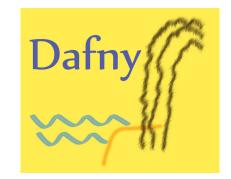
Dafny



- A Language and Program Verifier for Functional Correctness designed to support the static verification of programs
 - https://www.microsoft.com/en-us/research/project/dafny-a-language-and-program-verifier-for-functional-correctness/
 - https://en.wikipedia.org/wiki/Dafny
- Available for download
 - https://github.com/Microsoft/dafny
- Run in browser
 - http://rise4fun.com/dafny/Hello
- Tutorial
 - http://rise4fun.com/Dafny/tutorial/Guide

Dafny uses annotations to reason about code

- Generates a proof that the code matches the annotations
- Annotations are a form of specification
- Example forall k: int :: 0 <= k < a.Length ==> 0 < a[k]
 - All elements of array a are greater than 0.
- Proves that there are no runtime errors, null references, etc.
- Syntax is unique
 - Not the same as Java, C++ etc.
 - Targets C#

- the smallest unit of verification is the method
- assignment operator is :=
- preconditions use the requires keyword
- postconditions use the ensures keyword

```
method MethodName( x: int, y: int ) returns ( z: int, w: int )
  requires x == 0 && y >= 0  // PRECONDITION
  ensures z != 0 || w != 0  // POSTCONDITION
  {
    ...
}
```

Hello World in Dafny

```
method Main() {
  print "hello, Dafny\n";
  assert 10 < 2; // this assertion fails
}</pre>
```

Fibonacci

```
function Fibonacci(n: int): int
  decreases n // this recursive condition is violated
{
  // what is wrong here?
  if n < 2 then n else Fibonacci(n+2) + Fibonacci(n+1)
}</pre>
```

This should be

```
function Fibonacci(n: int): int
  decreases n
{
  if n < 2 then n else Fibonacci(n-2) + Fibonacci(n-1)
}
// Decreases is like a decrement function</pre>
```

CSCI-2600 Spring 2020

Assertions

```
method Abs(x: int) returns (x': int)
{
    x' := x;
    if(x' < 0) { x' := x' * -1; }
}
method Testing()
{
    var v := Abs(3);
    assert v == 3;
    assert 0 < v;
    assert 0 <= v;
}</pre>
```

```
method Abs(x: int) returns (x': int)
  ensures x' >= 0
  ensures (x < 0 && x' == -1*x) || (x' == x)
{
    x' := x;
    if(x' < 0) { x' := x' * -1; }
}

method Testing()
{
    var v := Abs(3);
    assert v == 3;
    assert 0 < v;
    assert 0 <= v;
}

cscl-2600 Spring 2020</pre>
```

• Compute x + y

Recursively multiply x * y

```
method Mul(x: int, y: int) returns (r: int)
  requires 0 <= x && 0 <= y
  ensures r == x*y
  decreases x
{
   if x == 0 {
      r := 0;
   } else {
      var m := Mul(x-1, y); // var declares a new variable
      r := m + x; // is this correct?
   }
}</pre>
```

```
// Can you make the program verify?
method M(n: int) returns (r: int)
  ensures r == n
  // what precondition do we need?
{
  var i := 0;
  while i < n
    // what invariant do we need here?
  {
    i := i + 1;
  }
  r := i;
}</pre>
```

Needs requires and ensures Needs a break; statement after leap year test Loop needs a decreases statement

```
// a function returning a bool
Predicate method isLeapYear(y: int) {
 y \% 4 == 0 \&\& (y \% 100 != 0 | | y \% 400 == 0)
// Does this method terminate?
method WhichYear InfiniteLoop(d: int) returns (year: int) {
 var days := d;
 year := 1980;
 while days > 365 {
  if isLeapYear(year) {
   if days > 366 {
    days := days - 366;
    year := year + 1;
  } else {
   days := days - 365;
   year := year + 1;
    CSCI-2600 Spring 2020
```

```
method WhichYear_InfiniteLoop(d: int) returns (year: int)
requires d > 0
ensures year >= 1980
var days := d;
year := 1980;
while days > 365
decreases days
  if isLeapYear(year) {
   if days > 366 {
    days := days - 366;
    year := year + 1;
   else {
    break;
  } else {
   days := days - 365;
   year := year + 1;
```

Solution for the preceeding slide

CSCI-2600 Spring 2020

there is an infinite loop if it's a leap year and days is equal to 366

```
method Find(a: array<int>, key: int) returns (index: int)
  requires a != null
  ensures 0 <= index ==> index < a.Length && a[index] == key
  ensures index < 0 ==> forall k :: 0 <= k < a.Length ==> a[k] != key
{
  index := 0;
  while index < a.Length
    invariant 0 <= index <= a.Length
    invariant forall k :: 0 <= k < index ==> a[k] != key
{
    if a[index] == key { return; }
    index := index + 1;
}
  index := -1;
}
```

Binary Search

```
predicate sorted(a: array<int>)
 requires a != null
 reads a
 forall j, k :: 0 <= j < k < a.Length ==> a[j] <= a[k]
method BinarySearch(a: array<int>, value: int) returns (index: int)
 requires a != null && 0 <= a.Length && sorted(a)
 ensures 0 <= index ==> index < a.Length && a[index] == value
 ensures index < 0 ==> forall k :: 0 <= k < a.Length ==> a[k] != value
 var low, high := 0, a.Length;
 while low < high
   invariant 0 <= low <= high <= a.Length
   invariant forall i ::
    0 <= i < a.Length && !(low <= i < high) ==> a[i] != value
   var mid := (low + high) / 2;
   if a[mid] < value {
    low := mid + 1;
   else if value < a[mid] {
    high := mid;
   else {
     return mid;
 return -1;
            CSCI-2600 Spring 2020
```