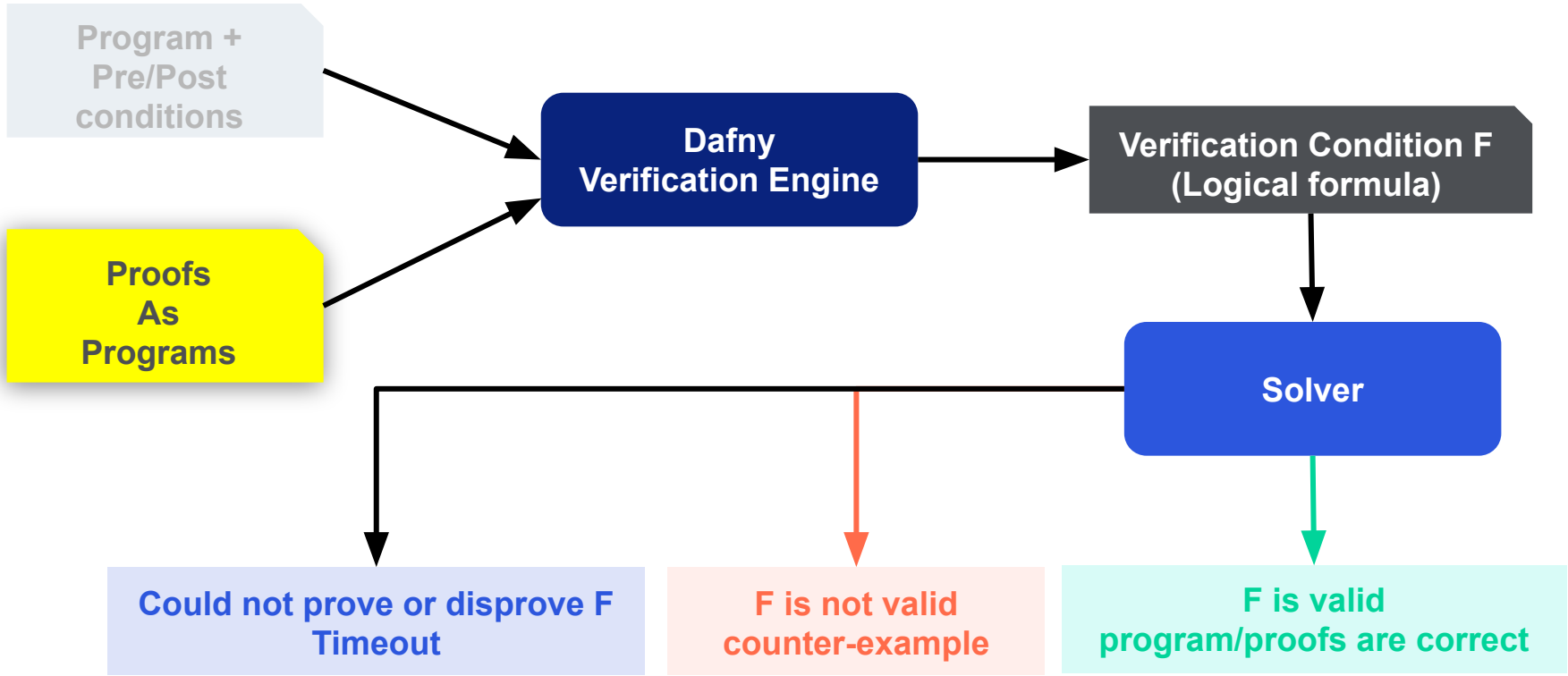
CONSENSYS

# Program Verification with Dafny (Part 2)

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

# Dafny – A *Verification-Friendly* Programming Language



# Objectives of this session

1. Write functional/recursive specifications
2. Write proofs as programs (lemmas)
   2.1. Calculations
   2.2. Induction
3. Define/Use Abstract Data Types: Trees
   3.1. Write type and (functional) algorithm
   3.2. Write proofs (induction)
4. More induction: Lists
   4.1. Define lists, length, append, reverse functions
   4.2. Prove idempotence theorem for reverse

# Examples in VSCode

# Basic CLI commands (Reminder)

#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**

Further reading for this session:
https://cseweb.ucsd.edu/~npolikarpova/publications/vstte13.pdf

# Example 0

```
function powerOf2(n: nat):
nat
    decreases n
{
    if n == 0 then
        1
    else
        2 * powerOf2(n - 1)
}
```

**Task 1**

1. Prove 2^n + 2^n = 2^(n + 1)

Use (Verified) Calculations

**Task 2**

1. Monotonicity: n <= m ⇒ 2^n <= 2^m
2. Identity1: 2^n * 2^m == 2^(n + m)

Use (Verified) Induction

# Example 1.a

```
datatype Tree =
      Leaf
    | Node(left: Tree, right: Tree)
```

**Task**

1. Define nodesCount(root)
2. Define leavesCount(root)
3. Prove nodesCount(root) <= 2^(height(root)) - 1
4. Prove leavesCount(root) <= 2^(height(root) - 1)

```
function height(root : Tree) : nat
    ensures height(root) >= 1
    decreases root
{
    match root
        case Leaf => 1
        case Node(lc, rc) => 1 + max(height(lc), height(rc))
}
```

# Example 1.b (Optional)

```
datatype Tree =
      Leaf
   |  Node(left: Tree, right: Tree)
```

**Task**

1. Define complete (or perfect) trees
2. Prove nodesCount(root) == 2^(height(root)) - 1
3. Prove leavesCount(root) == 2^(height(root) - 1)

# Example 2 – Inductive/Generic type

```
datatype List<T> =
     Nil
   | Cons(..., ...)
```

**Task**

1. Define lists
2. Define length of lists
3. Define append/concatenate two lists
4. Define reverse list
5. Prove
   reverse(append(l1, l2)) == append(reverse(l2), reverse(l1))
6. Prove reverse(reverse(l)) == l