Functional programming in Dafny

Exercise: Making Change  
Will Sonnex and Sophia Drossopoulou

These exercises are also about specification and verification of functional programming. They are a bit more demanding than those on lists. In particular, we now look at proofs involving universal and existential quantifiers.

Definitions are in the provided file “Change.dfy”. The subject of this exercise is making change from a given list of units for a given amount. Correct change for a given amount is defined as:

predicate correct\_change(

units: list<int>, amount: nat, change: list<int>)

{

All\_nat(units) && subset(change, units) && sum(change) == amount

}

For example, the following assertions hold:   
 correct\_change( [3,4,8], 16, [4,4,8]) and correct\_change( [3,4,8], 16, [3,3,3,3,4])

But the following assertions do not hold   
 correct\_change( [3,4,8], 16, [3,3,3,3,3]) and correct\_change( [3,4,8], 16, [5,5,3,3]).

(We have used the notation [..,..,…] to indicate lists.)

## Question 0 Preliminaries

Look at the definitions of subsequence and subset. Which of the following properties are true? Give a counter-example for those which are false.



## Question 1

Which of the following properties are true? Give a counter-example for those that are false.

## Question 2

Within the provided “Change.dfy” we have defined a simple algorithm for making change from a given set of units. It does not always terminate, and so Dafny will not compile it. Alter its pre-conditions so that it does always terminate, such that Dafny can prove termination and hence compile the function.

function make\_change(units: list<int>, amount: nat): list<int>

requires all\_nat(units);

Calculate the value of make\_change( [2,3], 6) and the value of make\_change( [2,3], 6).

## Question 3

Which of the following properties are true? Give a counter-example for those that are false.

Any calls to automatically enforce any pre-conditions you added in the previous question. Definitions of functions and predicates can be found in “Change.dfy”.

## Question 4

1. Our simple algorithm “make\_change” will not always produce correct change. Design a simple predicate good\_units for the input list of units such that it does:  
     
     
     
   The following property should therefore be valid (you don’t need to prove this yet…):

predicate good\_units(cs: list<int>)

It should be the case that the following holds: .

1. Check in Dafny that your good\_units predicate ensures change exists for any amount:

ghost method prop\_good\_units\_make\_change(cs: list<int>, n: nat)

requires good\_units(cs);

ensures change\_exists(cs, n);

is defined as .

To construct this existential provide any value for such that holds, viz. show the following for some :

assert correct\_change(cs, n, chg);

You will probably need to prove some auxiliary lemmas.  
 Hint: We have defined some functions for you which we haven’t used yet.

1. Informally explain in words why it is impossible to invent a predicate satisfying the property in a., but which also satisfies:

1. (**Hard**) If your definition for good units is strong enough it will also satisfy:  
     
   State this property in Dafny. The prove it.

## Question 5

Prove the correctness of make\_change in Dafny, i.e. that the property from the previous question holds, by giving a body to prop\_make\_change.

ghost method prop\_make\_change(cs: list<int>, n: nat)

requires good\_units(cs);

ensures correct\_change(cs, n, make\_change(cs, n));