Verification beyond programs

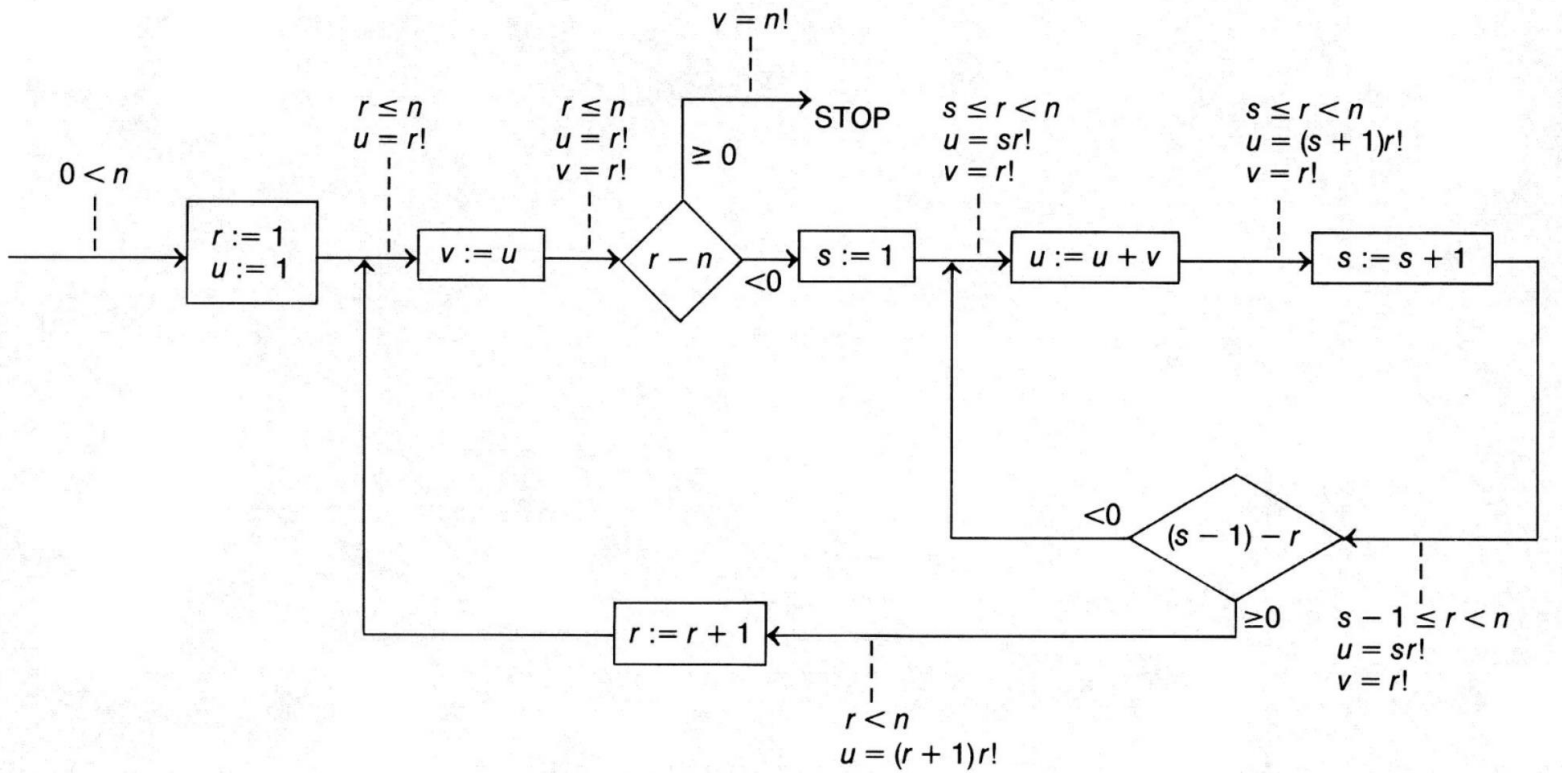
K. Rustan M. Leino

Principal Researcher
Research in Software Engineering (RiSE), Microsoft Research, Redmond

Visiting Professor
Department of Computing, Imperial College London

Workshop on Software Correctness and Reliability, 8 October 2016, ETH Zurich, Switzerland

Program verification: 1949



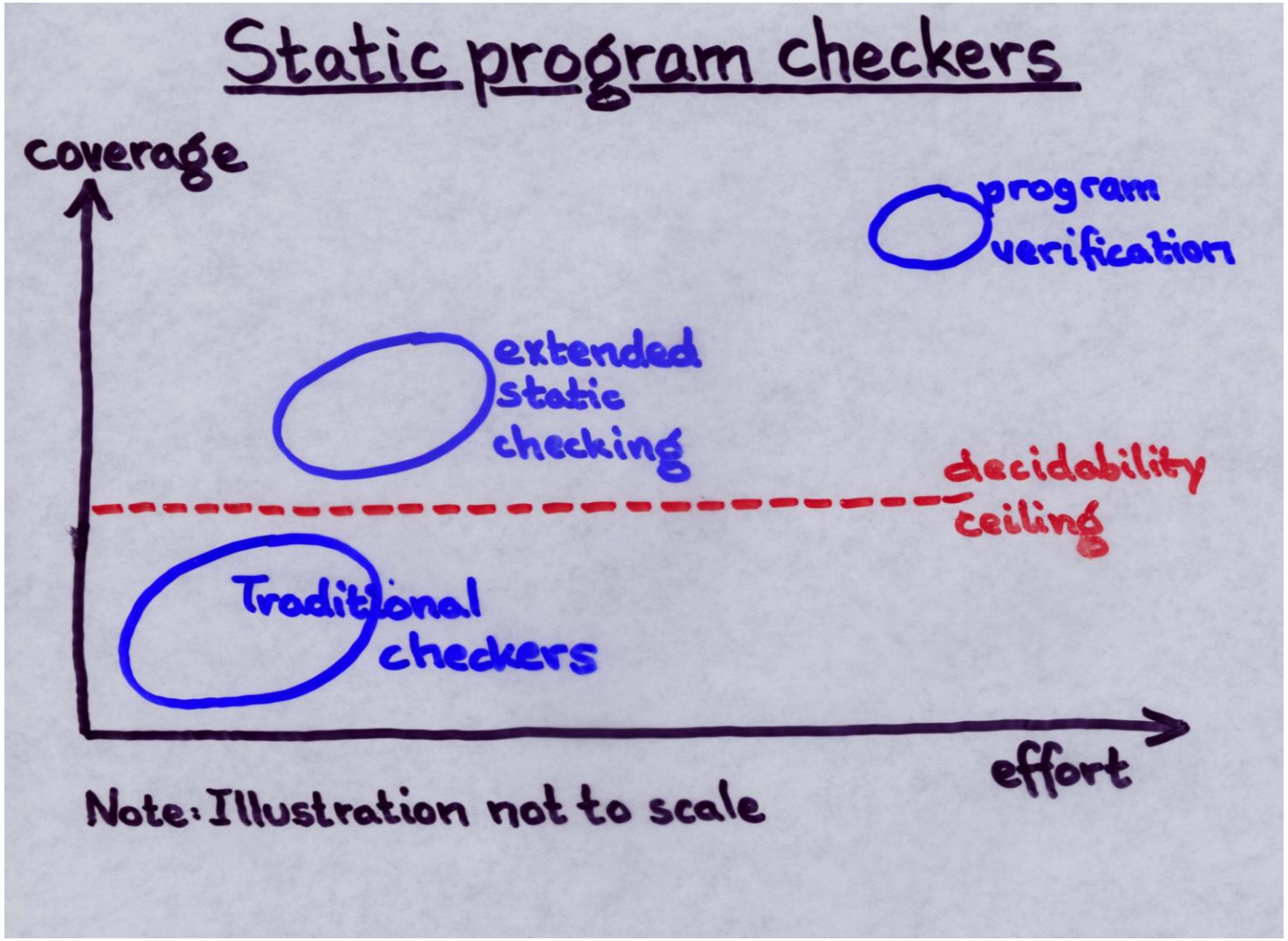
Source: “An Early Program Proof by Alan Turing” by F.L. Morris and C.B. Jones, Annals of the History of Computing, 1984

Program verification: 1971

```
begin
  comment  This program operates on an array A[1:N], and a
    value of f(1 ≤ f ≤ N). Its effect is to rearrange the elements
    of A in such a way that:
      ∀p,q(1≤p≤f≤q≤N⊃A[p]≤A[f]≤A[q]);
  integer m, n;  comment
      m ≤ f & ∀p,q(1≤p<m≤q≤N⊃A[p]≤A[q]),
      f ≤ n  & ∀p,q(1≤p≤n<q≤N⊃A[p]≤A[q]);
    m := 1;  n := N;
  while m < n do
  begin integer r, i, j, w;
    comment
        m ≤ i & ∀p(1≤p<i⊃A[p]≤r),
        j ≤ n  & ∀q(j<q≤N⊃r≤A[q]);
      r := A[f];  i := m;  j := n;
    while i ≤ j do
    begin while A[i] < r do i := i + 1;
      while r < A[j] do j := j − 1
      comment  A[j] ≤ r ≤ A[i];
      if i ≤ j then
      begin w := A[i];  A[i] := A[j];  A[j] := w;
        comment  A[i] ≤ r ≤ A[j];
        i := i + 1;  j := j − 1;
      end
    end increase i and decrease j;
    if f ≤ j then n := j
  else if i ≤ f then m := i
    else go to L
  end reduce middle part;
L:
end Find
```

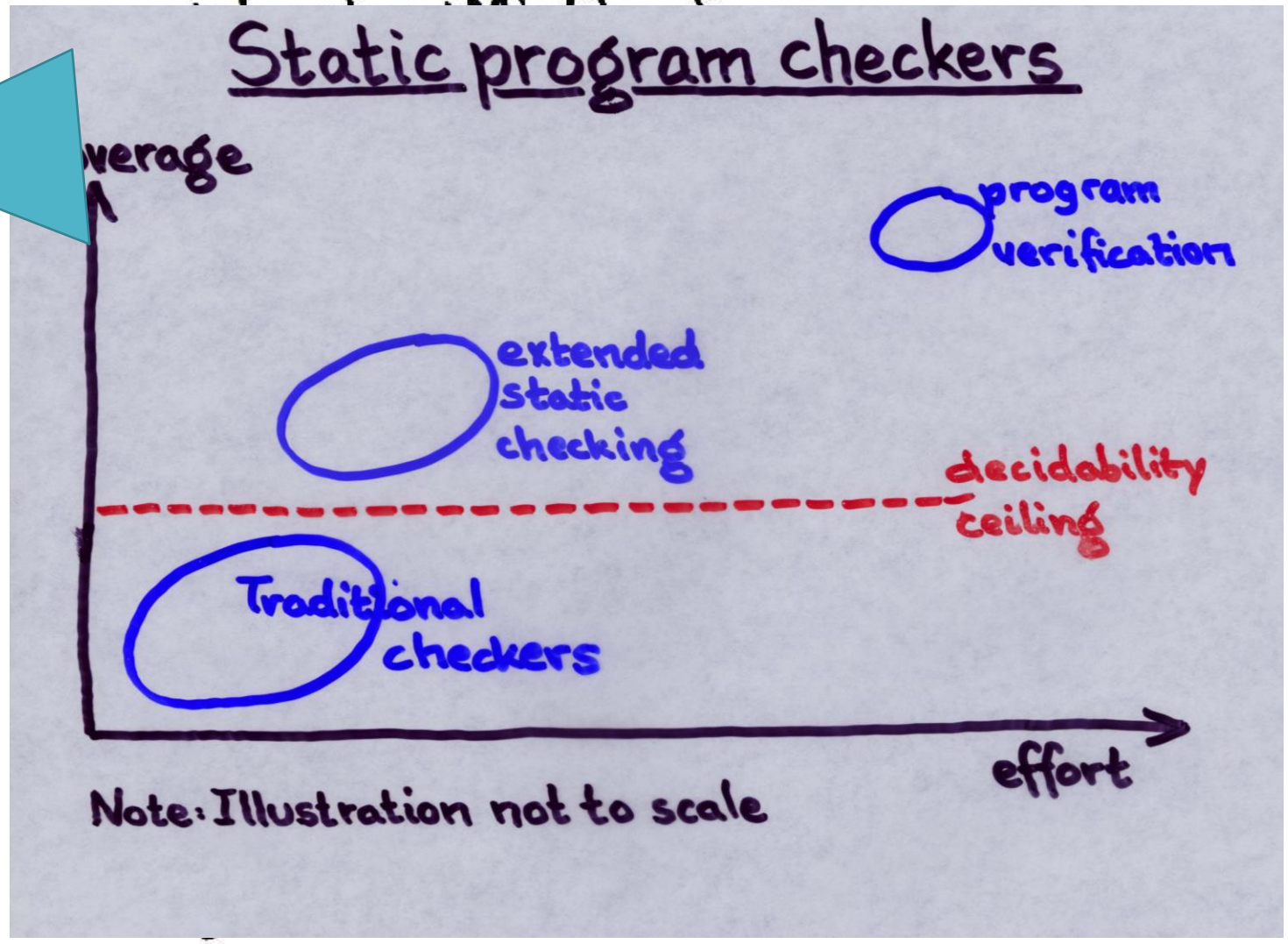
Source: “Proof of a program: FIND” by C.A.R. Hoare, CACM, 1971

Program verification: 1998



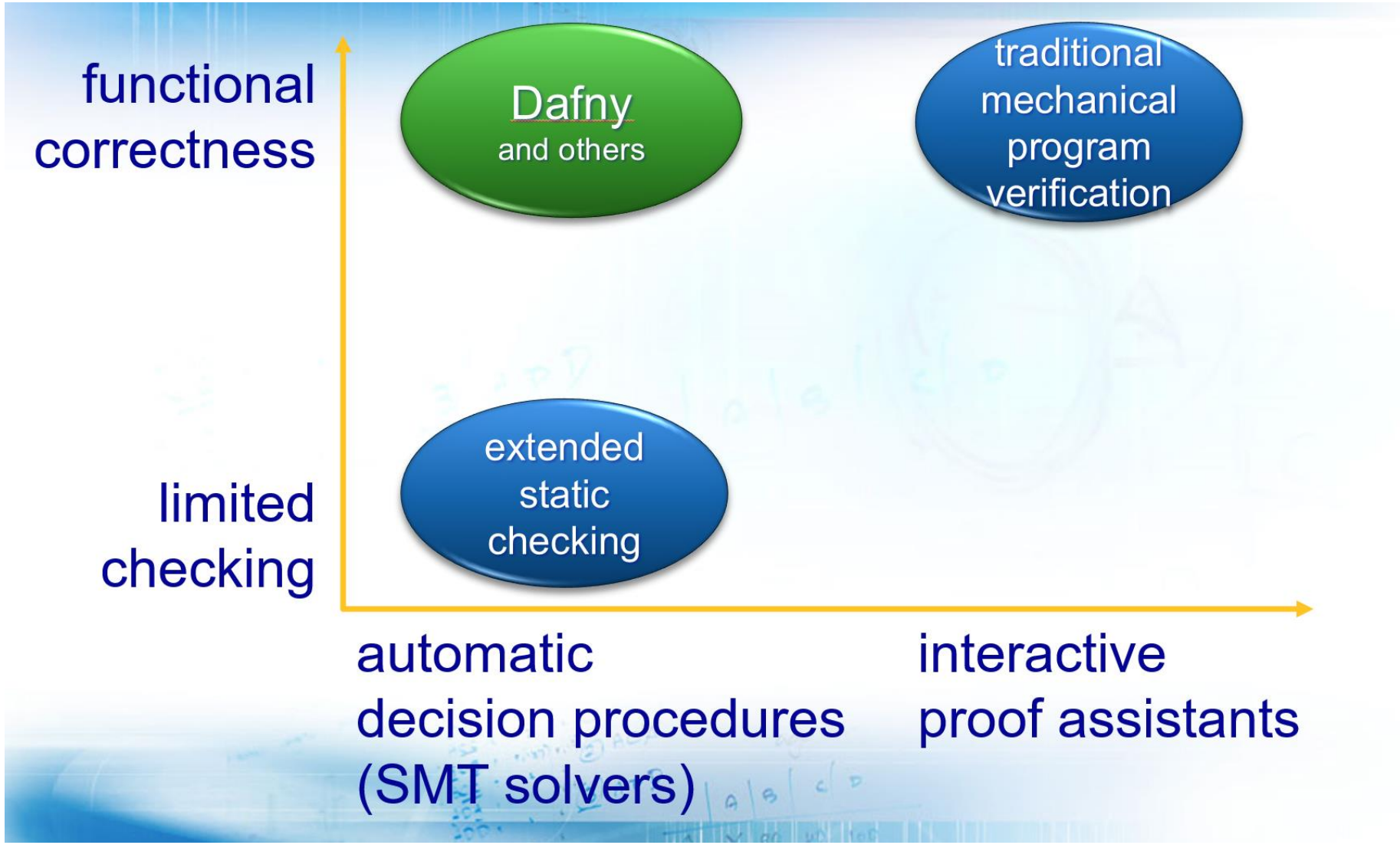
Source: Invited talk “ESC/Java” by K.R.M. Leino at Larch User’s Group Meeting, FM’99, Toulouse, France, Sep 1999

Program verification: 1998



Source: Talk and first (pre-tool) ESC/Java demo by K.R.M. Leino to Hopper et al. at DEC WRL, “Extended Static Checking for Java”, March 1998

Program verification: 2008



Source: Paper presentation on Dafny by K.R.M. Leino at LPAR-16, Dakar, Senegal, April 2010

Program verification: 2016

- Interactive proof assistants:
  - Applications in program verification
- Automated program verifiers:
  - Growth into meta mathematics
  - Uses as more general proof assistants

Photo credits: www.publicdomainpictures.net

Correctness of systems

CompCert

seL4 Verified

Ironclad, IronFleet

…

Demo

Iteration, induction, lemmas

Language illustration: INC

$$\mathrm{Cmd} ::= \mathrm{Inc} \mid \mathrm{Cmd}\,;\mathrm{Cmd} \mid \mathrm{Repeat}(\mathrm{Cmd})$$

Semantics given by the "big step" relation

$$(\mathrm{Cmd},\ \mathrm{State}) \rightarrow \mathrm{State}$$

where

$$(C,\ s) \rightarrow t$$

says that

there is an execution of command C from state s that terminates in state t

Semantics of INC

$\text{Cmd} ::= \text{Inc} \mid \text{Cmd}; \text{Cmd} \mid \text{Repeat}(\text{Cmd})$

$$\frac{t = s+1}{(\text{Inc}, s) \to t}$$

$$\frac{(c0, s) \to s' \quad (c1, s') \to t}{(c0; c1,\ s) \to t}$$

$$\frac{t = s}{(\text{Repeat}(body), s) \to t}$$

$$\frac{(body, s) \to s' \quad (\text{Repeat}(body), s') \to t}{(\text{Repeat}(body), s) \to t}$$

Semantics of INC

$$\text{Cmd} ::= \text{Inc} \mid \text{Cmd}\,;\,\text{Cmd} \mid \text{Repeat}(\text{Cmd})$$

$$\frac{t = s{+}1}{(\text{Inc}, s) \to t}$$

$$\frac{\exists\, s'.\ (c0, s) \to s' \quad (c1, s') \to t}{(c0\,;\,c1,\ s) \to t}$$

$$\frac{t = s}{(\text{Repeat}(body), s) \to t} \qquad \frac{\exists\, s'.\ (body, s) \to s' \quad (\text{Repeat}(body), s') \to t}{(\text{Repeat}(body), s) \to t}$$

Demo

INC

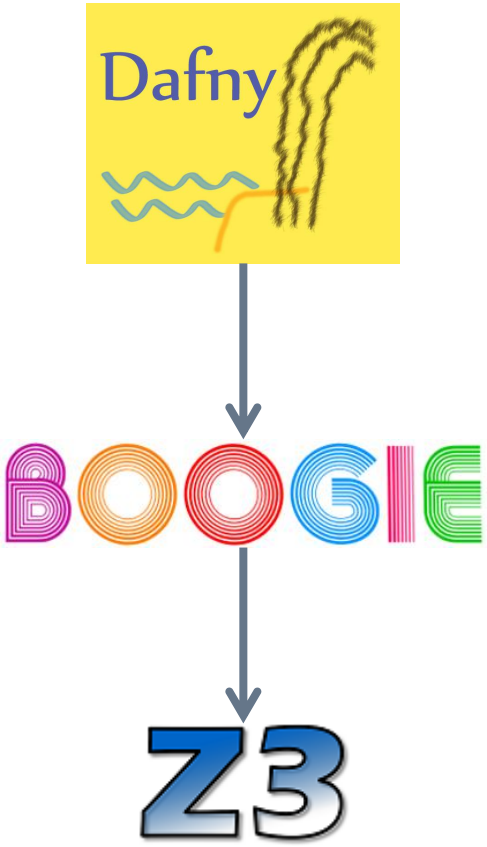
The recurrence equation

$$BigStep = \mathcal{F}(BigStep)$$

has many solutions in $BigStep$

We want the *least* solution

Verification tool architecture



Source language

Intermediate verification language (IVL)

Satisfiability modulo theories (SMT) solver

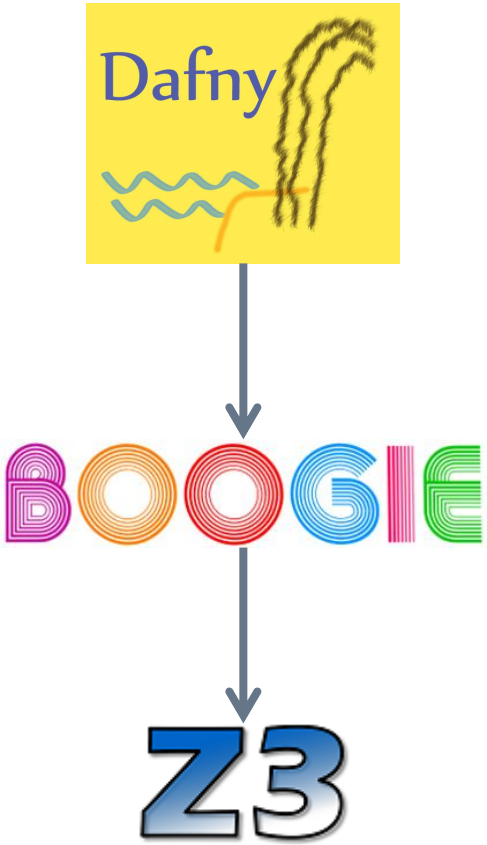
BENEFITS:

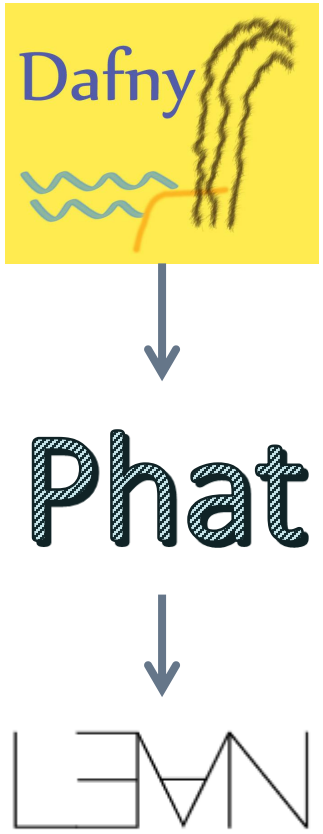
- Separation of concern
- Readable
- Editable

Show and tell

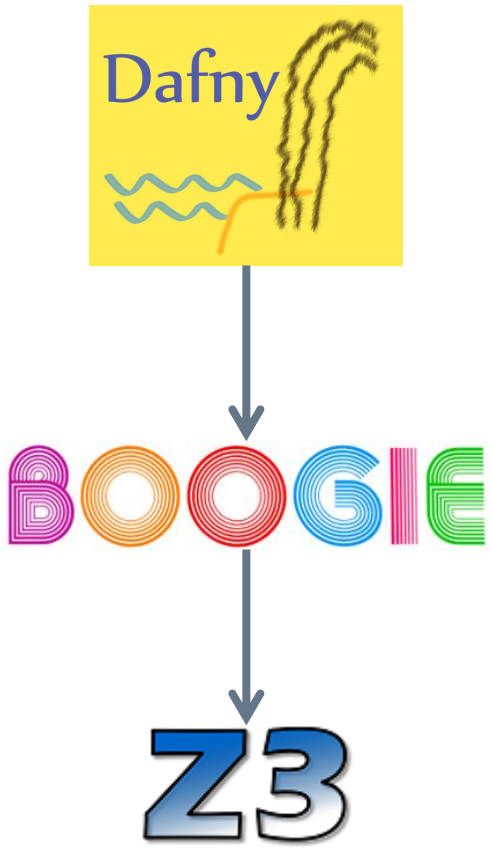
Source, IVL, SMT

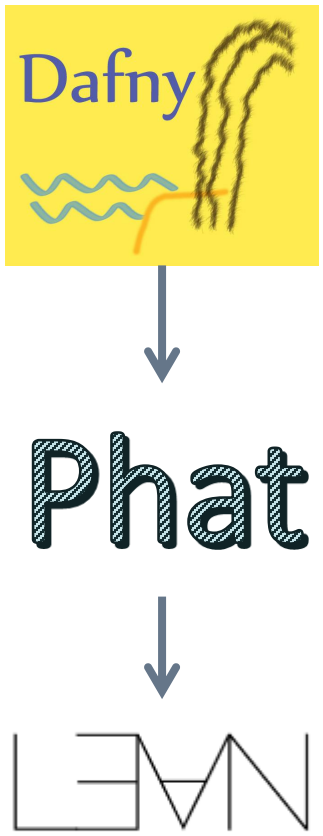
Intermediate verification language





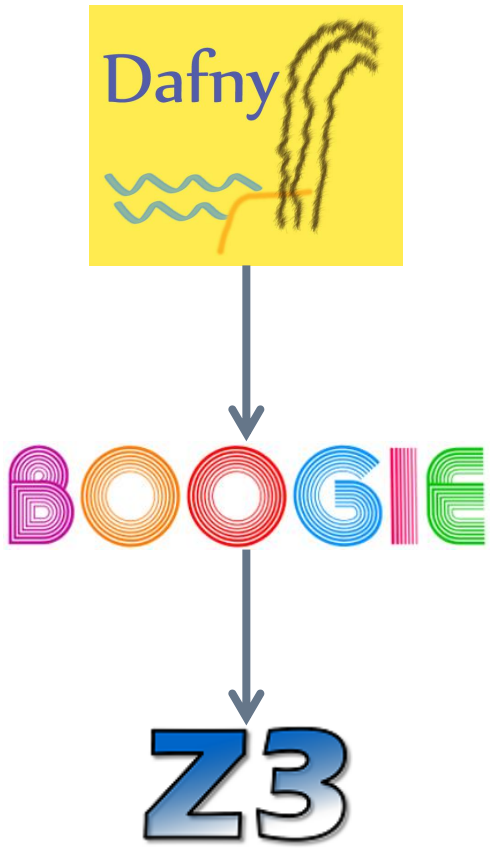
Intermediate verification language

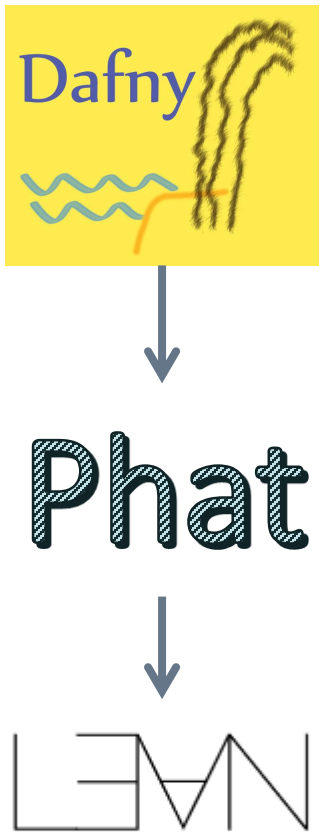




$\hat{P}$

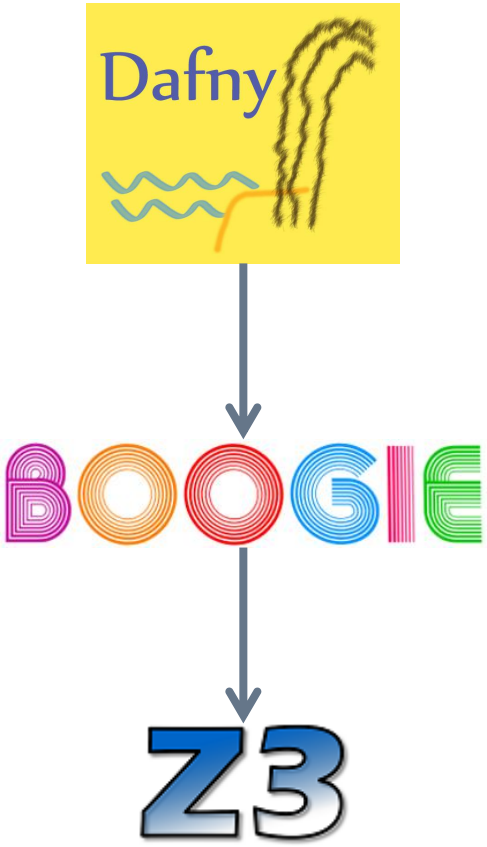
Intermediate verification language

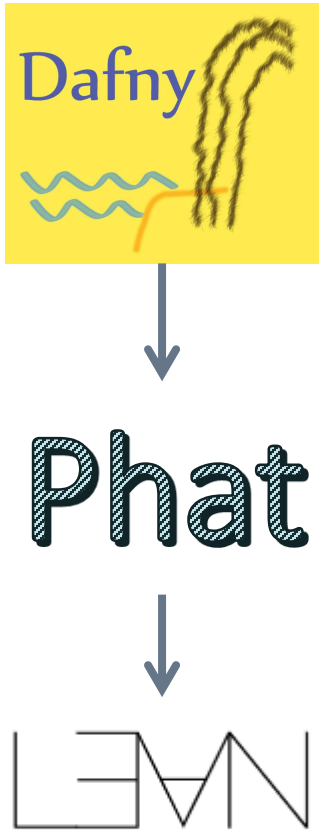




Picture credit: red figures from tihidi.wordpress.com

Intermediate verification language

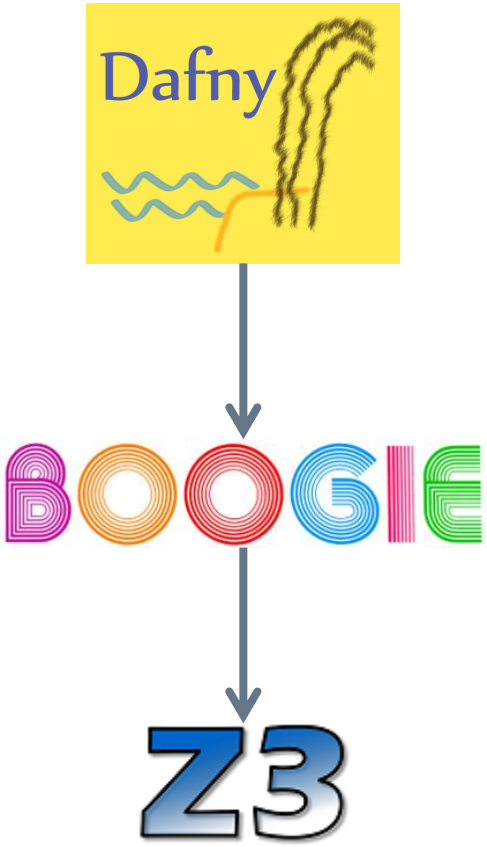


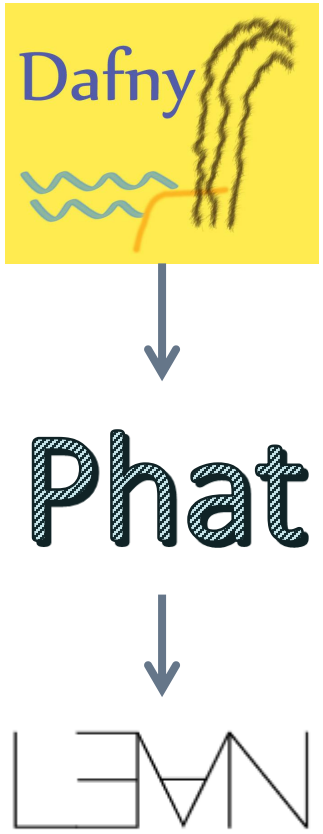


phat

1. cool
2. Pretty Hot And Tempting

Intermediate verification language

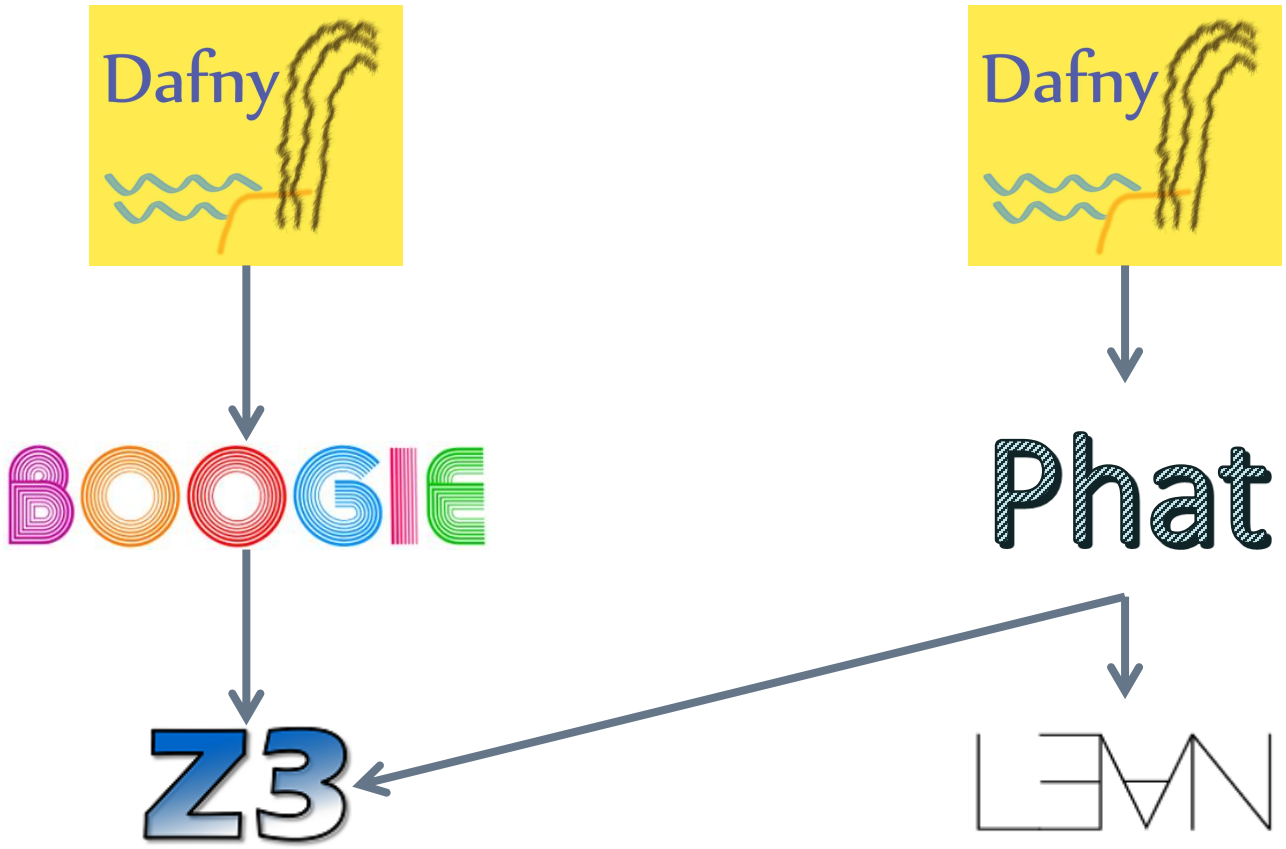




The problem with "phat" is that it is no longer in really. It has kind of phased out and is mostly used by wannabes, lowerclassmen in high school, or middle schoolers. It is now considered a slang faux pas. I wouldn't use it if I was you.

14 year old: "That's phat man."
22 year old: "Um, dude, that word got old in the late '90s"

Intermediate verification language



2 idioms

Idiom 0:

Check what I say, not what I assume

```
assert Y
```

```
assert Y
```

Check what I say, not what I assume

```
assert Y
```

```
assert X
assume Y
```

Check what I say, not what I assume

```
assert Y
```

```
assert X0
assert X1
assume Y
```

```
assert ∀ n •  P(n)
```

```
assert ∀ n •  (∀ k •  k < n ⇒ P(k))  ⇒  P(n)
assume ∀ n •  P(n)
```

induction hypothesis

Idiom 1:

Check it, then forget about it

```
assert Y
```

```
assert X
assume Y
```

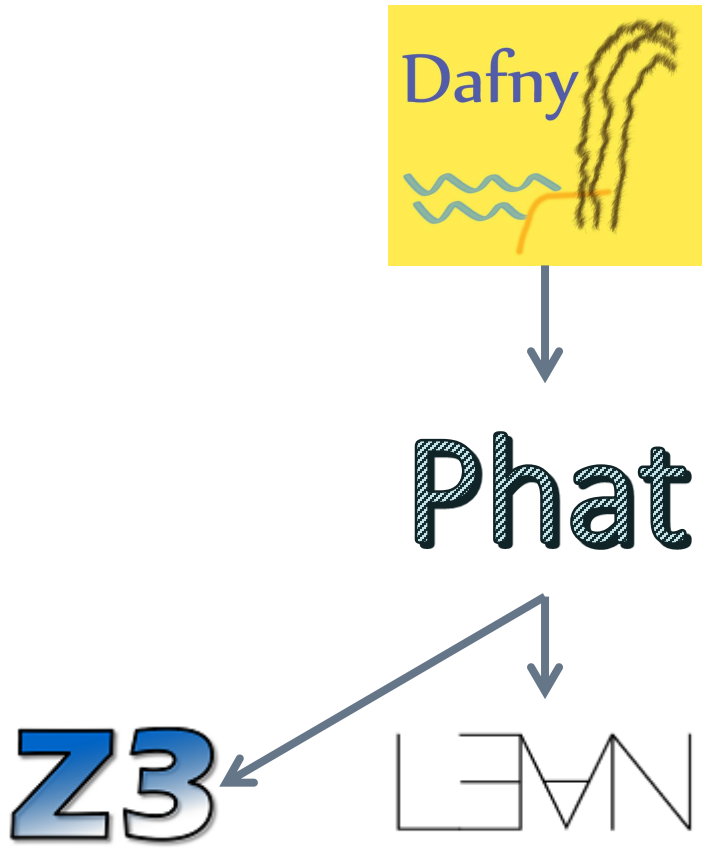
```
if * then
  assert X
  assume false
else
  assume Y
end
```

```
assert Y
```

```
assert X0
assert X1
assume Y
```

```
if * then
  assert X0
  assert X1
  assume false
else
  assume Y
end
```

Certified IVL: Goal



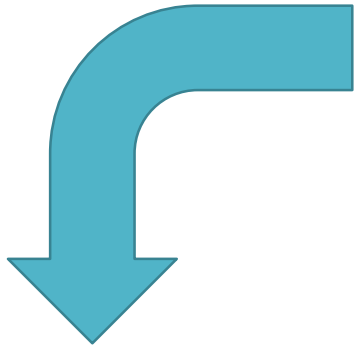
Quick-turnaround verification of source program

Soundness of encoding

Fast path:

Erase the meta correctness constructs

This verification is on the user's clock

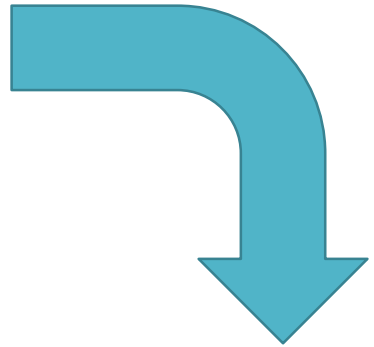


```
if * then
  a := assert X0
  b := assert X1
  assume false
else
  assume Y since a ∧ b ⇒ Y
end
```

Slow path:

Verification includes soundness of assumptions

This verification can be done overnight



Conclusions

Program verification is accessible to interested non-experts

Teaching reasoning does not require understanding complex logics or tactics

High automation of verification is not just for programs

`github.com/microsoft/dafny`

Try Dafny in your browser: `rise4fun.com/dafny`