Umjetna inteligencija – Laboratorijska vježba 2

UNIZG FER, ak. god. 2019/20.

Zadano: 8.4.2020. Rok predaje: 26.4.2020. do 23.59 sati.

Kuharica opovrgavanjem (24 boda)

U drugoj laboratorijskoj vježbi bavit ćemo se automatskim zaključivanjem pomoću rezolucijskog pravila. Problem kojim ćemo se baviti kroz ovu laboratorijsku vježbu bit će implementacija sustava temeljenog na postupku zaključivanja rezolucijom opovrgavanjem koji će vam asistirati pri kuhanju.

Napomena: uskprkos tome što kroz upute prolazimo kroz jedan primjer, vaša implementacija **mora** funkcionirati na **svim** datotekama priloženim uz laboratorijsku vježbu u razumnom vremenu (max. 2 minute). Osim priloženih datoteka, vaša implementacija će se pomoću *autogradera* također testirati na primjerima koji nisu objavljeni, stoga pripazite na rubne slučajeve u kojima bi vaša implementacija mogla pogriješiti. Uz laboratorijsku vježbu dobiti ćete uzorak testnih ulaza i izlaza da provjeru možete probati pokrenuti lokalno. Molimo vas da detaljno pročitate upute za formatiranje vaših ulaza i izlaza u poglavlju "Upute za autograder", kao i primjere ispisa u poglavlju "Primjeri ispisa", budući da se naknadne greške neće tolerirati.

1. Učitavanje podataka

Kako bi algoritam zaključivanja te kuharskog asistenta mogli primijeniti na više različitih zadataka, definirat ćemo format ulaznih podataka za algoritam zaključivanja. Za sve tipove problema ulazni podaci bit će zapisani u dvije tekstne datoteke: (1) popis klauzula te (2) popis korisničkih naredbi. Svaka od tekstnih datoteka može sadržavati linije s komentarima. Takve linije uvijek počinju sa simbolom `#` i vaša implementacija ih treba ignorirati.

Popis klauzula definira klauzule za naš algoritam zaključivanja. U svakom retku datoteke nalazi se po jedna klauzula u konjunktivnoj normalnoj formi (CNF) formatu. Disjunkcija je označena simbolom `v` koji sa svake strane ima po jedan razmak. Literali se sastoje od proizvoljno dugačkog niza slova, brojeva i znaka `_`, pri čemu se ne razlikuje između velikih i malih slova (dakle, vrijedi `a_1 == A_1)`. Formalno, sve ulazne podatke možete pretvoriti u mala slova. Negacija je označena simbolom ~ i uvijek direktno prethodi literalu koji negira.

Primjer jednog retka iz datoteke popisa klauzula:

`~a v b`

Ovaj redak sastoji se od disjunkcije (`v`) literala `a` i `b`, pri čemu je `a` negiran (~).

Popis korisničkih naredbi sadrži ulazne podatke za drugi dio laboratorijske vježbe. Svaki redak korisničkih naredbi sadrži jednu klauzulu te identifikator namjere korisnika. Zadnja dva simbola svakog retka uvijek će biti razmak i jedan od idućih znakova: `?`, `+` ili `-`. Konkretno značenje identifikatora namjere je detaljnije pojašnjeno u poglavlju "3. Kuharski asistent (12 bodova)".

Primjer jednog retka iz datoteke popisa korisničkih naredbi:
`∼a v b ?`

2. Rezulucija opovrgavanjem (12 bodova)

Jednom kad smo implementirali učitavanje podataka, naš idući zadatak jest implementirati algoritam rezolucije opovragavanjem. Pri rezoluciji opovrgavanjem, **zadnju** klauzulu iz datoteke popisa klauzula smatramo ciljnom klauzulom (onom koju pokušavamo dokazati).

U vašoj implementaciji rezolucije opovrgavanjem trebate koristiti (1) **upravljačku strategiju skupa potpore** i (2) **strategiju brisanja**, koja uključuje uklanjanje redundantnih klauzula i uklanjanje nevažnih klauzula.

Također, vaša implementacija rezolucije opovrgavanjem treba za zadanu ciljnu klauzulu ispisivati je li uspješno dokazana te, ako je uspješno dokazana, potrebno je ispisati i **rezolucijski postupak** koji je vodio do `NIL`. Primjer takvog ispisa za klauzule iz datoteke `small_example.txt`:

```
1. a
2. b v ∼a
3. ∼b v c
=============
4. ∼c
=============
5. ∼b (3, 4)
6. ∼a (2, 5)
7. NIL (1, 6)
=============
c is true
```

Uz svaku klauzulu koja je izvedena u rezolucijskom postupku potrebno je ispisati njen redni broj te redne brojeve roditeljskih klauzula (u primjeru naznačeno brojevima u zagradama). Ovi ispisi neće se provjeravati kroz autograder, no provjeravat će se na usmenom ispitivanju. U ispisu vaše implementacije, najprije moraju biti ispisane klauzule koje su zadane kao premise, potom klauzula (ili klauzule) negiranog cilja, a potom slijede sve izvedene klauzule.

3. Kuharski asistent (12 bodova)

Jednom kad smo uspješno implementirali rezoluciju opovrgavanjem, korak dalje je iskoristiti ju kako bismo si pripomogli u procesu odabira recepta na temelju trenutačno dostupnih sastojaka. Ako redovito kuhate, sigurno vam je poznata situacija u kojoj jednostavno niste sigurni što biste jeli, što vas dovodi do mukotrpnog procesa odlučivanja. Također, ako ste zaboravni, ovaj proces postaje daleko kompliciraniji, budući da iako se odlučite za neki recept, trebate provjeriti imate li sve namirnice. Zbog svega toga, implementirat ćemo sustav koji će kroz literale pamtiti koje namirnice imamo te sadržavati popis recepata koje često koristimo.

Naš sustav treba moći učitati kuharicu (popis klauzula) iz tekstne datoteke koji sačinjava početnu bazu znanja. U kuharici se nalazi popis namirnica i recepata zapisanih u klauzalnom obliku. Kako bi bio koristiv više od jednom, naš sustav mora podržavati: (1) upite, (2) dodavanje klauzula i (3) brisanje klauzula.

Sustav mora podržavati dva načina rada: (1) interaktivni i (2) testni. U **interaktivnom** načinu rada, sustav kao ulaz prima samo popis klauzula, dok se korisnički upiti, dodavanje i brisanje klauzula izvršavaju putem konzole. U **testnom** načinu rada, sustav kao ulaz prima putanju do datoteke s popisom klauzula i putanju do datoteke s popisom korisničkih naredbi, iz koje se naredbe trebaju čitati i izvršavati slijedno. Pri testnom načinu rada, treba se ispisivati **samo** odgovor na korisničke upite (znak `?`). Testni način rada koristit će se pri provjeru rješenja autograderom.

Detaljniji opis funkcionalnosti koje naš sustav **mora** podržavati:

1. Provjeru valjanosti zadane ciljne klauzule (upit)
   - Korisnik zadaje klauzulu te znakom `?` označava da se radi o upitu
     `>>> ∼c v a ?`
2. Dodavanje znanja (klauzula) u bazu znanja
   - Korisnik znakom `+` označava da želi dodati klauzulu u bazu znanja
     `>>> a +`
3. Brisanje znanja (klauzula) iz baze znanja
   - Korisnik znakom `-` označava da želi izbrisati klauzulu iz baze znanja, ako je u njoj prisutna
     `>>> a -`
4. Naredbu za prekid rada: `exit`

Naredbe koje ekspertni sustav mora podržati sastoje od ključne riječi `exit` ili klauzule koju prate jedan znak razmaka i identifikator naredbe (`?, +, -`). Naredba `exit` nikada se neće naći u testnim datotekama za autograder, već samo na ručnom pokretanju. Naredbe dodavanja i brisanja (`+`, `-`) **ne trebaju** ništa ispisivati u testnom načinu rada.

Zadnji znak svake naredbe koju korisnik zadaje (osim `exit`) uvijek će biti identifikator naredbe.

Jednostavan primjer rada našeg sustava:

```
Resolution system constructed with knowledge:
> a
> ∼b v c
> b v ∼a

Please enter your query
>>> c ?

1. a
2. b v ∼a
3. ∼b v c
=============
4. ∼c
=============
5. ∼b (3, 4)
6. ∼a (2, 5)
7. NIL (1, 6)
```

```
=============
c is true

Please enter your query
>>> ∼b v c -
removed ∼b v c

Please enter your query
>>> c ?
c is unknown
```

Zadaci za dodatne bodove: pametna rezolucija opovrgavanjem i automatska konverzija u CNF (+6 bodova)

Napomena: Rješenja svih dodatnih zadataka moraju se predati na Ferko u istoj arhivi kao i rješenje ostatka laboratorijske vježbe.

1. Pametna rezolucija opovrgavanjem (+2 boda)

Kroz korištenje našeg ekspertnog sustava primijetili smo da je moguće, a čak i često, da će sustav dati negativan odgovor na naše pitanje, makar nam fali samo vrijednost jednog nepoznatog literala. No, korisniku ne dajemo nikakvu informaciju o tome zašto je rezolucijski postupak dao negativan odgovor na njegov upit. Sa strane korisničkog iskustva, ovo je vrlo problematično.

Jedan mogući način na koji možemo poboljšati naš ekspertni sustav jest da mu omogućimo da i on postavlja pitanja korisniku ako misli da bi mu odgovor na ta pitanja mogao pomoći u pronalasku rješenja. Naravno, ta pitanja ne smiju biti preteška, tako da ćemo se kod pitanja ograničiti na vrijednosti literala. U slučaju da nismo uspjeli dokazati da je cilj logička posljedica premisa, naš novi algoritam rezolucije opovrgavanjem treba pronaći popis literala čija bi vrijednost, kad bismo ju imali u bazi znanja, možda pomogla dokazati originalni cilj.

S novim algoritmom rezolucije opovrgavanja možemo poboljšati naš ekspertni sustav. Ako rezolucija opovrgavanjem nije uspjela dokazati zadani cilj, trebamo pitati korisnika za vrijednost nekih literala. Korisnik može odgovoriti da je literal istinit, neistinit ili da ne zna odgovor (`T`, `F`, `?`). Ako smo od korisnika dobili korisnu informaciju, trebamo ponovno pokrenuti algoritam rezolucije opovrgavanja i pokušati dokazati cilj s novim znanjem.

Razmislite na koji bi, što jednostavniji način mogli doći do znanja koje vam treba od korisnika.

Jednostavan primjer rada našeg poboljšanog sustava na datoteci `cooking_examples/coffee.txt`:

```
>>> Please enter your query
>>> heater -
removed heater

>>> Please enter your query
>>> coffee ?
coffee is unknown
candidate questions: [hot_water, heater]
```

```
>>> hot_water?
>>> [Y/N/?]
>>> N

>>> heater?
>>> [Y/N/?]
>>> Y
coffee is true
```

U testnom način rada, radi jednostavnosti, kuharski asistent treba ispisati **samo** literale koji su kandidati za upite.

2. Automatska konverzija u CNF (+4 boda)

U dosadašnjem dijelu laboratorijske vježbe koristili smo jedno bitno pojednostavljenje, a to je da su nam svi logički izrazi zadani u konjunktivnoj normalnoj formi CNF. U sklopu ovog bonus zadatka potrebno je implementirati automatsku konverziju u CNF.

Ulazni logički izrazi biti će zadani putanjom (preko argumenta gdje bi inače bio popis klauzula). Implikacija će biti označena s `>`, ekvivalencija s `=` a logički I sa `&`. Logički ILI će i dalje biti označen s `v`. Svaki operator biti će odvojen razmacima sa svake strane. Elementi na koje se primjenjuje svaki logički operator (podsjetnik: logički operatori su **binarni**) bit će biti odvojeni zagradama od ostatka logičkog izraza. Te zagrade će vam pritom i eksplicitno označavati prioritet. Smjer implikacije će uvijek biti slijeva nadesno (znak `<` neće se pojavljivati u testnim datotekama).

Primjer jednog ovakvog logičkog izraza:

$(C \vee D) \Rightarrow (\neg A \Leftrightarrow B)$

Njegov zapis u datoteci:

```
(C v D) > (~A = B)
```

CNF format koji se očekuje dobiti automatskom konverzijom zadanog izraza:

```
((A v B) v ~C) & ((A v B) v ~D) & ((~A v ~B) v ~C) & ((~A v ~B) v ~D)
```

Za svaki redak s logičkim izrazom u ulaznoj datoteci, vaš program mora ispisati redak u kojemu se nalazi logička formula u CNF. Redoslijed klauzula u ispisu **nije** bitan (bitno je samo da je izraz logički ekvivalentan s rješenjem). Dodatni primjeri ulaza i izlaza za konverziju CNF mogu se naći u `autocnf/cnfconvert_tests.txt` i `autocnf/cnfconvert_expected_outputs.txt`.

Upute za autograder

Struktura uploadane arhive

Arhiva koju uploadate na Ferko **mora** biti naziva `JMBAG.zip`, dok struktura raspakirane arhive **mora** izgledati kao u nastavku (primjer u nastavku je za Python, a primjeri za ostale jezike slijede u zasebnim poglavljima):

```
|JMBAG.zip
|-- lab2py
```

```
|----solution.py [!]
|----resolution.py (optional)
|----...
```

Arhive koje nisu predane u navedenom formatu se **neće priznavati**. Vaš kod se mora moći pokrenuti tako da prima iduće argumente putem komandne linije:

1. Zastavica podzadatka: string
   Jedno od: `[resolution, cooking_test, cooking_interactive, smart_resolution_test, smart_resolution_interactive, autocnf]`
   Prva zastavica služi za provjeru implementacije prvog podzadatka, druge dvije služe za provjeru implementacije drugog podzadatka, dok preostale služe za provjeru bonus zadataka.

2. Putanja do datoteke s popisom klauzula

3. Putanja do datoteke korisničkih naredbi (opcionalno)

4. Zastavica `verbose` (opcionalno)
   Ova zastavica je po defaultu isključena a služi za ispisivanje opširnijih ipsisa koje možda imate (npr. redoslijeda u rezoluciji opovrgavanjem). Ti ispisi se koriste samo za vrijeme usmenog ispitivanja.

Primjetite da su zadnja dva argumenta opcionalna, tako da je **moguće** da se zastavica `verbose` pojavi na trećem mjestu.

Vaš kod će se pokretati na linux-u. Ovo nema poseban utjecaj na vašu konkretnu implementaciju osim ako ne hardkodirate putanje do datoteka (što **ne bi smjeli**). Vaš kod ne smije koristiti **nikakve dodatne vanjske biblioteke**. Koristite encoding `UTF-8` za vaše datoteke s izvornim kodom.

Primjer pokretanja vašeg koda pomoću autogradera (u nastavku za Python):

```
>>> python solution.py resolution resolution_examples/small_example.txt
> output.txt
```

Upute: Python

Ulazna točka vašeg koda **mora** biti u datoteci `solution.py`. Kod možete proizvoljno strukturirati po ostalim datotekama u direktoriju, ili sav kod ostaviti u `solution.py`. Vaš kod će se uvijek pokretati iz direktorija vježbe (`lab2py`).

Struktura direktorija i primjer naredbe mogu se vidjeti na kraju prethodnog poglavlja. Verzija Pythona na kojoj će se vaš kod pokretati biti će `Python 3.7.4`.

Upute: Java

Uz objavu laboratorijske vježbe objaviti ćemo i predložak Java projekta koji možete importati u vaš IDE. Struktura unutar arhive `JMBAG.zip` definirana je u predlošku i izgleda ovako:

```
|JMBAG.zip
|--lab2java
|----src
|------main.java.ui
```

```
|--------Solution.java [!]
|--------Resolution.java (optional)
|--------...
|----target
|----pom.xml
```

Ulazna točka vašeg koda **mora** biti u datoteci `Solution.java`. Kod možete proizvoljno strukturirati po ostalim datotekama unutar direktorija, ili sav kod ostaviti u `Solution.java`. Vaš kod će se kompajlirati pomoću Mavena.

Primjer pokretanja vašeg koda pomoću autogradera (iz direktorija `lab2java`):

```
>>> mvn compile
>>> java -cp target/classes ui.Solution
```

Informacije vezano za verzije Mavena i Jave:

```
>>> mvn -version
Apache Maven 3.6.1
Maven home: /usr/share/maven
Java version: 13.0.2, vendor: Oracle Corporation, runtime: /opt/jdk-13.0.2
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "5.3.0-45-generic", arch: "amd64", family: "unix"

>>> java -version
java version "13.0.2" 2020-01-14
Java(TM) SE Runtime Environment (build 13.0.2+8)
Java HotSpot(TM) 64-Bit Server VM (build 13.0.2+8, mixed mode, sharing)
```

Upute: C++

Struktura unutar arhive `JMBAG.zip` mora izgledati ovako:

```
|JMBAG.zip
|--lab2cpp
|----solution.cpp [!]
|----resolution.cpp (optional)
|----resolution.h (optional)
|----Makefile (optional)
|----...
```

Ako u arhivi koju predate ne postoji `Makefile`, za kompajliranje vašeg koda iskoristiti će se `Makefile` koji je dostupan uz vježbu. **Ako** predate `Makefile` u arhivi (ne preporučamo, osim ako stvarno ne znate što radite), očekujemo da on funkcionira.

Primjer pokretanja vašeg koda pomoću autogradera (iz direktorija `lab2cpp`):

```
>>> make
>>> ./solution resolution resolution_examples/small_example.txt >
    output.txt
```

Informacije vezano za gcc:

```
>>> gcc --version
```

```
gcc (Ubuntu 9.2.1-9ubuntu2) 9.2.1 20191008
Copyright (C) 2019 Free Software Foundation, Inc.
```

Primjeri ispisa

Uz primjere ispisa biti će priložena i naredba kojom je kod pokrenut. Naredba pokretanja pretpostavlja da je izvorni kod u Pythonu, no argumenti će biti isti za ostale jezike.

1. Rezolucija opovrgavanjem

Za rezoluciju opovrgavanjem smo pripremili više jednostavnih primjera u datotekama na kojima će se vrtiti testovi. Ispis ćemo prikazati samo za tri primjera, dok ostale ulazne datoteke možete vidjeti sami u direktoriju `resolution_examples`.

Jednostavni primjer s točnim rezultatom

```
>>> python solution.py resolution resolution_examples/small_example.txt
    verbose


1. a
2. b v ∼a
3. ∼b v c
=============
4. ∼c
=============
5. ∼b (3, 4)
6. ∼a (2, 5)
7. NIL (1, 6)
=============
c is true


>>> python solution.py resolution resolution_examples/small_example.txt


c is true
```

Jednostavni primjer bez NIL (ispis je isti sa i bez verbose)

```
>>> python solution.py resolution resolution_examples/small_example_3.txt


c is unknown
```

Primjer s opisnim literalima: kuhanje kave

```
>>> python solution.py resolution resolution_examples/coffee.txt verbose


1. heater
2. coffee_powder
3. water
4. ∼water v hot_water v ∼heater
5. ∼coffee_powder v coffee v ∼hot_water
===============
```

```
6. ∼coffee
===============
7. ∼coffee_powder v ∼hot_water (5, 6)
8. ∼coffee_powder v ∼water v ∼heater (4, 7)
9. ∼coffee_powder v ∼heater (3, 8)
10. ∼heater (2, 9)
11. NIL (1, 10)
===============
coffee is true
```

2. Kuharski asistent

Primjer s opisnim literalima: kuhanje kave

```
>>> python solution.py cooking_interactive cooking_examples/coffee.txt
    verbose
```

Ispis nakon inicijalizacije **(nije nužan u vašoj implementaciji)**:

```
Testing cooking assistant with standard resolution
Constructed with knowledge:
> water
> heater
> coffee_powder
> ∼heater v ∼water v hot_water
> coffee v ∼coffee_powder v ∼hot_water
```

Rad kuharskog asistenta:

```
>>> Please enter your query
>>> hot_water ?
1. heater
2. water
3. ∼heater v ∼water v hot_water
===============
4. ∼hot_water
===============
5. ∼heater v ∼water (3, 4)
6. ∼heater (2, 5)
7. NIL (1, 6)
===============
hot_water is true

>>> Please enter your query
>>> coffee ?
1. heater
2. water
3. ∼heater v ∼water v hot_water
4. coffee_powder
5. coffee v ∼coffee_powder v ∼hot_water
```

```
===============
6. ∼coffee
===============
7. ∼coffee_powder v ∼hot_water (5, 6)
8. ∼hot_water (4, 7)
9. ∼heater v ∼water (3, 8)
10. ∼heater (2, 9)
11. NIL (1, 10)
===============
coffee is true

>>> Please enter your query
>>> exit
```

Primjer **testnog** načina rada:

```
>>> python solution.py cooking_test cooking_examples/coffee.txt
    cooking_examples/coffee_input.txt > cooking_examples/coffee_output.txt
```

3. Pametna rezolucija opovrgavanjem

Primjer s opisnim literalima: kuhanje kave

```
>>> python solution.py smart_resolution_interactive
    cooking_examples/coffee.txt verbose
```

Ispis nakon inicijalizacije **(nije nužan u vašoj implementaciji)**:

```
Testing cooking assistant with standard resolution
Constructed with knowledge:
> water
> heater
> coffee_powder
> ∼heater v ∼water v hot_water
> coffee v ∼coffee_powder v ∼hot_water
```

Rad kuharskog asistenta: `cooking_examples/coffee.txt`:

```
>>> Please enter your query
>>> heater -
removed heater

>>> Please enter your query
>>> coffee ?
coffee is unknown
candidate questions: [hot_water, heater]

>>> hot_water?
>>> [Y/N/?]
>>> N
```

```
>>> heater?
>>> [Y/N/?]
>>> Y
coffee is true
```

Primjer **testnog** načina rada:

```
>>> python solution.py smart_resolution_test cooking_examples/coffee.txt
    smart_resolution/coffee_input.txt > smart_resolution/coffee_output.txt
```

4. Automatska konverzija u CNF

Parovi ulaza i ispisa za niz primjera dostupni su u datotekama `autocnf/autocnf_sample_tests.txt` i `autocnf/autocnf_sample_output.txt`.

Primjer pokretanja provjere za autocnf koje treba generirati (logički) ekvivalentnu izlaznu datoteku:

```
>>> python solution.py autocnf autocnf/sample_tests.txt >
    autocnf/sample_output.txt
```