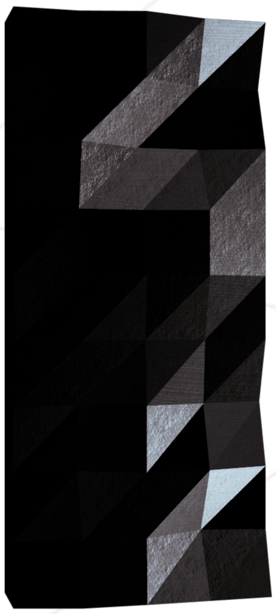

Expert system

How to do simple logic. Or not-so-simple.

Summary: The goal of this project is to make a propositional calculus expert system.

Version: 2

Contents

Chapter I

Foreword

Here's a little quote from the movie Spaceballs :

```
Dark Helmet          : Careful! I said ACROSS her nose, not up it!
Laser Gunner         : Sorry sir! I'm doing my best!
Dark Helmet          : ... who made that man a gunner?
Officer              : I did sir. He's my cousin.
Dark Helmet          : Who is he?
Colonel Sandurz      : He's an A*, sir.
Dark Helmet          : I know that! What's his name?
Colonel Sandurz      : That is his name sir. A*, Major A*!
Dark Helmet          : And his cousin?
Colonel Sandurz      : He's an A* too sir, Gunner's mate First
                       Class Philip A*!
Dark Helmet          : How many A*s do we have on this ship, anyway?

[Entire bridge crew stands up and raises a hand]

Entire Bridge Crew   : Yo!
Dark Helmet          : I knew it. I'm surrounded by A*s!

[Dark Helmet pulls his face shield down]

Dark Helmet          : Keep firing, A*s!
```

This project is a lot easier to do if you have watched `Spaceballs` at least three times.

Chapter II

Objectives

The goal of this project is to make an expert system for propositional calculus.

Chapter III

General guidelines

- You are free to use whatever programming language you want to make the engine, but keep in mind that your choice in language could impact performance, and while there will not be a direct influence on your grade as such, we will probably have a competition between everyone who did the project at some point.

  Of course, the language used in the input files to describe the facts and rules is non-negociable.

Chapter IV

Mandatory part

You must implement a backward-chaining inference engine. Rules and facts will be given as a text file, the format of which is described in the appendix. A fact can be any uppercase alphabetical character.

Your program must accept one parameter, which is the input file. It will contain a list of rules, then a list of initial facts, then a list of queries. For each of these queries, the program must, given the facts and rules given, tell if the query is true, false, or undetermined.

By default, all facts are false, and can only be made true by the initial facts statement, or by application of a rule. A fact can only be undetermined if the ruleset is ambiguous, for example if I say "A is true, also if A then B or C", then B and C are undetermined.

If there is an error in the input, for example a contradiction in the facts, or a syntax error, the program must inform the user of the problem.

Here's a list of the features we would like your engine to support. You can only get 100% of the grade by implementing all of them.

- "AND" conditions. For example, `"If A and B and [...]  then X"`
- "OR" conditions. For example, `"If C or D then Z"`
- "XOR" conditions. For example, `"If A xor E then V"`. Remember that this means "exclusive OR". It is only true if one and only one of the operands is true.
- Negation. For example, `"If A and not B then Y"`
- Multiple rules can have the same fact as a conclusion
- "AND" in conclusions. For example, `"If A then B and C"`
- Parentheses in expressions. Interpreted in much the same way as an arithmetic expression.

Chapter V

Bonus part

- Interactive fact validation : The system allows the user to change facts interactively to check the same query against a different input without changing the source file, or to clarify an undeterminable fact, for example from an OR conclusion without further information.
- Reasoning visualisation : For a given query, provide some feedback to explain the answer to the user, for example "We know that A is true. Since we know A | B => C, then C is true", or any other type of visualization you like. Even better, output everything in formal logic notation, and go show Thor : If he likes it, you'll win a beer.
- "OR" and "XOR" in conclusions. For example, `"If A then B or C"`
- Biconditional rules. For example, `"A and B if-and-only-if D"`. In case it's unclear, this means not only `"If A and B then D"` but also `"If D then A and B"`
- Whatever other interesting bonus you want, as long as it's coherent with the rest of the project.

Chapter VI

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.

For the defense session, be prepared to :

- Explain your implementation of the inference engine.
- Justify your decisions in terms of algorithms and data structures.
- Run your program on varied examples that demonstrate your successful implementation of the project. You're expected to provide some examples for the defense. Any capability of your program that is not demonstrated by your examples will NOT be counted as present.

Chapter VII

Appendix

VII.1 Symbols and rules of precedence

The following symbols are defined, in order of decreasing priority:

- (and) which are fairly obvious. Example : `A + (B | C) => D`
- `!` which means NOT. Example : `!B`
- `+` which means AND. Example : `A + B`
- `|` which means OR. Example : `A | B`
- `ˆ` which means XOR. Example : `A ˆ B`
- `=>` which means "implies". Example : `A + B => C`
- `<=>` which means "if and only if". Example : `A + B <=> C`

VII.2 Input file format

```
:~/expert/$ cat -e example_input.txt
# this is a comment$
# all the required rules and symbols, along with the bonus ones, will be
# shown here. spacing is not important

C          => E       # C implies E
A + B + C  => D       # A and B and C implies D
A | B      => C       # A or B implies C
A + !B     => F       # A and not B implies F
C | !G     => H       # C or not G implies H
V ^ W      => X       # V xor W implies X
A + B      => Y + Z   # A and B implies Y and Z
C | D      => X | V   # C or D implies X or V
E + F      => !V      # E and F implies not V
A + B      <=> C      # A and B if and only if C
A + B      <=> !C     # A and B if and only if not C

=ABG                  # Initial facts : A, B and G are true. All others are false.
                      # If no facts are initially true, then a simple "=" followed
                      # by a newline is used

?GVX                  # Queries : What are G, V and X ?
:~/expert/$
```