COMPUTING LAB I (CS69011): 2021 Autumn
Assignment 03 : MasterMind Gaming Using SAT Solver

ADARSH G KRISHNAN (21CS60R55)

Mastermind is a popular mind game, requiring considerable logical reasoning skills.

- The game is played using a decoding board, with a shield at one end covering a secret row of four large holes, and ten additional open rows – each containing four large holes next to a set of four small holes.
- There are code pegs of eight different colours with round heads, which will be placed in the large holes on the board.
- There are also key pegs, some coloured black, some white, which are flat-headed and smaller than the code pegs; they will be placed in the small holes on the board.
- One player becomes the code maker, the other the codebreaker. The code maker chooses a pattern of four code pegs and the codebreaker tries to crack it.
- The code maker will randomly choose the hidden code and store it in an array. Thereafter, in each iteration, it will invoke the codebreaker for a new guess. After the codebreaker makes a guess, the code maker will compare the guess with the hidden code and provide the codebreaker the number of black and white key pegs awarded
- The black and white key pegs received in response to each guess enhances the knowledge of the codebreaker. A smart codebreaker will not make guesses, where one or more pegs are wrong based on its existing knowledge. For example, if the guess, red-red-red-red, received no key pegs, then it knows that there are no red pegs in the hidden code, and therefore no future guess should include a red peg. Therefore, a key requirement for a codebreaker is to find a guess that does not contradict its existing knowledge. We shall use a Boolean Satisfiability Solver (SAT-solver) to find guesses that are consistent with the existing knowledge.
- Boolean SAT solvers check the satisfiability of logical formulas over Boolean variables. The first step in modelling a constraint satisfaction problem in SAT is to choose the variables modelling the system.
- We use a TogaSAT solver to find a new guess that does not have any contradictions with the clauses generated from the previous iterations. When it receives the scores (in terms of the white and black key pegs) from the code maker, it generates suitable clauses and pushes them into the SAT solver.

Approach Used for Encoding Colours & Positions :

Positions: 1 (Left), 2, 3, 4 (Right)
Colour codes: R – red, B – blue, G – green, Y – yellow, O – orange, P – purple, W – white, K – black

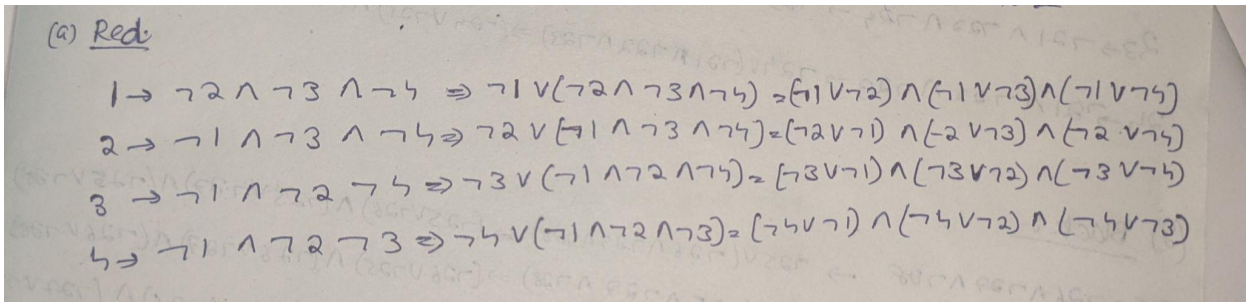
Colour	**Position 1 Index**	**Position 2 Index**	**Position 3 Index**	**Position 4 Index**
RED	1	2	3	4
BLUE	5	6	7	8
GREEN	9	10	11	12
YELLOW	13	14	15	16
ORANGE	17	18	19	20
PURPLE	21	22	23	24
WHITE	25	26	27	28
BLACK	29	30	31	32

Problem Statement #1 : Repetition of Colour is not allowed.

Initial Constraints have to be generated which can be considered as rules of the game.These constraints are then feeded into the SAT-Solver and depending on that SAT Solver will be giving an Initial Guess for the codebreaker.Initial Constraints are given below :

Constraint #1 : Same colour should not be repeated at different positions.

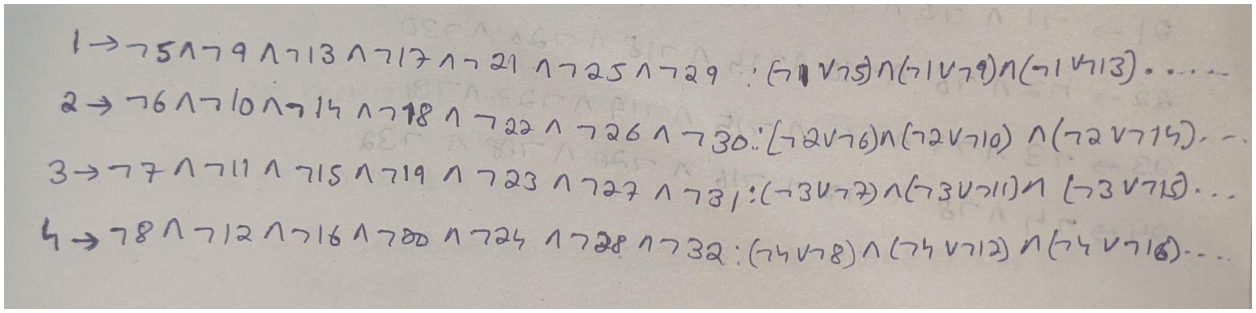
Example Clauses for RED Colour .



Likewise we can add for all the 8 colours.

Constraint #2 : No 2 Colour should be present at the same Position.

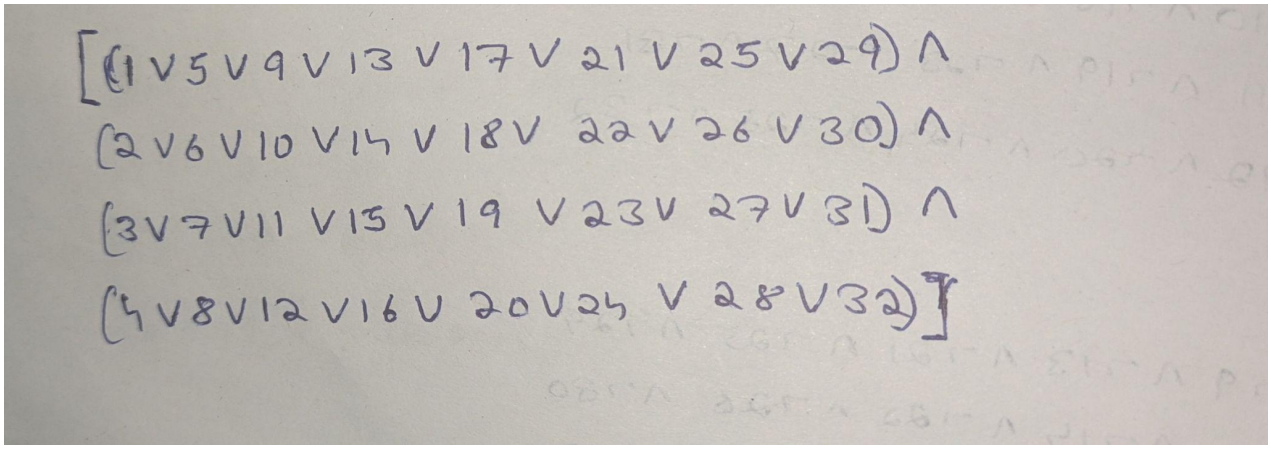
Example Clauses for 4 positions :



Likewise we can add for the remaining positions.

Constraint #3 : At Least 4 Colors should be present in the output of SAT Solver.

Clauses for encoding the above constraint.



Boolean constraints added by the codebreaker & how are they encoded?
Here **a,b,c,d** are the positions of colours in the current Guess.

```
int a = CodeBreakerGuess[0];
int b = CodeBreakerGuess[1];
int c = CodeBreakerGuess[2];
int d = CodeBreakerGuess[3];
```

Above code snippet shows how a,b,c,d values are assigned.

Constraint #1 : If **no** colour is in the correct position.

Clause : $\neg a \wedge \neg b \wedge \neg c \wedge \neg d$: Negate all the positions inside the correct guess.

Constraint #2 : If **one** colour is in the correct position.

Clause 1 : $\neg a \wedge \neg b$
Clause 2 : $\neg c \wedge \neg d$
Clause 3 : $\neg b \wedge \neg c$
Clause 4 : $\neg b \wedge \neg d$
Clause 5 : $\neg a \wedge \neg c$
Clause 6 : $a \wedge b \wedge c \wedge d$

Constraint #3 : If **two** colours are in the correct position.

Clause 1 : $\neg a \wedge \neg b \wedge \neg d$
Clause 2 : $\neg a \wedge \neg b \wedge \neg c$
Clause 3 : $\neg a \wedge \neg c \wedge \neg d$
Clause 4 : $\neg b \wedge \neg c \wedge \neg d$
Clause 5 : $a \wedge b \wedge d$
Clause 6 : $a \wedge b \wedge c$
Clause 7 : $a \wedge c \wedge d$
Clause 8 : $b \wedge c \wedge$d

Constraint #4 : If **three** colours are in the correct position.

Clause 1 : $a \wedge b$
Clause 2 : $c \wedge d$
Clause 3 : $b \wedge c$
Clause 4 : $b \wedge d$
Clause 5 : $a \wedge c$
Clause 6 : $a \wedge d$
Clause 7 : $\neg a \wedge \neg b \wedge \neg c \wedge \neg d$

Above given clauses are the boolean constraints added by the code breaker.

1.All the clauses are added into the SAT Solver in CNF Form.
2.Encoding is done using Sequential counter Encoding.