An Axiomatic Basis for Computer Programming: Summary

Oliver Vecchini

31st January 2019

Abstract

The following is a summary of the contents, context, and consequences of a seminal paper in theoretical computer science, namely '*An Axiomatic Basis for Computer Programming*', written by C. A. R. Hoare in 1969 [1]. It is submitted as a part of the assessed coursework component of the COMP0007 'Directed Reading' module at UCL.

1 Background

While it may seem apparent that the history of program correctness should go as far back as the history of computer programming itself, even a short review of history would appear to indicate otherwise. The first ever program written for a stored-program computer, written by J. Von Neumann to illustrate usage of the EDVAC (the first stored-program computer to be designed), itself contained errors and did not work as intended: neither did the first non-trivial program executed on the EDSAC (analogous to the EDVAC) despite being written by the lead of the EDSAC construction team M. V. Wilkes. Wilkes later came upon the realization that 'a good part of the remainder of [his] life was going to be spent in finding errors in [his] own programs' [2].

The ease of writing error-ridden programs did not subside after this, however; programming errors only increased in frequency and cost through to the 1960s, with a notable example being the failure of the guidance systems of the *Mariner 1* spacecraft in 1962 due to a programming error, causing the destructive abortion of its flight and over \$18.5 million in costs (equivalent to over \$150 million in 2019) [3]. The situation was improving, with symbolic high-level programming languages mitigating errors; however, the merits of structured programming were still hotly debated, with E. W. Dijkstra's seminal letter naming the concept appearing only in 1968 [4]; potentially dangerous programming constructs such as jumps (explicitly noted by Hoare as difficult to reason about [1]) remained in common use.

It is interesting to examine the 'lineage' of papers addressing program verification written before Hoare's own. Program verification was not a commonly examined field of computer science, perhaps because of A. M. Turing's demonstration that proving termination (and thus, total correctness) of a general program was impossible [5]; however, J. McCarthy set clear goals for the topic in 1961 (discussion reserved for next paper, where content is more relevant). Several years later, two papers were written independently by P. Naur [6] and R. W. Floyd [7], and published within a short period of eachother, with striking similarity. Naur detailed an informal proof system of annotating real programs (written in ALGOL 60) with informal statements about the computer's state between each statement of the program, dubbed 'General Snapshots' (akin to the *inductive assertions* of the Hoare method); however, no formal logical basis is given, and axioms/inference rules are omitted from the 'proofs' (although triples of pre/postconditions and programs are included, akin to those found in Floyd's and Hoare's papers). On the other hand, Floyd details a system almost identical to that proposed in Hoare's paper, complete with formal predicate logic and axioms/inference rules (including the concept of the loop invariant), but tailored to flowcharts instead of textual programs. Floyd's intention of the paper was more of an incentive for development of machine-independent semantics (which would allow proofs of this kind) rather than the development of an actual proof system.

All of the above are cited by Hoare as influences on his paper, but of additional interest are papers written before even the earliest of these reflecting the same sentiments. Von Neumann and H. H. Goldstine, in writing the first 'textbook' of computation in the U.S. [8], describe several algorithms, each in terms of flowcharts (indeed, the first ever application of these to programming); they annotate the edges of these with 'assertions', as well as approximately describing an inference rule for deriving these across edges. While serving as more of a side-note than of the main body of the paper, it illustrates the consideration of algorithm correctness well before the previously cited material. Another paper with similar material was penned by Turing in 1949 [9]; however, the assertions, while fairly formal, were neither expressed in a particular logic, nor had strict rules of inference to connect them across program statements.

2 Contents of the article

In his paper [1], Hoare builds upon the work of his peers to provide a practical system for proving properties of programs: specifically, one both relevant to text-based procedural programming languages [6] and based in formal (here, predicate) logic [7]. Hoare states that the goals of this are to not only to provide such a system for its application alone, but also to provide impetus for programming languages of the time to properly (i.e. axiomatically) define the semantics of various parts of the language, at a time when many of these were implementation-defined.

Hoare first defines a small set of axioms to describe Peano arithmetic on non-negative integers in the context of computation (adapted from A. van Wijngaarden [10]), generalised to abstract particulars of implementation. Hoare then notes that the result of a program can be expressed with assertions regarding the values of variables, after the program has executed; to use this as a tool in proving properties of program, Hoare describes a construct consisting of precondition, program, and postcondition (written in modern texts as *Hoare triples*) expressed in the form $\{P\}S\{Q\}$, where S is a program, and P and Q are statements formed of predicate logic combined with the axioms of computer arithmetic, representing the pre- and post-conditions respectively. These Hoare triples express the notion that if P is true before executing S, then R will be true upon completion of S. Hoare stresses the '**upon completion**' requirement, as developing a system for proving total correctness would be fruitless (as corollary of the halting problem), hence Hoare focuses on a general system for partial correctness.

Hoare then defines the axiom (schema) $D0$, expressed here (in modern syntax):

$$D0\,\frac{}{\vdash \{A[a/x]\}S\{A\}}$$

As well as the rules of inference $D1$, $D2$, and $D3$ (ditto):

$$D1_a\,\frac{\vdash P \implies P' \qquad \vdash \{P'\}S\{Q\}}{\vdash \{P\}S\{Q\}} \qquad D1_b\,\frac{\vdash Q \implies Q' \qquad \vdash \{P\}S\{Q\}}{\vdash \{P\}S\{Q'\}}$$

$$D2\,\frac{\vdash \{P\}S_1\{Q\} \qquad \vdash \{Q\}S_2\{R\}}{\vdash \{P\}S_1;S_2\{R\}} \qquad D3\,\frac{\vdash \{I \wedge C\}S\{I\}}{\vdash \{I\}\ \textbf{while}\ C\ \textbf{do}\ S\ \{\neg C \wedge I\}}$$

Where $D0$ is the *Axiom of Assignment*, $D1$ is the *Rule of Consequence* (split into $D1_a$, pre-condition strengthening, and $D1_b$, post-condition weakening), $D2$ is the *Rule of Composition*, and $D3$ is the *Rule of Iteration*; these are all taken from Floyd's paper [7], although $D0$ is 'reversed' in the sense that substitution occurs here in the 'backwards' direction. Proofs of program properties consist of a sequence of Hoare triples generated with the inference rules, as the paper illustrates in an example.

Hoare also discusses the benefits of automated program proof other than simply producing correct programs. For instance, axiomatically specifying language features gives stronger platform-independent guarantees about their behaviour, while also explicitly indicating what behaviour *is not* guaranteed, making the task of language implementation more lenient; likewise, this concept can be applied to the specifying and writing of particular programs/functions and *their* expected behaviour.

3 Legacy

As Hoare notes, his rules of inference can be supplanted with new rules corresponding to new constructs, such as *for* loops, *it-else*, and *skip*, which have all been added to the repertoire of modern Hoare logic. His own rules can also be 'strengthened', for instance as shown by Dijkstra in strengthening the $D3$ rule [11]:

$$D3' \ \frac{\vdash [I \land C \land t \in D \land t = z] S [I \land t \in D \land t < z]}{\vdash [I \land t \in D] \ \textbf{while} \ C \ \textbf{do} \ S \ [\neg C \land I \land t \in D]}$$

This rule allows termination and hence total correctness to be proven in some cases, using the principle of least integers and the well-founded ordering of $<$ on D (the change from $\{\}$ to $[]$ reflects the additional guarantee of termination). Henceforth the term t and predicate I of rule $D3$ are referred to as the *loop variant* and *loop invariant*, respectively.

It should be noted that the system Hoare proposes (*Hoare logic*) cannot automate the process of verifying either termination (corollary of the undecidability of the halting problem) or partial verification (corollary of the undecidability of non-trivial semantic properties of programs, i.e. Rice's theorem [12]). However, Hoare logic *does* allow for human-assisted automated proofs, where the human assists with establishing loops variants and invariants, which can be used to prove termination and partial correctness, respectively. There is no guarantee that these are easy to find, or even exist; for instance, the solution of whether the formula of the Collatz conjecture terminates is dependent on finding a suitable loop variant (so far unachieved [13]) and for partial correctness, Peano arithmetic (and hence Hoare logic, which uses it) is incomplete, and hence there exist true statements which cannot be proved.

Hoare logic has seen great practical use in the time since its introduction. The axiomatic system alone would continue to be used (by Hoare himself) in the definition of the Pascal programming language [14], but the proof system would have the most influence, with numerous semi-automatic implementations of Hoare logic created soon after, with early examples including the Stanford Pascal verifier, Gypsy [15], and a verifier written within the year of Hoare's paper (indeed, by a PhD student of Floyd [16]). Separation logic, an extension of Hoare logic to allow reasoning about shared mutable data manipulated by pointers [17], is also commonly used; modern examples of verifiers using it include Smallfoot, SpaceInvader, Facebook's Infer (which in particular verifies only particular properties about memory access/validity at the benefit of full automation) [18], and Why3, a verification 'front-end' that uses SMT solvers to prove programs expressed in Hoare logic [19]. Prominent automated proofs using separation logic include those of the μC/OS-II [20] and seL4 [21] operating system kernels. Additionally, using the Curry-Howard isomorphism between proofs and programs, it has been possible to *construct* programs *from* proofs of desired properties, expressed in Hoare logic [22].

Finally, the work of Hoare has spurred further research in verification in domains other than Hoare logic; common proof assistants such as Isabelle/HOL and Coq, the proof assistant behind the formally verified C compiler CompCert, all operate on higher-order logic [23].

4 In Review

It would be unfair to attribute all of what is referred to as 'Hoare logic' to this paper alone, given that it mostly just amalgamates the work of his peers as noted above; however, given his explicit attribution of a portion of these peers (and the possibility of unawareness of the others), added to the clarity of the system he sets forth as well as the level of discussion of the benefits of such a system, it is understandable that Hoare's paper would be the most well-renowned of the collection.

References

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

[2] Brian Hayes. The discovery of debugging. *The Sciences*, 33(4):10–13, 1993.

[3] York Times By GLADWIN HILL Special to The New. For want of hyphen venus rocket is lost, Jul 28 1962.

[4] Edsger W Dijkstra. Letters to the editor: go to statement considered harmful. *Communications of the ACM*, 11(3):147–148, 1968.

[5] Alan M Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

[6] Peter Naur. Proof of algorithms by general snapshots. *BIT Numerical Mathematics*, 6(4):310–316, 1966.

[7] Robert W Floyd. Assigning meanings to programs. *Mathematical aspects of computer science*, 19(19-32):1, 1967.

[8] Herman Heine Goldstine and John Von Neumann. Planning and coding of problems for an electronic computing instrument. 1947.

[9] Alan Turing. Checking a large routine. In *The early British computer conferences*, pages 70–72. MIT Press, 1989.

[10] A Van Wijngaarden. Numerical analysis as an independent science. *BIT Numerical Mathematics*, 6(1):66–81, 1966.

[11] Edsger W Dijkstra. Guarded commands, nondeterminacy, and formal derivation of programs. In *Programming Methodology*, pages 166–175. Springer, 1978.

[12] Henry Gordon Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953.

[13] Ştefan Andrei and Cristian Masalagiu. About the collatz conjecture. *Acta Informatica*, 35(2):167–179, 1998.

[14] Charles Antony Richard Hoare and Niklaus Wirth. An axiomatic definition of the programming language pascal. *Acta Informatica*, 2(4):335–355, 1973.

[15] Didier Brocard and P Gagert. *The programming and proof system ATES: advanced techniques integration into efficient scientific software*, volume 1. Springer Science & Business Media, 2013.

[16] James C King. A program verifier. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1969.

[17] John C Reynolds. Separation logic: A logic for shared mutable data structures. In *Logic in Computer Science, 2002. Proceedings. 17th Annual IEEE Symposium on*, pages 55–74. IEEE, 2002.

[18] Open-sourcing facebook infer: Identify bugs before you ship. `https://research.fb.com/open-sourcing-facebook-infer-identify-bugs-before-you-ship/`.

[19] Jean-Christophe Filliâtre and Andrei Paskevich. Why3—where programs meet provers. In *European Symposium on Programming*, pages 125–128. Springer, 2013.

[20] Fengwei Xu, Ming Fu, Xinyu Feng, Xiaoran Zhang, Hui Zhang, and Zhaohui Li. A practical verification framework for preemptive os kernels. In *International Conference on Computer Aided Verification*, pages 59–79. Springer, 2016.

[21] seL4. sel4/l4v. `https://github.com/seL4/l4v`, Jan 2019.

[22] Iman Poernomo and John N Crossley. The curry-howard isomorphism adapted for imperative program synthesis and reasoning. In *Proceedings Of The 7th And 8th Asian Logic Conferences*, pages 343–376. World Scientific, 2003.

[23] Magnus O Myreen and Scott Owens. Proof-producing synthesis of ml from higher-order logic. In *ACM SIGPLAN Notices*, volume 47, pages 115–126. ACM, 2012.