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

dafny.org

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

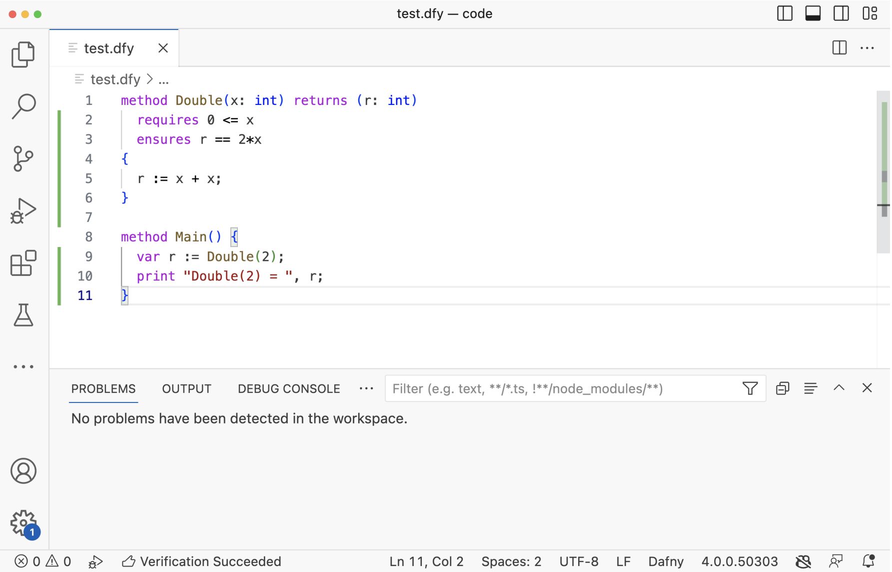
- bounded and unbounded quantifiers,
- calculational proofs and the ability to use and prove lemmas,
- pre- and post-conditions, termination conditions, loop invariants, and read/write

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



Example 2: Triple

test.dfy — code

test.dfy > Triple

```
}

method Triple(x: int) returns (r: int)
  ensures r == 3*x
{
  if 0 <= x {
    var y := Double (x);
    r := x + y;
  } else {
    var y := Double (-x);
    r := x - y;
  }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

No problems have been detected in the workspace.

0 0 Verification Succeeded Ln 10, Col 2 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303

Example 3: Simple Loop Invariant

test.dfy — code

test.dfy

```
method M(n: int) returns (r: int)
  requires n >= 0
  ensures n == r
{
  var i := 0;
  while i < n
    invariant i <= n
  {
    i := i + 1;
  }
  r := i;
}

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

No problems have been detected in the workspace.

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 > ...

```
function Sum(n: nat): nat
{
  if n == 0 then 0 else n + Sum(n-1)
}

method ComputeSum(n: nat) returns (s: nat)
  ensures s == Sum(n)
{
  var i := 1;
  var sum := 0;
  while i <= n
    invariant 1 <= i <= n+1
    invariant sum == Sum(i-1)
  {
    sum := sum + i;
    i := i + 1;
  }
  assert(i == n + 1);
  return sum;
}
```

0 ⚠ 0 Verification Succeeded Ln 21, Col 1 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303

Example 5: ComputeFib

test.dfy — code

test.dfy

```
function fib(n: nat): nat
{
  if n == 0 then 0 else
  if n == 1 then 1 else
          fib(n - 1) + fib(n - 2)
}
method ComputeFib(n: nat) returns (b: nat)
  ensures b == fib(n)
{
  if n == 0 { return 0; }
  var i: int := 1;
  var c := 1;
    b := 1;
  while i < n
    invariant 0 < i <= n
    invariant b == fib(i)
    invariant c == fib(i + 1)
  {
    b, c := c, b + c;
    i := i + 1;
  }
}
```

0 0 Verification Succeeded Ln 22, Col 2 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303

Example 6: Simple Recursion

test.dfy — code

test.dfy > ...

```
method Mul(x: int, y: int) returns (r: int)
  requires 0 <= x && 0 <= y
  ensures r == x * y
{
  if x == 0 {
    r := 0;
  } else {
    var m := Mul(x-1, y);
    r := m + y;
  }
}
```

Verification Succeeded   Ln 12, Col 1   Spaces: 2   UTF-8   LF   Dafny   4.0.0.50303

Example 7: Quantification

test.dfy — code

test.dfy

```
method SumMax (N: int, a: array<int>) returns (sum: int, max: int)
  requires 0 <= N && a.Length == N
  ensures sum <= N * max
  ensures forall k :: 0 <= k < N ==> a[k] <= max
{
  sum := 0;
  max := 0;
  var i := 0;
  while i < N
    invariant i <= N && sum <= i * max
    invariant forall k :: 0 <= k < i ==> a[k] <= max
  {
    if max < a[i] { max := a[i]; }
    sum := sum + a[i];
    i := i + 1;
  }
}

```

0 0 Verification Succeeded Ln 18, Col 1 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303

Example 8: Find

test.dfy — code

test.dfy > Find

```
method Find(a: array<int>, key: int) returns (index: int)
  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;
}
```

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

Example 9: LinearSearch

test.dfy — code

test.dfy

test.dfy > ...

```
method LinearSearch (a: array<int>, l: int, u: int, e: int) returns (b: bool)
  requires 0 <= l <= u && u < a.Length
  ensures (b <==> exists i :: l <= i <= u && a[i] == e);
{
  var i := l;
  b := false;
  while (i <= u)
    invariant l <= i <= a.Length
    invariant forall j :: l <= j < i ==> a[j] != e
  {
    if a[i] == e { b := true; break; }
    i := i + 1;
  }
}

```

0 0 Verification Succeeded Ln 15, Col 1 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303

Example 10: FindMax

```
method FindMax (a: array<int>) returns (r: int)
  requires a.Length > 0
  ensures forall k :: 0 < k < a.Length ==> a[k] <= r
  ensures exists k :: 0 < k < a.Length ==> a[k] == r
{
  var i := 0;
  r := a[0];
  while (i < a.Length)
    invariant 0 <= i <= a.Length
    invariant forall k :: 0 <= k < i ==> r >= a[k]
  {
    if a[i] > r  { r := a[i]; }
    i := i + 1;
  }
}
```

Example 11: Binary Search

test.dfy — code

test.dfy > ...

```
predicate sorted(a: array<int>, l: int, u: int)
  reads a
{
  forall j, k :: 0 <= l <= j <= k <= u && u < a.Length ==> a[j] <= a[k]
}

method BinarySearch(a: array<int>, value: int) returns (index: int)
  requires 0 <= a.Length && sorted(a, 0, a.Length - 1)
  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;
}
```

0 0 Verification Succeeded Ln 24, Col 1 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303

Example 12: Recursive Binary Search

test.dfy — code

test.dfy

test.dfy > RecursiveBinarySearch

```
predicate sorted(a: array<int>, l: int, u: int)
  reads a
{
  forall j, k :: 0 <= l <= j <= k <= u && u < a.Length ==> a[j] <= a[k]
}

method RecursiveBinarySearch (a: array<int>, l: int, u: int, e: int) returns (rv: bool)
  requires 0 <= l && u < a.Length && sorted(a, l, u)
  ensures (rv <==> exists i :: l <= i <= u && a[i] == e);
  decreases u-l
{
  if l > u { rv := false; }
  else {
    var m := (l + u) / 2;
    if a[m] == e { rv := true; }
    else if a[m] < e { rv := RecursiveBinarySearch(a, m+1, u, e); }
    else { rv := RecursiveBinarySearch(a, l, m-1, e); }
  }
}
```

⊗ 0 ⚠ 0 Verification Succeeded Ln 13, Col 9 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303

Quiz: Bubble Sort

test.dfy — code

test.dfy 1

test.dfy > ...

```
predicate sorted(a: array?<int>, l: int, u: int)
  reads a
  requires a != null
  {
    forall i, j :: 0 <= l <= i <= j <= u < a.Length ==> a[i] <= a[j]
  }

predicate partitioned(a: array?<int>, i: int)
  reads a
  requires a != null
  {
    forall k, k' :: 0 <= k <= i < k' < a.Length ==> a[k] <= a[k']
  }

method BubbleSort(a: array?<int>)
  modifies a
  requires a != null
  ensures sorted(a, 0, a.Length-1)
  {
    var i := a.Length - 1;
    while(i > 0)
      // TODO
      {
        var j := 0;
        while (j < i)
          // TODO
          {
            if(a[j] > a[j+1])
              {
                a[j], a[j+1] := a[j+1], a[j];
              }
              j := j + 1;
          }
          i := i -1;
      }
  }
```

⊗ 1 ⚠ 0 Could not prove the declaration Ln 37, Col 1 Spaces: 2 UTF-8 LF Dafny 4.0.0.50303