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

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

Source: “Proof of a program: FIND” by C.A.R. Hoare, CACM, 1971
Program verification: 1998

Program verification: 1998

Program verification: 2008

- Functional correctness
- Limited checking

- Dafny and others
- Extended static checking

- Automatic decision procedures (SMT solvers)
- Interactive proof assistants

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

\[ \text{Cmd ::= } \text{Inc} \mid \text{Cmd} \cdot \text{Cmd} \mid \text{Repeat}(\text{Cmd}) \]

Semantics given by the “big step” relation

\[(\text{Cmd, State}) \rightarrow \text{State} \]

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

\[ \text{Cmd ::= Inc | Cmd} \oplus \text{Cmd} | \text{Repeat(Cmd)} \]

\[
\begin{align*}
t &= s + 1 \\
(\text{Inc, } s) &\rightarrow t
\end{align*}
\]

\[
\begin{align*}
(c_0, s) &\rightarrow s' \\
(c_1, s') &\rightarrow t
\end{align*}
\]

\[
\begin{align*}
(c_0 \oplus c_1, s) &\rightarrow t
\end{align*}
\]

\[
\begin{align*}
t &= s \\
(\text{Repeat(body), } s) &\rightarrow t
\end{align*}
\]

\[
\begin{align*}
(b_0, s) &\rightarrow s' \\
(\text{Repeat(body), } s') &\rightarrow t
\end{align*}
\]

\[
\begin{align*}
(\text{Repeat(body), } s) &\rightarrow t
\end{align*}
\]
Semantics of INC

\[ \text{Cmd ::= } \text{Inc} \mid \text{Cmd} \downarrow \text{Cmd} \mid \text{Repeat(Cmd)} \]

\[
t = s + 1 \\
(\text{Inc}, s) \rightarrow t
\]

\[
\exists s'. (c0, s) \rightarrow s' \quad (c1, s') \rightarrow t \\
(c0\downarrow c1, s) \rightarrow t
\]

\[
t = s \\
(\text{Repeat(body)}, s) \rightarrow t
\]

\[
\exists s'. (\text{body}, s) \rightarrow s' \quad (\text{Repeat(body)}, s') \rightarrow t \\
(\text{Repeat(body)}, s) \rightarrow t
\]
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
Intermediate verification language
Intermediate verification language

Dafny

boogie

Z3

Dafny

Phat

urban dictionary

phat
1. cool
2. Pretty Hot And Tempting
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

Dafny

Boogie

Z3

Dafny

Phat

LEAN
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 $\forall n \cdot P(n)$

**assert** $\forall n \cdot (\forall k \cdot k < n \Rightarrow P(k)) \Rightarrow P(n)$

**assume** $\forall n \cdot P(n)$

**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 \( X_0 \)

assert \( X_1 \)

assume \( Y \)

if * then
  assert \( X_0 \)
  assert \( X_1 \)
  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 \land b \Rightarrow 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