CSCI 4430 Programming Languages
Fall 2023
Programming Assignment 3
Deadline: 7:00pm EST, November 30, 2023

This assignment is to be done either individually or in pairs. Do not show your code to any other group and do not look at any other groups code. Do not put your code in a public directory or otherwise make it public. However, you may get help from the mentors, TAs, or the instructor. You are encouraged to use the LMS Discussion Forum to post problems so that other students can also answer/see the answers.

This assignment will strengthen your understanding of logical and constraint programming through application in either Prolog or Oz.

Part 1: Relationship Reasoning (45%)

One classic usage of logic programming is to reason about the relationships given some facts. For part 1 of this assignment, you will be provided with a rule file in Prolog or Oz that encodes a chunk of characters in the novel *Harry Potter*. The relationship is constructed using facts in the form of one of the following predicates:

Provided Facts

- `teacherOf(Teacher, Student)` (This is a directional relationship, Teacher gives lectures to Student. But Student cannot give lectures to Teacher.)
- `directSeniorOf(Senior, Junior)` (Senior spends more time in Hogwarts than Junior. Notice that this relationship indicates the close senior relationship. For example, Albus Dumbledore is the direct senior of Minerva McGonagall and Minerva McGonagall is the direct senior of Dolores Umbridge. But Albus Dumbledore is not the direct senior of Dolores Umbridge. In addition, the age indicated by `birthYear` is not related to the time spent in Hogwarts, i.e. `directSeniorOf` relationship). Note that not all students or teachers have `directSeniorOf` relationship since the years for some characters spend in Hogwarts are unknown.
- `birthYear(Person, Year)` (For the provided characters, you can safely assume there will be no two characters born in the same year. That means the birth year might not match the novel. For example, Ron Weasley was born in 1980 which is in the same year as Harry Potter. We intentionally change it to 1981 to avoid two characters born in the same year.)
- `houseOf(House, Student)` (This fact indicates which house Student belongs to. There are four houses in total: Gryffindor, Hufflepuff, Ravenclaw and Slytherin.)
- `farLocation(HouseA, HouseB)` (This fact indicates whether two houses are geologically far away. For example, the house Gryffindor is far away from the house Ravenclaw. If two houses are far away, that indicates that the students from one house live far away from the students from another house).
- `quidditchTeamOf(Team, Student)` (This fact indicates which Quidditch team Student belongs to.)

Rules To Be Written:

1. `classmates(StudentOne, StudentTwo)`: `StudentOne` and `StudentTwo` are classmates if they have the same teacher. Note that `StudentOne` and `StudentTwo` must be distinct people (i.e. not the same person; You can't be your own classmate for the purposes of this assignment).
2. `liveFarAway(StudentOne, StudentTwo)`: Two students live far away from each other if they belong to two different houses and two houses are far away as indicated by the fact `farLocation(HouseA, HouseB)`.
3. `isSeniorOf(PersonA, PersonB)`: `PersonA` can be directly senior to `PersonB` as indicated by `directSeniorOf(PersonA, PersonB)`. One of other possibilities is that PersonA is directly senior to PersonC and PersonC is senior to PersonB. In other words, there can be multiple persons.

4. `listSeniors(Person, Seniors)`: Produce a list of all seniors of `Person`, i.e. all people who are more senior than Person.

5. `listJuniors(Person, Juniors)`: Produce a list of all juniors of `Person`, i.e. all people who are more junior than Person.

6. `oldestStudent(Person, House)`: Give the oldest student in House. You need to use `birthYear` to determine which student is oldest. Note that Hogwarts teachers are not considered students. You can figure out if one is considered as student using `houseof(House, Student)`.

7. `youngestStudent(Person, House)`: `Person` is the youngest student in House. Similarly, you need to use `birthYear`, and Hogwarts teachers are not considered students.

8. `oldestQuidditchStudent(Team, Student)`: `Student` is the oldest among all the students within the Quidditch `Team`.

9. `youngestQuidditchStudent(Team, Student)`:`Student` is the youngest among all the students within the Quidditch `Team`.

10. `rival(StudentOne, StudentTwo)`: Two students are rivals if they come from different houses.

11. `farRival(StudentOne, StudentTwo)`: Two students are far rivals if they come from two different houses and these two houses are far away as indicated by `farLocation`.

Example results of each rule

The grade for part one will be determined by your rules' ability to be correctly used in inference. A good starting point to verify that you have a correct implementation of these rules is by testing out the following queries. With the above rules working properly, your program should be able to show, with the provided knowledge base, that:

- Test 1: Harry Potter and Ron Weasley are classmates
- Test 2: Harry Potter and Draco Malfoy live far away
- Test 3: Albus Dumbledore is senior to Ginny Weasley
- Test 4: Seniors of James Potter are Minerva Mcgonagall and Albus Dumbledore
- Test 5: Juniors of Bill Weasley are Fred Weasley, Ron Weasley and Ginny Weasley
- Test 6: Oldest student in the House Gryffindor is Alicia Spinnet
- Test 7: Youngest student in the House Gryffindor is Ginny Weasley
- Test 8: Oldest student in the Hufflepuff team is David Smith
- Test 9: Youngest student in the Hufflepuff team is Eli Olivander
- Test 10: Harry Potter and Draco Malfoy are rivals
- Test 11: Harry Potter and Draco Malfoy are not only rivals but also live far away from each other

How to Run in Prolog:

Download starter code at `https://www.cs.rpi.edu/academics/courses/fall23/proglang/pa3/prolog-pa3-start.zip`.

Write your rule definitions in a file called `relations.pl`, replacing the hard-coded answers in the provided version of the file. At any time, you can use `part1Tester.pl` to see if your code for the rules is working. With the two above files in the same folder, along with the provided `hogwarts.pl`, run the following from the command line to perform a test:

```
swipl -s part1Tester.pl -t main --quiet > part1Result.txt
```

This should run the `main` function in `part1_tester.pl` and produce some output automatically. While this method is good in a pinch, for this part it is recommended to load the knowledge-base file and the rules file into the SWI-Prolog interpreter manually and play around with your rules. This can be done, for example, like:

```
?- [hogwarts, relations].
true.

?- listSeniors(harry_potter, Seniors).
```

After the last line above is entered, pressing ; on your keyboard several times should show all senior students for Harry Potter. You can (and should) run other tests as well, to make sure your rules function like their descriptions dictate.

How to Run in Oz:

Download starter code at `https://www.cs.rpi.edu/academics/courses/fall23/proglang/pa3/oz-pa3-start.zip`.

The relationship encoding and given rules are provided in `hogwarts.oz`. Write your rule definitions in the file `relations.oz`. The sample queries are provided for you in `part1Tester.oz`. This file can also be extended with your own custom tests. To test your relations, simply verify by compiling `relations.oz` and `part1Tester.oz` and running `part1Tester.ozf`

```
ozc -c hogwarts.oz # Only need to run this once
ozc -c relations.oz
ozc -c part1Tester.oz
ozengine part1Tester.ozf
```

Part 2: Hogwarts Fundraising Constraint Problem(45%)

Hogwarts has multiple living and learning communities for its students under the supervision of different houses. The Hogwarts Administration wants to redecorate the dormitories to make more room for new students while raising funds for Quidditch. To do this, they have to catalog each student's items and then, based on the desired space and funds, decide what to sell away and keep. The administrators visited the dormitories to list items by their estimated price and volume.

For this problem, you are to implement a natural language processing program that will read the list of the items with their price and volume. Each house has constraints on the price and volume of items to sell for the house. In addition to the item descriptions, you must also parse the constraint information based on the grammar provided below. After parsing the given information, your program should run to output all possible lists of items with their total price and their volume that satisfy the constraint computation by matching each person to their corresponding house.

For privacy reasons, the administrators only wrote down the items name as an alphabet. Hence, items will only be alphabetically between **[a, z]**. For convenience, they also just used integer values to estimate the volume and the price of each element. For convenience, the name of the students and houses are going to be the same as the ones in `hogwarts.[pl/oz]`.

Grammar

For item:

[Person] has item [Item] that costs [Value] dollars, and occupies [Value] cubic feet.

- **Person**: `houseOf(_, Person)`
- **Item**: {a, b, c, ..., x, y, z}
- **Value**: {1, 2, 3, ... }

For Constraint

[House] house wants [AttributeValue] and [AttributeValue].

- **House**: `houseOf(House, _)`
- **AttributeValue**: **[Attribute] [Comparison] [Value] [Unit]**
- **Comparison**: **less than** | **greater than**
- **Attribute**: **total price** | **total volume**
- **Unit** : **dollars** | **cubic feet**
- **Value**: {1, 2, 3, ... }

Objective

You would need to run using a main file, either **part2.pl** or **part2.oz** depending on your chosen language. As provided, both main files read in sentences from an input file `example1.txt`. You would need to:

Parsing Problem: Parse the Natural Language using the grammar mentioned above and store the key information in different variables

Constraint Problem: Solve the constraint problem by listing all of the items that satisfy the constraint for each houses

Output: Output the results in stdout as provided in the example

```
House
write('\t')[TotalPrice, TotalVolume]:[List of Possible Items]
...
```

Example

Notice: In both languages, all atoms are made lowercase in the tokenization step (In Prolog: **read_line**, In Oz: **Helper.tokenize**) before they reach any of processing you do with your NLP grammar, so do not worry about dealing with upper case letters that are present in the spec files.

`example1.txt`

```
susan_bones has item x that costs 400 dollars, and occupies 10 cubic feet.
eli_olivander has item y that costs 260 dollars, and occupies 15 cubic feet.
vince_glover has item z that costs 490 dollars, and occupies 35 cubic feet.
terry_boot has item u that costs 350 dollars, and occupies 50 cubic feet.
duncan_inglebee has item v that costs 125 dollars, and occupies 26 cubic feet.
cho_chang has item w that costs 235 dollars, and occupies 34 cubic feet.
draco_malfoy has item c that costs 600 dollars, and occupies 60 cubic feet.
terence_higgs has item c that costs 1200 dollars, and occupies 60 cubic feet.
daphne_greengrass has item d that costs 400 dollars, and occupies 15 cubic feet.
harry_potter has item a that costs 1000 dollars, and occupies 100 cubic feet.
ron_weasley has item d that costs 233 dollars, and occupies 13 cubic feet.
hermione_granger has item b that costs 200 dollars, and occupies 20 cubic feet.
hermione_granger has item c that costs 100 dollars, and occupies 30 cubic feet.
hermione_granger has item c that costs 300 dollars, and occupies 10 cubic feet.
gryffindor house wants total price greater than 500 dollars and total volume greater than 60 cubic feet.
slytherin house wants total volume less than 50 cubic feet and total price greater than 600 dollars.
ravenclaw house wants total volume greater than 80 cubic feet and total price greater than 700 dollars.
hufflepuff house wants total volume less than 20 cubic feet and total price greater than 250 dollars.
```

Results:

```
gryffindor
 [1833,173]:[a,d,b,c,c]
 [1533,163]:[a,d,b,c]
 [1733,143]:[a,d,b,c]
 [1433,133]:[a,d,b]
 [1633,153]:[a,d,c,c]
 [1333,143]:[a,d,c]
 [1533,123]:[a,d,c]
 [1233,113]:[a,d]
 [1600,160]:[a,b,c,c]
 [1300,150]:[a,b,c]
 [1500,130]:[a,b,c]
 [1200,120]:[a,b]
 [1400,140]:[a,c,c]
 [1100,130]:[a,c]
 [1300,110]:[a,c]
 [1000,100]:[a]
 [833,73]:[d,b,c,c]
 [533,63]:[d,b,c]
slytherin
ravenclaw
 [710,110]:[u,v,w]
hufflepuff
 [400,10]:[x]
 [260,15]:[y]
```

How to Run in Prolog

A file, `part2.pl`, is provided that will read in a file with **definitions** and **constraints** until encountering the end of the file (EOF). Note that `part2.pl` imports the file `part2Helper.pl`, which is also provided in the PA3 web directory. As it stands, all it does is print out the parsed data from the statements. See the **What Is To Be Done** section above on what you need to add to `part2.pl`.

To test this file, run the following:

```
swipl -s part2.pl  -t main --quiet -- example1.txt  > part2Result1.txt
```

Notes for Oz Programmers

Compiling and Running

All coding will be done in `part2.oz`. `part2Helper.oz` contains a helper (`Helper.tokenize`) that will read and tokenize the spec file for you.

```
ozc -c part2Helper.oz #Only need to do this once
ozc -c part2.oz
ozengine part2.ozf example1.txt #Run one or both of these to test
ozengine part2.ozf example2.txt
```

A Warning About Attempting to Circumvent The Natural Language Processing Grammar in Oz

In Oz, it is possible to do much more advanced list manipulation out of the box than in Prolog. Hence, many people may be inclined to make an attempt at extracting the relevant information from the token list by looking at the positions of tokens in the line or looking for keywords that surround a token and extracting from there. While it is certainly possible to do this, we have explicitly designed the grammar so this approach will be very difficult. Thus, you are strongly encouraged to make use of a Natural Language Processing Grammar and to avoid naive extraction approaches.

What To Submit & Grading

For **Part 1**, please submit either `relations.pl` or `relations.oz`, depending on the language you choose. These files will be tested with a main file similar to `part1Tester.pl` or `part1Tester.oz`, but with queries not revealed publicly. This should be no issue, as properly implemented rules should be rather flexible.

For **Part 2**, please submit either your completed version of `part2.pl` or `part2.oz`, depending on the language you choose. The `read_line` (`Helper.tokenize`) files in both languages are provided and do not need to be edited.

You should also submit a `readme.txt` file with your name (or names if you have a partner). If you have any comments; reports of bugs, self-assessments, anything, put it in your `readme`.

For all submitted files, please zip them up into a `.zip` file, with its filename matching the pattern:

```
[your_rcs_id]_[language_identifier].zip
```

For example, `jsmith_pl.zip` and `jdoe_oz.zip` are acceptable. For partners, please specify both of your RCS IDs, like so: `jdoe_jsmith_pl.zip`.

The **grade components** are as follows:

Part	**Percentage**
1: Relationship Reasoning	45%
2: Hogwarts Fundraising Constraint Problem	(see sub-parts below)
2a: NLP Parser	20%
2b: Constraint Calculation	25%
Code Clarity & Comments	10%

Warning About Posting Solutions Publicly & Inter-Team Code Sharing

Do not post any code you submit for this assignment onto any public website or network share. Also, your code from your team (which can be at most two people) should be wholly yours; Do not look at or use code made by other students and/or teams as your own. If a violation of either of these rules is detected, your grade will be heavily affected.

Links & Documentation for (SWI) Prolog

- Quickstart documentation:
  - `https://www.swi-prolog.org/pldoc/man?section=quickstart`
- Installers for version 9.0.4 (Windows/MAC):
  - `https://www.swi-prolog.org/download/stable` (for Linux, check your package manager for `swi-prolog`)
- Definite Clause Grammar documentation:
  - `https://www.swi-prolog.org/pldoc/man?section=DCG`
- Constraint Logic Programming over Finite Domains documentation:
  - `https://www.swi-prolog.org/pldoc/man?section=clpfd`