

# 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  $\{ Q \}$

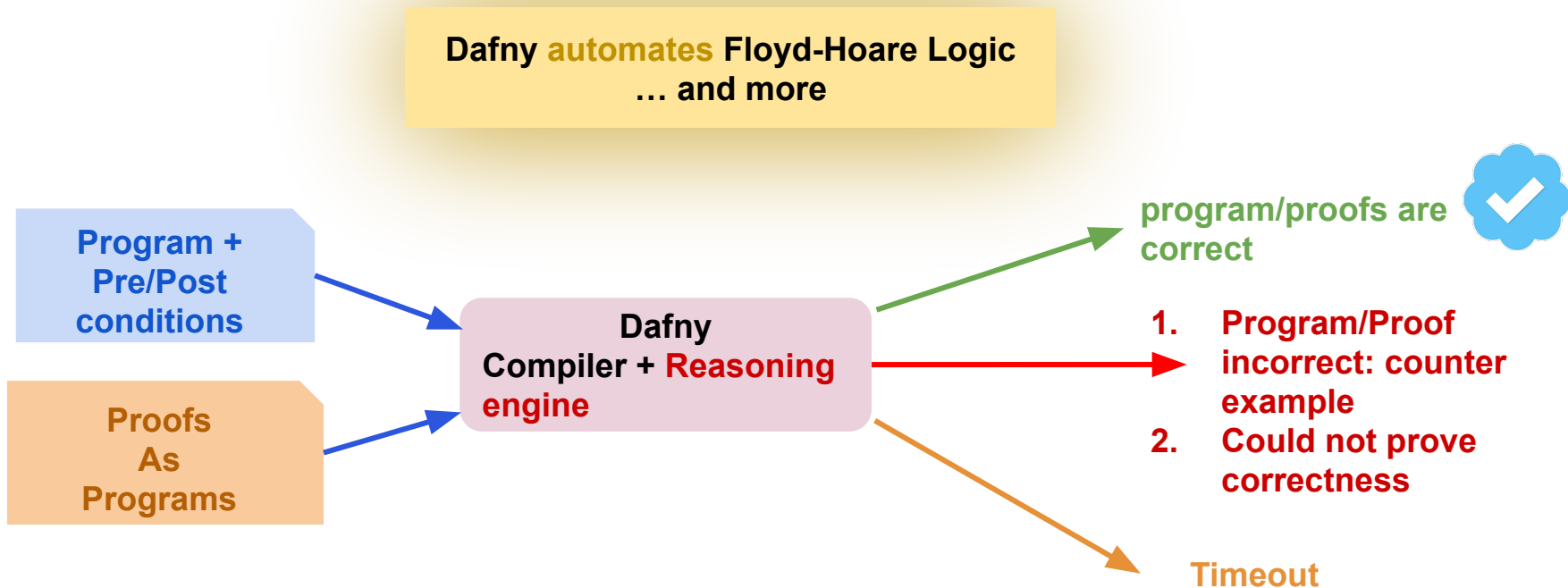
$\{ \text{true} \} x := 2 \{ x \geq 2 \}$  ✓

$\{ x \geq 0 \} x := x + 1 \{ x \geq 4 \}$

$\{ i \geq 0 \}$  while  $i > 0$  do  $i := i - 1$ ; od  $\{ i == 0 \}$  ✓

$\{ \text{true} \}$   
while true do skip; od  
 $\{ \text{false} \}$

# Dafny – A verification-aware programming language (1)




**Dafny automates**  
**Verification of Floyd-Hoare**  
**Logic specifications**  
**... and more**

Could not prove or disprove F  
Timeout

F is not valid  
counter-example

F is valid  
program/proofs are correct

# Checking Hoare Triples – SAT Problem

$\{ x \geq 1 \} x := x + 1 \{ x \geq 2 \}$  

$x\_1 \geq 1 \ \&\& \ x\_2 == x\_1 + 1 \ \&\& \ \text{not}(x\_2 \geq 2)$

UNSAT

$\{ x \geq 0 \} x := x + 1 \{ x \geq 4 \}$

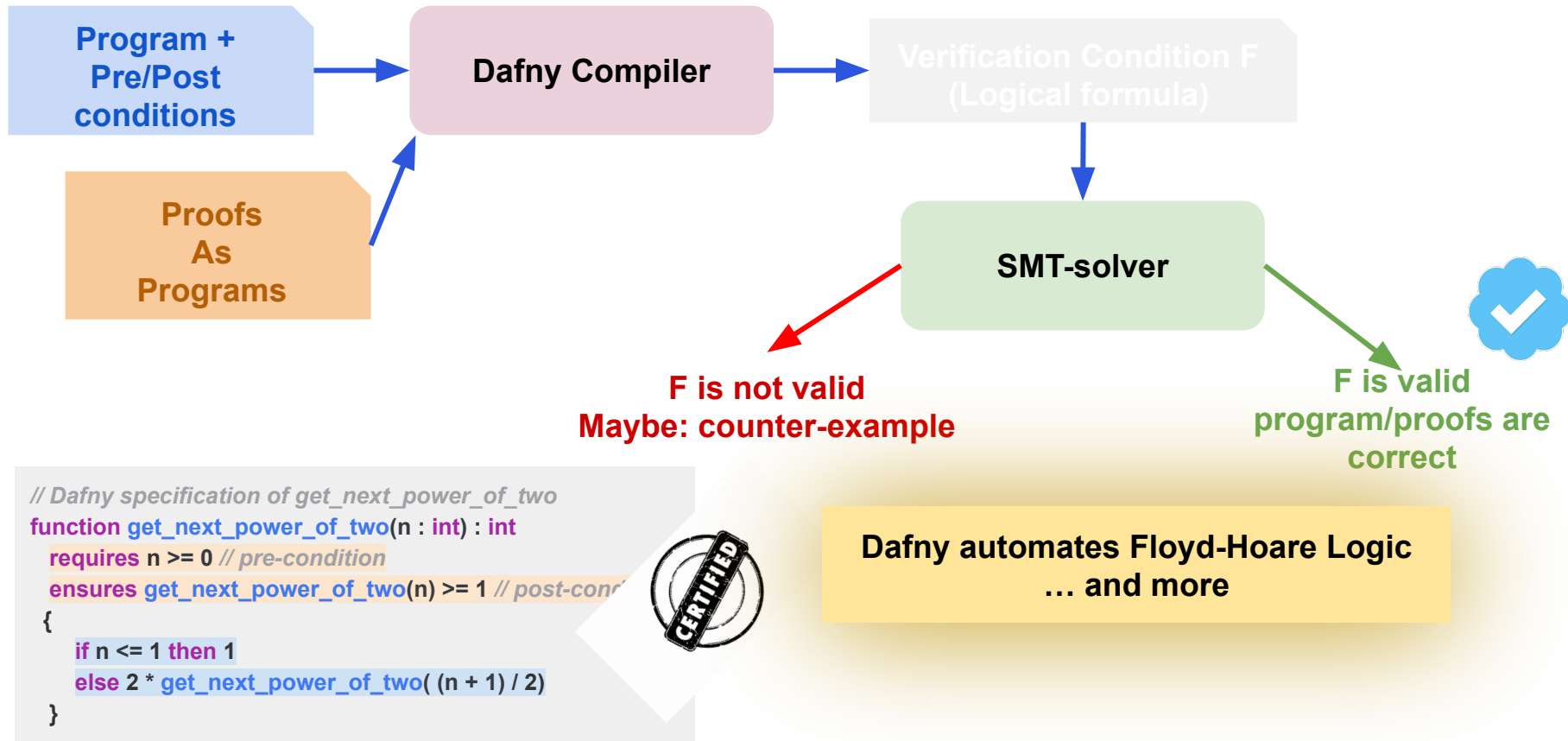
$x\_1 \geq 0 \ \&\& \ x\_2 == x\_1 + 1 \ \&\& \ \text{not}(x\_2 \geq 4)$

SAT

$x\_1 = 1$   
 $x\_2 = 2$



## Dafny – A verification-aware programming language (2)



# Examples in VSCode

## Basic CLI commands (Not needed with Dafny VSCode plugin)

#help

**dafny /help**

# Compile (in memory) and execute Main:

**dafny /noVerify /compile:4 training1.dfy**

# Verify a file, don't compile

**dafny /dafnyVerify:1 /compile:0 training1.dfy**

## 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  $\geq 0$
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

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

Conclusion

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

Ensure P implied  
by Inv and loop exit

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

# 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;
}
```

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

## Task

### 1. Add invariant to prove assert:

- Hint1:  $\text{not}(C) \Leftrightarrow i \geq n$
- Hint2:  $\text{Inv} \wedge (i \geq n) \Rightarrow i == n$

### 2. Termination: add decreasing measure to prove termination

- Hint1: bounded from below
- 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;
}
```

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

$i \leq n$  Holds initially?

$(i == 0 \ \&\& \ 0 \leq n)$

$i \leq n$  Loop invariant premise satisfied?



assume  $(i_0 \leq n) \ \&\& \ (i_0 < n)$

Fact 1:  $i_0 < n$

$i_1 := i_0 + 1$ ; // effect of body

Fact 2:  $i_0 < n \iff$

$i_0 + 1 \leq n \iff i_1 \leq n$

## Other Rules

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

$\{P\} P1 ; P2 \{R\}$

Sequence

$\{P \wedge C\} P1 \{R\} \wedge \{P \wedge \neg C\} P2 \{R\}$


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

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;
}
```

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

$i \leq n$  Holds initially? 

$(i == 0 \ \&\& \ 0 \leq n)$

$i \leq n$  Loop invariant premise satisfied? 

assume  $(i_0 \leq n) \ \&\& \ (i_0 < n)$

Fact 1:  $i_0 < n$

$i_1 := i_0 + 1$ ; // effect of body

Fact 2:  $i_0 < n \iff$

$i_0 + 1 \leq n \iff i_1 \leq n$

# 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 :  $n - i \geq 0$ ? Yes.

**n - i strictly decreasing?**

Initially:  $n - i = n - i_0$

$i_1 := i_0 + 1$ ; // effect of body

After body:  $n - i = n - i_1 = (n - i_0) - 1$

## 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  $s == \text{reverse}(s)$

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

```
method isPalindrome(a: seq<char>) returns (b: bool)
  requires true
  ensures |a| <= 1  $\Rightarrow$  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