

Verification-Driven Program Development

Exercise Sheet, Week 3

Ștefan Ciobâcă

13.10.2024

Exercises

1. Dafny supports algebraic data types. Consider the following datatype defining a list and a function returning the number of elements in a list:

```
datatype List = Nil | Cons(info : int, rest : List)

function length(l : List) : int
  decreases l
{
  match l {
    case Nil => 0
    case Cons(info, rest) => 1 + length(rest)
  }
}
```

- (a) Write a recursive function to compute the sum of all elements in a list.
 - (b) Write a predicate that checks whether all elements in a list are positive.
2. Dafny has support for *sequences*. Here is a short example of using sequences:

```
function concat(s1 : seq<int>, s2 : seq<int>) : seq<int>
{
  s1 + s2
}

method Main()
{
  var s1 : seq<int> := [ 1, 2, 3 ];
  var s2 : seq<int> := [ 4, 5, 6 ];
  var s3 := concat(s1, s2);
  print s3[2..4]; // [3, 4, 5]
  print s3[4];   // 4
  print |s3|;    // 6
  var s4 := s3[0 := 10]; // update of element on position 0 (inefficient)
  print s4;      // [10, 2, 3, 4, 5, 6]
}
```

- (a) Write a recursive function to compute the sum of all elements in a sequence.
- (b) Write a function to convert a `List` into a `seq<int>`.
- (c) Write a ghost predicate that checks whether a sequence is sorted in increasing order.

3. Here is an example of using sets:

```
function union(s1 : set<int>, s2 : set<int>) : set<int>
{
  s1 + s2
}

predicate belongsTo(x : int, s : set<int>)
{
  x in s
}

method Main()
{
  var s1 := { 1, 2, 3 };
  var s2 := { 2, 3, 4 };
  var s := union(s1, s2);
  assert s == { 1, 2, 3, 4 };
  assert |s| == 4; // cardinality
  var s' := s1 * s2; // intersection
  assert s' == { 2, 3 };
  assert belongsTo(3, s); // membership
  assert 10 !in s; // not membership
  assert s1 < s; // strict subset
  assert s <= s; // subset
  assert s1 <= s; // subset
  var s3 := { 5, 6, 7 };
  assert s1 !| s3; // disjointness
}
```

- (a) Create a data type for binary search trees (BST).
- (b) Write a function that computes the set of all values in a BST.
- (c) Write a recursive predicate to check whether a tree is a BST (for all nodes in the tree, all elements to the left are smaller than the current node and all elements to the right are larger).
- (d) Write a function to compute a sequence of all integers in a BST (in order).
- (e) Prove as a post condition that the sequence is sorted, assuming the input is truly a BST.

4. Extra exercises.

- (a) Write a predicate `predicate all(l : List, p : int -> bool)` that checks whether all elements in a list satisfy the property `p`.
- (b) Implement the function `function map(l : List, f : int -> int) : List`.
- (c) Implement the function `function filter(l : List, f : int -> bool) : List`.
- (d) Implement the function `function reduce(l : List, init : int, f : int -> int -> int) : int`.
- (e) Write a ghost predicate that checks whether a sequence contains distinct elements.
- (f) Write a function that computes a `set<int>` from a `seq<int>`.