Introduction to Software Verification

Orna Grumberg

Lectures Material
winter 2017-18

Lecture 1

Why (formal) Verification?

- Safety-critical applications:
  - Air-traffic controllers
  - Medical equipment
  - Cars

Bugs are unacceptable!

- Bugs found in later stages of design are expensive, e.g. Intel's Pentium bug in floating-point division

- Testing does not provide full coverage

The goal of the course:
Formal Verification

Given

- A (model of) hardware or software system and
- a formal specification

does the system satisfy the specification?

Not decidable!

Formal Verification

Solutions:

- “Program correctness”:
  Provide non-automated verification methods

- “Automatic verification / Model Checking”:
  restrict the problem to a decidable one:
  - **Finite-state** reactive systems
  - **Propositional** temporal logics

Specifications

- Should be given for a system by the designer, developer, programmer, user

- Examples:
  - Does the program always terminate?
  - Does the program compute correctly multiplication of its inputs?

Specifications

- Additional examples:
  - When we press a sequence of buttons on the control panel of an airplane / microwave - do we get the desired result?
  - When we deposit money – does it get to our account?
  - Can a user access data only if he has the appropriate authorization?

Verification tools

Are developed and used in

- Hardware industry: Intel, IBM, Cadence, Mellanox, …
- Software industry: Microsoft, NASA, Amazon, Facebook…
- Universities

Part 1 of the course

Program Correctness

- Non-automated
- Verifies program with possibly infinite number of states
- Refers to the programs as input-output transformation

Ingredients for Formal Verification

1. Specification language
   - With formal semantics

2. Programming language
   - with formal semantics

3. Proof rules
   - For proving "Program P has the property φ"

Requirements from the proof rules

- Soundness of the rules: if we were able to prove correctness of program P w.r.t. specification φ using the proof rules, then P is correct w.r.t. φ

- Completeness of the rules: if P is correct w.r.t. specification φ, then our proof rules <u>can</u> prove it

We handle:

- Deterministic programs
  - Exactly one computation for every input
  - At most one output for each input
- Properties
  - Partial correctness
  - Termination
  - Total Correctness

Some notations

- Program variables: $\bar{x} = (x_1,\ldots,x_n)$
- A state of the program σ is a function from program variables to their domains
- The set of program states is defined by:

  $D_1 \times \ldots \times D_n \cup \{\perp\}$

  Where D_i is the domain of variable x_i

Program states: Examples

- A program with integer variable x, Boolean variable b
  - States: (5, F), (-17, T)

- Elevator on 3 floors:
  elev_at ∈ {1, 2, 3}
  on_floor1, on_floor2, on_floor3: Boolean
  in_elev1, in_elev2, in_elev3: Boolean
  direction ∈ {up, down}, door ∈ {open, close}
  - State: (2, F,T,T, T,T,F, up, close)

Defining the Specification

Specification is a pair $<\mathbf{q_1(\bar{x}), q_2(\bar{x})}>$ where:

- $q_1(\bar{x}), q_2(\bar{x})$ are first order formulas over program variables
- $\mathbf{q_1(\bar{x})}$ describes a condition holding before the execution of the program
- $\mathbf{q_2(\bar{x})}$ describes a condition holding at the end of the execution of the program

Examples

Specification example

- $\langle (x \geq 0 \wedge y > 0), (z = x/y \wedge z \geq 0) \rangle$

A program with $x \in \mathrm{N}$, $y \in R$, $b \in \{T, F\}$

States: (5, 5.0, T), (7, 3.111, F)

$q_1(x, y, b) = x > 0 \wedge b$

$q_2(x, y, b) = x + y > 0 \wedge \neg b$

Computations of Programs

- $\pi(P,\sigma)$ denotes a computation of program P from state σ
- $\pi(P,\sigma)$ is a finite $(\sigma_1, \ldots, \sigma_k)$ or infinite $(\sigma_1,\sigma_2,\ldots)$ sequence of states where:
  - $\sigma_1 = \sigma$
  - σ_{i+1} is a result of applying an action from the program on σ_i
- This definition is ***not*** **a full definition**

More notations

- $\perp$ - bottom : the undefined value

- **val(π)** denotes the final state of computation π (if exists)
  - val(π) = σ_k if $\pi = (\sigma_1, \ldots, \sigma_k)$
  - val(π) = $\perp$ if $\pi = (\sigma_1, \sigma_2, \ldots)$
    - π is an infinite computation

- $\sigma \models \mathbf{q}(\bar{\mathbf{x}})$ if $q(\bar{x})$ is true when free variables in q are replaced with matching values in σ

- Important remark:
  $\perp \not\models q(\bar{x})$ for every $q(\bar{x})$ (even $\perp \not\models$ true)

- Example of formulas and their meaning:
  $q(y) = \forall x(y|x \vee 2\nmid x)$ where x,y are naturals
  - For a state σ(x)=1, σ(y)=2, σ(z)=1
    $\sigma \models q(y)$ since $\forall x(2|x \vee 2\nmid x)$ is true

Partial Correctness

- A program **P** is partially correct with respect to specification $<\mathbf{q_1(\bar{x}), q_2(\bar{x})}>$ iff for every computation π of P from an initial point of P, and for every state σ_0 :

  if

  - the computation starts from state σ_0 which satisfies $q_1(\bar{x})$ and
  - the computation terminates

  then

  - $q_2(\bar{x})$ holds at the end of the computation

Partial Correctness

- For every computation π and every state σ_0 :

$$(\sigma_0 \models q_1(\bar{x}) \text{ and } val(\pi(P, \sigma_0)) \neq \perp) \Rightarrow$$
$$val(\pi(P, \sigma_0)) \models q_2(\bar{x})$$

- Notation: $\{q_1\}P\{q_2\}$

Total Correctness

- A program **P** is totally correct with respect to specification $<\mathbf{q_1(\bar{x}), q_2(\bar{x})}>$ iff for every computation π of P from an initial point of P, and for every state σ_0 :
  if
  - the computation starts from state σ_0 which satisfies $q_1(\bar{x})$

  then
  - the computation terminates, and
  - $q_2(\bar{x})$ holds at the end of the computation

Total Correctness

- For every computation π and every state σ_0 :

$$\sigma_0 \models q_1(\bar{x}) \Rightarrow \text{val}(\pi(P, \sigma_0)) \neq \perp \text{ and}$$
$$\text{val}(\pi(P, \sigma_0)) \models q_2(\bar{x})$$

- Notation: **$<q_1>P<q_2>$**

How do we write the specification:

"P terminates if the initial state satisfies $\mathbf{q_1}$ "

Separation Lemma

- For every program P and specification $<q_1,q_2>$:

$$\models <q_1>P<q_2>$$

if and only if

$$\models \{q_1\}P\{q_2\} \text{ and } \models <q_1>P<true>$$

Examples

- Which programs satisfy {true}P{false} ?

- Which programs satisfy <true>P<false> ?

Logical Variables in Specifications

Example 1:

Specify a program with a single variable x whose value at the end of the computation is twice its value at the beginning

Logical Variables in Specifications

Solution: add fresh variables which are

- not part of the program and therefore
- their value does not change during the execution of the program

These variables are called logical variables

Convention: We use logical variable X to preserve the value of variable x

Logical Variables in Specifications

Example 2:

Program which returns in variable z the multiplication of variables x and y

Convention:

Assertions q_1 ,q_2 are now defined over $\bar{\mathbf{x}}$ that includes program variables as well as logical variables