GROUP 16: Mastermind

Ryan Jacobson (21rkj6)

Evan Kreutzwiser (21ejk12)

Liam Beenken (22ltb1)

Jacob McMullen (21jwm21)

Course Modelling Project

CISC/CMPE 204

Logic for Computing Science

November 2, 2023

Abstract

Our setting attempts to model the board game Mastermind. Mastermind is a code guessing game where a correct code of 4 pegs of 8 colors is chosen in a specific order. The player then makes code guesses and receives feedback. For each peg in a player's guess they are told if the peg matches the corresponding peg in the correct guess, or if the peg's color exists somewhere in the correct guess, or neither.

Our model attempts to arrive at a final code which matches the correct one by making informed guesses which are based on its previous guesses and the feedback it received from those guesses. Our constraints serve to dictate how the board develops and how the program knows which colors are valid in the right positions.

Propositions

There are 2 primary types of propositions. The X propositions represent the state of the game board, and the C propositions store the correct answer, which the model aims to reach.

X_{rnc}: True if the color c is present in the slot at row r, column n of the board.

C_{nc}: True the answer has color c in column n

For each guess, board information is recorded in the X_{rnc} propositions, and a row of these propositions without a preexisting value are where the solver's next guess is recorded. The functionality of the red and white feedback pegs of a typical mastermind game are implemented as logic directly involving the correct answer (See memory/complexity limitations in model exploration for reasoning).

We use a set of 8 colors:

Red, Orange, Yellow, Green, Blue, Purple, White, and Silver

Constraints

Capital R as a row number refers to the row the model is current making a guess for.

In any position on the board, no more than a single color can be present

E.g.: $X_{rnp} \rightarrow \neg X_{rnr} \wedge \neg X_{rno} \wedge \neg X_{rny} \wedge \neg X_{rng} \wedge \neg X_{rnb} \wedge \neg X_{rnw} \wedge \neg X_{rns}$

A color in the correct position in a previous guess is used in the same spot again

$(X_{rnc} \wedge C_{nc}) \rightarrow X_{Rnc}$

If a color is present in the answer but used in the wrong position in a previous

guess, the color must be used in one of the other 3 positions in the next one.

E.g., for a color in column 0:

$(X_{r0c} \wedge (C_{1c} \vee (C_{2c} \vee C_{3c}))) \rightarrow ((X_{R1c} \vee X_{R2c} \vee X_{R3c}) \wedge \neg X_{R0c})$

A guess must contain 1 color in each column.

$X_{Rnr} \vee X_{Rno} \vee X_{Rny} \vee X_{Rng} \vee X_{Rnb} \vee X_{Rnp} \vee X_{Rnw} \vee X_{Rns}$

This pairs with the constraint preventing multiple colors from sharing a position to ensure exactly one color is present in each position of the guess.

Model Exploration

When deciding our constraints, one potential issue became apparent quickly: how long would it take a Python model using logic to solve a full game of Mastermind? This question then became: how much RAM would it take to solve a full game? We knew this would be a problem from the start so we had to change a rule to the game, deciding that each feedback peg would correspond to a specific guessing position. This new rule would make it much easier for the Python model using logic to solve a game of Mastermind, though time and memory issues still occurred.

In the first functioning model that was made, we had the model play the whole game at once. The model would always start with a random guess since it had no information to start with. Then it would proceed to do make more guesses based off of the constraints such as using new colors when others were not in the solution, moving colors around from the wrong positions until they're in the correct positions, and keeping colors in place when in the correct positions. However, this model came with a major flaw which used up too much RAM and took too long to execute. Essentially, the model would run the entire board of 8 rows through the constraints all at once, and calculate every possibility that would lead to winning the game. The massive amount of possible solutions and time it took the model to process them quickly consumed too many resources. Due to this issue, all of our following models processed one row at a time.

First-Order Extension

If we were to shift our model to a predicate logic based system, we would have to update our basic propositions. Firstly, X_{rnc} is used to control the state of the board, denoting where previous guesses have been made.

This can be updated to the predicate $X(r, n, c)$, which is true if the colour c is present in the slot at row r, column n of the board, where r and n are previously guessed slots of the board.

The same can be done for C_{nc}, which represents the known state of the answer. It can be changed to $C(n, c)$, which is true if the answer has colour c in column n.

Next, our constraints could be updated to more clearly reflect the English language version of the problem. For example, our first constraint denotes that "In any position on the board, no more than a single colour can be present". This statement can be written in a predicate form as:

$\forall r \forall n.(X(r,n,c) \rightarrow \neg X(r,n,j))$, meaning "For all positions (r,n), if X is true for a colour c, it is not true for any other colour j.

Our second constraint is "A colour in the correct position in a previous guess is used in the same spot again", and it is used to ensure the solver continues guessing on the path closest to the known solution. This can be written using predicate logic in a similar form, as:

$\forall r \forall n \forall c.((X(r,n,c) \wedge C(n,c)) \rightarrow X(R,n,c))$, where R is the current guessing row.

Then, for our constraint "If a colour is present in the answer but used in the wrong position in a previous guess, the colour must be used in one of the other 3 positions in the next one", we can write it as:

$\forall r \forall n \forall c.(X(r,n,c) \wedge C(j,c) \wedge -C(n,c)) \rightarrow \neg X(R,n,c))$, in English: "For all previous guesses (r,n,c), if c is present in the answer but not at column n, the next guess must not include colour c at column n.

Finally for the constraint "A guess must contain 1 colour in each column", we can write it as:

$\forall r \forall n.(X(r,n,c))$, meaning for all previous rows r, and previous columns n, $X(r,n,c)$ must be true for some colour c.

Jape Proofs

- Our group has struggled with how to utilize our model in smaller scope jape proofs, and feedback is appreciated -