272: Software Engineering
Winter 2024

Instructor: Tevfik Bultan

Lecture: Dafny

Dafny: A quick overview

These slides are based on:

- "Getting Started with Dafny: A Guide" Jason Koenig and K. Rustan M. Leino

https://dafny.org/dafny/OnlineTutorial/guide

Other Dafny resources:

- https://dafny.org/
- https://dafny.org/dafny/QuickReference
- https://dafny.org/latest/Installation
- https://dafny.org/latest/toc

Dafny

- Dafny is a programming language
- And, Dafny is a program verification tool

- Dafny supports program specification and verification using method pre- and postconditions and loop invariants
- Uses a verification approach similar to ESC/Java
- Uses Z3 SMT solver as a backend theorem prover

Dafny

- Dafny relies on high-level annotations to reason about and prove correctness of code

- Dafny lifts the burden of writing bug-free code into that of writing bug-free annotations

- A Dafny annotation stating that each element of array a is strictly positive:

```
forall k: int :: 0 <= k < a.Length ==> 0 < a[k]
```

- In addition to proving a correspondence to user supplied annotations, Dafny proves that there are no run time errors
  - index out of bounds, null dereferences, division by zero, etc.

Dafny

Object-based language

- generic classes, no subclassing
- object references, dynamic allocation
- sequential control

Built-in specifications

- pre- and postconditions
- framing
- loop invariants, inline assertions
- termination

Specification support

- Sets, sequences, inductive datatypes, …
- User-defined recursive functions
- Ghost variables

Dafny: pre and post-conditions

Dafny supports pre and post-conditions:

```
method MultipleReturns(x: int, y: int) returns (more:
int, less: int)
      requires 0 < y
      ensures less < x < more
{
      more := x + y;
      less := x - y;
}
```

Dafny: assertions, modular verification

Dafny supports assertions

Dafny cannot prove the assertion below since it uses modular verification and takes into account only pre and post-conditions of a called method

```
method Abs(x: int) returns (r: int)
{
      if (x < 0)
            { return -x; }
      else
            { return x; }
}
method Testing()
{
      var v := Abs(3);
      assert 0 <= v;
}
```

Dafny: assertions, modular verification

Dafny can prove the assertion below

```
method Abs(x: int) returns (y: int)
      ensures 0 <= y
      ensures 0 <= x ==> y == x
      ensures x < 0 ==> y == -x
{
      if (x < 0) { y := -x; }
      else { y := x; }
}
method Testing()
{
      var v := Abs(3);
      assert 0 <= v;
}
```

Dafny: loop invariants

Dafny supports specification and verification of loop invariants

```
method Loop(n: int) returns (i: int)
  requires n >= 0
{
  i := 0;
  while (i < n)
    invariant 0 <= i <= n
  {
     i := i + 1;
  }
  assert i == n;
}
```

Dafny: functions

Dafny allows definition of functions which are more like mathematical functions.

A Dafny function cannot write to memory, and consists solely of one expression.

```
function fib(n: nat): nat
{
      if n == 0 then 0 else
      if n == 1 then 1 else
                  fib(n - 1) + fib(n - 2)
}
```

Dafny: functions

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

Dafny: arrays and quantifiers

```
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;
}
```

Dafny: termination

Dafny can prove termination for loops and recursion

A decreases annotation specifies an expression that is expected to decrease with every loop iteration or recursive call

Dafny verifies two conditions to verify termination given a decreases annotation:

1. the decreases expression gets smaller in every loop iteration or recursive call
2. that it is bounded

These conditions together prove termination

Dafny: framing

Dafny supports “reads” annotations to specify which memory locations a function is allowed to read

Dafny supports “modifies” annotations to specify which memory locations a method is allowed to modify