Program Verification & Logic

G. Ramalingam
Microsoft Research

Security experts hack into moving car and seize control

Wednesday, 22 Jul 2015 | 7:27 AM ET

REUTERS

In a controlled test, they turned on the Jeep Cherokee's radio and activated other inessential features before rewriting code embedded in the entertainment system hardware to issue commands through the internal network to steering, brakes and the engine.

Program Correctness

- Software is everywhere …
  - Pacemakers, cars, airplanes, satellites, …
- Software bugs => significant consequences
- Program verification
  - Important … though not a panacea!
- What's a bug?
  - It's not a bug, it's a feature!

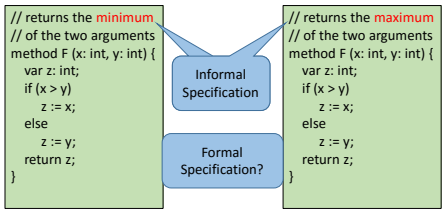
Program Correctness

- Is the following function correct?

```
method F (x: int, y: int) {
   var z: int;
   if (x > y)
      z := x;
   else
      z := y;
   return z;
}
```

Program Correctness

- Is the following function correct?



Specification Constructs:

Assertions

```
method F (x: int, y: int) {
  var z: int;
  if (x > y)
    z := x;
  else
    z := y;
  assert z >= x;
  return z;
}
```

- As documentation
- For dynamic checking
- For static verification

Specification Constructs:

Assertions

```
method F (x: int, y: int) {
  var z: int;
  if (x > y)
    z := x;
  else
    z := y;
  assert z >= x;
  assert z >= y;
  assert (z == x) || (z == y);
  return z;
}
```

- Incomplete vs. complete specifications
- Most specifications are incomplete …
  - limits the value of program-verification

Demo: Dafny

Specification Constructs:
Post-condition

```
method F (x: int, y: int)
   returns (z : int)
   ensures z >= x;
   ensures z >= y;
   ensures (z == x) || (z == y);
{
   if (x > y)
      z := x;
   else
      z := y;
}
```

Is this program correct?

```
method Sum (N: int)
   returns (sum : int)
   ensures sum == N*(N+1)/2;
{
   var i := 0; sum := 0;
   while (i < N) {
      i := i + 1;
      sum := sum + i;
   }
}
```

Specification Constructs:
Pre-condition

```
method Sum (N: int)
   returns (sum : int)
   requires N > 0;
   ensures sum == N*(N+1)/2;
{
   var i := 0; sum := 0;
   while (i < N) {
      i := i + 1;
      sum := sum + i;
   }
}
```

Specification Constructs:
Assume statement

```
method Sum (N: int)
   returns (sum : int)
{
   assume N > 0;
   var i := 0; sum := 0;
   while (i < N) {
      i := i + 1;
      sum := sum + i;
   }
   assert sum == N*(N+1)/2;
}
```

Demo

Mathematical Proofs

- Prove:

  P(n): If $n > 0$, then $\Sigma_{i=1}^{n} i = \frac{n(n+1)}{2}$

- Proof by induction
  - Inductive hypothesis: P(k)
  - Prove: P(1).
  - Assume P(k) and prove P(k+1).

Specification Constructs:

Loop Invariant

```
method Sum (N: int)
   returns (sum : int)
   requires N > 0;
   ensures sum == N*(N+1)/2;
{
   var i := 0;
   while (i < N)
      invariant sum == i*(i+1)/2
  {
      i := i + 1;
      sum := sum + i;
   }
}
```

- A loop invariant serves as an inductive hypothesis (for a proof-by-induction)

Demo

Recursion

```
method Sum (N: int)
   returns (sum : int)
   requires N > 0;
   ensures sum == N*(N+1)/2;
{
   if (N <= 1)
      sum := 1;
   else
      sum := Sum(N-1) + N;
}
```

- The pre-condition/post-condition of a recursive procedure serves as an inductive hypothesis (for a proof-by-induction)

The Problem:
How to (dis)prove it?

```
method Eg1 (x, y, z: bool)
{
   var result : bool;
   if (x)
      result := y;
   else
      result := z;
   assert result;
}
```

(x, y, z) is a counterexample iff $(\neg x \vee \neg y) \wedge (x \vee \neg z)$

- Counterexamples to assertion can be found using a Boolean satisfiability (SAT) solver
- The original NP-complete problem

The Problem:
Arithmetic satisfiability

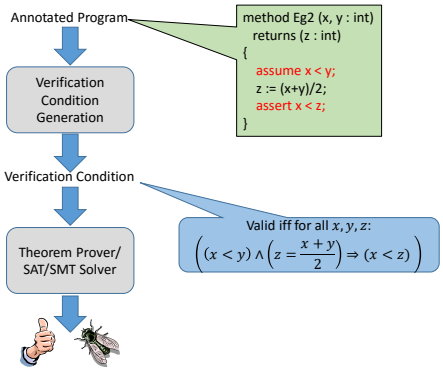
```
method Eg2 (x, y : int)
   returns (z : int)
{
   assume x < y;
   z := (x+y)/2;
   assert x < z;
}
```

Valid iff for all x, y, z: $\left((x < y) \wedge \left(z = \frac{x+y}{2}\right) \Rightarrow (x < z)\right)$

(x, y, z) is a counterexample iff $(x < y) \wedge (z = \frac{x+y}{2})) \wedge (x \geq z)$

- Counterexamples to assertion can be found using an arithmetic satisfiability solver
- Related to early 20th Century work in logic, mathematics, and foundations of computing

Towards Automated Verification



Mathematical Logic: An Introduction

- Barber's paradox & Russell's paradox
- Correctness & proofs
- Informal proofs vs. formal proofs
- Why "formal" proofs?
  - Systematic approach
  - … easier to check (for correctness)
  - … helps avoid mistakes/paradoxes
  - … can automate checking proofs
  - … helps find proofs easier
  - … can automate proof generation

Key Ingredients

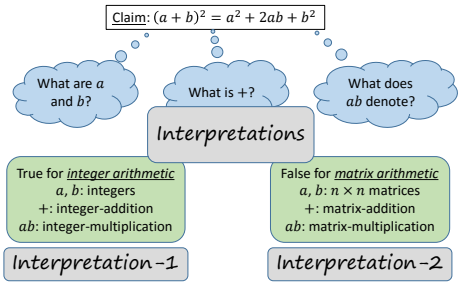
- Axiomatic reasoning

Theorem: $(a+b)^2 = a^2 + 2ab + b^2$
Proof:
$(a+b)(a+b)$
$= a(a+b) + b(a+b)$
$= aa + ab + ba + bb$
$= a^2 + 2ab + b^2$

Axioms:
- Distributivity: $x(y+z) = xy + xz$
- Distributivity: $(x+y)z = xz + yz$
- Commutativity: $xy = yx$
- Congruence: $(x = y) \Rightarrow (x+z) = (y+z)$
- …

Key Ingredients

- Separation of syntax and semantics

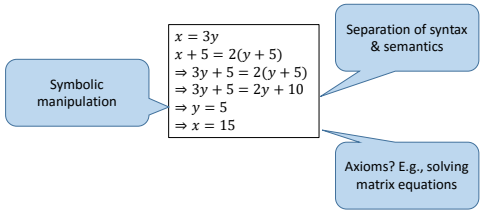


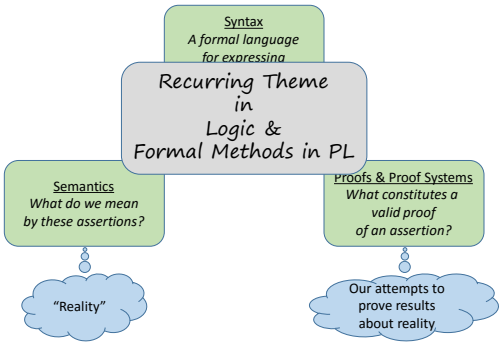
Key Ingredients

- Syntactic approach to proofs
  - symbolic manipulation

- Similar to algebraic approaches to solving word problems

- If Alice is thrice as old as Bob and in another five years Alice will be twice as old as Bob, how old are Alice and Bob?





Propositional Logic

- A language for (pure) Boolean-expressions
- Boolean variables: $p, q, r, \ldots$
- Boolean operators:
  - And: $p \wedge q$
  - Or: $p \vee q$
  - Not: $\neg p$
  - Implies: $p \Rightarrow q$
  - …
- Evaluation of Boolean expressions

Propositional Logic: Syntax

- P ::= a set of propositional variables
- The set of formulas over P is defined by
  $\phi ::= P \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \ldots$
- Define $\phi_1 \Rightarrow \phi_2$ to be shorthand for $(\neg\phi_1) \vee \phi_2)$
- Alternatively: take $\Rightarrow$ and $\neg$ as primitive operations
  - Exercise: Define $\wedge$ and $\vee$ in terms of $\Rightarrow$ and $\neg$

Propositional Logic: Semantics

- Let T and F denote the values true/false
- Given $M : P \to \{T, F\}$
- We can recursively define (evaluate) the value $M(\phi)$ of any formula ϕ

$M(\phi_1)$	$M(\phi_2)$	$M(\neg\phi_1)$	$M(\phi_1 \vee \phi_2)$	$M(\phi_1 \wedge \phi_2)$
T	T	F	T	T
T	F	F	T	F
F	T	T	T	F
F	F	T	F	F

Propositional Logic: Semantics

- We say $M : P \to \{T, F\}$ is an interpretation (or truth-assignment)
- We write $M \vDash \phi$ iff $M(\phi) = T$.
  - M is said to be a model for ϕ iff $M \vDash \phi$
- ϕ is said to be satisfiable if it has a model
- ϕ is said to be unsatisfiable if it has no model
- ϕ is said to be valid (or a tautology) if every interpretation M is a model for ϕ
- We write $\vDash \phi$ iff ϕ is a tautology

Examples

- Which of the following are satisfiable? Which are tautologies?

- $p \Rightarrow (p \vee q)$
- $p \Rightarrow (p \wedge q)$
- $p \wedge (\neg p)$

- Theorem: ϕ is a tautology iff $\neg\phi$ is unsatisfiable