An Axiomatic Basis for Computer Programming: Review

Gosha Krikun

Innopolis University
g.krikun@innopolis.ru

February 16, 2017

Abstract

This is review on C.A.R. Hoare's paper, An Axiomatic Basis for Computer Programming. (WIP)

I. Introduction

"People are not perfect", this is what we learned so fourth. Every one could make a mistake. In some areas it could be unnoticed, but in others it could cause serious consequences.

There are a lot of people with their needs. Majority of them lie outside safety-critical sector, error and failures in which could leads to deaths. People won't spent big money for time of qualified developers to avoid minor bugs.

So nowadays main approach to find failures and errors is testing. And I can't not mentioned my favorite quote about it:

> *" Program testing can be used to show the presence of bugs, but never to show their absence! "*
>
> — Edsger Wybe Dijkstra

But more and more critical tasks entrust to computer systems. When errors and failures cost increases, we will pay for diligence and accuracy as much as needed.

To be exactly sure, that we don't make mistakes we make premises and develop techniques for reasoning about properties and correctness. And for proving it we used our logic and deduction.

II. Purpose and Applications

So Hoare suggested triple $\{P\}Q\{R\}$ in which developer need define precondition (premises) and postcondition (conclusion) which hold for program Q after execution.

This technique helps prove correctness of a program Q, according to defined logical predicates P and R.

It doesn't help derive true predicates from requirements, but it provide ability to use common proof techniques for deducting correctness. In other words, it helps us check that program do things right, but doesn't affirm that what we want, is right things.

For next discussion suppose that we define right P and R from some requirements.

Speaking about correctness we should understand, that any answer could be in three state: answer received and it is correct, answer not received, answer is incorrect.

So when answer received and it is right, for all input values in the domain, then we call it total correctness.

If answer could be not received (e.g. program doesn't terminate) for some values in the domain, but is right for others - partial correctness.

And incorrectness in last case.

Using axioms and inference rules what was proposed by Hoare, we could deduct that if P holds before execution of Q, and if Q terminates, then R will holds after execution (correctness of Q).

In 1969 paper Hoare didn't mentioned that Q should terminates, so he defined rules just for deducting partial correctness (particular in rule D3). But later he add definition for total correctness too.

For example lets take task on finding real roots for quadratic equation:

$$ax^2 + bx + c = 0$$

We restrict input domain as set of real numbers and according to discriminant in predicate P (suppose we used general quadratic formula):

$$P \equiv (a, b, c \in \mathbb{R}) \wedge (b^2 - 4ac \geq 0)$$

Then we could find values by some program Q. Suppose as a result program returns set of values X (Don't think about implementation yet). And finally check whether equation holds or not in R:

$$R \equiv (X \subseteq \mathbb{R}) \wedge (\forall x \in X \bullet ax^2 + bx + c = 0)$$

P like a guard, filter input on what program will work correctly, and R check results to be correct.

III. Axioms and Rules

In the '69 article Hoare states four basic rules:

- Axiom of Assignment
- Rule of Consequence
- Rule of Composition
- Rule of Iteration

But sometimes another two rules is stated - Skip Axiom and Condition Rule. First one could be derived from Axiom of Assignment, and second from Rule of Iteration.

All this rules implicitly assumed the absence of side effects for Q (nothing changes except state which we define in pre and postconditions). Because in other case, we couldn't consider correctness of overall state.

Using this rules we can make deductive proof from our premise to conclusion, that R will holds after execution.

But only after, not during.

It could be a case when program Q during execution change state which doesn't hold conditions P and R.

This case could appear when specification describe something in general, but doesn't divide execution into small pieces of actions.

If we think about it in more details, we could decompose it to two sequence of statements (by rule of composition D2), in which first part hold some postcondition R_1 and second have R_1 as precondition. But R_1 could be arbitrary, for example $\neg P$.

As we can see, there is nothing scary in this case if we still could prove correctness of our program.

Lets return to our example about real roots.

Suppose we choose general quadratic formula for solving equation:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1}$$

We know that $b^2 - 4ac$ not negative, but we don't know weather a is equals to zero. In this case equation became linear, but still have solutions.

Suppose we didn't see that general formula is derived from equation. Lets just apply substitution all occurrence of x in postcondition by stated expression (1).

After polynomial expansion we could see that equation takes the form:

$$\frac{b^2}{2a} \mp \frac{b\sqrt{b^2 - 4ac}}{2a} - c - \frac{b^2}{2a} \pm \frac{b\sqrt{b^2 - 4ac}}{2a} + c = 0$$

I little bit cheated, because I already reduce some parts by a, but nevertheless, after reducing the remaining we get:

$$0 = 0$$

But we should remember, that we could reduce only by not zero value, thus postcondition holds only if $a \neq 0$.

Lets predicate $A \equiv a \neq 0$, then precondition for our statement:

$$P_1 \equiv P \wedge A$$

By the manipulations above we actually proved that our statement correct, with respect to precondition P_1 and postcondition R.

And we could notice that predicate P_1 is stronger that predicate P. By "stronger" we mean that $P_1 \Rightarrow P$, or informally, if predicate P_1 holds for some values, predicate P will holds either. But opposite is not true (in our case $x = 0$ doesn't hold in P_1).

So we could make preconditions stronger, program will still be correct.

With postcondition predicates, on the contrary, we couldn't make it stronger. Because if we don't change program Q, it will produce range of results, which satisfy weaker predicate. This range is superset of set which satisfy stronger predicate.

So Q will produce values which don't satisfy postcondition - program will be incorrect according to stronger predicate.

But we could make postcondition weaker, program will still be correct.

This is actually Rule of Consequence stated in Hoare paper.

But lets return to our example.

In case when $x = 0$ (or $P \wedge \neg A$) equation takes the form:

$$bx + c = 0, \text{ and } x = \frac{-c}{b} \tag{2}$$

Yet again, this remains true when $B \equiv b \neq 0$ is true, so this statement is correct according to precondition $P \wedge \neg A \wedge B$ and postcondition R.

As you can see, we try discharge proof by cases, and then, using Condition Rule (or Rule of Iteration), prove correctness of their combination.

In case when $b = 0$ or $\neg B$, equation takes the form:

$$c = 0$$

And again, two cases. Either $c = 0$ and equation holds for all $x \in \mathbb{R}$:

$$X = \mathbb{R} \tag{3}$$

Or $c \neq 0$ and equation doesn't have solutions:

$$X = \varnothing \tag{4}$$

So lets combine this four statements and prove correctness of our program.

We will use Condition Rule, which could be derived from Rule of Iteration:

$$\frac{\{P \wedge A\}Q_1\{R\}, \{P \wedge \neg A\}Q\{R\}}{\{P\} \textbf{ if } A \textbf{ then } Q_1 \textbf{ else } Q \textbf{ end } \{R\}}$$

Q_1 correspond to equation (1), Q_2 to (2) and so on. Q represent nested statements:

Therefore, we prove that program holds postcondition R for all values satisfying P and it is total correct:

$\{P\}$ **if** A **then** Q_1 **else**
  if B **then** Q_2
    if C **then** Q_3 **else** Q_4 **end**
  end
end $\{R\}$

□

Actually we could replace whole program with just one assignment:

$$X = \varnothing$$

And prove it correctness according to P and R too, because universal quantification expressed that every member of domain is satisfied some predicate. This is true, when there is no members.

That's why I stressed that it doesn't help obtain right predicates.

IV. Shortcomings

V. Conclusion

References

[Hoa69] C. Antony R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.