EECS 498 - 003 Lab

Keshav Singh

Lab #2

Agenda

- Quick Recap of Lecture Material
- Lab Exercises
- Office Hours(?)

Reference Material for Dafny Syntax

- If you need a reference for any dafny syntax when doing programming assignments, this guide has everything you need:

  https://dafny.org/dafny/DafnyRef/DafnyRef

Lemmas

- Lemmas consist of asserts, which are checked statically
- Once an assert is proven, it is assumed for all subsequent asserts
- Lemmas are opaque: anything you prove inside the body of a lemma can't be seen outside
- Use pre-conditions to restrict when lemma can be called, and post-conditions to export facts you prove in the lemma

```
lemma IntegerOrdering(a: int, b: int)
  requires b == a + 3
  ensures a < b
{
  assert a < b;
}
```

Precondition: statically checked anywhere this lemma is called

Postcondition: an exported assertion

Functions/Predicates

- Declarative styled functions (unlike those you usually write in C++)
- Unlike lemmas, not ghost unless specified (in this class, you usually want it to be specify as ghost)
- Function bodies are visible (can be made opaque for performance purposes)
- Functions also can have preconditions and postconditions
- Useful for defining specifications for more involved code
- Note: a predicate is just a function with return type bool

```
ghost function sumSpec(s:seq<int>) : (sum:int) {
    if |s| == 0 then
        0
    else
        s[0] + sumSpec(s[1..])
        // sumSpec(s[..|s|-1]) + s[|s|-1]
}
```

Methods

- Imperative styled functions (like you usually have in C++)
- Not ghost unless specified (not usually the case)
- Method bodies are opaque
- Use preconditions and postconditions as you usually do for lemmas
- Common to use function for spec of the methods as method bodies are visible

Methods

```
ghost method sumImperative(s:seq<int>) returns (sum:int)
    ensures sum == sumSpec(s)
{
    var i := 0;
    sum := 0;
    while (i < |s|)
    {
        sum := sum + s[i];
        i := i + 1;
    }
}
```

Boolean operators

- Common operators: !, &&, ||, ==, =>, <==>, forall, exists
- Example: forall a:int | Q(a) :: R(a)
- In order to verify an exist statement, you need to provide a witness (or example) to dafny where the condition holds
- In other words dafny cannot automatically find an example for you (in most cases)

Examples

```
lemma assertions()
{
    assert forall x:nat | x%4 == 0 :: x % 2 == 0;
    assert forall x:nat, y:nat | x == -1*y :: x+y == 0;
    assert 3*4 == 12;
    assert exists x:nat, y:nat :: x*y == 12;
}
```

Sets Syntax

```
a: set<int>
{1, 3, 5}   {}
7 in a
a <= b
a + b
a - b
a * b
a == b
|a|
set x: nat |
  x < 100 && x % 2 == 0
```

Sequences Syntax

a: seq<int>, b: seq<int>

[1, 3, 5]   []

7 in a

a + b

a == b

|a|

a[2..5]   a[3..]

seq(5, i => i * 2)

seq(5, i requires 0<=i => sqrt(i))

Map Syntax

a: map<int, set<int>>

map[2:={2}, 6:={2,3}]

7 in a   7 in a.Keys

a == b

a[5 := {5}]

map k | k in Evens()

        :: k/2

Multiset Syntax

a: multiset<int>

multiset{1, 3, 1, 5}   multiset{}, multiset([1, 3, 1, 5])

multiset({1,1}) == multiset{1}

7 in a

a[7]

a <= b

a + b

a - b

a * b

a == b

|a|

Loop invariants

- Need loop invariants for dafny to know what properties hold after/during a loop
- Helpful to come up with them using inductive reasoning
- Sometimes “decreases” causes can also be added to show dafny a loop terminates (like potential function)