Verification-Driven Program Development
Exercise Sheet, Week 2

Ștefan Ciobâcă

09.10.2024

Exercises

1. Note how the following function makes use of the `if-then-else` expression.

```
function max(x : int, y : int) : int
{
  if x > y then
    x
  else
    y
}
```

   Write a function `abs` that returns the absolute value of an integer.

2. There is no difference in Dafny between a predicate and a function returning `bool`.

```
predicate isPositive(x : int)
{
  x > 0
}
```

   Write a predicate `isEven` that checks whether an integer is an even number.

3. You can write recursive functions, as long as you can convince Dafny that they terminate.

```
function fib(n : int)
  requires n >= 0
  decreases n
{
  if n <= 1 then
    n
  else
    fib(n - 1) + fib(n - 2)
}
```

   Write a function `factorial`.

4. 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)
  }
}
```

Write a function that returns the sum of the elements in a list.

5. Write a predicate that checks whether all elements in a list are positive.

6. Dafny supports higher-order functions. For example, the function `function apply(f: int -> int, n: int): int` is a second-order function, because it takes a function as an argument.

```
function apply(f : int -> int, x : int) : int
{
  f(x)
}

function succ(x : int) : int
{
  x + 1
}

method Main()
{
  print apply(succ, 10);
}
```

   (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`.

7. 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]
}
```

See `https://dafny.org/teaching-material/Lectures/1-1-Programming-Functional.html#/sec-sequences` for a cheat sheet on sequences.

(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) Here is a ghost predicate that checks whether all elements in a sequence are positive:

```
ghost predicate allPositive(s : seq<int>)
{
  forall i :: 0 <= i < |s| ==> s[i] > 0
}
```

Write a ghost predicate that checks whether a sequence is sorted in increasing order.

(d) Write a ghost predicate that checks whether a sequence contains distinct elements.

(e) Strings (type `string`) are sequences of characters (type `seq<char>`). Write a method that extracts `Hello` from `Hello, World!`, using sequence slice notation.

8. Sets are... sets (finite). 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
}
```

Find more information here:

- `https://dafny.org/teaching-material/Lectures/1-1-Programming-Functional.html#/sec-sets`
- `https://dafny.org/latest/OnlineTutorial/Sets`

(a) Write a ghost predicate that checks whether all elements in a `set<int>` are positive.

(b) Write a function that computes a `set<int>` from a `seq<int>`.

9. Create a data type for binary search trees (BST).

10. Write a function that computes the set of all values in a BST.

11. 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).

12. Write a function to compute a sequence of all integers in a BST (in order). Prove as a post condition that the sequence is sorted (assuming the input is truly a BST).