Detection of fallacy in natural text

Final project for CS135: Computational Semantics

Amin Saeidi, Georg Konwisser (Shlomo), and Aaditya Prakash

1 Abstract

In this project we investigate and build a simple fallacy detection system in Haskell from natural text. This project implements relation extraction at level sufficient to allow for conversion of natural text into logical form. Once the logical form is obtained, we check if the input expression matches the pattern of the fallacy expression. Doing so we accept certain pattern variance appropriate for fallacy detection. This project is limited to propositional fallacies like 'affirming the disjunct', , only and does not support other types of fallacies, for example quantification fallacies and formal syllogistic fallacies. It has been aimed to keep the parsed input as a free natural text but certain constrained and structure has been assumed.

2 Design

After careful thought and experimenation with several Haskell packages for First Order Logic conversion (FOL Solver, Haskell Inference Engine, Type Theory based theorem prover, Haskell Inductive Prover, University of Zurich's Attempto Project and Jan van Eijck's TOTP), we decided to write our own relation extraction customized for fallacy proving. There were several reasons for this decision, but few of them were -

1. Conversion to first order logic, was an overkill for our purposed
2. Most of the libraries we experimented, had their fair share of issues with compatibility, execution or performance.
3. We wanted to put into practice some of the semantics knowledge obtained from this class.

2.1 Converting sentences to logical form

One of the major challenges of this project was to be able to write down the provided set of sentences into logical form of simplified knowledge sets as premises and a conclusion. It was important to track the modifiers. As mentioned earlier in Abstract, we decided to focus only on propositional fallacies, thus we did not have to deal with quantifiers. Following schematic logic was used to convert the text to logical form.

1. Take the sentence
2. Set the boundaries so that seperate knowledge can be extracted. Replace the following explicit logic operators -
   (a) AND "but" with keyword "and" "." with keyword "and"
   (b) IMPLICATION "therefore" with "=>" "so" with "=>"
3. Stem the Sentence as a whole (just normal stemming and not chunking) (Stemming seperates the sentences with "." as different array elements so that we get different knowledge set for separate variable)
4. Tag the sentence (stemming before so that we get the same tags for the same verb but represented in different format)
5. Extract the premise and the conclusion part.
6. Make the premise and the conclusion part, separately into well formed expression
   (a) Find, NOT in the proposition, extract and apply it to the proposition as unary operator
   (b) Recursively apply OR (Disjunction) and AND (Conjunction) to the list of propositions variables

2.2 Detecting fallacy from logical form

Once the given set of sentences were converted into logical form, we employ the following recursive pattern matching algorithm. Matching succeeds if the expression trees only differ at places where the fallacy expression has leaves (variables). In those cases however the corresponding part in the input expression tree has to imply the variable in the fallacy expression. In order to check this implication we evaluate if it is a tautological relation. According to our exploit of definition of tautology in propositional logic, tautologies are extended to any logically valid formula.

Since the tree-leaf implication relies on having the same variables in input and fallacy expressions, we extract the variable set from the input and use it to substitute the variable set in the fallacy expression.

3 Fallacies implemented

We have implemented the detection of following propositional fallacies-

3.1 Affirming the disjunct

Affirming the disjunct is a type of fallacy which occurs when one of the logical expression in a set of disjuncts is concluded to be false in the evidence of another premise. In logical formula, this is expressed as follows:

$$p \lor q$$

$$p$$

$$\implies \neg q$$

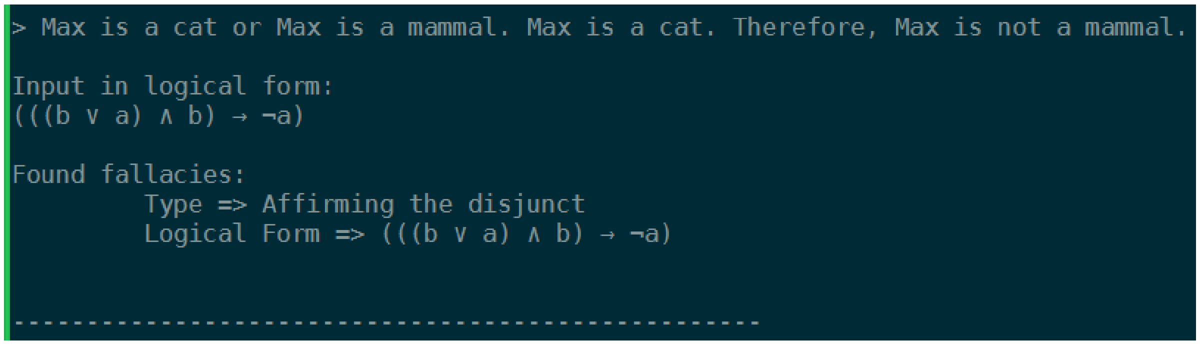
Example of this type of fallacy from Wikipedia:

Max is a cat or Max is a mammal.

Max is a cat.

Therefore, Max is not a mammal.

Following is the screenshot of parsing the above example in our program.



```
> Max is a cat or Max is a mammal. Max is a cat. Therefore, Max is not a mammal.

Input in logical form:
(((b ∨ a) ∧ b) → ¬a)

Found fallacies:
        Type => Affirming the disjunct
        Logical Form => (((b ∨ a) ∧ b) → ¬a)


---------------------------------------------
```

Figure 1: Detecting affirming the disjunct

3.2 Affirming the consequent

Affirming the consequent fallacy occurs when given a conditional and a premise assumes the validity of the consequent. In logical form this can be expressed as follows:

$$p \to q$$

$$q$$

$$\implies p$$

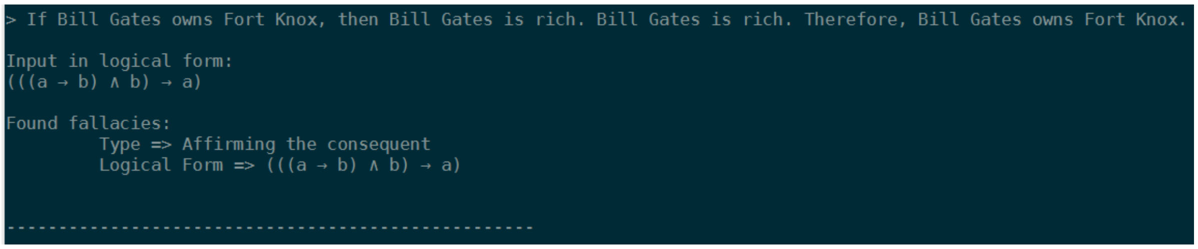
Example of this type of fallacy from Wikipedia:

If Bill Gates owns Fort Knox, then he is rich.

Bill Gates is rich.

Therefore, Bill Gates owns Fort Knox.

Following is the screensho of parsing the above example (without anaphora resolution) in our program:



```
> If Bill Gates owns Fort Knox, then Bill Gates is rich. Bill Gates is rich. Therefore, Bill Gates owns Fort Knox.

Input in logical form:
(((a → b) ∧ b) → a)

Found fallacies:
        Type => Affirming the consequent
        Logical Form => (((a → b) ∧ b) → a)


--------------------------------------------------
```

Figure 2: Detecting affirming the consequent

3.3 Denying the antecedent

Denying the antecedent is similar in strucutre to affirming the consequent but different in intepretation. In denying the antecedent, given a conditional and invalidity of the antecedent, fallacy is to assume invalidity of consequent. In logical form this can be expressed as follows:

$$p \to q$$

$$\neg p$$

$$\implies \neg q$$

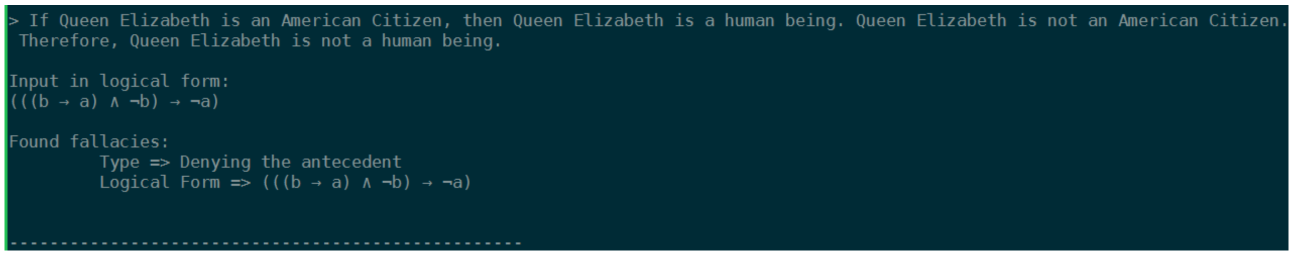
Example of this type of fallacy from Wikipedia:

If Queen Elizabeth is an American citizen, then she is a human being.

Queen Elizabeth is not an American citizen.

Therefore, Queen Elizabeth is not a human being.

Following is the screensho of parsing the above example (without anaphora resolution) in our program:



```
> If Queen Elizabeth is an American Citizen, then Queen Elizabeth is a human being. Queen Elizabeth is not an American Citizen.
 Therefore, Queen Elizabeth is not a human being.

Input in logical form:
(((b → a) ∧ ¬b) → ¬a)

Found fallacies:
        Type => Denying the antecedent
        Logical Form => (((b → a) ∧ ¬b) → ¬a)


--------------------------------------------------
```

Figure 3: Detecting denying the antecedent

4 Code

A brief description of each of the haskell file has been provided below. For detailed analsis, kindly refer to the function headers and inline comments in those code files.

Source	Description
Main.hs	Serves as the entry point of the application. See README.md* for usage.
Detector.hs	Checks for fallacy, when provided with expression in propositional logic.
LogicalShortcuts.hs	Makes shortcuts for the logical operations in propostional logic.
Tests.hs	Includes the unit tests for different modules of the application.
TextToLogical.hs	Converts the given natural text into propositional logic form for the fallacy.

*On the root folder of the submission, README.md file has been provided with detailed instruction on running the modules and the unit tests.

5 Further Work

We see this as start of the project and envision this to extend for all kinds of fallacies and for much wider acceptance of text. We have also been working on anaphora resolution, so that simpler pronouns in the given premise may be bound to same knowledge set. Another immediate extension would be to extend this from propositional logic to first order logic so that quantification fallacies like existential fallacy could be detected. We also aim to make the parser work with implicit conclusion rather than explicit conclusion with verbal cues.