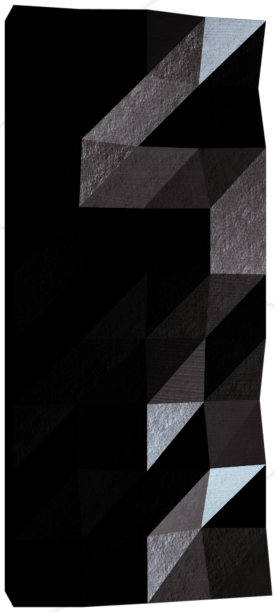

Expert system

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

Summary: The goal of this project is to create an expert system for propositional calculus.

Version: 3.1

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 any programming language to build the engine. However, keep in mind that your choice of language may affect performance. While this will not directly influence your grade, there will likely be a competition among all participants who completed the project at some point.

  Naturally, the language used in the input files to describe the facts and rules is non-negotiable.

Chapter IV

Mandatory Part

You must implement a backward-chaining inference engine. Rules and facts will be provided 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: the input file. This file will contain a list of rules, a list of initial facts, and a list of queries. For each query, the program must determine, based on the provided facts and rules, whether the query is true, false, or undetermined.

By default, all facts are considered false and can only be made true through the initial facts statement or the application of a rule. A fact can only be undetermined if the ruleset is ambiguous. For example, if the input states, "A is true, and 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 notify the user of the issue.

Below is a list of features we expect your engine to support. To achieve 100% of the grade, you must implement 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," which is true if and only if one of the operands is true.
- **Negation.** For example, `"If A and not B then Y"`.
- **Multiple rules with the same conclusion.** For example, several rules can result in the same fact as their conclusion.
- **"AND" in conclusions.** For example, `"If A then B and C"`.
- **Parentheses in expressions.** These should be interpreted similarly to how they are used in arithmetic expressions.

Chapter V

Bonus Part

- **Interactive Fact Validation:** The system allows the user to change facts interactively to check the same query against different inputs without modifying the source file. This could also help clarify an undetermined fact, for example, resulting from an OR conclusion without additional information.
- **Reasoning Visualization:** For a given query, provide feedback to explain the answer to the user. For instance: `"We know that A is true. Since we know A | B => C, then C is true"`. Alternatively, you can use any visualization method you like. Even better, output the reasoning in formal logic notation and show it to Thor—if he approves, you might win a beer.
- **"OR" and "XOR" in Conclusions:** Support conclusions with "OR" or "XOR" logic. For example: `"If A then B or C"`.
- **Biconditional Rules:** For example, `"A and B if-and-only-if D"`. This means not only `"If A and B then D"` but also `"If D then A and B"`.
- **Other Bonuses:** Feel free to implement any other interesting bonus feature, as long as it remains coherent with the rest of the project.

Chapter VI

Submission and Peer Evaluation

Submit your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Make sure 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 choices in terms of algorithms and data structures.
- Run your program on various examples that demonstrate the successful implementation of the project. You are expected to provide these examples for the defense. Any functionality of your program that is not demonstrated by your examples will NOT be considered as implemented.

Chapter VII

Appendix

VII.1 Symbols and Rules of Precedence

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

- (and): Used for grouping expressions. Example: `A + (B | C) => D`.
- `!`: Represents NOT. Example: `!B`.
- `+`: Represents AND. Example: `A + B`.
- `|`: Represents OR. Example: `A | B`.
- `^`: Represents XOR (exclusive OR). Example: `A ^ B`.
- `=>`: Represents "implies". Example: `A + B => C`.
- `<=>`: Represents "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/$
```