_n))$, takav da $\exists i,\ 1 \leq i \leq n$, za koji vrijedi $(x_i, l, y_i) \in A_i(T)$, te $\forall j,\ 1 \leq j \leq n,\ j \neq i \rightarrow x_j \equiv y_j$, gdje je $(x_1,...,x_n) \in S$, te $(y_1,...,y_n) \in S$

5) F je podskup elemenata A(S) koji zadovoljavaju uvjet da $\forall (x_1,...,x_n) \in C(F),\ \forall i,\ x_i \in A_i(F)$

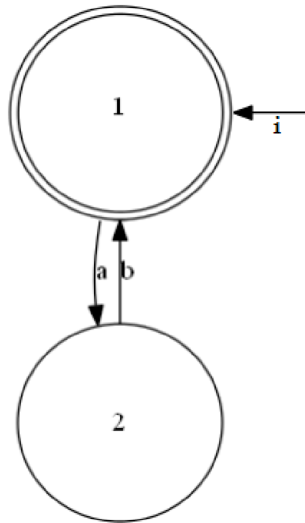
3.2 Važnost

Asinkronim produktom operandi se "stapaju" u jedan rezultantni automat koji može simulirati sve moguće kombinacije stanja u kojem možemo zateći operande kada bi oni djelovali potpuno nezavisno. U skladu s time smo prijelaze i modelirali tako da se bilo koja dva "susjedna" stanja u rezultantnom skupu stanja razlikuju u najviše jednom "podstanju"(članu n-torke). Ovo nam omogućuje da cijeli jedan sustav opišemo i analiziramo pomoću samo jednog grafa odnosno automata, tako si omogućivši jednostavnije ostvarenje, analizu i simulaciju teoretskih i stvarnih sustava.

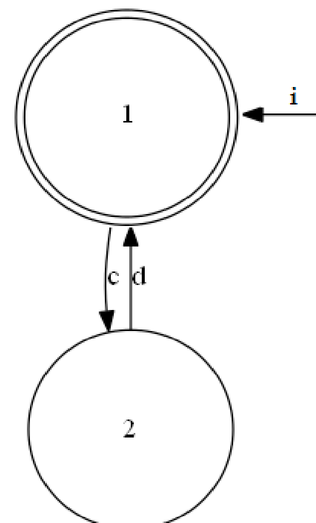
Naravno, dalje po potrebi možemo ponovno pretvoriti takav konačni automat u usmjereni labelirani graf.

3.3 Primjer

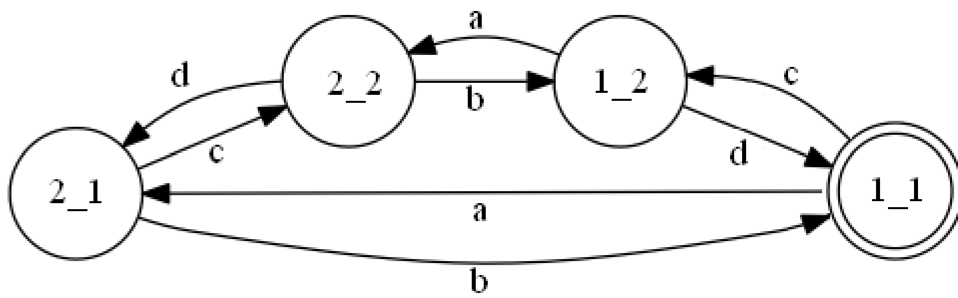
Operacijom asinkronog produkta gdje su operandi grafovi A1 i A2 na slici 3.1., dobiva se rezultantni graf graf C na slici 3.2.



Slika 3.1: Graf A



Slika 3.2: Graf B



Slika 3.3: Graf C - Asinkroni produkt grafova A i B

LTS zapis grafa C glasi:

INIT
1_3
STATES
1_3
2_3
1_4
2_4
LABELS
0 i
1 a
2 b
3 c
4 d
TRANSITIONS
1_3 2_3 1
1_3 1_4 3
2_3 1_3 2
2_3 2_4 3
1_4 1_3 4
1_4 2_4 1
2_4 1_4 2
2_4 2_3 4
FINAL
2_4

4. Implementacija

4.1 Odabir jezika i racionalizacija

Uzimajući u obzir parametre problema, kao i svoje mogućnosti svladavanja novih znanja u hodu, kao najpogodniji jezik za implementaciju algoritma asinkronog produkta odabrao sam skriptni jezik Python, iza čega stoji više razloga. Napomenuo bih i kako sam na početku algoritam namjeravao implementirati u jeziku Perl, no od toga sam odustao jer ne posjeduje neka od korisnih svojstava opisanih u ostatku ovog poglavlja.

Python ima vrlo bogat fond modula, metoda i sl. koje olakšavaju kodiranje složenijeg zadatka ovog tipa. Nadalje, osnovni moduli i potencijalno procesorski zahtjevne operacije korištene u implementaciji algoritma zapravo su pisane u jeziku C koji pruža velike mogućnosti optimizacije, čime se osigurava dovoljno velika brzina izvođenja.

Primjerice, jedan koristan modul je itertools koji je inspiriran konstruktima iz APL-a, Haskella te SML-a. Itertools modul normira osnovni skup brzih, memorijski učinkovitih alata koji su korisni pojedinačno ili u kombinaciji. Zajedno, formiraju "iteratorsku algebru", omogućujući jezgrovitu i učinkovitu konstrukciju specijaliziranih alata. Itertoolsove napredne mogućnosti generiranja iteratora olakšavaju implementaciju, ali i ubrzavaju rad skripte. Primjerice, kada imamo veliku for petlju koja dodaje elemente u listu ili skup, umjesto da Python za svaki element posebno alocira potrebnu memoriju, pri uporabi generiranog iteratora sustav unaprijed zna koliko memorije je potrebno te ju alocira odjednom, ostvarujući potencijalno značajnu uštedu procesorskog vremena. [1]

Nadalje, pri računanju Kartezijevog produkta dvaju ili više skupova, itertools ima metodu *product* koja računa Kartezijev produkt višestruko efikasnije nego što bi to činile ugniježđene for petlje. Makar to nije toliko potrebno kad se izvršava produkt samo dva automata, ipak spomenimo da, vještom kombinacijom s metodom *imap,* gore navedenoj metodi možemo predati cijelu listu skupova nad kojima radimo Kartezijev produkt, u neku ruku ostvarujući takozvani *kod koji se sam piše*, čime možemo izbjeći nesavršenu uporabu sustavskog stoga ili stoga općenito.

Još jedna korisna osobina Pythona je gotova implementacija raznih naprednijih struktura podataka koje se pokazuju korisnim, kako u učinkovitosti izvođenja asinkronog produkta, tako i u osiguravanju lakoće i prigodne razine apstrakcije pri implementaciji istoga. Neke od tih struktura ukratko su opisane u ostatku poglavlja.

4.1.1 Skupovi

Jedna od najkorisnijih struktura podataka pri implementaciji asinkronog produkta su skupovi(*engl. Set)* koji su ekvivalent skupovima u matematičkom smislu.

Osim svojstva jedinstvenosti članova(referirati se na matematičku definiciju skupa), iznimno korisnim se pokazuje i to što se nad skupom može izvoditi pretraživanje složenosti O(1). Budući da algoritam opisan u poglavlju 2.3. uključuje

mnogo pretraživanja koja s povećanjem broja operanada i njihovih veličina rastu vrlo brzo, skupovi se pri efikasnoj implementaciji algoritma pokazuju neophodnima.

Nadalje, operacije među skupovima poput unije, presjeka i sl. obavljaju se također vrlo učinkovito te s lakoćom od strane programera.[2]

4.1.2 Riječnici

Još jedan vrlo koristan alat su riječnici*(engl. dictionaries).* Riječnici, inače u nekim drugim jezicima zvani asocijativna polja, rade na principu uparivanja ključa i vrijednosti koju on indeksira.

Riječnik je najlakše zamisliti kao neuređeni skup uređenih parova (ključ: vrijednost). Dakle, ne indeksiraju se rednim brojevima kao tradicionalna polja, već ključ može biti bilo koji tip podatka koji je neizmjenjiv(*engl. immutable*), kako bi se održao integritet riječnika. U kontekstu ovog seminara, struktura podataka korištena kao ključ je ntorka(*engl. tuple).*

Slično kao i kod skupova, dohvat vrijednosti pomoću ključa odvija se u složenosti O(1). Ono što izdvaja riječnik od običnog polja je to što se vrijednost dohvaća na temelju hash vrijednosti ključa. To je, primjerice, korisno kod uklanjanja neke vrijednosti iz riječnika, jer pri tome ostale vrijednosti ne moraju mijenjati svoju lokaciju u memoriji, što čini riječnik iznimno brzim i fleksibilnim. Ta dva svojstva riječnika se pokazuju iznimno korisnim kod čestih izmjena prijelaza pri implementaciji algoritma, kao i kod sortiranja vrijednosti pri ispisu rezultata i sl.[3]

5. Primjene

Postoje brojne i raznovrsne primjene usmjerenih labeliranih grafova i konačnih automata te raznih operacija među istima. Neke od njih gdje se koristi asinkroni produkt su verifikacija sustava[4], kriptografija[5], sigurnost internetskih i distribuiranih aplikacija[7], te mnoge druge. Primjene same operacije asinkronog produkta uglavnom se svode na konstrukciju, simulaciju te analizu proizvoljnog broja operanada pomoću jednog grafa ili automata.

Imajući na umu zahtjevnost tematike nekih od tih primjena, kao i činjenicu da njihova opsežnost nadmašuje tematiku ovoga seminara, zadovoljit ćemo se samo ovim njihovim kratkim spomenom i referencama na odgovarajuću literaturu.

6. Zaključak

Za konačne automate odnosno njihov vizualni prikaz kao usmjereni labelirani graf postoje brojne primjene.

Odgovarajućim algoritmom ostvarenim u programskom jeziku Python nad operandima čije se definicije nalaze u ulaznim datotekama možemo izvršiti operaciju asinkronog produkta.

Po potrebi, ulazne datoteke nepovoljnog formata pomoću programskog jezika Perl možemo pretvoriti u format s kojim gore navedena Python skripta može funkcionirati.

Asinkronim produktom N operanada "povezujemo" u jedan rezultantni graf čija će stanja sadržavati sve moguće kombinacije stanja operanada od kojih je nastao, kao i odgovarajuće prijelaze i labele. Nakon toga, rezultantnim grafom možemo opisati i analizirati cijeli sustav, tako si omogućivši jednostavnije ostvarivanje i simulaciju teoretskih i stvarnih sustava.

7. Literatura

1: , 9.7. itertools — Functions creating iterators for efficient looping, http://docs.python.org/library/itertools.html,18.4.2012.
2: , class set([iterable]), http://docs.python.org/library/stdtypes.html#set,01.04.2012.
3: , 5.5. Dictionaries, http://docs.python.org/tutorial/datastructures.html#dictionaries,03.05.2012.
4: Peter Ochsenschlager, Jurgen Repp, Roland Rieke, Verication of Cooperating Systems - An Approach Based on Formal Languages,
5: Sigrid Gurgens, Peter Ochsenschlager, Carsten Rudolph, Role based specification and security analysis of cryptographic protocols using asynchronous product automata, 2002
7: Isaac Agudo, Javier Lopez, Specification and Formal verification of security requirements, 2004

8. Sažetak

Za usmjerene labelirane grafove postoje brojne primjene, uključujući rutiranje, teoriju kodiranja, kriptografiju, umjetnu inteligenciju, adresne sustave u komunikacijskim mrežama, kozmologiju, biologiju, financijsku analizu, kristalografsku analizu, prevođenje programskih jezika te mnoge druge. Grafovima možemo manipulirati na brojne načine, a jedna od operacija koje nad njima možemo obavljati je asinkroni produkt.

Ovim radom ostvario sam aralts => lts parser u skriptnom jeziku Perl te oblikovao i implementirao algoritam asinkronog produkta usmjerenih labeliranih grafova u skriptnom jeziku Python.

Osnovni pojmovi i definicije koje se ovdje koriste opisao sam u poglavlju 2. Formalni opis algoritma asinkronog produkta nalazi se u poglavlju 3. Površni opis same implementacije u skriptnom jeziku Python te razlozi iza iste sadržani su u poglavlju 4. Jednostavnije primjene ukratko su spomenute u poglavlju 5. Naposljetku, u dodatku na kraju dokumenta nalaze se dvije gore navedene skripte.

9. Dodatak

9.1 Dodatak A – Perl aralts2lts parser

```perl
#! /usr/bin/perl

use Getopt::Long;
#use warnings;
#use strict;
use Switch 'Perl6';

my $aralts2lts = 0;     # option variable with default value (false)
my $input;

GetOptions ('aralts2lts' => \$aralts2lts, # flag
            'i:s' => \$input,         #
            );

$init_state; # initial automata state
@states;     # array of all states
@labels;     # array of file lines, each line is one label
@transitions; # array of file lines, each line is one transition

# process input file
if(defined($input)) {
    printf "#generated from aralts: %s\n", $input;

    if(! open(INPUT, "<$input" ) ) {          # Open the file
        die "(cope-4211tb) FATAL ERROR: Can't open file $input \n";
      }
    @infile = <INPUT>;      # Read file into an array
    $line_index;
    close(IN) ;                          # Close the file
} # end file input

else {
    die "\nERROR: No input file!\n
         Please define an input file using the option -i
         and file path as argument.\n";
}

if( $aralts2lts == 1 ) { # if option on, call the appropriate subroutine
    &input_aralts;
    &print_as_lts_moduformat;
    exit;
}

# processes aralts input file
sub input_aralts{
    $line_index = -1;
    #chomp(@infile); #clear newlines from each line

    # let's process the file
    foreach (@infile) {
        $line_index = $line_index + 1;

        given ($_) {
            when (/^LTS/){ #
                (/\|(.+)\|/);
                $name = $1;
            }
            when (/^ROOT/){ # determine initial state
```

```
                (/\|([0-9]+)\|/);
                my $root = int($1);
                $init_state = $root;
            }
            when (/^NODE_COUNT/){ # generate implicitly given set of states
                (/\|([0-9]+)\|/);
                my $node_count = int($1);
                for ($pom = 0; $pom < $node_count; $pom++) {
                    $states[$pom] = $init_state+$pom; #
                }
            }
            when (/^ARC_COUNT/){ #
                (/\|([0-9]+)\|/);
                my $arc_count = int($1);
            }
            when (/^SYMBOL_COUNT/){ #
                (/\|([0-9]+)\|/);
                my $symbol_count = int($1);
            }
            when (/^SYMBOLS/){
                $symbols_start = $line_index;
            }
            when (/^ARCS/){
                $arcs_start = $line_index;
            }
            when (/^END_LTS/){
                $lts_end = $line_index;
            }
            default {
            }
        } ### end switch

    } ## end file mapping pass
}

sub print_as_lts_moduformat{
    print "#$name\n#COpe\n#$name.aralts\n#4211 TB\n\n"; # print header

    print "INIT\n";
    print "$init_state\n";

    print "STATES\n";
    foreach (@states){
        print "$_\n";
    }

    print "LABELS\n";
    $bla = $symbols_start + 1;
    $_ = $infile[$bla];
    until (/^END_SYMBOLS/){
        print $infile[$bla];
        $bla++;
        $_ = $infile[$bla];
    }

    print "TRANSITIONS\n";
    $bla = $arcs_start + 1;
    $_ = $infile[$bla];
    until (/^END_ARCS/){
        print $infile[$bla];
        $bla++;
        $_ = $infile[$bla];
    }
}
```

9.2 Dodatak B – Python skripta za asinkroni produkt

```
import os
import sys
import re
import itertools
import argparse
from collections import defaultdict

parser = argparse.ArgumentParser(description=
        'Asynchronously multiply N directed labeled graphs')
parser.add_argument('-i', '--input', nargs='+',
                    action="store", dest="input", default=sys.stdin,
                    help="Putanje do ulaznih definicija grafova")

#parser.add_argument('-o', '--output', nargs='?',
#                     dest="output", action="store", default=None,
#                     help="Putanja do izlazne datoteke za rezultat")

parse_results = parser.parse_args(sys.argv[1:])


###############

# pretpostavlja se da lista_automata sadrzi dva clana(jer radimo postupno)
def asyn_prod(lista_automata):
    novo_pocetno = (lista_automata[0].pocetno_stanje,
                    lista_automata[1].pocetno_stanje)

    stanja = set(itertools.product(
                *itertools.imap(lambda x: x.stanja, lista_automata)))

    labele = set()
    labele.update(lista_automata[0].labele)

    rijecnik = dict() # za konvertiranje labela 2. operanda

    nove_labele2 = set() # zbog pretrage kod ugnijezdenih petlji
    a = len(labele)
    for labela in lista_automata[1].labele:
        if labela in labele:
            rijecnik[labela[0]] = labela[0] # sve isto
            nove_labele2.add(labela)
            continue
        rijecnik[labela[0]] = str(a)
        lab = (str(a), labela[1])
        nove_labele2.add(lab)
        labele.add(lab)
        a += 1

    lista_automata[1].labele = nove_labele2

    novi_prijelazi = defaultdict(list)
    for kljuc in lista_automata[1].prijelazi:
        novi_kljuc = ((kljuc[0], str(rijecnik[kljuc[1]])))
        novi_prijelazi[novi_kljuc] += (lista_automata[1].prijelazi[kljuc])

    del rijecnik

    # pripaziti na ovo ako se u skriptu bude dodavalo
    # jos neke funkcionalnosti osim produkta
    lista_automata[1].prijelazi = novi_prijelazi

    prijelazi = defaultdict(list)
```

```
    # neka vas ne uplase for petlje, ne izvode se dugo
    for indeks, automat in enumerate(lista_automata): # kratka(2 kruga)
        for rezultantno_stanje in stanja: # kratka(2 kruga)
            for labela in lista_automata[indeks].labele:
                kljuc = (rezultantno_stanje[indeks], labela[0])

                if kljuc in lista_automata[indeks].prijelazi:
                    sljedstanja = lista_automata[indeks].prijelazi[kljuc]
                    for sljedece in sljedstanja: # relativno kratka
                        if indeks == 0:
                            sljedece = (sljedece, rezultantno_stanje[1-indeks])
                        elif indeks == 1:
                            sljedece = (rezultantno_stanje[1-indeks], sljedece)

                        if sljedece in stanja:
                            prijelazi[(rezultantno_stanje, kljuc[1])].append(
                                                                        sljedece)

    finalna_stanja = set(itertools.product(*itertools.imap(
                lambda x: x.finalna_stanja, lista_automata)))

    rezultantni_automat = Automat()
    rezultantni_automat.pocetno_stanje = novo_pocetno
    rezultantni_automat.stanja = stanja
    rezultantni_automat.labele = labele
    rezultantni_automat.prijelazi = prijelazi
    rezultantni_automat.finalna_stanja = finalna_stanja

    return rezultantni_automat

class Automat(object):
    def ucitaj_iz_datoteke(self, datoteka):
        niz = None
        with open(datoteka, 'r') as fp:
            niz = [line.strip('\r\n') for line in fp]

        self.ime_automat = datoteka

        for indeks, redak in enumerate(niz):
            if (redak == 'INIT'):
                self.pocetno_stanje = niz[indeks+1]
                self.pocetno_stanje = (self.pocetno_stanje.split(' '))[0]
            elif (redak == 'STATES'):
                init_stanja = indeks
            elif (redak == 'LABELS'):
                init_labele = indeks
            elif (redak == 'TRANSITIONS'):
                init_prijelazi = indeks
            elif (redak == 'FINAL'):
                init_final = indeks

        niz_stanja = [] # list()
        niz_stanja = niz[init_stanja+1:init_labele]
        niz_labela = niz[init_labele+1:init_prijelazi]
        niz_prijelaza = niz[init_prijelazi+1:init_final]
        niz_finalnih = niz[init_final+1:]

        del niz

        self.stanja = set()
        for stanje in niz_stanja:
            stanje = (stanje.split(' '))[0] # ne uzmi u obzir opis stanja
            self.stanja.add(stanje)

        #labela = (ident, naziv)
```

```
        self.labele = set()
        for labela in niz_labela:
            m = re.search('(\A[^ ]*)(.*)',labela) # split po prvom razmaku
            ident = m.group(1)
            komentar = m.group(2)[1:]
            self.labele.add((ident,komentar))

        self.prijelazi = defaultdict(list)
        for prijelaz in niz_prijelaza:
            # (trenutno, sljedece, znak)
            prijelaz = prijelaz.split(' ')
            self.prijelazi[(prijelaz[0], prijelaz[2])].append(prijelaz[1])

        self.finalna_stanja = set()
        for finalno in niz_finalnih:
            self.finalna_stanja.add(finalno.split(' ')[0])

    # prilagodava medurezultat sljedecem mnozenju
    # (ako mnozimo vise od 2 automata odjednom)
    # stedljivije za memoriju nego drzanje svih
    # operanada i cijelog medurezultata u memoriji odjednom
    def ucitaj_iz_memorije(self):
        self.pocetno_stanje = '_'.join(self.pocetno_stanje)

        self.stanja = set(map(lambda x: '_'.join(x), self.stanja))

        self.finalna_stanja = set(map(lambda x: '_'.join(x),
                                  self.finalna_stanja))

        # labele ostaju kako jesu

        for key in self.prijelazi:
            new_key = ('_'.join(key[0]), key[1])
            for indeks, prijelaz in enumerate(self.prijelazi[key]):
                self.prijelazi[new_key].append('_'.join(
                            self.prijelazi[key][indeks]))
            del self.prijelazi[key]

    def ispisi_se(self):

        #self.print_header()

        print 'INIT'
        print '_'.join(self.pocetno_stanje)

        print 'STATES'
        for stanje in sorted(self.stanja):
            print '_'.join(stanje)

        print 'LABELS'
        for labela in sorted(self.labele):
            print labela[0], labela[1]

        print 'TRANSITIONS'
        for key in sorted(self.prijelazi):
            for indeks, prijelaz in enumerate(sorted(self.prijelazi[key])):
                print '_'.join(key[0]),
                print '_'.join(self.prijelazi[key][indeks]),
                print key[1]

        print 'FINAL'
        for stanje in sorted(self.finalna_stanja):
            print '_'.join(stanje)

def print_header(automati):
```

```
    print '# Generated from:',
    for automat in automati:
        print automat.ime_automat,
    print '\n# COpe'
    print '# 4211 TB\n'

if __name__ == '__main__':


    automati = [] # list()

    # datoteke s definicijama zadane preko komandne linije
    for indeks, definicija in enumerate(parse_results.input):
        automati.append(Automat())
        automati[indeks].ucitaj_iz_datoteke(definicija)

    print_header(automati)

    # prima listu automata(samo dva jer radimo postupno)
    rezultantni_automat = asyn_prod([automati[0], automati[1]])
    del automati[:2]

    for automat in automati:
        rezultantni_automat.ucitaj_iz_memorije()
        rezultantni_automat = asyn_prod([automat, rezultantni_automat])

    rezultantni_automat.ispisi_se()
```