Cs 340 Final Project Writeup

Overview

This project was created as part of our final for Wellesley CS 340. We designed an interactive, story-driven game where players navigate through a series of word logic puzzles. Each puzzle is formally modeled using **Z3** and **Alloy**, and integrated into a themed website built using **Flask** with connected HTML, CSS, and JavaScript assets. At the end of the journey, players are rewarded with a Pearl of Undecidability.

Project Goals

We modeled five types of logic puzzles:

River Crossing Puzzles

These puzzles involve transporting people or items across a river without violating constraints (like leaving the fox with the goose). We used Alloy to model this because its temporal logic capabilities made it well-suited for expressing step by step transitions and enforcing progression.

Lying and Truth-Teller Puzzles

Inspired by Knights and Knaves problems, these involve characters who either always lie or always tell the truth. The goal is to deduce each character's identity based on logical statements. Z3 was ideal for these puzzles because of its support for logical implications.

Age Riddles

These puzzles rely on clues involving sums, products, and age relationships. Z3 was well-suited to solving these because of its built-in integer arithmetic support.

Mislabeled Boxes

A classic logic puzzle where each box is labeled incorrectly. You're allowed to peek inside one, and must use that information to deduce the correct contents of all boxes. This was modeled in Z3 using propositional constraints like "label ≠ content."

Magic Squares

These puzzles require arranging numbers so the rows, columns, and diagonals all sum to the same value. We used Z3 to model the constraints using integer variables and arithmetic relationships.

Tradeoffs in Modeling Tools

Each formal verification tool we used had strengths and weaknesses. Alloy was most effective for modeling dynamic systems over time, while Z3 was stronger for arithmetic-heavy or propositional logic problems. In general:

- **Z3** was ideal for puzzles involving numbers, constraints, and logic-based deductions.
- **Alloy** was preferable for puzzles requiring states, transitions, or actions unfolding over steps.

Model Limitations

To keep computation tractable, we made some simplifying assumptions:

- **River crossing**: We allowed up to 100 steps in Alloy to ensure bounded analysis wouldn't miss a valid trace.
- **Lying puzzle**: Limited puzzle size to 3–5 characters, each with one statement.
- **Age puzzle**: Constrained the sum to be between 2 and 50 and the product between 1 and 100. Solver attempts are capped at 100 iterations.
- **Magic squares**: Limited user-generated squares to a maximum size of 5x5 and limited the size of the magic squares for users to solve to 4x4 due to increasing solve times.

Project Evolution

Our original plan included only three puzzles. As we progressed, we added the mislabeled boxes and magic square puzzles to enrich the experience. Initially, we tried to model the river crossing in Z3 but found step-wise modeling too difficult. Switching to Alloy made it much more intuitive, and this pivot helped us better understand when to use each tool.

From this experience, we learned:

- **Z3** is better for solving static logical or arithmetic constraints.
- **Alloy** excels in modeling dynamic systems with sequential states and constraints over time.

Failed Attempts and Insights

We attempted but did not finalize models for two additional puzzles:

- **Water Jug Puzzle**: Initially thought Alloy would help due to its temporal modeling, but arithmetic operations like subtraction and bounded integers made it awkward. This puzzle would be better suited for Z3.

- **Blue-Eyed Islander Puzzle**: While Z3 can encode the "leave on day *k*" rule directly, it doesn't capture the recursive reasoning process that makes the puzzle interesting. Modeling evolving beliefs and nested knowledge proved too complex

Acknowledgements

During development, we used generative AI as a coding assistant—particularly in drafting early HTML, CSS, and JavaScript files for the interface. These outputs served as initial scaffolding, which we modified heavily to match our vision. All puzzle logic, ideas, and models were our own. The story, interactive flow, styling, and game progression were iteratively designed and tested by us.

Additional Information not included in our Read Me

Did your goals change at all from your proposal? Did you realize anything you planned was unrealistic, or that anything you thought was unrealistic was doable?

Initially, we considered incorporating NLP, such as allowing users to input a custom riddle and translating it into Z3 logic to check for solvability. However, we ultimately decided to focus more on the website aspect of our proposal. We spent additional time integrating all the individual components into a cohesive storyline and ensuring our Python script could generate random variations of the problems to be displayed dynamically. Early on, we also shifted from using Z3 to Alloy for the river crossing problem. This pivot changed our goals, as generating random variations of the river crossing puzzle proved too time-consuming, and integrating Alloy with JavaScript presented additional challenges within our time constraints.

How should we understand an instance of your model, if applicable?

A valid instance of our model is different in every one of our logic puzzles. In the river crossing model in Alloy, an instance will be a completed sequence of steps that gets everyone from one side to the other side without violating any of the constraints. In the lying and truth teller puzzle in Z3, an instance is one in which we are able to correctly assign who is a knight and who is a knave without contradicting any of the statements given. In the age riddle, a SAT instance will give us the set of ages that satisfy the sum and product rules put into the solver. For the mislabeled boxes, an instance will be one where the boxes have the right labels on them after

peeking at one of the objects. The satisfying instance of the magic square will be the instance that gives the numbers that have the same sum in all rows, columns, and diagonals.