CS 245

Program Verification

Huth and Ryan chap 4

Topics: Introduction

Pre and Post Conditions

Conditional statements

while loops

partial correctness

total correctness

arrays

Given a program, P, and a specification of the expected behavior of P we would like to check that all the behaviors of P satisfy the given specifications.

Reasons for checking the behavior of hardware/software:

- Documentation - some software/hardware artifacts will be used for several years. Having a well written document explaining what the artifact is intended to do ⟶

and under what circumstances and environments it is intended to be used is important, a matter of basic business practice.

Time to market: analyzing and debugging software / hardware artifacts can be enormously more costly the later in the design cycle the analysis occurs.

Buggy software:

See a list of software bugs on wikipedia, etc.

There are several different ways to address the problems of finding bugs and analyzing software/hardware artifacts.

Analysis methods include:

- inspection and code walk-through
- testing (white box and black box)
- formal verification

" Testing can be a very effective way to show the presence of bugs but is hopelessly inadequate for showing their absence, never their absence. "

E. Dijkstra

Testing: check a programs behavior on a (small number) of (carefully) chosen examples.

In general, the method cannot be exhaustive

Formal verification:

give a well defined syntax and semantics to the program and its specification.

In a mathematically rigorous manner prove that the programs behaviors meet the program's specification.

Formal verification techniques can offer several advantages over other program analysis methods.

- Reduce the number of bugs
- Can be applied early in the design stage
- Used to prove the correctness of safety critical, system critical and business critical components

Current Industrial Practice

- Writing software/hardware specifications and requirements is widespread/standardized

- verification of hardware, embedded systems, software drivers, routing protocols is wide spread/common (e.g. Intel, IBM, etc.)

- Formal verification of general software components is more challenging.

A framework for formal verification

- Gather input from stakeholders to create an informal set of software requirements/specifications

- Convert the informal requirements/specifications into 'equivalent' specifications in a formal logic.

Write a program P that realizes the specifications.

Then prove that the program P satisfies the formal logic specifications.

In this course we focus on this last step, how to show that a program satisfies a specification.

Programs will be written in an imperitive programming language similar in style to C/C++ and Java.

The core features of the language are

- integer and boolean valued expressions and variables
- assignment statements
- sequences of program statements

- if-then-else conditional branching statements
- while-loops
- for-loops
- array variables

Imperative programs:

- program steps read from program variables and may assign integer (or boolean) values to integer (or boolean) variables

- A program state is given by a tuple of values to each of the program variables including a value to the program counter (which points to the next line of program code to be executed).

- Program expressions involving one or more program variables are evaluated relative to the current values of the variables

- Executing a program statement changes the state of the program, updating the program counter and variables.

An example program Fac 1

```
y = 1;
z = 0;
while ( z != x ) {
    z = z + 1;
    y = y * z;
}
```

Consider what happens
if the program points at
the while statement

→ while (z != x) {

and the state of the program
is given by...

The program is intended to compute the factorial of x and store the value in y.

What does a specification of correct behavior for Fact look like?

Example:

Compute a number y whose square is less than the input x.

A specification such as $y^2 < x$ may look reasonable.

However, what happens if $x = -4$?

We can revise the requirement:

If the input x is a positive number, compute a number whose square is less than x.

The specification for the program P speaks about the state of the program before P begins execution and about the state of the program after the program finishes executing.

Program specifications will then appear as triples

$$\{\phi\}\ P\ \{\psi\}$$

Informally, this means, if the program, P, starts execution in a state that satisfies ϕ, then if P terminates, it terminates in a state that satisfies ψ.

An assertion of the form

$$\{\phi\}\ P\ \{\psi\}$$

is known as a Hoare triple after Sir Tony Hoare.

The logic we are studying is also know as Floyd-Hoare logic after Robert Floyd and Tony Hoare.

For the given example:

Let P be the program that computes a number whose square is less than x.

We may write:

$$(\lambda x > 0)\ P\ (\lambda y \cdot y < x)$$

A triple of the form

$\{ x > 0 \}\ P\ \{ y \cdot y < x \}$

may apply to several programs.

E.g.

P_1:

```
y = 0;
```

P_2:

```
y = 0;
while (y*y < x){
  y = y+1;
}
y = y-1;
```

Hoare logic develops a form of proof that allows us to show that the program P satisfies the pre condition ϕ and post condition ψ by reasoning about the basic steps of the program P.

To do this a program calculus is developed that combines formulae of first order logic ϕ and ψ with programs, P, to form triples

$$\{\phi\}\ P\ \{\psi\}.$$

The proof rules for the calculus are based on the individual program constructs such as assigment statements, sequential composition, if-then-else, etc.

A triple $\{\phi\}\ P\ \{\psi\}$ may be stated under partial or total correctness.

A (program) state provides a valuation to all program variables (including the program counter).

For partial correctness

$\vDash_{par} \{\phi\}\ P\ \{\psi\}$

holds iff

for all program states, s,
if s satisfies ϕ, then if
P, starting in s, terminates
in a state t, then t
satisfies ψ.

Example:

Write appropriate ϕ and ψ such that

$[\![\phi]\!]$ while true { x = 0; } $[\![\psi]\!]$

For total correctness

$$\models_{tot} \{\phi\}\ P\ \{\psi\}$$

iff

for all program states, s,

if s satisfies ϕ and if

P begins execution in s,

then P terminates in a state

t and t satisfies ψ.