Perhaps the conclusion from the last section was not very thrilling, especially considering that $\mathbb{N}$ is a candidate to be replaced by a more suitable unsigned integer type when compiling to Haskell anyway. More relevant to the average Haskell programmer are containers, and their associated laws.

As an example in the same vein as the last section, we could define a type of inefficient lists, and then define a type of more efficient trees. We can show the two to be equivalent again, so that if we show that lists trivially satisfy a set of laws, then trees will satisfy them as well. But even before that, let us reconsider the concept of containers, and inspect why trees are more efficient than lists to begin with.

Rather than defining inductively defining a container and then showing that it is represented by a lookup function, we can go the other way and define a type by insisting that it is equivalent to such a function. This approach, in particular the case in which one calculates a container with the same shape as a numeral system was dubbed numerical representations in [purely functional], and is formalized in [calcdata]. Numerical representations form the starting point for defining more complex datastructures based off of simpler basic structures, so let us run through an example. fix citations

0.1 Vectors from Peano

decide on consistently using Peano or N

We can compute the type of vectors starting from the Peano naturals [this is worked out in full detail in calcdata]. For simplicity, we define them as a type computing function via the "use-as-definition" notation from before. Recall that we expect $VAn = Finn- > A$, so we should define $Finn$ first. In turn $Finn = \Sigma[m \in N]m < n$. fix citation

[SIP doesn't mesh very well with indexed stuff] We can some basic operations [lookup/tail] and show some properties. Again, we can transport these proofs to vectors. fix the inline code fix this code

(This computation can of course be generalized to any arity zeroless numeral system; unfortunately beyond this set of base types, this "straightforward" computation from numeral system to container loses its efficacy. In a sense, the n-ary natural numbers are exactly the base types for which the required steps are convenient type equivalences like $(A + B)- > C = A- > C \times B- > C$?) define a "gap" type of comment do this