

# **EECS498-008**

# **Formal Verification**

# **of Systems Software**

Material and slides created by  
Jon Howell and Manos Kapritsos

# The **var** expression

```
lemma foo()  
{  
  var set1 := { 1, 3, 5, 3 };  
  var seq1 := [ 1, 3, 5, 3 ];  
  
  assert forall i | i in set1 :: i in seq1;  
  assert forall i | i in seq1 :: i in set1;  
  assert |set1| < |seq1|;  
}
```

# Algebraic datatypes (“struct” and “union”)

```
datatype HAlign = Left | Center | Right
```



```
datatype VAlign = Top | Middle | Bottom
```

```
datatype TextAlign = TextAlign(hAlign:HAlign,  
vAlign:VAlign)
```



```
datatype Order = Pizza(toppings:set<Topping>  
| Shake(flavor:Fruit, whip: bool)
```

# Checking for types

```
predicate IsCentered(va: VAlign) {  
    !va.Top? && !va.Bottom?  
}
```

```
function DistanceFromTop(va: VAlign) : int {  
    match va  
        case Top => 0  
        case Middle => 1  
        case Bottom => 2  
}
```

# Hoare logic composition

```
lemma DoggiesAreQuadrupeds(pet:
Pet)
  requires IsDog(pet)
  ensures |Legs(pet)| == 4 { ... }
```

```
lemma StaticStability(pet: Pet)
  requires |Legs(pet)| >= 3
  ensures IsStaticallyStable(pet)
{ ... }
```

```
lemma DoggiesAreStaticallyStable(pet: Pet)
  requires IsDog(pet)
  ensures IsStaticallyStable(pet)
{
  DoggiesAreQuadrupeds(pet);
  StaticStability(pet);
}
```

# Autograder submissions: the RULES

- You may not use `/* */` comments
- You must leave the existing `/* */` comments in place
- You may only change text between `/*{*/` and `/*}*/`
- You are not allowed to add axioms (or to otherwise trivialize the proof)
  
- You are given three submissions per day
- You are given three late day tokens throughout the semester

# Example: exercise01.dfy

```
//#title Lemmas and assertions

lemma IntegerOrdering()
{
  // An assertion is a **static** check of a boolean expression -- a mathematical
  truth.
  // This boolean expression is about (mathematical) literal integers.
  // Run dafny on this file. See where it fails. Fix it.
  assert /*{*/ 5 < 3 /*}*/;
}
```

# Chapter 1 progress

- **Some** of you have already submitted
- Pleeeenty of time left, don't worry...
- Chapter 2 will be released on Wednesday evening



# Logistics

- I can't stay for "impromptu office hours" after class today
  - I have a dental appointment right after class

# Detour to Imperativeland

```
lemma loop(target:nat) returns (result:nat)
  ensures result == target
{
  result := 0;
  while (result < target)
    invariant result <= target
    {
      result := result + 1;
    }
  return result;
}
```

Dafny needs an invariant to reason about the loop's body

# Detour to Imperativeland

```
predicate IsMaxIndex(a:seq<int>, x:int) {  
  && 0 <= x < |a|  
  && (forall i | 0 <= i < |a| :: a[i] <= a[x])  
}
```

Note that the order of conjuncts matters!

And the same is true for ensures/requires: their order matters

# Imperativeland

**method** findMaxIndex(a:seq<int>) returns (x:int)

*requires*  $|a| > 0$

*ensures* IsMaxIndex(a, x)

{

var i := 1;

var ret := 0;

while(i < |a|)

*invariant*  $0 \leq i \leq |a|$

*invariant* IsMaxIndex(a[..i], ret)

{

if(a[i] > a[ret]) {

ret := i;

}

i := i + 1;

}

return ret;

}

```
predicate IsMaxIndex(a:seq<int>, x:int) {
  && 0 <= x < |a|
  && (forall i | 0 <= i < |a| :: a[i] <=
a[x])
}
```

# Recursion: exporting ensures

```
function Evens(count:int) : (outseq:seq<int>)  
  ensures forall idx :: 0<=idx<|outseq| ==> outseq[idx] == 2 * idx  
{  
  if count==0 then [] else Evens(count) + [2 * (count-1)]  
}
```