**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ORGANIZACIJE I INFORMATIKE**
**V A R A Ž D I N**

**Juraj Rasonja**

# PAMETNA KUĆA KAO UČEĆI VIŠEAGENTNI SUSTAV

**DIPLOMSKI RAD**

**Varaždin, 2014.**

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ORGANIZACIJE I INFORMATIKE**
**V A R A Ž D I N**

**Juraj Rasonja**

**Matični broj: 40552/11–IZV**

**Studij: Baze podatak, baze znanja**

# PAMETNA KUĆA KAO UČEĆI VIŠEAGENTNI SUSTAV

**DIPLOMSKI RAD**

**Mentor:**
Doc.dr.sc. Markus Schatten

**Varaždin, prosinac 2014**

# Sadržaj

# 1. Uvod

Kao i što svemu ostalome dođe kraj, tako dođe kraj mome studiranju. Za kraj svog studiranja, odnosno diplomski rad, želio sam napraviti projekt koji bi se mogao dalje nadograđivati, usavršavati i primjeniti u praktičnoj primjeni. Tako mi došla ideja o izradi sustava koji će se biti "mozak" pametne kuće. Sustav bi u konačnici morao samostalno obavljati zadatke uz što manji broj korisničke interakcije. Ujedno, sustav treba moći prepoznati ponašanje korisnika te iz njegovog ponašanja izvući pravila koja će se primijeniti u njegovu bazu pravila prema kojima se sustav ponaša.

U uvodu bih htio naznačiti razliku dva pojma koji imaju slična značenja. Automatizirana kuća prestavlja kuću u kojoj su znani repetativni zadaci automatizirani te se počinju izvršavati prisustvom signala iz okoline[1]. Tako na primjer postoji sustavi gdje se vanjska rasvjeta pali u određeno vrijeme, te se gasi u neko drugo određeno vrijeme. Sustav za grijanje prostora gdje se grijanje uključuje kada sobna temperatura padne ispod određene temperature. Isto tako sustav gasi grijanje kada sobna temperatura bude iznad zadane temperature. To su svi sustavi kojima je unaprijed definirano ponašanje te se promjene u ponašanju mijenjaju samo korisničkim upravljanjem. Pametna kuća, za razliku od automatizirane kuće, može zaključiti kako se korisnik ponaša i primijeniti pravila prilikom upravljanja kućom. Najednostavniji primjer komponente pametne kuće je prepoznavanja kada sustav treba upaliti svijetlo. Sustav prema senzorima koji se nalaze u prostoru prepoznaje da je korisnik upalio svjetlo, no ono samo treba zaključiti zašto je korisnik upalio svjetlo. Prema čitanjima iz senzora, sustav može zaključiti da ako je osvjetljenje manje od određene razine da je tada potrebno uključiti rasvjetu, te obrnuto ukoliko treba isključiti rasvjetu.

Kroz diplomski rad, navesti ću teoriju koja podupire rad sustava. Metoda koju koristim za učenje sustava novim pravilima se naziva induktivno logičko programiranje koja je jedna metoda koje pripadaju strojnom učenju. Metoda u suštini iz osnovnih činjenica koje prikupi iz okoline pomoću različitih senzora zaključuje pravila pomoću indukcije.

Također ću opisati programske jezike te programe koje sam koristio prilikom implementacije sustava. Veći dio projekta je baziran na Python skriptnom programskom jeziku. U njemu je implementirana platforma SPADE koji služi za razvoj višeagentnih

[1] Loseto G., Scioscia F., Ruta M., Di Sciascio E. *Semantic-base Smart Homes: a Multi-Agent Approach*

sustava. Za stvaranje hipoteza na temelju prikupljenih činjenica koristio sam program GOLEM kojeg poziva agent koji u sebi sadrži činjenice na temelju kojih treba napraviti hipotezu. Te na kraju XSB kojemu ću postavljati upite što treba agent napraviti.

# 2. Teorija

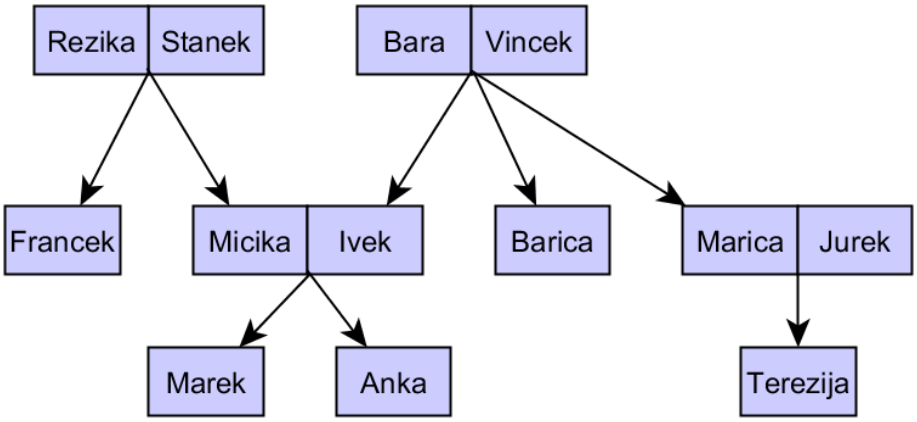
Prije definiranje same teorije koja je korištena u diplomskom radu, spomenuti ću simbole i pojmove koji će se koristiti radi lakšeg snalaženja i razumijevanja.

Naredna tablica sadrži osnovne simbole koji se koriste.

| **Simbol** | **Pojašnjenje** |
|---|---|
| $\top$ | Logička istina (tautologija) |
| $\bot$ | Logička laž (kontradikcija) |
| *a, b, c, d, e...* | Članovi skupa, elementi, činjenice |
| *A, B, C, D, E ...* | Skupovi |
| *U* | Univerzalni skup u kojem su sadržane sve činjenice. |

**Tabela 1. Osnovni simboli**

Prilikom opisivanja teorije koristi ću klasični primjer kojeg većina autora koji su navedeni u literaturi. Riječ je o prorodičnom stablu gdje između osoba postoje rodbinske veze. Formalno rečeno, kod porodičnog stabla između članova postoje relacije poput : *muško, žensko, kćerOd, sinOd, otacOd, majkaOd, roditeljOd, predakOd, potomakOd*, itd. S obzirom na navedene relacija postoje unarne i binarne relacije ovisno o arnosti relacije, odnosno na broj argumenata koje relacija može imati. Tako su relacije *muško* i *žensko* unarne relacije, dok su ostale navedene relacije binarne. U daljnjem tekstu i primjerima neću koristiti hrvatske dijakletičke znakove u imenima objekata zbog nekompaktibilnosti sa programskim alatima koje koristim u diplomskom radu.




**Slika 1. Primjer porodičnog stabla**

Primjer porodičnog stabla prikazan slikom Slika 1. određuje rodbinske veze između članova porodice. Pri vrhu porodičnog stabla se nalaze pretci, a pri dnu potomci. Tako su Rezika i Stanek roditelji Franceka i Micike, Marica i Jurek imaju kćer Tereziju, Anka je potomak Staneka, itd.

Kod dijela teorije gdje će se pokazivati kako se iz činjenica dobiva hipoteza biti će prikazano na jednostavnom primjeru za učenje relaciju *kcerOd*. Za učenje hipoteze koristiti će se činjenice i pozadinsko znanje navedene u Tabeli 2. Činjenice su definirane prema porodičnom stablu prikazano Slikom 1.

Kao rezultat stvaranje hipoteze prema primjeru iz Tablice 2, potrebno je dobiti hipotezu koja će biti definirana na sljedeći način:

*kcerOd(X,Y) :- zensko(X),roditeljOd(Y,X).*

Navedena hipoteza vrijedni za sve pozitivne činjenice, jer tako mora biti. Također, hipoteza ne smije vrijediti za negativne činjenice jer u suprotnome hipoteza nije valjana. Pozadinsko znanje se koristi kod konstrukcije hipoteze. Pozadinsko znanje donosi dodatno znanje na temelju kojeg će algoritam moći stvoriti nove hipoteze.

| Pozitivne činjenice | Negativne činjenice | Pozadinsko znanje |
|---|---|---|
| *kcerOd(micika,stanek)* | *kcerOd(francek,stanek)* | *roditeljOd(stanek,micika)* |
| *kcerOd(anka,micika)* | *kcerOd(rezika,micika)* | *roditeljOd(stanek,francek)* |
| *kcerOd(anka,ivek)* | *kcerOd(micika,anka)* | *roditeljOd(rezika,mocika)* |
| *kcerOd(barica,bara)* | *kcerOd(rezika,anka)* | *roditeljOd(rezika,francek)* |
| *kcerOd(marica,vincek)* | *kcerOd(vincek,francek)* | *roditeljOd(micika,anka)* |
| | | *roditeljOd(micika,marek)* |
| | | *roditeljOd(bara,barica)* |
| | | *roditeljOd(bara,ivek)* |
| | | *roditeljOd(jurek,terezija)* |
| | | *roditeljOd(marica,terezija)* |
| | | *zensko(rezika)* |
| | | *zensko(micika)* |
| | | *zensko(anka)* |
| | | *zensko(bara)* |
| | | *zensko(barica)* |
| | | *zensko(marica)* |
| | | *zensko(terezija)* |
| | | *musko(stanek)* |
| | | *musko(francek)* |
| | | *musko(ivek)* |
| | | *musko(marek)* |
| | | *musko(vincek)* |
| | | *musko(jurek)* |

**Tabela 2. Početne činjenice i pozadinsko znanje**

## 2.1. Učenje koncepta indukcijom

Definicije su preuzete iz Lavrač i Džeroski (1994)[2]

**Učenje koncepta *C* indukcijom.** Za dani set *E* pozitivnih i negativnih primjera koncepta *C*, treba pronaći hipotezu *H*, izraženu u zadanom jeziku za opis koncepta *L*, tako da vrijedi:

- svaki pozitivni primjer $e \in E^+$ je pokriven sa *H*
- ni jedan negativni primjer $e \in E^-$ nije pokriven sa *H*

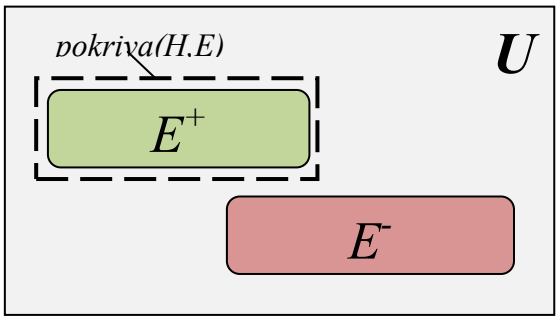
Za testiranje pokrivenosti činjenice sa hipotezom, potrebno je uvesti funkciju s kojom određujemo da li je činjenica *e* pokrivena sa hipotezom *H*. Za potrebe testiranja možemo definirati funkciju *pokriva(H,e)* koja će za rezultat vratiti ⊤ ako je *e* pokriveno sa *H*, a u suprotnome vratiti ⊥.

Fukncija *pokriva(H,e)* se može proširiti na takav način da se može primijeniti i na skupove. Tada definiramo funkciju *pokriva(H,E)* na sljedeći način:

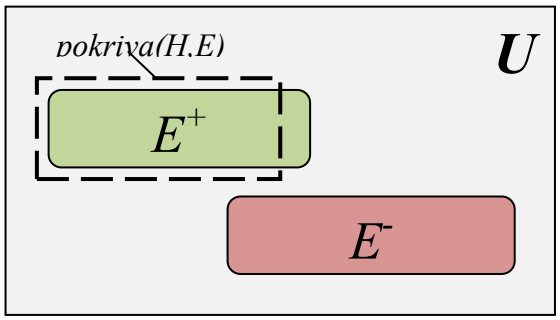
$pokriva(H,E) = \{ e \in E \mid pokriva(H,e) = \top \}$

Prilikom učenja koncepta indukcijom, hipoteza *H* mora pokrivati sve pozitivne primjere iz skupa primjera *E*. Istodobni, ne smije pokrivati niti jedan negativni primjer iz skupa primjera *E*. Za takvu hipotezu *H* kažemo da je potpuna i konzistetna s obzirom na primjere *E*. Za hipotezu *H* kažemo da je potpuna s obzirom na primjere *E* ukoliko pokriva sve pozitivne primjere $E^+$, odnosno ukoliko vrijedi $pokriva(H,E) = E^+$ . Za hipotezu *H* kažemo da je konzistentna s obzirom na primjere *E* ukoliko ne pokriva ni jedan negativni primjer iz $E^-$, odnosno ukoliko vrijedi $pokriva(H,E) = \emptyset$. Naredne slike prikazuju odnos između stanja hipozete s obzirom na potpunost i konzistentnost.

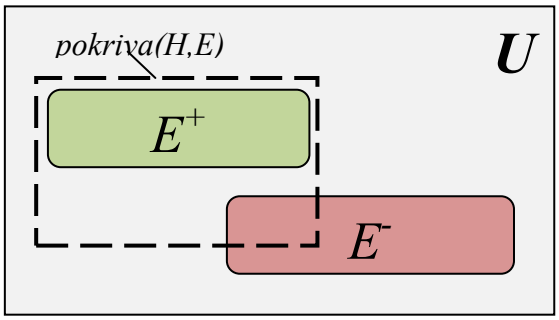
[2] Lavrač, N. And Džeroski. S.(1994). Inductive Logic Programming: Tehniques and Applications.



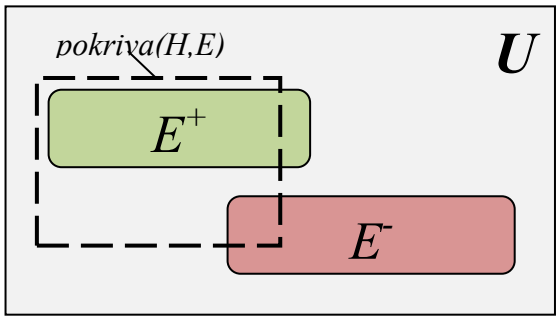

Hipoteza H – potpuna i konzistentna




Hipoteza H – nepotpuna i konzistentna




Hipoteza H – potpuna i nekonzistentna




Hipoteza H–nepotpuna i nekonzistentna

Prethodno navedeno, ne mora biti uvjet da bi se koncept naučio iz skupa primjera. Također, prema definiciji učenja koncepta indukcijom, hipoteza može pokrivati primjere koji nisu u skup primjera *E* za učenje koncepta *C*.

Kako je cilj učenja koncepta indukcijom klasificirati neklasificirane objekte s obzirom na koncept *C*, za hipotezu *H* možemo reći možemo reći da nam služi kao alat za klasificiranje novih objekata. Lavrač i Džeroski (1994)[3] definiraju kriterije, odnosno mjere za uspješnost sustava za učenje

**Preciznost klasifikacije** je postotak objekata koji su točno klasificirani pomoću hipoteze *H*.

**Transparentnost** je mjera za hipotezu H koja označava koliko je ona jasna i razumljiva ljiduma. Kao moguća mjera može biti duljina hipoteze izražena kao ukupni broj uvjeta koji definiraju hipotezu.

**Statistička signifikantnost** je mjera pomoću koje testiramo da li hipoteza H stvarno predstavlja pravilnost u primjerima za učenje, a ne slučajnu pravilnost.

[3] Lavrač, N. And Džeroski. S.(1994). Inductive Logic Programming: Tehniques and Applications.

**Sadržaj informacija** ili razina relativne informacije za hipotezu H označava uspješnost klasificiranja s obzirom na težinu koncepta kojeg treba naučiti. Treba uzeti u obzir vjerovatnost da se koncept *C* može naučiti, odnosno postotak objekata koji pripadaju konceptu *C* u odnosu sa sve objekte iz *U*.

## 2.2. Pozadinsko znanje

Do sada je bilo samo govora o učenju koncepta indukcijom samo na temelju pozitivnih i negativnih primjera s kojima raspolažemo. U tom slučaju koncept se pokušao naučiti iz primjera bez dodatnog znanja, odnosno pravila koja se mogu primjeniti prilikom učenja. Sljedeće što možemo napraviti je uz pozitivne i negitvne primjere dodati i dodatna pravila kako bi se olakšalo dobivanje hipoteze za koncept koji se želi naučiti. Samim time, postupkom učenja koncepta indukcijom, hipoteza neće sadržavati samo pravila koja su rezultat postupka, nego već i unaprijed definirana pravila ili činjenice.Također na takav način, hipoteza može biti jasnija ljudima prilikom čitanja jer smo pravila sami dodali u pozadinsko znanje. Ta dodatna pravila se nazivaju pozadinsko znanje. Sada učenju koncepta indukcijom dodajemo pozadinsko znanje.

**Učenje koncepta indukcijom sa pozadinskim znanjem[4].** Za dani skup primjera *E* i pozadinsko znanje *B*, potrebno je pronaći hipotezu *H* koja je izražena u nekom od odabranom opisnom jeziku *L*, tako da je hipoteza *H* kompletna i koncistentna s obzirom na pozadinsko znanje *B* i primjere *E*.

Prema ovoj definici može se proširiti funkcija pokriva na sljedeći način:

*pokriva(B, H, e) = pokriva(B ∪ H, e)* i

*pokriva(B, H, E) = pokriva(B ∪ H, E).*

Također možemo promijeniti definiciju za potpunost i konzistentnost

**Potpunost.** Hipoteza H je potpuna s obzirom na pozadinsko znanje B i skup primjera E ako su svi pozitivni primjeri pokriveno, odnosno $pokriva(B, H, E^+) = E^+$ .

**Konzistentnost.** Hipoteza H je konzistentna u odnosu na pozadinsko znanje B i skup primjera E ako ni jedan negativni primjer nije pokrivem, odnosno $pokriva(B, H, E^-) = Ø$.

Za jezik L navedenog u definicije koristim Hournove klauzule koje se koriste u Prologu.

[4] Lavrač, N. And Džeroski. S.(1994). Inductive Logic Programming: Tehniques and Applications

## 2.3. Induktivno logičko programiranje

Iduktivno logičko programiranje je jedna od metoda strojnog učenja.

### 2.3.1. Dimenzije induktivnog logičkog programiranja

Prema Lavrač i Džeroski 1994[5] definirane su dimenzije sustava za induktivno logičko programiranje. Sustavi se mogu podijeliti na sljedeće načine:

- S obzirom na koliki broj koncepata može učiti, odnosno da li može naučiti jedan koncept ili više koncepata
- Da li su sustavu potrebni primjeri iz kojih će učiti prije samog procesa učenja ili se primjeri unose jedan po jedan
- Da li je procesu učenja potrebna potvrda ispravnosti generalizacije i/ili prema klasificiranim primjerima generiranih od strane procesa učenja, odnosno interaktivni sustavi i neinteraktivni sustavi
- Proces učenja može započeti sa početnom hipotezom koja se tokom procesa provjerava ili može sam stvoriti hipotezu

Sustavi koji trenutno postoje su uglavnom podijeljeni u dvije dijametralno suprotne skupine. Tako postoje sustavi koji se baziraju na velikim brojem primjera za učnje, bez prisustva interakcije i uči samo jednu hipotezu bez početne hipoteze. Dok su sa druge strane interaktivni sustavi koji provjeravaju više hipoteza dodavajući im dodatne primjere za učenje.

Prema De Raedt [1992][6] prvi tip sustava nazivamo empiriskim ILP sustavi, dok drugi tip sustava nazivamo interaktivni ILP sustavi.

Jedan od primjera empirijskog ILP sustava je GOLEM [Muggleton i Feng 1990][7] koji koristim u implementaciji višeagentnog sustava.

---

[5] Lavrač, N. And Džeroski. S.(1994). Inductive Logic Programming: Tehniques and Applications

[6] De Raedt, L. (1992). Ineractive Theory Revision: An Inductive Logic Programing Approach

[7] Muggleton, S. And Feng. C.(1990). Efficient induction of logic programs. In Proc. First Conference on Algorithmic Learning Theory, stranice 368-381.

### 2.3.2. Terminologija logičkog programiranja

Kao bi se opisalo teorija induktivno logičkog programiranja, najprije je potrebno defirnirati osnovne pojmove koji će se koristiti. U izvoru Lavrač i Džeroski 1994[8] definirani su sljedeći pojmovi

**Abeceda** prvog reda se sastoji on varijabli, predikata i funkcija.

**Varijabla** se označava prvim velikim slovom, dok ostala slova mogu biti mala ili brojevi.

**Funkcija** je označena sa prvim malim slovom, dok ostala slova mogu biti mala ili brojevi.

**Predikat** je označena sa prvim malim slovom, dok ostala slova mogu biti mala ili brojevi.

**Hornova klauzula** je klauzula koja sadrži više od jednog pozitivnog literala.

**Definitivna programska klauzula** je klauzula koja sadrži samo jedan pozitivni literal i oblika je

*T <- L1,L2,....,Lm*

Gdje *su T, L1,...,Lm* atomi.

**Definitivni program** je skup definitivnih programskih klauzula

**Programska klauzula** je klauzula oblika

*T <- L1,...,Lm*

Gdje je *T* atom i svaki of *L1,...Lm* je oblika *L* ili not *L*, gdje je *L* atom.

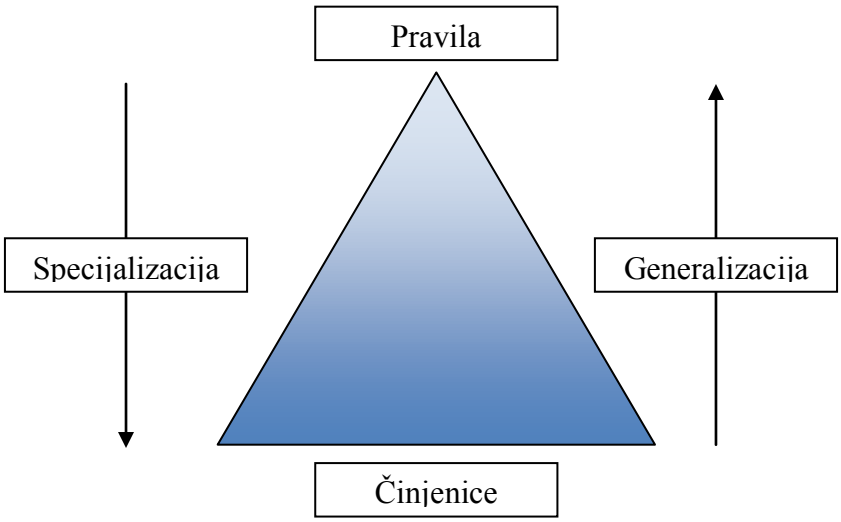
**Normalni program** je skup programskih klauzula.

**Definicja predikata** je se programskih klauzula sa istim predikatom i arnosti glave.

[8] Lavrač, N. And Džeroski. S.(1994). Inductive Logic Programming: Tehniques and Applications

### 2.3.3. Relativno najmanja generalna generalizacija

Traženje hipoteze se može izvoditi pomoću generalizacije ili specijalizacije. Kod generalizacije postupak koristi činjenice kako bi došao do generalne hipoteze, odnosno postupak ide od odozdo prema gore. Dok specijalizacija traži hipotezu na način da počinje od generalnog koncepta do specijalnog koncepta, odnosno od odozgo prema dolje.




Slika . Odnos između generalizacije i specijalizacije

Relativno najmanja generalna generalizacija (eng. Relative least general generalization, *rlgg*) koristi generalizaciju za dobivanje hipoteza.

Naprije moramo definirati osnovne funkcije koje se koriste u navedenoj metodi.

**Substitucija** $\Theta = \{X_1/t_1,...,X_k/t_k\}$ je funkcija koja zamjenju je varijable sa odgovarajućim termovima. Za neku klauzulu, substitucija zamjenjuje sve pojave varijable $X_i$ sa istim termom $t_i$.

***Θ*-subsumpcija**. Neka su c i c' dvije klauzule. Klauzula c *Θ*-subsumpcira c' ako postoji takva substitucija *Θ* da vrijedi $c\Theta \subseteq c'$. Dvije klauzule c i d su ekvivalentne prema *Θ*-subsumpciji ako c *Θ*-subsumpcira d i d *Θ*-subsumpcira c. Klauzula je reducirana ako nije ekvivalentna prema *Θ*-subsumpciji samoj sebi ili podskupu sebe.

*Θ*-subsumpcija omogućava uvođenje notacije za generalnost. Tako možemo reći da klauzula c je najmanje generalna kao klauzula c' (c < c') ako c *Θ*-subsumpcira c'. Klauzula c je više generalna nego c' (c < c') ako c < c' vrijedi i c' < c ne vrijedi. U tom slučaju kažemo da je c' specijalizacija klauzule c, odnosno c je generalizacija c'. Dodatno možemo reći da ukoliko

su klauzule c i d $\Theta$-subsumpcirano ekvivalentne, reducirana klauzule je ona koja ima manji broj termova u tijelu klauzule.

Dodatno možemo uvesti relaciju $\leq$ na skupu reduciranih klauzula. Sa time možemo odrediti za bilo koje dvije klauzule njihovu najmanju gornju granicu i njihovu najveću donju granicu.

**Najmanja generalna generalizacija** (eng. Least general generalization, lgg) od dvije reducirane klauzule c i c' je najmanja gornja granica od c i c' s obzirom na $\Theta$-subsumpciju.

**Relativno najmanja generalna generalizacija** (*rlgg*) od dvije klauzule $c_1$ i $c_2$ je njihova najmanja generalna generalizacija $lgg(c_1,c_2)$ relativno s obzirom na pozadinsko znanje $B$.

# 3. Implementacija

Sustav je implementiran na način da koristi različite programe i programske jezike. Kostur sustava čini platforma Spade koji omogućava izradu više-agentnih sustava. Spade je implementiran u Pythonu stoga je prirodno koristiti Python kao programski jezik za implementaciju ostatka sustava. Dio vezan za logičko programiranje se oslanja na program XSB koji je implementacija logičkog programskog jezika Prolog. Dok za indukciju pravila se koristi program Golem.

## 3.1. Python

Programski jezik koji sam koristio pri implentaciji sustava je Python. To je skriptni programski jezik što znači da se postoji interpreter koji izvršava programski kod prilikom pokretanja programa. Python nema određeno područje za koje ga je najbolje koristiti i nema neki specifičan način kako se u njemu može programirati. On podržava različite paradigme programiranja poput objektno-orijentiranog, imperativnog, functionalnog i proceduralnog programiranja što programeru daje veliku slobodu prilikom izabira stila programiranja.

Python ima bogatu riznicu modula koji se mogu uključivati po potrebi što ga čini poželjnim ukoliko se programer ne želi zamarati programiranjem funkcionalnosti koja je već implementirana u nekom modulu. Programeru je dovoljno uključiti modul u svoju skriptu i koristiti funkcionalnost po potrebi.

Python ima specifičnu sintaksu kada je u pitanju odjeljivanje blokova koda. U C, C++-u, Javi i srodnim programskim jezicima, koriste se vitičaste zagrade za odjeljivanje blokova koda, dok se u Pythonu koriste uvlake kao označavanje koji linija koda pripada kojem kodnom bloku. Takav način pisanja koda uvelike olakšava čitanje programsko koda. Kod pisanja programskog koda u drugim programski jezicima, mogu se koristiti uvlake radi čitljivosti koda što je uobičajena praksa pisanja dobrog koda, no uvlake nisu obavezni dio sintakse te ih interpreteri i prevoditelji igrnoriraju.

Prilikom pokretanja skripte, Python provjera sintaksu koda i prevodi ga u binarnu datoteku što ubrzava sljedeće pokretanje skripte. Ukoliko se programski kod promijeni, tada se skripta ponovno prevodi u binarni kod.

Primjer sadrži kratak programski kod koji je napisan u Pythonu.

***Primjer 1.:***

```
import random

def ispis(broj):
        for i in range(broj):
                print „ispis“, i

broj_ispisa = random.randint(1,20)

ispis(broj_ispisa)
```

Linija *import random* uključuje modul koji se koristi prilikom generiranja slučajnih brojeva i u primjeru se koristi funkcija *randint* za generiranje slučajnog cijelog broja. Definiranje funkcije (isto tako i metode ukoliko se radi o objektno-orijentiranom programiranju) započinje za ključnom riječi *def*. Poslije nje slijedi naziv funkcije, te u zagradi parametri koji se koriste prilikom poziva funkcije i na kraju lineje ide dvotočka. Nakon dvotočke ide novi red koji počinje sa uvlakom što označava da se sve sljedeće uvučene linije na istoj razini odnose na implementaciju funkcije. Sljedeća linija sadži *for i in range(broj):* što označava početak petlje. U ovoj specifičnoj linije, varijabli *i* će se za svaki korak petlje pridodijeliti vrijednost od raspona 0, pa sve do vrijednosti varijable *broj*. Sljedeća linija je još jedampun uvučena što znači da ona pripada petlji. Linija *print „ispis“, i* ispisuje na zaslon tekst *„ispis“* i vrijednost varijable *i*. Sljedeća linija *broj_ispisa = random.randint(1,20)* poziva funkciju *randint* iz modula *random* koja vraća slučajni broj između brojeva 1 i 20 te tu slučajnu vrijednost pridodjeljuje varijabli *broj_ispisa*. Zadnja linija poziva definiranu funkciju *ispis* i za parametar ima vrijednost varijable *broj_ispisa*. U konačnici, prilikom pokretanja programa na ekran će se ispisati slučajan broj linija sa odgovarajućim tekstom.

## 3.2. Spade

Platforma SPADE (Smart Python Agent Development Enviroment) je platforma koja omogućava razvoj višeagentnih sustava. SPADE također sadrži module koji se koriste za stvaranje novih agenata koji mogu koristiti platformu. Koristi standarde koje je utemeljila organizacija FIPA[9]( Foundation for Intelligent Physical Agents).

Spade koristi Jabber/XMPP protokol koji se koristi za komunikaciju između agenata. Taj se protokol također koristi u drugim poznatim alatima za tekstualnu komunikaciju. S ovim izborom protokola za komunikaciju, značajno je olakšana implementacija komunikacije između agenta i stvarne osobe.

Cijeli sustav se može administrirati preko Web sučelja. U sučelju se između ostalih stvari nalaze popis aktivnih agenata, poruke koje se izmjenjuju između agenata.

[9] FIPA (www.fipa.org)

## 3.3. XSB

XSB je implementacija logičkog programskog jezika Prolog. Program napisan u Prologu se sastoji od predikata, odnosno, činjenica i pravila. Činjenice predstavljaju istinite tvrdnje. Pravilo se sastoji od glave i tijela pravila. Kako bi se utvrdila istinitost glave, potrebno je provjeriti istinitost tijela pravila. Predikati završavaju sa točkom.

Program se pokreće na način ta se prvo pokrene sučelje XSB-a. Potom je potrebno učitati program, te nakon što se program učita, XSB nam daje mogućnost postavljanja upita nad programom.

Primjer 2 prikazuje jednostavni program koji broji elemente u ponuđenoj listi. Lista se definira sa uglatim zagradama i između se navede elementi odvojeni zarezom (npr. [a,b,c,d,e,f] ). Prva linija prestavlja činjenicu koja kaže da za prazna lista ima nula broj elemenata. Druga linija predstavlja pravilo koja pomoću rekurzije broji koliko elemenata ima u listi. [G|R] označava listu, ali na način da varijabla G predstavlja prvi element liste, a R označava ostatak liste. Tako sa rekurzijom u svakom koraku uzimamo glavu liste i tražimo broje elemenata u ostatku liste. Broj elemenata liste se u svakom koraku rekurzije povećava za jedan. Nakon što učitamo program u XSB, možemo postaviti upit tipa *broj_elemenata( [a, b, c, d, e, f], X).* XSB će vratiti *X = 6* kao odgovor na upit. Također možemo postaviti upit tipa *broj_elemenata([a, b, c, d, e, f], 10).* i XSB će vratiti *false* kao odgovor zato jer lista sadrži 6 elemenata ,a ne 10.

***Primjer 2.:***

```
broj_elemenata( [], 0).
broj_elemenata([G|R],C) :- broj_elemenata(R,C1), C is C1 + 1.
```

Primjer 3. je klasičan primjer obiteljskog stabla. U logičkom programu definirana su 3 predikata i to *roditeljOd/2*, *predak/2* i *potomak/2*. Predikat *roditeljOd/2* predstavlja odnos takav da je osoba A je roditelj osobi B i definiran je sa činjenicama. Predikat *predak/2* je definira pomoću dva pravila. Prvo pravilo kaže da je osoba A predak osobi B ukoliko je osoba A roditelj osobi B Ukoliko postavimo upit *predak(stanek,POTOMAK)*, XSB će kao rezultat vratiti sve osobe koje su potomci sosobi *stanek.*U programu je također definiran predikat *potomak/2* pomoću pravila *predak/2*. To smo mogli učiniti zato što je tako u stvarnosti. Recimo ako je *stanek* predak *mareku*, onda možemo reći da je *marek* potomak *staneku*. Zbog

toga ne mora se posebno pisati pravila za predikat *potomak/2* zato jer možemo iskoristiti već definirani predikat *predak/2*.

***Primjer 3.:***

```
roditeljOd(stanek,micika).
roditeljOd(stanek,francek).
roditeljOd(rezika,mocika).
roditeljOd(rezika,francek).
roditeljOd(micika,anka).
roditeljOd(micika,marek).
roditeljOd(bara,barica).
roditeljOd(bara,ivek).
roditeljOd(jurek,terezija).
roditeljOd(marica,terezija).

predak(A,B) :- roditeljOd(A,B).
predak(A,B) :- roditeljOd(A,X),predak(X,B).

potomak(A,B) :- predak(B,A).
```

## 3.4. GOLEM

Golem je alata za induktivno logičko programiranje kojeg je razvio Stephen Muggelton. Alat koristi za indukciju metodu relativno najmanju generalnu generalizaciju, odnosno rlgg.

Alat se jednostavno koristi kroz komandnu liniju. Prije pokretanja programa, potrebno je definirati ulazne datoteke koje sadrže sljedeće:

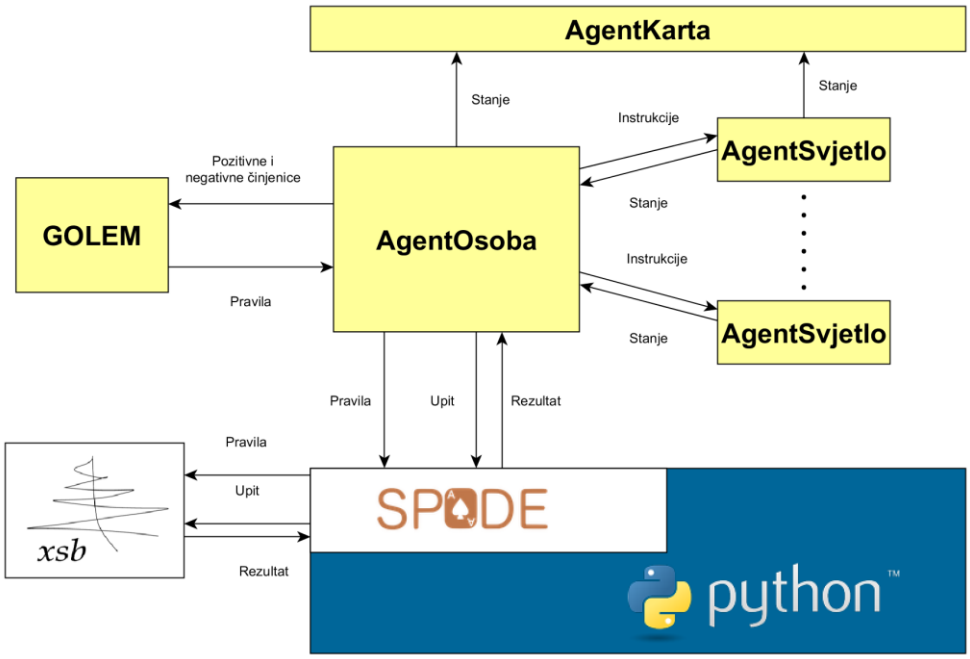
- Pozadinsko znanje (datoteka sa ekstenzijom *.b*)
- Pozitivni primjeri (datoteka sa ekstenzijom *.f*)
- Negativni primjeri (datoteka sa ekstenzijom *.n*)

Sve navedene datoteke moraju sadržavati činjenice koje odgovaraju sadržaju datoteke. Također sve datoteke moraju imati isti naziv, zato jer je tako program napravljen.

Program pokrećemo sa naredbom *golem <naziv_datoteke>* pri čemu je parametar samo naziv daoteke, bez njezine ekstenzije. Kao rezulta stvorena je četvrta datoteka sa istim nazivom, ali sa ekstenzijom *.r*. U stvorenoj datoteci nalaze se inducirana pravila pomoću rlgg metode. Dobivena pravila se lako mogu koristiti u XSB kako bi se nad njima izvršavali upiti.

## 3.5. Implementacija sustava

Slika prikazuje blok dijagram kako pojedini dijelovi sustava komuniciraju.



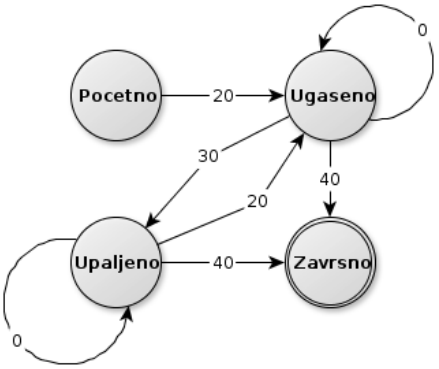

**Slika 2. Komunikacija unutar sustava**

### 3.5.1. Agenti

U sustavu postoje tri tipa agenta koji se koriste u sustavu. Svaki tip agenta je zadužen za svoj dio posla koji mora obaviti.

AgentSvjetlo je jednostavni agent koji ima četiri stanja kako je prikazano slikom 3. Za agent se može reći da se radi o konačno automatu. *Pocetno* stanje je stanje u kojem se agent inicijalizira i postavlja početne vrijednosti. Odmah poslije, agent prelazi u stanje *Ugaseno* i javlja agentu AgentKarta u kojem se stanju nalazi kako bi se na karti moglo prikazati. AgentSvjetlo prima dva tipa naredbi od agenta AgentOsoba. Jedna naredba je za upaliti svjetlo, odnosno *upali*. Prijelaz stanja za upaliti svjetlo je prikazano na slici sa prijelazom 30, što označava prijelaz iz *Ugaseno* u *Upaljeno* stanje, i sa prijelazom 0 ukoliko se svjetlo već nalazi u stanju *Upaljeno* i pri tome ne mora promijeniti stanje. Druga naredbaje je za ugasiti svjetlo, odnosno *ugasi*. Prijelaz stanja za ugasiti svjetlo je prikazano na slici sa prijelazom 20,

što označava prijelaz iz *Pocetno* ili *Upaljeno* stanje u *Ugaseno*, i sa prijelazom 0 ukoliko se svjetlo već nalazi u stanju *Ugaseno* i pri tome ne mora mijenjati stanje. U *Zavrsno* stanje agent prelazi kada je kraj programa i ne treba više primati naredbe. Trenutno u sustavu ima 16 soba, odnosno 16 agenata AgentSvjetlo koji su predstavljani sa slovima od *a* do *o*. Implementacija AgentSvjetlo se nalazi u Prilogu 1.




**Slika 3. AgentSvjetlo - stanja i prijelazi**

AgentKarta prima stanja ostalih agenata i iscrtava na karti. Implementacije agenta nalazi se u Prilogu 2. Od agenata koji predstvaljaju svijetlo, prima stanje svjetala. Od agenta AgentOsoba dobiva koordinate u kojoj se prostoriji nalazi.

AgentOsoba je agent koji simulira kretanje stvarne osobe i odlučuje da li treba upaliti , ugasiti svjetlo ili ništa učiniti ukoliko je svjeto već u željenom stanju. Agent uči pravila kako se ponašati kada se situaciju u okruženju promijeni. Agent dobiva instrukcije od korisnika u koju sobu treba prijeći i što treba napraviti sa svjetlom. Implementacija agenta se nalazi u Prilogu 3.

### 3.5.2. Učenje agenta

U početku agent nema nikakvo znanje što treba učiniti sa svjetlom, stoga pita korisnika što treba učiniti. Korisniku su ponuđene tri opcije:

- Upaliti svjetlo – naredba *upali*
- Ugasiti svjetlo – naredba *ugasi*
- Ostavi kako je – naredba *nop*

Nakon što korisnik odabere naredbu, stvaraju se činjenice na temelju kojih će se stvarati pravila za buduće radnje. Primjer činjenice izgleda *svjetlo( Lokacija, Vrijeme, StanjeSvjetla, Naredba).* pri čemu *Lokacija* označava sobu u kojoj se osoba nalazi, *Vrijeme* označava sat u kojem treba napraviti akciju, *StanjeSvjetla* označava stanje u kojem se nalazi svjetlo u trenutnoj prostoriji (prostoriji *Lokacija*), te *Naredba* označava koju naredbu treba učinit u trenutnom stanju. Činjenice se stvaraju imajući na umu da se radi o sustavu zatvorenog svijeta, što znači da će se za jednu odabranu naredbu napraviti takve činjenice da ta koja opisuje odabranu naredbu ulazi u skup pozitivnih primjera, dok ostale dvije činjenice (za ostale dvije naredbe) ulaze u skup negativnih činjenica. Ovakav pristup omogućava agentu brže učenje pravila po kojima bi trebao upravljati sa svjetlima. Tako na primjer u situaciji gdje se osoba nalazi u prostoriji c, u 23 sata, svjetlo u prostoriji je trenutno ugašeno, potrebno je upaliti svjetlo. Tako činjenica *svjetlo(c,23,ugaseno,upali).* ulazi u skup pozitivnih primjera (skup $E^+$ , odnosno u datoteku sa ekstenzijom *.f*), dok činjenice *svjetlo(c,23,ugaseno,ugasi).*i *svjetlo(c,23,ugaseno,nop).* ulazi u skup negativnih primjera (skup $E^-$ , odnosno u datoteku sa ekstenzijom *.n*). Kada se zapišu činjenice u datoteke, slijedi osvježavanje znanja, odnosno ponovno induciranja pravila na temelju proširenog skupa činjenica. U tom trenutku Golem čita činjenice iz datoteka i zapisuje inducirana pravila. Potom se inducirana pravila učitavaju u XSB putem Spade-a. Na taj način je agentu omogućeno raditi upite o budućim naredbama. Kada osoba promijeni prostoriju, agent postavlja upit XSB-u što treba učiniti. Na temelju prethodno definiranih činjenica i induciranih pravila, XSB vraća odgovor koju naredbu treba napraviti ili vrati varijalbu što označava da XSB ne zna što treba učiniti. Ukoliko je vraćena varijabla, postavk se izvršava kako je do sada opisano. Ukoliko XSB vrati konkretnu narebdu koju treba izvršiti, tada agent postavlja pitanje korisniku da li je naredba valjana. Ako je naredba valjana, tada agent šalje naredbu svjetlu i stvara nove činjenice na način kako je predhodno definirano. Ako naredba nije valjana, tada korisnik mora unijeti valjanu naredbu. Nakon unesene naredbe, ponavljaju se postupci kako su predhodno opisani.

# 4. Zaključak

Mogućnosti korištenja ovakvog pristupa implementaciju sustava za pametne kuće su višestruke. Pristup je moguće iskorisiti u sustavim gdje postoje kompliciraniji objekti za upravljanje poput televizijskog prijemnika, sustava grijanja na više različitih resursa i slično. Također sustav nije ograničen na kakvom će se uređaju izvršavati.

Kod ovakvih metoda se javlja problem ukoliko stanje sustava opisujemo sa puno dimenzija, pri čemu skup svih mogućih činjenica ekponencijalno raste sa brojem dimenzija. Ovakvi sustavi za induktivno logičko programiranje su u stanju stvoriti pravila na temelju manjeg skupa činjenica, no upitno je da li su pravila apsolutno točna. Naravno da sa povećanjem brojem činjenica, pravila postaju točnija.

Jedna od prepreka su postojeće implementacije sustava za induciranje. Većina sustava koji se nalaze na Internetu su implentirani u posebnim uvjetim u kojima autor nije naglasio koje parametre treba postaviti priliko prevođenja u izvršni kod, stoga ih je potrebno pogađati.

# 5. Literatura


[1] Aranda G., Palanca P. *SPADE User's Manual, For SPADE 2.1*. Preuzeto 16.08.2014. s http://pythonhosted.org//SPADE/

[2] Bratko I .(2012). *Prolog Programming for Artificial Intelligence*. Pearson, England.

[3] De Raedt, L. (1992). *Interactive Theory Revision: An Inductive Logic Programing Approach*. Academic Press,London

[4] Lavrač, N. And Džeroski. S.(1994). *Inductive Logic Programming: Tehniques and Applications*. Ellis Horwood, New York.

[5] Loseto G., Scioscia F., Ruta M., Di Sciascio E (2014). *Semantic-base Smart Homes: a Multi-Agent Approach*

[6] Lovrenčić S. (2008), *Prezetacije iz laboratoriskih vježbi iz kolegija „Uvod u formalne metode“*

[7] Muggleton, S. And Feng. C.(1990). *Efficient induction of logic programs. In Proc. First Conference on Algorithmic Learning Theory*, stranice 368-381. Ohmsha, Tokyo.

[8] Muggleton S. i De Raedt (1994). *Inductive logic programming: Theory and Methods.Journal of Logic Programming 1994:19,20*, stranice 629-679.Association for Logic Programming, London.

[9] Muggleton, S. ; *Induction Logic Programming – Theory*. Preuzeto 16.08.2014 s http://www.doc.ic.ac.uk/~shm/ilp_theory.html

[10] Muggleton, S.. *Inductive Logic Programming* . Preuzeto 16.08.2014 s www.doc.ic.ac.uk/~shm/Papers/ilp.pdf

[11] Muggleton, S. ; *Golem*. Preuzeto 16.08.2014 s http://www.doc.ic.ac.uk/~shm/Software/golem/

[12] Nienhuys-Cheng SH., De Wolf R. (1997), *Fundation of Inductive Logic Programming*. Springer, Berlin

[13] Schatten M., *Materijali za kolegiji Logičko programiranje*. Preuzeto 16.08.2014 s http://autopoiesis.foi.hr/wiki.php?name=Logi%C4%8Dko+programiranje+-+FOI

[14] *XSB – Logic Programming and Deductive Database system*. Preuzeto 16.08.2014 s http://xsb.sourceforge.net/

[15] *Python programm language*. Preuzeto 16.08.2014 s https://www.python.org/

[16] *SPADE: Smart Python multi-Agent Development Environment*. Preuzeto 16.08.2014 s https://pypi.python.org/pypi/SPADE

# 6. Prilog 1. Python skripta – automatskosvjetlo.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  import spade
4  import time
5  class AgentSvjetlo(spade.Agent.Agent):
6      class PocetnoStanje(spade.Behaviour.OneShotBehaviour):
7          def _process(self):
8              time.sleep(1)
9              self._exitcode = self.myAgent.PRIJELAZ_U_UGASENO
10
11
12     class UgasenoStanje(spade.Behaviour.OneShotBehaviour):
13         def _process(self):
14             time.sleep(1)
15             self._exitcode = self.myAgent.PRIJELAZ_PRETPOSTAVLJENI
16             if self.myAgent.akcija:
17                 if self.myAgent.akcija == "upali":
18                     self._exitcode = self.myAgent.PRIJELAZ_U_UPALJENO
19
20                 elif self.myAgent.akcija == "ugasi":
21                     self._exitcode =
                       self.myAgent.PRIJELAZ_PRETPOSTAVLJENI
22
23                 elif self.myAgent.akcija == "zatvori":
24                     self._exitcode = self.myAgent.PRIJELAZ_U_ZAVRSNO
25
26                 self.myAgent.akcija = None
27
28
29
30     class UpaljenoStanje(spade.Behaviour.OneShotBehaviour):
31         def _process(self):
32             time.sleep(1)
33             self._exitcode = self.myAgent.PRIJELAZ_PRETPOSTAVLJENI
34             if self.myAgent.akcija:
35                 if self.myAgent.akcija == "upali":
36                     self._exitcode =
                       self.myAgent.PRIJELAZ_PRETPOSTAVLJENI
37
38                 elif self.myAgent.akcija == "ugasi":
39                     self._exitcode = self.myAgent.PRIJELAZ_U_UGASENO
40
```

```
41                      elif self.myAgent.akcija == "zatvori":
42                              self._exitcode = self.myAgent.PRIJELAZ_U_ZAVRSNO
43
44                      self.myAgent.akcija = None
45
46
47      class ZavrsnoStanje(spade.Behaviour.OneShotBehaviour):
48          def _process(self):
49              time.sleep(1)
50              self.myAgent._kill()
51
52
53      class PosaljiStanje(spade.Behaviour.PeriodicBehaviour):
54          def _onTick(self):
55              primatelj = spade.AID.aid(name="karta@127.0.0.1",
                addresses=["xmpp://karta@127.0.0.1"])
56              poruka = self.myAgent.fsm._actualState
57
58              self.msg = spade.ACLMessage.ACLMessage()
59              self.msg.setPerformative("inform")
60              self.msg.setOntology("svjetlo_stanje")
61              self.msg.addReceiver(primatelj)
62              self.msg.setContent(poruka)
63
64              self.myAgent.send(self.msg)
65
66
67
68      class PorukeAkcija(spade.Behaviour.Behaviour):
69          def _process(self):
70              self.msg = None
71              self.msg = self._receive(True)
72              self.myAgent.akcija = self.msg.content.strip()
73
74
75      class PorukeStanje(spade.Behaviour.Behaviour):
76          def _process(self):
77              self.msg = None
78              self.msg = self._receive(True)
79              if self.msg.content.strip() == "upit":
80                      self.send_msg = None
81                      self.send_msg = spade.ACLMessage.ACLMessage()
82                      self.send_msg.setPerformative("inform")
83                      self.send_msg.setOntology("svjetlo_stanje")
84                      self.send_msg.addReceiver(self.msg.getSender())
```

```
85                      self.send_msg.setContent(self.myAgent.fsm._actualState)
86
87                      self.myAgent.send(self.send_msg)
88
89
90
91      def _setup(self):
92          time.sleep(2)
93          print self.getName(),self.lokacija
94
95          self.akcija = None
96
97
98          self.POCETNO_STANJE = 1
99          self.UGASENO_STANJE = 2
100         self.UPALJENO_STANJE = 3
101         self.ZAVRSNO_STANJE = 4
102
103         self.PRIJELAZ_PRETPOSTAVLJENI = 0
104         self.PRIJELAZ_U_UGASENO = 20
105         self.PRIJELAZ_U_UPALJENO = 30
106         self.PRIJELAZ_U_ZAVRSNO = 40
107
108         p = spade.Behaviour.FSMBehaviour()
109
110         p.registerFirstState(self.PocetnoStanje(), self.POCETNO_STANJE)
111         p.registerState(self.UgasenoStanje(), self.UGASENO_STANJE)
112         p.registerState(self.UpaljenoStanje(), self.UPALJENO_STANJE)
113         p.registerLastState(self.ZavrsnoStanje(), self.ZAVRSNO_STANJE)
114
115         p.registerTransition(self.POCETNO_STANJE, self.UGASENO_STANJE,
            self.PRIJELAZ_U_UGASENO)
116         p.registerTransition(self.UGASENO_STANJE, self.UGASENO_STANJE,
            self.PRIJELAZ_PRETPOSTAVLJENI)
117         p.registerTransition(self.UGASENO_STANJE, self.UPALJENO_STANJE,
            self.PRIJELAZ_U_UPALJENO)
118         p.registerTransition(self.UGASENO_STANJE, self.ZAVRSNO_STANJE,
            self.PRIJELAZ_U_ZAVRSNO)
119         p.registerTransition(self.UPALJENO_STANJE, self.UPALJENO_STANJE,
            self.PRIJELAZ_PRETPOSTAVLJENI)
120         p.registerTransition(self.UPALJENO_STANJE, self.UGASENO_STANJE,
            self.PRIJELAZ_U_UGASENO)
121         p.registerTransition(self.UPALJENO_STANJE, self.ZAVRSNO_STANJE,
            self.PRIJELAZ_U_ZAVRSNO)
122         self.addBehaviour(p, None)
123         self.fsm = p
```

```
124
125
126             akcija_template = spade.Behaviour.ACLTemplate()
127             akcija_template.setOntology("svjetlo_akcija")
128             at = spade.Behaviour.MessageTemplate(akcija_template)
129             pa = self.PorukeAkcija()
130             self.addBehaviour( pa, at )
131
132             stanje_template = spade.Behaviour.ACLTemplate()
133             stanje_template.setOntology("svjetlo_stanje")
134             st = spade.Behaviour.MessageTemplate(stanje_template)
135             ps = self.PorukeStanje()
136             self.addBehaviour( ps, st )
137
138             time.sleep(2)
139             p1 = self.PosaljiStanje(1)
140             self.addBehaviour(p1, None)
141
142 if __name__ == '__main__':
143     agenti=dict()
144
145     for i in range(16):
146             naziv = "%s@127.0.0.1"%chr(97+i)
147             lozinka = "tajna"
148             agenti[naziv] = AgentSvjetlo( naziv, lozinka )
149             agenti[naziv].lokacija=(i%4,i/4)
150             agenti[naziv].start()
```

# 7. Prilog 2. Python skripta – karta.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  import spade
4  import time
5  import os
6
7  class AgentKarta(spade.Agent.Agent):
8      class IspisKarte(spade.Behaviour.PeriodicBehaviour):
9          def _onTick(self):
10             os.system ('clear')
11
12             ispis = dict()
13             l = dict()
14             for i in range(97,113):
15                 a = chr(i)
16                 ispis[a] = " "
17                 l[a] = " "
18                 if a in self.myAgent.svjetla_stanja.keys():
19                     if self.myAgent.svjetla_stanja[a] ==
                       str(self.myAgent.SVJETLO_POCETNO_STANJE):
20                         ispis[a] = "P"
21                     if self.myAgent.svjetla_stanja[a] ==
                       str(self.myAgent.SVJETLO_UGASENO_STANJE):
22                         ispis[a] = "O"
23                     if self.myAgent.svjetla_stanja[a] ==
                       str(self.myAgent.SVJETLO_UPALJENO_STANJE):
24                         ispis[a] = "I"
25                     if self.myAgent.svjetla_stanja[a] ==
                       str(self.myAgent.SVJETLO_ZAVRSNO_STANJE):
26                         ispis[a] = "Z"
27             if self.myAgent.osoba_lokacija:
28                 l[self.myAgent.osoba_lokacija] = "X"
29
30
31             print "---------------------------------"
32             print "| a    | b    | c    | d    |"
33             print "|  %s  |  %s  |  %s  |  %s
               |"%(ispis["a"],ispis["b"],ispis["c"],ispis["d"])
34             print "|    %s |    %s |    %s |    %s |"%(l["a"],l["b"],l["c"],l["d"])
35             print "---------------------------------"
36             print "| e    | f    | g    | h    |"
37             print "|  %s  |  %s  |  %s  |  %s
               |"%(ispis["e"],ispis["f"],ispis["g"],ispis["h"])
```

```
38                  print "|    %s |    %s |    %s |    %s |"%(l["e"],l["f"],l["g"],l["h"])
39                  print "---------------------------------"
40                  print "| i    | j    | k    | l    |"
41                  print "|  %s  |  %s  |  %s  |  %s
                    |"%(ispis["i"],ispis["j"],ispis["k"],ispis["l"])
42                  print "|    %s |    %s |    %s |    %s |"%(l["i"],l["j"],l["k"],l["l"])
43                  print "---------------------------------"
44                  print "| m    | n    | o    | p    |"
45                  print "|  %s  |  %s  |  %s  |  %s
                    |"%(ispis["m"],ispis["n"],ispis["o"],ispis["p"])
46                  print "|    %s |    %s |    %s |    %s |"%(l["m"],l["n"],l["o"],l["p"])
47                  print "---------------------------------"
48
49      class PorukeSvjetloStanje(spade.Behaviour.Behaviour):
50          def _process(self):
51              self.msg = None
52              self.msg = self._receive(True)
53              if self.msg:
54                  self.myAgent.svjetla_stanja[self.msg.getSender().getName()[0]]
                    = self.msg.content
55              else:
56                  print "NISTA!!!"
57
58
59      class PorukeOsobaLokacija(spade.Behaviour.Behaviour):
60          def _process(self):
61              self.msg = None
62              self.msg = self._receive(True)
63              if self.msg:
64                  self.myAgent.osoba_lokacija = self.msg.content
65              else:
66                  print "NISTA!!!"
67
68
69      def _setup(self):
70          p = self.IspisKarte(2)
71          self.addBehaviour(p, None)
72
73          self.SVJETLO_POCETNO_STANJE = 1
74          self.SVJETLO_UGASENO_STANJE = 2
75          self.SVJETLO_UPALJENO_STANJE = 3
76          self.SVJETLO_ZAVRSNO_STANJE = 4
77
78          self.svjetla_stanja = dict()
79          self.osoba_lokacija = None
```

```
80
81              stanje_template = spade.Behaviour.ACLTemplate()
82              stanje_template.setOntology("svjetlo_stanje")
83              mt = spade.Behaviour.MessageTemplate(stanje_template)
84
85              pss = self.PorukeSvjetloStanje()
86              self.addBehaviour(pss,mt)
87
88              lokacija_template = spade.Behaviour.ACLTemplate()
89              lokacija_template.setOntology("osoba_lokacija")
90              lt = spade.Behaviour.MessageTemplate(lokacija_template)
91
92              pol = self.PorukeOsobaLokacija()
93              self.addBehaviour(pol,lt)
94
95
96
97  if __name__ == '__main__':
98      agent = AgentKarta("karta@127.0.0.1" , "tajna")
99      agent.start()
```

# 8. Prilog 3. Python skripta – osoba.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  import spade
4  import time
5  import os
6
7
8  class AgentOsoba(spade.Agent.BDIAgent):
9
10     class Upravljanje(spade.Behaviour.Behaviour):
11         def _process(self):
12             self.myAgent.akcija = None
13
14             self.myAgent.lokacija = raw_input("Unesi novu lokaciju:")
15             p = self.myAgent.PosaljiUpitStanje()
16             self.myAgent.addBehaviour(p, None)
17             p = self.myAgent.PosaljiLokaciju()
18             self.myAgent.addBehaviour(p, None)
19             self.myAgent.vrijeme = raw_input("Unesi sate (0-23):")
20
21
22             odgovori = self.myAgent.askBelieve("svjetlo({},{},{},Akcija)".format(
               self.myAgent.lokacija, self.myAgent.vrijeme,
               self.myAgent.svjetlo_stanje))
23             if odgovori:
24                         for odgovor in odgovori:
25                                     akcija = odgovor["Akcija"]
26                                     if akcija not in ["upali","ugasi","nop"]:
27                                                 continue
28                                     odg = raw_input("Akcija \"{}\" je ispravna
                                       (d/n):".format(akcija))
29
30                                     if odg == "d":
31                                                 self.myAgent.akcija = akcija
32                                                 break
33
34             if not self.myAgent.akcija:
35                         self.myAgent.akcija = raw_input("Unesi akciju
                           (upali/ugasi/nop):")
36
37             upali = "svjetlo({},{},{},{})".format( self.myAgent.lokacija,
               self.myAgent.vrijeme, self.myAgent.svjetlo_stanje, "upali")
```

```
38          ugasi = "svjetlo({},{},{},{})".format( self.myAgent.lokacija,
            self.myAgent.vrijeme, self.myAgent.svjetlo_stanje, "ugasi")
39          nop = "svjetlo({},{},{},{})".format(self.myAgent.lokacija,
            self.myAgent.vrijeme, self.myAgent.svjetlo_stanje, "nop")
40
41          if self.myAgent.akcija == "upali":
42                      print upali
43                      p = self.myAgent.PosaljiAkciju()
44                      self.myAgent.addBehaviour(p, None)
45                      self.myAgent.znanje["positive"].add(upali)
46                      self.myAgent.znanje["negative"].add(ugasi)
47                      self.myAgent.znanje["negative"].add(nop)
48          elif self.myAgent.akcija == "ugasi":
49                      print ugasi
50                      p = self.myAgent.PosaljiAkciju()
51                      self.myAgent.addBehaviour(p, None)
52                      self.myAgent.znanje["positive"].add(ugasi)
53                      self.myAgent.znanje["negative"].add(upali)
54                      self.myAgent.znanje["negative"].add(nop)
55          elif self.myAgent.akcija == "nop":
56                      print nop
57                      self.myAgent.znanje["positive"].add(nop)
58                      self.myAgent.znanje["negative"].add(upali)
59                      self.myAgent.znanje["negative"].add(ugasi)
60
61          print
62
63          self.myAgent.osvjeziZnanje()
64
65      class PosaljiLokaciju(spade.Behaviour.OneShotBehaviour):
66          def _process(self):
67          primatelj = spade.AID.aid(name="karta@127.0.0.1",
            addresses=["xmpp://karta@127.0.0.1"])
68          poruka = str(self.myAgent.lokacija)
69
70          self.msg = spade.ACLMessage.ACLMessage()
71          self.msg.setPerformative("inform")
72          self.msg.setOntology("osoba_lokacija")
73          self.msg.addReceiver(primatelj)
74          self.msg.setContent(poruka)
75
76          self.myAgent.send(self.msg)
77
78      class PosaljiAkciju(spade.Behaviour.OneShotBehaviour):
79          def _process(self):
```

```
80              naziv = self.myAgent.lokacija + "@127.0.0.1"
81              primatelj = spade.AID.aid(name=naziv,addresses=["xmpp://"+naziv])
82              poruka = self.myAgent.akcija
83
84              self.msg = spade.ACLMessage.ACLMessage()
85              self.msg.setPerformative("inform")
86              self.msg.setOntology("svjetlo_akcija")
87              self.msg.addReceiver(primatelj)
88              self.msg.setContent(poruka)
89
90              self.myAgent.send(self.msg)
91
92      class PosaljiUpitStanje(spade.Behaviour.OneShotBehaviour):
93          def _process(self):
94              naziv = self.myAgent.lokacija + "@127.0.0.1"
95              primatelj = spade.AID.aid(name=naziv,addresses=["xmpp://"+naziv])
96
97              self.msg = spade.ACLMessage.ACLMessage()
98              self.msg.setPerformative("inform")
99              self.msg.setOntology("svjetlo_stanje")
100             self.msg.addReceiver(primatelj)
101             self.msg.setContent("upit")
102             self.myAgent.send(self.msg)
103
104     class PrimiOdgovorStanje(spade.Behaviour.Behaviour):
105         def _process(self):
106             self.msg = None
107             self.msg = self._receive(True)
108             stanje = ""
109             if self.msg.content.strip() == "1":
110                                 stanje = "pocetno"
111             elif self.msg.content.strip() == "2":
112                                 stanje = "ugaseno"
113             elif self.msg.content.strip() == "3":
114                                 stanje = "upaljeno"
115             elif self.msg.content.strip() == "4":
116                                 stanje = "zatvoreno"
117             self.myAgent.svjetlo_stanje = stanje
118
119     def _setup(self):
120
121         stanje_template = spade.Behaviour.ACLTemplate()
122         stanje_template.setOntology("svjetlo_stanje")
123         st = spade.Behaviour.MessageTemplate(stanje_template)
124         pos = self.PrimiOdgovorStanje()
```

```
125         self.addBehaviour( pos, st )
126
127         time.sleep(2)
128
129         self.lokacija = "a"
130         self.svjetlo_stanje = None
131
132         p = self.PosaljiUpitStanje()
133         self.addBehaviour(p, None)
134
135         self.configureKB("XSB", None, "./XSB/bin/xsb")
136         self.znanje = self.procitajZnanje()
137         print self.znanje
138         for c in self.znanje["background"] | self.znanje["rules"]:
139             if ":-" in c:
140                                 self.addBelieve("("+c+")")
141             else:
142                                 self.addBelieve(c)
143
144         p = self.PosaljiLokaciju()
145         self.addBehaviour(p, None)
146
147         p1 = self.Upravljanje()
148         self.addBehaviour(p1, None)
149
150     def procitajZnanje(self):
151         znanje = dict()
152         znanje["background"] = self.procitajZnanjeIzDatoteke("znanje.b")
153         znanje["positive"] = self.procitajZnanjeIzDatoteke("znanje.f")
154         znanje["negative"] = self.procitajZnanjeIzDatoteke("znanje.n")
155         znanje["rules"] = self.procitajZnanjeIzDatoteke("znanje.r")
156         return znanje
157
158     def zapisiZnanje(self):
159         self.zapisiZnanjeUDatoteku(self.znanje["positive"],"znanje.f")
160         self.zapisiZnanjeUDatoteku(self.znanje["negative"],"znanje.n")
161
162     def procitajZnanjeIzDatoteke(self,datoteka):
163         temp = set()
164         znanje = set()
165         with open(datoteka, "r") as f:
166             temp.update(f.readlines())
167         for line in temp:
168             znanje.add(line[:-2])
169         return znanje
```

```
170
171     def zapisiZnanjeUDatoteku(self, znanje, datoteka):
172         with open(datoteka, "w") as f:
173             for line in znanje:
174                             f.write(line+".\n")
175
176
177     def osvjeziZnanje(self):
178         self.zapisiZnanje()
179         os.system("./golem znanje > /dev/null 2>&1")
180
181         novo_znanje = self.procitajZnanje()
182
183         treba_obrisati = self.znanje["rules"] - novo_znanje["rules"]
184         treba_dodati = novo_znanje["rules"] - self.znanje["rules"]
185
186         for c in treba_obrisati:
187             if ":-" in c:
188                             self.removeBelieve("("+c+")")
189             else:
190                             self.removeBelieve(c)
191         for c in treba_dodati:
192             if ":-" in c:
193                             self.addBelieve("("+c+")")
194             else:
195                             self.addBelieve(c)
196
197         self.znanje = novo_znanje
198
199 if __name__ == '__main__':
200     agent = AgentOsoba("osoba@127.0.0.1","tajna")
201
202     agent.start()
```

# 9. Prilog 4. Pozadinsko znanje – znanje.b

```
 1  noc(0).
 2  noc(1).
 3  noc(2).
 4  noc(3).
 5  noc(4).
 6  noc(5).
 7  noc(6).
 8  dan(7).
 9  dan(8).
10  dan(9).
11  dan(10).
12  dan(11).
13  dan(12).
14  dan(13).
15  dan(14).
16  dan(15).
17  dan(16).
18  dan(17).
19  dan(18).
20  dan(19).
21  dan(20).
22  noc(21).
23  noc(22).
24  noc(23).
```