Background Questionnaire

Thomas Lacroix

November 6, 2018

1 Functional programming

```
fun SNOC x [] = [x]
  | SNOC x (y::ys) = y::(SNOC x ys)
```

1.1. What is the result of `SNOC 5 [7,3,2]`?

`SNOC` appends a number to a list.

`-> [7,3,2,5]`

1.2. Describe informally (and very briefly) what the function `SNOC` does.

First, while the list (`y`, second argument) isn't empty, it recursively calls itself without `y`'s head. Then, adds `x` to the empty list and puts back `y`'s content.

2. Write a functional program `APPEND` that appends two lists. Try to use SML notation. `APPEND` should satisfy the following example behaviours:

- `APPEND [1,2,3] [4,5] = [1,2,3,4,5]`
- `APPEND [] [1] = [1]`

```
fun APPEND [] ys = ys
  | APPEND x::xs ys = x::(APPEND xs ys)
```

3. What is tail-recursion and why is it important for functional programming?

It is the ability to recursively call a function an infinite amount of time without allocating memory (stack frame), if the recursive call is the last instruction (the returned value) of the function. It's important because, among else, it enables immutability.

4. Write a tail-recursive version of `APPEND`.

It is, I think.

2 Induction proofs

1. Prove that the following method to calculate the sum of the first n natural numbers is correct (notice $0 \notin \mathbb{N}$):

$$\forall n \in \mathbb{N}, \sum_{1 \leq i \leq n} i = \frac{n \cdot (n+1)}{2} \tag{1}$$

Proof. For $n = 1$, this is immediate. Then, we need to prove that

$$\forall n \in \mathbb{N}, \sum_{1 \leq i \leq n+1} i = \frac{(n+1)\,(n+2)}{2} \tag{2}$$

given (1). Let $n \in \mathbb{N}$:

$$\frac{n\,(n+1)}{2} + (n+1) = (n+1)\left(\frac{n}{2} + 1\right) = (n+1)\,\frac{n+2}{2} = \frac{(n+1)\,(n+2)}{2}$$

□

Prove that ∀x l. `LENGTH (SNOC x 1) = LENGTH l + 1` holds for the function `SNOC` given above. You can use arithmetic facts and the following properties of `LENGTH`:

- `LENGTH [] = 0`
- `∀l x xs. (x::xs) ++ l = x :: (xs ++ l)`

If we use the notation `++` for list concatenation, then we can write `SNOC x l = [x] ++ l`. Then, the property is immediate.

Proof. More formally, we can prove it by induction. With `l` the empty list, it is immediate that `LENGTH (SNOC x []) = LENGTH [x] = 1 = LENGTH 0 + 1`. We'll use `l` containing one element as the base case for this proof. By definition, we immediately have:

$$\texttt{∀x y. SNOC x [y]} \overset{def}{=} \texttt{y::(SNOC x [])} \overset{def}{=} \texttt{y::[x] = [y,x]} \tag{3}$$

whose length is 2.

Then, for the induction step, we have to show (because we've covered the empty `l` case)

$$\texttt{∀x y ys. LENGTH (SNOC x y::ys) = LENGTH y::ys + 1} \tag{4}$$

given

$$\texttt{∀x ys. LENGTH (SNOC x ys) = LENGTH ys + 1} \tag{5}$$

Starting from (4):

$$\begin{aligned}
\texttt{LENGTH (SNOC x y::ys)} &= \texttt{LENGTH (y::(SNOC x ys))} \\
&= \texttt{LENGTH (SNOC x ys) + 1} \\
&= \{ \text{ using (5) } \} \\
&= \texttt{(LENGTH ys + 1) + 1} \\
&= \texttt{LENGTH y::ys + 1}
\end{aligned}$$

□

3. Prove that `APPEND` is associative. You can use the following properties of `APPEND`. Use the notation `l1 ++ l2` for `APPEND l1 l2`.

- $\forall$ `l. [] ++ l = l`
- $\forall$ `x xs. (x :: xs) ++ l = x :: (xs ++ l)`

We have to show that, $\forall$ `l1 l2 l3`:

$$\texttt{(l1 ++ l2) ++ l3 = l1 ++ (l2 ++ l3)} \tag{6}$$

To show that, we can show that the left-hand side is equal to the right-hand side by "transferring" `l2`'s elements from the lhs to the rhs one-by-one. I don't see an easier way.

4. Which induction principles do you know? Which ones did you use above?

Apart from the induction proof, I don't know any other induction principle; or I cannot name them.

3 Logic

1. Explain briefly what's wrong with the following reasoning: "No cat has two tails. A cat has one more tail than no cat. Therefore, a cat has three tails."

The two occurrences of "no cat" don't refer to the same thing, so it's not correct to use them this way. The first one means "it doesn't exist any cat featuring two tails", but the second one isn't valid English (as far as I know) because it refers to an empty set as an element.

2. *Spartans* and *Athenians*

2.a) Formalise the relevant parts of the text above in first order logic. Model *is coward* as *is not brave* and *is moron* as *is not wise*.

Let's use $S(x)$ instead of $Spartan(x)$, and similarly for Athenians and for the philosophers' names.

Then, we can formalize them as follows:

(a) $\forall x, S(x) \Rightarrow \text{brave}(x) \equiv \neg S(x) \vee \text{brave}(x)$

(b) $\forall x, A(x) \Rightarrow \text{wise}(x) \equiv \neg A(x) \vee \text{wise}(x)$

(c) $\forall x, (S(x) \vee \neg A(x)) \vee (\neg S(x) \vee A(x))$

(d) $\neg (S(x) \wedge A(x))$

(e) $S(P) \Rightarrow \neg\text{brave}(D) \equiv \neg S(P) \vee \neg\text{brave}(D)$

(f) $S(E) \Rightarrow \neg\text{brave}(D) \equiv \neg S(E) \vee \neg\text{brave}(D)$

(g) $A(D) \Rightarrow \neg\text{brave}(E) \equiv \neg A(D) \vee \neg\text{brave}(E)$

(h) $A(P) \Rightarrow \neg\text{wise}(E) \equiv \neg A(P) \vee \neg\text{wise}(E)$

With (c) and (d), then (a) or (b), we can simplify as follows:

(e) $\neg S(P) \vee A(D)$

(f) $\neg S(E) \vee A(D)$

(g) $\neg A(D) \vee A(E)$

(h) $\neg A(P) \vee S(E)$

Then, to prove that Platon is from Sparta, we can add $\neg S(P)$ and prove it leads to a contradiction using the resolution rule, as follows:

$$\frac{\neg S(P),\ \neg S(P) \vee A(D),\ \neg S(E) \vee A(D),\ \neg A(D) \vee A(E),\ \neg A(P) \vee S(E)}{\neg S(P),\ \neg S(P) \vee A(D),\ \neg S(E) \vee A(E),\ \neg A(P) \vee S(E)}$$

$$\frac{\neg S(P),\ \neg S(P) \vee A(D),\ \neg S(E),\ \neg A(P) \vee S(E)}{\neg S(P),\ \neg S(P) \vee A(D),\ \neg A(P)}$$

$$\frac{\neg S(P),\ \neg A(P)}{\neg S(P),\ S(P)}$$

3. *myst* function

3.a) Which type does *myst* have?

myst is a predicate, because it is a function that takes two arguments of the same type and returns a boolean.

3.b) What concept does the type $\alpha \to \alpha \to$ *bool* represent?

High-order functions (can I say curryfication?).

3.c) Translate the formula in English using as high level concepts as possible.

Q is a restriction of R which must also be *true* when both its arguments are equals, and its *true* area must be of one block and without holes.

3.d) What concept does the function *myst* define?

It seems to defines some kind of area: it fills the gaps of the "*true* zone" of the given predicate.