University of Strathclyde Science

CS208 (Semester 1) Topic 5 : Specification and Verification

Dr. Robert Atkey

Computer & Information Sciences

University of Strathclyde Science

Specification and Verification, Part 1

Validation and Verification

https://twitter.com/jessiedotjs/status/667118579075141632

From the birth of programming...

By June 1949 people had begun to realize that it was not so easy to get programs right as at one time appeared. I well remember when this realization first came on me with full force.

...

From the birth of programming...

...

The EDSAC was on the top floor of the building and the tape-punching and editing equipment one floor below. [...] It was on one of my journeys between the EDSAC room and the punching equipment that "hesitating at the angles of stairs" the realization came over me with full force that a **good part of the remainder of my life was going to be spent in finding errors in my own programs.**

Maurice Wilkes
Memoirs of a Computer Pioneer, MIT Press, 1985, p. 145.

(The EDSAC was an early "stored program" computers, and first ran in May 1949)

Validation and Verification

The two big questions for any software system:

1. **Validation**: Are we building the right thing?
2. **Verification**: Are we building the thing right?

Validation: What should it do?

A question answered by interaction with stakeholders:

- Users
- Purchasers (not necessarily the users!)
- Data subjects (may not be the above!)
- Regulatory bodies (e.g., ICO)
- Maintaince and deployment engineers
- ...

Validation: What should it do?

A question answered by interaction with stakeholders:

- Users
- Purchasers (not necessarily the users!)
- Data subjects (may not be the above!)
- Regulatory bodies (e.g., ICO)
- Maintaince and deployment engineers
- ...

Validation is hard, and beyond the scope of this course.

Verification: Are we doing it right?

Answered by examining the system and the way it is built:

- Good coding practices
- Code review
- Testing
- Constructing arguments, formally or informally

Specification

Validation & Verification meet at **Specification**.

A **Specification** is the description of what the system ought and ought not to do.

Specification

Validation & Verification meet at **Specification**.

A **Specification** is the description of what the system ought and ought not to do.

Validation : what is the specification?
Verification : do we meet the specification?

Specifying a Game

Technical specifications:

Must work on platform X, Y, ...

Specifying a Game

Technical specifications:

Must work on platform X, Y, ...

- Must work on platform X last updated in 2010, ...

Specifying a Game

Technical specifications:

Must work on platform X, Y, ...

- Must work on platform X last updated in 2010, ...
- Must work with version 1.0.23.92 of library Z, because we won't pay for the newer one

Specifying a Game

Technical specifications:

Must work on platform X, Y, ...

- Must work on platform X last updated in 2010, ...
- Must work with version 1.0.23.92 of library Z, because we won't pay for the newer one

Must have framerate of over 60fps.

Specifying a Game

Feature specifications:

1. Must allow online multiplayer
2. Must not leak personal user data when online
3. Must have screenshot feature
4. Must have player chat
5. Must have blocking feature

Specifying a Game

"Obvious things"

1. Must not crash
2. Must be installable
3. When controls say "go foward", player's character goes forward
   - 3.1 Unless there is an obstacle
   - 3.2 Or the player is frozen
   - 3.3 ...
4. ...

Specifying a Game

1. Makes player feel happy
2. Makes player feel slightly frustrated, but the good kind of frustrated, not the bad one...
3. Player must not get stuck behind fences
4. Game must be solvable
5. Game must get U rating
6. ...

Real Specifications

Specifying complete systems is **hard**.

Real-world requirements are:

- Fuzzy (the game must be “fun”)
- Involve other systems (e.g., the OS, the Internet)
- Involve people / physical reality
- Generally very complex

Formal Specifications

Nevertheless, written down specification is important:

- To communicate clearly with other engineers and with stakeholders
- To expose conflicting requirements and ambiguity
- To enable formal argumentation

University of Strathclyde Science

Specification and Verification, Part 2

Formal Specification and Verification

Formal Methods

The dream:

1. Stakeholders get together, come to agreement, and write down a clear, complete, and unambiguous specification P;
2. Implementors are given P and produce an implementation C;
3. Implementation C is verified against P, formally, using proof.
4. Software is perfect. Everyone is happy.

Formal Methods

Clearly, that is only ever going to be a dream.

Formal Methods

Clearly, that is only ever going to be a dream.

But, we can formally specify and sometimes verify:

- High-value, critical, aspects of a system
- Low-level, unambiguous aspects of components
- Simple but useful properties that apply “everywhere”

Formal Specifications

Example:

> If a transaction is acknowledged by the server, then it must have already been logged in the journal and written to durable storage.

Specifying an aspect of behaviour that is critical to the system's operation.

Formal Specifications

Example:

> This function must sort the input array in ascending order.

Specifying a (relatively) low-level aspect of part of a system that other parts rely on.

Formal Specifications

Example:

If a function's type says it returns an `int`, then it either returns an `int`, or raises an exception, or does not return.

An example of a specification written in the type system of a language, and enforced by the language implementation.

Logical Specifications

Formal logic can be used to write down a specification of a system, assuming **we have a formal model of how the system operates**.

Logical Specifications

Formal logic can be used to write down a specification of a system, assuming **we have a formal model of how the system operates**.

So the problem becomes:

1. Fix a formal model of how the system operates
2. Prove the formal model meets the specification

University of Strathclyde Science

Formal Models of Programs

1. As functions, with behaviour specified by equations.
   This is what you did in `ask` in CS106.

2. As logical proofs themselves, following the "evidence" interpretation. See CS410.

3. Modelled as predicates in the logic.
   We will do this here.

4. Many others...

A Simple Formal Model

A simple formal model of program execution can be defined in terms of a single predicate:

$$\mathrm{exec}(\mathrm{prog}, \mathrm{initState}, \mathrm{finalState})$$

meaning:

- the program prog,
- when started in initial state $\mathrm{initState}$,
- can terminate with final state $\mathrm{finalState}$.

A Simple Formal Model

We will flesh out what exec means by using axioms.

Even without knowing what the exact axioms are, we can make some general definitions.

A Simple Formal Model

A program prog...

... terminates starting in state s if:

$$\exists \mathrm{fs}.\ \mathrm{exec}(\mathrm{prog}, \mathrm{s}, \mathrm{fs})$$

... terminates **on all inputs** if:

$$\forall \mathrm{s}. \exists \mathrm{fs}.\ \mathrm{exec}(\mathrm{prog}, \mathrm{s}, \mathrm{fs})$$

A Simple Formal Model

A program prog ...

... is deterministic if:

$$\forall s.\forall s_1.\forall s_2.\mathrm{exec}(\mathrm{prog}, s, s_1) \rightarrow \mathrm{exec}(\mathrm{prog}, s, s_2) \rightarrow s_1 = s_2$$

A Simple Formal Model

This model is *very* simple.

It does not include:

1. Resources, such as amount of time required
2. Interactive computation, input is provided all at once
3. Online computation, output is provided all at once
4. Errors or exceptions distinct from states

A Simple Formal Model

However, it is useful:

1. *Partial* and *non-deterministic* computation
2. Programs and data can be mixed
3. Enough to talk about specifications

University of Strathclyde Science

Specification and Verification, Part 3

Hoare Logic

Specifying Programs' Behaviour

The classic specification of a program's behaviour:

1. If the initial state s_1 satisfies some predicate $P(s_1)$; and
2. the program starts in state s_1 and finishes in state s_2, then
3. the final state s_2 satisfies a predicate $Q(s_2)$.

P is the *precondition*; Q is the *postcondition*.

Hoare Triples

This is often written like this:

$$\{P\}\,\mathrm{prog}\,\{Q\}$$

which is called a *Hoare triple*.

The formal definition is:

$$\forall s_1.\forall s_2.P(s_1) \rightarrow \mathrm{exec}(\mathrm{prog}, s_1, s_2) \rightarrow Q(s_2)$$

If P is true before executing prog, then Q is true afterwards.

Hoare Triples

Hoare triples

$$\{P\}\,\mathrm{prog}\,\{Q\}$$

specify *partial* correctness.

Only says *if the program finishes*, then the postcondition Q holds.

Total correctness will be defined later...

Hoare Logic

Hoare Logic is a deductive system for judgements of the form:

$$\{P\}\,\mathrm{prog}\,\{Q\}$$

The “Big idea” is that (mostly) the structure of the proof follows the structure of the program.

Hoare Logic: Rules

Let $\mathrm{skip}()$ be the program that does nothing, successfully.

Its rule:

$$\frac{}{\{P\}\,\mathrm{skip}()\,\{P\}}\ \textsc{Skip}$$

$\mathrm{skip}()$ doesn't alter the state: whatever was true before is true after.

Hoare Logic: Rules

Let $\mathrm{seq}(p_1, p_2)$ be the program that first does p_1 then does p_2.

Its rule:

$$\frac{\{P\}\, p_1 \,\{R\} \qquad \{R\}\, p_2 \,\{Q\}}{\{P\}\, \mathrm{seq}(p_1, p_2) \,\{Q\}}\ \textsc{Seq}$$

If p_1 gets us from P to R, and p_2 gets us from R to Q, then doing them in sequence gets us from P to Q.

Hoare Logic: Rules

Let doUpdate(s) be some function on states, and update be the *program* that performs that update.

There are two rules, depending on whether we are reasoning "forwards" or "backwards".

Hoare Logic: Rules

Rule, variant 1 (*weakest precondition*):

$$\frac{}{\{P[s := \mathsf{doUpdate}(s)]\}\,\mathrm{update}()\,\{P\}}\ \textsc{Update-Bwd}$$

Rule, variant 2 (*strongest postcondition*):

$$\frac{}{\{P\}\,\mathrm{update}()\,\{\exists s'.s = \mathsf{doUpdate}(s') \wedge P[s := s']\}}\ \textsc{Update-Fwd}$$

Hoare Logic: Rules

Let $\mathrm{if}C(p_1, p_2)$ be the program that runs p_1 if C is true, and p_2 if C is not true.

$$\frac{\{C \wedge P\}\, p_1\, \{Q\} \qquad \{\neg C \wedge P\}\, p_2\, \{Q\}}{\{P\}\, \mathrm{if}C(p_1, p_2)\, \{Q\}}\ \textsc{If}$$

Hoare Logic: Rules

Let $\mathrm{while}\mathrm{C}(\mathrm{p})$ be the program that runs p until condition C is false.

$$\frac{\{\mathrm{C} \wedge \mathrm{P}\}\,\mathrm{p}\,\{\mathrm{P}\}}{\{\mathrm{P}\}\,\mathrm{while}\mathrm{C}(\mathrm{p})\,\{\neg\mathrm{C} \wedge \mathrm{P}\}}\ \textsc{While}$$

In this case, P is the *loop invariant*.

Hoare Logic: Rules

Consequence:

$$\frac{P' \vdash P \qquad \{P\}\,p\,\{Q\} \qquad Q \vdash Q'}{\{P'\}\,p\,\{Q'\}}\ \textsc{Consequence}$$

1. *Strengthen* the precondition: $P' \vdash P$
2. *Weaken* the postcondition: $Q \vdash Q'$

Hoare Logic

Each of these rules can be proved sound with respect to the partial correctness property:

$$\forall s_1.\forall s_2.P(s_1) \rightarrow \mathrm{exec}(\mathrm{prog}, s_1, s_2) \rightarrow Q(s_2)$$

We need to assume some axioms for each of the different program constructs; this is our **trusted base**. If we get these axioms wrong, then a proof in Hoare logic means nothing.

Hoare Logic: A Quirk

If C is the condition that is always true, then the program

$$\mathrm{while}\mathrm{C}(\mathrm{skip}())$$

satisfies *any* specification:

$$\{\mathrm{P}\}\,\mathrm{while}\mathrm{C}(\mathrm{skip}())\,\{\mathrm{Q}\}$$

Why?

Total Correctness

Total correctness is usually written as:

$$[\mathrm{P}]\ \mathrm{prog}\ [\mathrm{Q}]$$

and is defined as:

$$\forall s_1.\mathrm{P}(s_1) \rightarrow (\exists s_2.\mathrm{exec}(\mathrm{prog}, s_1, s_2) \wedge \mathrm{Q}(s_2))$$

All of the rules are the same as for the partial version, except for the rule for $\mathrm{while}\mathrm{C}(\mathrm{p})$.

Summary

- Formal Specification and Verification require a formal model
- The execution predicate is a simple but useful model
- Hoare logic is a deductive system for proving programs satisfy specifications in this model
- It comes in partial and total variants.