Proving Properties of Imperative Programs

Exercise: Max and Products  
Will Sonnex and Sophia Drossopoulou

In these questions we will prove properties of imperative programs which use arrays and loops.

We will demonstrate how to specify, prove, and use auxiliary lemmas which are used in the proof of the imperative program.

## 1st Question

The function Max is supposed to calculate the maximal number in an array, but does not fulfil its specification. Namely, if you compile, Dafny will give you error messages. Fix the definition, without changing the pre- and post-conditions so that the specification can be verified by Dafny.

method max (a: array<int>) returns (max: nat)

requires a != null && a.Length > 0;

ensures forall j:: 0<=j<a.Length ==> a[j] <= max;

ensures exists j:: 0<=j<a.Length && a[j]==max;

{

max := 0;

var i := 0;

while (i < a.Length)

invariant 0 <= i <= a.Length;

invariant forall j:: 0<=j<i ==> a[j] <= max;

invariant i!=0 ==> exists j:: 0<=j<i && a[j]==max;

invariant i==0 ==> a[0] >= max;

{

if (a[i] > max) {

max := a[i];

}

i := i + 1;

}

}

### Fix the definition, *without changing the pre- and post-conditions* so that the specification can be verified by Dafny.

## 2nd Question

### The following function product(a) returns the product of all numbers in array a.

### 

function product(a: array<int>): int

requires a!=null;

reads a;

{

productAux(a,0,a.Length-1)

}

### It is defined in terms of the auxiliary function productAux(a,from,to), which returns the product of all numbers in array a between (and including) from and to. It calculates the product starting from the rightmost index of the interval [from..to].

function productAux(a: array<int>, from:nat, to:int): int

requires a!=null && to < a.Length;

reads a;

decreases to-from;

{

if ( from > to )

then 1

else if ( from==to )

then a[to]

else productAux(a,from,to-1) \* a[to]

}

### The same mathematical function may be defined in terms of the alternative auxiliary function productAuxAlt(a,from,to), which also returns the product of all numbers in array a between (and including) from and to, but calculates the product starting from the left-most index of the interval [from..to].

function productAuxAlt(a: array<int>, from:nat, to:int): int

requires a!=null && to < a.Length;

reads a;

decreases to-from;

{

if ( from > to )

then 1

else if ( from==to )

then a[to]

else a[from] \* productAuxAlt(a,from+1,to)

}

### **2a. Question**

State and prove a Lemma in Dafny, which ascertains that productAux(a,from,to) returns the same result as productAuxAlt(a,from,to).

### **2b. Question**

Consider the definition of the method Product, which calculates the product of all numbers in an array a, by traversing the array left-to-right.

method Product(a: array<int>) returns (p: int)

requires a != null;

ensures p == product(a);

{

var i := 0;

p := 1;

while (i < a.Length)

invariant ????;

{

p := p \* a[i];

i := i + 1;

}

}

### Prove that the code of Product satisfies its specification.

### **3rd Question**

Consider now an alternative definition, which calculates the product of all number in an array a, by traversing the array right-to-left.

method ProductAlt(a: array<int>) returns (p: int)

requires a != null;

ensures p == product(a);

{

var i := a.Length-1;

p := 1;

while (i >= 0)

invariant ????

{

p := p \* a[i];

i := i - 1;

}

}

Prove that the method above satisfies its specification.

### **Sample Answers**

Sample answers can be found in the file Max\_Product.dfy.