

The Dafny Programming Language and Static Verifier

Stefan Zetsche

Amazon Web Services

March 8, 2024

1 Introduction

Dafny Example

[Live Demo](#)

Dafny

- A *verification-aware* programming language
 - Familiar object-oriented constructs
 - Deeply integrated automated reasoning
- Two parts to a program in Dafny
 - *What* should it do?
 - *How* should it do it?
- Both written in the same language
- Automatically confirm that they agree
 - Equivalent to testing on an infinite number of cases

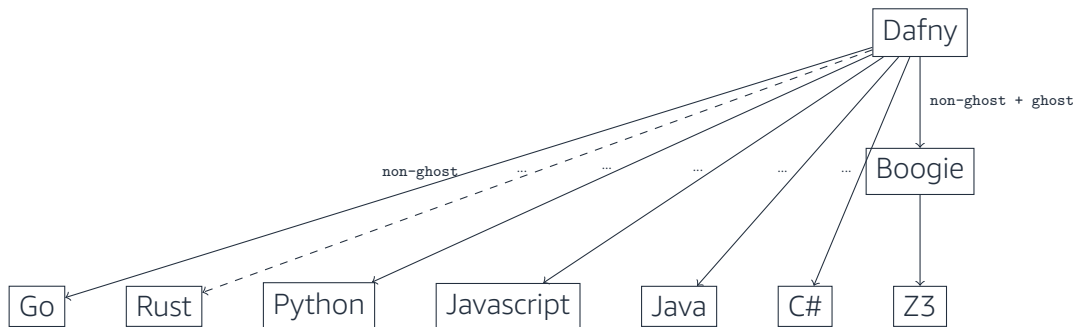


Benefits of Verification

- Higher *confidence* of correctness with fewer tests
 - Focus on integration tests
- Fearless *refactoring*
- Clear *understanding* of design
 - Less ambiguous than design doc
 - Maintainable along with implementation
- Key caveat: moves quality assurance effort *earlier*
 - But can save *overall* effort, reducing bug fixing time later

Multi-Target Compilation

- Most Dafny programs: verified *components* of larger systems
 - Compilation to many targets to support this



Dafny Use Cases

POPL24 Sun 14 - Sat 20 January 2024
London, United Kingdom

Attending • Program • Tracks • Organization • Search • Notes •

#POPL24 January | Dafny 2024 (closed)

Dafny 2024

About Program Accepted Papers Social Call for Papers

Accepted Papers

• Title

- 1. Cesium: A Verifier for Probabilistic Programs
Philip Schuster, Aaron Bass, Benjamin-Lucien Nannan, José-Pedro Katoen, Christoph Hahn
- 2. CLOVER: Closed-Loop Verifiable Code Generation
Chang Sun, Ying Sheng, Chao-Peng, Qing Chen
- 3. Counting Flags with Dafny & SMT
Jan de Muijck-Hughe, James Hulse
- 4. Dafny Test Demonstration
Alexander Fiedler, Jeffrey B. Foster, Dominik Heintze, Andrei Sorb
- 5. Day closing
Boris Zbarsky, Joseph Tassarotti
- 6. Demarcating Automation
Pavel Simion, Chao-Peng, Jan Heule, Andrei Laftau
- 7. Enhancing Proof Stability
Ben McLaughlin, George-Jack Jorgensen, Tingting Wang, Peter Haber
- 8. Generation of Verified Assembly Code Using Dafny and Reinforcement Learning
Christopher Wu, Jean-Baptiste Viot
- 9. Improving the Stability of Type Safety Proofs in Dafny
Joseph W. Gulte, Michael Noll, Emma Tiede
- 10. Incremental Proof Development in Dafny with Module-Based Induction
Soyun, Delbert P. Cloud
- 11. Learn: Verifying Dafny
James Hulse
@james_hulse
- 12. Portfolio Solving for Dafny
Erik Hugueny, Ben McLaughlin, Aaron Bass
- 13. Randomized Testing of the Dafny Compiler
Austen H. Davidson, Omer Sheth, Jean-Baptiste Viot, Alex Lerner
- 14. Teaching Logic and Set Theory with Dafny
Hao Wang, Neil Davies
- 15. Testing Specifications in Dafny
Björn Bräutigam, Stefan Yu, Aaron VMC, Niklas Rujter
- 16. Towards the verification of a generic infrastructure layer: Dafny meets parameterized model checking
Remond Deneck, Martin Dreyer, Bernhard Hahn
- 17. Verifying a concurrent file system with sequential reasoning
Yu Chen
- 18. Verifying Dafny Contract Integrity
Cecily Vestly, Eric Mamer
- 19. VMC: a Dafny Library for Verified Monte Carlo Algorithms
Felix Jost, Boris Zbarsky, Jean-Baptiste Viot

Important Dates

POPL 2024 (14-20 Jan)
Sun 14 Jan 2024
Wed 16 Jan 2024
Wed 18 Jan 2024
Wed 18 Jan 2024

Submission Link

<https://dafny.sigplan.org>

Program Chairs

Joseph Tassarotti
NPU
United States
Program Chair

Boris Zbarsky
Amazon Web Services
United States
Program Chair

Program Committee

Alexis Chikula
Massachusetts Institute of Technology
United States
Program Committee

John Christopher Filmer
ONRS
France
Program Committee

Andreas Laftau
Weber Research
Germany
Program Committee

Stefan Li
NPU
United States
Program Committee

Peter Müller
ETH Zurich
Switzerland
Program Committee

Nadia Polubarsky
University of California at San Diego
Program Committee

POPL24 Dafny [Dafny24] Improving the Stability of Type Safety Proofs in Dafny 95 Author - ver 4 Tager	POPL24 Dafny [Dafny24] Colouring Flags with Dafny & Idits 19 Author - ver 4 Tager	POPL24 Dafny [Dafny24] VMC: a Dafny Library for Verified Monte Carlo Algorithms 15 Author - ver 4 Tager	POPL24 Dafny [Dafny24] Cesium: A Verifier for Probabilistic Programs 9 Author - ver 4 Tager
POPL24 Dafny [Dafny24] Verifying Dafny Contract Integrity 10 Author - ver 4 Tager	POPL24 Dafny [Dafny24] Testing Specifications in Dafny 17 Author - ver 4 Tager	POPL24 Dafny [Dafny24] Generation of Verified Assembly Code Using Dafny and Reinforcement Learning 31 Author - ver 4 Tager	POPL24 Dafny [Dafny24] CLOVER: Closed-Loop Verifiable Code Generation 10 Author - ver 4 Tager
POPL24 Dafny [Dafny24] Day closing 9 Author - ver 4 Tager	POPL24 Dafny [Dafny24] Learn: Verifying Dafny 23 Author - ver 4 Tager	POPL24 Dafny [Dafny24] Teaching Logic and Set Theory with Dafny 28 Author - ver 4 Tager	POPL24 Dafny [Dafny24] Randomised Testing of the Dafny Compiler 14 Author - ver 4 Tager

<https://popl24.sigplan.org/home/dafny-2024>

<https://www.youtube.com/playlist?list=PLyr1k8Xaylp4LXDucxBhCGanPBArfmk7Q>

Dafny Use Cases at Amazon

- Authorization
- Cryptography
 - AWS Encryption SDK
 - AWS Cryptographic Material Providers Library
- Differential Privacy
 - Probabilistic Samplers and Shuffling

<https://github.com/aws/aws-encryption-sdk-dafny>

<https://github.com/aws/aws-cryptographic-material-providers-library>

<https://github.com/dafny-lang/Dafny-VMC>

Table of Contents

Introduction

Dafny as a Programming Language

Dafny for the Verification of Programs

Dafny as a Research Assistant

2 Dafny as a Programming Language

Multi-Paradigms

- Supports familiar concepts
 - polymorphism
 - inductive datatypes
 - `datatype` `List<T> = Nil | Cons(head: T, tail: List)`
 - coinductive datatypes
 - `codatatype` `Lang<T> = Alpha(eps: bool, delta: T -> Lang)`
 - lambda expressions
 - `var` `f: int --> int := x requires x > 0 => x-1`
 - higher-order functions
 - `function` `F(f: int -> int -> int, n: int): int -> int { f(n) }`
 - while-loops
 - classes with mutable state, traits
 - ...
- Easy to adapt to by engineers

3 Dafny for the Verification of Programs

3.1 Dependent Verification of Functional Programs

Intertwining Ghost and Non-Ghost

```
function Fib(n: nat): nat {  
  if n <= 1 then n else Fib(n - 1) + Fib(n - 2)  
}  
  
method ComputeFib(n: nat) returns (fib: nat)  
  ensures fib == Fib(n)  
{  
  var i, currentFib, nextFib := 0, 0, 1;  
  while i < n  
    invariant i <= n && currentFib == Fib(i) && nextFib == Fib(i + 1)  
    {  
      i, currentFib, nextFib := i + 1, nextFib, currentFib + nextFib;  
    }  
  return currentFib;  
}
```

Limitations of Dependent Verification

```
function Append(xs: List, ys: List): List
//ensures forall zs :: Append(Append(xs, ys), zs) == Append(xs, Append(ys, zs))
{
  match xs
  case Nil => ys
  case Cons(head, tail) => Cons(head, Append(tail, ys))
}
```

3.2 Independent Verification of Functional Programs

Conditional

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

```
lemma AbsPositive(x: int)  
  ensures Abs(x) >= 0  
{  
  if x < 0 {  
    assert -x > 0;  
  } else {  
    assert x >= 0;  
  }  
}
```

Recursion and Induction

```
function Append(xs: List, ys: List): List {  
  match xs  
    case Nil => ys // recursion base  
    case Cons(head, tail) => Cons(head, Append(tail, ys)) // recursion step  
}
```

```
lemma AppendLength(xs: List, ys: List)  
  ensures Length(Append(xs, ys)) == Length(xs) + Length(ys)  
{  
  match xs  
    case Nil => // induction base  
    case Cons(head, tail) => AppendLength(tail, ys); // induction step  
}
```

Proofs as Programs

- *Universal* Quantification

```
lemma ForAll()  
  ensures forall n: int :: P(n)
```

```
lemma ForAllAlternative(n: int)  
  ensures P(n)
```

- *Existential* Quantification

```
lemma Exists()  
  ensures exists n: int :: Q(n)
```

```
lemma ExistsAlternative() returns (n: int)  
  ensures Q(n)
```

3.3 Structured Proofs

Conjunction

```
lemma ProofOfConjunction() {  
  assert A && B by {  
    assert A by {  
      // Proof of A  
    }  
    assert B by {  
      // Proof of B  
    }  
  }  
}
```

Contradiction

```
lemma ProofByContradiction() {  
  assert B by {  
    if !B {  
      assert false by {  
        // Proof of false  
      }  
    }  
  }  
}
```

Calculations

```
lemma UnitIsUnique(bop: (T, T) -> T, unit1: T, unit2: T)
  requires A1: forall x :: bop(unit1, x) == x
  requires A2: forall x :: bop(x, unit2) == x
  ensures unit1 == unit2
{
  calc {
    unit1;
  == { reveal A2; }
    bop(unit1, unit2);
  == { reveal A1; }
    unit2;
  }
}
```

3.4 Termination

Termination Metrics

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

Termination Metrics

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

Termination Metrics

```
function SumFromTo(m: nat, n: nat): nat
  requires m <= n
{
  if m == n then
    n
  else
    m + SumFromTo(m+1, n)
}
```

Termination Metrics

```
function SumFromTo(m: nat, n: nat): nat
  requires m <= n
  decreases n - m
{
  if m == n then
    n
  else
    m + SumFromTo(m+1, n)
}
```

3.5 Verification of Imperative Programs

Dynamic Frames and Counterexamples

Live Demo

4 Dafny as a Research Assistant

Big Step Semantics

Syntax

$$c \in \text{cmd} ::= \text{inc} \mid c_0; c_1 \mid c^*$$

Semantics

$$\frac{t = s + 1}{s \xrightarrow{\text{inc}} t} \quad \frac{s \xrightarrow{c_0} s' \quad , \quad s' \xrightarrow{c_1} t}{s \xrightarrow{c_0; c_1} t} \quad \frac{t = s}{s \xrightarrow{c^*} t} \quad \frac{s \xrightarrow{c} s' \quad , \quad s' \xrightarrow{c^*} t}{s \xrightarrow{c^*} t}$$

$\rightarrow \subseteq \text{state} \times \text{cmd} \times \text{state}, \quad \text{state} = \text{int}$

Big Step Semantics in Dafny

```
datatype cmd = Inc | Seq(cmd, cmd) | Repeat(cmd)

type state = int

least predicate BigStep(s: state, c: cmd, t: state) {
  match c
  case Inc =>
    t == s + 1
  case Seq(c0, c1) =>
    exists s' :: BigStep(s, c0, s') && BigStep(s', c1, t)
  case Repeat(c0) =>
    (t == s) || (exists s' :: BigStep(s, c0, s') && BigStep(s', Repeat(c0), t))
}

least lemma Increasing(s: state, c: cmd, t: state)
  requires BigStep(s, c, t)
  ensures s <= t
{}
```

The End

<https://dafny.org/>