The Towers of Hanoi with 4 pegs

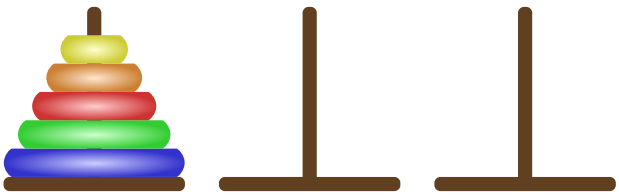
Laurent Théry
Laurent.Thery@sophia.inria.fr

Abstract

This notes presents a formalisation of the towers of Hanoi with 4 pegs done in the Coq proof assistant using the SSReflect extension.

1 Introduction

The towers of Hanoi is a famous problem in computer science. It is often used to teach recursion. The problem is composed of three pegs and some disks of different size. Here is a drawing of the initial configuration for 5 disks[1]:



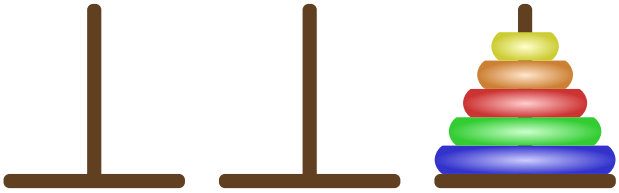
Initially, all the disks are pilled-up in decreasing order of size on the first peg. The goal is to move them all to another peg. There are two rules. First, only one disk can be moved at a time. Second, a larger disk can never be put on top of a smaller one.

Writing a program that solves this problem can easily be done using recursion : one builds the program $P_{3,n+1}$ that solves the problem for $n+1$ disks using the program $P_{3,n}$ that solves the problem for n disks. Let us suppose we want to transfer the $n+1$ disks to the last peg. We first call $P_{3,n}$ to move the n-top disks to the intermediate peg.

[1] We use macros designed by Martin Hofmann and Berteun Damman for our drawings.

Then we move the largest disk to its destination.

Finally, we use $P_{3,n}$ to move the n disks on the intermediate peg to their destination.



This simple recursive algorithm is also optimal: it produces the minimal numbers of moves. In particular, if we look at each recursion depth, the key idea is that largest disk always moves once from its current peg to its destination and there are two recurive calls. This gives $|P_{3,n}| = 2^n - 1$ moves to solve the problem with n disks.

This problem and its generalised version where one wants to move from an arbitrary position to another arbitray one has already been formalised. See [4] for more information. What we are interested here is the problem where one peg is added.

Here we can use a more elaborate strategy for building the $P_{4,n}$ program. This is the so-called Frame-Stewart algorithm. We choose an arbitrary k smaller than n and use $P_{4,k}$ to move the top k-disks to an intermediate peg.

The remaining $n-k$ disks can now freely move except on this intermediate peg, so we can use $P_{3,n-k}$ to move them to their destination.

and reuse $P_{4,k}$ to move the k top disks to their destination.

We can use the k parameter to minimize the number of moves, so $|P_{4,n}| = \min_{k<n} 2|P_{4,k}| + (2^{n-k} - 1)$. Note that this strategy can be generalised to an arbitrary numbers of m pegs,

leading to $|P_{m,n}| = \min_{k<n} 2|P_{m,k}| + |P_{m-1,n-k}|$. If we take $m = 3$, we get back the usual algorithm as $|P_{2,n-k}|$ is a valid program only if $k = n - 1$.

Knowing if this general program is optimal is an open question. What we have formalised is the proof given in [1] that shows it is optimal for $m = 4$. The proof is rather technical. As a matter of fact this technicality was a motivation of our formalisation. The proof is very well written and very convincing but contains several cases. A formalisation ensures that no detail has been overlooked. As an anecdote, we had started a formalisation of [3]. Stuck on the formalisation of Corollary 3, we have contacted the author that told us that there was a flaw in the proof as documented in [2].

Here, we will only highlight the overall structure of the proof. For the details we refer to [1]. The first step is to relate $|P_{4,n}|$ with triangular numbers. Following [1], we write $|P_{4,n}|$ as $\Phi(n)$. We introduce the notation Δn for the sum of the n^{th} first natural numbers :

$$\Delta n = \sum_{i \leq n} i = \frac{n(n+1)}{2}.$$

A number n is triangular if it is a Δi for some i. By analogy to the square root, we introduce the triangular root ∇n :

$$\Delta(\nabla n) \leq n < \Delta(\nabla(n+1)).$$

Now, we can give explicit formula for $\Phi(n)$:

$$\Phi(n) = \sum_{i<n} 2^{\Delta i}$$

The key ingredient of the proof is a valuation function Ψ that takes a finite set over the natural numbers and returns a natural number:

$$\Psi E = \max_{L \in \mathbb{N}}((1-L)2^L - 1 + \sum_{n \in E} 2^{\min(\nabla n, L)})$$

The idea is that E will represent a set of disks on a particular peg. If we consider the set $[n]$ of all the natural numbers smaller than n (i.e. all the disks smaller than n are on the peg)

$$[n] = \{\text{set } i \mid i < n\}$$

we get the relation between Φ and Ψ:

$$\Psi[n] = \frac{\Phi(n+1) - 1}{2} = \frac{\sum_{i<n} 2^{\nabla(i+1)}}{2}$$

Now we can present the central theorem. We consider two configurations u and v of n disks on 4 pegs. We denote $d(u, v)$ the minimum number of moves to go from configuration u to configuration v. If v is such that there is no disk on pegs p_0 and p_1 and E is defined as

$$E = \{\text{set } i \mid \text{the disk } i \text{ is on the peg } p_0 \text{ in } u\}$$

we have:

$$d(u, v) \geq \Psi E$$

The proof proceeds by strong induction of the number of disks. It examinates a path p from u to v that realizes the minimal number of moves. If p_2 is the peg where the disk n is in v (it cannot be p_0 nor p_1), it considers T the largest disk that was initially on the peg p_0 and visits at least one time the peg p_3. If such a disk does not exist, the inequality easily holds. Then, it considers on the path p the configuration x_0 before which the disk T leaves the peg p_0 for the first time and the configuration x_3 in which the disk T reaches the peg p_3 for the first time. Similarly, it considers the configuration z_0 before which the disk n leaves the peg p_0 for the first time and the configuration z_2 before which the disk n reaches the peg p_2 for the last time. Examining the respective positions of x_0, x_3, z_0 and z_2 and applying some surgery on the configurations of the path p in order to apply the inductive hypothesis, it is shown that in every cases the inequality holds.

2 The formalisation

In this section, we present the different elements of our formalisation, starting with pegs, disks, configurations and moves, then we describe the main operations that has made the formalisation possible.

2.1 Pegs

The type $\mathtt{I}_n$ of natural numbers strictly smaller than n is used to represent our pegs.

```
Definition peg k := I_k.
```

In the proof we mostly use elements of *peg* 4 but in some cases, we use *peg* 3 in order to consider the restriction to the problem with 3 pegs.

Similarly, a disk is represented as an element of $\mathtt{I}_n$.

```
Definition disk n := I_n.
```

The comparison of the respective size of two disks is simply performed by comparing the natural numbers they represent.

2.2 Configurations

Disks are ordered from the largest to the smallest on a peg. This means that a configuration just needs to record which disk is on which peg. It is then defined as a finite function from disks to pegs.

```
Definition configuration k n := {ffun disk n → peg k}.
```

Note that in this encoding, we do not have invalid configurations.

A configuration is called *perfect* if all its disks are on a single peg. It is the case for the initial and final configurations. The constant function that always returns p is then the perfect configuration where all the disks are on the peg p.

```
Definition perfect p := [ffun d ⇒ p].
```

2.3 Moves

A move is defined as a relation between configuration. It is parameterised by a relation r on the pegs: $r\ p_1\ p_2$ indicates that it is possible to go from peg p_1 to peg p_1. Assumption will be added on the relation (such as irreflexivity or symmetry) to prove some basic properties of moves.

```
Definition on_top (d : disk n) (c : configuration k n) :=
    [forall d1 : disk n, c d == c d1 ==> d ≤ d1].

Definition move : rel (configuration k n) :=
    [rel c1 c2 |
      [exists d1 : disk n,
        [&& r (c1 d1) (c2 d1), [forall d2, d1 != d2 ==> c1 d2 == c2 d2],
            on_top d1 c1 & on_top d1 c2]]].
```

We first define the predicate *on_top* that checks that the disk d is on top of its peg in the configuration c. The definition of *move* simply states that there is a disk d_1 that fulfills 4 conditions. The move of d_1 from its peg on c_1 to its peg on c_2 is compatible with r. It is the unique disk that has moved. The disk d_1 is in on top of its peg in c_1 and in c_2.

2.4 Distance

What we have defined previously is a relation over configurations. As configurations belong to a finite type, we can benefit from the element of graph theory that are present in the Mathematical Library. For example, (*rgraph move c*) returns the set of all the configurations that are reachable from c in one move, (*connect move* c_1 c_2) indicates that c_1 and c_2 can be connected through moves, or (*path move c cs*) gives that the sequence *cs* of configurations is a path that connects c with the last element of *cs*.

Unfornately, there is no built-in notion of distance, so we have to define one. For this, we first build recursively the function *connectn* that computes the set of elements that are connected to x in n steps for the relation r.

```
Fixpoint connectn r n x :=
  if n is n1.+1 then bigcup_(y in (rgraph r x)) connectn r n1 y
  else [set x].
```

Now, we can define the distance between two points x and y as the smallest n such as (*connectn r n x y*).

```
Definition gdist r x y := find (fun n => y \in connectn r n x) (iota 0 #|T|).
```

Here, T is the finite type corresponding to the relation r and ($iota$ 0 #$|T|$) produces all the natural numbers smaller than the cardinality of the finite set. Note that the distance is a total function and because of the definition of *find*, the distance between two elements that are not connected is #$|T|$. A notation is added so *gdist* r x y can be written as d[x, y]_r.

Finally, we introduce the notion of geodesic path: a path that realises the distance.

```
Definition gpath r x y p :=
    [&& path r x p, last x p == y & d[x, y]_r == size p].
```

Companion theorems are derived for these basic notions. For example, the following lemma shows that concatenation behaves well with geodesic paths for distances.

```
Lemma gdist_cat r x y p1 p2 :
    gpath r x y (p1 ++ p2) → d[x, y]_r = d[x, last x p1]_r + d[last x p1, y]_r.
```

2.5 Configuration surgery

The proof we have formalised uses strong induction and takes a geodesic path and consider the path resulting in taking various subsets of discs. These operations need to be reflected in the formal proof. We have defined a set of such operations. The first one and maybe the more natural one is the one that uses the type to drive the subset selection : if a configuation is of type (*configuration* k $(m+n)$), it allows to see it as a (*configuration* k m) taking the m^{th} largest disks or as a (*configuration* k m) taking the n^{th} smallest disks. This is done as follows:

```
Lemma tlshift_subproof n (i : I_m) : i + n < m + n.
Lemma trshift_subproof m (i : I_n) : i < m + n.
Definition tlshift n (i : I_m) : I_(m + n) := Ordinal (tlshift_subproof n i).
Definition trshift m (i : I_n) : I_(m + n) := Ordinal (trshift_subproof m i).
Definition clshift (c : configuration k (m + n)) : configuration k m :=
  [ffun i => c (tlshift n i)].
Definition crshift (c : configuration k (m + n)) : configuration k n :=
  [ffun i => c (trshift m i)].
```

Which one takes the largest disks and which one the smallest disks has been decided in such way that taking the *crshift* of a configuration c of type (*configuration* k n`.+1`) leads to a configuration of type (*configuration* k n) where the largest disk has been removed. A dedicated notation $\downarrow$`[`c`]` is associated with this operation. Now that we have these operations on configuration, we need to lift them to path. For this, we need an operation on sequences that removes repetion.

```
Fixpoint rm_rep (A : eqType) (a : A) (s : seq A) :=
  if n is b :: s₁ then
    if a == b then rm_rep b s₁ else b :: rm_rep b s₁
  else [::]
```

It is then possible to derive properties on path:

```
Lemma path_clshift (c : configuration (m + n)) cs :
    path move c cs →
    path move (clshift c) (rm_rep (clshift c) [seq (clshift i) | i <- cs]).
Lemma path_crshift (c : configuration (m + n)) cs :
    path move c cs →
    path move (crshift c) (rm_rep (crshift c) [seq (crshift i) | i <- cs]).
```

Other kinds of surgery have been defined:

```
Definition ccut (C : c ≤ n) (c : configuration k n) : configuration k c.
Definition ctuc (C : c ≤ n) (c : configuration k n) : configuration k (n − c).
Definition cset (s : {set (disk n)}) (c : configuration k n) : configuration k #|s|.
Definition cset2 (sp : {set (peg k)}) (sd : {set (disk n)})
                 (c : configuration k n) : configuration #|sp| #|sd|.
```

ccut and *ctuc* are the equivalent of *crshift* and *clshift* but directed by the proof C rather than the type. *cset* considers a subset of disks but with the same number of peg. *cset2* is the most general and considers both a subset of disks and a subset of pegs.

3 Final Proof

The six-page long proof of the main theorem 2.9 in [1] translates to a 1000-line long Coq proof.

```
Lemma gdist_le_psi (u v : configuration 4 n) (p0 p2 p3 : peg 4) :
  [∧ p3 != p2, p3 != p0 & p2 != p0] →
  (codom v) \subset [:: p2 ; p3] →
  Ψ [set i | u i == p0] ≤ d[u, v]_move.
```

From which, we easily derive the expected theorem

```
Lemma gdist_phi (n : nat) (p1 p2 : peg 4) :
  p1 != p2 → d[perfect p1, perfect p2]_move = Φ n.
```

4 Conclusion

We have presented a formalisation of the towers of Hanoi with 4 pegs. It proposes a formal translation of an interesting and non-trivial result. The entire development is available at https://github.com/thery/hanoi.

References

[1] Thierry Bousch. La quatrième tour de Hanoï. *Bull. Belg. Math. Soc. Simon Stevin*, 21(5):895–912, 2014.

[2] Thierry Bousch, Andreas M. Hinz, Sandi Klavzar, Daniele Parisse, Ciril Petr, and Paul K. Stockmeyer. A note on the frame-stewart conjecture. *Discrete Math., Alg. and Appl.*, 11(4), 2019.

[3] Roberto Demontis. What is the least number of moves needed to solve the k-peg towers of hanoi problem? *Discrete Math., Alg. and Appl.*, 11(1), 2019.

[4] Laurent Théry. A Formalisation of the Generalised Towers of Hanoi. January 2017.