1 Ornaments

1.1 Numerical representations as ornaments

We could vigorously recompute a bunch of datastructures from their numerical representation, but we note that these computations proceed with roughly the same pattern: each constructor of the numeral system gets assigned a value, and is then replaced by a constructor which holds elements and subnodes using this value as a "weight". But wait! The "modification of constructors" is already made formal by the concept of ornamentation!

fix citations

Ornamentation, exposed in [algebraic ornaments; progorn], lets us formulate what it means for two types to have "similar" recursive structure. This is achieved by interpreting (indexed inductive) datatypes from descriptions, between which an ornament is seen as a certificate of similarity, describing which fields or indices need to be introduced or dropped. Furthermore, a one-sided ornament: an ornamental description, lets us describe new datatypes by recording the modifications to an existing description.

This links back to the construction in the previous section, since Nat and Vec share the same recursive structure, so Vec can be formed by introducing indices and adding a field A at each node.

We can already tell that attempting the same for trees and binary numbers fails: they have very different recursive structures! Still, the correct tree constructors relate to those of binary numbers via the size of the resultant tree. In fact, this relation is regular enough that we can "fold in" trees into a structure which *can* be described as an ornament on binary numbers.

1.2 Folding in

fix citations

Let us describe this procedure of folding a complex recursive structure into a simpler structure more generally, and relate this to the construction of binary heaps in [progorn].