112-2 SoC Verification Final Project Report

Yu-Ting Cheng, Electrical Engineering, National Taiwan University
Group: 10
email: b08202013@ntu.edu.tw
Name of the Project: *SeqEngine*

***Abstract*—The 15-puzzle is a classic game that has intrigued puzzle enthusiasts for over a century. It's been proved that half of the starting tile arrangement is solvable. In this article, we try to use sequence engine (ITP, BMC and PDR) to solve the reachability problem. From experiment results, we conclude that while PDR engine outperformed in simple testcases, ITP and BMC are more stable and reliable in complex testcases.**

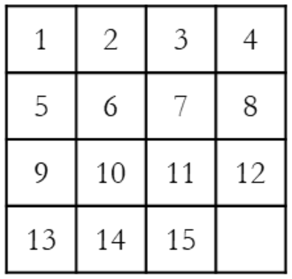
Keywords—15-puzzle, formal verification, model checking, interpolation, property-directed reachability, bounded model checking

I. Introduction

The 15-puzzle, also known as the sliding puzzle, gem puzzle, or mystic square, is a classic game that has intrigued puzzle enthusiasts and researchers alike for over a century. The puzzle consists of a 4x4 grid containing 15 numbered tiles and one empty space. The objective is to rearrange the tiles from a given randomized configuration into numerical order by sliding the tiles into the empty space. The final arrangement should be as same as shown in Figure 1.

The origins of the 15-puzzle date back to the late 19th century. It is widely believed to have been invented by Noyes Chapman, a postmaster in Canastota, New York, around 1874. The puzzle gained immense popularity in the United States in the 1880s, leading to what was known as the "15-puzzle craze." During this period, the puzzle captivated the public, and numerous variations and challenges were created, sparking widespread interest and competition.

The puzzle also holds a significant place in the history of combinatorial game theory and recreational mathematics. It has been studied extensively, leading to various mathematical insights and algorithmic approaches. Notably, the 15-puzzle contributed to the early development of theories regarding permutations and group theory.



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Figure 1: The final (target) arrangement of 15-puzzle

II. Prior Work

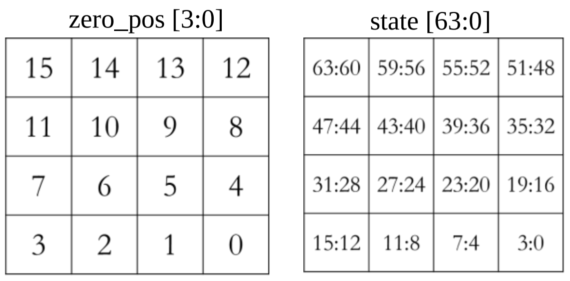
In 1879, W.W. Johnson and W.E. Story provided a mathematical proof regarding the solvability of the 15-puzzle. They demonstrated that exactly half of the possible initial positions of the puzzle are impossible to solve. This insight was derived from their analysis of the puzzle's permutations and inversions. Specifically, they showed that the puzzle's configuration space can be divided into two distinct sets: those that can be solved and those that cannot. Each set contains an equal number of configurations, confirming that the probability of a randomly shuffled 15-puzzle being solvable is exactly 50%.

Johnson and Story's work was pioneering in understanding the mathematical foundations of the 15-puzzle. Their proof involved considering the parity of permutations, highlighting that a single tile swap changes the parity of the permutation, thus making it impossible to transform an odd permutation into an even one through valid moves. This fundamental property underlies the solvability criteria used to determine whether a given 15-puzzle configuration can be solved

III. Modeling

To model the 15-puzzle game into a SAT problem, we want to model the "position of puzzles" as states, the "action of sliding" as input, and "legal moves" into the state transition function. Theoretically, the 15-puzzle game is equivalent to the permutation of number from 0 to 15, with the blank tile be 0. Mathematically, there're 16! possible permutation for 16 distinct numbers. For simplicity, we model the states by a 64-bits register, with 16 puzzle position and each position can be ranged from 0 to 15 (4-bits). Although this modeling is not the most efficient in terms of register, as it take 16^{16} state space, we believe that this model is simple and straightforward for understanding and implementation.

Overall, the system consist of 64-bit state register and 4-bit register to record the position of blank tile. There're three inputs, clock, reset signal and action of sliding in the system. There's



zero_pos [3:0]

15	14	13	12
11	10	9	8
7	6	5	4
3	2	1	0

state [63:0]

63:60	59:56	55:52	51:48
47:44	43:40	39:36	35:32
31:28	27:24	23:20	19:16
15:12	11:8	7:4	3:0

Figure 2: In the system, "zero_pos" register records the position of the blank tile. "state" register records the tiles arrangement.

only one output in the system, which assert if the state is a target permutation.

IV. Experiment

After the modeling process, the 15-puzzle game is equivalent to SAT problem by specifying different initial tile arrangement (state) and see if it can reach the target arrangement (!p). We resort to three different sequence engines, include interpolation-based bounded model checking (ITP), bounded model checking (BMC) and property-directed reachability (PDR). Originally, we also attempted to use BDD engine to solve the SAT problem. However, the engine failed to build transition relation and thus cannot prove SAT or UNSAT. Therefore, we use ITP, BMC and PDR to solve the problem.

For experiment testcases, we design ten tile arrangements SAT testcases which are shown in Table 1. The blank tile is marked in red. For every testcases, the (n)-th testcase can be reached by sliding one step from (n-1)-th testcases, for n = 2, 3, … 10. For example, the 2nd testcase can be reached by sliding tile 14 of the 1st testcase rightward. The 4th testcase can be reached by sliding tile 9 of the 3rd testcase downward. The experiment results and charts are shown in Table 2 and Figure 3.

1	2	3	4	5
1 2 3 4	1 2 3 4	1 2 3 4	1 2 3 4	1 2 3 4
5 6 7 8	5 6 7 8	5 6 7 8	5 6 7 8	🟥 6 7 8
9 10 11 12	9 10 11 12	9 10 11 12	🟥 10 11 12	5 10 11 12
13 14 🟥 15	13 🟥 14 15	🟥 13 14 15	9 13 14 15	9 13 14 15

6	7	8	9	10
🟥 2 3 4	2 🟥 3 4	2 3 🟥 4	2 3 4 🟥	2 3 4 8
1 6 7 8	1 6 7 8	1 6 7 8	1 6 7 8	1 6 7 🟥
5 10 11 12	5 10 11 12	5 10 11 12	5 10 11 12	5 10 11 12
9 13 14 15	9 13 14 15	9 13 14 15	9 13 14 15	9 13 14 15

Table 1: SAT testcases tile arrangement

In the experiment data, we can see that the time periods required to solve the problem increases as the testcases frame increases. It's reasonable and understandable since it takes longer to reach the target arrangement for more complicated testcases. Also, both ITP and BMC estimate the correct timeframe to reach the target arrangement, which is the same as we expected.

For PDR engine, it initially outperformed ITP and BMC for easier testcases. However, the time period suddenly increase at testcase 9 and even fail to prove SAT in testcase 10 (after running for more than 24 hours). Also, the timeframe estimated by PDR is not the same as we expected and designed in the testcases. Overall, we conclude that PDR is good at solving simple and easy testcases. However, for more complicated testcases, ITP and BMC are more stable and reliable.

For UNSAT cases, we feed the most simple unsat testcases into ITP, BMC and PDR engine. However, none of these engine can manage to prove the unsat property in 24 hours. Despite these engines didn't prove sat property meanwhile, we believe that the unsat property can be proved by simpler and more dedicated modeling.

	Testcase	1	2	3	4	5	6	7	8	9	10
ITP	time (s)	0.18	0.39	0.57	1.89	3.25	7.32	7.53	13.36	21.71	26.38
	frame	1	2	3	4	5	6	7	8	9	10
BMC	time (s)	0.14	0.15	0.3	0.21	0.96	1.9	4.12	8.17	12.91	23.91
	frame	1	2	3	4	5	6	7	8	9	10
PDR	time (s)	0.17	0.18	0.13	0.21	0.19	0.19	0.21	0.41	109.9	N/A*
	frame	2	3	2	3	2	2	3	3	7	N/A*

Table 2: SAT testcases experiment data

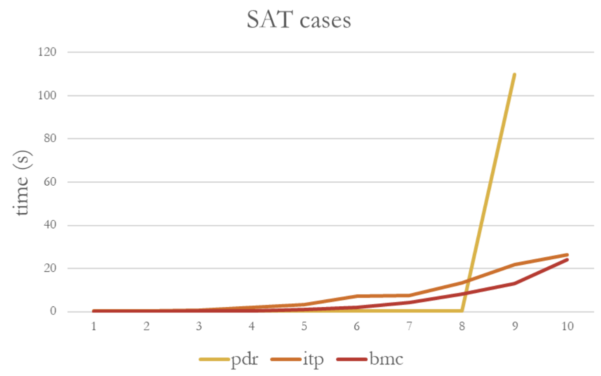


Figure 3: SAT testcases experiment chart

V. How to compile & test

Before compile and test my codes, make sure you've placed the /final directory under the gv engine. There're only two instruction to test my codes. First, type "./gv" to enter the gv engine. Second, type "do ./final/tests/itp.dofile" to run the dofile of itp. To test BMC or PDR, simply change "itp.dofile" with "bmc.dofile" or "pdr.dofile".

To test different testcases, users can specify different initial state in sequential always block (around line 381~425). There're ten SAT testcases and one UNSAT testcases provided.

VI. Demo Video Link

https://youtu.be/RI0Y836qa3g

References

[1] Johnson, W. W., & Story, W. E. (1879). Notes on the "15" puzzle. *American Journal of Mathematics*, *2*(4), 397-404.

[2] Slocum, J., & Sonneveld, D. (1880). The 15 puzzle: how it drove the world crazy. *The puzzle that started the craze of*.

[3] Archer, A. F. (1999). A modern treatment of the 15 puzzle. *The American Mathematical Monthly*, *106*(9), 793-799.

[4] Spitznagel Jr, E. L. (1967). A new look at the fifteen puzzle. *Mathematics Magazine*, *40*(4), 171-174.

[5] Parberry, I. (2014). A Memory-Efficient Method for Fast Computation of Short 15-Puzzle Solutions. *IEEE Transactions on Computational Intelligence and AI in Games*, *7*(2), 200-203.

[6] Hayes, R. (2001). *The Sam Loyd 15-Puzzle*. Trinity College Dublin, Department of Computer Science.