This short note describes some of the structure of the domain \( \text{LIST}(A) \), which should help with fixed-point problems on lists. Like every domain, the domain \( \text{LIST}(A) \) has a bottom element, 
\[
\bot_\text{LIST}(A).
\]

We’ll call this the “undefined list.”

The domain structure has as refinement ordering, which I’ll write 
\[
x \sqsubseteq y
\]
and pronounce “\( x \) refines to \( y \)” or “\( y \) is at least as defined as \( x \).” Refinement is a complete partial order:

- It is reflexive, transitive, and antisymmetric.
- Every infinite ascending chain has a least upper bound.

“Ascending” here really means “monotone.” The elements in the sequence don’t have to get more defined; they just have to stay at least as defined.

For example, here are some trivial infinite ascending chains:
\[
\bot_\text{LIST}(A) \sqsubseteq \bot_\text{LIST}(A) \sqsubseteq \cdots \sqsubseteq \bot_\text{LIST}(A) \sqsubseteq \cdots
\]
\[
\bot_\text{LIST}(A) \sqsubseteq \bot_\text{LIST}(A) \sqsubseteq \cdots \sqsubseteq \bot_\text{LIST}(A) \sqsubseteq \text{nil}_\text{LIST}(A) \sqsubseteq \text{nil}_\text{LIST}(A) \cdots \sqsubseteq \text{nil}_\text{LIST}(A) \cdots
\]

The first chain has a least upper bound of \( \bot_\text{LIST}(A) \); the second has a least upper bound of \( \text{nil}_\text{LIST}(A) \).

These two example chains are trivially ascending because there’s no actual climbing going on: the end of the chain is just an infinite sequence of the same element repeating over and over.

If two elements are related but not equal, I’ll say that one is strictly more defined than the other. For example, in the domain of lists, both nil and cons are strictly more defined than bottom:
\[
\bot_\text{LIST}(A) \sqsubseteq \text{nil}_\text{LIST}(A)
\]
\[
\bot_\text{LIST}(A) \sqsubseteq \text{cons} \sqsubseteq_a \bot_\text{LIST}(A)
\]
These bounds are tight: there is nothing in between \( \bot_{\text{LIST}(A)} \) and \( \text{nil}_{\text{LIST}(A)} \), and there is nothing in between \( \bot_{\text{LIST}(A)} \) and \( \text{cons} \ \bot_A \ \bot_{\text{LIST}(A)} \). (Here \( \bot_A \) is the bottom element of domain \( A \)—the “undefined \( A \)” if you will.)

We’re interested in ascending chains, because it’s the ascending chains that give us the solution to fixed-point equations, like the final problem on the homework. How can we climb higher (“get more defined”) in this partial order?

- There is no value that is strictly more defined than \( \text{nil} \). The empty list \( \text{nil} \) is the end of the road, and it can’t be refined any further. We say that \( \text{nil} \) is totally defined. It’s pretty easy to spot a totally defined value because it doesn’t mention any \( \bot \)’s anywhere.

- By contrast, there are lots of values more defined than \( \text{cons} \ \bot_A \ \bot_{\text{LIST}(A)} \). To find them, we exploit monotonicity.

Function \( \text{cons} \) is monotonic, and so is \( \text{cons} \ x \), for any \( x \). (All functions definable using lambda calculus are monotonic and continuous.) That means

- \( x \sqsubseteq x' \) implies \( \text{cons} \ x \ x s \sqsubseteq \text{cons} \ x' \ x s \)
- \( x s \sqsubseteq x s' \) implies \( \text{cons} \ x \ x s \sqsubseteq \text{cons} \ x \ x s' \)

We can use the refinements above, together with monotonicity, to get two more relations:

\[
\begin{align*}
\text{cons} \ \bot_A \ \bot_{\text{LIST}(A)} & \sqsubseteq \text{cons} \ \bot_A \ \text{nil}_{\text{LIST}(A)} \\
\text{cons} \ \bot_A \ \bot_{\text{LIST}(A)} & \sqsubseteq \bot_A \ \left( \text{cons} \ \bot_A \ \bot_{\text{LIST}(A)} \right)
\end{align*}
\]

At this point I’m going to drop the subscript from \( \bot_{\text{LIST}(A)} \), and I’m going to write \( \text{cons} \) using ML notation, including allowing the infix :: to associate to the right:

\[
\begin{align*}
\bot_A :: \bot & \sqsubseteq \bot_A :: \text{nil} \\
\bot_A :: \bot & \sqsubseteq \bot_A :: \bot
\end{align*}
\]

Here is a longer chain:

\[
\begin{align*}
\bot & \sqsubseteq \bot_A :: \bot \sqsubseteq \bot_A :: \bot \sqsubseteq \bot_A :: \bot_A :: \bot_A :: \bot_A :: \bot_A :: \bot_A :: \bot_A :: \bot_A :: \text{nil}
\end{align*}
\]

By refining the last \( \bot \) to \( \text{nil} \) instead of another cons cell, I limit the amount of further refinement available. Every value that is above the last one is a list of four values drawn from domain \( A \). Domains like integers and Booleans don’t have any infinite ascending chains, so if for example I let \( A = \text{BOOL} \), then there are \( 2^5 \) values that are at least as defined as \([\bot_A, \bot_A, \bot_A, \bot_A, \bot_A] \), and of these values, \( 2^4 \) are totally defined. The totally defined values are exactly the lists of length 4 in which every element is a totally defined Boolean.
It’s quite useful to think about what values are above a given value, because thinking about the possible refinements tells us what we might find on an infinite ascending chain. For example, what values are above

$$⊥_{BOOL} :: true :: ⊥_{LIST(BOOL)}$$

These are “all of the lists of Booleans of length at least 2 whose second element is $true$.” For example, the following relations hold:

$$⊥_{BOOL} :: true :: ⊥_{LIST(BOOL)} \sqsubseteq false :: true :: nil$$
$$⊥_{BOOL} :: true :: ⊥_{LIST(BOOL)} \sqsubseteq false :: true :: false :: nil$$

I’ll wrap up this note with an example of a nontrivial infinite ascending chain. This chain is produced by the fixed-point construction (the method of successive approximations) when we try to solve the recursion equation

$$xs = true :: xs.$$ 

The first step in solving this problem is to convert to a function

$$g = λxs. true :: xs$$

and then to find the least defined fixed point of $g$. We have

$$g(⊥) = true :: ⊥$$
$$g(g(⊥)) = true :: true :: ⊥$$
$$g(g(g(⊥))) = true :: true :: true :: ⊥$$

This sequence of values forms a nontrivial ascending chain. The limit of this chain is the infinite list of Booleans all of whose elements are true. (This limit, which is required to exist by completeness, is what we take as the definition of the informal notion “infinite list.”) It’s relatively easy to prove that no finite list of Booleans is above every element of this chain.