Magnetic Tower of Hanoi in ACL2

Sean Kelly and Dylan Huang

Northeastern University, Boston, MA

https://github.com/huangdylan08/mtoh-solver

Introduction

Similar to Latin Square puzzles, which were the focus of our previous project, the Tower of Hanoi puzzle is an easy-to-understand mathematical game that has a rich logical and algorithmic underpinning. The Tower of Hanoi puzzle works in the following way: there are three rods, and a given number of disks are initially stacked on one of the rods. The disks are stacked in such a way that each disk is smaller than the disks below it. The goal of the game is to move the stack of disks from the starting peg to one of the other two pegs while maintaining the invariant that no disk is placed on top of a disk that is larger than it.

As has been proven before, the Tower of Hanoi puzzle is solvable for any number of disks, *n*, in $2^n - 1$ moves (Leighton 1-6). Additionally, the algorithm to solve the Tower of Hanoi puzzle has been implemented in ACL2 with an associated proof that the puzzle can be solved in $2^n - 1$ moves (Young).

However, like Latin Square puzzles, there are variations of the Tower of Hanoi puzzle that introduce additional restraints and complexities. One such variation is called the *Magnetic* Tower of Hanoi puzzle that was coined by Uri Levy in 2010. As the name suggests, each disk in the Magnetic Tower of Hanoi puzzle is magnetic and thus has a north and south pole. Like the traditional Tower of Hanoi puzzle, the Magnetic Tower of Hanoi puzzle starts off with a stack of *n* disks on one of three pegs, with the constraint that no disk can be placed on a disk that is smaller than it.

Additionally, in the Magnetic Tower of Hanoi puzzle, each disk must be flipped when it is moved, and the player has to ensure that no two south poles or two north poles are touching. This is because if two south poles or two north poles were to face each other, the force of magnetism would push the two disks away. So, in addition to maintaining correct size order, the Magnetic Tower of Hanoi puzzle must be solved while obeying the laws of magnetism.

This project has a two-fold goal. The first goal is to implement a robust Magnetic Tower of Hanoi algorithm in ACL2, and the second goal is to prove that the algorithm uses $\frac{3^n-1}{2}$ moves, which is the accepted lower bound on the number of moves needed to solve the Magnetic Tower of Hanoi puzzle for *n* disks (Levy 1). The first goal of this project will be accomplished by using a Magnetic Tower of Hanoi algorithm devised by Levy, and the second goal of this project will be accomplished through a proof on the algorithm's corresponding measure function.

Overall Structure

Our project consists of two sections: the first being the code for the Magnetic Tower of Hanoi solver which prints out all the moves needed to solve an *n*-disk game, and the second section comprising of our proof that our solution for the Magnetic Tower of Hanoi will use $\frac{3^n-1}{2}$ moves. Our Magnetic Tower of Hanoi solver calls a function called solver which, by using a boolean flag, recurs between two patterns to generate the solution for our puzzle. Solutions to Magnetic Tower of Hanoi can consist of many different pairs or groups of patterns, but in our solution, we will use the algorithms developed by Levy called "The RBB1000 / RRB1000 Optimal Algorithms" (Levy 12). These two algorithms involve multiple recursive calls to themselves as well as to the other function to create the full solution for the puzzle. In our program we denote the RBB1000 algorithm as the North-South-South pattern, and the RRB1000 algorithm as the North-North-South pattern. Our overall Magnetic Tower of Hanoi function begins with using the North-South-South pattern which we indicate by passing in *t* for the boolean flag input. Then in the second section of our project, our goal is to prove that our solver uses $\frac{3^n-1}{2}$ moves for the full solution.

Walkthrough

Our proof begins by creating a function called solver-moves which acts both as a measure function for solver but also counts the number of move function calls. The function mirrors the form of the solver function except instead of appending the recursive calls, we add the recursive calls and replace the move call with a 1, which is added to the total number of calls. Then our goal is to create a theorem which shows that solver-moves is equal to $\frac{3^n-1}{2}$ for any given input *n*. However when ACL2 attempts to prove this theorem, it requires a sub-lemmata to be proven. That sub-lemmata that we want to prove is that solver-moves returns the same value for a given *n* regardless of which boolean flag is given, meaning that whichever pattern is used, NSS or NNS, the same amount of moves are used. With a theorem for this sub-lemmata created, our overall theorem passes and we therefore have indirectly proved that our Magnetic Tower of Hanoi solver makes $\frac{3^n-1}{2}$ moves.

Personal Progress

Throughout this project, we had two main sources of inspiration that paved the way for us to implement and make proofs about the Magnetic Tower of Hanoi. The first source of inspiration was Young's implementation of a Tower of Hanoi algorithm in ACL2, and the second source was Levy's research and algorithms for the Magnetic Tower of Hanoi. With the work of these two individuals under our belt, we sought to draw from their work to implement Levy's algorithm in ACL2.

Initially, we followed Levy's Magnetic Tower of Hanoi algorithm very strictly, line-by-line. In his algorithm, there exists two mutually recursive functions called RBB and RRB, so we went ahead and used mutual recursion in ACL2 to implement Levy's algorithm using two functions called NSS and NNS, which are analogous to RBB and RRB. Afterwards, we ran some manual test cases, and all seemed to work fine. However, when we attempted to develop generalized theorems, we ran into a dilemma: since the two functions were defined in terms of each other, we could not make a proof about one of the functions without making an additional claim about its counterpart. As a result, our thms for NSS and NNS ended up being several hundred characters long and were very difficult to understand and to debug.

Ultimately, to reduce the complexity of our thms, we chose an alternate approach that combines the NSS and NNS functions into one function that has an additional boolean flag as a variable. If the flag is true, then the NNS pattern is performed, otherwise the NSS pattern is performed.

Switching to an all-in-one approach not only eased the process of debugging our code and proving theorems, but it also reduced the overall number of proofs that we needed. For example, whereas we had individual measure functions for NSS and NNS, an all-in-one function enables us to use a singular measure function, meaning that we need fewer theorems relating to functions and their measures.

Conclusion

As a result of the work we have done thus far, there are a few corollaries that are closer to being solved. The main proof that our work advances is a proof for the *correctness* of the Magnetic Tower of Hanoi algorithm that we used. Although we implemented Levy's RBB1000 / RRB1000 Optimal algorithm and tested how many moves it generates, we did not prove that the moves it generates are the right moves. Ultimately, we did not choose to prove the correctness of the algorithm because we felt it would be *too* big of an undertaking for this project. Nevertheless, we think it would be an interesting next step.

In order to make the jump from our proof to a correctness proof, three lemmas are needed. The first lemma would be a proof that our algorithm maintains the disk size invariant (i.e. a larger disk cannot be placed on a smaller disk). The second lemma would be a proof that our algorithm maintains the disk polarity invariant (i.e. no repelling magnetic forces). Finally, the third lemma would be a proof that our algorithm successfully moves the *n* disks from one peg to another. With these three lemmas, it would be fair to make the claim that our algorithm successfully solves the Magnetic Tower of Hanoi problem.

That being said, in this project we also made significant progress towards the correctness proof by developing a refined implementation of Levy's Magnetic Tower of Hanoi Algorithm as well as by proving the number of steps the algorithm produces for a given number of disks *n*. In the future, we hope that the work that we did in this project will serve as a baseline for future explorers in the same way that Young's Tower of Hanoi implementation in ACL2 and Levy's Magnetic Tower of Hanoi algorithm served as a baseline for us.

Works Cited

Leighton, Tom, and Ronitt Rubinfeld. “Recurrences I.” *Massachusetts Institute of Technology*, Massachusetts Institute of Technology, 24 Oct. 2006, web.mit.edu/neboat/Public/6.042/recurrences1.pdf.

Young, Bill. “TUTORIAL1-TOWERS-OF-HANOI.” *The University of Texas at Austin: Computer Science*, The University of Texas, www.cs.utexas.edu/users/moore/acl2/v6-1/TUTORIAL1-TOWERS-OF-HANOI.html.

Levy, Uri. “The Magnetic Tower of Hanoi and their Optimal Solutions .” ArXiv, Cornell University, 5 Aug. 2010, https://arxiv.org/pdf/1011.3843.pdf.

Appendix

```
; Magnetic Tower of Hanoi Solver and Proof by: Dylan Huang and Sean Kelly
; Function to create a list describing
; a move made from peg 'x' to 'y'
(defun move (x y)
 (declare (xargs :guard t))
 (list 'move 'from 'peg x 'to 'peg y))
#|
NOTE: The 'start', 'inter', and 'dest' inputs for the following functions
     act as tags for each of the three pegs, with the 'start' and 'dest' tagged pegs
     indicating from where and to where we are moving the n amount of disks.
|#
(set-termination-method :measure)
(set-well-founded-relation n<)
(set-defunc-typed-undef nil)
(set-defunc-generalize-contract-thm nil)
(set-gag-mode nil)

#|
Looking at the recursive formula of solver, we create a measure function
solver-moves which mimics solver in form, and instead of appending the outputs,
adds the number of calls to move.
    pattern - boolean flag to indicate which pattern is being used
    start   - tag for the starting peg
    inter   - tag for the intermediate peg
    dest    - tag for the destination peg
    n       - number of total disk that all begin
              stacked on the starting peg
|#
(defun solver-moves (pattern start inter dest n)
 (declare (irrelevant start inter dest))
 (if (zp n)
   0
   (if pattern
     (+ (solver-moves pattern start dest inter (1- n))
        1
        (solver-moves (not pattern) inter dest start (1- n))
        (solver-moves pattern start inter dest (1- n)))
     (+ (solver-moves pattern start inter dest (1- n))
        (solver-moves (not pattern) dest start inter (1- n))
        1
        (solver-moves pattern inter start dest (1- n))))))
```

```
#|
For our solver, when (equal pattern t) refers to the North-South-South Pattern
and the (equal pattern nil) refers to the North-North-South Pattern.
    pattern - boolean flag to indicate which pattern is being used
    start - tag for the starting peg
    inter - tag for the intermediate peg
    dest  - tag for the destination peg
    n     - number of total disk that all begin
            stacked on the starting peg
|#

(defun solver (pattern start inter dest n)
 (declare (xargs :measure (solver-moves pattern start inter dest n)
                 :guard (natp n)))
 (if (zp n)
   nil
   (if pattern
     (append (solver pattern start dest inter (1- n))
             (cons (move start dest)
                   (append (solver (not pattern) inter dest start (1- n))
                           (solver pattern start inter dest (1- n)))))
     (append (solver pattern start inter dest (1- n))
             (append (solver (not pattern) dest start inter (1- n))
                     (cons (move start dest)
                           (solver pattern inter start dest (1- n))))))))

#|
Main function to solve a MToH puzzle
  start - tag for the starting peg
  inter - tag for the intermediate peg
  dest  - tag for the destination peg
  n     - number of total disks that all begin
          stacked on the starting peg
|#
(defun mtoh (start inter dest n)
 (if (zp n)
   nil
   (solver t start inter dest n)))
```

```
#|
Number of moves for n amount of disks:
1 disk : 1 : 3^0
2 disks : 4 : 3^1 + 1
3 disks : 13 : 3^2 + 3^1 + 3^0
4 disks : 40 : 3^3 + 3^2 + 3^1 + 3^0
5 disks : 121 : 3^4 + 3^3 + 3^2 + 3^1 + 3^0
n disks : (3^n - 1)/2
|#

; Proves that starting at either pattern will result in the same number of moves
; alo proves that there is no difference in the number of moves whether you
; use the NSS or NNS pattern.
(defthm solver-moves-equal
 (implies (natp n)
          (equal (solver-moves t 'a 'a 'a n)
                 (solver-moves nil 'a 'a 'a n))))

; Proves that our solver-moves method returns (3^n - 1) / 2 moves for n starting disks
(defthm num-moves-solver
 (implies (natp n)
          (equal (solver-moves t 'a 'a 'a n)
                 (/ (1- (expt 3 n)) 2))))
```