Uvod u umjetnu inteligenciju – Laboratorijska vježba 2

UNIZG FER, ak. god. 2024/25.

Zadano: 5.3.2025. Rok predaje: 10.4.2025. 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.

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).

Provjera koliko dobro vaša implementacija radi provodit će se pomoću *autogradera*. Očekuje se da znate pokrenuti svoje rješenje ispred ispitivača pri predaji laboratorijske vježbe pomoću *autogradera*. "Upute za autograder" pisane su uzevši u obzir da ćemo **mi** pokretati vaše rješenje što ove godine **nećemo** raditi, nego se **vi** morate pobrinuti da se vaše rješenje može pokrenuti i evaluirati *autograderom* ispred ispitivača. Dobit ćete sve testne primjere unaprijed.

Rok za predaju arhive s rješenjem na Moodle **ne uključuje** posljednju minutu u danu, pa se pobrinite da svoju arhivu uploadate **prije** 23:59. Predaja u točno 23:59 ili nakon toga smatrat će se kasnom predajom.

Detaljno pročitajte upute za formatiranje vaših ulaza i izlaza u poglavlju "Upute za autograder", kao i primjere ispisa u poglavlju "Primjeri ispisa". Predana laboratorijska vježba koja se ne može pokrenuti na autograderu će se bodovati s 0 bodova **bez iznimki**. Vaš kod ne smije koristiti **nikakve dodatne vanjske biblioteke**. Ako niste sigurni smijete li koristiti neku biblioteku, provjerite je li ona dio standardnog paketa biblioteka za taj jezik.

Ukupan broj bodova po podzadacima u ovoj laboratorijskoj vježbi iznosi 24 boda. Broj bodova koji ostvarite prilikom predaje vježbe bit će skaliran na 7.5 bodova.

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 zapisana u konjunktivnoj normalnoj formi (CNF). 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 (6 bodova)".

Primjer jednog retka iz datoteke popisa korisničkih naredbi:

```
~a v b ?
```

2. Rezulucija opovrgavanjem (18 bodova)

Jednom kad smo implementirali učitavanje podataka, naš idući zadatak jest implementirati algoritam rezolucije opovragavanjem. Pri pokretanju algoritma rezolucije opovrgavanjem, vašem će se rješenju predati dva argumenta. Prvi argument bit će ključna riječ "`resolution`", kojom se označava da se treba pokrenuti algoritam rezolucije opovrgavanjem, a drugi argument bit će putanja do datoteke s popisom klauzula.

Primjer poziva za rješenje u Pythonu:

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

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. ~a v b
3. c v ~b
4. ~c
===============
5. ~b (3, 4)
6. ~a (2, 5)
7. NIL (1, 6)
===============
[CONCLUSION]: 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). 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.

Pri evaluaciji vašeg rješenja autograderom, evaluirat će se **samo** zadnja linija ispisa (prefiksirana s "`[CONCLUSION]:`"), dok će se ispis iz prethodnih redaka provjeravati pri usmenom ispitivanju. Samim time, vaš ispis u redovima prije posljednjeg ne mora odgovarati formatu danom u primjerima u ovom dokumentu, ali vaš ispis u tim linijama mora biti dovoljno deskriptivan kako bi se mogao pratiti postupak zaključivanja. To znači da uz svaku izvedenu klauzulu morate ispisati koje od prethodnih klauzula su korištene za izvođenje te klauzule, što je u prethodnom primjeru dano s rednim brojevima klauzula u zagradama. Ako ne ispisujete redni broj klauzula, onda morate uz klauzulu ispisati klauzule pomoću literala koji ih čine. Također, vaš ispis treba sadržavati samo klauzule koje su dovele do `NIL`, što znači da neiskorištene, ali izvedene klauzule ne smijete ispisivati.

Ako vaš ispis ne sadrži samo klauzule korištene za izvođenje `NIL` i informacije o roditeljskim klauzulama za izvedene klauzule, smatrat će se da nije dovoljno deskriptivan, te ćete biti penalizirani prilikom odgovaranja.

Također, pobrinite se da je u zadnjem retku vašeg rješenja ispisana odgovarajuća poruka u traženom formatu. U prethodnom primjeru ciljna klauzula je uspješno dokazana, pa je u posljednjem retku kao zaključak ispisana poruka: "`[CONCLUSION]: c is true`". U slučajevima u kojima ciljna klauzula nije dokazana u posljednjem retku treba biti ispisana poruka: "`[CONCLUSION]: clause is unknown`", pri čemu `clause` treba zamijeniti zadanom ciljnom klauzulom.

3. Kuharski asistent (6 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.

Za pokretanje kuharskog asistenta, vašem će se rješenju predati tri argumenta. Prvi argument bit će ključna riječ "`cooking`", kojom se označava da se treba pokrenuti sustav kuharskog asstenta. Drugi argument bit će putanja do datoteke s popisom klauzula, a treći argument bit će putanja do datoteke s popisom korisničkih naredbi, iz koje se naredbe trebaju čitati i izvršavati slijedno. Primjer poziva za rješenje u Pythonu:

```
>>> python solution.py cooking cooking_examples/coffee.txt
    cooking_examples/coffee_input.txt
```

U ovom primjeru popis klauzula učitava se iz datoteke `cooking_examples/coffee.txt`, a popis korisničkih naredbi iz datoteke `cooking_examples/coffee_input.txt`.

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 -`

Naredbe koje ekspertni sustav mora podržati sastoje od klauzule koju prate jedan znak razmaka i identifikator naredbe (`?, +, -`). Pri ispisu, vaš sustav **uvijek** mora ispisati odgovore na korisničke upite (znak `?`). Naredbe dodavanja i brisanja (`+`, `-`) **ne trebaju** ništa ispisivati. Zadnji znak svake naredbe koju korisnik zadaje uvijek će biti identifikator naredbe.

Jednostavan primjer rada sustava kuharskog asistenta dan je u nastavku. Primjer predstavlja ispis za gore navedeni poziv (popis klauzula iz `cooking_examples/coffee.txt` i popis naredbi iz `cooking_examples/coffee_input.txt`).

```
Constructed with knowledge:
coffee_powder
~heater v ~water v hot_water
~hot_water v coffee v ~coffee_powder
water
heater

User's command: water ?
1. water
2. ~water
===============
3. NIL (1, 2)
===============
[CONCLUSION]: water is true

User's command: hot_water ?
1. water
2. heater
3. ~heater v ~water v hot_water
4. ~hot_water
===============
5. ~heater v ~water (3, 4)
6. ~water (2, 5)
7. NIL (1, 6)
===============
[CONCLUSION]: hot_water is true

User's command: coffee ?
```

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

User's command: heater -
removed heater

User's command: hot_water ?
[CONCLUSION]: hot_water is unknown

User's command: coffee ?
[CONCLUSION]: coffee is unknown

User's command: heater +
Added heater

User's command: coffee ?
1. coffee_powder
2. water
3. heater
4. ~heater v ~water v hot_water
5. ~hot_water v coffee v ~coffee_powder
6. ~coffee
===============
7. ~coffee_powder v ~hot_water (5, 6)
8. ~heater v ~water v ~coffee_powder (4, 7)
9. ~water v ~coffee_powder (3, 8)
10. ~coffee_powder (2, 9)
11. NIL (1, 10)
===============
[CONCLUSION]: coffee is true
```

Kao i u slučaju algoritma rezolucije opovrgavanjem, pri provjeri vašeg sustava kuharskog asistenta autograderom također će se provjeravati **samo** linije koje počinju prefiksom "`[CONCLUSION]:`", odnosno ispis vašeg sustava za odgovore na korisničke upite (dani s identifikatorom `?`). Format zaključka nakon prefiksa "`[CONCLUSION]:`" je zadan na isti način kao u zadatku rezolucije opovrgavanjem.

Redoslijed odgovora na korisničke upite mora odgovarati redoslijedu kojim su upiti dani

u datoteci s korisničkim upitima. Ostatak vašeg ispisa možete oblikovati kako želite, budući da se neće provjeravati autograderom. Međutim, kao i u slučaju algoritma rezolucije opovrgavanjem, nužno je da se u ispisu nalazi postupak zaključivanja koji će se provjeravati pri usmenom ispitivanju.

Upute za autograder

U nastavku slijede upute o strukturi arhive za upload, a detaljne upute kako pokrenuti *autograder* koristeći zadanu strukturu nalaze se u `autograder.zip` arhivi u datoteci `README.md`.

Struktura uploadane arhive

Arhiva koju uploadate na Moodle **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]`
   Prva zastavica služi za provjeru implementacije prvog podzadatka (rezolucija opovrgavanjem), dok druga služi za provjeru drugog podzadatka (kuharski asistent).
2. Putanja do datoteke s popisom klauzula
3. Putanja do datoteke korisničkih naredbi (samo ako je zastavica podzadatka `cooking`)

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
```

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 resolution
    resolution_examples/small_example.txt
```

Informacije vezano za verzije Mavena i Jave:

```
>>> mvn -version
Apache Maven 3.6.3
Maven home: /opt/maven
Java version: 15.0.2, vendor: Oracle Corporation, runtime: /opt/jdk-15.0.2
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.15.0-139-generic", arch: "amd64", family: "unix"

>>> java -version
openjdk version "15.0.2" 2021-01-19
OpenJDK Runtime Environment (build 15.0.2+7-27)
OpenJDK 64-Bit Server VM (build 15.0.2+7-27, 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
```

Informacije vezano za gcc:

```
>>> gcc --version
gcc (Ubuntu 9.3.0-11ubuntu0~18.04.1) 9.3.0
```

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

Primjeri ispisa dani su za tri primjera, dok ostale ulazne datoteke možete isprobati sami. Datoteke možete pronaći u direktoriju `resolution_examples`.

Jednostavan primjer s točnim rezultatom

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


1. a
2. ~a v b
3. c v ~b
4. ~c
===============
5. ~b (3, 4)
6. ~a (2, 5)
7. NIL (1, 6)
===============
[CONCLUSION]: c is true
```

Jednostavan primjer bez NIL

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


[CONCLUSION]: c is unknown
```

Primjer s opisnim literalima: kuhanje kave

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


1. coffee_powder
2. water
3. heater
4. ~water v ~heater v hot_water
```

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

2. Kuharski asistent

Primjer ispisa za pokretanje kuharskog asistenta dan je u poglavlju "3. Kuharski asistent". Zbog duljine ispisa, nećemo ga ovdje ponavljati.