9 Proof by induction

The laws we use in equational reasoning about programs have to be justified. Because we are dealing with lists, which are a recursive data type, functions on lists are often defined by recursion, so proofs about them they will usually be justified by induction.

Think about

```
> pow :: Num a => a -> Int -> a
> pow x 0 = 1
> pow x n = pow x (n-1) * x
```

How would we prove

$$pow\ x\ (m+n) \quad = \quad pow\ x\ m \times pow\ x\ n$$

Being honest we would really want the n to be a natural number, and

```
> pow x 0         = 1
> pow x n | n > 0 = pow x (n-1) * x
```

is closer to it. In writing mathematics, however, one would be more likely to say

```
pow x  0    = 1
pow x (n+1) = pow x n * x
```

with the observation that since $n \varepsilon \mathbb{N}$, the $n+1$ cannot be zero. Older textbooks may have Haskell definitions that look like this, but the syntactic form was misleading and has been removed from the language.

9.1 Induction over natural numbers

A proof of $P(n)$ for every natural number n is an infinite number of proofs, one for $P(0)$, one for $P(1)$, one for $P(2)$, ... Clearly providing all of those will take a long time. A better alternative would be to provide a systematic way of generating any one of those proofs.

Every natural number is either 0, or is $n+1$ for some natural number n. To prove $P(n)$ for every natural number n it is enough to prove

1. $P(0)$
2. for every natural number n, if $P(n)$ then $P(n+1)$.

Since any natural number n can be made of a 0 and some number of applications of $(+1)$, the proof of $P(n)$ can be built out of a proof of $P(0)$ and some (the same) number of proofs of $P(0) \Rightarrow P(1)$, $P(1) \Rightarrow P(2)$, $\ldots P(n-1) \Rightarrow P(n)$. So a proof of

$$pow\ x\ (m+n) \quad = \quad pow\ x\ m \times pow\ x\ n$$

by induction on n would consist to two parts: one for $n = 0$,

$$\begin{array}{ll} & pow\ x\ (m+0) \\ = & \{\,\text{unit of addition}\,\} \\ & pow\ x\ m \\ = & \{\,\text{unit of multiplication}\,\} \\ & pow\ x\ m \times 1 \\ = & \{\,\text{definition of } pow\,\} \\ & pow\ x\ m \times pow\ x\ 0 \end{array}$$

and assuming, temporarily, $pow\ x\ (m+n) = pow\ x\ m \times pow\ x\ n$ for some n,

$$\begin{array}{ll} & pow\ x\ (m+(n+1)) \\ = & \{\,\text{definition of } (+) \text{ (perhaps associativity of } (+))\,\} \\ & pow\ x\ ((m+n)+1) \\ = & \{\,\text{definition of } pow\,\} \\ & pow\ x\ (m+n) \times x \\ = & \{\,\text{induction hypothesis}\,\} \\ & (pow\ x\ m \times pow\ x\ n) \times x \\ = & \{\,\text{associativity of multiplication}\,\} \\ & pow\ x\ m \times (pow\ x\ n \times x) \\ = & \{\,\text{definition of } pow\,\} \\ & pow\ x\ m \times pow\ x\ (n+1) \end{array}$$

Then we can discharge the assumption and read this as a proof of

$$\begin{array}{l} \text{if } pow\ x\ (m+n) = pow\ x\ m \times pow\ x\ n \text{ then} \\ \qquad pow\ x\ (m+(n+1)) = pow\ x\ m \times pow\ x\ (n+1) \end{array}$$

for every natural number n.

9.2 The take lemma

Sometimes proofs about lists can be reduced to proofs about natural numbers. The *take lemma* is the observation that two lists xs and ys are equal provided

that $take\ n\ xs = take\ n\ ys$ for all natural numbers n. That requirement can often be proved by induction on n, noting that the case $n = 0$ always holds because both sides are null.

For example, $map\ f\ (iterate\ f\ x) = iterate\ f\ (f\ x)$ because

$$
\begin{array}{ll}
 & take\ (n+1)\ (map\ f\ (iterate\ f\ x)) \\
= & \{\text{definition of } iterate\} \\
 & take\ (n+1)\ (map\ f\ (x : iterate\ f\ (f\ x))) \\
= & \{\text{definition of } map\} \\
 & take\ (n+1)\ (f\ x : map\ f\ (iterate\ f\ (f\ x))) \\
= & \{\text{definition of } take\} \\
 & f\ x : take\ n\ (map\ f\ (iterate\ f\ (f\ x))) \\
= & \{\text{induction hypothesis}\} \\
 & f\ x : take\ n\ (iterate\ f\ (f\ (f\ x))) \\
= & \{\text{definition of } take\} \\
 & take\ (n+1)\ (f\ x : iterate\ f\ (f\ (f\ x))) \\
= & \{\text{definition of } iterate\} \\
 & take\ (n+1)\ (iterate\ f\ (f\ x)))
\end{array}
$$

Notice that a proof about $n + 1$ and x requires a hypothesis about n and $f\ x$. Formally, the induction hypothesis is that *for all* x the result holds for n.

9.3 Induction over finite lists

Every finite list is either $[\,]$, or is $x : xs$ for some finite list xs. To prove $P(xs)$ for every finite list xs it is enough to prove

1. $P([\,])$
2. for every finite list xs, if $P(xs)$ then $P(x : xs)$.

Recall that

```
> (++) :: [a] -> [a] -> [a]
> []     ++ ys = ys
> (x:xs) ++ ys = x:(xs++ys)
```

We prove that $(xs +\!\!+ ys) +\!\!+ zs = xs +\!\!+ (ys +\!\!+ zs)$ for all finite lists xs by induction on xs. Firstly for empty lists

$$\begin{array}{ll} & ([] +\!\!+ ys) +\!\!+ zs \\ = & \{\,\text{definition of } (+\!\!+)\,\} \\ & ys +\!\!+ zs \\ = & \{\,\text{definition of } (+\!\!+)\,\} \\ & [] +\!\!+ (ys +\!\!+ zs) \end{array}$$

then assuming $(xs +\!\!+ ys) +\!\!+ zs = xs +\!\!+ (ys +\!\!+ zs)$

$$\begin{array}{ll} & ((x : xs) +\!\!+ ys) +\!\!+ zs \\ = & \{\,\text{definition of } (+\!\!+)\,\} \\ & (x : (xs +\!\!+ ys)) +\!\!+ zs \\ = & \{\,\text{definition of } (+\!\!+)\,\} \\ & x : ((xs +\!\!+ ys) +\!\!+ zs) \\ = & \{\,\text{induction hypothesis}\,\} \\ & x : (xs +\!\!+ (ys +\!\!+ zs)) \\ = & \{\,\text{definition of } (+\!\!+)\,\} \\ & (x : xs) +\!\!+ (ys +\!\!+ zs) \end{array}$$

so the result is true for all finite lists xs (and all lists ys and zs).

9.4 A second example

Given the definition

```
> reverse    []  = []
> reverse (x:xs) = reverse xs ++ [x]
```

a proof of $reverse\ (reverse\ xs) = xs$ for all finite lists xs proceeds by induction on xs.

$$\begin{array}{ll} & reverse\ (reverse\ []) \\ = & \{\,\text{definition of } reverse\,\} \\ & reverse\ [] \\ = & \{\,\text{definition of } reverse\,\} \\ & [] \end{array}$$

and then assuming $reverse\ (reverse\ xs) = xs$ for some xs

$$
\begin{array}{ll}
 & reverse\ (reverse\ (x : xs)) \\
= & \{\,\text{definition of } reverse\,\} \\
 & reverse\ (reverse\ xs +\!\!+ [x]) \\
= & \{\,\text{unjustified step}\,\} \\
 & x : reverse\ (reverse\ xs) \\
= & \{\,\text{inductive hypothesis}\,\} \\
 & x : xs
\end{array}
$$

The (as yet) unjustified step requires a lemma:

$$reverse\ (ys +\!\!+ [x]) = x : reverse\ ys$$

which can be proved by induction on ys.

$$
\begin{array}{ll}
 & reverse\ ([\,] +\!\!+ [x]) \\
= & \{\,\text{definition of } (+\!\!+)\,\} \\
 & reverse\ [x] \\
= & \{\,\text{definition of } reverse\,\} \\
 & reverse\ [\,] +\!\!+ [x] \\
= & \{\,\text{definition of } reverse\,\} \\
 & [\,] +\!\!+ [x] \\
= & \{\,\text{definition of } (+\!\!+)\,\} \\
 & [x] \\
= & \{\,\text{definition of } reverse\,\} \\
 & x : reverse\ [\,]
\end{array}
$$

and assuming it to be true for ys

$$
\begin{array}{ll}
 & reverse\ ((y : ys) +\!\!+ [x]) \\
= & \{\,\text{definition of } (+\!\!+)\,\} \\
 & reverse\ (y : (ys +\!\!+ [x])) \\
= & \{\,\text{definition of } reverse\,\} \\
 & reverse\ (ys +\!\!+ [x]) +\!\!+ [y] \\
= & \{\,\text{inductive hypothesis}\,\} \\
 & (x : reverse\ ys) +\!\!+ [y] \\
= & \{\,\text{definition of } (+\!\!+)\,\} \\
 & x : (reverse\ ys +\!\!+ [y]) \\
= & \{\,\text{definition of } reverse\,\} \\
 & x : reverse\ (y : ys)
\end{array}
$$

9.5 Induction over partial lists

A partial list is one with a tail that is either $\bot$ or a smaller partial list. There is a similar principle of induction over partial lists; to prove $P(xs)$ for every partial list xs it is enough to prove

1. $P(\bot)$
2. for every partial list xs, if $P(xs)$ then $P(x : xs)$.

For example $xs +\!\!+ ys = xs$ for all partial lists xs.

The case $\bot +\!\!+ ys = \bot$ is immediate from the strictness of $(+\!\!+)$, since it is defined by pattern matching on its left argument. Then assuming the result for some partial xs

$$
\begin{aligned}
& (x : xs) +\!\!+ ys \\
= \quad & \{ \text{definition of } (+\!\!+) \} \\
& x : (xs +\!\!+ ys) \\
= \quad & \{ \text{inductive hypothesis} \} \\
& x : xs
\end{aligned}
$$

9.6 Infinite lists

An infinite list can be thought of as a limit of a sequence of partial lists, for example $[0..]$ is the limit of the chain $\bot$, $0 : \bot$, $0 : 1 : \bot$, $0 : 1 : 2 : \bot$, ... and so on. Successive elements of this chain are related by the information ordering, $(\sqsubseteq)$. This is the partial order that has $\bot \sqsubseteq x$ for all x, and $(x : xs) \sqsubseteq (y : ys)$ whenever $x \sqsubseteq y$ and $xs \sqsubseteq ys$.

A property P is called chain complete if whenever $P(xs_i)$ holds for every element xs_i of a chain it also holds for the limit of that chain. So, chain complete properties that also hold for all partial lists will be ones that also hold for infinite lists.

It is enough to know that positive properties are chain complete. These include universally quantified equations between Haskell-definable expressions, and conjunctions ('and's) of positive properties. Inequalities need not be chain complete, nor need properties involving existential quantification be. For example "$drop\ n\ xs = \bot$ for some n" is true for all partial lists, but is not true for infinite lists. It is not one equation, but the disjunction ('or') of an infinite number of equations.

The earlier proof of $(xs +\!\!+ ys) +\!\!+ zs = xs +\!\!+ (ys +\!\!+ zs)$ for finite lists can be extended to partial lists xs, because $(\bot +\!\!+ ys) +\!\!+ zs = \bot +\!\!+ zs = \bot$ (by strictness of $(+\!\!+)$) so the result holds for all finite and partial lists, and since it is an equation between Haskell expressions it is chain complete and also holds for infinite lists. (Informally, both sides are equal to xs if xs is an infinite list.)

Can we extend the proof of $reverse\ (reverse\ xs) = xs$ to infinite lists? Certainly it holds for bottom: $reverse\ (reverse\ \bot) = reverse\ \bot = \bot$ (again, by strictness of *reverse*). This looks promising; but running `reverse (reverse [0..])` will produce no output: $reverse\ (reverse\ xs) = \bot$ for all partial xs.

What went wrong? The proof that

$$reverse\ (reverse\ xs) = xs \Rightarrow reverse\ (reverse\ (x : xs)) = x : xs$$

required a lemma $reverse\ (ys +\!\!+ [x]) = x : reverse\ ys$ which we only proved for finite lists ys. It is not true for infinite lists, nor indeed for partial lists, because $reverse\ (\bot +\!\!+ [x]) = \bot$ and so $reverse\ (ys +\!\!+ [x]) = \bot$ for all partial lists ys.

9.7 Aside: Continuity (beyond the scope of this course)

This use of the words *limit* and *continuity* turn out to be exactly the same as you might meet in analysis.

You can define a distance function $d :: ([a], [a]) \to \mathbb{R}_{\geqslant 0}$, a *metric*, which behaves like the distance between points in space. In particular, the triangle inequality $d(xs, ys) + d(ys, zs) \geqslant d(xs, zs)$ holds.

A suitable distance between xs and ys might be 2^{-n} for the biggest n for which the $take\ n\ xs = take\ n\ ys$. The successive elements of the sequence of partial approximations to an infinite list are arbitrarily close together, just like Cauchy sequences in $\mathbb{R}$. The infinite list is just the limit of that Cauchy sequence.

Continuity for real valued functions turns out to be equivalent to preserving limits. A function has an abrupt discontinuity exactly when you can find a convergent sequence leading up to a point of discontinuity, so that the image of the limit under the function is not the limit of the images of the elements of the sequence.

Computable functions turn out to have to be not only monotonic (put in more information into an argument, they cannot retract any information that has already come out in the result). More than that they need to be continuous (apply a continuous function to a limit of a chain, and you get back the limit of the images of the sequence).

Monotonicity tells you that none of the information in the image of of the limit of a sequence can contradict anything you might have learned from the image of an element of that sequence. Continuity tells you that all of the information in the image of the limit of the sequence will already have come out in the image of one of the elements of the sequence.

That is, if f is a computable function, and xs is an infinte list, then it is OK to compute $f\ xs$ by successively working out what you can know about $f\ xs$ by calculating $f\ (take\ n\ xs +\!\!+ \bot)$ for successively bigger n.

Chain completeness also turns out to be continuity, and continuity guarantees soundness. A proof can be thought of as a function returning one of two values,

with $\mathit{Invalid} \sqsubseteq \mathit{Valid}$. (You can think of it as $\underline{\mathit{False}} \sqsubseteq \underline{\mathit{True}}$, but do not confuse it with the usual information ordering on Bool.) A proof which is valid for all (except perhaps a finite number) of the elements of a chain cannot then make an abrupt change from valid to invalid at the infinite limit of the chain. Mathematical equality is continuous, so a mathematical equality of two Haskell expressions is the composition of continuous functions.