HW7 Collaborative

CS40 Fall'21

Benjamin Cruttenden (4672440)
Bharat Kathi (5938444)
Sean Oh (4824231)
Marco Wong (4589198)

Due: Thursday, Nov 18, 2021 at 10:00PM on Gradescope

In this assignment,

You will work with recursively defined sets and functions and prove properties about them, practicing induction and other proof strategies.

For all HW assignments:

Please see the instructions and policies for assignments on the class website and on the writeup for HW1. In particular, these policies address

- Collaboration policy
- Where to get help
- Typing your solutions
- Expectations for full credit

You will submit this assignment via Gradescope (https://www.gradescope.com) in the assignment called "HW7-Collaborative".

In your proofs and disproofs of statements below, justify each step by reference to the proof strategies we have discussed so far, and/or to relevant definitions and calculations. We include only induction-related strategies here; you can and should refer to past material to identify others.

Proof by Structural Induction: To prove that $\forall x \in X\ P(x)$ where X is a recursively defined set, prove two cases:

Basis Step:	Show the statement holds for elements specified in the basis step of the definition.
Recursive Step:	Show that if the statement is true for each of the elements used to construct new elements in the recursive step of the definition, the result holds for these new elements.

RNA related definitions

Consider the following definitions related to RNA strands:

Definition Set of bases $B = \{\texttt{A}, \texttt{C}, \texttt{U}, \texttt{G}\}$. The set of RNA strands S is defined (recursively) by:

$$\begin{array}{ll} \text{Basis Step:} & \texttt{A} \in S, \texttt{C} \in S, \texttt{U} \in S, \texttt{G} \in S \\ \text{Recursive Step:} & \text{If } s \in S \text{ and } b \in B, \text{ then } sb \in S \end{array}$$

where sb is string concatenation.

Definition The function *rnalen* that computes the length of RNA strands in S is defined recursively by $rnalen : S \to \mathbb{Z}^+$

$$\begin{array}{ll} \text{Basis Step:} & \text{If } b \in B, \text{ then } rnalen(b) = 1 \\ \text{Recursive Step:} & \text{If } s \in S \text{ and } b \in B, \text{ then } rnalen(sb) = 1 + rnalen(s) \end{array}$$

Definition The function *basecount* that computes the number of a given base b appearing in a RNA strand s is defined recursively by $basecount : S \times B \to \mathbb{N}$

$$\text{Basis step: If } b_1 \in B,\ b_2 \in B,\ basecount(b_1, b_2) = \begin{cases} 1 & \text{when } b_1 = b_2 \\ 0 & \text{when } b_1 \neq b_2 \end{cases}$$

$$\text{Recursive Step: If } s \in S,\ b_1 \in B,\ b_2 \in B,\ basecount(sb_1, b_2) = \begin{cases} 1 + basecount(s, b_2) & \text{when } b_1 = b_2 \\ basecount(s, b_2) & \text{when } b_1 \neq b_2 \end{cases}$$

Definition The function $mutate : S \times B \to S$ is defined recursively as:

$$\text{Basis step: If } b_1 \in B \text{ and } b_2 \in B,\ mutate(b_1, b_2) = \begin{cases} b_2 & \text{when } b_1 = \texttt{A} \\ b_1 & \text{when } b_1 \neq \texttt{A} \end{cases}$$

$$\text{Recursive Step: If } s \in S,\ b_1 \in B,\ b_2 \in B,\ mutate(sb_1, b_2) = \begin{cases} mutate(s, b_2)b_2 & \text{when } b_1 = \texttt{A} \\ mutate(s, b_2)b_1 & \text{when } b_1 \neq \texttt{A} \end{cases}$$

Linked list related definitions

Definition The set of linked lists of natural numbers L is defined by:

$$\begin{array}{ll} \text{Basis Step:} & [\,] \in L \\ \text{Recursive Step:} & \text{If } l \in L \text{ and } n \in \mathbb{N}\text{, then } (n, l) \in L \end{array}$$

Definition The function *removeTail* : $L \to L$ that removes the last node of a linked list (if it exists) is defined by:

$$\begin{array}{ll} & \mathit{removeTail} : L \to L \\ \text{Basis Step:} & \mathit{removeTail}([\,]) = [\,] \\ \text{Recursive Step:} & \text{If } l \in L \text{ and } n \in \mathbb{N}\text{, then} \\ & \mathit{removeTail}(\ (n, l)\ ) = \begin{cases} [\,], \text{when } l = [\,] \\ (n, \mathit{removeTail}(l)\ ), \text{when } l \neq [\,] \end{cases} \end{array}$$

Definition The function *remove* : $L \times \mathbb{N} \to L$ that removes a single node containing a given value (if present) from a linked list is defined by:

$$\begin{array}{llll} & & \mathit{remove} : L \times \mathbb{N} & \to L \\ \text{Basis Step:} & \text{If } m \in \mathbb{N} \text{ then} & \mathit{remove}([\,], m) & = [\,] \\ \text{Recursive Step:} & \text{If } l \in L, n \in \mathbb{N}, m \in \mathbb{N}\text{, then} & \mathit{remove}((n, l), m) & = \begin{cases} l & \text{when } n = m \\ (n, \mathit{remove}(l, m)) & \text{when } n \neq m \end{cases} \end{array}$$

Definition: The function *prepend* : $L \times \mathbb{N} \to L$ that adds an element at the front of a linked list is defined by:

$$\mathit{prepend}(l, n) = (n, l)$$

Definition The function *append* : $L \times \mathbb{N} \to L$ that adds an element at the end of a linked list is defined by:

$$\begin{array}{llll} & & \mathit{append} : L \times \mathbb{N} & \to L \\ \text{Basis Step:} & \text{If } m \in \mathbb{N} \text{ then} & \mathit{append}([\,], m) & = (m, [\,]) \\ \text{Recursive Step:} & \text{If } l \in L \text{ and } n \in \mathbb{N} \text{ and } m \in \mathbb{N}\text{, then} & \mathit{append}((n, l), m) & = (n, \mathit{append}(l, m)) \end{array}$$

Assigned Questions

Definitions related to RNA and linked list are listed on the previous two pages.

1. (*Graded for correctness*[1]) Calculate the following function applications. Include all intermediate steps, with justifications.

   Sample response that can be used as reference for the detail expected in your answer for this part:

   Calculating *basecount*(AGGGC, G), we have

$$\begin{aligned}
\textit{basecount}(\texttt{AGGGC},\ \texttt{G}) &= \textit{basecount}(\texttt{AGGG}, \texttt{G}) && \text{By recursive step of basecount: } b_1 = \texttt{C},\, b_2 = \texttt{G}\\
&= 1 + \textit{basecount}(\texttt{AGG}, \texttt{G}) && \text{By recursive step of basecount: } b_1 = \texttt{G},\, b_2 = \texttt{G}\\
&= 1 + 1 + \textit{basecount}(\texttt{AG}, \texttt{G}) && \text{By recursive step of basecount: } b_1 = \texttt{G},\, b_2 = \texttt{G}\\
&= 1 + 1 + 1 + \textit{basecount}(\texttt{A}, \texttt{G}) && \text{By recursive step of basecount: } b_1 = \texttt{G},\, b_2 = \texttt{G}\\
&= 1 + 1 + 1 + 0 && \text{By basis step of } \textit{basecount}\text{: } b_1 = \texttt{A}, b_2 = \texttt{G}
\end{aligned}$$

   (a) Calculate *basecount*(*mutate*(AGGGC, C), C)

   Answer: .
   mutate(AGGGC,C) = mutate(AGGG,C)C = mutate(AGG,C)GC = mutate(AG,C)GGC = mutate(A,C)GGGC = CGGGC

   basecount(CGGGC,C) = 1 + basecount(CGGG,C) = 1 + basecount(CGG,C) = 1 + basecount(CG,C) = 1 + basecount(C,C) = 1 + 1 = 2

   (b) Calculate *mutate*(*mutate*(GGUACAA, U), G)

   Answer: .
   mutate(GGUACAA,U) = mutate(GGUACA,U)U = mutate(GGUAC,U)UU = mutate(GGUA,U)C
   = mutate(GGU,U)UCUU = mutate(GG,U)UUCUU = mutate(G,U)GUUCUU = GGUU-CUU

   mutate(GGUUCUU,U) = mutate(GGUUCU,U)U = mutate(GGUUC,U)UU = mutate(GGUU,U)C
   = mutate(GGU,U)UCUU = mutate(GG,U)UUCUU = mutate(G,U)GUUCUU = GGUU-CUU

2. Provide a recursive definition of the function $copy : S \times \mathbb{N} \to S$ that takes as input an RNA strand s and a non-negative integer j and gives as output j copies of s concatenated together. For example, if $s =$ AUUG, then $copy(s, 4) =$ AUUGAUUGAUUGAUUG, and $copy(s, 0) = \lambda$ where λ denotes empty string. Give a recursive definition for $copy(s, j)$.

[1] This means your solution will be evaluated not only on the correctness of your answers, but on your ability to present your ideas clearly and logically. You should explain how you arrived at your conclusions, using mathematically sound reasoning. Whether you use formal proof techniques or write a more informal argument for why something is true, your answers should always be well-supported. Your goal should be to convince the reader that your results and methods are sound.

Answer: .

Definition The function copy : $S \times N \to S$ that takes as input an RNA strand s and a non-negative integer j and gives as output j copies of s concatenated together is defined recursively as:

Basis step: If $s \in S, copy(s, 0) = \emptyset$

Recursive Step: if $s \in S$, and $j \in N$, then $copy(s, j) = (s * copy(s, j - 1))$

3. Prove by structural induction that $\forall s \in S\ \forall t \in S\ (rnalen(st) = (\ rnalen(s) + rnalen(t)\ )$ by using the recursive definition of *rnalen*.

Answer: .

Theorem: $\forall s \in S \forall \in S(rnalen(st) = (rnalen(s) + rnalen(t))$

Proof: by structural induction

Base Case: $s \in B, t \in B(rnalen(st) = (rnalen(s) + rnalen(t))s = A, t = Arnalen(AA) = (rnalen(A) + rnalen(A)$

$LHS = rnalen(AA) = 1 + rnalen(A)$ by recursive step of rnalen
$1 + 1 = 2$ by basis step of rnalen

$RHS = rnalen(A) + rnalen(A) = 1 + 1 = 2$ by basis step of rnalen
LHS = RHS

Since rnalen(A) = rnalen(C) = rnalen(G) = rnalen(U), this base case is proven for all $s \in B, t \in B$

Inductive step: consider arbitary RNA strands $s = eb$ where $e \in S$ and $b \in B$ and strand $t = fg$ where $f \in S$ and $g \in B$ assume as inductive hypothesis that (rnalen(ef) = (rnalen(e) + rnalen(f))

I want to show that (rnalen(ebfg) = (rnalen(eb) + rnalen(fg))
LHS: By the recursive step of rnalen,
rnalen(ebfg) = 2 + rnalen(ef) because b and g are in set B
RHS: By the recursive step of rnalen,
(rnalen(eb) + rnalen(fg)) = (1 + rnalen(e) + 1 + rnalen(f)) because b and g are in set B
This equals 2 + rnalen(e) + rnalen(f) by algebra
Setting LHS = RHS,
2 + rnalen(ef) = 2 + rnalen(e) + rnalen(f)
This equals rnalen(ef) = rnalen(e) + rnalen(f)
This is the Inductive Hypothesis so therefore the proof is true

4. Consider the following alternate recursive definition of *removeTail* : $L \to L$ called *removeTailAlt* : $L \to L$. A correct definition removes the last node of a linked list (if it exists).

$$
\begin{array}{ll}
 & \textit{removeTailAlt} : L \to L \\
\text{Basis Step:} & \textit{removeTailAlt}([\,]) = [\,] \\
\text{Recursive Step:} & \text{If } l \in L, m \in \mathbb{N}, \text{ and } n \in \mathbb{N}, \text{ then} \\
 & \textit{removeTailAlt}(\ (n,(m,l)\ )) = \begin{cases} (n,[\,]), \text{when } l = [\,] \\ (n, removeTailAlt((m,l))\ ), \text{when } l \neq [\,] \end{cases}
\end{array}
$$

Identify the error in the definition of *removeTailAlt* and provide evidence for your answer by showing how the application of the function on a sample input of your choosing does not work.

Answer: .

Test case: removeTailAlt(9,(8,(7,(6,[])))) = (9,removeTailAlt(8,(7,(6,[])))) = (9,(8,removeTailAlt(7,(6,[])
= (9,(8,(7,[])))

removeTailAlt(4,[]) = ??

The error is in the basis step. The recursive step works as intended for —l— ¿2, but there is no case for —l— = 2. The basis step should be:

Basis Step: If $n \in N$ removeTailAlt(n,[]) = [], removeTailAlt([]) = []

5. (*Each part graded for correctness in evaluating statement and for fair effort completeness in the justification)* Statements like these are used to build the specifications for programs, libraries, and data structures (APIs) which spell out the expected behavior of certain functions and methods. In this HW question, you will analyze whether and how order matters for the *remove* and *prepend* functions.

   (a) Prove or disprove the following statement:

$$\forall l \in L\, \forall m \in \mathbb{N}\ (\ remove(\ prepend(l,m),\ m) = l\ ).$$

Answer: .

Theorem: $\forall l \in L \forall m \in N(remove(prepend(l,m),m) = l)$.
Proof by structural induction on $l \in L$
Base Case: l = [], m = 0 so (remove(prepend([], 0), 0) = []).
Evaluating: LHS = remove(prepend([], 0), 0)
prepend([], 0) = (0,[]) by definition of prepend
remove((0,[]),0) = [] by recursive definition of remove
[] = RHS

Inductive step: Consider arbitrary linked list l = (n, l'), n in N, l' in L and m was some arbitrary natural number $m \in N$ we can assume that the IH is remove(prepend(l', m), m) = l').

WTS: $remove(prepend(l,m),m) = l)remove(prepend((n,l'),m),m) = (n,l'))$.
Evaluating:
LHS = remove(prepend((n, l'), m), m) which equals
remove(m, (n, l')), m) by definition of prepend
(n, l') by recursive step of remove
= RHS Since LHS = RHS we know it is true.

(b) Prove or disprove the following statement:

$$\exists l \in L\, \exists m \in \mathbb{N} \ (\ remove(\ prepend(l, m),\ m) = l\ ).$$

Answer: .

$\exists l \in L \exists m \in N(remove(prepend(l, m), m) = l)$

Choose witnesses: l = [], m = 0 so (remove(prepend([], 0), 0) = []).

Evaluating: LHS = remove(prepend([], 0), 0)

prepend([], 0) = (0,[]) by definition of prepend

remove((0,[]),0) = [] by recursive definition of remove

[] $= RHS$

Attributions