Industrial experience with a verification-aware programming language

K. Rustan M. Leino (he/him)

Sr Principal Applied Scientist
Amazon Web Services

iFM 2023
Leiden, NL
14 Nov 2023

My talk

Reflect on

- Lessons learned from working with engineers to verify software
- How this experience has influenced the language and its tooling

Automated Reasoning at AWS

Use of sound logical tools and techniques to prove properties of software

To have lasting impact,
the tools must be applied with every code check-in

To scale,
the tools must be used by people outside the Automated Reasoning Group

Dafny

Verification-aware programming language

- Java-like language
- Designed to support formal verification

Coming up on 16 years

Open source

Dafny for every engineer

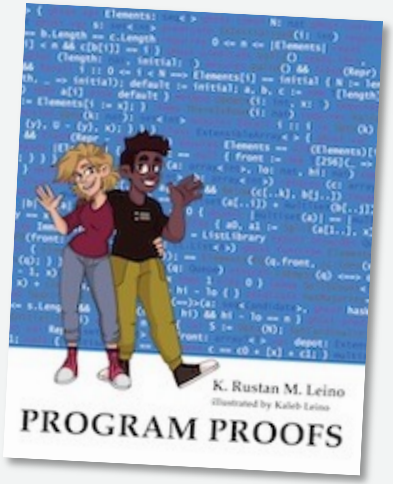
Is a programming language

Specifications are part of the language
    Not a bolt-on to non-verification language

Targets programmers
    Not type-theorists or logicians
    Uses curly braces

Used in teaching for 15+ years
    New book: *Program Proofs* (MIT Press)

Auto-active verification (interactive + automated)
    Centers on programs
    Proofs are part of the program text



amazon | science

Demo

FindLast

aws

Public uses of Dafny at AWS

AWS Encryption SDK

AWS Database Encryption SDK

Cedar authorization policy engine

Dafny leverage points

Write verified code

Write and verify once, compile to many

Abstract modeling

Dafny design: program expressions, specification expressions

They are the same

- Same syntax
- Same semantics

```
method FindLast<X>(arr: array<X>, x: X) returns (result: int)
  requires exists i :: 0 <= i < arr.Length && arr[i] == x
```

```
var b := exists i :: 0 <= i < arr.Length && arr[i] == x;
```

Demo…

Dafny design: program expressions, specification expressions

They are the same

- Same syntax
- Same semantics

Not all expressions are evaluated at run time

- Ghost declarations
- Specifications are always checked statically

```
var d := x + y + 1;
ghost var g := d + x;
```

```
ghost function IsBalanced(t: Tree): bool
```

Dafny design: expressions vs statements

Expressions
- deterministic
- do not modify the program state
- terminate

Functions
- body is expression
- behave like in mathematics

```
function Increase(x: nat): nat {
  var d := x + x;
  d + 1
}
```

Statements
- can be nondeterministic
- can modify the state
- can be specified to allow non-termination

Methods
- body is statement list

```
method Increase(x: nat) returns (r: nat) {
  var d := x + x;
  return d + 1;
}
```

Dafny design: importance that keywords convey right meaning

Example: Want statement that is
checked at run time (like `assume E;` in some languages)
assumed by verifier to hold (like `assume E;`)

```
expect E;
```

Keywords for Compiled vs. Ghost

	Compiled	Ghost
Variable	`var`	
Function	`function`	
Method	`method`	

Keywords for Compiled vs. Ghost

	Compiled	Ghost
Variable	`var`	`ghost var`
Function	`function`	
Method	`method`	

Keywords for Compiled vs. Ghost

	Compiled	Ghost
Variable	`var`	`ghost var`
Function	`function`	`ghost function`
Method	`method`	

Keywords for Compiled vs. Ghost

	Compiled	Ghost
Variable	`var`	`ghost var`
Function	`function`	`ghost function`
Method	`method`	`ghost method`

Keywords for Compiled vs. Ghost

	Compiled	Ghost
Variable	`var`	`ghost var`
Function	`function`	`ghost function`
Method	`method`	~~`ghost method`~~ `lemma`

Keywords for Compiled vs. Ghost

	Compiled	Ghost
Variable	`var`	`ghost var`
Function	~~`function method`~~ `function`	~~`function`~~ `ghost function`
Method	`method`	`lemma`

Developer expectations: Dafny ecosystem

Language
Compiler(s)
Verifier
Documentation, training
IDEs
Standard library
Build system
Testing tools
Foreign function interface
Linters
Verification debugger
…

amazon | science

Influence from customers

Unicode support

Handling failures

- Failure-compatible types

Change of definite-assignment rules

- stricter than required by sound verification
- expected by programmers, and catches common errors

Simplify for customers: loop alternatives

Demo…

Simplify for customers: Auto-accumulator tail recursion

```
function Filter<T>(s: seq<T>, p: T -> bool): seq<T> {
  if s == [] then
    []
  else if p(s[0]) then
    [s[0]] + Filter(s[1..], p)
  else
    Filter(s[1..], p)
}
```

Experience: Specifications

The process of writing specifications uncovers design bugs

Writing specifications is hard

- Better specifications are usually more abstract

Demo

SplitString

aws

Foreign-function interface (extern code)

Writing specifications for extern code is even harder

“Verification finds all bugs” can be misunderstood

Foreign-function interface: difficulty

```
method {:extern} Concat<X>(a: array<X>,
                           b: array<X>,
                           limit: int) returns (r: array<X>)
  ensures r[..] == (a[..] + b[..])[..Max(0, Min(|a| + |b|, limit))]
  ensures fresh(r)
```

But perhaps the extern method

- expects `limit` to be non-negative
- returns `a` or `b` if the other is empty
- returns `null` in some cases

Foreign-function interface: difficulty

```
method {:extern "Logger.Append"} LogEvent(s: string)
  ensures log.data == old(log.data) + [s]
```

```
method {:extern "Logger.Append"} LogEvent(s: string)
  modifies log
  ensures log.data == old(log.data) + [s]
```

Foreign-function interface: difficulty

```
function {:extern “System.DateTime.Now”} GetTime(): Time
```

```
method {:extern “System.DateTime.Now”} GetTime() returns (t: Time)
```


amazon | science

Foreign-function interface (extern code)

How to avoid errors in extern specifications?

- → auditor tool
- → `expect` statements
- → run-time specification checking
- → “bland externs”

Proofs

For programmers

Demo

Wildcard matching: declarative vs operational

aws

Automation

Early days of Dafny:

Automation always

Then:

Added repertoire of proof-authoring constructs

Now:

Favor stability over automation

Still need:

Helpful tools for proof construction

Helpful tools for verification debugging

Educate more

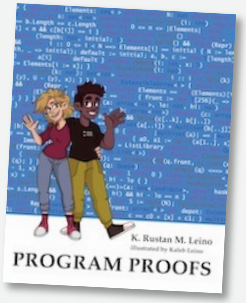
Conclusions

Programming with specifications and proofs, in practice

- Listen to customer complaints
- Don’t be too defensive
- Innovate on behalf of customers

Need more automated-reasoning savvy users

- Teach!



dafny.org
program-proofs.com

amazon | science