Chapter 5
Forward with Hoare

Mike Gordon and Hélène Collavizza

Abstract Hoare's celebrated paper entitled "An Axiomatic Basis for Computer Programming" appeared in 1969, so the Hoare formula $P\{S\}Q$ is now 40 years old! That paper introduced Hoare Logic, which is still the basis for program verification today, but is now mechanised inside sophisticated verification systems. We aim here to give an accessible introduction to methods for proving Hoare formulae based both on the forward computation of postconditions and on the backward computation of preconditions. Although precondition methods are better known, computing postconditions provides a verification framework that encompasses methods ranging from symbolic execution to full deductive proof of correctness.

5.1 Introduction

Hoare logic [12] is a deductive system whose axioms and rules of inference provide a method of proving statements of the form $P\{S\}Q$, where S is a program statement[1] and P and Q are assertions about the values of variables. Following current practice, we use the notation $\{P\}S\{Q\}$ instead of $P\{S\}Q$. Such a 'Hoare triple' means that Q (the 'postcondition') holds in any state reached by executing S from an initial state in which P (the 'precondition') holds. Program statements may contain variables V (`X`, `Y`, `Z`, etc.), value expressions (E) and Boolean expressions (B). They are

[1] The word 'statement' is overused: Hoare statements $P\{S\}Q$ (or $\{P\}S\{Q\}$) are either true or false, but program statements are constructs that can be executed to change the values of variables. To avoid this confusion program statements are sometimes called commands.

M. Gordon (✉)
University of Cambridge Computer Laboratory William Gates Building,
15 JJ Thomson Avenue, Cambridge CB3 0FD, UK
e-mail: Mike.Gordon@cl.cam.ac.uk

H. Collavizza
Université de Nice–Sophia-Antipolis – I3S/CNRS, 930, route des Colles,
B.P. 145 06903 Sophia-Antipolis, France
e-mail: helen@polytech.unice.fr

C.B. Jones et al. (eds.), *Reflections on the Work of C.A.R. Hoare*,
DOI 10.1007/978-1-84882-912-1_5,

built out of the skip (SKIP) and assignment statements ($V := E$) using sequential composition ($S_1 ; S_2$), conditional branching (IF B THEN S_1 ELSE S_2) and WHILE-loops (WHILE B DO S). The assertions P and Q are formal logic formulae expressing properties of the values of variables.

Hoare explicitly acknowledges that his deductive system is influenced by the formal treatment of program execution due to Floyd [8].[2] There is, however, a difference, which is exhibited below using Hoare triple notation (where the notation $M[E/V]$, where M can be a formula or expression, denotes the result of substituting E for V in M).

Floyd's assignment axiom: $\vdash \{P\}\, V := E \,\{\exists v.\, (V = E[v/V]) \wedge P[v/V]\}$

Hoare's assignment axiom: $\vdash \{Q[E/V]\}\, V := E \,\{Q\}$

These are axiom schemes: any instance obtained by replacing P, Q, V, E by specific terms and formulae is an axiom. Example instances of the axiom schemes, using the replacements $P \mapsto (\mathtt{X{=}Y})$, $Q \mapsto (\mathtt{X{=}2{\times}Y})$, $V \mapsto \mathtt{X}$ and $E \mapsto (\mathtt{X{+}Y})$, are:

Floyd: $\vdash \{\mathtt{X{=}Y}\}\,\mathtt{X := X{+}Y}\,\{\exists v.\, (\mathtt{X} = ((\mathtt{X{+}Y})[v/\mathtt{X}])) \wedge ((\mathtt{X{=}Y})[v/\mathtt{X}])\}$

Hoare: $\vdash \{(\mathtt{X{=}2{\times}Y})[(\mathtt{X{+}Y})/\mathtt{X}]\}\,\mathtt{X := X{+}Y}\,\{\mathtt{X{=}2{\times}Y}\}$

which become the following if the substitutions $M[E/V]$ are performed:

Floyd: $\vdash \{\mathtt{X{=}Y}\}\,\mathtt{X := X{+}Y}\,\{\exists v.\, (\mathtt{X} = v\mathtt{{+}Y}) \wedge (v\mathtt{{=}Y})\}$

Hoare: $\vdash \{(\mathtt{X{+}Y}){=}2{\times}\mathtt{Y}\}\,\mathtt{X := X{+}Y}\,\{\mathtt{X{=}2{\times}Y}\}$

These are both equivalent to $\vdash \{\mathtt{X{=}Y}\}\,\mathtt{X := X{+}Y}\,\{\mathtt{X{=}2{\times}Y}\}$, but the reasoning in the Hoare case is a bit simpler since there is no existential quantification.

In general, the Floyd and Hoare assignment axioms are equivalent, but it is the Hoare axiom that is more widely used, since it avoids an accumulation of an existential quantifier – one for each assignment in the program.

The axioms of Hoare logic include all instances of the Hoare assignment axiom scheme given above. The rules of inference of the logic provide rules for combining Hoare triples about program statements into Hoare triples about the result of combining the statements using sequential composition, conditional branches and WHILE-loops.

5.2 Weakest Preconditions and Strongest Postconditions

A few years after Hoare's pioneering paper, Dijkstra published his influential book "A Discipline of Programming" [6] in which a framework for specifying semantics based on 'predicate transformers' – rules for transforming predicates on states – is

[2] The fascinating story of the flow of ideas between the early pioneers of programming logic is delightfully told in Jones' historical paper [16].

described. Dijkstra regarded assertions like preconditions (P above) and postconditions (Q above) as predicates on the program state, since for a given state such an assertion is either true or false. His book introduces 'weakest preconditions' as a predicate transformer semantics that treats assignment statements in a way equivalent to Hoare's assignment axiom. A dual notion of 'strongest postconditions' corresponds to Floyd's treatment of assignments. We do not know who first introduced the concept of strongest postconditions. They are discussed in Dijkstra's 1990 book with Scholten [7], but several recent papers (e.g. [11]) cite Gries' 1981 textbook [10], which only describes them in an exercise. Jones [16, p. 12] mentions that Floyd discusses a clearly related notion of 'strongest verifiable consequents' in his 1967 paper.

If $P \Rightarrow Q$ then P is said to be stronger than Q. The strongest postcondition predicate transformer for a statement S transforms a precondition predicate P to a postcondition predicate $\mathtt{sp}\,S\,P$, which is the 'strongest' predicate holding after executing S in a state satisfying precondition P. This is strongest in the sense that if Q has the property that it holds of any state resulting from executing S when P, then $\mathtt{sp}\,S\,P$ is stronger than Q – i.e. $\mathtt{sp}\,S\,P \Rightarrow Q$. The strongest postcondition predicate transformer semantics of $V := E$ is

$$\mathtt{sp}\,(V := E)\,P \;=\; \exists\, v.\,(V = E[v/V]) \wedge P[v/V]$$

The definition of strongest postcondition entails that $\{P\}\,V := E\,\{Q\}$ holds if and only if $\mathtt{sp}\,(V := E)\,P$ entails Q.

If $P \Rightarrow Q$ then Q is said to be weaker than P. The weakest precondition predicate transformer for a statement S transforms a postcondition predicate Q to a precondition predicate $\mathtt{wp}\,S\,Q$, which is the 'weakest' predicate that ensures that if a state satisfies it then after executing S the predicate Q holds.[3] This is weakest in the sense that if P has the property that executing S when P holds ensures that Q holds, then $\mathtt{wp}\,S\,Q$ is weaker than P – i.e. $P \Rightarrow \mathtt{wp}\,S\,Q$. The weakest precondition predicate transformer semantics of $V := E$ is

$$\mathtt{wp}\,(V := E)\,Q \;=\; Q[E/V]$$

The definition of weakest precondition entails that $\{P\}\,V := E\,\{Q\}$ holds if and only if P entails $\mathtt{wp}\,(V := E)\,Q$.

Equations satisfied by $\mathtt{sp}\,S\,P$ and $\mathtt{wp}\,S\,Q$ are listed in Fig. 5.1 (the equations for assignments are repeated there for convenience). These equations were originally taken as axioms. The axiomatic approach is discussed in Hoare's paper: it has the advantage of allowing a partial specification of meaning, which gives freedom to compiler writers and can make language standards more flexible. However, a purely axiomatic approach is hard to scale to complex programming constructs, as the

[3] Dijkstra defined 'weakest precondition' to require termination of S – what we are calling 'weakest precondition' he calls 'weakest liberal precondition'. Dijkstra also uses different notation: in his first book he uses $\mathtt{wlp}(S, Q)$ and $\mathtt{wp}(S, Q)$. In the later book with Scholten he uses $\mathtt{wlp}.S.Q$ and $\mathtt{wp}.S.Q$. Thus our $\mathtt{wp}\,S\,Q$ is Dijkstra's $\mathtt{wlp}(S, Q)$ (or $\mathtt{wlp}.S.Q$). However, our use of 'strongest postcondition' corresponds to Dijkstra's, though our notation differs.

$$\begin{array}{l}
\mathtt{sp}\,\mathtt{SKIP}\,P = P \\
\mathtt{wp}\,\mathtt{SKIP}\,Q = Q \\
\\
\mathtt{sp}\,(V := E)\,P = \exists v.\,(V{=}E[v/V]) \wedge P[v/V] \\
\mathtt{wp}\,(V := E)\,Q = Q[E/V] \\
\\
\mathtt{sp}\,(S_1\,;S_2)\,P = \mathtt{sp}\,S_2\,(\mathtt{sp}\,S_1\,P) \\
\mathtt{wp}\,(S_1\,;S_2)\,Q = \mathtt{wp}\,S_1\,(\mathtt{wp}\,S_2\,Q) \\
\\
\mathtt{sp}\,(\mathtt{IF}\,B\,\mathtt{THEN}\,S_1\,\mathtt{ELSE}\,S_2)\,P = (\mathtt{sp}\,S_1\,(P \wedge B)) \vee (\mathtt{sp}\,S_2\,(P \wedge \neg B)) \\
\mathtt{wp}\,(\mathtt{IF}\,B\,\mathtt{THEN}\,S_1\,\mathtt{ELSE}\,S_2)\,Q = ((\mathtt{wp}\,S_1\,Q) \wedge B) \vee ((\mathtt{wp}\,S_2\,Q) \wedge \neg B) \\
\\
\mathtt{sp}\,(\mathtt{WHILE}\,B\,\mathtt{DO}\,S)\,P = (\mathtt{sp}\,(\mathtt{WHILE}\,B\,\mathtt{DO}\,S)\,(\mathtt{sp}\,S\,(P \wedge B))) \vee (P \wedge \neg B) \\
\mathtt{wp}\,(\mathtt{WHILE}\,B\,\mathtt{DO}\,S)\,Q = (\mathtt{wp}\,S\,(\mathtt{wp}\,(\mathtt{WHILE}\,B\,\mathtt{DO}\,S)\,Q) \wedge B) \vee (Q \wedge \neg B)
\end{array}$$

Fig. 5.1 Equations defining strongest postconditions and weakest preconditions

axioms and rules get complicated and consequently hard to trust. It is now more common to give a formal semantics of programming constructs (either operational or denotational) and to derive Hoare logic axioms and rules and predicate transformer laws from this [24].

Computing $\mathtt{sp}\,(S_1\,;S_2)\,P$ using the equations in Fig. 5.1 consists of starting from a precondition P, then first computing $\mathtt{sp}\,S_1\,P$ and then applying $\mathtt{sp}\,S_2$ to the resulting predicate. This is forwards symbolic execution. In contrast, computing $\mathtt{wp}\,(S_1\,;S_2)\,Q$ proceeds backwards from a postcondition Q by first computing $\mathtt{wp}\,S_2\,Q$ and then applying $\mathtt{wp}\,S_1$ to the resulting predicate. We have more to say about forwards versus backwards later (e.g. in Section 5.6).

5.3 Proving Hoare Triples via Predicate Transformers

The relationship between Hoare triples, strongest postconditions and weakest preconditions is that $\{P\}S\{Q\}$ holds if and only if $(\mathtt{sp}\,S\,P) \Rightarrow Q$ and also if and only if $P \Rightarrow \mathtt{wp}\,S\,Q$. These implications are purely logical formulae, so a pure logic theorem prover can be used to prove them. Thus strongest postconditions and weakest preconditions each provide a way of 'compiling' the problem of verifying a Hoare triple to a purely logical problem.[4] For a loop-free program S, the equations in Fig. 5.1 can be used as left-to-right rewrites to calculate $\mathtt{sp}\,S\,P$ and $\mathtt{wp}\,S\,Q$ (if S contains a `WHILE`-loop then such rewriting may not terminate). The right-hand side of the equation for $\mathtt{sp}\,(V := E)\,P$ contains an existentially quantified conjunction, whereas the right-hand side of the equation for $\mathtt{wp}\,(V := E)\,Q$ is just $Q[E/V]$, thus

[4] Actually this is an oversimplification: mathematical constants might occur in the formula, e.g. $+$, $-$, $\times$ from the theory of arithmetic, so the theorem prover may need to go beyond pure logic and solve problems in mathematical theories.

```
R := 0;
K := 0;
IF I < J THEN K := K + 1 ELSE SKIP;
IF K = 1 ∧ ¬(I = J) THEN R := J − I ELSE R := I − J
```

Fig. 5.2 The loop-free example program AbsMinus

the formulae generated by the equations for strongest postconditions will be significantly more complex than those for weakest preconditions. This is one reason why weakest preconditions are often used in Hoare logic verifiers. The reader is invited to compare the two proofs of $\{\mathtt{I} < \mathtt{J}\}\mathtt{AbsMinus}\{\mathtt{R} = \mathtt{J}-\mathtt{I} \wedge \mathtt{I}<\mathtt{J}\}$ (the loop-free program AbsMinus is given in Fig. 5.2) obtained by manually calculating $\mathtt{sp}\,\mathtt{AbsMinus}\,(\mathtt{I} < \mathtt{J})$ and $\mathtt{wp}\,\mathtt{AbsMinus}\,(\mathtt{R} = \mathtt{J}-\mathtt{I} \wedge \mathtt{I}<\mathtt{J})$.

Although the naive calculation of $\mathtt{sp}\,SP$ using the equations in Fig. 5.1 generates complicated formulae with nested existential quantifications, a more careful calculation strategy based on symbolic execution is possible. This can be used as a theoretical framework for symbolic execution in software model checking [11].

5.4 Symbolic Execution and Strongest Postconditions

Suppose all the variables in a program S are included in the list $X_1, \ldots, X_n$ (where if $m \neq n$ then $X_m \neq X_n$). We shall specify a set of states of the program variables symbolically by logical formulae of the form:

$$\exists x_1 \cdots x_n . X_1 = e_1 \wedge \cdots \wedge X_n = e_n \wedge \phi$$

where $x_1, \ldots, x_n$ are logical variables (think of x_i as symbolically representing the initial value of program variable X_i), $e_1, \ldots, e_n$ are expressions (e_i represents the current value of X_i) and ϕ is a logical formula constraining the relationships between the values of the variables. For reasons that will become clear later, it is required that neither $e_1, \ldots, e_n$ nor ϕ contain the program variables $X_1, \ldots, X_n$, though they may well contain the variables $x_1, \ldots, x_n$. For example, the formula

$$\exists i j . \mathtt{I} = i \wedge \mathtt{J} = j \wedge i < j$$

represents the set of states in which the value of program variable I (represented symbolically by i) is less than the value of program variable J (represented symbolically by j). This formula is logically equivalent to $\mathtt{I} < \mathtt{J}$. In general, any predicate P can be written as

$$\exists x_1 \cdots x_n . X_1 = x_1 \wedge \cdots \wedge X_n = x_n \wedge P[x_1, \ldots, x_n / X_1, \ldots, X_n]$$

where $P[x_1, \ldots, x_n / X_1, \ldots, X_n]$ (corresponding to ϕ above) denotes the result of replacing all occurrences of program variable X_i by variable x_i ($1 \leq i \leq n$) that symbolically represents its value. In these formulae, program variables X_i and variables x_i, representing symbolic values, are both just logical variables. Expressions

in programs (e.g. the right-hand side of assignments) are logic terms and tests in conditionals are logic formulae. This identification of program language constructs with logic terms and formulae is one of the reasons why Hoare logic is so effective. Although this identification might appear to be confusing the different worlds of programming languages and logical systems, it does have a sound semantic basis [2, 4, 13]. The reason for adopting this form of symbolic representation is because the strongest postcondition for assignment preserves it and introduces no new existential quantifiers.

$$\mathtt{sp}\,(X_i := E)\,(\exists x_1 \cdots x_n.\, X_1 = e_1 \wedge \cdots \wedge X_n = e_n \wedge \phi)$$
$$\exists x_1 \cdots x_n.\, X_1 = e_1 \wedge \cdots \wedge X_i = E[e_1 \cdots e_n/X_1 \cdots X_n] \wedge \cdots \wedge X_n = e_n \wedge \phi$$

Thus calculating $\mathtt{sp}\,(X_i := E)$ consists of evaluating E in the current state (i.e. $E[e_1, \ldots, e_n/X_1, \ldots, X_n]$) and then updating the equation for X_i to specify that this is the new value after the symbolic execution of the assignment.

If $X_1, \ldots, X_n, x_1, \ldots, x_n$ and $e_1, \ldots, e_n$ are clear from the context, then they may be abbreviated to $\overline{X}$, $\overline{x}$ and $\overline{e}$ respectively. We may also write $\overline{X} = \overline{e}$ to mean $X_1 = e_1 \wedge \cdots \wedge X_n = e_n$. With this notation the equation above becomes

$$\mathtt{sp}\,(X_i := E)\,(\exists \overline{x}.\, \overline{X} = \overline{e} \wedge \phi)$$
$$= \exists \overline{x}.\, X_1 = e_1 \wedge \cdots \wedge X_i = E[\overline{e}/\overline{X}] \wedge \cdots \wedge X_n = e_n \wedge \phi$$

The derivation of this equation follows below (an informal justification of each line is given in brackets just after the line). The validity of the equation depends on the restriction that neither $e_1, \ldots, e_n$ nor ϕ contain the program variables $X_1, \ldots, X_n$. In addition, we also need to assume below that $v, x_1, \ldots, x_n$ and $X_1, \ldots, X_n$ are all distinct and $v, x_1, \ldots, x_n$ do not occur in E. These restrictions are assumed from now on.

$$\mathtt{sp}\,(X_i := E)\,(\exists \overline{x}.\, \overline{X} = \overline{e} \wedge \phi)$$
$$= \exists v.\, \mathtt{X}_i{=}E[v/\mathtt{X}_i] \wedge (\exists \overline{x}.\, \overline{X} = \overline{e} \wedge \phi)[v/\mathtt{X}_i]$$
(Floyd assignment rule)
$$= \exists v.\, \mathtt{X}_i{=}E[v/\mathtt{X}_i] \wedge (\exists \overline{x}.\, \mathtt{X}_1{=}e_1 \wedge \cdots \wedge v{=}e_i \wedge \cdots \wedge \mathtt{X}_n{=}e_n \wedge \phi)$$
(distinctness of variables and X_i not in $e_1, \ldots, e_n$ or ϕ)
$$= \exists v\, \overline{x}.\, \mathtt{X}_i{=}E[v/\mathtt{X}_i] \wedge \mathtt{X}_1{=}e_1 \wedge \cdots \wedge v{=}e_i \wedge \cdots \wedge \mathtt{X}_n{=}e_n \wedge \phi$$
(pulling quantifiers to front: allowed as variables distinct, $\overline{x}$ not in E)
$$= \exists \overline{x}.\, \mathtt{X}_i{=}E[e_i/\mathtt{X}_i] \wedge \mathtt{X}_1{=}e_1 \wedge \cdots \wedge (\exists v.\, v{=}e_i) \wedge \cdots \wedge \mathtt{X}_n{=}e_n \wedge \phi$$
(restricting scope of v to the only conjunct containing v)
$$= \exists \overline{x}.\, \mathtt{X}_i{=}E[e_i/\mathtt{X}_i] \wedge \mathtt{X}_1{=}e_1 \wedge \cdots \wedge \mathsf{T} \wedge \cdots \wedge \mathtt{X}_n{=}e_n \wedge \phi$$
($\exists v.\, v{=}e_i$ is true)
$$= \exists \overline{x}.\, \mathtt{X}_1{=}e_1 \wedge \cdots \wedge \mathtt{X}_i{=}E[e_i/\mathtt{X}_i] \wedge \cdots \wedge \mathtt{X}_n{=}e_n \wedge \phi$$
(eliminate T and move equation for $\mathtt{X}_i$ to where it was in the conjunction)

$= \exists \overline{x}.\, \mathtt{X}_1{=}e_1 \wedge \cdots \wedge \mathtt{X}_i{=}E[e_1,\ldots,e_n/\mathtt{X}_1,\ldots,\mathtt{X}_n] \wedge \cdots \wedge \mathtt{X}_n{=}e_n \wedge \phi$
($\overline{X} = \overline{e}$ justify replacing $\mathtt{X}_1,\ldots,\mathtt{X}_n$ in E by $e_1,\ldots,e_n$)
$= \exists \overline{x}.\, \mathtt{X}_1{=}e_1 \wedge \cdots \wedge \mathtt{X}_i{=}E[\overline{e}/\overline{\mathtt{X}}] \wedge \cdots \wedge \mathtt{X}_n{=}e_n \wedge \phi$
(definition of $[\overline{e}/\overline{\mathtt{X}}]$ notation)

Since $\mathtt{sp}\,(S_1\,;S_2)\,P \;=\; \mathtt{sp}\,S_2\,(\mathtt{sp}\,S_1\,P)$, if S_1 and S_2 are assignments and P has the form $\exists \overline{x}.\, \overline{X} = \overline{e} \wedge \phi$, then to compute $\mathtt{sp}\,(S_1\,;S_2)\,P$, one just updates the equations in the conjunction corresponding to the variable being assigned by S_1 followed by that assigned by S_2.

For conditional branches, the equation for calculating strongest postconditions is: $\mathtt{sp}\,(\mathtt{IF}\,B\,\mathtt{THEN}\,S_1\,\mathtt{ELSE}\,S_2)\,P \;=\; (\mathtt{sp}\,S_1\,(P \wedge B)) \vee (\mathtt{sp}\,S_2\,(P \wedge \neg B))$. If P has the form $\exists \overline{x}.\, \overline{X} = \overline{e} \wedge \phi$ then $P \wedge B$ and $P \wedge \neg B$ can be put into this form. The derivation is below.

$P \wedge B \;=\; (\exists \overline{x}.\, \overline{X} = \overline{e} \wedge \phi) \wedge B$
(expanding P)
$= \exists \overline{x}.\, \overline{X} = \overline{e} \wedge (\phi \wedge B)$
(allowed if $x_1,\ldots,x_n$ do not occur in B, which is assumed)
$= \exists \overline{x}.\, \overline{X} = \overline{e} \wedge (\phi \wedge B[\overline{e}/\overline{X}])$
(conjuncts $\overline{X} = \overline{e}$ justify replacing $\overline{\mathtt{X}}$ in B by $\overline{e}$)

Similarly: $P \wedge \neg B \;=\; \exists \overline{x}.\, \overline{X} = \overline{e} \wedge (\phi \wedge \neg B[\overline{e}/\overline{X}])$.

If a conditional is in a sequence then as $\mathtt{sp}\,S\,(P_1 \vee P_2) = \mathtt{sp}\,S\,P_1 \vee \mathtt{sp}\,S\,P_2$ for any program S, it follows that

$\mathtt{sp}\,((\mathtt{IF}\,B\,\mathtt{THEN}\,S_1\,\mathtt{ELSE}\,S_2)\,;S_3)\,P$
$=\; \mathtt{sp}\,(S_1\,;S_3)\,(P \wedge B) \;\vee\; \mathtt{sp}\,(S_2\,;S_3)\,(P \wedge \neg B)$

thus the calculation of the strongest postcondition of a sequence starting with a conditional can proceed by separate symbolic evaluations of each arm.

If it can be shown that either $P \wedge B$ or $P \wedge \neg B$ are false (F) then, since for any S it is the case that $\mathtt{sp}\,S\,\mathsf{F} = \mathsf{F}$, one of the disjuncts can be pruned. If such pruning is not possible, then separate evaluations for each arm must be performed. These can be organised to maximise efficiency based on heuristics (e.g. depth-first or breadth-first). As an example illustrating how symbolic evaluation can be used to compute strongest postconditions, we calculate:

$\mathtt{sp}\ \mathtt{AbsMinus}\ (\mathtt{I} < \mathtt{J}) =$

$\mathtt{sp}(\mathtt{R} := 0;$
$\quad \mathtt{K} := 0;$
$\quad \mathtt{IF}\,\mathtt{I} < \mathtt{J}\,\mathtt{THEN}\,\mathtt{K} := \mathtt{K} + 1\,\mathtt{ELSE}\,\mathtt{SKIP};$
$\quad \mathtt{IF}\,\mathtt{K} = 1 \wedge \neg(\mathtt{I} = \mathtt{J})\,\mathtt{THEN}\,\mathtt{R} := \mathtt{J} - \mathtt{I}\,\mathtt{ELSE}\,\mathtt{R} := \mathtt{I} - \mathtt{J})$
$(\exists i\,j\,k\,r.\, \mathtt{I} = i \wedge \mathtt{J} = j \wedge \mathtt{K} = k \wedge \mathtt{R} = r \wedge i < j) =$

$\mathtt{sp}(\mathtt{K} := 0;$
$\quad \mathtt{IF}\,\mathtt{I} < \mathtt{J}\,\mathtt{THEN}\,\mathtt{K} := \mathtt{K} + 1\,\mathtt{ELSE}\,\mathtt{SKIP};$

$$\begin{array}{l}
\quad \texttt{IF K} = 1 \wedge \neg(\texttt{I} = \texttt{J}) \texttt{ THEN R} := \texttt{J} - \texttt{I} \texttt{ ELSE R} := \texttt{I} - \texttt{J}) \\
\quad (\exists i\,j\,k\,r.\, \texttt{I} = i \wedge \texttt{J} = i \wedge \texttt{K} = k \wedge \texttt{R} = 0 \wedge i < j) \;= \\
\texttt{sp}(\texttt{IF I} < \texttt{J THEN K} := \texttt{K} + 1 \texttt{ ELSE SKIP}; \\
\quad \texttt{IF K} = 1 \wedge \neg(\texttt{I} = \texttt{J}) \texttt{ THEN R} := \texttt{J} - \texttt{I} \texttt{ ELSE R} := \texttt{I} - \texttt{J}) \\
\quad (\exists i\,j\,k\,r.\, \texttt{I} = i \wedge \texttt{J} = i \wedge \texttt{K} = 0 \wedge \texttt{R} = 0 \wedge i < j) \;= \\
(\texttt{sp}(\texttt{K} := \texttt{K} + 1;\ \texttt{IF K} = 1 \wedge \neg(\texttt{I} = \texttt{J}) \texttt{ THEN R} := \texttt{J} - \texttt{I} \texttt{ ELSE R} := \texttt{I} - \texttt{J}) \\
\quad (\exists i\,j\,k\,r.\, \texttt{I} = i \wedge \texttt{J} = i \wedge \texttt{K} = 0 \wedge \texttt{R} = 0 \wedge (i < j \wedge (\texttt{I} < \texttt{J})[i,j/\texttt{I},\texttt{J}])) \\
\vee \\
\texttt{sp}(\texttt{SKIP};\ \texttt{IF K} = 1 \wedge \neg(\texttt{I} = \texttt{J}) \texttt{ THEN R} := \texttt{J} - \texttt{I} \texttt{ ELSE R} := \texttt{I} - \texttt{J}) \\
\quad (\exists i\,j\,k\,r.\, \texttt{I} = i \wedge \texttt{J} = i \wedge \texttt{K} = 0 \wedge \texttt{R} = 0 \wedge (i < j \wedge \neg(\texttt{I} < \texttt{J})[i,j/\texttt{I},\texttt{J}]))
\end{array}$$

Since $(\texttt{I} < \texttt{J})[i,j/\texttt{I},\texttt{J}] = i < j$ the precondition of the second disjunct above contains the conjunct $i < j \wedge \neg(i < j)$, which is false. Thus the second disjunct can be pruned:

$$\begin{array}{l}
(\texttt{sp}(\texttt{K} := \texttt{K} + 1;\ \texttt{IF K} = 1 \wedge \neg(\texttt{I} = \texttt{J}) \texttt{ THEN R} := \texttt{J} - \texttt{I} \texttt{ ELSE R} := \texttt{I} - \texttt{J}) \\
\quad (\exists i\,j\,k\,r.\, \texttt{I} = i \wedge \texttt{J} = i \wedge \texttt{K} = 0 \wedge \texttt{R} = 0 \wedge (i < j \wedge (\texttt{I} < \texttt{J})[i,j/\texttt{I},\texttt{J}])) \\
\vee \\
\texttt{sp}(\texttt{SKIP};\ \texttt{IF K} = 1 \wedge \neg(\texttt{I} = \texttt{J}) \texttt{ THEN R} := \texttt{J} - \texttt{I} \texttt{ ELSE R} := \texttt{I} - \texttt{J}) \\
\quad (\exists i\,j\,k\,r. \\
\qquad \texttt{I} = i \wedge \texttt{J} = i \wedge \texttt{K} = 0 \wedge \texttt{R} = 0 \wedge (i < j \wedge \neg(\texttt{I} < \texttt{J})[i,j/\texttt{I},\texttt{J}]))) \;= \\
\texttt{sp}(\texttt{K} := \texttt{K} + 1;\ \texttt{IF K} = 1 \wedge \neg(\texttt{I} = \texttt{J}) \texttt{ THEN R} := \texttt{J} - \texttt{I} \texttt{ ELSE R} := \texttt{I} - \texttt{J}) \\
\quad (\exists i\,j\,k\,r.\, \texttt{I} = i \wedge \texttt{J} = i \wedge \texttt{K} = 0 \wedge \texttt{R} = 0 \wedge i < j])) \;= \\
\texttt{sp}(\texttt{IF K} = 1 \wedge \neg(\texttt{I} = \texttt{J}) \texttt{ THEN R} := \texttt{J} - \texttt{I} \texttt{ ELSE R} := \texttt{I} - \texttt{J}) \\
\quad (\exists i\,j\,k\,r.\, \texttt{I} = i \wedge \texttt{J} = i \wedge \texttt{K} = (\texttt{K+1})[0/\texttt{K}] \wedge \texttt{R} = 0 \wedge i < j])) \;= \\
(\texttt{sp}(\texttt{R} := \texttt{J} - \texttt{I}) \\
\quad (\exists i\,j\,k\,r.\, \texttt{I} = i \wedge \texttt{J} = i \wedge \texttt{K} = 1 \wedge \texttt{R} = 0 \wedge (i < j \wedge (1 = 1 \wedge \neg(i = j)))) \\
\vee \\
\texttt{sp}(\texttt{R} := \texttt{I} - \texttt{J}) \\
\quad (\exists i\,j\,k\,r. \\
\qquad \texttt{I} = i \wedge \texttt{J} = i \wedge \texttt{K} = 1 \wedge \texttt{R} = 0 \wedge (i < j \wedge \neg(1 = 1 \wedge \neg(i = j))))) \;=
\end{array}$$

The second disjunct is pruned as $i < j \wedge \neg(1 = 1 \wedge \neg(i = j))$ simplifies to F.

$$\begin{array}{l}
\texttt{sp}(\texttt{R} := \texttt{J} - \texttt{I}) \\
\quad (\exists i\,j\,k\,r.\, \texttt{I} = i \wedge \texttt{J} = i \wedge \texttt{K} = 1 \wedge \texttt{R} = 0 \wedge i < j) \\
= (\exists i\,j\,k\,r.\, \texttt{I} = i \wedge \texttt{J} = i \wedge \texttt{K} = 1 \wedge \texttt{R} = (\texttt{J}{-}\texttt{I})[i,j/\texttt{I},\texttt{J}] \wedge i < j)
\end{array}$$

The right-hand side of this equation simplifies to $\texttt{R} = \texttt{J}{-}\texttt{I} \wedge \texttt{I} < \texttt{J}$ by performing the substitution and then using properties of existential quantifiers. Thus: $\texttt{sp AbsMinus } (\texttt{I} < \texttt{J}) = \texttt{R} = \texttt{J}{-}\texttt{I} \wedge \texttt{I} < \texttt{J}$ by the derivation above.

A similar calculation by symbolic execution (but pruning different branches of the conditionals) gives: $\texttt{sp AbsMinus } (\texttt{J} \leq \texttt{I}) = (\texttt{R} = \texttt{I}{-}\texttt{J} \wedge \texttt{J} \leq \texttt{I})$.

Since $\{P\}S\{Q\}$ if and only if $(\mathtt{sp}\,S\,P) \Rightarrow Q$ it follows from the results of calculations for `AbsMinus` above that $\{\mathtt{I} < \mathtt{J}\}\mathtt{AbsMinus}\{\mathtt{R} = \mathtt{J}-\mathtt{I} \wedge \mathtt{I} < \mathtt{J}\}$ and $\{\mathtt{J} \leq \mathtt{I}\}\mathtt{AbsMinus}\{\mathtt{R} = \mathtt{I}-\mathtt{J} \wedge \mathtt{J} \leq \mathtt{I}\}$. Hence by the disjunction rule for Hoare Logic

$$\frac{\vdash \{P_1\}S\{Q_1\} \qquad \vdash \{P_2\}S\{Q_2\}}{\vdash \{P_1 \vee P_2\}S\{Q_1 \vee Q_2\}}$$

we can conclude $\vdash \{\mathsf{T}\}\mathtt{AbsMinus}\{(\mathtt{R} = \mathtt{J}-\mathtt{I} \wedge \mathtt{I} < \mathtt{J}) \vee (\mathtt{R} = \mathtt{I}-\mathtt{J} \wedge \mathtt{J} \leq \mathtt{I})\}$.

This example suggests a strategy for proving Hoare triples $\{P\}S\{Q\}$. First split P into a disjunction $P \Leftrightarrow P_1 \vee \cdots \vee P_n$ where each P_i determines a path through the program S. Then, for each i, compute $\mathtt{sp}\,P_i\,S$ by symbolic execution. Finally check that $\mathtt{sp}\,P_i\,S \Rightarrow Q$ holds for each i. If these implications all hold, then the original Hoare triple follows by the disjunction rule above. This strategy is hardly new, but explaining it as strongest postcondition calculation implemented by symbolic evaluation with Hoare logic for combining the results of the evaluations provides a nice formal foundation and also provides a link from the deductive system of Hoare logic to automated software model checking, which is often based on symbolic execution.

5.5 Backwards with Preconditions

Calculating weakest preconditions is simpler than calculating strongest postconditions because the assignment rules need just one substitution and generate no additional quantifiers: $\mathtt{wp}\,(V := E)\,Q = Q[E/V]$. There is thus no need to use formulae of the form $\exists \overline{x}.\, \overline{X} = \overline{e} \wedge \phi$ as one can calculate with postconditions of any form. Furthermore, if the McCarthy conditional notation $(B \rightarrow P \mid Q)$ is defined by

$$(B \rightarrow P \mid Q) = (P \wedge B) \vee (Q \wedge \neg B)$$

then the `wp` rule for conditionals can be expressed as

$$\mathtt{wp}\,(\mathtt{IF}\,B\,\mathtt{THEN}\,S_1\,\mathtt{ELSE}\,S_2)\,Q = (B \rightarrow \mathtt{wp}\,S_1\,Q \mid \mathtt{wp}\,S_2\,Q)$$

which simplifies the calculation of `wp`. This is illustrated using the `AbsMinus` example (see Fig. 5.2). In the calculation that follows, assignment substitutions are performed immediately.

$$\mathtt{wp}\,\mathtt{AbsMinus}\,(\mathtt{R} = \mathtt{J}-\mathtt{I} \wedge \mathtt{I} < \mathtt{J}) =$$

$$\begin{array}{l}\mathtt{wp}(\mathtt{R} := 0;\\ \quad \mathtt{K} := 0;\\ \quad \mathtt{IF}\,\mathtt{I} < \mathtt{J}\,\mathtt{THEN}\,\mathtt{K} := \mathtt{K} + 1\,\mathtt{ELSE}\,\mathtt{SKIP};\\ \quad \mathtt{IF}\,\mathtt{K} = 1 \wedge \neg(\mathtt{I} = \mathtt{J})\,\mathtt{THEN}\,\mathtt{R} := \mathtt{J} - \mathtt{I}\,\mathtt{ELSE}\,\mathtt{R} := \mathtt{I} - \mathtt{J})\\ (\mathtt{R} = \mathtt{J}-\mathtt{I} \wedge \mathtt{I} < \mathtt{J}) =\end{array}$$

$$\begin{array}{l}\mathtt{wp}(\mathtt{R} := 0;\\ \quad \mathtt{K} := 0;\\ \quad \mathtt{IF\,I} < \mathtt{J\,THEN\,K} := \mathtt{K} + 1\,\mathtt{ELSE\,SKIP})\\ \quad (\mathtt{K} = 1 \wedge \neg(\mathtt{I} = \mathtt{J}) \rightarrow \mathtt{J{-}I} = \mathtt{J{-}I} \wedge \mathtt{I{<}J} \mid \mathtt{J{-}I} = \mathtt{I{-}J} \wedge \mathtt{I{<}J}) =\end{array}$$

$$\begin{array}{l}\mathtt{wp}(\mathtt{R} := 0;\\ \quad \mathtt{K} := 0;\\ (\mathtt{I{<}J} \rightarrow (\mathtt{K{+}1} = 1 \wedge \neg(\mathtt{I} = \mathtt{J}) \rightarrow \mathtt{J{-}I} = \mathtt{J{-}I} \wedge \mathtt{I{<}J} \mid \mathtt{J{-}I} = \mathtt{I{-}J} \wedge \mathtt{I{<}J})\\ \quad \mid (\mathtt{K} = 1 \wedge \neg(\mathtt{I} = \mathtt{J}) \rightarrow \mathtt{J{-}I} = \mathtt{J{-}I} \wedge \mathtt{I{<}J} \mid \mathtt{J{-}I} = \mathtt{I{-}J} \wedge \mathtt{I{<}J})) =\end{array}$$

$$\begin{array}{l}(\mathtt{I{<}J} \rightarrow (0{+}1 = 1 \wedge \neg(\mathtt{I} = \mathtt{J}) \rightarrow \mathtt{J{-}I} = \mathtt{J{-}I} \wedge \mathtt{I{<}J} \mid \mathtt{J{-}I} = \mathtt{I{-}J} \wedge \mathtt{I{<}J})\\ \quad \mid (0 = 1 \wedge \neg(\mathtt{I} = \mathtt{J}) \rightarrow \mathtt{J{-}I} = \mathtt{J{-}I} \wedge \mathtt{I{<}J} \mid \mathtt{J{-}I} = \mathtt{I{-}J} \wedge \mathtt{I{<}J}))\end{array}$$

This calculation can be simplified on-the-fly: $(\mathtt{K} + 1 = 1)$ simplifies to $(\mathtt{K} = 0)$, $(\mathtt{J{-}I} = \mathtt{J{-}I})$ simplifies to T and $(\mathtt{J{-}I} = \mathtt{I{-}J} \wedge \mathtt{I{<}J})$ simplifies to F.

$$\begin{array}{l}\mathtt{wp}(\mathtt{R} := 0;\\ \quad \mathtt{K} := 0;\\ \quad \mathtt{IF\,I} < \mathtt{J\,THEN\,K} := \mathtt{K} + 1\,\mathtt{ELSE\,SKIP};\\ \quad \mathtt{IF\,K} = 1 \wedge \neg(\mathtt{I} = \mathtt{J})\,\mathtt{THEN\,R} := \mathtt{J} - \mathtt{I}\,\mathtt{ELSE\,R} := \mathtt{I} - \mathtt{J})\\ \quad (\mathtt{R} = \mathtt{J{-}I} \wedge \mathtt{I{<}J}) =\end{array}$$

$$\begin{array}{l}\mathtt{wp}(\mathtt{R} := 0;\\ \quad \mathtt{K} := 0;\\ \quad \mathtt{IF\,I} < \mathtt{J\,THEN\,K} := \mathtt{K} + 1\,\mathtt{ELSE\,SKIP})\\ \quad (\mathtt{K} = 1 \wedge \neg(\mathtt{I} = \mathtt{J}) \rightarrow \mathtt{I{<}J} \mid \mathsf{F}) =\end{array}$$

$$\begin{array}{l}\mathtt{wp}(\mathtt{R} := 0;\\ \quad \mathtt{K} := 0;\\ \quad (\mathtt{I{<}J} \rightarrow (\mathtt{K} = 0 \wedge \neg(\mathtt{I} = \mathtt{J}) \rightarrow \mathtt{I{<}J} \mid \mathsf{F})\\ \qquad \mid (\mathtt{K} = 1 \wedge \neg(\mathtt{I} = \mathtt{J}) \rightarrow \mathtt{I{<}J} \mid \mathsf{F})) =\end{array}$$

$$(\mathtt{I{<}J} \rightarrow (1 = 1 \wedge \neg(\mathtt{I} = \mathtt{J}) \rightarrow \mathtt{I{<}J} \mid \mathsf{F}) \mid (0 = 1 \wedge \neg(\mathtt{I} = \mathtt{J}) \rightarrow \mathtt{I{<}J} \mid \mathsf{F}))$$

The last formula above is $\mathtt{wp\,AbsMinus}\,(\mathtt{R} = \mathtt{J{-}I} \wedge \mathtt{I} < \mathtt{J})$ and simplifies to $(\mathtt{I{<}J} \rightarrow \mathsf{T} \mid \mathsf{F})$, which simplifies to $\mathtt{I{<}J}$.

The Hoare triple $\{\mathtt{I{<}J}\}\mathtt{AbsMinus}\{\mathtt{R} = \mathtt{J} - \mathtt{I} \wedge \mathtt{I{<}J}\}$ then follows from $\mathtt{wp\,AbsMinus}\,(\mathtt{R} = \mathtt{J{-}I} \wedge \mathtt{I} < \mathtt{J}) = \mathtt{I{<}J}$.

5.6 Forwards Versus Backwards

It seems clear from the example in the preceding section that proving $\{P\}S\{Q\}$ by calculating $\mathtt{wp}\,SQ$ is simpler than proving it by calculating $\mathtt{sp}\,SP$, indeed many automated Hoare logic verifiers work by calculating weakest preconditions. There are, however, several applications where strongest postconditions have a role.

One such application is 'reverse engineering' where, given a precondition, one tries to deduce what a program (e.g. legacy code) does by discovering a postcondition by symbolic execution [9]. Related to this is the use of symbolic execution for testing [18]. Yet another application is to verify a given Hoare triple by symbolically executing separate paths and combining the results (this approach has already been outlined at the end of Section 5.2). This is a form of software model checking, where loops are unwound some number of times to create loop-free straight line code, the strongest postcondition is calculated and then an automatic tool (like an SMT solver) used to show that this entails the given postcondition.[5] The hope is that one runs enough of the program to expose significant bugs. In general, code with loops cannot be unwound to equivalent straight line code, so although this approach is a powerful bug-finding method it cannot (without further analysis) be used for full proof of correctness. An advantage of symbolic evaluation is that one can use the symbolic representation of the current state to resolve conditional branches and hence prune paths. The extreme case of this is when the initial precondition specifies a unique starting state, so that symbolic execution collapses to normal 'ground' execution. Thus calculating strongest postconditions by symbolic execution provides a smooth transition from testing to full verification: by weakening the initial precondition one can make the verification cover more initial states.

5.7 Loops and Invariants

There is no general way to calculate strongest postconditions or weakest preconditions for `WHILE`-loops. Rewriting with the equations in Fig. 5.1 may not terminate, so if S contains loops then the strategies for proving $\{P\}S\{Q\}$ by calculating $\mathtt{sp}\,P\,S$ or $\mathtt{wp}\,S\,Q$ will not work. Instead, we define 'approximate' versions of `sp` and `wp` called, respectively, `asp` and `awp`, together with formulae ('verification conditions') $\mathtt{svc}\,P\,S$ and $\mathtt{wvc}\,S\,Q$ with the properties that

$$\vdash \mathtt{svc}\,S\,P \Rightarrow \{P\}S\{\mathtt{asp}\,S\,P\}$$

$$\vdash \mathtt{wvc}\,S\,Q \Rightarrow \{\mathtt{awp}\,S\,Q\}S\{Q\}$$

See Fig. 5.3 for equations specifying `asp` and `svc` and Fig. 5.4 for `awp` and `wvc`. Let ϕ be a formula and $x_1, \ldots, x_n$ all the free variables in ϕ. It is clear that if S is loop-free then $\mathtt{svc}\,S\,P$ and $\mathtt{wvc}\,S\,Q$ are true (T) and also $\mathtt{asp}\,S\,P = \mathtt{sp}\,S\,P$ and $\mathtt{awp}\,S\,Q = \mathtt{wp}\,S\,Q$. Thus for loop-free programs the 'approximate' predicate transformers are equivalent to the exact ones.

[5] State-of-the-art bounded model checkers [1,3] generate the strongest postcondition using similar rules to those given in Fig. 5.1. However, they first transform programs into SSA (Static Single Assignment) form [5] and avoid the explicit use of existential quantifiers generated by assignments. The approach in this paper seems equivalent to the use of SSA, but we have not worked out a clean account of this. A feature of our method is that it applies directly to programs without requiring any preprocessing.

$$\texttt{asp}\,\texttt{SKIP}\,P = P$$
$$\texttt{svc}\,\texttt{SKIP}\,P = \mathsf{T}$$

$$\texttt{asp}\,(V := E)\,P = \exists v.\,(V{=}E[v/V]) \wedge P[v/V]$$
$$\texttt{svc}\,(V := E)\,P = \mathsf{T}$$

$$\texttt{asp}\,(S_1\,;S_2)\,P = \texttt{asp}\,S_2\,(\texttt{asp}\,S_1\,P)$$
$$\texttt{svc}\,(S_1\,;S_2)\,P = \texttt{svc}\,S_1\,P \wedge \texttt{svc}\,S_2\,(\texttt{asp}\,S_1\,P)$$

$$\texttt{asp}\,(\texttt{IF}\,B\,\texttt{THEN}\,S_1\,\texttt{ELSE}\,S_2)\,P = \texttt{asp}\,S_1\,(P \wedge B) \vee \texttt{asp}\,S_2\,(P \wedge \neg B)$$
$$\texttt{svc}\,(\texttt{IF}\,B\,\texttt{THEN}\,S_1\,\texttt{ELSE}\,S_2)\,P = (\texttt{UNSAT}(P \wedge B) \vee \texttt{svc}\,S_1\,(P \wedge B)) \wedge (\texttt{UNSAT}(P \wedge \neg B) \vee \texttt{svc}\,S_2\,(P \wedge \neg B))$$

$$\texttt{asp}\,(\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S)\,P = R \wedge \neg B$$
$$\texttt{svc}\,(\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S)\,P = (P \Rightarrow R) \wedge (\texttt{asp}\,S\,(R \wedge B) \Rightarrow R) \wedge \texttt{svc}\,S\,(R \wedge B)$$

Fig. 5.3 Approximate strongest postconditions and verification conditions

$$\texttt{awp}\,\texttt{SKIP}\,Q = Q$$
$$\texttt{wvc}\,\texttt{SKIP}\,Q = \mathsf{T}$$

$$\texttt{awp}\,(V := E)\,Q = Q[E/V]$$
$$\texttt{wvc}\,(V := E)\,Q = \mathsf{T}$$

$$\texttt{awp}\,(S_1\,;S_2)\,Q = \texttt{awp}\,S_1\,(\texttt{awp}\,S_2\,Q)$$
$$\texttt{wvc}\,(S_1\,;S_2)\,Q = \texttt{wvc}\,S_1\,(\texttt{awp}\,S_2\,Q) \wedge \texttt{wvc}\,S_2\,Q$$

$$\texttt{awp}\,(\texttt{IF}\,B\,\texttt{THEN}\,S_1\,\texttt{ELSE}\,S_2)\,Q = (B \rightarrow \texttt{awp}\,S_1\,Q \mid \texttt{awp}\,S_2\,Q)$$
$$\texttt{wvc}\,(\texttt{IF}\,B\,\texttt{THEN}\,S_1\,\texttt{ELSE}\,S_2)\,Q = \texttt{TAUT}(Q) \vee (\texttt{wvc}\,S_1\,Q \wedge \texttt{wvc}\,S_2\,Q)$$

$$\texttt{awp}\,(\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S)\,Q = R$$
$$\texttt{wvc}\,(\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S)\,Q = (R \wedge B \Rightarrow \texttt{awp}\,S\,R) \wedge (R \wedge \neg B \Rightarrow Q) \wedge \texttt{wvc}\,S\,R$$

Fig. 5.4 Approximate weakest postconditions and verification conditions

In Fig. 5.3, the operator UNSAT is true if its argument is unsatisfiable: $\texttt{UNSAT}(\phi) = \neg\,\exists\,x_1 \cdots x_n.\,\phi$. The operator TAUT is true if its argument is a tautology: $\texttt{TAUT}(\phi) = \forall\,x_1 \cdots x_n.\,\phi$. The relation between UNSAT and TAUT is: $\texttt{UNSAT}(\phi) = \texttt{TAUT}(\neg\phi)$. Point 4 of the first proof in the appendix uses the fact that $\{P\}S\{Q\}$ holds vacuously if $\texttt{UNSAT}(P)$. Point 4 of the second proof in the appendix uses the dual fact that $\{P\}S\{Q\}$ holds vacuously if $\texttt{TAUT}(Q)$.

To establish $\{P\}S\{Q\}$, the Hoare logic 'Rules of Consequence' ensure that it is sufficient to prove either the conjunction $(\texttt{svc}\,S\,P) \wedge (\texttt{asp}\,S\,P \Rightarrow Q)$ or the conjunction $(\texttt{wvc}\,S\,Q) \wedge (P \Rightarrow \texttt{awp}\,S\,Q)$. The early development of mechanised program verification uses ideas similar to weakest preconditions to generate verification conditions [14, 15, 17, 19, 20, 23]. Strongest postconditions are less used for reasoning about programs containing loops, but generalisations of symbolic execution for them have been developed [22].

Reasoning about loops usually requires invariants to be supplied (either by a human or by some invariant-finding tool). Hoare logic provides the following WHILE-rule (which is a combination of Hoare's original 'Rule of Iteration' and his 'Rules of Consequence'). Following standard practice, we have added the invariant R as an annotation in curly brackets just before the body S of the WHILE-loop

$$\frac{\vdash P \Rightarrow R \qquad \vdash \{R \wedge B\}S\{R\} \qquad R \wedge \neg B \Rightarrow Q}{\vdash \{P\}\,\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S\{Q\}}$$

This rule is the logical basis underlying methods based on invariants for verifying WHILE-loops.

The ideas underlying asp, svc, awp and wvc are old and mostly very well known. The contribution here is a repackaging of these standard methods in a somewhat more uniform framework. The properties stated above connecting asp and svc, and awp and wvc are easily verified by structural induction. For completeness the proofs are given in an appendix.

The equations for asp in Fig 5.3 are the same as those for sp in Fig. 5.1, except for the additional equation for $\texttt{asp}\,\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S\,P$. For symbolic execution, as described in Section 5.4, this equation can be written as

$$\begin{aligned}&\texttt{asp}\,(\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S)\,(\exists\overline{x}.\,X_1 = e_1 \wedge \cdots \wedge X_n = e_n \wedge \phi)\\ &= \exists\overline{x}.\,\overline{X} = \overline{x} \wedge (R \wedge \neg B)[\overline{x}/\overline{X}]\end{aligned}$$

Thus symbolically executing $\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S$ consists in throwing away the precondition and restarting in a new symbolic state corresponding to the state specified as holding after the WHILE-loop by the Hoare rule. This is justified if the verification conditions for the WHILE-loop hold, namely:

$$\begin{aligned}&\texttt{svc}\,(\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S)\,(\exists\overline{x}.\,X_1 = e_1 \wedge \cdots \wedge X_n = e_n \wedge \phi)\\ &= ((\exists\overline{x}.\,X_1 = e_1 \wedge \cdots \wedge X_n = e_n \wedge \phi) \Rightarrow R)\\ &\quad \wedge\ (\texttt{asp}\,S\,(R \wedge B) \Rightarrow R) \wedge \texttt{svc}\,S\,(R \wedge B)\\ &= (\forall\overline{x}.\,\phi \Rightarrow R[\overline{e}/\overline{X}]) \wedge (\texttt{asp}\,S\,(R \wedge B) \Rightarrow R) \wedge \texttt{svc}\,S\,(R \wedge B)\end{aligned}$$

This says that the precondition must entail the invariant evaluated in the state when the loop is started ($R[\overline{e}/\overline{X}]$), the invariant R must really be an invariant ($\texttt{asp}\,S\,(R \wedge B) \Rightarrow R$) and any verification conditions when checking R is an invariant must hold ($\texttt{svc}\,S\,(R \wedge B)$). To verify that R is an invariant a recursive symbolic execution of the loop body, starting in a state satisfying $R \wedge B$, is performed. Note that:

$$\texttt{asp}\,S\,(R \wedge B) \;=\; \texttt{asp}\,S\,(\exists\overline{x}.\,\overline{X} = \overline{x} \wedge (R \wedge B)[\overline{x}/\overline{X}])$$

The equations for symbolic execution and verification conditions on symbolic state formulae are given in Fig 5.5. Note that $\texttt{UNSAT}(\exists\overline{x}.\,\overline{X} = \overline{e} \wedge \phi) = \texttt{UNSAT}(\phi)$.

As an example, consider the program Div in Fig. 5.6. For simplicity, assume the values of the variables in Div (i.e. R, X and ERR) are non-negative integers. First we compute asp Div (Y=0).

$$\texttt{asp SKIP}\ (\exists \bar{x}.\ \overline{X} = \bar{e} \wedge \phi) = (\exists \bar{x}.\ \overline{X} = \bar{e} \wedge \phi)$$

$$\texttt{svc SKIP}\ (\exists \bar{x}.\ \overline{X} = \bar{e} \wedge \phi) = \mathsf{T}$$

$$\texttt{asp}\ (X_i := E)\ (\exists \bar{x}.\ \overline{X} = \bar{e} \wedge \phi)$$
$$= \exists \bar{x}.\ X_1 = e_1 \wedge \ldots \wedge X_i = E[\bar{e}/\overline{X}] \wedge \ldots \wedge X_n = e_n \wedge \phi$$

$$\texttt{svc}\ (X_i := E)\ (\exists \bar{x}.\ \overline{X} = \bar{e} \wedge \phi) = \mathsf{T}$$

$$\texttt{asp}\ (S_1\ ;\ S_2)\ (\exists \bar{x}.\ \overline{X} = \bar{e} \wedge \phi) = \texttt{asp}\ S_2\ (\texttt{asp}\ S_1\ (\exists \bar{x}.\ \overline{X} = \bar{e} \wedge \phi))$$

$$\texttt{svc}\ (S_1\ ;\ S_2)\ (\exists \bar{x}.\ \overline{X} = \bar{e} \wedge \phi) = \texttt{svc}\ S_1\ (\exists \bar{x}.\ \overline{X} = \bar{e} \wedge \phi)\ \wedge\ \texttt{svc}\ S_2\ (\texttt{asp}\ S_1\ (\exists \bar{x}.\ \overline{X} = \bar{e} \wedge \phi))$$

$$\texttt{asp}\ (\texttt{IF}\ B\ \texttt{THEN}\ S_1\ \texttt{ELSE}\ S_2)\ (\exists \bar{x}.\ \overline{X} = \bar{e} \wedge \phi)$$
$$= \texttt{asp}\ S_1\ (\exists \bar{x}.\ \overline{X} = \bar{e} \wedge (\phi \wedge B[\bar{e}/\overline{X}]))\ \vee\ \texttt{asp}\ S_2\ (\exists \bar{x}.\ \overline{X} = \bar{e} \wedge (\phi \wedge \neg B[\bar{e}/\overline{X}]))$$

$$\texttt{svc}\ (\texttt{IF}\ B\ \texttt{THEN}\ S_1\ \texttt{ELSE}\ S_2)\ (\exists \bar{x}.\ \overline{X} = \bar{e} \wedge \phi)$$
$$= (\texttt{UNSAT}(\phi \wedge B[\bar{e}/\overline{X}]) \vee \texttt{svc}\ S_1\ (\exists \bar{x}.\ \overline{X} = \bar{e} \wedge (\phi \wedge B[\bar{e}/\overline{X}])))$$
$$\wedge\ (\texttt{UNSAT}(\phi \wedge \neg B[\bar{e}/\overline{X}]) \vee \texttt{svc}\ S_2\ (\exists \bar{x}.\ \overline{X} = \bar{e} \wedge (\phi \wedge \neg B[\bar{e}/\overline{X}])))$$

$$\texttt{asp}\ (\texttt{WHILE}\ B\ \texttt{DO}\{R\}\ S)\ (\exists \bar{x}.\ X_1 = e_1 \wedge \ldots \wedge X_n = e_n \wedge \phi)$$
$$= \exists \bar{x}.\ \overline{X} = \bar{x} \wedge (R \wedge \neg B)[\bar{x}/\overline{X}]$$

$$\texttt{svc}\ (\texttt{WHILE}\ B\ \texttt{DO}\{R\}\ S)\ (\exists \bar{x}.\ X_1 = e_1 \wedge \ldots \wedge X_n = e_n \wedge \phi) =$$
$$= (\forall \bar{x}.\ \phi \Rightarrow R[\bar{e}/\overline{X}])\ \wedge\ (\texttt{asp}\ S\ (R \wedge B) \Rightarrow R)\ \wedge\ \texttt{svc}\ S\ (R \wedge B)$$

Fig. 5.5 Approximate strongest postconditions and verification conditions

```
R := X; Q := 0; ERR := 0;
IF Y=0 THEN ERR := 1 ELSE WHILE Y < R DO{X = R + Y × Q} (R := R−Y; Q := 1+Q)
```

Fig. 5.6 The example program `Div`

Let S be $\texttt{WHILE Y < R DO}\{\texttt{X = R + Y} \times \texttt{Q}\}\ (\texttt{R := R−Y; Q := 1+Q})$, then:

$$\texttt{asp}\ (\texttt{R := X; Q := 0; ERR := 0; IF Y=0 THEN ERR := 1 ELSE}\ S)\ (\texttt{Y=0}) =$$

$$\texttt{asp}\ (\texttt{R := X; Q := 0; ERR := 0; IF Y=0 THEN ERR := 1 ELSE}\ S)\ (\exists xyqre.\ \texttt{X=}x \wedge \texttt{Y=}y \wedge \texttt{Q=}q \wedge \texttt{R=}r \wedge \texttt{ERR=}e \wedge y=0) =$$

$$\texttt{asp}\ (\texttt{IF Y=0 THEN ERR := 1 ELSE}\ S)\ (\exists xyqre.\ \texttt{X=}x \wedge \texttt{Y=}y \wedge \texttt{Q=}0 \wedge \texttt{R=}x \wedge \texttt{ERR=}0 \wedge y=0) =$$

$$\texttt{asp}\ (\texttt{ERR := 1})\ (\exists xyqre.\ \texttt{X=}x \wedge \texttt{Y=}y \wedge \texttt{Q=}0 \wedge \texttt{R=}x \wedge \texttt{ERR=}0 \wedge y=0 \wedge (y=0))$$
$$\vee$$
$$\texttt{asp}\ S\ (\exists xyqre.\ \texttt{X=}x \wedge \texttt{Y=}y \wedge \texttt{Q=}0 \wedge \texttt{R=}x \wedge \texttt{ERR=}0 \wedge y=0 \wedge \neg(y=0)) =$$

$$\texttt{asp}\ (\texttt{ERR := 1})\ (\exists xyqre.\ \texttt{X=}x \wedge \texttt{Y=}y \wedge \texttt{Q=}0 \wedge \texttt{R=}x \wedge \texttt{ERR=}0 \wedge y=0 \wedge (y=0)) =$$

$$(\exists xyqre.\ \texttt{X=}x \wedge \texttt{Y=}y \wedge \texttt{Q=}0 \wedge \texttt{R=}x \wedge \texttt{ERR=}1 \wedge y=0 \wedge (y=0)) =$$

$$\texttt{X=R} \wedge \texttt{Y=}0 \wedge \texttt{ERR=}1$$

Next we compute $\mathtt{svc}\ \mathtt{Div}\ (\mathtt{Y}{=}0)$ (simplifying $\mathsf{T} \wedge \phi$ to ϕ on-the-fly).

$$\begin{array}{l}
\mathtt{svc} \\
(\mathtt{R} := \mathtt{X};\ \mathtt{Q} := 0;\ \mathtt{ERR} := 0;\ \mathtt{IF}\ \mathtt{Y}{=}0\ \mathtt{THEN}\ \mathtt{ERR} := 1\ \mathtt{ELSE}\ S) \\
(\mathtt{Y}{=}0) = \\
\mathtt{svc} \\
(\mathtt{R} := \mathtt{X};\ \mathtt{Q} := 0;\ \mathtt{ERR} := 0;\ \mathtt{IF}\ \mathtt{Y}{=}0\ \mathtt{THEN}\ \mathtt{ERR} := 1\ \mathtt{ELSE}\ S) \\
(\exists xyqre.\, \mathtt{X}{=}x \wedge \mathtt{Y}{=}y \wedge \mathtt{Q}{=}q \wedge \mathtt{R}{=}r \wedge \mathtt{ERR}{=}e \wedge y{=}0) = \\
\mathtt{svc} \\
(\mathtt{IF}\ \mathtt{Y}{=}0\ \mathtt{THEN}\ \mathtt{ERR} := 1\ \mathtt{ELSE}\ S) \\
(\exists xyqre.\, \mathtt{X}{=}x \wedge \mathtt{Y}{=}y \wedge \mathtt{Q}{=}0 \wedge \mathtt{R}{=}x \wedge \mathtt{ERR}{=}0 \wedge y{=}0) = \\
(\mathtt{UNSAT}(y = 0 \wedge y = 0) \vee \mathtt{svc}\,(\mathtt{ERR} := 1)\,(\cdots)) \\
\wedge \\
(\mathtt{UNSAT}(y = 0 \wedge \neg(y = 0)) \vee \mathtt{svc}\,S\,(\cdots)) \ = \ (\mathsf{F} \vee \mathsf{T}) \wedge (\mathsf{T} \vee \cdots) \ = \ \mathsf{T}
\end{array}$$

Thus $\{\mathtt{Y}{=}0\}\mathtt{Div}\{\mathtt{X}{=}\mathtt{R} \wedge \mathtt{Y}{=}0 \wedge \mathtt{ERR}{=}1\}$, since $\forall\, SP.\, \mathtt{svc}\, SP \Rightarrow \{P\}S\{\mathtt{asp}\, SP\}$. Whilst this might seem to be a very heavyweight derivation of a trivial case, the point is that the derivation is a forward symbolic execution with some on-the-fly simplification. This simplification enables the calculation of `asp` and `svc` for the `WHILE`-loop to be avoided.

For the $\mathtt{Y} > 0$ case, it is necessary to analyse the loop, which we now do.

$$\begin{array}{l}
\mathtt{asp} \\
(\mathtt{R} := \mathtt{X};\ \mathtt{Q} := 0;\ \mathtt{ERR} := 0;\ \mathtt{IF}\ \mathtt{Y}{=}0\ \mathtt{THEN}\ \mathtt{ERR} := 1\ \mathtt{ELSE}\ S) \\
(\mathtt{Y}{>}0) = \\
\mathtt{asp} \\
(\mathtt{R} := \mathtt{X};\ \mathtt{Q} := 0;\ \mathtt{ERR} := 0;\ \mathtt{IF}\ \mathtt{Y}{=}0\ \mathtt{THEN}\ \mathtt{ERR} := 1\ \mathtt{ELSE}\ S) \\
(\exists xyqre.\, \mathtt{X}{=}x \wedge \mathtt{Y}{=}y \wedge \mathtt{Q}{=}q \wedge \mathtt{R}{=}r \wedge \mathtt{ERR}{=}e \wedge y{>}0) = \\
\mathtt{asp} \\
(\mathtt{IF}\ \mathtt{Y}{=}0\ \mathtt{THEN}\ \mathtt{ERR} := 1\ \mathtt{ELSE}\ S) \\
(\exists xyqre.\, \mathtt{X}{=}x \wedge \mathtt{Y}{=}y \wedge \mathtt{Q}{=}0 \wedge \mathtt{R}{=}x \wedge \mathtt{ERR}{=}0 \wedge y{>}0) = \\
\mathtt{asp}\,(\mathtt{ERR} := 1)\,(\exists xyqre.\, \mathtt{X}{=}x \wedge \mathtt{Y}{=}y \wedge \mathtt{Q}{=}0 \wedge \mathtt{R}{=}x \wedge \mathtt{ERR}{=}0 \wedge y{>}0 \wedge (y{=}0)) \\
\vee \\
\mathtt{asp}\,S\,(\exists xyqre.\, \mathtt{X}{=}x \wedge \mathtt{Y}{=}y \wedge \mathtt{Q}{=}0 \wedge \mathtt{R}{=}x \wedge \mathtt{ERR}{=}0 \wedge y{>}0 \wedge \neg(y{=}0)) \ = \\
\mathtt{asp}\,S\,(\exists xyqre.\, \mathtt{X}{=}x \wedge \mathtt{Y}{=}y \wedge \mathtt{Q}{=}0 \wedge \mathtt{R}{=}x \wedge \mathtt{ERR}{=}0 \wedge y{>}0 \wedge \neg(y{=}0))
\end{array}$$

Since S is $\mathtt{WHILE}\ \mathtt{Y} < \mathtt{R}\ \mathtt{DO}\{\mathtt{X} = \mathtt{R} + \mathtt{Y} \times \mathtt{Q}\}\ (\cdots)$, it follows (see Fig. 5.5) that

$$\begin{array}{l}
\mathtt{asp}\ \mathtt{Div}\ (\mathtt{Y}{>}0) \ = \\
\mathtt{asp}\,S\,(\exists xyqre.\, \mathtt{X}{=}x \wedge \mathtt{Y}{=}y \wedge \mathtt{Q}{=}0 \wedge \mathtt{R}{=}x \wedge \mathtt{ERR}{=}0 \wedge y{>}0 \wedge \neg(y{=}0)) \ = \\
(\exists xyqre.\, \mathtt{X}{=}x \wedge \mathtt{Y}{=}y \wedge \mathtt{Q}{=}q \wedge \mathtt{R}{=}r \wedge \mathtt{ERR}{=}e \wedge (x = r{+}y{\times}q \wedge \neg(y{<}r))) = \\
(\mathtt{X} = \mathtt{R} + \mathtt{Y} \times \mathtt{Q} \wedge \neg(\mathtt{Y} < \mathtt{R}))
\end{array}$$

Thus if svc Div (Y>0) holds (which it does, see below) then

$$\{\texttt{Y>0}\}\texttt{Div}\{\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q} \wedge \neg(\texttt{Y} < \texttt{R})\}$$

To verify svc Div (Y>0), first calculate:

$$\begin{array}{l}
\texttt{svc} \\
(\texttt{R := X; Q := 0; ERR := 0; IF Y=0 THEN ERR := 1 ELSE } S) \\
(\texttt{Y>0}) = \\
\texttt{svc} \\
(\texttt{R := X; Q := 0; ERR := 0; IF Y=0 THEN ERR := 1 ELSE } S) \\
(\exists xyqre.\, \texttt{X}{=}x \wedge \texttt{Y}{=}y \wedge \texttt{Q}{=}q \wedge \texttt{R}{=}r \wedge \texttt{ERR}{=}e \wedge y{>}0) = \\
\texttt{svc} \\
(\texttt{IF Y=0 THEN ERR := 1 ELSE } S) \\
(\exists xyqre.\, \texttt{X}{=}x \wedge \texttt{Y}{=}y \wedge \texttt{Q}{=}0 \wedge \texttt{R}{=}x \wedge \texttt{ERR}{=}0 \wedge y{>}0) = \\
(\texttt{UNSAT}(y > 0 \wedge y = 0) \vee \texttt{svc}\,(\texttt{ERR := 1})\,(\cdots)) \\
\wedge \\
(\texttt{UNSAT}(y > 0 \wedge \neg(y = 0)) \vee \texttt{svc}\,S\,(\cdots)) = (\mathsf{T} \vee \mathsf{T}) \wedge (\mathsf{F} \vee \texttt{svc}\,S\,(\cdots)) = \\
\texttt{svc}\,S\,(\exists xyqre.\, \texttt{X}{=}x \wedge \texttt{Y}{=}y \wedge \texttt{Q}{=}0 \wedge \texttt{R}{=}x \wedge \texttt{ERR}{=}0 \wedge (y{>}0 \wedge \neg(y = 0)))
\end{array}$$

S is a WHILE-loop; Fig. 5.5 shows the verification conditions generated:

1. $\forall xyqre.\, (y{>}0 \wedge \neg(y = 0)) \Rightarrow (\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q})[x, y, 0, x, 0/\texttt{X}, \texttt{Y}, \texttt{Q}, \texttt{R}, \texttt{ERR}]$
2. $\texttt{asp}\,(\texttt{R := R}{-}\texttt{Y; Q := 1}{+}\texttt{Q})\,((\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q}) \wedge \texttt{Y} < \texttt{R}) \Rightarrow (\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q})$
3. $\texttt{svc}\,(\texttt{R := R}{-}\texttt{Y; Q := 1}{+}\texttt{Q})\,((\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q}) \wedge \texttt{Y} < \texttt{R})$

The first (1) is $(y{>}0 \wedge \neg(y = 0)) \Rightarrow (x = x{+}y{\times}0)$, which is clearly true. The third (3) is also clearly true as $\texttt{svc}\,S\,P = \mathsf{T}$ if S is loop-free. The second (2) requires a symbolic execution:

$$\begin{array}{l}
\texttt{asp}\,(\texttt{R := R}{-}\texttt{Y; Q := 1}{+}\texttt{Q})\,((\texttt{X} = \texttt{R}{+}\texttt{Y} \times \texttt{Q}) \wedge \texttt{Y} < \texttt{R}) = \\
\texttt{asp} \\
(\texttt{R := R}{-}\texttt{Y; Q := 1}{+}\texttt{Q}) \\
(\exists x\, y\, q\, r.\, \texttt{X} = x \wedge \texttt{Y} = y \wedge \texttt{Q} = q \wedge \texttt{R} = r \wedge ((x = r + y{\times}q) \wedge y < r)) = \\
(\exists x\, y\, q\, r.\, \texttt{X} = x \wedge \texttt{Y} = y \wedge \texttt{Q} = 1{+}q \wedge \texttt{R} = r{-}y \wedge ((x = r + y{\times}q) \wedge y < r))
\end{array}$$

Thus to show verification condition 2 above, we must show:

$$\begin{array}{l}
(\exists x\, y\, q\, r.\, \texttt{X} = x \wedge \texttt{Y} = y \wedge \texttt{Q} = 1{+}q \wedge \texttt{R} = r{-}y \wedge ((x = r + y{\times}q) \wedge y < r)) \\
\Rightarrow \\
(\texttt{X} = \texttt{R} + \texttt{Y}{\times}\texttt{Q})
\end{array}$$

i.e.: $(((x = r + y \times q) \wedge y < r)) \Rightarrow (x = (r{-}y) + y \times (1{+}q))$, which is true.
So all three verification conditions, 1, 2 and 3 above, are true.

The application of weakest precondition methods (wvc and awp) to the Div example is completely standard and the details are well known, so we do not give them here. However, the general remarks made in Section 5.6 continue to apply when there are loops. In particular, using forward symbolic execution, one

can separately explore non-looping paths through a program using software model checking methods. Deductive theorem proving needs only be invoked on paths with loops. The general framework based on `svc` and `asp` enables these separate analysis methods to be unified.

5.8 Discussion, Summary and Conclusion

Our goal has been to provide a review of some classical verification methods, especially Hoare logic, from the perspective of mechanical verification. We reviewed how both weakest preconditions and strongest postconditions could be used to convert the problem of verifying Hoare triples to problems in pure logic, suitable for theorem provers. Although 'going backwards' via weakest preconditions appears superficially simpler, we have tried to make a case for going forward using strongest postconditions. The benefits are that computing strongest postconditions can be formulated as symbolic execution, with loops handled via forward verification conditions. This provides a framework that can unify both deductive methods for full proof of correctness with automatic property checking based on symbolic execution.

Although the development in this paper has been based on classical Hoare logic, over the years there have been many advances that add new ideas. Notable examples are VDM [16] that generalises postconditions to relations between the initial and final states (rather than just predicates on the final state) and separation logic [21] that provides tractable methods for handling pointers. Separation logic tools are often based on symbolic execution (though it remains to to be seen whether anything here provides a useful perspective on this).

The contribution of this paper is to explain how old methods (Floyd-Hoare logic) and new ones (software model checking) can be fitted together to provide a spectrum of verification possibilities. There are no new concepts here, but we hope to have provided a clarifying perspective that shows that Hoare's pioneering ideas will be going strong for another 40 years!

References

1. Mantovani, J., Armando, A., Platania, L.: Bounded model checking of software using smt solvers instead of sat solvers. Int. J. Software Tools Technol. Transfer **11**(1), 69–83 February. (2009).
2. Apt, K.R.: Ten years of Hoare's logic: A survey – part I. ACM Trans. Program. Lang. Syst. **3**(4), 431–483 (1981).
3. Clarke E., Kroening, D., Lerda F.: A tool for checking ANSI-c programs. In: TACAS 2004, vol. 2988 of LNCS, pp. 168–176. Springer-verlag (2004).
4. Cook, S.A.: Soundness and completeness of an axiom system for program verification. SIAM J. Comput. **7**(1), 70–90 (1978).

5. Cytron, R., Ferrante, J., Rosen, B.K., Wegman, M.N., Zadeck, F.K.: Efficently computing static single assignment form and the control dependence graph. Trans. Program. Lang. Systems, **13**(4), 451–490 (1991).
6. Dijkstra, E.W.: A Discipline of Programming. Prentice Hall (October 1976).
7. Dijkstra, E.W., Scholten, C.S.: Predicate Calculus and Program Semantics. Texts and Monographs in Computer Science. Springer, New York, Inc., New York, NY, USA (1990).
8. Floyd, R.W.: Assigning meanings to programs. In: Procsedings of the. Sympos. Applied. Mathematics., Vol. XIX, pp. 19–32. Amer. Math. Soc., Providence, R.I., (1967).
9. Gannod, G.C., Cheng, B.H.C.: Strongest postcondition semantics as the formal basis for reverse engineering. In: WCRE '95: Proceedings of the Second Working Conference on Reverse Engineering, p. 188, IEEE Computer Society. Washington, DC, USA (1995).
10. Gries, D.: The Science of Programming. Springer (1981).
11. Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: Abstractions from proofs. In: Jones, N. D., Leroy, Xa. (eds.), POPL, pp. 232–244. ACM (2004).
12. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. the ACM **12**(10), 576–580 October (1969).
13. Hoare, C.A.R., Lauer, P.E.: Consistent and complementary formal theories of the semantics of programming languages. Acta Inf. **3**, 135–153 (1974).
14. Igarashi, S., London, R.L., Luckham, D.C.: Automatic program verification I: A logical basis and its implementation. Acta Inf. **4**, 145–182 (1975).
15. Igarashi, S., London, R.L., Luckham, D.C.: Automatic program verification i: a logical basis and its implementation. Technical report, Stanford University, Stanford, CA, USA (1973).
16. Jones, C.B.: The early search for tractable ways of reasoning about programs. IEEE Ann. Hist. Comput. **25**(2), 26–49, (2003).
17. King, J.C.: A program verifier. In: IFIP Congress (1), pp. 234–249 (1971).
18. King, J.C.: Symbolic execution and program testing. Commun. ACM **19**(7), 385–394 (July 1976).
19. Cornelius King, J.: A program verifier. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA (1970).
20. Luckham, D.C.: A brief account: Implementation and applications of a pascal program verifier (position statement). In: ACM '78: Proceedings of the 1978 Annual conference, pp. 786–792 ACM, New York, NY, USA (1978).
21. O'Hearn, P.W., Reynolds, J.C., Yang, H.: Local reasoning about programs that alter data structures. In: Proceedings of 15th Annual Conference of the European Association for Computer Science Logic, vol. 2142 of Lecture Notes in Computer Science, pp. 1–19. Springer (September 2001).
22. Pasareanu, Co.S., Visser, W.: Verification of java programs using symbolic execution and invariant generation. In: Graf, S. Mounier, L. (eds.), SPIN, vol. 2989 of Lecture Notes in Computer Science, pp. 164–181. Springer (2004).
23. von Henke, F.W, Luckham, D.C.: A methodology for verifying programs. In: Proceedings of the International Conference on Reliable Software, pp. 156–164 ACM, New York, NY, USA (1975).
24. Winskel, G.: The Formal Semantics of Programming Languages: an Introduction. MIT, Cambridge, MA, USA (1993).

Appendix: Proofs Relating `svc`, `asp`, `wvc`, `awp`

Proof by Structural Induction on S that $\forall S\, P.\ \texttt{svc}\, S\, P \Rightarrow \{P\}S\{\texttt{asp}\, S\, P\}$

1. `SKIP`.
   Follows from $\vdash \{P\}\,\texttt{SKIP}\,\{P\}$.

2. $V := E$.
   Follows from $\vdash \{P\}\, V := E \{\exists v.\ (V = E[v/V]) \wedge P[v/V]\}$.

3. $S_1 ; S_2$.
   Assume by induction:
   $\forall P.\ \texttt{svc}\, S_1\, P \Rightarrow \{P\}S_1\{\texttt{asp}\, S_1\, P\}$
   $\forall P.\ \texttt{svc}\, S_2\, P \Rightarrow \{P\}S_2\{\texttt{asp}\, S_2\, P\}$
   Specialising P to $\texttt{asp}\, S_1\, P$ in the inductive assumption for S_2 yields:
   $\texttt{svc}\, S_2\, (\texttt{asp}\, S_1\, P) \Rightarrow \{\texttt{asp}\, S_1\, P\}S_2\{\texttt{asp}\, S_2\, (\texttt{asp}\, S_1\, P)\}$
   Hence by inductive assumption for S_1 and the Hoare sequencing rule:
   $\texttt{svc}\, S_1\, P \wedge \texttt{svc}\, S_2\, (\texttt{asp}\, S_1\, P) \Rightarrow \{P\}S_1 ; S_2\{\texttt{asp}\, S_2\, (\texttt{asp}\, S_1\, P)\}$
   Hence by definitions of $\texttt{svc}\, (S_1 ; S_2)\, P$ and $\texttt{asp}\, (S_1 ; S_2)\, P$:
   $\texttt{svc}\, (S_1 ; S_2)\, P \Rightarrow \{P\}S_1 ; S_2\{\texttt{asp}\, (S_1 ; S_2)\, P\}$.

4. $\texttt{IF}\, B\, \texttt{THEN}\, S_1\, \texttt{ELSE}\, S_2$.
   Assume by induction:
   $\forall P.\ \texttt{svc}\, S_1\, P \Rightarrow \{P\}S_1\{\texttt{asp}\, S_1\, P\}$
   $\forall P.\ \texttt{svc}\, S_2\, P \Rightarrow \{P\}S_2\{\texttt{asp}\, S_2\, P\}$
   Specialising these with $P \wedge B$ and $P \wedge \neg B$, respectively, yields:
   $\texttt{svc}\, S_1\, (P \wedge B) \Rightarrow \{P \wedge B\}S_1\{\texttt{asp}\, S_1\, (P \wedge B)\}$
   $\texttt{svc}\, S_2\, (P \wedge \neg B) \Rightarrow \{P \wedge \neg B\}S_2\{\texttt{asp}\, S_2\, (P \wedge \neg B)\}$
   Applying the 'postcondition weakening' Hoare logic Rule of Consequence:
   $\texttt{svc}\, S_1\, (P \wedge B) \Rightarrow \{P \wedge B\}S_1\{\texttt{asp}\, S_1\, (P \wedge B) \vee \texttt{asp}\, S_2\, (P \wedge \neg B)\}$
   $\texttt{svc}\, S_2\, (P \wedge \neg B) \Rightarrow \{P \wedge \neg B\}S_2\{\texttt{asp}\, S_1\, (P \wedge B) \vee \texttt{asp}\, S_2\, (P \wedge \neg B)\}$
   Hence by definition of $\texttt{asp}\, (\texttt{IF}\, B\, \texttt{THEN}\, S_1\, \texttt{ELSE}\, S_2)\, P$:
   $\texttt{svc}\, S_1\, (P \wedge B) \Rightarrow \{P \wedge B\}S_1\{\texttt{asp}\, (\texttt{IF}\, B\, \texttt{THEN}\, S_1\, \texttt{ELSE}\, S_2)\, P\}$
   $\texttt{svc}\, S_2\, (P \wedge \neg B) \Rightarrow \{P \wedge \neg B\}S_2\{\texttt{asp}\, (\texttt{IF}\, B\, \texttt{THEN}\, S_1\, \texttt{ELSE}\, S_2)\, P\}$
   Since Hoare triples are true if the precondition is unsatisfiable:
   $\texttt{UNSAT}(P \wedge B) \Rightarrow \{P \wedge B\}S_1\{\texttt{asp}\, (\texttt{IF}\, B\, \texttt{THEN}\, S_1\, \texttt{ELSE}\, S_2)\, P\}$
   $\texttt{UNSAT}(P \wedge \neg B) \Rightarrow \{P \wedge \neg B\}S_2\{\texttt{asp}\, (\texttt{IF}\, B\, \texttt{THEN}\, S_1\, \texttt{ELSE}\, S_2)\, P\}$
   By definition of $\texttt{svc}\, (\texttt{IF}\, B\, \texttt{THEN}\, S_1\, \texttt{ELSE}\, S_2)\, P$ and Hoare logic rules:
   $\texttt{svc}\, (\texttt{IF}\, B\, \texttt{THEN}\, S_1\, \texttt{ELSE}\, S_2)\, P$
   $\Rightarrow \{P\}\,\texttt{IF}\, B\, \texttt{THEN}\, S_1\, \texttt{ELSE}\, S_2\{\texttt{asp}\, (\texttt{IF}\, B\, \texttt{THEN}\, S_1\, \texttt{ELSE}\, S_2)\, P\}$

5. $\texttt{WHILE}\, B\, \texttt{DO}\{R\}\, S$.
   Assume by induction:
   $\forall P.\ \texttt{svc}\, S\, P \Rightarrow \{P\}S\{\texttt{asp}\, S\, P\}$

Specialise P to $R \wedge B$:
$\texttt{svc}\, S\, (R \wedge B) \Rightarrow \{R \wedge B\}\, S\, \{\texttt{asp}\, S\, (R \wedge B)\}$
By definition of $\texttt{svc}\, (\texttt{WHILE}\, B\, \texttt{DO}\{R\}\, S)\, P$ and Consequence Rules:
$\texttt{svc}\, (\texttt{WHILE}\, B\, \texttt{DO}\{R\}\, S)\, P \Rightarrow \{R \wedge B\}\, S\, \{R\}$
By definition of $\texttt{svc}\, (\texttt{WHILE}\, B\, \texttt{DO}\{R\}\, S)\, P$ and Hoare WHILE-rule:
$\texttt{svc}\, (\texttt{WHILE}\, B\, \texttt{DO}\{R\}\, S)\, P \Rightarrow \{P\}\, \texttt{WHILE}\, B\, \texttt{DO}\{R\}\, S\, \{R \wedge \neg B\}$
Hence by definition of $\texttt{asp}\, (\texttt{WHILE}\, B\, \texttt{DO}\{R\}\, S)\, P$:
$\texttt{svc}\, (\texttt{WHILE}\, B\, \texttt{DO}\{R\}\, S)\, P \Rightarrow$
$\{P\}\, \texttt{WHILE}\, B\, \texttt{DO}\{R\}\, S\, \{\texttt{asp}\, (\texttt{WHILE}\, B\, \texttt{DO}\{R\}\, S)\, P\}$

Proof by Structural Induction on S that:
$\forall\, S\, Q.\ \texttt{wvc}\, S\, Q \Rightarrow \{\texttt{awp}\, S\, Q\}\, S\, \{Q\}$

1. SKIP.
   Follows from $\vdash \{Q\}\, \texttt{SKIP}\, \{Q\}$.

2. $V := E$.
   Follows from $\vdash \{Q[E/V]\}\, V := E\, \{Q\}$.

3. $S_1\, ;\, S_2$.
   Assume by induction:
   $\forall\, Q.\, \texttt{wvc}\, S_1\, Q \Rightarrow \{\texttt{awp}\, S_1\, Q\}\, S_1\, \{Q\}$
   $\forall\, Q.\, \texttt{wvc}\, S_2\, Q \Rightarrow \{\texttt{awp}\, S_2\, Q\}\, S_2\, \{Q\}$
   Specialising Q to $\texttt{awp}\, S_2\, Q$ in the inductive assumption for S_1 yields:
   $\texttt{wvc}\, S_1\, (\texttt{awp}\, S_2\, Q) \Rightarrow \{\texttt{awp}\, S_1\, (\texttt{awp}\, S_2\, Q)\}\, S_1\, \{\texttt{awp}\, S_2\, Q\}$
   Hence by inductive assumption for S_2 and the Hoare sequencing rule:
   $\texttt{wvc}\, S_1\, (\texttt{awp}\, S_2\, Q) \wedge \texttt{wvc}\, S_2\, Q \Rightarrow \{\texttt{awp}\, S_1\, (\texttt{awp}\, S_2\, Q)\}\, S_1\, ;\, S_2\, \{Q\}$
   Hence by definitions of $\texttt{wvc}\, (S_1\, ;\, S_2)\, Q$ and $\texttt{awp}\, (S_1\, ;\, S_2)\, Q$:
   $\texttt{wvc}\, (S_1\, ;\, S_2)\, Q \Rightarrow \{\texttt{awp}\, (S_1\, ;\, S_2)\, Q\}\, S_1\, ;\, S_2\, \{Q\}$.

4. $\texttt{IF}\, B\, \texttt{THEN}\, S_1\, \texttt{ELSE}\, S_2$.
   Assume by induction:
   $\texttt{wvc}\, S_1\, Q \Rightarrow \{\texttt{awp}\, S_1\, Q\}\, S_1\, \{Q\}$
   $\texttt{wvc}\, S_2\, Q \Rightarrow \{\texttt{awp}\, S_2\, Q\}\, S_2\, \{Q\}$
   Strengthening the preconditions using the Rules of Consequence
   $\texttt{wvc}\, S_1\, Q \Rightarrow \{\texttt{awp}\, S_1\, Q \wedge B\}\, S_1\, \{Q\}$
   $\texttt{wvc}\, S_2\, Q \Rightarrow \{\texttt{awp}\, S_2\, Q \wedge \neg B\}\, S_2\, \{Q\}$
   Rewriting the preconditions using $\texttt{awp}\, (\texttt{IF}\, B\, \texttt{THEN}\, S_1\, \texttt{ELSE}\, S_2)\, Q$
   $\texttt{wvc}\, S_1\, Q \Rightarrow \{\texttt{awp}\, (\texttt{IF}\, B\, \texttt{THEN}\, S_1\, \texttt{ELSE}\, S_2)\, Q \wedge B\}\, S_1\, \{Q\}$
   $\texttt{wvc}\, S_2\, Q \Rightarrow \{\texttt{awp}\, (\texttt{IF}\, B\, \texttt{THEN}\, S_1\, \texttt{ELSE}\, S_2)\, Q \wedge \neg B\}\, S_2\, \{Q\}$
   Since Hoare triples are true if the postcondition is a tautology
   $\texttt{TAUT}(Q) \Rightarrow \{\texttt{awp}\, (\texttt{IF}\, B\, \texttt{THEN}\, S_1\, \texttt{ELSE}\, S_2)\, Q \wedge B\}\, S_1\, \{Q\}$
   $\texttt{TAUT}(Q) \Rightarrow \{\texttt{awp}\, (\texttt{IF}\, B\, \texttt{THEN}\, S_1\, \texttt{ELSE}\, S_2)\, Q \wedge \neg B\}\, S_2\, \{Q\}$

By the definition of $\texttt{wvc}\,(\texttt{IF}\,B\,\texttt{THEN}\,S_1\,\texttt{ELSE}\,S_2)\,Q$ and the Hoare conditional rule
$\texttt{wvc}\,(\texttt{IF}\,B\,\texttt{THEN}\,S_1\,\texttt{ELSE}\,S_2)\,Q$
$\Rightarrow \{\texttt{awp}\,(\texttt{IF}\,B\,\texttt{THEN}\,S_1\,\texttt{ELSE}\,S_2)\,Q\}\,\texttt{IF}\,B\,\texttt{THEN}\,S_1\,\texttt{ELSE}\,S_2\,\{Q\}$

5. $\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S$.
Assume by induction:
$\forall\, Q.\, \texttt{wvc}\,S\,Q \Rightarrow \{\texttt{awp}\,S\,Q\}\,S\,\{Q\}$
Specialise Q to R. By $\texttt{wvc}\,(\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S)\,Q$ and Consequence Rule:
$\texttt{wvc}\,(\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S)\,Q \Rightarrow \{R \wedge B\}\,S\,\{R\}$
By Hoare WHILE-rule:
$\texttt{wvc}\,(\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S)\,Q \Rightarrow \{R\}\,\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S\,\{R \wedge \neg B\}$
By definitions of awp and wvc for $\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S$, and Consequence Rule:
$\texttt{wvc}\,(\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S)\,Q \Rightarrow$
$\{\texttt{awp}\,(\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S)\,Q\}\,\texttt{WHILE}\,B\,\texttt{DO}\{R\}\,S\,\{Q\}$