Programme Verification with Dafny

Amar Banerjee

Programme Verification

March 12, 2020

Outline

What is Dafny ?

- Dafny is a programming language, verifier, and compiler
- Designed from the ground-up with static verification in mind
- Uses SMT solver to automatically prove correctness
- When proof not possible, requests proof annotations

Dafny Verifies What ?

- Dafny makes it easier to write correct code
- Correctness means two things:
  - No runtime errors: null deref., div. by 0, index o.o.b, ...
  - Program does what you indended
  - Terminates (when applicable)
- Your intentions are captured with a specification

Methods: Introduction

- Unit of executable code
- Note that:
  - Types for parameters and return values are required
  - Types are given after names, followed by ":"
  - Return values are named
- Methods can have multiple return values

```
method Abs(x: int) returns (r: int)
{
...
}
method M() returns (r1: int, r2:int)
{
...
}
```

Methods: Continued...

- To return a value, assign to the named return variable
- You can assign to the same return value multiple times
- Assignments use :=, not =
- No valid syntax in Dafny uses a single =

```
method MultipleReturns(x: int, y: int)
 returns (r1: int, r2:int)
{
r1 := x + y;
r2 := x - y;
// Comments are given
/* in typical C/Java fashion */
}
```

Methods: Something more...

- You can also use return statements
- Input parameters are always read-only
- Compound statements (if, while, ...) always need curly braces

```
method Abs(x: int) returns (x': int)
{
if(x < 0) {
return -x;
} else {
return x;
}
```

Functions

- Think: pure mathematical functions
- Cannot write to memory
- Body is a single expression
- Single return value
- Not compiled and executed
- Used directly in annotations: Pre/post-conditions, Assertions, Invariants

```
function abs(x: int): int
{
if x < 0 then -x else x
}

method CheckFunc(){
assert abs(3) == 3;
}
```

Type System

- int: Integer
- nat: Natural Numbers
- float: Decimal
- array: List

```
method FindMax(a: array<int>) returns (max: int)
```

Inductive

Sequence

Pre-Conditions

- Expression that must be true when method is called
- Again, these are statically-checked by Dafny
- Your job: assume pre-conditions, make sure post-conditions hold

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

Post-Conditions

- Expression that is always true after method executes
- These are statically-checked by Dafny
- Note: could have also written less $<$ x $<$ more

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

Loop Invariance

Loop invariants hold:

- Upon entering the loop
- After every iteration of the loop body

Dafny must consider all possible executions of the program

- Loops present a problem: how many times will it execute?
- Invariants let Dafny make assumptions about what carries through any number of loop executions

```
var i := 0;
while(i < n)
invariant 0 <= i;
{
i := i + 1;
}
```

Assertions

- Expression that must be true when execution reaches statement
- Dafny will attempt to prove that the assertion holds
- Aside: local variable types can usually be inferred
- Notice: methods can't be called from Boolean exprs.

```
method CheckAbs(x: int)
{
var v := Abs(x);
assert 0 <= v;
}
```

Terminations

Dafny proves termination

- Specification element: *decreases* annotation
  - Attach to loops and recursive functions
  - Provide termination metric
- Termination metric:
  - Gets smaller every iteration
  - Has a lower bound

```
while (i < n)
invariant 0 <= i <= n;
decreases n - i;
{
i := i + 1;
}
```

Reading Materials

- Strongly encouraged: complete the main tutorial at `http://rise4fun.com/Dafny/tutorial`
- Getting started guide: `http://goo.gl/mJ1Grr`
- Slightly older guide: `http://goo.gl/MVYsbq`
- Main webpage: `http://goo.gl/G1XDiK`
- Reference manual: `http://goo.gl/IGVbYY` (note: this is a work in progress)