Univerzitet Crne Gore
Elektrotehnički fakultet

Prof. dr Vesna Popović-Bugarin

PYKNOW – promjenljive, uparivanje šablona

Ograničavači vrijednosti polja Field Constraints

- Koriste se da nametnu ograničenje vrijednosti koje neki atribut/element može imati.
- Koriste se tri tipa ograničavača vrijednosti
- L (Literal Field Constraint) – traži tačno poklapanje vrijednosti polja sa vrijednošću zadatom ograničenjem
- *Primjer: Izvući činjenicu Fact čiji je prvi element 3*

```
@Rule(Fact(L(3))) # Rule(Fact(3))
def _():
    pass
```

- Ovo je podrazumijevani ograničavač polja, koji se koristi i ako nema ograničenja

Ograničavači vrijednosti polja
Field Constraints

```
[2]  from pyknow import *
```

```
[3]  class Ilustracija(KnowledgeEngine):
         @Rule(Fact(3))
         def Zadovoljeno(self):
             print("Zadovoljeno ogranicenje\n")
```

```
[4]  engine = Ilustracija()
     engine.reset()
     engine.declare(Fact(3))
```

```
Fact(3)
```

```
[5]  engine.run()
```

```
Zadovoljeno ogranicenje
```

Ograničavači vrijednosti polja
Field Constraints

- W (Wildcard Field Constraint) – ovo ograničenje zadovoljava element sa bilo kojom vrijednošću - Džoker
- Primjer: Zadovolji ukoliko je prisutna činjenica Fact sa bilo kojom neimenovanom vrijednošću prvog elementa.

```
@Rule(Fact(W()))
def _():
    pass
```

- Primjer: Zadovolji ukoliko je prisutna činjenica Fact sa ključem (key) mykey – bez obzira na njegovu vrijednost.

```
@Rule(Fact(mykey=W()))
def _():
    pass
```

Ograničavači vrijednosti polja

```
[45]  class Ilustracija(KnowledgeEngine):
          @Rule(Fact(W()))
          def Zadovoljeno(self):
              print("Zadovoljeno ogranicenje")
              print("Ima neki prvi element")
```

```
[46]  engine = Ilustracija()
      engine.reset()
      engine.declare(Fact(2))
      engine.declare(Fact("Neka vrijednost"))
```

```
      Fact('Neka vrijednost')
```

```
[47]  engine.run()
```

```
      Zadovoljeno ogranicenje
      Ima neki prvi element
      Zadovoljeno ogranicenje
      Ima neki prvi element
```

Ograničavači vrijednosti polja
Field Constraints

```
class Ilustracija(KnowledgeEngine):
    @Rule(Fact(vrijednost = W()))
    def Zadovoljeno(self):
        print("Zadovoljeno ogranicenje\n")
        print("Ima key vrijednost")
```

```
engine = Ilustracija()
engine.reset()
engine.declare(Fact(vrijednost = "Bilo sto"))
```

```
Fact(vrijednost='Bilo sto')
```

```
engine.run()
```

```
Zadovoljeno ogranicenje

Ima key vrijednost
```

Ograničavači vrijednosti polja
Field Constraints

- P (Predicate Field Constraint) – Zadovoljenje ovog elementa je rezultat primjene date funkcije na vrijednost uzetu iz pripadajuće činjenice. Ukoliko je rezultat funkcije true, zadovoljeno je ograničenje, u suprotnom nije
- Primjer: Zadovolji ograničenje ukoliko je prisutna činjenica Fact čiji je prvi element veći od nule

```
@Rule(Fact(P(lambda x: x >0)))

def _():

    pass
```

Ograničavači vrijednosti polja

```
[22]  class Ilustracija(KnowledgeEngine):
          @Rule(Fact(P(lambda x : x > 0)))
          def Zadovoljeno(self):
              print("Zadovoljeno ogranicenje\n")
              print("Ima prvi element veci od nule")
```

```
[23]  engine = Ilustracija()
      engine.reset()
      engine.declare(Fact(2))
```

```
      Fact(2)
```

```
[24]  engine.run()
```

```
      Zadovoljeno ogranicenje

      Ima prvi element veci od nule
```

Promjenljive - varijable

- Koriste se za smještanje vrijednosti.
- Koriste se u LHS pravila da bi se u njima sačuvala npr. vrijednost nekog atributa ili neki element, koji se kasnije koristi u nastavku LHS ili u RHS.
- Dodjeljivanje vrijednosti nekoj promjenljivoj se naziva vezivanjem vrijednosti za promjenljivu – engl. **bind** i **bound.**
- Kada se promjenljiva prvi put upotrijebi u nekom pravilu, biva joj dodijeljena vrijednost koju zadržava kroz cijelo pravilo. Njeno dalje navođene korišćenjem MATCH objekta ispituje da li je odgovarajuća vrijednost ključa / elementa identična kao vrijednost vezane promjenljive. **Ne može se promijeniti!!!**

Promjenljive - varijable

- MATCH objekti omogućavaju čitljivo vezivanje promjenljivih za vrijednosti nekog atributa, ili za element – ukoliko nema ključa atributa.
- Primjer. Za promjenjljivu *myvalue* veži prvi element

```
@Rule(Fact(MATCH.myvalue))
def _(myvalue):
    pass
```

- Je isto što i vezivanje operatorom <<, ali se gornje više koristi:

```
@Rule(Fact("myvalue" << W()))
def _(myvalue):
    pass
```

Promjenljive - varijable

```
[25]  class IlustracijaPromjenljive(KnowledgeEngine):
          @Rule(Fact(MATCH.vrijednost))
          def Zadovoljeno(self,vrijednost):
              print("Prvi element cinjenice je %s\n" %vrijednost)
```

```
[30]  engine = IlustracijaPromjenljive()
```

```
[31]  engine.reset()
      engine.declare(Fact(3))
      engine.declare(Fact("Neka vrijednost"))
```

```
      Fact('Neka vrijednost')
```

```
[32]  engine.run()
```

```
      Prvi element cinjenice je Neka vrijednost

      Prvi element cinjenice je 3
```

Promjenljive – varijable mogu biti vezane za vrijednost zadatog ključa

- Primjer. Za promjenjljivu *myvalue* veži vrijednost ključa *mykey*

```
@Rule(Fact(mykey = MATCH.myvalue))

def _(myvalue):

      pass
```

Za promjenljivu veži vrijednost zadatog atributa - ključa

```
[33] class IlustracijaPromjenljive(KnowledgeEngine):
         @Rule(Fact(kljuc = MATCH.ime_promjenljive))
         def Zadovoljeno(self,ime_promjenljive):
             print("Key kljuc ima vrijednost %s\n" %ime_promjenljive)
```

```
[34] engine = IlustracijaPromjenljive()
```

```
[35] engine.reset()
     engine.declare(Fact(kljuc = 3))
     engine.declare(Fact(kljuc = "Neka vrijednost"))
```

```
Fact(kljuc='Neka vrijednost')
```

```
[36] engine.run()
```

```
Key kljuc ima vrijednost Neka vrijednost

Key kljuc ima vrijednost 3
```

Print se koristi za štampanje formatiranog stringa

```
print(“Prva %s druga promjenljiva %s” %(a,b))
```

Promjenljive i adresa činjenice

- AS objekat se koristi za vezivanje adrese činjenice za željenu promjenljivu
- Nakon vezivanja adrese činjenice za neku promjenljivu, ta se promjenljiva može koristiti u svima naredbama koje očekuju indeks činjenice (modify, duplicate, retract).
- Primjer: Za promjenljivu *myfact* veži činjenicu Fact koja ima bilo koju vrijednost za prvi element

```
@Rule(AS.myfact << Fact(W()))
def _(myfact):
    pass
```

- Je isto kao, ali se gornje više koristi:

```
@Rule("myfact" << Fact(W()))
def _(myfact):
    pass
```

Promjenljive – varijable mogu biti vezane za adresu činjenice

```
watch("ACTIVATIONS","FACTS")
class IlustracijaAdresePromjenljive(KnowledgeEngine):
    @Rule(AS.f1 << Fact(MATCH.vrijednost))
    def Zadovoljeno(self,vrijednost,f1):
        print("Vrijednost prvog elementa cinjenice je %s" %vrijednost)
        self.retract(f1)
```

- watch omogućava praćenje aktivacija pravila, unošenja i izbacivanja činjenica, stanja agende i korisna je prilikom debagovanja programa i praćenja izvršavanja
- Obratiti pažnju na narednom slajdu da se prvo izvršava posljednje aktivirano pravilo

```
engine = IlustracijaAdresePromjenljive()
engine.reset()
engine.declare(Fact(3))
engine.declare(Fact("Neka vrijednost"))
```

```
INFO:pyknow.watchers.FACTS: ==> <f-0>: InitialFact()
INFO:pyknow.watchers.FACTS: ==> <f-1>: Fact(3)
INFO:pyknow.watchers.ACTIVATIONS: ==> 'Zadovoljeno': <f-1>
INFO:pyknow.watchers.FACTS: ==> <f-2>: Fact('Neka vrijednost')
INFO:pyknow.watchers.ACTIVATIONS: ==> 'Zadovoljeno': <f-2>
Fact('Neka vrijednost')
```

```
engine.run()
```

```
INFO:pyknow.watchers.FACTS: <== <f-2>: Fact('Neka vrijednost')
INFO:pyknow.watchers.ACTIVATIONS: <== 'Zadovoljeno': <f-2> [EXECUTED]
INFO:pyknow.watchers.FACTS: <== <f-1>: Fact(3)
INFO:pyknow.watchers.ACTIVATIONS: <== 'Zadovoljeno': <f-1> [EXECUTED]
Vrijednost prvog elementa cinjenice je Neka vrijednost
Vrijednost prvog elementa cinjenice je 3
```

TEST

- Omogućava dodatno postavljanje ograničenja za vrijednosti atributa / vrijednosti elemenata činjenice, izvršavanjem funkcije koja provjerava da li je zadovoljen zadati uslov. Uslovi se zadaju korišćenjem logičkih operatora i operatora poređenja, koji su isti kao u C, C++
- Primjer: Veži brojeve a,b,c, takve da je *a>b>c*

```
@Rule(Number(MATCH.a),
      Number(MATCH.b),
      TEST(lambda a, b: a > b),
      Number(MATCH.c),
      TEST(lambda b, c: b > c))
def _(a, b, c):
      pass
```

Vezivanje promjenljivih – primjer

- Realizovati ES kojim će se iz liste činjenica koje sadrže podatke o imenu, prezimenu, godinama, boji kose i očiju, izdvojiti osobe sa crnim očima.

```
from pyknow import *
class Osoba(Fact):
    pass
```

```
class Upari(KnowledgeEngine):
    @DefFacts()
    def NekiLjudi(self):
        yield(Osoba(ime = "Marko Markovic", godine = 35, kosa = "plava",
                    oci = "plave"))
        yield(Osoba(ime ="Lazar Andric", godine = 27, oci = "crne",
                    kosa = "crna"))
        yield(Osoba (ime = "Mirko Markovic", godine = 18, oci = "zelene",
                     kosa = "braon"))
        yield(Osoba (ime = "Igor Ivanovic", godine = 28, oci = "crne",
                     kosa = "plava"))
    @Rule(Osoba(ime = MATCH.ime,oci = "crne"))
    def ImaCrneOci(self,ime):
        print("%s ima crne oči \n" %ime)
```

```
unwatch("FACTS")
engine = Upari()
engine.reset()
engine.facts
```

```
INFO:pyknow.watchers.ACTIVATIONS: ==> 'ImaCrneOci': <f-2>
INFO:pyknow.watchers.ACTIVATIONS: ==> 'ImaCrneOci': <f-4>
FactList([(0, InitialFact()),
          (1,
           Osoba(ime='Marko Markovic', godine=35, kosa='plava', oci='plave')),
          (2, Osoba(ime='Lazar Andric', godine=27, oci='crne', kosa='crna')),
          (3,
           Osoba(ime='Mirko Markovic', godine=18, oci='zelene', kosa='braon')),
          (4,
           Osoba(ime='Igor Ivanovic', godine=28, oci='crne', kosa='plava'))])
```

```
engine.run()
```

```
Igor Ivanovic ima crne oči

Lazar Andric ima crne oči
```

- Izdvojiti osobe koje imaju boju očiju zadatu činjenicom Oci .

```
class Osoba(Fact):
   pass

class Oci(Fact):
  pass
class UpariZadatuBoju(KnowledgeEngine):
    @DefFacts()
    def NekiLjudi(self):
        yield(Osoba(ime = "Marko Markovic", godine = 35, kosa = "plava",
                    oci = "plave"))
        yield(Osoba(ime ="Lazar Andric", godine = 27, oci = "crne",
                    kosa = "crna"))
        yield(Osoba (ime = "Mirko Markovic", godine = 18, oci = "zelene",
                     kosa = "braon"))
        yield(Osoba (ime = "Igor Ivanovic", godine = 28, oci = "crne",
                     kosa = "plava"))
    @Rule(Oci(MATCH.boja),#za boja se veze vrijednost prvog elementa
              #cinjenice Oci
              Osoba(ime = MATCH.ime,oci = MATCH.boja))
          #ponovnim koriscenjem MATCH.boja se ispituje da
          #li je vrijednost kljuca oci jednaka vrijednosti
          #promjenljive boja
    def ImaTrazenuBojuOciju(self,ime,boja):
        print("%s ima %s oči" %(ime,boja))
```

- Za boja se veže vrijednost prvog polja činjenice Oci u prvom šablonu.
- Za ime se veže vrijednosti atributa ime one činjenice koja za vrijednost atributa oci ima vrijednost vezanu iz činjenice oči u promjenljivu boja

- Unosenjem cinjenice Oci, aktivirano je pravilo ImaTrazenuBojuOciju

```
engine = UpariZadatuBoju()
engine.reset()
engine.facts
```

```
FactList([(0, InitialFact()),
          (1,
           Osoba(ime='Marko Markovic', godine=35, kosa='plava', oci='plave')),
          (2, Osoba(ime='Lazar Andric', godine=27, oci='crne', kosa='crna')),
          (3,
           Osoba(ime='Mirko Markovic', godine=18, oci='zelene', kosa='braon')),
          (4,
           Osoba(ime='Igor Ivanovic', godine=28, oci='crne', kosa='plava'))])
```

```
engine.declare(Oci("crne"))
engine.run()
```

```
INFO:pyknow.watchers.ACTIVATIONS: ==> 'ImaTrazenuBojuOciju': <f-2>, <f-5>
INFO:pyknow.watchers.ACTIVATIONS: ==> 'ImaTrazenuBojuOciju': <f-5>, <f-4>
Igor Ivanovic ima crne oči
Lazar Andric ima crne oči
```

- Unosenjem cinjenice Oci, aktivirano je pravilo ImaTrazenuBojuOciju

```
engine.declare(Oci("plave"))
```

```
INFO:pyknow.watchers.ACTIVATIONS: ==> 'ImaTrazenuBojuOciju': <f-1>, <f-6>
Oci('plave')
```

```
engine.run()
```

```
Marko Markovic ima plave oči
```

Adresa činjenica

- Primjer 3. Kreirati ES kojim će se izvršiti modifikacija adrese osobe koja se preselila, a ime osobe i novu adresu zadaje korisnik.

```
from pyknow import *
class Osoba(Fact):
    pass
class Preseljen(Fact):
    pass
class Procesiraj_preseljenu_osobu(KnowledgeEngine):
    @Rule(AS.f1 << Preseljen(ime = MATCH.ime1,
                             adresa = MATCH.nova_adresa),
          AS.f2 << Osoba(ime = MATCH.ime1))
    def PromijeniAdresu(self,f1,f2,nova_adresa):
        print("Nova adresa  %s je %s" %(f2["ime"],nova_adresa))
        #smijem koristiti adresu prije izbacivanja cinjenice
        self.retract(f1)
        self.modify(f2,adresa = nova_adresa)
```

Primjer 3...

```
engine = Procesiraj_preseljenu_osobu()
engine.reset()
watch("AGENDA")
engine.declare(Osoba(ime = "Marko Markovic", adresa = "Cvijetna bb"))
engine.declare(Osoba(ime = "Nikola Markovic", adresa = "Cvijetna bb"))
engine.declare(Preseljen(ime = "Marko Markovic", adresa = "Studentska bb"))
engine.facts
```

```
FactList([(0, InitialFact()),
          (1, Osoba(ime='Marko Markovic', adresa='Cvijetna bb')),
          (2, Osoba(ime='Nikola Markovic', adresa='Cvijetna bb')),
          (3, Preseljen(ime='Marko Markovic', adresa='Studentska bb'))])
```

```
engine.run()
engine.facts
```

```
DEBUG:pyknow.watchers.AGENDA:0: 'PromijeniAdresu' '<f-1>, <f-3>'
Nova adresa  Marko Markovic je Studentska bb
FactList([(0, InitialFact()),
          (2, Osoba(ime='Nikola Markovic', adresa='Cvijetna bb')),
          (4, Osoba(ime='Marko Markovic', adresa='Studentska bb'))])
```

Primjer 3 - komentari

- Varijable se mogu vezati bilo za vrijednost nekog ključa iz činjenice, element činjenice, bilo za adresu činjenice.
- Uklanjanje činjenice *Preseljen* u RHS pravila *Procesiraj_preseljenu_osobu* je bitno da bi pravilo funkcionisalo ispravno. Pogledajmo šta bi se desilo ako bi se uklanjanje ove činjenice izostavilo:

```
[41]  from pyknow import *
      class Osoba(Fact):
          pass
      class Preseljen(Fact):
          pass
      class Procesiraj_preseljenu_osobu(KnowledgeEngine):
          @Rule(Preseljen(ime = MATCH.ime1,
                                  adresa = MATCH.nova_adresa),
                AS.f2 << Osoba(ime = MATCH.ime1))
          def PromijeniAdresu(self,f2,nova_adresa):
              print("Nova adresa  %s je %s" %(f2["ime"],nova_adresa))
              #smijem koristiti adresu prije izbacivanja cinjenice
              self.modify(f2,adresa = nova_adresa)
```

Primjer 3 – modifikovan (loše)

```
engine = Procesiraj_preseljenu_osobu()
engine.reset()
engine.declare(Osoba(ime = "Marko Markovic", adresa = "Cvijetna bb"))
engine.declare(Osoba(ime = "Nikola Markovic", adresa = "Cvijetna bb"))
engine.declare(Preseljen(ime = "Marko Markovic", adresa = "Studentska bb"))
watch("FACTS")
```

```
engine.run(3)
```

```
DEBUG:pyknow.watchers.AGENDA:0: 'PromijeniAdresu' '<f-1>, <f-3>'
INFO:pyknow.watchers.FACTS: <== <f-1>: Osoba(ime='Marko Markovic', adresa='Cvijetna bb')
INFO:pyknow.watchers.FACTS: ==> <f-4>: Osoba(ime='Marko Markovic', adresa='Studentska bb')
DEBUG:pyknow.watchers.AGENDA:0: 'PromijeniAdresu' '<f-3>, <f-4>'
INFO:pyknow.watchers.FACTS: <== <f-4>: Osoba(ime='Marko Markovic', adresa='Studentska bb')
INFO:pyknow.watchers.FACTS: ==> <f-5>: Osoba(ime='Marko Markovic', adresa='Studentska bb')
DEBUG:pyknow.watchers.AGENDA:0: 'PromijeniAdresu' '<f-3>, <f-5>'
INFO:pyknow.watchers.FACTS: <== <f-5>: Osoba(ime='Marko Markovic', adresa='Studentska bb')
INFO:pyknow.watchers.FACTS: ==> <f-6>: Osoba(ime='Marko Markovic', adresa='Studentska bb')
Nova adresa  Marko Markovic je Studentska bb
Nova adresa  Marko Markovic je Studentska bb
Nova adresa  Marko Markovic je Studentska bb
```

Primjer 3 - komentari

- Program ulazi u beskonačnu petlju jer unosi novu činjenicu nakon što modifikuje činjenicu vezanu za f2.
- Ovo će biti dovoljno za reaktiviranje pravila *Procesiraj-preseljenu-osobu*. Naime, aktiviraju ga dvije činjenice, od kojih je jedna svaki put nova (dobijena modifikovanjem stare).
- U slučaju da su obje činjenice ostale iste (sa istim identifikatorom), pravilo se ne bi ponovo aktiviralo.
- Beskonačna petlja u PyKnow-u se prekida pritiskom na tastere CTRL+M ili *Runtime/Interupt execution.*

Ključna riječ `in`

- Koriste se kada je potrebno testirati da li postoji neka vrijednost u Dictionary-ju, listi ili stringu
- Primjer 4. Odštampati broj socijalnog osiguranja svih osoba sa određenim prezimenom.

```
from pyknow import *

class Osoba(Fact):
    pass

class Broj_za_osobu(Fact):
    pass
```

Primjer 4...

```
class Broj_za_osobu(Fact):
    pass

class Nadji_broj_socijalnog(KnowledgeEngine):
    @DefFacts()
    def Lista_osoba(self):
        yield(Osoba(ime_i_prezime = "Marko M. Markovic",
                    broj_socijalnog = "123-56-5689"))
        yield(Osoba(ime_i_prezime = "Janko Markovic",
                    broj_socijalnog = "123-56-5690"))
        yield(Osoba(ime_i_prezime = "Nikola M. Nikolic",
                    broj_socijalnog = "133-16-5689"))
        yield(Broj_za_osobu("Markovic"))
    @Rule(Osoba(ime_i_prezime = MATCH.ime1,broj_socijalnog = MATCH.broj),
          Broj_za_osobu(MATCH.ime),
          TEST(lambda ime, ime1 : ime in ime1))
    def Stampa_broj_so(self,broj,ime1):
        print("Br socijalnog osiguranja za %s je %s\n" %(ime1, broj))
```

```
engine = Nadji_broj_socijalnog()
engine.reset()
engine.run()
```

```
Br socijalnog osiguranja za Janko Markovic je 123-56-5690
Br socijalnog osiguranja za Marko M. Markovic je 123-56-5689
```

Planiranje – ekspertni sistem

- Svijet blokova – postoje dva steka blokova. U prvom je blok A na vrhu steka, ispod njega je blok B, dok je na podu blok C. U drugom steku je blok D na vrhu, ispod njega je blok E, a na dnu je blok F.
- Definiše se jedan cilj – pomjeriti blok blok1 na vrh od bloka blok2.
- Na početku je najbolje pisati pseudokod. Razmisli se na koje se sve situacije može naići i kako ih riješiti.

1. Oba bloka blok1 i blok2 su na vrhu svojih stekova, te se blok1 može direktno prenijeti na blok2.

Pravilo pomjeri-direktno

AKO cilj je pomjeriti blok1 na blok2 I

blok1 je na vrhu svog steka I

blok2 je na vrhu svog steka

ONDA pomjeri direktno blok1 na blok2

Planiranje – ekspertni sistem

2. Ukoliko blok1 nije na vrhu steka, potrebno je sve blokove koji su iznad njega pomjeriti na pod.

Pravilo cisti-gornji-blok

AKO cilj je pomjeriti blok1

blok1 nije na vrhu svog steka

gornji je iznad blok1

ONDA pomjeri gornji na pod

3. Ukoliko blok2 nije na vrhu steka, potrebno je sve blokove koji su iznad njega pomjeriti na pod.

Pravilo cisti-donji-blok

AKO cilj je pomjeriti neki blok na blok2

blok2 nije na vrhu svog steka

gornji je iznad blok2

ONDA pomjeri gornji na pod

Planiranje – ekspertni sistem

4. Pomjeriti blok na pod.

Pravilo pomjeri-na-pod

AKO cilj je pomjeriti blok na pod

blok je na vrhu svog steka

ONDA pomjeri blok na pod

- Neka je cilj pomjeriti blok C na blok E.
- Vrijednost ključa gornji (gornji = “nista“) činjenice *na_vrhu_bloka* je indikator da je blok na vrhu steka.
- Aktivira se pravilo cisti-gornji-blok. Detektuje da je blok B iznad bloka C i traži da se pomjeri na pod. Detektuje se blok A iznad bloka B i traži se njegovo pomjeranje na pod. Ovaj blok je na vrhu steka, pomjera se na pod, sada je B na vrhu steka, pomjera se na pod i blok C ostaje na vrhu steka.
- Pravilima cisti-donji-blok i pomjeri-na-pod se blok E čini blokom na vrhu steka, te se aktivira pravilo pomjeri-direktno za pomjeranje bloka C na blok E.

Planiranje – ekspertni sistem

- Potrebne su nam činjenice:
- Na_vrhu_bloka, da za svaki blok pamti šta je ispod i iznad njega
- Cilj koja određuje šta se pomjera i na koji blok se pomjera.
- Blok – pošto ćemo imati za vrijednosti atributa konstante „pod“ i „nista“. Ovom činjenicom se utvrđuje da je vezana vrijednost atributa blok, a ne konstanta.

```
from pyknow import *

class Na_vrhu_bloka(Fact):
    pass

class Blok(Fact):
    pass

class Cilj(Fact):
    pass
```

Planiranje – početna BZ

```
class Promjeri_blokove(KnowledgeEngine):
    @DefFacts()
    def Pocetno_stanje(self):
        yield(Blok("A"))
        yield(Blok("B"))
        yield(Blok("C"))
        yield(Blok("D"))
        yield(Blok("E"))
        yield(Blok("F"))
        yield(Na_vrhu_bloka(gornji = "nista", donji = "A"))
        yield(Na_vrhu_bloka(gornji = "A", donji = "B"))
        yield(Na_vrhu_bloka(gornji = "B", donji = "C"))
        yield(Na_vrhu_bloka(gornji = "C", donji = "pod"))
        yield(Na_vrhu_bloka(gornji = "nista", donji = "D"))
        yield(Na_vrhu_bloka(gornji = "D", donji = "E"))
        yield(Na_vrhu_bloka(gornji = "E", donji = "F"))
        yield(Na_vrhu_bloka(gornji = "F", donji = "pod"))
        yield(Cilj(pomjeri = "C",na_vrh = "E"))
```

```
#"""Pravilo pomjeri-direktno
 #AKO cilj je pomjeriti blok1 na blok2 I
 #blok1 je na vrhu svog steka I
 #blok2 je na vrhu svog steka
 #ONDA pomjeri direktno blok1 na blok2
 #"""
 @Rule(AS.f1 << Cilj(pomjeri = MATCH.blok1,na_vrh = MATCH.blok2),
       Blok(MATCH.blok1),
       Blok(MATCH.blok2),
       #blok koji se pomjera je blok
       Na_vrhu_bloka(gornji = "nista",donji = MATCH.blok1),
       #nema nista iznad bloka koji se pomjera
       AS.f2 << Na_vrhu_bloka(gornji = "nista",donji = MATCH.blok2),
       #nema nista na bloku blok2 na koji se premjesta blok1
       #uzima mu se adresa jer ce nakon pomjeranja na njemu biti blok1
       AS.f3 << Na_vrhu_bloka(gornji = MATCH.blok1)
       #uzima se adresa bloka koji se nalazi ispod blok1,
       #da bi se azurirao njegov vrh
      )
 def PomjeriDirektno(self,f1,f2,f3,blok1,blok2):
   self.modify(f3,gornji = "nista")
   #samo se modifikuje atribut gornji, jer je pomjeren blok sa vrha
   self.modify(f2,gornji = blok1)
   self.retract(f1)
   #izvrsena je trazena akcija
   print("%s pomjeren na %s" %(blok1,blok2))
```

Planiranje – cisti-gornji-blok

```
#Pravilo cisti-gornji-blok
#AKO cilj je pomjeriti blok1
#blok1 nije na vrhu svog steka
#gornji je iznad blok1
#ONDA pomjeri gornji na pod
@Rule(Cilj(pomjeri = MATCH.blok1),
      #ovaj se cilj ne brise, jer nakon izvrsavanja ovog
      #pravila on nije
      #izvrsen, ovo pravilo samo pomjera blokove sa blok1
      #koji se treba pomjeriti ranije
      Blok(MATCH.blok1),
      Na_vrhu_bloka(gornji = MATCH.gornji,donji = MATCH.blok1),
       TEST(lambda gornji : gornji != "nista"))
def CistiGornjiBlok(self,gornji):
  self.declare(Cilj(pomjeri = gornji, na_vrh = "pod"))
```

Planiranje – cisti-donji-blok

```
#Pravilo cisti-donji-blok
#AKO cilj je pomjeriti neki blok na blok2
#blok2 nije na vrhu svog steka
#gornji je iznad blok2
#ONDA pomjeri gornji na pod
@Rule(Cilj(na_vrh = MATCH.blok2),
      #ovaj se cilj ne brise, jer nakon izvrsavanja ovog
      #pravila on nije
      #izvrsen, ovo pravilo samo pomjera blokove sa blok2
      #koji se treba pomjeriti ranije
      Blok(MATCH.blok2),
      Na_vrhu_bloka(gornji = MATCH.gornji,donji = MATCH.blok2),
       TEST(lambda gornji : gornji != "nista"))
def CistiDonjiBlok(self,gornji):
  self.declare(Cilj(pomjeri = gornji, na_vrh = "pod"))
```

Planiranje – pomjeri-na-pod

```
engine = Promjeri_blokove()
engine.reset()
engine.run()
```

```
D pomjeren na pod
A pomjeren na pod
B pomjeren na pod
C pomjeren na E
```

```
engine.facts
```

```
FactList([(0, InitialFact()),
          (1, Blok('A')),
          (2, Blok('B')),
          (3, Blok('C')),
          (4, Blok('D')),
          (5, Blok('E')),
          (6, Blok('F')),
          (7, Na_vrhu_bloka(gornji='nista', donji='A')),
          (11, Na_vrhu_bloka(gornji='nista', donji='D')),
          (13, Na_vrhu_bloka(gornji='E', donji='F')),
          (14, Na_vrhu_bloka(gornji='F', donji='pod')),
          (18, Na_vrhu_bloka(gornji='D', donji='pod')),
          (21, Na_vrhu_bloka(gornji='nista', donji='B')),
          (22, Na_vrhu_bloka(gornji='A', donji='pod')),
          (23, Na_vrhu_bloka(gornji='nista', donji='C')),
          (24, Na_vrhu_bloka(gornji='B', donji='pod')),
          (25, Na_vrhu_bloka(gornji='nista', donji='pod')),
          (26, Na_vrhu_bloka(gornji='C', donji='E'))])
```

Izvršavanje

```
#Pravilo pomjeri-na-pod
#AKO cilj je pomjeriti blok na pod
#blok je na vrhu svog steka
#ONDA pomjeri blok na pod
@Rule(AS.f1 << Cilj(pomjeri = MATCH.blok,na_vrh = "pod"),
      #nakon izvrsenja pravila, ova cinjenica ne vazi
      Blok(MATCH.blok),
      Na_vrhu_bloka(gornji = "nista",donji = MATCH.blok),
      AS.f2 << Na_vrhu_bloka(gornji = MATCH.blok))
     #uzmi adresu bloka koji se nalazi iznad bloka
     # koji ces pomjeriti na pod jer ce se osloboditi
def PomjeriNaPod(self,f1,f2,blok):
  self.modify(f2,gornji = "nista")
  self.declare(Na_vrhu_bloka(gornji = blok, donji = "pod"))
  #bitno je unijeti cinjenicu koja je nastala prebacivanjem
  #bloka na pod
  self.retract(f1)
  print("%s pomjeren na pod" %blok)
```

Implementacija steka...

- Prva dodata vrijednost se zadnja uzima - povlači sa steka.

```
from pyknow import *
class Stek(Fact):
    pass
class Cilj(Fact):
    pass
class Izbaci(Fact):
    pass
class BlokoviElegantnije(KnowledgeEngine):
    @DefFacts()
    def PocetnoStanje(self):
        yield(Stek(["A","B","C"]))
        yield(DodataVrijednost("nova"))
    @Rule(AS.f1 << Stek(),
          AS.f2 << DodataVrijednost(MATCH.dodaj))
    def DodajNaStek(self,dodaj,f1,f2):
        self.declare(Stek([dodaj] + unfreeze(f1[0])))
        self.retract(f1)
        self.retract(f2)
        print("Na stek je dodato %s" %dodaj)
```

Implementacija steka...

```
@Rule(AS.f1 << Stek(W()), AS.f2 << Izbaci(),
      TEST(lambda f1 : len(f1[0]) != 0))
def IzbaciSaSteka(self,f1,f2):
    #modify(f1,ostatak)
    print("sa steka je izbaceno %s" %f1[0][0])
    self.declare(Stek(list((f1[0][1:]))))
    self.retract(f1)
    self.retract(f2)
@Rule(AS.f1 << Stek(W()), AS.f2 << Izbaci(),
      TEST(lambda f1 : len(f1[0]) == 0))
def PrazanStek(self,f1):
    print("Stek je prazan\n")
    self.retract(f1)
```

Implementacija steka...

```
engine = BlokoviElegantnije()
engine.reset()
```

```
INFO:pyknow.watchers.FACTS: ==> <f-0>: InitialFact()
INFO:pyknow.watchers.FACTS: ==> <f-1>: Stek(frozenlist(['A', 'B', 'C']))
INFO:pyknow.watchers.FACTS: ==> <f-2>: DodataVrijednost('nova')
INFO:pyknow.watchers.ACTIVATIONS: ==> 'DodajNaStek': <f-2>, <f-1>
```

```
engine.run()
```

```
INFO:pyknow.watchers.FACTS: ==> <f-3>: Stek(frozenlist(['nova', 'A', 'B', 'C']))
INFO:pyknow.watchers.FACTS: <== <f-1>: Stek(frozenlist(['A', 'B', 'C']))
INFO:pyknow.watchers.FACTS: <== <f-2>: DodataVrijednost('nova')
INFO:pyknow.watchers.ACTIVATIONS: <== 'DodajNaStek': <f-2>, <f-1> [EXECUTED]
```

```
Na stek je dodato nova
```

Izvršavanje programa - Stek

```
engine.declare(Izbaci())
```

```
INFO:pyknow.watchers.FACTS: ==> <f-4>: Izbaci()
INFO:pyknow.watchers.ACTIVATIONS: ==> 'IzbaciSaSteka': <f-4>, <f-3>
```

```
Izbaci()
```

```
engine.run()
```

```
INFO:pyknow.watchers.FACTS: ==> <f-5>: Stek(frozenlist(['A', 'B', 'C']))
INFO:pyknow.watchers.FACTS: <== <f-3>: Stek(frozenlist(['nova', 'A', 'B', 'C']))
INFO:pyknow.watchers.FACTS: <== <f-4>: Izbaci()
INFO:pyknow.watchers.ACTIVATIONS: <== 'IzbaciSaSteka': <f-4>, <f-3> [EXECUTED]
```

```
sa steka je izbaceno nova
```

```
: unwatch('ACTIVATIONS',"FACTS")
```

```
engine.declare(Izbaci())
```

```
Izbaci()
```

```
engine.run()
```

```
sa steka je izbaceno A
```

```
engine.declare(Izbaci())
```

```
Izbaci()
```

```
engine.run()
```

```
sa steka je izbaceno B
```

```
engine.declare(Izbaci())
```

```
Izbaci()
```

```
engine.run()
```

```
sa steka je izbaceno C
```

```
: engine.declare(Izbaci())
```

```
: Izbaci()
```

```
: engine.run()
```

```
Stek je prazan
```