

Logic for Zebra-Puzzle Style Problems

April 3, 2011

1 Problem Definition

Logic puzzles are commonly used both for entertainment and educational purposes. One popular example of this type of puzzle, known as the zebra puzzle, gives a series of clues about a neighborhood and the person must determine who in the neighborhood has a zebra in their home. The aim of this project is to write an algorithm to solve this type of problem.

Many people approach them as hobbies or something to occupy the mind, but in the Law School Admission Test logic puzzles help decide a person's future. Showing the reasoning behind the solution to a puzzle can not only help publishers check solutions to the problems they create, but it can help teach students how to solve such problems. This system is a way to give a computer methods of reasoning, however basic. Also, it can be used to help understand the common problems in knowledge based artificial intelligence.

2 Related Work

Most of the problems that can be solved with logic can be solved with CSPs as well, and are usually more efficient.

For example, in [Salavati et al., 2010] many local search algorithms for solving the Zebra Puzzle have been implemented and tested, which show how the problem can be effectively solved by a search algorithm. The paper compares backtracking algorithms with different methods (such as forward chaining, minimum conflicts and others) and suggests a way to reduce the depth of the search tree by adding new constraints to the problem. Knowledge-based algorithms are not taken into account, so logic is not used in any of the implemented algorithms.

On the contrary, in [Costa et al., 1991] the authors present a new Prolog System that visits And-Or-Trees in parallel using multiprocessor programming, providing an interesting speed-up for computation of Prolog scripts. They then compare the speed-

up improvement of the new system, solving various problems among which is the Zebra Puzzle (also known as Einstein's puzzle). Unfortunately the implemented algorithm for solving the problem is not directly stated, but the results are shown in form of runtime and number of inferences made by the various algorithms.

3 Approach

Initially, we searched for Prolog style inference engines in Python. This search found only wrappers for Prolog. This allowed us to run some tests, but we found that the wrappers had the same flaw as Prolog. All of them would enter an infinite loop, with no hope of returning a result. Instead, we decided to implement a logical reasoning system based off of the initial, partially implemented code that [Russell and Norvig, 2009] provides. In order to use this to solve the type of propositional logic problems described in [Russell and Norvig, 2009], we needed to convert our logical language to propositional logic.

Our logical language can be considered a decidable subset of first order logic. We can define attributes and elements, which can be used to define the basic assertions for reasoning. Variables can be defined in order to represent unknown elements that may or may not make some assertions true.

The four major problems we needed to address to obtain a working system were disambiguating the variables, restricting their domains, converting into propositional logic, and returning the results of queries. Solving these four problems allows us to interface with the code from [Russell and Norvig, 2009]. The following are our solutions to these problems, which left us with a system that could solve very basic logic puzzles.

3.1 Problem Representation

For the reader who is unaware of Zebra Puzzle style logic problems, they are composed of a sequence of carefully worded statements from which specific facts

may be inferred. One such statement in the Zebra Puzzle is "The Englishman lives in the red house." We have translated the sentences from this form into our own logical language. The statement provided is transformed into $lives(X, englishMan) \Leftrightarrow color(X, red)$. In the Zebra Puzzle, this and thirteen other statements provide information about five houses and their owners, pets, drinks, and cigarettes. The questions that must be answered are "Who owns the Zebra?" and "Who drinks water?"

3.2 Variable Disambiguation

Variables in different assertions refer to different values. In a knowledge base where we need to concatenate all of the assertions, there is a need to disambiguate them. A way to achieve this is to scan the assertion for variables when "telling" the knowledge base. The variable is then replaced with a new, unique instance. For example, the two assertions,

$$lives(X, englishMan) \Leftrightarrow color(X, red)$$

$$right(X, Y) \Leftrightarrow left(Y, X)$$

are translated into

$$lives(X_1, englishMan) \Leftrightarrow color(X_1, red)$$

$$right(X_2, X_3) \Leftrightarrow left(X_3, X_2)$$

3.3 Variable Domains

In order to find a possible solution, we need to be able to assign values to variables. We implemented a very naïve way to do this. Essentially, our program is attempting all of the possible combinations of variables values until the query is proven true. A way to reduce the number of assignments attempted is to set the domain for each attribute. *Lives*, for instance, takes *houses* as the first parameter and *people* as the second. These parameters have a restricted domain, where *houses* and *people* are subsets of all the possible values. In this way, we can reduce the domain of a particular variable, therefore reducing the number of possible assignments.

3.4 Propositional Logic Conversion

We had to convert our logical statements into propositional logic to take advantage of what [Russell and Norvig, 2009] had provided us with. A way to do this is, given an assignment for all of the variables, to replace the attributes with propositional logic literals. An example of this conversion follows.

$$right(X_2, X_3) \Leftrightarrow left(X_3, X_2)$$

$$(\neg right(X_2, X_3) \vee left(X_3, X_2)) \wedge$$

$$(\neg left(X_3, X_2) \vee right(X_2, X_3))$$

$$(\neg right(a, b) \vee left(b, a)) \wedge (\neg left(b, a) \vee right(a, b))$$

$$(\neg rab \vee lba) \wedge (\neg lba \vee rab)$$

As is shown, the assertion is first converted into a CNF formula. This is followed by values being assigned to the variables. Finally, each clause is changed into its own literal.

3.5 Query Variable Matching

A way to ask queries is to use variables in order to find possible assignments that makes the query true, given the knowledge base. An example of this is the query $lives(X, englishMan) \wedge color(X, Y)$, which asks the color of the house where the Englishman lives. When the knowledge base contains only two assertions,

$$lives(X, englishMan) \Leftrightarrow color(X, red)$$

$$lives(a, englishMan)$$

the algorithm will return all of the matches of the query variables. In this case, $(X = a, Y = red)$.

4 Evaluation

In Figure 1, and in tabular format in Table 1, we can see how the number of variables/values associations changes while "telling" the knowledge base new assertions. The first 10 sentences did not contain variables, so they have been omitted. It is clear that restricting the variables to their own domains significantly reduced the number of possible combinations for our program to test, even if the exponential growth is still present. In Table 1, our results using the domain are in the column "With" and those that lack the domain restriction are under "Without." The last two rows in the "Without" have too many possibilities to compute and, thus, have been omitted.

It has been possible to represent the Zebra Problem with just 27 assertions, 8 attributes ("color", "pet", etc.) and 30 elements ("red", "zebra", etc.) Unfortunately we were not able to collect data on execution time and number of inferences, due to the enormous amount of computation time required. Our program cannot solve any but the most basic of logic puzzles right now.

5 Discussion

Our results were not what we were hoping for.

The problem is not solvable using a naïve approach, as clearly shown in the chart. The algorithm without variable domain was able to perform reasoning with at most 7 variables in the knowledge base, while inferring the variable domain from the attributes let us reach a higher number of variables, more than 11, which is still not good enough to perform a complete inference of new knowledge.

The problem is NP, but with more intelligent algorithm, such as backtracking or forward-chaining ([Russell and Norvig, 2009]), the number of variables assignments may be reduced drastically. Local Search algorithm may also be applied to obtain a result more quickly, but losing completeness.

A way to solve the problem is to reduce the number of variables in the knowledge base, trying to infer new knowledge from the current assertions, and then restricting the variables domain to just one element. This is not a trivial task and will require further analysis.

There is much more work to be done on this project. First, we need to find a way to reduce the number of assignments that are attempted by our system. The current implementation takes too long to compute anything beyond basic inferences. Secondly, we wanted to be able to print out what logical steps the system took to answer the queries we gave it. Right now, the system is unable to do this, and can only print the result. One way to implement this would be to reduce the problem into smaller and easier subproblems that can be used to solve the final query. Finally, since we focused more on implementing the logical inference system, we were unable to include the natural language processing portion of our original proposal. We would have liked to group the noun and verb phrases to build the relations and will

work on this in the future.

It is clear to us now that reasoning is not a task that can be easily attempted in two weeks of work, but has great potential and research in this field may lead to important results in the scientific world.

References

- [Costa et al., 1991] Costa, V., Warren, D., and Yang, R. (1991). Andorra I: a parallel Prolog system that transparently exploits both And-and or-parallelism. *ACM SIGPLAN Notices*, 26(7):93.
- [Russell and Norvig, 2009] Russell, S. and Norvig, P. (2009). *Artificial intelligence: a modern approach*. Prentice hall, 3rd edition.
- [Salavati et al., 2010] Salavati, S., Hajjarzadeh, S., and Mazloom, M. (2010). An Optimized Method for Solving Zebra Puzzle. In *Computer and Electrical Engineering, 2009. ICCEE'09. Second International Conference on*, volume 1, pages 448–451. IEEE.

Sent.	Clauses	With	Without	Literals	Vars
10	10	1	1	0	0
11	11	5	5	2	1
12	13	125	125	10	3
13	15	3,125	3,125	22	5
14	17	15,625	117,649	26	6
15	19	78,125	4,782,969	30	7
16	21	390,625	214,358,881	34	8
17	23	1,953,125	--	38	9
18	26	48,828,125	--	42	11

Table 1: The number of possible assignments with and without the domain assigned to the variables.

```

[scale=0.5] arrows VertexStyle=[shape = circle,
                                draw] [style=->]
[x=0,y=0]PER1,1 [x=2,y=0]PER2,2
[x=4,y=0]PER3,3 [x=0,y=2]PW1,1 [x=2,y=2]PW2,2
[x=4,y=2]PW3,3 [x=2,y=4]Individual
[x=2,y=8]Team [x=6,y=6]HomeTeamwins

```

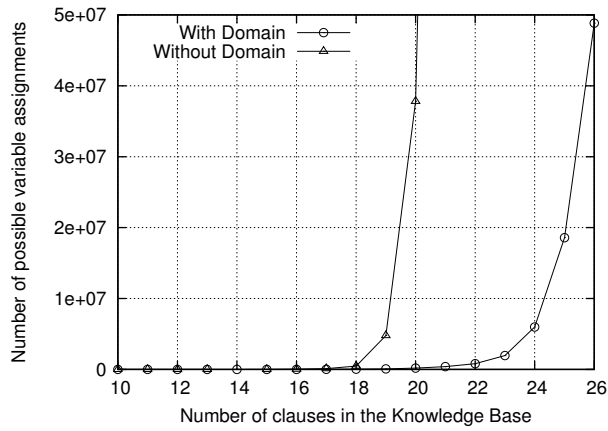


Figure 1: Number of variable assignments by number of clauses in the knowledge base, with and without restricting the variables domain.