UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG
School of Computer Science & Applied Mathematics

Artificial Intelligence (COMS4033A/7044A)
Knowledge Representation and Reasoning

1 Introduction

In this lab, we'll be tackling a few problems using the *Prolog* programming language discussed in class. A version of Prolog known as SWI-Prolog is available for download on all operating systems. Feel free to install this on your own system. However, there is also an online Prolog interpreter located at `https://swish.swi-prolog.org/` that we can use to write and execute programs. To use this online interpreter, click on the button to create a new program. The main window is where you will write code, while the query window is where you will ask questions of your knowledge base.

2 Variables

In Prolog, variables start with an uppercase letter and constants with a lowercase letter. For example, in singing(mark), `mark` is a constant while in singing(X), `X` is a variable. We can use it to assign a value to a variable. For example:

```
isTwo(X):- X is 2.
```

3 Facts and Rules Syntax

Every fact, rule and query ends with a full stop. Below is an example of a knowledge base, which is made up of facts and rules. In this example, facts are as follows:

```
singing(elon).
playingGuitar(mark).
playingGuitar(satya).
singing(jeff).
```

These facts have a natural interpretation; for example, they state that there is an object `elon` who is singing, another object `mark` who is playing the guitar, etc.

The knowledge base also possesses rules or relations. Examples of such rules are as follows:

```
playingGuitar(elon):- happy(elon).
happy(mark):- singing(mark), playingGuitar(mark).
happy(satya):- singing(satya); playingGuitar(satya).
sad(satya):- not(playingGuitar(satya)).
grumpy(jeff):- singing(jeff)-> false ; true
```

We describe the various syntactical elements of the language below.

3.1 If

The most important symbol in Prolog is arguably `:-`, which represents the phrase "is implied by". You can think of this as implication, but where implication is usually left-to-right, here it is right-to-left. Consider the rule

```
playingGuitar(elon):- happy(elon).
```

The above line can be read as `elon` plays the guitar **if** `elon` is happy. Mathematically, we can think of this as `happy(elon)` $\Rightarrow$ `playingGuitar(elon)`, so when `elon` is happy, it follows that `elon` is playing the guitar.

3.2 Boolean Operators

Consider the next rule

```
happy(mark):- singing(mark), playingGuitar(mark).
```

The above can be read as `mark` is happy if `mark` is singing **and** playing the guitar. The comma in Prolog is used to denote Boolean "and".

The next rule demonstrates the case of "or", which is denoted by a semicolon:

```
happy(satya):- singing(satya); playingGuitar(satya).
```

Here we have the rule that `satya` is happy if `satya` is singing **or** playing the guitar.

Finally, the negation operator is specified by the `not(...)` function.

```
sad(satya):- not(playingGuitar(satya)).
```

The above means that `satya` is sad when `satya` is **not** playing the guitar.

3.3 Other Prolog Elements

There are, of course, many other facets to Prolog. Since it is a Turing complete language, anything you can do in a regular procedural language can be done in Prolog. I recommend taking a look at online tutorials if you're interested in learning more about the language (e.g., `https://www.tutorialspoint.com/prolog/index.htm`), but for now, we will just list a few more concepts of interest:

- We use the == symbol to check whether two variables or terms are equal to one another. To check if they are not equal, use `\=`. However, if we wish to check equality to a numeric value (e.g. checking if a variable has some value), we use `=:=`.
- In Prolog, lists are indicated by square brackets. For example, `[X, Y Z]` is a list with three elements. We can check whether an element is contained in a list by using the `member` function. For example, `member(A, [X,Y,Z])` would return false.
- When performing operations on lists, it is often useful to use the head-tail notation `[H | T]` where the variable `H` (it can be any variable name) represents the first element in the list, and `T` represents the rest of the list. This allows us to write recursive predicates. For example, `f([H])` applies the predicate `f` to a list with one element, while `f([H|T]` applies the same predicate to a list with more than one element.

Armed with the basics of Prolog, we will now tackle two problems requiring logical inference—something that Prolog is very good at!

4 Family Tree

In this task, you will be required to build a program to determine relations in a large family. You should write your knowledge base to encode the facts and rules that are presented below. You should also use the ability to query the knowledge base to ensure that your rules are logically sound.

4.1 Facts

First, encode the following facts. These are the only facts, and you may not add any more facts to your knowledge base.

1. The following people are males: `greg`,`adam`,`trent`,`martin`,`marcus`,`gabriel`,`dave`,`michael`.
2. The following people are females: `lucy`,`amy`,`kgomotso`,`naledi`,`karen`,`michelle`.
3. `trent` is the parent of `greg`.
4. `naledi` is the parent of `greg`.
5. `trent` is the parent of `adam`.
6. `trent` is the parent of `kgomotso`.
7. `karen` is the parent of `adam`.
8. `karen` is the parent of `kgomotso`.
9. `gabriel` is the parent of `marcus`.
10. `gabriel` is the parent of `michelle`.
11. `gabriel` is the parent of `naledi`.
12. `amy` is the parent of `marcus`.
13. `amy` is the parent of `michelle`.
14. `amy` is the parent of `naledi`.
15. `dave` is the parent of `trent`.
16. `dave` is the parent of `martin`.
17. `lucy` is the parent of `trent`.
18. `lucy` is the parent of `martin`.
19. `martin` is the parent of `michael`.
20. `amy` is married to `gabriel`.
21. `lucy` is married to `dave`.
22. `naledi` is married to `trent`.

Hint: After encoding the above, you should have four predicates: `male`, `female`, `parent`, `married`

4.2 Rules

Now you should encode the rules about how relations are computed. The rules you must write as follows:

1. `spouse`: takes two inputs `X` and `Y` and returns whether `X` and `Y` are married.
2. `mother`: takes two inputs `X` and `Y` and returns whether `X` is the mother of `Y`.
3. `father`: takes two inputs `X` and `Y` and returns whether `X` is the father of `Y`.
4. `sibling`: takes two inputs `X` and `Y` and returns whether `X` and `Y` are siblings. Siblings share the same mother and father (hint: can you be your own sibling?).
5. `brother`: takes two inputs `X` and `Y` and returns whether `X` is the brother of `Y`. A brother is defined as a male sibling.
6. `sister`: takes two inputs `X` and `Y` and returns whether `X` is the sister of `Y`. A sister is defined as a female sibling.
7. `half_sibling`: takes two inputs `X` and `Y` and returns whether `X` and `Y` are half-siblings. Half-siblings share one common parent, but not two.
8. `half_brother`: takes two inputs `X` and `Y` and returns whether `X` is the half-brother of `Y`. A half-brother is defined as a male half-sibling.
9. `half_sister`: takes two inputs `X` and `Y` and returns whether `X` is the half-sister of `Y`. A half-sister is defined as a female half-sibling.
10. `uncle`: takes two inputs `X` and `Y` and returns whether `X` is the uncle of `Y`. An uncle is defined as one of your parent's brother.
11. `aunt`: takes two inputs `X` and `Y` and returns whether `X` is the aunt of `Y`. An aunt is defined as one of your parent's sister.
12. `grandparent`: takes two inputs `X` and `Y` and returns whether `X` is the grandparent of `Y`. A grandparent is defined as one of your parent's parent.
13. `grandmother`: takes two inputs `X` and `Y` and returns whether `X` is the grandmother of `Y`. A grandmother is a female grandparent.
14. `grandfather`: takes two inputs `X` and `Y` and returns whether `X` is the grandfather of `Y`. A grandfather is a male grandparent.
15. `nephew`: takes two inputs `X` and `Y` and returns whether `X` is the nephew of `Y`. A nephew is the son of one's brother or sister.
16. `niece`: takes two inputs `X` and `Y` and returns whether `X` is the niece of `Y`. A niece is the daughter of one's brother or sister.
17. `cousin`: takes two inputs `X` and `Y` and returns whether `X` is the cousin of `Y`. Two people are cousins if either of their parents are siblings.
18. `in_law`: takes two inputs `X` and `Y` and returns whether `Y` is an in-law of `X`. Your in-law is your spouse's parents or siblings.

19. `brother_in_law`: takes two inputs `X` and `Y` and returns whether `Y` is a brother-in-law of `X`. Your brother-in-law is your spouse's brother.

20. `sister_in_law`: takes two inputs `X` and `Y` and returns whether `Y` is a sister-in-law of `X`. Your sister-in-law is your spouse's sister.

4.3 Marking

Please submit a single `pl` file containing your entire knowledge base on Moodle. There is no automarking for this submission. Each correct fact implemented is worth $1/2$ marks. Each correct rule implemented is worth 1 mark. Total marks $= 31$.

5 Logic Puzzle

There are five children. Each child is standing next to each other in a line numbered 1–5. Each child is characterised by six properties: the colour of their shirt, their name, their favourite superhero, the drink they are holding, their pet animal and their place in line. All of these values are unique: no two children share the same characteristic.

We are now given the following clues about the children and must work out, for each child numbered 1–5, their six characteristics.

1. Ntokozo is wearing a red shirt.
2. Saul owns a dog.
3. Coffee is drunk by the child with the green shirt.
4. Kgosi drinks tea.
5. The child with the green shirt is immediately to the right of the child with the white shirt.
6. The child who likes Superman owns snails.
7. The child with the yellow shirt likes Hulk.
8. The child in the middle of the lineup drinks milk.
9. Jay is the first child in the lineup.
10. The child who likes Ironman is adjacent to the child who has a pet parrot.
11. The child who likes Hulk is adjacent to the child who owns a horse.
12. The child who likes Spiderman drinks orange juice.
13. John likes Batman.
14. Jay is next to the child with the blue shirt.

You are now required to write a Prolog program to determine possible assignments such that the above rules hold true. To begin thinking about this problem, note that each child can be thought of as a list of 6 variables, with each variable representing a particular characteristic. For example, the declaration

```
C = [red, ntokozo, coffee, superman, snails, 2]
```

represents a child with a red shirt called Ntokozo, who drinks coffee and likes Superman, has snails as pets and is in position 2 in line. The order of characteristics is arbitrary, but we must ensure that it is consistent throughout the program. Here we have listed them in the order colour, name, drink, superhero, pet, position.

To start you off, let's look at the first rule: "Ntokozo is wearing a red shirt". This rule can be encoded as follows:

```
rule1(C1,C2,C3,C4,C5):- member(C, [C1,C2,C3,C4,C5]), C = [red, ntokozo,_,_,_,_].
```

The rule takes in five variables, representing each child, and finds the child who is a) one of those five and b) whose characteristics are `red` and `Ntokozo`. Note that we use the underscore for irrelevant variables. Given this as a basis, proceed to encode the remaining rules. Refer to each rule by the predicate `RuleN` where `N` represents which rule number is being encoded.

- Rules 2,3,4,6,7,8,9,12,13 are similar to the first rule.
- For the remaining rules, consider using the position of the children to determine whether they are next to one another.

5.1 Solving the Puzzle

Before we can solve the puzzle, we need one final predicate. In particular, when the program outputs an answer, we must be sure to check that the answer contains all five colours, names, superheroes, etc. Therefore, write a predicate that accepts two lists and returns true if all elements of the first list are members of the second list. You may want to use recursion and ideas regarding head-tail notation to solve this problem. The predicate(s) for this should be called `members`.

Once you have this, add the following to your knowledge base:

```
solve(C1,C2,C3,C4,C5) :-
C1 = [C1_col,C1_name,C1_drink,C1_superhero,C1_pet,1],
C2 = [C2_col,C2_name,C2_drink,C2_superhero,C2_pet,2],
C3 = [C3_col,C3_name,C3_drink,C3_superhero,C3_pet,3],
C4 = [C4_col,C4_name,C4_drink,C4_superhero,C4_pet,4],
C5 = [C5_col,C5_name,C5_drink,C5_superhero,C5_pet,5],
rule1(C1,C2,C3,C4,C5),
rule2(C1,C2,C3,C4,C5),
rule3(C1,C2,C3,C4,C5),
```

```
rule4(C1,C2,C3,C4,C5),
rule5(C1,C2,C3,C4,C5),
rule6(C1,C2,C3,C4,C5),
rule7(C1,C2,C3,C4,C5),
rule8(C1,C2,C3,C4,C5),
rule9(C1,C2,C3,C4,C5),
rule10(C1,C2,C3,C4,C5),
rule11(C1,C2,C3,C4,C5),
rule12(C1,C2,C3,C4,C5),
rule13(C1,C2,C3,C4,C5),
rule14(C1,C2,C3,C4,C5),
members([white, green, red, blue, yellow],
[C1_col,C2_col,C3_col,C4_col,C5_col]),
members([saul, john, ntokozo, kgosi, jay],
[C1_name,C2_name,C3_name,C4_name,C5_name]),
members([orange, coffee, milk, tea, water],
[C1_drink,C2_drink,C3_drink,C4_drink,C5_drink]),
members([spiderman, batman, superman, ironman, hulk],
[C1_superhero,C2_superhero,C3_superhero,C4_superhero,C5_superhero]),
members([dog, cat, snails, horse, parrot],
[C1_pet,C2_pet,C3_pet,C4_pet,C5_pet]).
```

If you query the knowledge base with `solve(C1,C2,C3,C4,C5).` it should produce a valid solution to the problem.

5.2 Marking

Please submit a single `pl` file containing your entire knowledge base on Moodle. There is no automarking for this submission. Each correct rule implemented is worth 1 mark. Total marks is 14.