ICS 1019/2206 – Knowledge Representation & Reasoning
30% Project

Joel Azzopardi

March 2017

This project will require you to implement 2 reasoning methods that were covered during lectures. These are:

1. The parsing of Horn Clauses in CNF, and reasoning with them using the back-chaining SLD procedure (covered in Chapter 5 of the Brachman and Levesque book and slides).
2. Construction of Inheritance Networks, and reasoning with them using pre-emption and redundancy (covered in Chapter 10 of the Brachman and Levesque book and slides).

1 Parsing of Horn Clauses, and Reasoning using Back-Chaining

This section is based on Chapter 5 of the Brachman and Levesque book. It involves the construction of a Knowledge Base from a list of Horn Clauses provided in Conjunctive Normal Form (CNF), and then performing reasoning on them using the Back-chaining algorithm.

1.1 Aim

The aim is to use any third generation language of your choice (Java, Javascript, C, C++, C#, Python, ...) to implement a program that can read and parse clauses written in CNF and perform reasoning on the resulting knowledge base. The user can then query your program to check if a query clause can be resolved to an empty clause. These are described in more detail in the subsections below.

You are not expected to produce any fancy interfaces. A basic interface (even a command line interface) will suffice as long as the system allows the user inputs and provides the relevant outputs as described below.

1.2 Input clauses

The input to your system (can be read from a text-file or by direct user input) consists of statements in the following format:

```
[ <literal 1>, <literal 2>, ...]
```

An example input can be:

```
[Toddler]
[¬Toddler, Child]
[¬Child, ¬Male, Boy]
[¬Infant, Child]
[¬Child, ¬Female, Girl]
[Female]
[Male]
```

You are to assume that the input clauses are all horn clauses and that they do not contain any errors. Your system does not need to try to detect incorrect input, or be robust enough to cater for errors in the input.

Note also that the input clauses can **only** contain atomic literals (or their negation). You should not worry about handling functions or predicates.

1.3 User Queries

The user should be allowed to query the system by submitting his/her negated query (i.e. $\neg\alpha$) in CNF. Your system does not need to attempt to do the query negation automatically.

An example query to the above Knowledge Base is:

```
[¬Girl]
```

1.4 Resolution

Your system should load the knowledge base and the user query (all in CNF). It should then use the back-chaining SLD procedure to try to resolve to the empty clause (`[]`). This procedure is found in Page 92 of the book, and in Slide 87 of the corresponding slide deck. As output, the system only needs to output messages such as "**SOLVED**" if the empty clause was obtained through resolution and "**NOT SOLVED**" if not.

You can have your system output informative messages as well before the final '**SOLVED**" or "**NOT SOLVED**".

Note that the back-chaining SLD procedure uses recursion. You need to be careful from infinite loops when implementing it.

2 Construction and Reasoning with Inheritance Networks

This section is based on Chapter 10 of the Brachman and Levesque book. It involves the construction of simple inheritance networks, and then performing reasoning on these inheritance networks to resolve conflicts arising from multiple inheritance.

2.1 Aim

The aim is to use any third generation language of your choice (Java, Javascript, C, C++, C#, Python, ...) to implement a program that can construct an inheritance network based on the input provided, and perform reasoning on the constructed network. The user can then query your program on possible inheritances. These are described in more detail in the subsections below.

You are not expected to produce any fancy interfaces. A basic interface (even a command line interface) will suffice as long as the system allows the user inputs and provides the relevant outputs as described below.

2.2 Input to construct Inheritance Network

The input to your system (can be read from a text-file or by direct user input) consists of statements in the following format:

```
<Sub-Concept> IS-A <Super-Concept>
<Sub-Concept> IS-NOT-A <Super-Concept>
```

Given the example below, your program should construct the inheritance network shown in Figure 1.

```
Clyde IS-A FatRoyalElephant
FatRoyalElephant IS-A RoyalElephant
Clyde IS-A Elephant
RoyalElephant IS-A Elephant
RoyalElephant IS-NOT-A Gray
Elephant IS-A Gray
```

Note that you are **not** required to provide a graphical representation of the inheritance network.

You are to assume that concept names do not contain any spaces, and that the input provided does not contain any errors (your system does not need to try to understand incorrect input). You are also to assume that the inheritance network is **acyclic**.

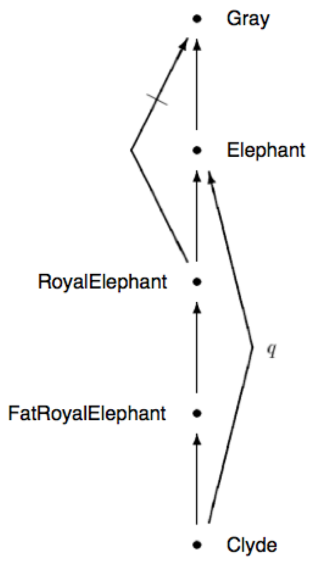

Figure 1: Example inheritance network

2.3 User Queries

The user should be allowed to query the system using only one type of command:

<Sub–Concept> **IS-A** <Super–Concept>

For example,

Clyde **IS-A** Gray

2.4 Query Handling

2.4.1 Searching for all possible paths

Upon receiving a query, the system should output all the possible paths from the query's sub-concept to the query's super-concept.

For the example above, the system should output the following paths:

- Clyde **IS-A** Elephant **IS-A** Gray

- Clyde **IS-A** FatRoyalElephant **IS-A** RoyalElephant **IS-A** Elephant **IS-A** Gray
- Clyde **IS-A** FatRoyalElephant **IS-A** RoyalElephant **IS-NOT-A** Gray

Remember that a path can only have a single **IS-NOT-A** link, and this can only occur at the very end. If whilst searching for the possible paths, your system encounters a **IS-NOT-A** link, it does not need to search further from the node at the end of this **IS-NOT-A** link.

2.4.2 Outputting the preferred path/s

The system should then provide which path is preferred according to the **shortest distance** metric, and according to the **inferential distance** metric. Note the preferred path calculated using *inferential distance* should not include any **inadmissible links** – i.e. neither **redundant links** nor **pre-empted** links. More information about this may be found in slides 173 – 174

For the example above, these outputs will be:

- Preferred Path (Shortest Distance):
  - Clyde **IS-A** Elephant **IS-A** Gray
- Preferred Path (Inferential Distance):
  - Clyde **IS-A** FatRoyalElephant **IS-A** RoyalElephant **IS-NOT-A** Gray

If there is more than 1 preferred path (or the system can not determine the best path), all 'preferred' paths should be output.

3 Documentation / Report

You are to write a short documentation (about 2-4 pages per problem) that describes your system, the data structures used, and how the operations were handled. Your report should not exceed 9 pages in total.

Most importantly, your documentation should include specific (short) instructions on how to execute your program/s, as well as an input data examples which are known to work. The instructions should not direct the user to use NetBeans, Eclipse or any other IDE. If you use Java, you should provide instructions that are based on the *java* command. Ideally you provide a jar file.

You should also specify: the parts of the project that have been implemented and are working successfully; the ones that are partially working; and what is not working. Marks will be given accordingly, but effort for parts that are not or are partially working will also be taken into consideration.

4 Mark Distribution

- Back-Chaining on Horn Clauses
  - Construction of knowledge base according to user inputs – 15%
  - Resolution by backchaining – 25%.
  - Documentation – 10%.
- Inheritance Networks
  - Construction of inheritance network according to user inputs – 12%
  - Output of possible paths according to user queries – 12%.
  - Determination of shortest path – 4%.
  - Determination of preferred path according to inferential distance – 12%.
  - Documentation – 10%.

5 Group Work

Note that this project is an individual project. No group work is permitted and plagiarism will be penalised.

6 Deadline

The deadline for this project is **Friday 26th May, 2017** at **noon**. The project should be uploaded on VLE.

7 Difficulties

In case of difficulties, please do not hesitate to send an email on *joel.azzopardi@um.edu.mt.*

It is recommended that you are familiar with the relevant concepts from the lecture notes, and that you actually use the algorithms provided in the book/slides.