CSC2108HS: Automated Reasoning with Machine Learning

Assignment I

Author: Nico Schiavone, ID 1011339222

Description of Files per question can be found in README.md

Q1a – Translation

I solved this problem using z3-solver, framing it as an SMT problem. After loading in the encrypted text, I first stripped it of punctuation and removed the duplicate words, as we are not considering sentence structure or neighbouring words when determining valid sentences. Then, I set up a dictionary mapping the valid characters (a-z + 0-9) to letters in the alphabet (a-z).

For each encrypted word, I iterated through the list of 10000 English words and selected candidate words with the same length as the encrypted word. For each of the candidate words, I formed the conjunction expression for the mapping from encrypted character to decrypted character (e.g. XYZ <-> AND means X <-> A ∧ Y <-> N ∧ Z <-> D) to form the constraints for that decryption. One of the decryptions must be true, so I formed a disjunction with all the possible candidate words.

Once formed, I pass this list of constraints to z3 solver to generate a possible solution, convert it to a readable dictionary, and use the dictionary to translate the passages.

The mapping key and translation are as follows (also in the P1a_result.txt file)

{'a': 'n', 'b': 'a', 'c': 'o', 'd': 'a', 'e': 's', 'f': 'a', 'g': 'm', 'h': 'p', 'i': 'a', 'j': 'k', 'k': 'w', 'l': 'i', 'm': 'z', 'n': 'h', 'o': 'g', 'p': 'd', 'q': 'b', 'r': 'a', 's': 'c', 't': 'f', 'u': 'l', 'v': 'r', 'w': 'v', 'x': 'a', 'y': 'x', 'z': 'a', '0': 'a', '1': 'a', '2': 'a', '3': 'u', '4': 'e', '5': 'y', '6': 'a', '7': 't', '8': 'a', '9': 'a'}

WE EXTEND OUR BEST WISHES TO YOU, RESIDENTS OF ANOTHER WORLD. AFTER READING THE FOLLOWING MESSAGE, YOU SHOULD HAVE A BASIC UNDERSTANDING OF CIVILIZATION ON EARTH. THROUGH HARD WORK AND CREATIVITY, PEOPLE HAVE CREATED A GREAT CIVILIZATION FILLED WITH MANY DIFFERENT CULTURES. WE HAVE ALSO BEGUN TO UNDERSTAND THE LAWS GOVERNING THE NATURAL WORLD AND THE DEVELOPMENT OF HUMAN SOCIETIES. WE VALUE EVERYTHING WE HAVE ACHIEVED. BUT OUR WORLD IS NOT PERFECT. HATE EXISTS, AS DOES WAR. BECAUSE OF CONFLICTS BETWEEN THE FORCES OF PRODUCTION AND THE RELATIONS OF PRODUCTION, WEALTH IS NOT FAIRLY DISTRIBUTED, AND MANY PEOPLE LIVE IN POVERTY AND SUFFERING. HUMAN SOCIETIES ARE WORKING HARD TO RESOLVE THE DIFFICULTIES AND PROBLEMS THEY FACE, WORKING HARD TO CREATE A BETTER FUTURE FOR EARTH CIVILIZATION. THE COUNTRY THAT SENT THIS MESSAGE IS ENGAGED IN THIS EFFORT.

WE ARE DEDICATED TO BUILDING AN IDEAL SOCIETY, WHERE THE LABOR AND VALUE OF EVERY MEMBER OF THE HUMAN RACE ARE FULLY RESPECTED, WHERE THE MATERIAL AND SPIRITUAL NEEDS OF EVERYONE ARE FULLY MET, SO THAT CIVILIZATION ON EARTH MAY BECOME MORE PERFECT. WE HOPE TO MAKE CONNECTIONS WITH OTHER ADVANCED SOCIETIES IN THE UNIVERSE. WE LOOK FORWARD TO WORKING TOGETHER WITH YOU TO BUILD A BETTER LIFE IN THIS VAST UNIVERSE.

THIS WORLD HAS RECEIVED YOUR MESSAGE. I AM A PERSON WHO BELIEVE IN PEACE IN THIS WORLD. IT IS THE LUCK OF YOUR CIVILIZATION THAT I AM THE FIRST TO RECEIVE YOUR MESSAGE. I AM WARNING YOU, DO NOT ANSWER. DO NOT ANSWER. DO NOT ANSWER. THERE ARE MILLIONS OF STARS IN YOUR DIRECTION. AS LONG AS YOU DO NOT ANSWER, THIS WORLD WILL NOT BE ABLE TO FIGURE OUT WHERE YOUR MESSAGE CAME FROM. BUT IF YOU DO ANSWER, THE SOURCE WILL BE LOCATED RIGHT AWAY. PEOPLE FROM THIS WORLD WILL TARGET YOUR PLANET AND TAKE OVER YOUR WORLD. DO NOT ANSWER. DO NOT ANSWER. DO NOT ANSWER.

Q1b – Palindrome

I solved this problem using z3-solver, framing it as an SMT problem. Using the assumption that the two constituent numbers would have the same bit length, I formed two Boolean vectors of length (n.bit_length() - 1). I chose this bit length as the resultant bit length must be n.bit_length(), so the only possible configuration is for both constituents to be n.bit_length() - 1 length and for the carry bit at the end to give the MSB of n.

Thus, the Boolean representation encodes each bit of the constituent numbers as a Boolean value. The palindrome condition is easily enforced by having the i'th bit be equal to the (n-i-1)'th bit. I then perform a bit-by-bit addition using an encoded full-adder and a starting carry of 0. Each sum bit expression is stored in the result, and the end carry expression is constrained to evaluate to 1 (by the assumption above).

I then iterate through the result, matching each bit in the binary representation of n to the desired state of the expressions in result.

Finally, I call z3 solver and extract the two binary numbers from the model.

Below is a snippet of the results (the last case binary is omitted from this PDF due to length, but is present in P1b_results.txt):

1.

44 = 17 + 27

In binary: 101100 = 10001 + 11011

2.

426 = 195 + 231

In binary: 110101010 = 11000011 + 11100111

3.

130686 = 65151 + 65535

In binary: 11111111001111110 = 1111111001111111 + 1111111111111111

4.

7887885102 = 3952331223 + 3935553879

In binary: 111010110001001111001001100101110 =
11101011100100111100100111010111 + 11101010100100111100100101010111

5.

3017435147447357088839800445926438687686904519406704578731798375848055
0292850726010027516741429727891945486366493691712785577992990514585975
9141468926324014180668109424490555196408928664480223861034652769306700
0795579380080347934761405906051456566922733133648076135930820362212251
10204142908645394181908936406 =
1508717573723678544419852638422148054785442721703411492547974987597574
4140198465158018777269704572616573208729386207086360600260903644431236
5809279790376263392069124054385749935401958825550680058920709788141880
2870209842385908932134712358857769663369203745796871976808351524720682
17648806210920165105102265195 +
1508717573723678544419947807504290632901461797703293086183823388250480
6152652260852008739471725155275372277637107484626424977732086870154739
3332189135947750788598985370104805261006969838929543802113942981164819
7925369537694439002626693547193686903553529387851204159122468837491568
92555336697725229076806671211

(binary omitted for length)

Q2 – Palindrome in Lean

Solution in Palindrome.lean

Q3 – Maze Solver

I chose to approach this problem by decoding the maze from Lean4 into text characters, solving via an external solver, and passing the resulting instructions back into Lean4.

To accomplish this, I extracted all the wall coordinates from the maze structure, along with the starting (x, y) position, and the maze size (x_size, y_size). I passed all of these to stdin as plaintext numbers, one line at a time, and used a companion Python3 program to encode the maze into an array. The maze array represented the walls as 1, the floor (movable area) as 0, and defined the exit as any floor on the edge of the maze.

Starting from the given player starting position, the algorithm runs a breadth-first search in each direction. For every step, the algorithm records the coordinates, so it does not revisit the same location twice. In this way, the algorithm guarantees that if an exit is found, that it was found by the shortest path possible. The algorithm iterates, treating each new coordinate as a source and moving in each possible direction (N, S, E, W) without revisiting cells or running into walls. At each step, the algorithm tries to 'escape' by checking if it is on the edge and on a floor tile. If escape is possible, the path thus far is returned in the form of spaced numbers with the encoding (N = 0, E = 1, S = 2, W = 3, EXIT = 4).

This string of one-spaced numbers is interpretable by Lean4, so this is outputted directly to stdout, solving the maze. The algorithm directly solves each of the given mazes in a reasonable time (time complexity O(nm) where n = # of rows, m = # of columns). This gives a solution to each maze, proving they can be solved.