Assignment 3

Advanced Computing concepts for Engineering

Due 2018 Feb 7

Note that the work that you turn in for this assignment must represent your individual effort. You are welcome to help your fellow students to understand the material of the course and the meaning of the assignment questions, however, the answer that you submit must be created by you alone.

Q0 Decimals

(a) Use the method of invariants to find an efficient iterative solution to the following problem: Given a sequence of decimal digits in an array, calculate the corresponding natural number. In particular, let n be a variable holding a natural number and, let d be variable, holding a sequence of n natural numbers, implement $\left\langle x' = \sum_{i \in \{0,..n\}} d(i) \times 10^i \right\rangle$. Be sure to clearly state the invariant. [For this part don't worry about having exponentiation in your code.]

(b) If your solution to part (a) requires the exponentiation, introduce a tracking variable so that the algorithm only requires multiplications and additions. How does this tracking variable affect the invariant?

(d) If in parts (a) and (b) you processed the array from right to left, repeat part (a) processing the array from left to right. And conversely.

(e) Repeat part (b) for the algorithm in part (d).

(f) How could you solve this problem without the method of invariants?

Q1 Linked list reversal

Use the method of invariants to find an efficient iterative solution to the problem of reversing a linked list.

We will represent a linked list by two arrays[1] of size n: *data* and *next*. The items of the linked list are in *data* and the links are in *next*. Each item of *next* is an integer. We'll use use -1 to represent the end of the list. We can define a predicate[2] *isList* on integers numbers to be the strongest predicate that satisfies

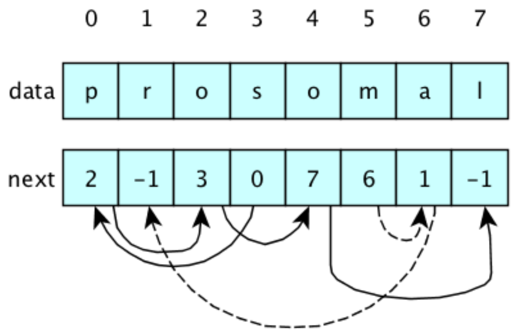
$$\forall i \in \{0,..n\} \cdot \mathit{isList}(\mathit{next}(i)) \Rightarrow \mathit{isList}(i)$$
$$\mathit{isList}(-1)$$

By saying that *isList* is the strongest predicate that satisfies these properties, we mean that the predicate is only true where it must be true to satisfy these

[1] By array, I just mean a variable that holds a sequence.

[2] A predicate on natural numbers is just a function from the natural numbers to the booleans. In this case the predicate is implicitly also a function of a state which defines the value of *next*.

two properties. For example $isList(-2)$ is false, since it does not need to be true to satisfy these two properties. In a state where $next(i) = i$, for some $i \in \{0, ..n\}$, we would have $isList(i) = \mathfrak{false}$, since there would be no way to show $isList(i) = \mathfrak{true}$ using only the properties above. In the state



,

we have $isList(-1)$,and so $isList(1)$, and so $isList(6)$, and so $isList(5)$. In fact, in this state, $\forall i \in \{0, ..8\} \cdot isList(i)$

We can define a partial function *toSeq* that extracts the sequence from a linked list. Like *isList*, *toSeq* is implicitly a function of the state. The *toSeq* function is only defined when *isList* is true. We can define *toSeq* by the equations.

$$toSeq(-1) = [\,]$$
$$\forall i \in \{0, ..n\} \cdot toSeq(i) = [data(i)]\hat{\ } toSeq(next(i))$$

Here $[\,]$ is a sequence of length 0; $[a]$ is a sequence of length 1; and $\hat{\ }$ concatenates two sequences. For example, in the state shown above, we have $toSeq(-1) = [\,]$, and so $toSeq(1) = [\text{`r'}]$, and so $toSeq(6) = [\text{`a'}, \text{`r'}]$, and so $toSeq(6) = [\text{`m'}, \text{`a'}, \text{`r'}]$.

We also need a function that reverses a sequence

$$\begin{array}{rcl} reverse([\,]) & = & [\,] \\ \forall a \cdot reverse([a]\hat{\ }s) & = & reverse(s)\hat{\ }[a] \end{array}$$

This function does not depend on the state. For example, $reverse([\text{`m'}, \text{`a'}, \text{`r'}])$ is $[\text{`r'}, \text{`a'}, \text{`m'}]$.

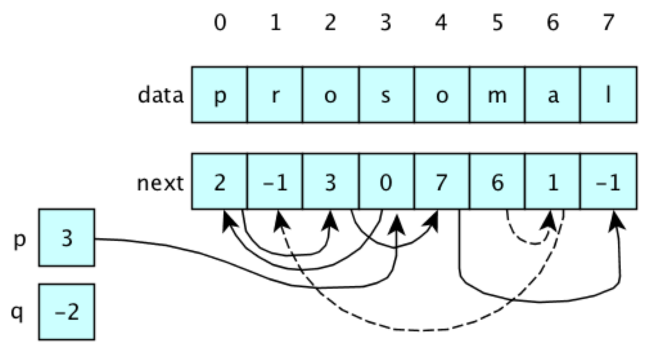
We want to reverse the sequence represented by a list. The specification is[3]

$$\Big\langle isList(p) \wedge (s = toSeq(p)) \Rightarrow isList(q)' \wedge (reverse(toData(q)') = s) \wedge \big(data = data'\big)\Big\rangle$$

Here s is a sequence valued variable that is used for the purpose of specification. You should not refer to s in your program code. For example, if the

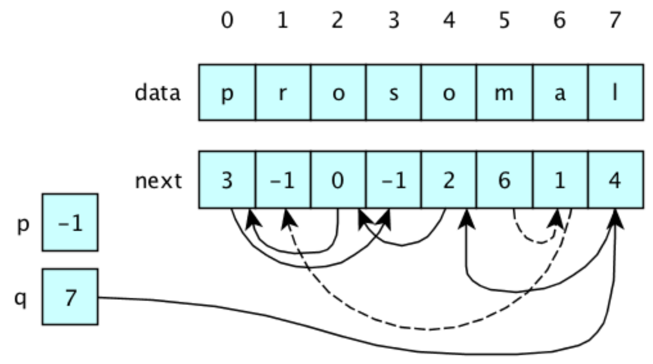
[3]The use of prime on an expression means the value of that expression in the final state. Thus $isList(q)'$ means $isList'(q')$ where $isList'$ is a function just like *isList* except that it uses $next'$ instead of *next*. Similarly for $toData(q)'$.

initial state is



,

an acceptable final state is



(a) What invariant can be used?

(b) What initialization statement can be used to make the invariant true.

(c) What loop guard should be used?

(d) What is the specification h of the loop body?

(e) We haven't covered assignments to individual array items. We can understand an assignment $a(i) := x$ to mean that we assign to a a value that is just like its initial value except that item i is x. For example, if in the initial state a maps to $[5, 9, 13]$ and i is 1, then after executing $a(i) := 8$, the value of a would be $[5, 8, 13]$.

What is a loop body that implements the specification h of the body given in part (d)? [No proof is needed, but you should convince yourself that it refines h.]

(f) What bound can be used to show that the loop terminates?

Q2. Designing a hardware divider

(a) Apply the abstract binary search algorithm given in notes 0-6 to the problem

$$\langle r' = \lfloor x/y \rfloor \rangle$$

of finding a the integer part of a quotient. The goal set G will be $\{\lfloor x/y \rfloor\}$ i.e., $\{z \in \mathbb{N} \mid zy \leq x < (z+1)y\}$. You may assume x is a natural number and y is a positive natural number. Try to eliminate all set operations from the code. [By the way you, may think that finding the average of two numbers involves a division by 2, and it does. But division by 2 followed by rounding, requires only a shift in the binary representation. So, if the numbers are represented in binary, dividing by 2 followed by rounding is far easier and cheaper than dividing by an arbitrary positive natural number.]

(b) If the algorithm from part (a) contains any multiplications (other than by small constants like 2), try to eliminate them by introducing one or more tracking variables.

(c) Draw the data path of a divider circuit based on your algorithm from part (b).

(d) Suppose you represent a set $\{p, ..r\}$ by two numbers p and r, where $r - p$ is a power of 2. You can also represent such a set by two numbers c and i such that $p = c2^i$ and $r = (c+1)\,2^i$.Can you modify your algorithm from part b so that it maintains as an invariant that the size of the search space S is always a power of 2? If so, data transform the algorithm so that represents the set as suggested above.

(e) Draw the data path of a divider circuit based on your algorithm from part (d).