Program Verification with Dafny (Part 1)

Franck Cassez
Trustworthy Smart Contracts Team, ConsenSys Software R&D
https://franck44.github.io/
March 2021

Objectives of this session

1. Write specifications with pre/post-conditions
2. Loop invariants & loop termination
3. Applications using Dafny
   - 3.1. Check correctness
   - 3.2. Loop invariant & loop termination
   - 3.3. Debug (counter-example)
4. Compile (and Run in VSCode)
5. Write an algorithm, its spec and verify it.

Hoare logic proof with Dafny

How does it work? Pre/post conditions – Floyd-Hoare Logic

Specification
What properties should the result satisfy?

Implementation
How the result is computed

```
function get_next_power_of_two(n : int) : int
  requires n >= 0 // pre-condition
  ensures get_next_power_of_two(n) >= 1 // post-condition
{
  if n <= 2 then 2
  else 2 * get_next_power_of_two( (n + 1) / 2)
}
```

Sir C.A.R. Hoare (1934 –)
Turing Award 1980

Floyd-Hoare Logic

R. W. Floyd (1936 – 2001)
Turing Award 1978

Floyd-Hoare Logic – Partial Correctness

Pre-condition

Post-condition

{ P } program { Q }

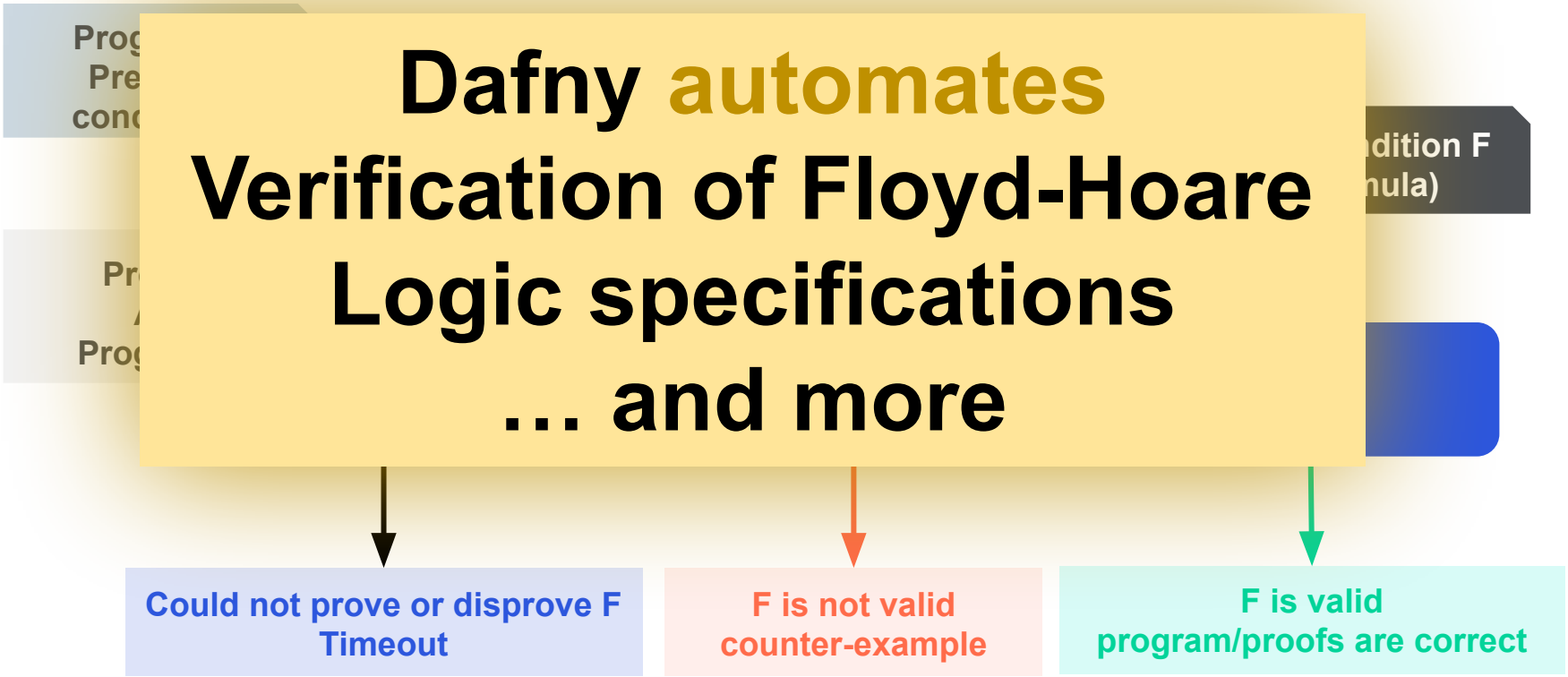
{ true } x := 2 { x >= 2 } ✅

{ x >= 0 } x := x + 1 { x >= 4 }

{ true }
while true do skip; od
{ false }

{ i >= 0 } while i > 0 do i := i - 1; od { i == 0 } ✅

Dafny – A *Verification-Friendly* Programming Language



Checking Hoare Triples – SAT Problem

{ x >= 1 } x := x + 1 { x >= 2 } ✅

x_1 >= 1 && x_2 == x_1 + 1 && not(x_2 >= 2)

UNSAT

{ x >= 0 } x := x + 1 { x >= 4 }

x_1 >= 0 && x_2 == x_1 + 1 && not(x_2 >= 4)

SAT

x_1 = 1
X_2 = 2

Examples in VSCode

Example 0.a

```
method abs(x: int) returns (y : int)
    requires  true;
    ensures  true;
{
    if (x < 0) {
        return -x;
    } else {
        return x;
    }
}
```

Task

1. Add post condition to specify result is >= 0
2. Find a pre-condition that guarantees the post condition.
3. Call abs and try to verify a property

Example 0.b

```
method max(x: int, y: int)returns(m : int)
requires true;
ensures true;
{
    var r : int;
    if ( ) {
        r := ... ;
    } else {
        r := ... ;
    }
    r := m;
    return m;
}
```

Task

1. Add code to compute max
2. Compile/Execute
3. Write pre/post conditions for max
4. Verify
5. Simplify (use of r)

Example 1 – Loop

```
method ex1(n: int)
    requires true
    ensures true
    decreases *
{
    var i := 0;
    while (i < n)
    {
        i := i + 1;
    }
    /** Property to prove:*/
    assert i == n;
}
```

Loop Invariant Rule

Premise

$$\{ I \wedge C \}\ BODY\ \{ I \}$$

$$\{ I \}\ \text{While}\ C\ \text{do}\ BODY\ \text{od}\ \{ I \wedge \neg C \}$$

Conclusion

```
while ( C )
  invariant I;
{
  BODY ;
}
assert I ∧ ¬ C
assert P;
```

$$I \wedge \neg C \Rightarrow P$$

Ensure P implied by Inv and loop exit

Example 1 – Loop

```
method ex1(n: int)
    requires true
    ensures true
    decreases *
{
    var i := 0;
    while ( i < n )
        invariant I ;
        decreases *;
    {
        i := i + 1;
    }
    /** Property to prove:*/
    assert i == n;
}
```

$$\{\ I \wedge C\ \}\ \text{BODY}\ \{\ I\ \}$$

$$\{\ I\ \}\ \text{While}\ C\ \text{do}\ \text{BODY}\ \text{od}\ \{\ I \wedge \neg C\ \}$$

Task

1. Add invariant to prove assert:
   a. Hint1: not(C) ⇔ i >= n
   b. Hint2: Inv ⋀ (i >= n) ⇒ i == n
2. Termination: add decreasing measure to prove termination
   a. Hint1: bounded from below
   b. Hint2: strictly decreasing.

Mechanical Proof of Loop Invariant Rule

```
method ex1(n: int)
    requires true
    ensures true
    decreases *
{
    var i := 0;
    while ( i < n )
        invariant i <= n ;
        decreases *;
    {
        i := i + 1;
    }
    /* To prove:*/
    assert i == n;
}
```

$$\{\ I \wedge C\ \}\ \text{BODY}\ \{\ I\ \}$$

$$\{\ I\ \}\ \text{While}\ C\ \text{do}\ \text{BODY}\ \text{od}\ \{\ I \wedge \neg C\ \}$$

```
 i <= n  Holds initially? ✅
     (i == 0 && 0 <= n)

 i <= n  Loop invariant premise satisfied?
✅
assume(i0 <= n) && ( i0 < n )
 Fact 1: i0 < n
 i1 := i0 + 1; //   effect of body
 Fact 2: i0 < n <==>
      i0 + 1 <= n <==>  i1 <= n
```

Other Rules

$$\frac{\{P\}\ P1\ \{Q\} \wedge \{Q\}\ P2\ \{R\}}{\{P\}\ P1\ ;\ P2\ \{R\}}$$

Sequence

$$\frac{\{P \wedge C\}\ P1\ \{R\} \wedge \{P \wedge \neg C\}\ P2\ \{R\}}{\{P\}\ \text{if}\ C\ \text{then}\ P1\ \text{else}\ P2\ \{R\}}$$

If-then-else

Mechanical Proof of Loop Invariant Rule

```
method ex1(n: int)
    requires true
    ensures true
    decreases *
{
    var i := 0;
    while ( i < n )
        invariant i <= n ;
        decreases *;
    {
        i := i + 1;
    }
    /* To prove:*/
    assert i == n;
}
```

$$\{\ I \wedge C\ \}\ \text{BODY}\ \{\ I\ \}$$

$$\{\ I\ \}\ \text{While}\ C\ \text{do}\ \text{BODY}\ \text{od}\ \{\ I \wedge \neg C\ \}$$

```
 i <= n  Holds initially? ✅
     (i == 0 && 0 <= n)

 i <= n  Loop invariant premise satisfied?
✅
assume(i0 <= n) && ( i0 < n )
 Fact 1: i0 < n
 i1 := i0 + 1; //   effect of body
 Fact 2: i0 < n <==>
      i0 + 1 <= n <==>  i1 <= n
```

Termination

```
method ex1(n: int)
    requires true
    ensures true
    decreases *
{
    var i := 0;
    while ( i < n )
        invariant  i <= n ;
        decreases  n - i ;
    {
        i := i + 1;
    }
    /* To prove:*/
    assert i == n;
}
```

Loop Termination rule (well-founded order)

1. Define a measure m
2. Show that m bounded from below
3. Show that m strictly decreases

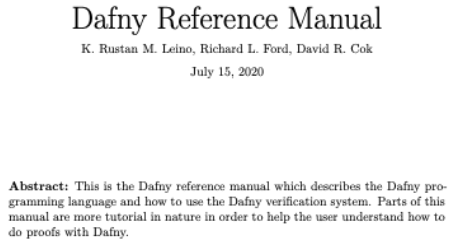
```
n - i bounded from below?
 Invariant : n - i >= 0? Yes.

n - i strictly decreasing?
 Initially: "n - i" = n - i0
 i1 := i0 + 1; //   effect of body
 After body: "n - i" = n - i1 = (n - i0)- 1
```

Example 2 – Find a key in an sequence (array)

```
method find(a: seq<int>, key: int) returns (index : int)
  requires true;
  ensures true
{
    index := 0;
    while (index < |a|)
        invariant true ;
        {
            if ( a[index] == key ) {
                ...
            }
            ...
        }
    ...
}
```

section 10.3, for operations on seq.



Dafny Reference Manual

K. Rustan M. Leino, Richard L. Ford, David R. Cok

July 15, 2020

Abstract: This is the Dafny reference manual which describes the Dafny programming language and how to use the Dafny verification system. Parts of this manual are more tutorial in nature in order to help the user understand how to do proofs with Dafny.

Task

1. Fill in the body
2. Compile/Execute
3. Write specs
4. Verify

Example 3 – Palindrome

```
A string s is a palindrome iff s == reverse(s)

Check whether a string (seq of chars) is a palindrome.

method isPalindrome(a: seq<char>) returns (b: bool)
    requires true
    ensures |a| <= 1 ⇒ isPalindrome(s)
    ensures ...
{
...
}
```

Task

1. Fill in the body
2. Compile/Execute
3. Write specs
4. Verify

Example 4 – Remove duplicates in sorted sequence

```
predicate sorted(a: seq<int>)
{
   forall j, k::0 <= j < k < |a|  ==> a[j] <= a[k]
}

method unique(a: seq<int>) returns (b: seq<int>)
   requires sorted(a)
   ensures true
{
...
}
```

Task

1. Fill in the body
2. Compile/Execute
3. Write specs
4. Verify