Solving the Klotski Puzzle in OCaml

by

Douglas Lewit of Northeastern Illinois University;

Master’s Project with Peter Kimmel, Jesse Tov, and Graciela Perera.

Abstract: Klotski is a sliding block puzzle with ten wooden blocks of different sizes to slide around on a game board, the goal being to move the largest block to the board’s exit, the exit being traditionally located at the bottom center of the board.

Introduction:

OCaml is a statically, strongly typed functional-first programming language whose developers wanted to address some of the shortcomings of Standard ML, the latter usually being regarded as the original prototype upon which OCaml was initially developed. In other words, one could argue that OCaml could be renamed Standard ML++ or SML++ for short! I got the idea for this project when I participated in an online MOOC (Massive Open Online Course) in OCaml in 2016.

My program solves the game by trying every possible move in a breadth-first search, stopping at the first winning position. (The winning position is determined by the “final” function, a Boolean function that tests for the winning configuration of the board.) Because my algorithm examines all positions reachable in n or fewer moves before examining the solution composed of n+1 moves, my algorithm is not only guaranteed to find a solution if one exists; the solution is also guaranteed to be a shortest solution. This is really a variation of Dijkstra’s algorithm for finding a shortest path in a graph.

The project is divided into two distinct parts: Part A and Part B. Part A involves writing the code to implement a generic solver for any type of game or puzzle while Part B defines the data structures and functions needed to use Part A for the solution of the Klotski Puzzle. Both parts make heavy use of arrays and lists (which are definitely not the same data structure in OCaml) and also sets as well as numerous recursive functions. (While sets CAN be represented as arrays or lists without repeated elements, I instead used OCaml’s built-in Set data structure, which is really implemented as a binary tree—although the details of that implementation is hidden from the programmer who is using the Set data type.)

There were many challenges along the way, and unfortunately my program does not seem to be very efficient. My program is able to solve a simplified version of the Klotski Puzzle (i.e. it has 9 blocks instead of the standard 10) in about 3 minutes on an Apple/Macintosh laptop from 2009, but the solution to the original Klotski puzzle goes on apparently indefinitely. After many attempts to profile the program, after minimizing my append operations, and after working hard to ensure that all my recursive functions are tail-recursive rather than forward-recursive, I am still experiencing the same issues. I am at a loss to explain this problem in my program, but I am still thankful that my program is able to solve simplified versions of the original puzzle.