Pensieve header: October 16: The Towers of Hanoi Puzzle.

Today. The Towers of Hanoi, then whatever you may suggest, then EIWL 9-12, then, if time, Patterns.

Topics (in no particular order). Whatever you may suggest; whatever comes to my mind; ~~the Fibonacci numbers~~; the Catalan numbers; ~~the Jones polynomial~~; ~~a more efficient Jones algorithm~~; ~~a riddle on spheres~~; Khovanov homology; Γ-calculus; the Hopf fibration; Hilbert's 13th problem; non-commutative Gaussian elimination; free Lie algebras; the Baker-Campbell-Hausdorff formula; wacky numbers; an order 4 torus; the Schwarz Lantern; knot colourings; the Temperley-Lieb pairing; the dodecahedral link; sound experiments; barycentric subdivisions; a Peano curve; braid closures and Vogel's algorithm; the insolubility of the quintic; phase portraits; the Mandelbrot set; shadows of the Cantor aerogel; quilt plots; some image transformations; De Bruijn graphs; the Riemann series theorem; finite type invariants and the Willerton fish; the Towers of Hanoi; Hochschild homology of (some) coalgebras; convolutions and image improvements.

An Image Manipulation Challenge

The image at http://drorbn.net/bbs/show?shot=17-1750-171013-121553.jpg is pathetic. Can you improve it? Whatever you do, should also work well with all other images at http://drorbn.net/bbs/show.php?prefix=17-1750.

The Towers of Hanoi

```
move[1, a_, b_, c_] := Print["Move #1 from ", a, " to ", c];
move[n_, a_, b_, c_] := (
  move[n - 1, a, c, b];
  Print["Move #", n, " from ", a, " to ", c];
  move[n - 1, b, a, c]
 )
```

```
move[3, A, B, C]
```

```
Move #1 from A to C

Move #2 from A to B

Move #1 from C to B

Move #3 from A to C

Move #1 from B to A

Move #2 from B to C

Move #1 from A to C
```

```
move[0, a_, b_, c_] := Null;
move[n_, a_, b_, c_] := (
  move[n - 1, a, c, b];
  Print["Move #", n, " from ", a, " to ", c];
  move[n - 1, b, a, c]
 )
```

```
move[3, A, B, C]
```

```
Move #1 from A to C
Move #2 from A to B
Move #1 from C to B
Move #3 from A to C
Move #1 from B to A
Move #2 from B to C
Move #1 from A to C
```

```
? /;
```

patt /; *test* is a pattern which matches only if the evaluation of *test* yields True.
lhs :> *rhs* /; *test* represents a rule which applies only if the evaluation of *test* yields True.
lhs := *rhs* /; *test* is a definition to be used only if *test* yields True. >>

```
Range[10] /. {k_Integer /; k < 7 :> k^2}
```

```
{1, 4, 9, 16, 25, 36, 7, 8, 9, 10}
```

```
move1[n_, a_, b_, c_] /; n ≥ 1 := (
  move1[n - 1, a, c, b];
  Print["Move #", n, " from ", a, " to ", c];
  move1[n - 1, b, a, c];
 )
```

```
move1[3, A, B, C]
```

```
Move #1 from A to C
Move #2 from A to B
Move #1 from C to B
Move #3 from A to C
Move #1 from B to A
Move #2 from B to C
Move #1 from A to C
```

```
NotebookGet[EvaluationNotebook[]];
```

```
? Sow
```

Sow[*e*] specifies that *e* should be collected by the nearest enclosing Reap.
Sow[*e*, *tag*] specifies that *e* should be
    collected by the nearest enclosing Reap whose pattern matches *tag*.
Sow[*e*, {tag_1, tag_2, ...}] specifies that *e* should be collected
    once for each pattern that matches a tag_i. >>

```
? Reap
```

```
Reap[expr] gives the value of expr together with all expressions to
   which Sow has been applied during its evaluation. Expressions sown using
   Sow[e] or Sow[e, tag_i] with different tags are given in different lists.
Reap[expr, patt] reaps only expressions sown with tags that match patt.
Reap[expr, {patt_1, patt_2, ...}] puts
   expressions associated with each of the patt_i in a separate list.
Reap[expr, patt, f] returns {expr, {f[tag_1, {e_11, e_12, ...}], ...}}.  >>
```

```
Reap[1 + 1; Sow[7]; 2 + 2; Sow[3 + 3]; 5 + 5]
```

```
{10, {{7, 6}}}
```

```
move2[n_, a_, b_, c_] /; n ≥ 1 := (
  move2[n - 1, a, c, b];
  Sow@{n, a, c};
  move2[n - 1, b, a, c];
 )
```

```
Reap[move2[3, A, B, C]]
```

```
{Null, {{{1, A, C}, {2, A, B}, {1, C, B}, {3, A, C}, {1, B, A}, {2, B, C}, {1, A, C}}}}
```

```
move3[n_, a_, b_, c_] /; n ≥ 1 := (
   move3[n - 1, a, c, b];
   Sow@{n, a, c};
   move3[n - 1, b, a, c];
  );
move3[n_] := Reap[move3[n, A, B, C]][[2, 1]]
```

```
move3[4]
```

```
{{1, A, B}, {2, A, C}, {1, B, C}, {3, A, B}, {1, C, A}, {2, C, B}, {1, A, B},
 {4, A, C}, {1, B, C}, {2, B, A}, {1, C, A}, {3, B, C}, {1, A, B}, {2, A, C}, {1, B, C}}
```

```
move4[n_, a_, b_, c_] /; n ≥ 1 := (
   move4[n - 1, a, c, b];
   AppendTo[moves, {n, a, c}];
   move4[n - 1, b, a, c];
  );
move4[n_] := (moves = {}; move4[n, A, B, C]; moves)
```

```
move4[3]
```

```
{{1, A, C}, {2, A, B}, {1, C, B}, {3, A, C}, {1, B, A}, {2, B, C}, {1, A, C}}
```

```
move5[n_, a_, b_, c_] /; n ≥ 1 := (
   move5[n - 1, a, c, b];
   state[[a]] = Most[state[[a]]];
   AppendTo[state[[c]], n];
   Sow@state;
   move5[n - 1, b, a, c];
  );
move5[n_] := (
  state = {Tower @@ Reverse[Range[n]], Tower[], Tower[]};
  Reap[Sow[state]; move5[n, 1, 2, 3]][[2, 1]]
 )
```

```
move5[5]
```

```
{{Tower[5, 4, 3, 2, 1], Tower[], Tower[]}, {Tower[5, 4, 3, 2], Tower[], Tower[1]},
 {Tower[5, 4, 3], Tower[2], Tower[1]}, {Tower[5, 4, 3], Tower[2, 1], Tower[]},
 {Tower[5, 4], Tower[2, 1], Tower[3]}, {Tower[5, 4, 1], Tower[2], Tower[3]},
 {Tower[5, 4, 1], Tower[], Tower[3, 2]}, {Tower[5, 4], Tower[], Tower[3, 2, 1]},
 {Tower[5], Tower[4], Tower[3, 2, 1]}, {Tower[5], Tower[4, 1], Tower[3, 2]},
 {Tower[5, 2], Tower[4, 1], Tower[3]}, {Tower[5, 2, 1], Tower[4], Tower[3]},
 {Tower[5, 2, 1], Tower[4, 3], Tower[]}, {Tower[5, 2], Tower[4, 3], Tower[1]},
 {Tower[5], Tower[4, 3, 2], Tower[1]}, {Tower[5], Tower[4, 3, 2, 1], Tower[]},
 {Tower[], Tower[4, 3, 2, 1], Tower[5]}, {Tower[1], Tower[4, 3, 2], Tower[5]},
 {Tower[1], Tower[4, 3], Tower[5, 2]}, {Tower[], Tower[4, 3], Tower[5, 2, 1]},
 {Tower[3], Tower[4], Tower[5, 2, 1]}, {Tower[3], Tower[4, 1], Tower[5, 2]},
 {Tower[3, 2], Tower[4, 1], Tower[5]}, {Tower[3, 2, 1], Tower[4], Tower[5]},
 {Tower[3, 2, 1], Tower[], Tower[5, 4]}, {Tower[3, 2], Tower[], Tower[5, 4, 1]},
 {Tower[3], Tower[2], Tower[5, 4, 1]}, {Tower[3], Tower[2, 1], Tower[5, 4]},
 {Tower[], Tower[2, 1], Tower[5, 4, 3]}, {Tower[1], Tower[2], Tower[5, 4, 3]},
 {Tower[1], Tower[], Tower[5, 4, 3, 2]}, {Tower[], Tower[], Tower[5, 4, 3, 2, 1]}}
```

```
Reverse[{1, 2, 3}]
```

```
{3, 2, 1}
```

```
? Append
```

```
Append[expr, elem] gives expr with elem appended.
Append[elem] represents an operator form of Append that can be applied to an expression. >>
```

```
rev[{}] = {};
rev[l_List] := Append[rev[Rest[l]], First[l]]
```

```
rev[{"A", 4, π}]
```

```
{π, 4, A}
```

```
rev2[{}] = {};
rev2[l_List] /; l =!= {} := Append[rev2[Rest[l]], First[l]]
```

```
rev2[Range[5]]
```

```
{5, 4, 3, 2, 1}
```

```
rev3[{}] = {};
rev3[{f_, r___}] := Append[rev3[{r}], f]
```

```
rev3[{1, 2, 3}]
```

```
{3, 2, 1}
```