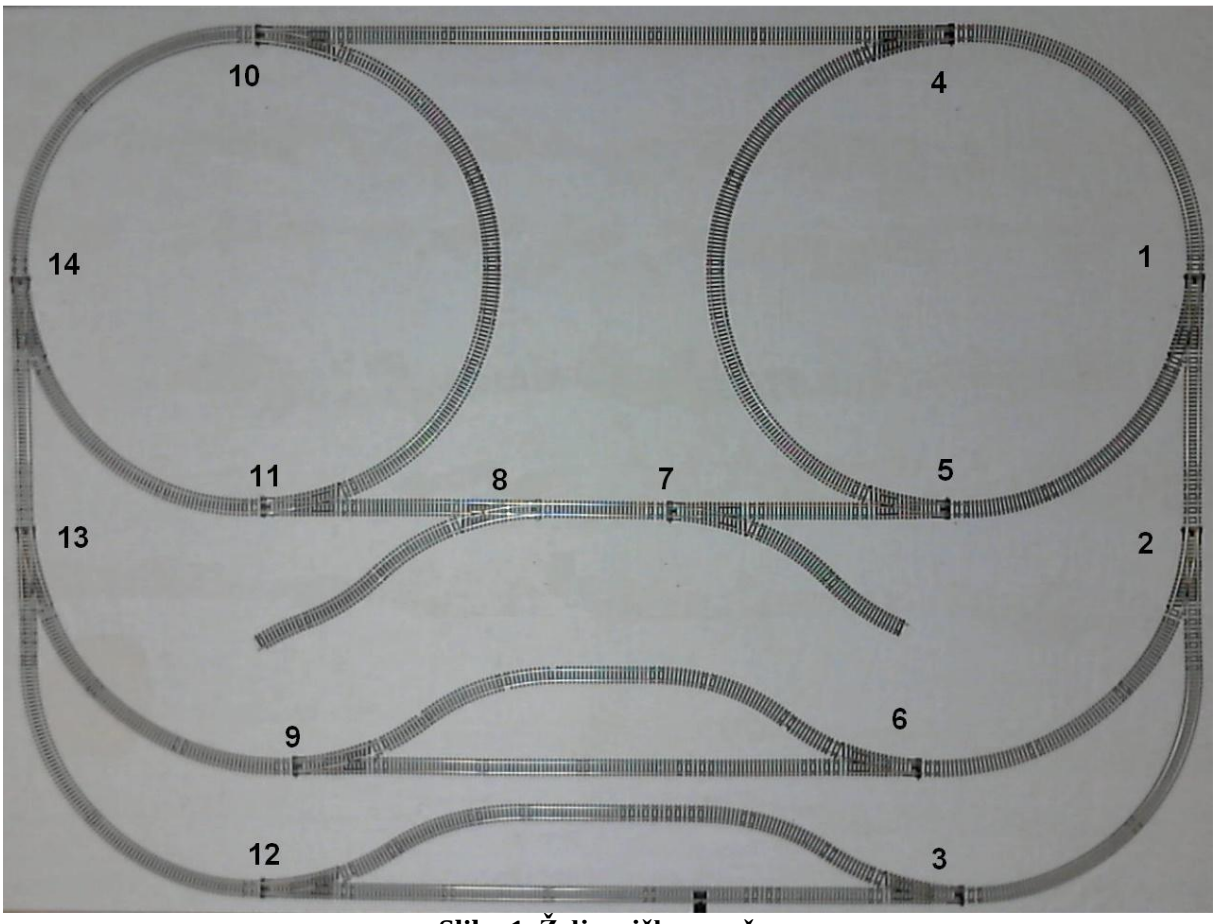
BARTULOVIĆ MIHOVIL, 0036453342 ERDELIĆ TOMISLAV, 0036450291 PAVLOVIĆ ALEN, 0036452163 1.D_AUT AUTOMATIKA	FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA ZAGREB ZAVOD ZA AUTOMATIKU I RAČUNALNO INŽENJERSTVO **Sustavi s diskretnim događajima** 2. seminarski zadatak	22.6.2013.

Zadatak

Da bi se moglo definirati upravljanje sustavom, svakoj skretnici (čvoru) u sustavu pridružuje se jedinstvena oznaka. Na Slici 1 je prikazana željeznička mreža s oznakama čvorova.



Slika 1. Željeznička mreža

Staze po kojima vlakovi voze zadane su sa:

```
'red': 4; 10; 14; 11; 8; 7; 5; 1
'green': 5; 4; 1
'blue': 10; 11; 14
```

Iz tako zadanih staza dobiva se sljedeća lista poslova - v:

```
v['red'] = [(4,10),(10,14),(14,11),(11,8),(8,7),(7,5),(5,1),(1,4)]
v['green'] = [(5,4),(4,1),(1,5)]
v['blue'] = [(10,11),(11,14),(14,10)
```

Prvi brid staze (**v['boja'][0]**) je ulazno mjesto sustava te se vlak u početnom trenutku nalazi na tom bridu. Taj brid predstavlja i izlazno mjesto sustava. Da bi se u korisničkom programu provjerilo da li je vlak na ulaznom mjestu sustava, ispituje se da li je **vc['boja'][0] == 1**. Da bi vlak napustio sustav, postavlja se **vs['boja'][0] = 1**.
Svaki brid u sustavu odgovara jednom resursu. Svi su resursi kapaciteta jedan. Drugim riječima, na svakom bridu smije se istovremeno nalaziti samo jedan vlak. Lista resursa je:

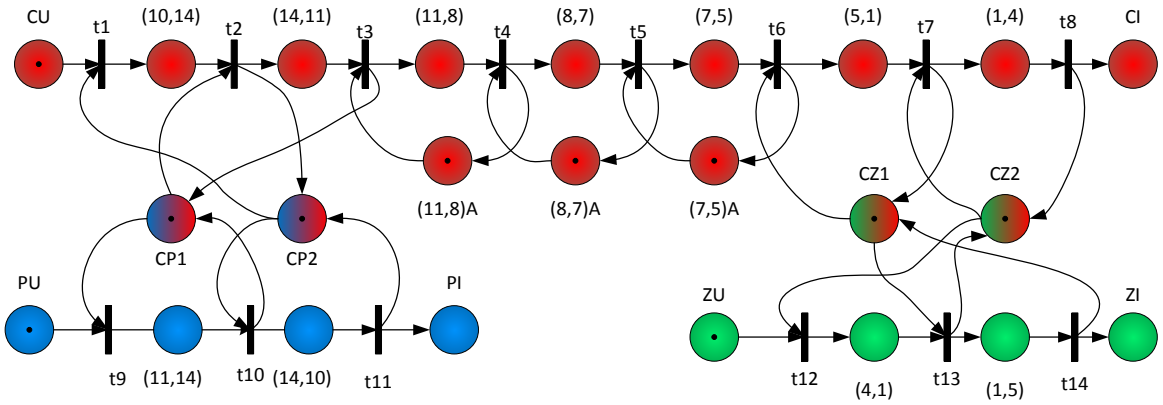
```
r = [(10,11),(11,14),(10,14),(4,5),(1,4),(1,5),(4,10),(8,11), (7,8),(5,7)]
```

Iako su ulazni/izlazni bridovi navedeni u listi r, oni se u Petrijevoj mreži ne razmatraju kao resursi već kao ulazno/izlazno mjesto. Trenutno stanje resursa zapisano je u vektoru slobodnih resursa - **ra**. Vrijednost od **ra[i]** jednaka je 1 ako je pripadni resurs slobodan tj. ako na njemu nema vlaka, inače je 0. Npr. ako se jedan vlak nalazi na bridu (4,1), jedan na (8,7), a jedan na (11,14), tada je **ra**:

```
ra = [1,0,1,1,0,1,1,1,0,1]
```

Vrijednost od **ra** za određeni resurs postavlja se u 1 tek kad vlak u potpunosti napusti taj resurs. Promjena stanja vektora **ra** implementirana je u nižoj razini upravljanja

Slikom 2. prikazan je graf Petrijeve mreže zadanog sustava danog zadatkom i Slikom 1.



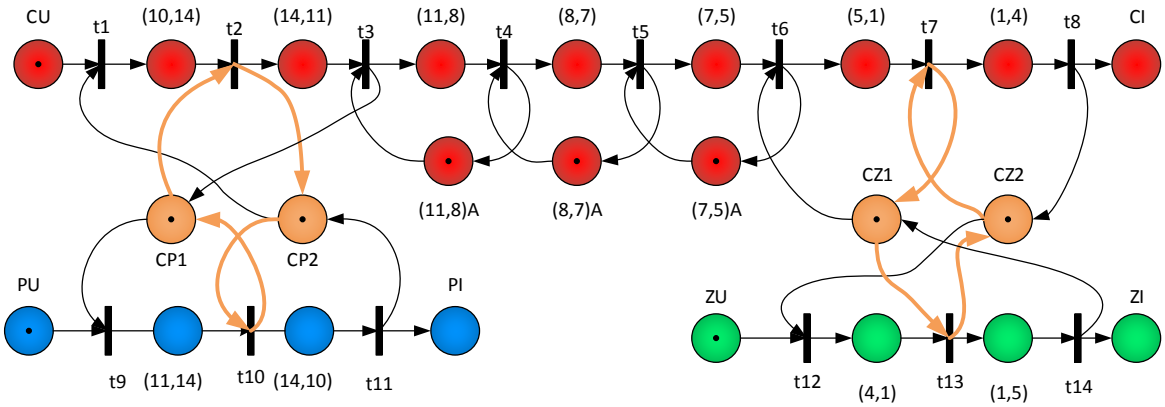
Slika 2. Graf Petrijeve mreže

Također, na Slici 2. vide se prijelazi, resursi i poslovi te početni uvjeti koji odgovaraju zadatku. Značenja pojedinih mjesta su sljedeća :

- xU – ulazno mjesto crvenog/plavog/zelenog vlaka
- xI – izlazno mjesto crvenog/plavog/zelenog vlaka
- (X,Y) – resursi tj. rute kroz koje pojedini vlak mora proći (Slikom 2. pojedine staze s obojane odgovarajućom bojom), npr. (11,8) odgovara ruti od mjesta 11 do mjesta 8 sa Slike 1.
- (X,Y)A – indikator dostupnosti resursa (ne uključuje višeradne resurse)
- CPx – indikator dostupnosti višeradnog resursa koji je vezan uz crveni i plavi vlak
- CZx – indikator dostupnosti višeradnog resursa koji je vezan uz crveni i zeleni vlak

Kružno čekanje tražimo na način da se gibamo suprotnim smjerom od smjera strelica na grafu. Krećemo od određenog resursa i prolazeći isključivo po resursima tražimo postojanje određene „staze“ koja će nas vratiti do resursa od kojeg smo krenuli. Kroz resurs se smije proći točno jednom želimo li odrediti određenu kružnu stazu.

Zadani sustav opisan Petrijevom mrežom na slici 2 sadrži točno dva kružna čekanja. U nastavku će biti dani postupak kako iz kružnih čekanja odrediti kritični sifon, kritični podsustav i ostale podatke dok je na slici 3 istaknuti kružna čekanja sustava željenice.



Slika 3. Kružno čekanje (kružna čekanja označena narančastom bojom)

Pri određivanju kritičnog sifona jasno je da će on sadržavati kružno čekanje i moramo naći sve prijelaze u C koji nisu spriječeni resursima iz C. Drugim riječima striktna definicija sifona glasi $\cdot S \subset S \cdot$. Sa slike 4 lako se mogu uočiti dva kružna čekanja :

$$C_1 = \{CP_1, CP_2\}$$
$$C_2 = \{CZ_1, CZ_2\}$$

Ispišimo sve ulazne i izlazne prijelaze u skup C_1.

$$\cdot C_1 = \{t_2, t_3, t_{10}, t_{11}\} = T_{C1}^i$$
$$C_1 \cdot = \{t_1, t_2, t_9, t_{10}\}.$$

Računamo dalje T_C :

$$T_{C1} = \cdot C_1 \cap C_1 \cdot = \{t_2, t_{10}\}.$$

Sada možemo dobiti sve ulazne prijelaze koji nisu blokirani resursima u C_1:

$$T_{S1} = T_{C1}^i \backslash T_{C1} = \{t_2, t_3, t_4, t_5\} \backslash \{t_2, t_3, t_4\} = \{t_3, t_{11}\}.$$

Nadalje, moramo odrediti skup poslova sifona $J_S(C_1)$ preko poslova kružnog čekanja $J(C_1)$:

$$\cdot T_{S1} = \{(14,11), (14,10)\}$$
$$J(C_1) = \{(10,14), (14,10), (11,14), (14,11)\}$$
$$J_S(C_1) = \cdot T_S \cap J(C_1) = \{(14,11), (14,10)\}.$$

Kritični sifon određujemo pomoću iduće relacije:

$$S_{C_1} = C_1 \cup J_S(C_1)$$

Konačno, dobivamo sifon S_C :

$$S_{C_1} = \{CP_1, CP_2, (14,11), (14,10)\}$$

Potpuno analogan postupak provodimo i za $C_2 = \{CZ_1, CZ_2\}$ bez dodatnih objašnjenja.

$$C_2 = \{CZ_1, CZ_2\}$$

$$\cdot C_2 = \{t_7, t_8, t_{13}, t_{14}\} = T_{C2}^i$$

$$C_2 \cdot = \{t_6, t_7, t_{12}, t_{13}\}$$

$$T_c = \cdot C_2 \cap C_2 \cdot = \{t_7, t_{13}\}$$

$$T_{S2} = T_{C2}^i \backslash T_{C2} = \{t_8, t_{14}\}$$

$$\cdot T_{S2} = \{(1,4), (1,5)\}$$

Skup poslova sifona:

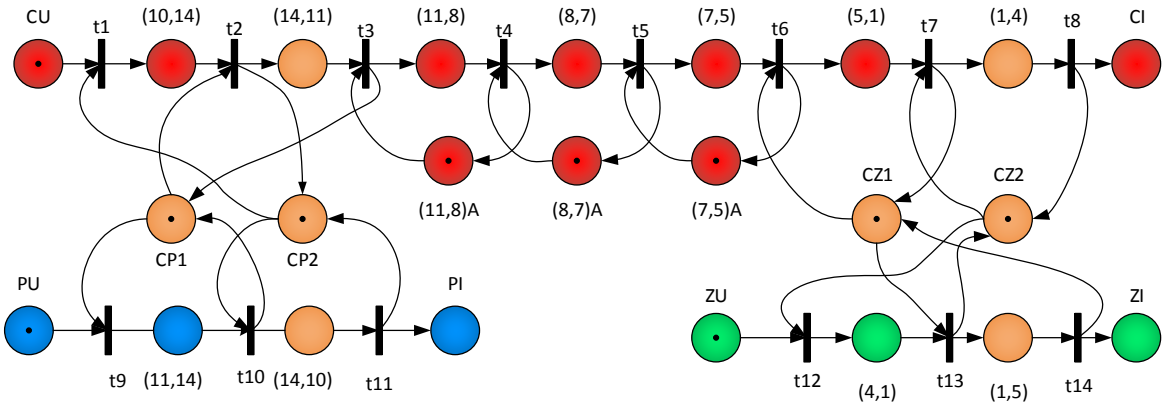
$$J(C_2) = \{(1,5), (5,1), (1,4), (4,1)\}$$

$$J_S(C_2) = \cdot T_S \cap J(C_2) = \{(1,4), (1,5)\}$$

Kritični sifon je sljedeći:

$$S_{C_2} = C_2 \cup J_S(C_2) = \{CZ_1, CZ_2, (1,4), (1,5)\}$$

Kritični sifoni naznačeni su na slici 4.



Slika 4. Kritični sifoni Petrijeve mreže (označeni narančastom bojom)

Iz slike 5 može se uočiti da kritični sifoni pojedinog kružnog čekanja osim kružnog čekanja sadrže i operacije $(14,11), (14,10)$ za kružno čekanje C_1 te $(1,4), (1,5)$ za kružno čekanje C_2. Velika prednost u ovoj Petrijevoj mreži je što ne postoje složena kružna čekanja što znatno olakšava račun te kasnije upravljački algoritam.

U ovoj fazi izračuna poznajemo skup poslova kružnih čekanja $J(C)$ i skup poslova sifona $J_S(C)$. Relacija koja povezuje skup poslova kružnog čekanja, skup poslova sifona i kritični podsustav dana je idućom jednadžbom:

$$J(C) = J_S(C) \cup J_0(C)$$

Važno je znati da cijelo vrijeme radimo algebru nad skupovima te prethodni izraz možemo preinačiti tako da izrazimo $J_0(C)$.

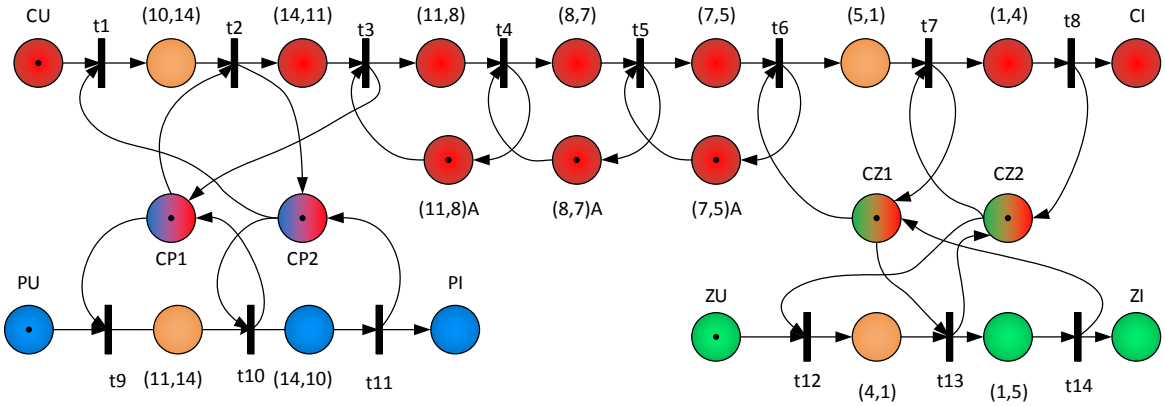
$$J_0(C) = J(C) \setminus J_S(C)$$

Sljedeću jednakost je potpuno opravdano izvesti iz razlog što J_S te J_0 ne smiju sadržavati iste operacije, ovime je olakšan postupak računa jer nije potrebno tražiti skup poslova zamke itd; a rezultat je isti. Kritični podsustav za sustav na Slici 5 potrebno je odrediti za oba kružna čekanja te su rješenja sljedeća :

$$J_0(C_1) = J(C_1) \setminus J_S(C_1) = \{(10,14), (11,14)\}$$

$$J_0(C_2) = J(C_2) \setminus J_S(C_2) = \{(5,1), (4,1)\}$$

Fizikalno na maketi željenice kritični situaciju predstavljaju mogući sudar, tj. dva vlaka će ići jedan prema drugome. Kritični podsustavi željenice prikazan je na Slici 6.



Slika 5. Kritični podsustav (kritični podsustavi označeni narančastom bojom)

Jedini način da se zadani sustav zaglavi u slučaju kada se kritični sifon isprazni, tj. drugim riječima ako S_{C_1} ili S_{C_2} izgubi sve oznake koje posjeduje inicijalno. U sustavu upravljačkim algoritmom možemo spriječiti da dođe do konflikata, tj. upravljačkim algoritmom sustav se može izbjeći zaglavljenje.

Jedini način kako osigurati da sustav ne uđe u zaglavljenje taj da se upravljačkim algoritmom osigura da u kritičnom sifonu ima uvijek barem jedna oznaka. Odnosno, mora vrijediti iduća relacija:

$$m_0(C) = m(S_C) + m\big(J_0(C)\big).$$

Pri čemu je zadano da je $m_0(C) = 2$. Od tuda slijedi da $m\big(J_0(C)\big)$ smije najviše biti jednak 1 tj. mora biti barem jedna oznaka u sifonu S_{C_1} te S_{C_2}. Razvidno je da ispražnjenjem sifona imamo $m(S_C) = 0$. Tada su sve oznake, koje su u početnom stanju bile u kritičnom sifonu, dospjele u kritični podsustav što uzrokuje zaglavljenje.

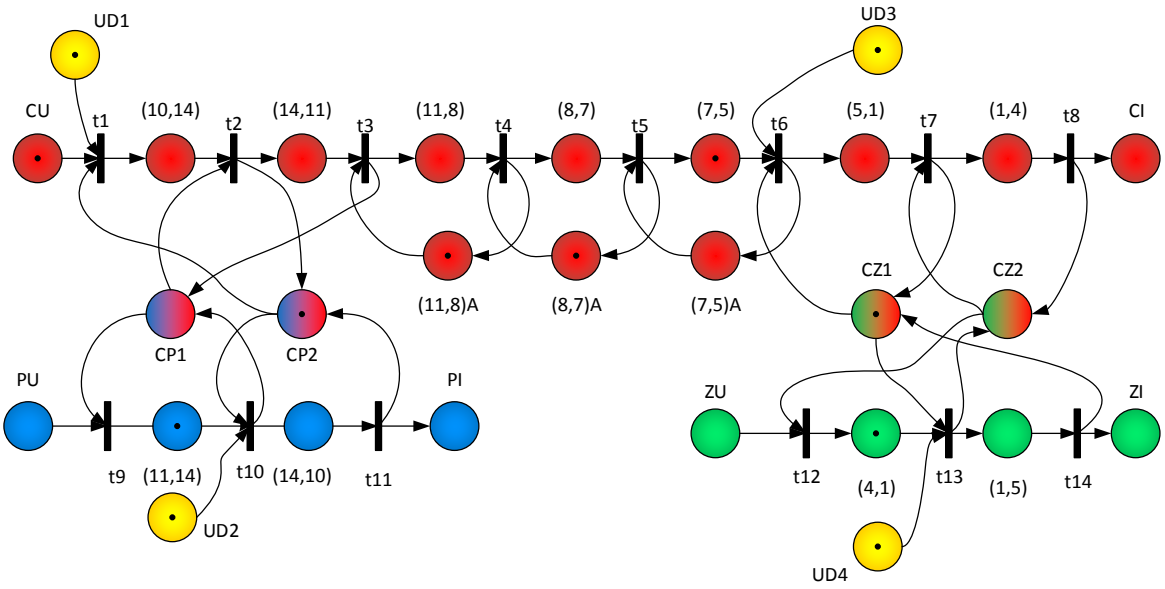
Nužno je prvo riješiti potencijalne konflikte vezane uz višeradni resurs – u sustavu željenice to je dio tračnica na kojem prometuju dva vlaka svaki svojom putanju sa zajedničkim jednim djelom pruge koja time čini višeradni resurs. U Pythonu je konflikt razriješen tako da crveni vlak uvijek ima prednost ulaska u „tunel“ pred plavim i zelenim vlakom. Odnosno plavi i zeleni vlak ne ulaze u „tunel“ samo ako je crveni već u njemu i ako je crveni taman stigao u čvor 10, tj. čvor 5. U Pythonu su još osigurana vremena između pojedinih ulaska i izlazaka iz „tunela“ u iznosu od 7 sekundi, kako ne bi došlo do sudara. Za detekciju pozicije vlakova korišten je vektor slobodnih resursa $\boldsymbol{r_a}$ koji nam zapravo govori na kojoj se dionici nalazi pojedini vlak.

Kružna čekanja na početku imaju po dvije oznake, što znači za upravljanje da je potrebno imati barem jednu oznaku u kritičnom sifonu te ne smije ostati prazan. Upravljački algoritam osigurava da u kritičnom sifonu postoji barem jedna oznaka tako da provjera broj oznaka u kritičnom podsustavu, te ako postoji jedna oznaka u kritičnom podsustavu ne dopuštamo ulazak oznake u kritični podsustav odnosno dopuštamo jedino da oznaka napusti kritični podsustav. Upravljački algoritam napisan u PetriNet sintaksi je sljedeći :

$$IF\big((10{,}14) + (11{,}14) == 1\big)\ THEN\ (UD1 = 0\ AND\ UD2 = 1)$$

$$IF\big((5{,}1) + (4{,}1) == 1\big)\ THEN\ (UD3 = 0\ AND\ UD4 = 1)$$

Prethodno navedenim upravljački algoritmom osigurat će se pražnjenje kritičnog podsustava, odnosno „punjenje“ sifona čime će se osigurati da u sustavu ne postoji zaglavljenje. Slika 7 pokazuje situaciju kada se aktivira upravljački algoritam za sprječavanje zaglavljenja.



Korisnički program se upisuje se u dio unutar Python skripte My_supervisor.py. Korisnik se pri izradi novog upravljačkog mora pridržavati unaprijed određene strukture funkcije koja implementira razvijeni algoritam upravljanja, ali unutar funkcije ima potpunu slobodu pisanja vlastitog koda. Korisnički algoritam upravljanja piše se u Python programskom jeziku. Funkcija tj. skripta koja sadrži upravljački algoritam poziva se pri svakom koraku diskretizacije upravljačkog dijela. Kao ulaz prima podatke o poziciji vlaka na pruzi, a kao izlaz daje vrijednosti upravljačkih signala za vlakove. Upravljački algoritam dan je u dodatku A.

Pri završetku upravljanja željeznicom, sustav generira .txt datoteku u kojoj se nalaze vs_time i vc_time. U vs_time su zapisana vremena kada je postavljena jedinica u vs na nekom čvoru (skretnici) za bilo koji vlak, odnosno govori na vrijeme kada je pojedini vlak krenuo iz pojedinog čvora. Analogno vc_time govori vrijeme kada je pojedini vlak došao do pojedinog čvora. Vrijeme se broji u sekundama od početka 1970 godine.

PLAVI		
Skretnica 10 k-ti put	Vrijeme polaska iz skretnice 10	Trajanje kruga [s]
0. početak	1371194810.496	
1. krug	1371194850.231	39.735
2. krug	1371194878.315	28.084
3. krug	1371194918.321	40.006
CRVENI		
Skretnica 4 k-ti put	Vrijeme polaska iz skretnice 4	Trajanje kruga [s]
0. početak	1371194810.987	
1. krug	1371194875.304	64.317
ZELENI		
Skretnica 5 k-ti put	Vrijeme polaska iz skretnice 5	Trajanje kruga [s]
0. početak	1371194810.739	

1. krug	1371194827.005	16.266
2. krug	1371194854.227	27.222
3. krug	1371194897.405	43.178
4. krug	1371194923.781	26.376

Tablica 1. Mjerenja na stvarnom sustavu na temelju *vs_time* za sustav s 3 vlaka

Iz tablice 1. je vidljivo da plavi i zeleni vlak, kad mogu bez problema ući u „tunel“ imaju trajanje kruga oko 28 sekundi, što se podudara s pretpostavljenim trajanjima tih manjih putanja (7+7+14). Na početku su očevidno trajanja krugova manja zbog položaja vlakova pri pokretanju sustava. Trajanje putanje crvenog kruga bi trebala prema podacima iz uputa iznositi 60 sekundi, a mi dobijemo 64.317 što je zadovoljavajuće. Proračun se mogao provoditi i na temelju *vc_time*, dobili bi se vrlo slični rezultati. Razlika u vrijednostima može se pripisati tome što trajanja pojedinih dijelova putanje nisu u sekundu točno definirana. Dodatni uzrok mogućim netočnim podacima jest kamera koja u nekim slučajevima prekasno detektira dolazak vlaka ispred skretnice, pa se vlak prekasno zaustavi što može rezultirati sudarim dvaju vlakova. Još jedan uzrok je što pri brzine pri pokretanju vlakova nisu baš iste, jedan krene brže drugi sporije. Rezultati s boljom preciznošću bi se dobili da smo uzeli još nekoliko krugova, ali smo imali probleme s ispadanjem vlaka na jednom dijelu putanje. Sustav ima brojne probleme, tako da je bilo potrebno konstantno resetiranje svih uređaja, te velika pažnja pri postavljanju i pratnji vlakova kako bi se dobili približno točni rezultati.

Dodatak A

Skripta *mySupervisor_grupaB.py*.

```
# -*- coding: cp1250 -*-
"""
Upravljanje maketom željenice. Seminar 2 Sustavi s diskretnim
događajima

Bartulović Mihovil
Erdelić Tomislav
Pavlović Alen

@FER, 14.06.2013

---> Upravljački algoritam za upravljanje maketom željenice, na kojoj
se nalaze tri vlaka.
      >crveni
      >plavi
      >zeleni
U .cfg datoteku je potrebno definirati staze vlaka.

'blue': 10; 11; 14
'red':4; 10; 14; 11; 8; 7; 5; 1
'green': 5; 4; 1

Algoritam upravljanje moguće je testirati na realnom sustavu.
"Liste poslova za pojedini vlak , spremljene u tip podataka
DICTIONARY"
Za primjer tri vlaka na temelju staze zapisane u .cfg datoteci dobiva
se:
v['red'] = [(4,10),(10,14),(14,11),(11,8),(8,7),(7,5),(5,1),(1,4)]
v['green'] = [(5,4),(4,1),(1,5)]
v['blue'] = [(10,11),(11,14),(14,10)

"""

"Učitavanje modula za mjerenje vremena"
from time import time
from Supervisor import Supervisor

class mySupervisor(Supervisor):

    def __init__(self,state,trains,turnout_ctrl):
        Supervisor.__init__(self,state,trains,turnout_ctrl)

        #INICIJALIZACIJA VARIJABLI KOJE JE POTREBNO PAMTITI TIJEKOM
RADA SUSTAVA
        #zamijeniti x imenom varijable
        """Inicjalizacija vremena zbog sigurnosnih razloga,
        kako ne bi došlo do sudara prilikom ulaska i izlazska iz
"tunela" """
        self.startRed1 = time()
        self.startRed2=time()-1000
        self.startGreen=time()
        self.startBlue=time()-1000
        "U Vektor resursa stavimo početni položaj vlakova"
```

```
        self.ra=[0,1,1,0,1,1,0,1,1,1]

    def new_scan(self):
        vc = self.state.vc
        vs = self.state.vs
        firstScan = self.first_scan

        #PRIPREMA  (zamijeniti x imenom varijable)
        startRed1=self.startRed1
        startRed2=self.startRed2
        startGreen=self.startGreen
        startBlue=self.startBlue
        ra=self.ra


#########################################################################
        ############ DODATI UPRAVLJACKI KOD
#################################
        if firstScan:
            """ Pokretanje pojedinog vlaka s odgovarajuće pozicije,
            prema početnim uvjetima danim u pripremi.
            """

            vs['blue'] [0]   = 1
            vs['red']  [0]   = 1
            vs['green'][0]   = 1

        else:

    # ------------------------- BLUE ---------------------------------

            if vc['blue'].count(1) != 0 :

                "Upravljački algoritam PLAVOG vlaka"

                if   vc['blue'][0]  == 1 and ra[1] and ra[2] and
(not(vc['red'][0]==1)) and ((time()-startRed1)>7):

                    vs['blue'][1]  =  1
                    ra[1]=0
                    ra[0]=1


                elif vc['blue'][1] ==  1 and ra[2]:

                    vs['blue'][2]  =  1
                    ra[1]=1
                    ra[2]=0

                elif vc['blue'][2]  == 1 and ra[0]:

                    vs['blue'][0]  =  1
                    startBlue=time()
                    ra[2]=1
                    ra[0]=0
```

```
    # ------------------------ GREEN ---------------------------------
-

            if vc['green'].count(1) != 0 :

                "Upravljački algoritam ZELENOG vlaka"

                if   vc['green'][0]  == 1  and ra[4] and ra[5] and
(not(vc['red'][5]==1)) and ((time()-startRed2)>7):

                    vs['green'][1]  =  1
                    ra[3]=1
                    ra[4]=0

                elif vc['green'][1] ==  1 and ra[5]:
                    vs['green'][2]  =  1
                    ra[4]=1
                    ra[5]=0

                elif vc['green'][2]  == 1:

                    vs['green'][0]   = 1
                    startGreen=time()
                    ra[5]=1
                    ra[3]=0

    # ------------------------ RED ---------------------------------

            if vc['red'].count(1) !=  0 :

                "Upravljački algoritam CRVENOG vlaka "

                if vc['red'][0]  == 1 and ra[1] and ra[2] and (time()-
startBlue)>7:

                    vs['red'][1]  =   1
                    ra[6]=1
                    ra[2]=0


                elif vc['red'][1] == 1 and ra[1]:

                    vs['red'][2]   = 1
                    ra[2]=1
                    ra[1]=0

                elif vc['red'][2] == 1 and ra[7]:

                    vs['red'][3]   = 1
                    startRed1   = time()
                    ra[1]=1
                    ra[7]=0

                elif vc['red'][3] == 1 and ra[8]:

                    vs['red'][4]   = 1
                    ra[7]=1
                    ra[8]=0
```

```
                elif vc['red'][4] == 1 and ra[9]:

                    vs['red'][5]   = 1
                    ra[8]=1
                    ra[9]=0

                elif vc['red'][5] == 1 and ra[4] and ra[5] and
(time()-startGreen)>7:

                    vs['red'][6]   = 1
                    ra[9]=1
                    ra[5]=0

                elif vc['red'][6] == 1 and ra[4]:

                    vs['red'][7]   = 1
                    ra[5]=1
                    ra[4]=0

                elif vc['red'][7] == 1 and ra[6]:

                    vs['red'][0]   = 1
                    ra[4]=1
                    ra[6]=0
                    startRed2   = time()



        #SPREMANJE (zamijeniti x imenom varijable)
        self.startRed1 = startRed1
        self.startRed2=startRed2
        self.startGreen=startGreen
        self.startBlue=startBlue
        self.ra=ra

        ############################## KRAJ
###################################

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

        self.state.check_timer()

        for color in vc:
            if (vc[color] != [0]*len(vc[color])) and (self.tag[color]
== 0):
                self.vc_time[color].append(time())
                print 'vc: %s' %str(self.state.vc)
                self.tag[color] = 1

        self.state.vs = vs

        for color in vs:
            if vs[color] != [0]*len(vs[color]):
                print 'vs: %s' %str(self.state.vs)
                self.state.start_timer(color)
                self.translate(color)
```

```
                self.vs_time[color].append(time())
                self.tag[color] = 0

        if (self.first_scan == 1):
            self.first_scan = 0
```