Inference Based Theorem Prover

Arman Shah shahx252@umn.edu
Ry Wiese wiese176@umn.edu

November 21, 2018

1 Description

Many inference algorithms, such as resolution, forward chaining, and backward chaining, exist to determine whether a given quantified statement is entailed by a knowledge base. Our problem is to, given a quantified statement and a knowledge base of known theorems and axioms, generate a proof that the quantified statement is, in fact entailed. A proof is a list of statements, each of which either binds a new variable to a value or is a statement that obviously entails from a statement in the knowledge base or from a previous statement in the proof.
For the scope of our problem, we will start by proving first order logic statements similar to those given in class, and then improve our algorithm to prove higher order logic statements, particularly those over sets and functions, similar to the types of proofs given in a college level intro to real analysis course. Concretely, we aim to prove a statement similar to the following:

$$\forall \epsilon > 0, \exists K \in \mathbb{N}, \forall j \in \mathbb{N}, (j \geq K) \implies (|x^5 + 2x^2 + x| < \epsilon).$$

2 Approach

As stated by Howard in [2], there is a direct correspondence between algorithms and mathematical proofs. We will implement an OCaml algorithm which, on input of a knowledge base and a statement to be proved, uses this correspondence and the recursive grammar of quantified statements to parse the statement and bind the appropriate variables to values that will make the statement true. The difficult part of this problem is choosing the proper values to bind to variables. As a simple example, consider the statement

$$\forall x, (\exists y, (y^2 = x)).$$

To prove this statement, we first bind the variable x by adding the statement "given x", to the proof, adding x to our list of bound variables, and recursively proving

$$\exists y, (y^2 = x).$$

Now, we must bind the variable y by adding "let $y = \sqrt{x}$" to the proof and adding $y = \sqrt{x}$ to the knowledge base, then recursively prove

$$y^2 = x.$$

The full proof is as follows:

Given x, let $y = \sqrt{x}$. $y^2 = (\sqrt{x})^2 = x$.

Because we chose the "right" value of y, the rest of the proof follows immediately. However, for more difficult proofs and for computers that do not have the same sense of intuition as a mathematician, this choice is far from obvious. For this task, we will use a few different OCaml functions that will be called from our main parser algorithm that will implement different combinations of resolution, forward chaining, and backward chaining to infer the proper choice. We will also implement principles from a few other algorithms to make our inference process more robust. We will borrow principles from the Markov logic network described in [3] to assign a heuristic value to each statement in the knowledge base based on the probability that the statement will be used in the proof. The program described in [1] includes a technique of ordering the knowledge base in a way that improves the efficiency of the forward chaining process. There are certain features presented in [4], such as a model-elimination reduction rule and an implementation of Iterative Deepening Depth First Search, that guarantee completeness. We will selectively implement these features in ways that will improve the efficacy of our solution.
Once all variables have been bound, the remainder of the proof simply requires running a similar inference algorithm to show that our statement is entailed by the updated knowledge base, modified to print out each of the statements in the chain.

3 Software

We will be writing our own OCaml program from scratch, since the functional nature of OCaml is ideal for the structure of the statements we are trying to prove and for the efficient use of object lists, which is how we will represent our knowledge base. We will be implementing existing algorithms for resolution and forward/backwards chaining, as well as parts of the algorithms cited above, but these will need to be re-writtin in OCaml using our typed representation of logical quantified statements.

4 Experimental Setup

Once our algorithms have been written, we must first test the quality of our results on a sample of several statements. This will be determined by the percentage of these statements that we are actually able to find proofs for. Moreover, any number of correct proofs may exist for a given problem, but not all

of them are clear or concise for a human reader. We will have to compare the generated proofs to known proofs for these problems to determine whether our proofs are of acceptable readability.
Beyond testing the quality of the proofs, we would like to experiment with different implementations of our inference sub-algorithms. We will be writing a few different versions of our algorithm, one that uses only forward chaining, one that uses only backwards chaining, and some that use both, as well as ideas from the other known algorithms, for different purposes within our program. We will compare the runtimes of these different implementations to see where in our program the each of the different inference algorithms and techniques is most efficient.

5 Timeline

Time is limited for the project. To ensure success we have devised a weekly plan of what we should accomplish each week. From November 25 to December 1 we plan on getting more familiarity with the analysis proofs ourselves without computers. That same week we will also gather all the different analysis problems we want to test the algorithms on and write the code for the program to parse through the proof and get a general outline of the program. From December 2 to December 8 the rest of the code will be written for the algorithms we are testing. The week of December 9 will be spent running our tests and collecting the data, while writing the paper in parallel. December 16 to December 19 will be saved to finish up the paper.

References

[1] A. J. Nevins. Plane geometry theorem proving using forward chaining. *Artificial Intelligence*, 6(1):1–23, 1975.

[2] C.-H. L. Ong and C. A. Stewart. A curry-howard foundation for functional computation with control. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '97, pages 215–227, New York, NY, USA, 1997. ACM.

[3] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1):107–136, Feb 2006.

[4] M. E. Stickel. A prolog technology theorem prover: Implementation by an extended prolog compiler. *Journal of Automated Reasoning*, 4(4):353–380, Dec 1988.