Functional programming in Dafny

Exercise: List Functions  
Will Sonnex and Sophia Drossopoulou

*Aims:* In this exercise the students will practice proving simple inductive properties of recursive functions. The students will also “check” assertions, .e., they will be given some assertion, and they will have to either find a counterexample, or prove the assertion. They will also be asked to write a function which satisfies some given properties.

## 

Start by opening the provided “List.dfy”, Observe the four given function definitions, and the properties we have expressed about them. Make sure this file can be verified by Dafny before proceeding.

datatype list<T> = Nil | Cons(T, list<T>);  
// appends two lists

function app(xs: list, ys: list): list

// the length of the given list

function len (xs: list): nat

// "n" is an element of list "xs"

predicate elem<T> (n: T, xs: list<T>)

// reverses the list “xs”

function rev(xs: list):list

## Question 1

Define a function “del”:

function del<T>(n: T, xs: list<T>): list<T>

{

…

}

such that the following three properties hold:

Write the body of the function del, and then express the three properties from above as ghost methods. Start by giving these methods the empty body, and then write out the Dafny proofs for these methods.

ghost method prop\_del\_notelem<T>(n: T, xs: list<T>)

requires !elem(n, xs);

ensures del(n, xs) == xs;

Hint: Property 1   
can be expressed as:

## Question 2

Modify your definition for “del” to give a function “del1” :

function del1<T>(n: T, xs: list<T>): list<T>

{

…

}

such that the following three properties hold:

As in the first Question, write the body of the function del1, and then express the three properties from above as ghost methods. Start by giving these methods the empty body, and if necessary, write out the Dafny proofs for these methods.

Hint: Compare the three properties from Question1 and Question 2, to find the difference between del and del1.

## Question 3

Read the definition of the function count and the definition of rev. The function count satisfies the following two properties:

Prove in Dafny that count satisfies these properties. For the second property, you will need to use some of the properties proven earlier.

Unfortunately these two properties are also fulfilled by dud\_count, defined as follows:

function dud\_count<T>(n: T, xs: list<T>): nat { 0 }

Prove in Dafny that dud\_count satisfies these properties.

Write a property P which count fulfils but dud\_count does not (don not just restate the definition of count). Verify in Dafny that count satisfies P, and dud\_count does not.

## Question 4

Prove that the following holds for “rev”:

ghost method prop\_rev\_app(xs: list, ys: list)

ensures rev(app(xs, ys)) == app(rev(ys), rev(xs));

You may need to use some of the properties already defined and proven earlier.

## Question 5

Use the property you proved in the previous question to verify the following property of “rev”:

## Question 6

Write a function which does list indexing in Dafny, with the following signature:  
  
  
  
This should return the n-th element of the list. Remember that all functions in Dafny must terminate, so you may have to add pre-conditions.  
  
We now have a “safe” list indexing function, something that is impossible in Haskell (and most other programming languages).

function nth<T>(n: nat, xs: list<T>): T

## Question 7

In this question we will prove equivalence between our existing reverse function, and a faster tail-recursive version which uses accumulators called “itrev”.

Here is the definition of itrev, which you should enter into your Dafny code:

function itrev(xs: list, ys: list): list

{

match xs

case Nil => ys

case Cons(x, xs') => itrev(xs', Cons(x, ys))

}

Prove the following property about itrev in Dafny:

Hint: You will need to define and prove an auxiliary lemma, and you will need to use some of the lemmas we have already proven,

## Sample Answers

In the file List\_SA.dfy.,