

AAA528: Computational Logic

Lecture 5 — Program Verification using Dafny

Hakjoo Oh
2023 Spring

Program Verification

Techniques for specifying and verifying program properties:

- **Specification:** precise statement of program properties in first-order logic. Also called program annotations.
 - ▶ The language of FOL offers precision.
 - ▶ Partial correctness properties vs Total correctness properties.
- **Verification methods:** for proving partial/total correctness
 - ▶ Inductive assertion method
 - ▶ Ranking function method



The Dafny Programming and Verification Language



Dafny is a verification-aware programming language that has native support for recording specifications and is equipped with a static program verifier. By blending sophisticated automated reasoning with familiar programming idioms and tools, Dafny empowers developers to write provably correct code (w.r.t. specifications). It also compiles

Dafny code to familiar development environments such as C#, Java, JavaScript and Go (with more in progress, such as Python) so Dafny can integrate with your existing workflow. Dafny makes rigorous verification an integral part of development, thus reducing costly late-stage bugs that may be missed by testing.

In addition to a verification engine to check implementation against specifications, the Dafny ecosystem includes several compilers, plugins for common software development IDEs, a LSP-based Language Server, a code formatter, a reference manual, tutorials, power user tips, books, the experiences of professors teaching Dafny, and the accumulating expertise of industrial projects using Dafny.

Dafny has support for common programming concepts such as

- mathematical and bounded integers and reals, bit-vectors, classes, iterators, arrays, tuples, generic types, refinement and inheritance,
- [inductive datatypes](#) that can have methods and are suitable for pattern matching,
- [lazily unbounded datatypes](#),
- [subset types](#), such as for bounded integers,
- [lambda expressions](#) and functional programming idioms,
- and [immutable and mutable data structures](#).

Dafny also offers an extensive toolbox for mathematical proofs about software, including

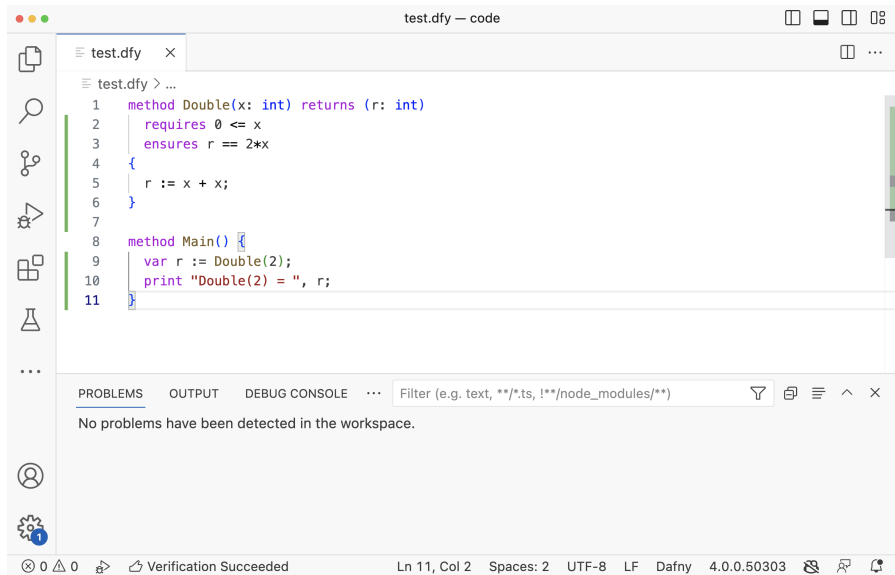
- [bounded and unbounded quantifiers](#),
- [calculational proofs](#) and the ability to use and prove lemmas,
- [pre- and post-conditions](#), [termination conditions](#), [loop invariants](#), and [read/write](#)

Quick Links

- [Installation](#) (or a [VSCode plugin for Dafny](#))
- [Dafny Reference Manual and User Guide](#)
- [Dafny Resources for Users](#)
- [Dafny GitHub project](#) (for developers of the [Dafny tools themselves](#))
- [Other documentation snapshots](#)
- [Book on Program Proofs using Dafny!](#): *Program Proofs*, by Rustan Leino, MIT Press

Blog

Example 1: Double



test.dfy — code

```
test.dfy > ...
1  method Double(x: int) returns (r: int)
2      requires 0 <= x
3      ensures r == 2*x
4  {
5      r := x + x;
6  }
7
8  method Main() {
9      var r := Double(2);
10     print "Double(2) = ", r;
11 }
```

PROBLEMS OUTPUT DEBUG CONSOLE ... Filter (e.g. text, **/*.ts, !**/node_modules/**)

No problems have been detected in the workspace.

Ln 11, Col 2 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303

Example 2: Triple

The screenshot shows the Dafny IDE with a file named `test.dfy` open. The editor displays the following code:

```
6 }  
7  
8 method Triple(x: int) returns (r: int)  
9   ensures r == 3*x  
10 {  
11   if 0 <= x {  
12     var y := Double (x);  
13     r := x + y;  
14   } else {  
15     var y := Double (-x);  
16     r := x - y;  
17   }  
18 }
```

Below the editor, the **PROBLEMS** tab is selected, showing the message: "No problems have been detected in the workspace." The status bar at the bottom indicates "Verification Succeeded" and provides details: "Ln 10, Col 2 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303".

Example 3: Simple Loop Invariant



The screenshot shows the Dafny IDE interface. The main editor window displays the file `test.dfy` with the following code:

```
1  method M(n: int) returns (r: int)
2      requires n >= 0
3      ensures n == r
4  {
5      var i := 0;
6      while i < n
7      invariant i <= n
8      {
9          i := i + 1;
10     }
11     r := i;
12 }
13
```

The left sidebar contains icons for Explorer, Search, Source Control, Run and Debug, Extensions, Testing, and Settings. The bottom status bar shows "Verification Succeeded" and "Ln 13, Col 1".

test.dfy — code

test.dfy > ...

1 method M(n: int) returns (r: int)

2 requires n >= 0

3 ensures n == r

4 {

5 var i := 0;

6 while i < n

7 invariant i <= n

8 {

9 i := i + 1;

10 }

11 r := i;

12 }

13

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Filter (e.g. text, **/*.ts...)

No problems have been detected in the workspace.

0 0 0 Verification Succeeded Ln 13, Col 1 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303

Example 4: ComputeSum



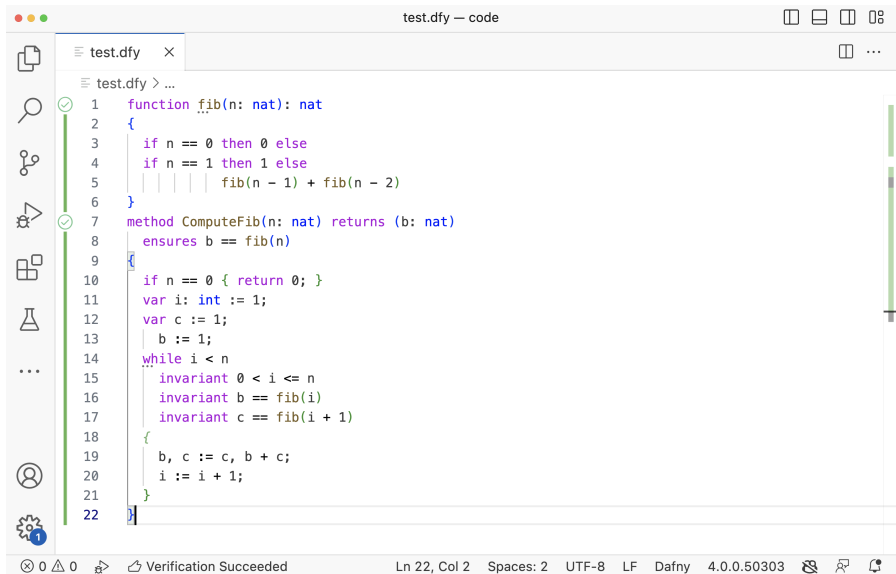
test.dfy — code

```
test.dfy > ...  
1  function Sum(n: nat): nat  
2  {  
3    if n == 0 then 0 else n + Sum(n-1)  
4  }  
5  
6  method ComputeSum(n: nat) returns (s: nat)  
7  ensures s == Sum(n)  
8  {  
9    var i := 1;  
10   var sum := 0;  
11   while i <= n  
12   invariant 1 <= i <= n+1  
13   invariant sum == Sum(i-1)  
14   {  
15     sum := sum + i;  
16     i := i + 1;  
17   }  
18   assert(i == n + 1);  
19   return sum;  
20 }  
21 |
```

Ln 21, Col 1 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303

Verification Succeeded

Example 5: ComputeFib



```
test.dfy — code

test.dfy > ...

1  function fib(n: nat): nat
2  {
3      if n == 0 then 0 else
4      if n == 1 then 1 else
5      | | | | fib(n - 1) + fib(n - 2)
6  }
7  method ComputeFib(n: nat) returns (b: nat)
8      ensures b == fib(n)
9  {
10     if n == 0 { return 0; }
11     var i: int := 1;
12     var c := 1;
13     b := 1;
14     while i < n
15     invariant 0 < i <= n
16     invariant b == fib(i)
17     invariant c == fib(i + 1)
18     {
19         b, c := c, b + c;
20         i := i + 1;
21     }
22 }
```

Ln 22, Col 2 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303

Verification Succeeded

Example 6: Simple Recursion

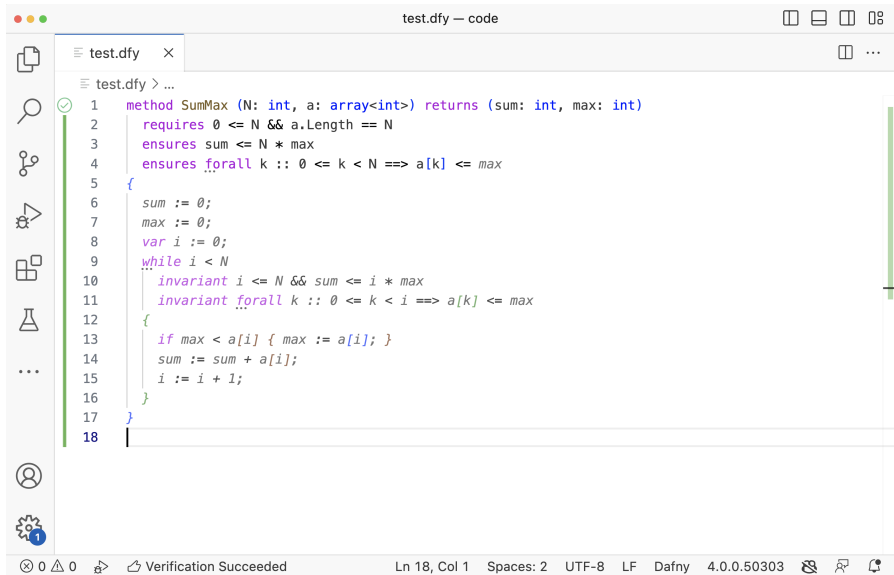


The screenshot shows the Dafny IDE interface. The title bar reads "test.dfy — code". The editor window displays a Dafny program for a recursive multiplication function. The code is as follows:

```
1 method Mul(x: int, y: int) returns (r: int)
2   requires 0 <= x && 0 <= y
3   ensures r == x * y
4   {
5     if x == 0 {
6       r := 0;
7     } else {
8       var m := Mul(x-1, y);
9       r := m + y;
10    }
11  }
12
```

The status bar at the bottom indicates "Verification Succeeded" and shows the current position as "Ln 12, Col 1". Other status information includes "Spaces: 2", "UTF-8", "LF", "Dafny", and the version "4.0.0.50303".

Example 7: Quantification



The screenshot shows the Dafny IDE with a file named `test.dfy`. The code defines a method `SumMax` that takes an integer `N` and an array of integers `a`, and returns a sum and a maximum value. The method is annotated with several logical properties:

- `requires 0 <= N && a.Length == N`: A precondition ensuring `N` is non-negative and the array `a` has the correct length.
- `ensures sum <= N * max`: A postcondition ensuring the computed sum is not greater than `N` multiplied by the maximum value.
- `ensures forall k :: 0 <= k < N ==> a[k] <= max`: A postcondition ensuring that every element in the array is less than or equal to the computed maximum.

The method body implements a loop that iterates through the array, calculating the sum and finding the maximum value. It includes loop invariants to maintain the correctness of the computation:

- `invariant i <= N && sum <= i * max`: An invariant ensuring the current index `i` is within bounds and the partial sum is bounded by `i` times the current maximum.
- `invariant forall k :: 0 <= k < i ==> a[k] <= max`: An invariant ensuring that all elements processed so far are less than or equal to the current maximum.

The status bar at the bottom indicates that verification succeeded for the current state (Ln 18, Col 1).

```
1 method SumMax (N: int, a: array<int>) returns (sum: int, max: int)
2   requires 0 <= N && a.Length == N
3   ensures sum <= N * max
4   ensures forall k :: 0 <= k < N ==> a[k] <= max
5   {
6     sum := 0;
7     max := 0;
8     var i := 0;
9     while i < N
10    {
11      invariant i <= N && sum <= i * max
12      invariant forall k :: 0 <= k < i ==> a[k] <= max
13      {
14        if max < a[i] { max := a[i]; }
15        sum := sum + a[i];
16        i := i + 1;
17      }
18    }
```

Example 8: Find



test.dfy — code

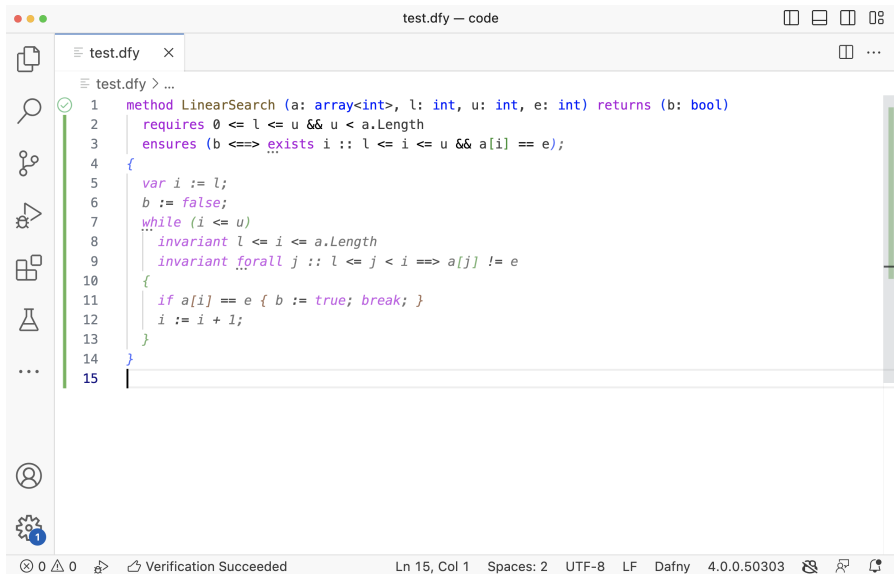
test.dfy ×

test.dfy > Find

```
1  method Find(a: array<int>, key: int) returns (index: int)
2      ensures 0 <= index ==> index < a.Length && a[index] == key
3      ensures index < 0 ==> forall k :: 0 <= k < a.Length ==> a[k] != key
4
5      index := 0;
6      while index < a.Length
7      {
8          invariant 0 <= index <= a.Length
9          invariant forall k :: 0 <= k < index ==> a[k] != key
10         {
11             if a[index] == key { return; }
12             index := index + 1;
13         }
14     }
15     index := -1;
```

0 0 Verification Succeeded Ln 13, Col 18 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303

Example 9: LinearSearch

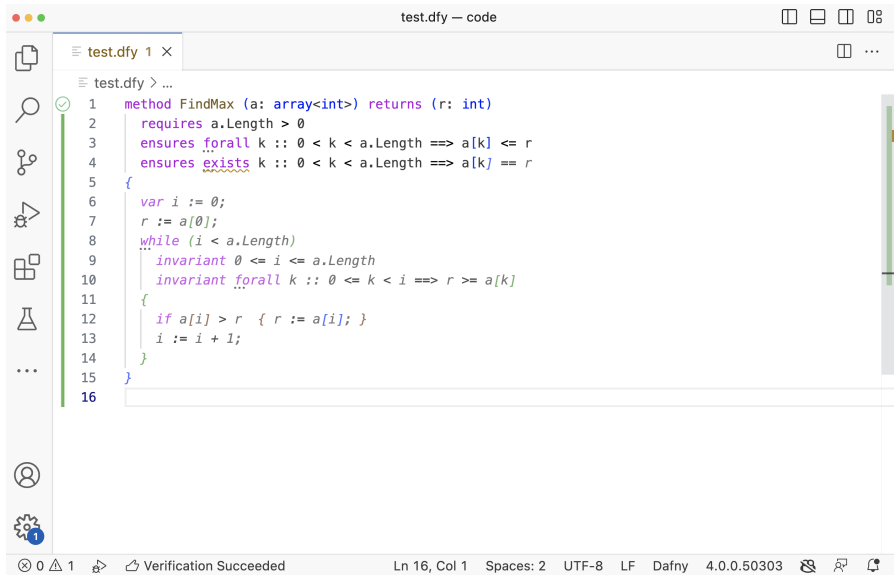


The screenshot shows the Dafny IDE with a file named 'test.dfy'. The code defines a method 'LinearSearch' that takes an array 'a' and three integers 'l', 'u', and 'e'. It returns a boolean 'b'. The method is annotated with a 'requires' clause and an 'ensures' clause. The implementation uses a 'while' loop to iterate from 'l' to 'u', checking if 'a[i] == e'. If found, it sets 'b' to true and breaks. If not found, it sets 'b' to false. The method is annotated with a 'requires' clause and an 'ensures' clause.

```
test.dfy — code
test.dfy > ...
1  method LinearSearch (a: array<int>, l: int, u: int, e: int) returns (b: bool)
2      requires 0 <= l <= u && u < a.Length
3      ensures (b <==> exists i :: l <= i <= u && a[i] == e);
4  {
5      var i := l;
6      b := false;
7      while (i <= u)
8      {
9          invariant l <= i <= a.Length
10         invariant forall j :: l <= j < i ==> a[j] != e
11         {
12             if a[i] == e { b := true; break; }
13             i := i + 1;
14         }
15     }
```

Ln 15, Col 1 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303

Example 10: FindMax

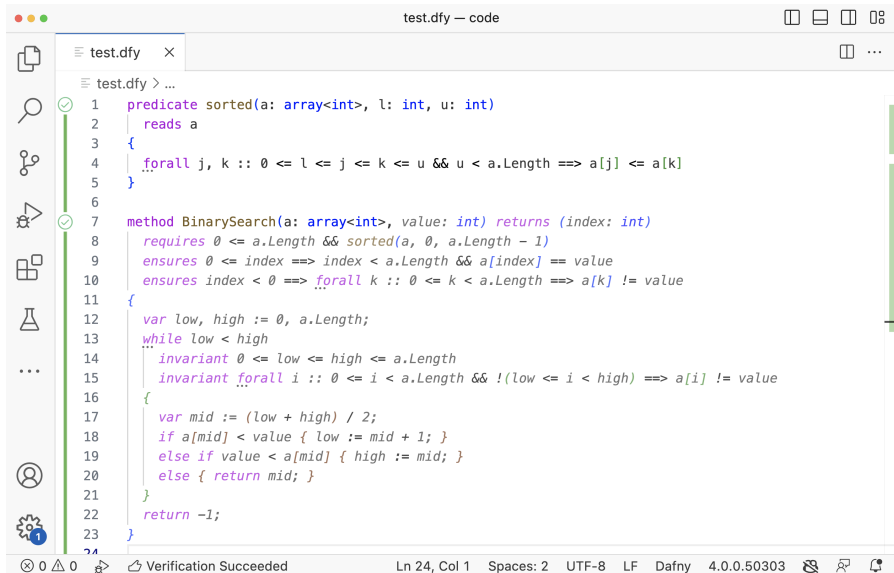


The screenshot shows the Dafny IDE with a file named `test.dfy` open. The code defines a method `FindMax` that takes an array `a` of integers and returns the maximum value `r`. The method includes preconditions, postconditions, and a loop with invariants.

```
test.dfy 1 x
test.dfy > ...
1  method FindMax (a: array<int>) returns (r: int)
2      requires a.Length > 0
3      ensures forall k :: 0 < k < a.Length ==> a[k] <= r
4      ensures exists k :: 0 < k < a.Length ==> a[k] == r
5  {
6      var i := 0;
7      r := a[0];
8      while (i < a.Length)
9      ...
10     invariant 0 <= i <= a.Length
11     invariant forall k :: 0 <= k < i ==> r >= a[k]
12     {
13         if a[i] > r { r := a[i]; }
14         i := i + 1;
15     }
16 }
```

The status bar at the bottom indicates: `Ln 16, Col 1 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303`. The verification status is `Verification Succeeded`.

Example 11: Binary Search

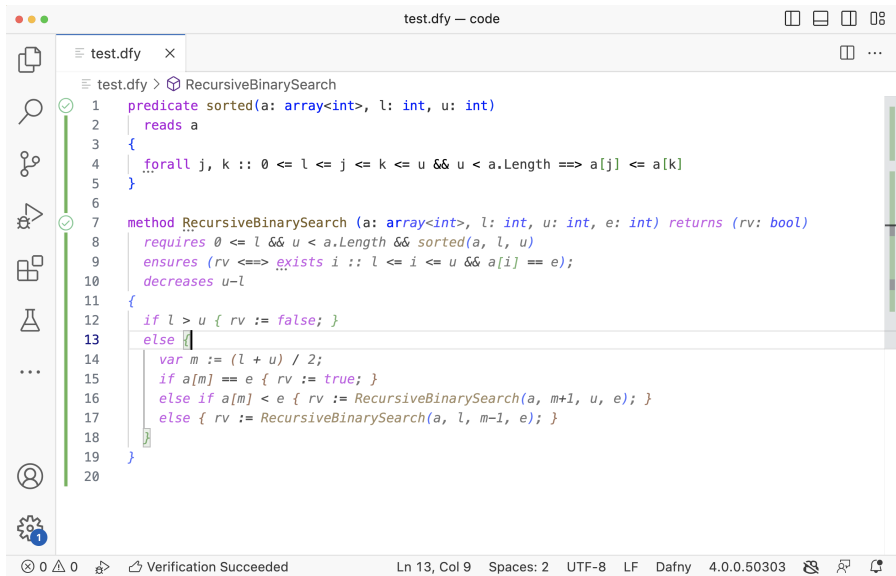


The screenshot shows the Dafny IDE interface. The title bar reads "test.dfy — code". The editor window contains the following code:

```
1 predicate sorted(a: array<int>, l: int, u: int)
2   reads a
3   {
4     forall j, k :: 0 <= l <= j <= k <= u && u < a.Length ==> a[j] <= a[k]
5   }
6
7 method BinarySearch(a: array<int>, value: int) returns (index: int)
8   requires 0 <= a.Length && sorted(a, 0, a.Length - 1)
9   ensures 0 <= index ==> index < a.Length && a[index] == value
10  ensures index < 0 ==> forall k :: 0 <= k < a.Length ==> a[k] != value
11  {
12    var low, high := 0, a.Length;
13    while low < high
14    invariant 0 <= low <= high <= a.Length
15    invariant forall i :: 0 <= i < a.Length && !(low <= i < high) ==> a[i] != value
16    {
17      var mid := (low + high) / 2;
18      if a[mid] < value { low := mid + 1; }
19      else if value < a[mid] { high := mid; }
20      else { return mid; }
21    }
22    return -1;
23  }
```

The status bar at the bottom indicates "Verification Succeeded" and "Ln 24, Col 1". The Dafny version is 4.0.0.50303.

Example 12: Recursive Binary Search



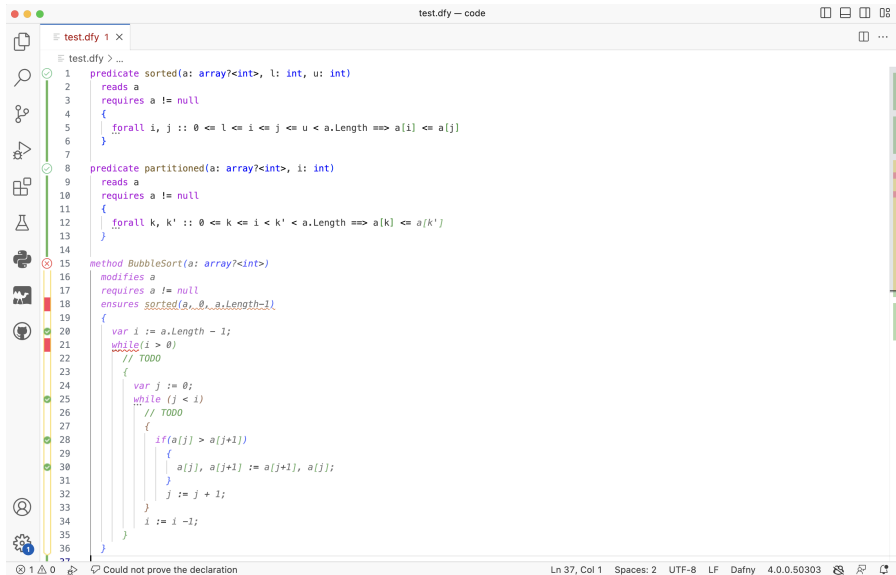
The screenshot shows the Dafny IDE with a file named `test.dfy` open. The code defines a `RecursiveBinarySearch` method. The IDE interface includes a sidebar with icons for file management, search, and other tools. The status bar at the bottom indicates that verification succeeded.

```
test.dfy — code

test.dfy > RecursiveBinarySearch
1 predicate sorted(a: array<int>, l: int, u: int)
2   reads a
3 {
4   forall j, k :: 0 <= l <= j <= k <= u && u < a.Length ==> a[j] <= a[k]
5 }
6
7 method RecursiveBinarySearch (a: array<int>, l: int, u: int, e: int) returns (rv: bool)
8   requires 0 <= l && u < a.Length && sorted(a, l, u)
9   ensures (rv <==> exists i :: l <= i <= u && a[i] == e);
10  decreases u-l
11 {
12   if l > u { rv := false; }
13   else {
14     var m := (l + u) / 2;
15     if a[m] == e { rv := true; }
16     else if a[m] < e { rv := RecursiveBinarySearch(a, m+1, u, e); }
17     else { rv := RecursiveBinarySearch(a, l, m-1, e); }
18   }
19 }
20
```

Ln 13, Col 9 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303

Quiz: Bubble Sort



The screenshot shows the Dafny IDE with a file named `test.dfy`. The code defines two predicates, `sorted` and `partitioned`, and a `BubbleSort` method. The `sorted` predicate checks if an array is sorted between indices `l` and `u`. The `partitioned` predicate checks if an array is partitioned around a pivot `i`. The `BubbleSort` method sorts an array `a` in place. A proof error is shown on line 18, where the `ensures` clause `sorted(a, 0, a.Length - 1)` cannot be proven.

```
test.dfy 1 x
test.dfy > ...
1 predicate sorted(a: array?<int>, l: int, u: int)
2   reads a
3   requires a != null
4   {
5     forall i, j :: 0 <= l <= i <= j <= u < a.Length ==> a[i] <= a[j]
6   }
7
8 predicate partitioned(a: array?<int>, i: int)
9   reads a
10  requires a != null
11  {
12    forall k, k' :: 0 <= k <= i < k' < a.Length ==> a[k] <= a[k']
13  }
14
15 method BubbleSort(a: array?<int>)
16   modifies a
17   requires a != null
18   ensures sorted(a, 0, a.Length - 1)
19 {
20   var i := a.Length - 1;
21   while(i > 0)
22   // TODO
23   {
24     var j := 0;
25     while (j < i)
26     // TODO
27     {
28       if(a[j] > a[j+1])
29       {
30         a[j], a[j+1] := a[j+1], a[j];
31       }
32       j := j + 1;
33     }
34     i := i - 1;
35   }
36 }
37
Could not prove the declaration
Ln 37, Col 1 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303
```