Program Verification Using Hoare Logic

Kovács Laura Ildikó

Institute e-Austria Timişoara, Romania
Research Institute for Symbolic Computation, Hagenberg, Austria
lkovacs@risc.uni-linz.ac.at

Abstract

Program Verification is a systematic approach to proving the correctness of programs. Correctness means that the programs enjoy certain desirable properties. The aim of this paper is to provide a systematic exposition of one of the most common approaches to program verification, namely program verification using Hoare Logic, which is based on an axiomatic approach involving assertions for the verification of correctness properties of a correctness formula {P} S {Q}, where S is the program (sequence of statements), P is the precondition and Q is the postcondition of the program. We present a proof system in a syntax-directed manner, by induction on the program syntax, giving the verification rules for statements and sequence of statements. At the end we present an example, showing how the verification rules are generated using the presented verification rules from Hoare Logic.

Computer programs are becoming more and more part of systems that we use or rely in our daily lives. Therefore they should work correctly, meaning that they should satisfy their requierments. A challenge for computer science is to develop methods that ensure program correctness.

The approach presented here is usually called Hoare Logic [1] where program correctness is expressed by so-called correctness formulas. These are the so-called Hoare triples from the Hoare axiom systems, and they are of the form: {P} S {Q} where S is a program and P and Q are assertions. The assertion P is the precondition of the program and Q is the postcondition. The precondition describes the set of initial states in which the program S is started and the postcondition describes the set of desirable final or output states. The program S is a finite sequence of statements. A statement denotes single commands (like assignments, conditional, loops, etc.) of programming languages and we consider a program as a procedure (with output parameters, without return values). In our consideration, there is at most one instruction to be executed "next" (i.e. the program is deterministic), so that form a given initial state (form precondition P) only one execution sequence is generated. Also, we do not deal with recursivity.

Informally a (deterministic) program is correct if it satisfies the intended input/output relation. More precisely, we are interested in two interpretations of correctness [1],[4]

- a correctness formula {P} S {Q} is true in the sense of partial correctness if every terminating computation of S that starts in a state satisfying P terminates in a state satisfying Q.
- a correctness formula {P} S {Q} is true in the sense of total correctness if every computation of S that starts in a state satisfying P terminates and its final state satisfies Q.

Thus, in the case of partial correctness diverging computations of S are not taken into account.

Our main interest is to verify program correctness, namely we want to prove the consistency between specification and program for every possible input. To this end we investigate the correctness formulas. We present a verification system in a syntax-directed manner [3],[2],[1], and we study the partial and the total correctness. The difference between the proof systems of partial and total correctness appears -in our consideration- only:

- in the case of While loops: verification of termination is based on a specific termination term;
- in the case when the program has also procedure or function call (which may not terminate): termination of the subroutines has to be proven separately.

Corresponding to the formally specified program, we generate predicate logic formulae, called verification conditions, such that the proof of these verification conditions insures the correctness of a program. In the proof of the verification conditions we also rely on the auxiliary rule for weaker specifications (or consequence rule), namely: for any program S, having: P⇒P', Q⇒Q' and {P'} S {Q'} we also have: {P} S {Q}.

First we give the verification rule for a sequence of statements. The program specification {P} S; s {Q} where S is a program and s is one statement is correct if {P} S {R} and {R} s {Q}. Furthermore, we define the verification rules for one statement, namely we define the verification rules for each type of statement: empty statement, assignment, conditionals, for loops, while loops (with and without termination) and procedure calls.

In practice, program verification using the inference rules of Hoare Logic can be complicated, because intermediate assertions are needed between the statements. Therefore, one uses verification rules based on Weakest Precondition [4].

Example:

This is a sorting program expressed in the Theorema imperative language [4], together with its specifications and the verification assertions obtained with the help of the Theorema system.

$$Specification["SortMax", SortMax[\updownarrow a], Pre \to (|a| \geq 2),$$
$$Post \to \left(\forall_{k=1,\ldots,|a|-1}(a_k \geq a_{k+1})\right)$$

$$Program["SortMax", SortMax[\updownarrow a], IF[a_1 \leq a_2, m := a_2; a_2 := a_1; a_1 := m];$$
$$FOR[i, 2, (|a|-1), pos := i; FOR[j, i+1, |a|,$$
$$IF[a_j > a_{pos}, pos := j], (*IF*)$$
$$Invariant \to \left(\forall_{k=2,\ldots,i}(a_{k-1} \geq a_k) \bigwedge \forall_{l=i,\ldots,j-1}(a_{pos} \geq a_l)\right)]; (*FOR*)$$
$$IF[a_{pos} > a_i, m := a_{pos}; a_{pos} := a_i; a_i := m],$$
$$Invariant \to (\forall_{k=2,\ldots,i}(a_{k-1} \geq a_k))\ \ ](*FOR*),$$
$$Specification \to Specification["SortMax"]]$$

The verification rules -given by VCG-are:
Lemma (SortMax):
for any: a
(FOR.Inv1)

$$\left(\forall_{k=2,\ldots,i}(a_{k-1} \geq a_k) \bigwedge \forall_{l=i,\ldots,j-1}(a_{pos} \geq a_l)\right) \bigwedge (i+1 \leq j \wedge j \leq |a|) \Rightarrow$$
$$\left(a_j > a_{pos} \Rightarrow \forall_{k=2,\ldots,i}(a_{k-1} \geq a_k) \bigwedge \forall_{l=i,\ldots,(j+1)-1}(a_j \geq a_l)\right) \bigwedge$$
$$\left((\neg(a_j > a_{pos})) \Rightarrow \forall_{k=2,\ldots,i}(a_{k-1} \geq a_k) \bigwedge \forall_{l=i,\ldots,(j+1)-1}(a_{pos} \geq a_l)\right)$$

(FOR.Inv2)

$$\left(\forall_{k=2,\ldots,i}(a_{k-1} \geq a_k) \bigwedge \forall_{l=i,\ldots,(|a|+1)-1}(a_{pos} \geq a_l)\right) \Rightarrow$$
$$(a_{pos} > a_i \Rightarrow \forall_{k=2,\ldots,i+1}((a\|pos \leftarrow a_i\|\|i \leftarrow a_{pos}\|)_{k-1} \geq (a\|pos \leftarrow a_i\|\|i \leftarrow a_{pos}\|)_k)) \bigwedge$$
$$((\neg(a_{pos} > a_i)) \Rightarrow \forall_{k=2,\ldots,i+1}(a_{k-1} \geq a_k))$$

(FOR.Inv1)

$$\forall_{k=2,\ldots,i}(a_{k-1} \geq a_k) \bigwedge (2 \leq i \wedge i \leq (|a|-1)) \Rightarrow$$
$$\forall_{k=2,\ldots,i}(a_{k-1} \geq a_k) \bigwedge \forall_{l=i,\ldots,(i+1)-1}(a_i \geq a_l)$$

(FOR.Inv2)

$$\forall_{k=2,\ldots,(|a|-1)+1}(a_{k-1} \geq a_k) \Rightarrow$$
$$\forall_{k=1,\ldots,|a|-1}(a_k \geq a_{k+1})$$

(Init)

$$|a| \geq 2 \Rightarrow (a_1 \leq a_2 \Rightarrow \forall_{k=2,\ldots,2}((a\|2 \leftarrow a_1\|\|i \leftarrow a_2\|)_{k-1} \geq (a\|2 \leftarrow a_1\|\|1 \leftarrow a_2\|)_k)) \bigwedge$$
$$((\neg(a_1 \leq a_2)) \Rightarrow \forall_{k=2,\ldots,2}(a_{k-1} \geq a_k))$$

References

1. Krzysztof R. Apt and Ernst-Rüdiger Olderog: Verification of Sequential and Concurrent Programs. Second Edition, Springer, 1997.
2. Bruno Buchberger and Franz Lichtenberger: Mathematik für Informatiker I: Die Methode der Mathematik. Springer-Verlag, 1981.
3. C.A.Hoare: An Axiomatic Basis for Computer Programming. Comm ACM 12, 1969.
4. Martin Kirchner: Program Verification with the Mathematical Software System Theorema, Phd Thesis, Research Institute of Symbolic Computation, Linz, Austria, 1999.