Sudoku Helper

Ryan Rice (ryanrice@stanford.edu) & Matt Roche (mtroche@stanford.edu)

For our final project, we decided to build an applet that would help a user solve a Sudoku puzzle. As a user fills out the sudoku board, our program is able to offer hints regarding where the user should mark next and alert the user of any current errors that may be on the board.

We built the applet using worksheets and Epilog. While this project could be completed by other programming methods, logic programming is the best choice for the project because, at its core, solving Sudoku puzzles is an exercise in constraint satisfaction, which logic programming is well equipped for.

We considered and implemented several approaches to the problem of helping a user solve an arbitrary Sudoku puzzle:

1. First, we discussed the approach of enumerating the rules of Sudoku as a CSP and solving them with a logic program as we did for the zebra puzzle. However, this approach is not very intellectually insightful, because it does not reflect how a human would solve the game. We implemented this approach as a baseline, and found that it was prohibitively slow for an interactive applet.
2. The second approach we implemented was domain tracking with recursive updates. In other words, we kept track of the domains of possible values that each cell could hold (starting with the values 1 through 9), and removed values from each cell's domain if another cell in the same row, column, or box took on that value. Then, if we saw that the domain had been refined to size 1, we would automatically mark that cell with the only value left, and recursively update the other cells according to this change. This approach

is not quite as powerful as the first (eg, it cannot complete a puzzle that requires guessing), but it was noticeably more time-efficient. With that being said, it did not always produce results immediately, and still lacked true interactivity because it essentially solved most puzzles before providing any feedback to the user.

3. To solve these problems, we finally pursued domain tracking with depth-1 updates. This approach is similar to the previous in that it keeps track of current possible values for cells, but it only updates the domains of immediately affected cells when one is marked. This is favorable because it runs faster and is more interactive: the recursive permeation would often nearly solve the puzzle before offering hints, whereas depth-1 updates allow for natural progression with the user. Additionally, it is not any less powerful than the second approach, so long as the user is intelligent enough to abide by the hints.

Lastly, our minimalist UI is designed for ease of usability. We chose to watermark cells with their current domains to help the user solve the puzzle without being intrusive, and we allowed the user to toggle on or off hints whenever they wanted. We chose to alert the user of errors by marking the numbers in inconsistent cells as red, which we felt was a fairly intuitive design.

Course Feedback

We felt that the class was a solid introduction to the logic programming paradigm, and that the progression from databases to rules to transitions states (etc.) was pretty natural. However, the transition from Datalog to Prolog and from Prolog to Epilog was not as smooth for me because it was not immediately as obvious what features we were allowed to use on the homeworks when there were time overlaps with later units.