A GENERALIZATION OF OWICKI-GRIES’S HOARE LOGIC 
FOR A CONCURRENT WHILE LANGUAGE

Colin STIRLING
Department of Computer Science, University of Edinburgh, Edinburgh EH8 9YL, United Kingdom

Abstract. A syntax-directed generalization of Owicki-Gries’s Hoare logic for a parallel while language is presented. The rules are based on Hoare asserted programs of the form \( \{I, A\} p \{B, A\} \) where \( I, A \) are sets of first-order formulas. These triples are interpreted with respect to an operational semantics involving potential computations where \( I, A \) are sets of invariants.

Introduction

Consider the pair of Hoare logic rules:

\[
\begin{align*}
\{A\} \ p \ \{B\} \ & \ \{B\} \ q \ \{C\} \\
\{A\} \ p; q \ \{C\} \\
\text{if} \ \{A \land D\} \ p \ \{B\} \ & \ \{A \land \neg D\} \ q \ \{B\} \\
\{A\} \ & \ \text{if} \ D \ \text{then} \ p \ \text{else} \ q \ \{B\}
\end{align*}
\]

Understanding (and applying) these rules is independent of knowing the program structure of \( p \) and \( q \) in the premises, and similarly for the logical structure of the formulas in the sequential composition rule. But some knowledge of logical structure is needed in the if rule: the two preconditions must be of the form \( A \land D, A \land \neg D \) although no knowledge of the structures of \( A \) and \( D \) is necessary.

Well known is the difficulty in maintaining the virtues of the above rules in the case of a Hoare rule for the parallel combinator \( || \) (assumed, here, to be binary). We would then expect such a proof rule to have the form

\[
\begin{align*}
\{A\} \ p \ \{B\} \ & \ \{C\} \ q \ \{D\} \\
\{f(A, C)\} \ p \ || \ q \ \{g(B, D)\}
\end{align*}
\]

where \( f, g \) are uniform logical operators, possibly dependent on the logical structure of their arguments (which we may insist to be of a definite form). Unfortunately, the following kind of example undermines any interesting version of this rule. Let \( p_n \) for each \( n > 0 \) and \( q \) be the programs

\[
\begin{align*}
p_n &= x := 0 \text{ if } x > 0 \text{ then } x := n \text{ else } x := 0, \\
q &= x := 1; \ x := 0.
\end{align*}
\]

For each \( n, \{tt\} \ p_n \ \{x = 0\} \) and \( \{tt\} \ q \ \{x = 0\} \) hold where \( tt \) is true. If \( f(tt, tt) \) is defined, then it could hardly be other than equivalent to \( tt \). But, for each \( n, \{tt\} \ p_n \ || \ q \ \{x = 0 \lor x = n\} \). Assume that \( x \) ranges over the natural numbers. Then we are forced to conclude that if \( g(x = 0, x = 0) \) is defined, then it also has to be equivalent to \( tt \) (since, for each \( n, g(x = 0, x = 0) \leftrightarrow x = 0 \lor x = n \)).
A Hoare asserted program \(\{A\} p \{B\}\) carries insufficient information in the presence of \(\|\). There are two interwoven deficiencies. On the one hand, the meaning of \(\{A\} p \{B\}\) tells us little about the properties of the intermediate states an execution of \(p\) may pass through when \(A\) is initially true. On the other hand, the usual meanings of \(p\) do not include the possible effects of interference. For instance, the meaning of \(p_n\) above (either as an input/output function or as a family of histories, of sequences of states), does not include the possibility that \(x := n\) may be executed.

Owicki–Gries overcome the first shortcoming by appealing to proof outlines [9, 10]: a Hoare triple should not be viewed as an independent object but as the result of a proof outline which carries information about the intermediate states. Then the second deficiency is dealt with using interference freedom of proof outlines: two outlines are interference-free if the preconditions and final postcondition in either are "invariants" relative to the subproofs of the indivisible subprograms of the other.

Other ways of logically coding the additional information have been proposed. One idea is to explicitly invoke a more expressive first-order language than normally countenanced in Hoare logics for sequential while programs by including history variables ranging over sequences of values of shared variables [11]. This not only increases the amount of detail needed for a specification of a parallel program, but it also depends on a (nonlogical) notion of compatibility of histories for the parallel rule. Alternatively, one can try to abstract from such detail by appealing to various notions of invariant. One version is to reinterpret the Hoare triple \(\{A\} p \{B\}\) making \(A\) an invariant [7]. Another is to include noninterference conditions as invariants in the specification [6]. But there is also the unfortunate need to weaken the role of invariants by appealing to auxiliary variables or spatial predicates ranging over program places. These are by no means the only proposals [4].

Starting from the second deficiency we reformulate and generalize Owicki–Gries's Hoare logic as a system of syntax-directed rules. Like the original system it has the drawback of depending on auxiliary variables. Following [1, 3] we interpret a program in terms of its potential computations, as if it were executed in parallel with other programs. This results in two kinds of actions, according to who performs them; either the environment or the program. Then potential computations can be classified in terms of the invariant properties their environment (program) actions satisfy. Based on this framework we reformulate interference freedom semantically. The resulting Hoare logic is similar to [6]. The Hoare triples have the form \(\{\Gamma, A\} p \{B, A\}\) where \(\Gamma\) is, in effect, a set of rely and \(A\) a set of guarantee conditions. Two such triples can be composed in parallel if the rely conditions of either are guarantee conditions of the other. Part of the generality of the system is due to a logic of invariants, which we develop. The Hoare system is shown to be sound with respect to the semantics and complete relative to Owicki–Gries's system.

In Section 1 we shall introduce potential computations and a logic of invariants. Section 2 reformulates interference freedom and the semantics of the extended Hoare triples \(\{\Gamma, A\} p \{B, A\}\). The proof system together with a sample proof showing that it may be of use for developing programs concludes the paper.
1. Potential computations and invariants

Besides parallel, Owicki-Gries's programming language contains the await construct: await \( B \) then \( p \), where \( B \) is a boolean condition. No interference in its execution is permitted: execution only happens when \( B \) is true and then the whole construct (including the evaluation of \( B \)) is executed as an indivisible action. Let \( \land \) be the empty program with its usual properties: \( \land ; p = p = p ; \land \) and \( \land \parallel p = p \parallel \land \). The abstract syntax of the programming language where \( D \) ranges over appropriate booleans is

\[
p ::= \land \mid x := t \mid p \mid p \mid p ; p \mid \text{if } D \text{ then } p \text{ else } q \mid \text{while } D \text{ do } p \mid \text{await } D \text{ then } p
\parallel p.
\]

For simplicity we assume that \( \parallel \) is a binary operator. No conditions on the atomicity of assignment or on programs appearing within the scope of an await are imposed.

We offer a structured operational semantics for this concurrent language in the style of [5] but extended to "potential" computations. Usually, operational models understand a program in terms of its execution sequences. But those of \( p \parallel q \) are not definable from the sequences of \( p \) and \( q \). Instead, it has been suggested by [1, 3], amongst others, that a program should be thought of as being executed in an arbitrary "environment" (that is, in parallel with other programs). This leads to understanding a program in terms of its potential computations.

Assume \( S \) is the set of states where a state associates a data value with each identifier. Boolean conditions and the first-order formulas of a Hoare logic can be interpreted on states in the usual way: let \( L \) be the first-order language built from terms of the programming language. The notation \( s \models A \) means: the assertion \( A \) is true at state \( s \). Moreover, let \( s[t/x] \) where \( t \) is a data term be the new state which is like \( s \) except that it associates the value of \( t \) in \( s \) to the identifier \( x \). A labelled transition relation is defined between program state pairs:

\[
(p, s) \xrightarrow{I} (q, s')
\]

The relation \( \xrightarrow{I} \) represents a performance of an indivisible action, a change from program \( p \) in state \( s \) to \( q \) in \( s' \). The label \( I \) indicates the performer: when \( I = \text{P} \), it is the program; otherwise, the environment acts and \( I = \text{E} \). These relations are defined as follows.

**Definition 1.1**

(a) \( \xrightarrow{E} : (p, s) \xrightarrow{E} (p, s') \);

(b) \( \xrightarrow{P} \) is the least relation such that

(i) \( (x := t, s) \xrightarrow{P} (\land, s[t/x]) \),

(ii) \( (p; q, s) \xrightarrow{P} (p'; q, s') \) if \( (p, s) \xrightarrow{P} (p', s') \),

(iii) (if \( D \) then \( p \) else \( q, s \) \( \xrightarrow{P} (p, s) \) if \( s \models D \),

(iv) (if \( D \) then \( p \) else \( q, s \) \( \xrightarrow{P} (q, s) \) if \( s \models \neg D \),

(v) (while \( D \) do \( p, s \) \( \xrightarrow{P} (p; \text{while } D \text{ do } p, s) \) if \( s \models D \).
(vi) $\langle \text{while } D \text{ do } p, s \rangle \rightarrow^p \langle \land, s \rangle$ if $s \models \neg D$.

(vii) $\langle \text{await } D \text{ then } p, s \rangle \rightarrow^p \langle \land, s \rangle$ if $s \models D$ and either $p = \land$ or, for some $n \geq 1$ and for all $i, 1 \leq i \leq n$, with $p_0 = p$, $p_n = \land$, $s_0 = s$ and $s_n = s'$: $\langle p_{i-1}, s_{i-1} \rangle \rightarrow^p \langle p_i, s_i \rangle$.

(viii) $\langle p || q, s \rangle \rightarrow^p \langle p', q, s' \rangle$ if $\langle p, s \rangle \rightarrow^p \langle p', s' \rangle$.

(ix) $\langle p || q, s \rangle \rightarrow^p \langle q', s' \rangle$ if $\langle q, s \rangle \rightarrow^p \langle q', s' \rangle$.

An indivisible program action, according to this definition, is an assignment, an await, or an evaluation of a boolean condition. A finer grain of indivisibility as in [9, 7] could be introduced instead. Notice the sole condition for $\rightarrow^E: \langle p, s \rangle \rightarrow^E \langle p', s' \rangle$ represents interference, an updating of the state $s$ to $s'$ by the environment. We now define potential computation.

**Definition 1.2.** A potential computation (PC) from $p_0$ is any finite or infinite sequence of the form: $\langle p_0, s_0 \rangle \rightarrow^0 \langle p_1, s_1 \rangle \rightarrow^1 \cdots$ where, for each defined $i, l \in \{ P, E \}$ and $\langle p_i, s_i \rangle \rightarrow^l \langle p_{i+1}, s_{i+1} \rangle$.

Two example pcs where $p - \text{if } x = 0 \text{ then } p_1 \text{ else } p_2$ and $q - \text{await } x = 2 \text{ then } x := 0$ are

(i) $\langle x := 1 \text{ ; } p, s \rangle \rightarrow^p \langle p, s[1/x] \rangle \rightarrow^E \langle p, s[2/x] \rangle \rightarrow^E \langle p, s[0/x] \rangle \rightarrow^p \langle p_1, s[0/x] \rangle$.

(ii) $\langle q, s \rangle \rightarrow^E \langle q, s[1/x] \rangle \rightarrow^E \langle q, s[2/x] \rangle \rightarrow^p \langle \land, s[0/x] \rangle \rightarrow^E \langle \land, s[0/x] \rangle$.

Example (i) illustrates how interference may affect computational behaviour. When there are only sequential contexts, $x := 1 \text{ ; } p$ can be deemed equivalent to $x := 1$; if $x = 0$ then $p_3 \text{ else } p_2$. For any $p_2$, $p_1$ and $p_3$ cannot contribute to the behaviour of these programs in any such context. But not so in the presence of concurrency: the environment may make the boolean condition $x = 0$ true by changing the value of $x$ as in example (i). In example (ii) the program $q$ is blocked in the state $s[1/x]$ and can only proceed if the environment unblocks it (by making $x = 2$ true). Both (i) and (ii) involve the same sequences of states and at no point do they both perform a program action. So they can be conjoined (in the sense of Aczel) to yield the following pc from $(x := 1 \text{ ; } p) || q$:

(iii) $\langle x := 1 \text{ ; } p || q, s \rangle \rightarrow^p \langle p || q, s[1/x] \rangle \rightarrow^E \langle p || q, s[2/x] \rangle$.

Further conjoinings with (iii) result in pcs from programs with additional parallel components. Conversely, any such pc can be divided into component pcs like (i) and (ii): each represents a viewpoint of the overall behaviour. The value, therefore, of semantics based on pcs is that $\parallel$ can be understood compositionally: pcs from a concurrent program are definable in a fixed fashion from those of its components.

Some notation we shall use in the sequel. Where $\sigma$ is the pc

$$\sigma = \langle p_0, s_0 \rangle \xrightarrow{l_0} \langle p_1, s_1 \rangle \xrightarrow{l_1} \cdots,$$
then for defined $i$, $\sigma(i)$ is the $i$th state $s_i$; $\sigma(\lambda i)$ is the $i$th label $l_i$; and $\sigma(\pi i)$ is the $i$th program $p_i$. We let $[p]$ be the set of all pcs from $p$. (Using a small number of (somewhat messy) sequence operations, the sets $[p]$ can be defined denotationally). Some of the details can be gleaned from the later proofs of Lemma 2.4, and Theorem 3.1. Finally, an important subset of $[p]$ is $\mathcal{A}[p]$ the set of actual computations from $p$: $\sigma$ is actual if it contains no $E$ labels—so the environment does not contribute to the behaviour of $\sigma$.

We would like a logical understanding, which we can appeal to when developing a Hoare logic, of when pcs represent viewpoints of an overall behaviour. (A much richer understanding, using temporal logic, can be found in [3].) To this end we introduce a logic of invariants. Actions or events are encapsulated in the operational semantics in state changes: a state change is a pair of states $(s, s')$ both in $S$ where $s$ is the state before and $s'$ the state after the action. Each first-order $A \in L$ determines a set of changes which are invariant with respect to it. We let $I(A)$ be this set; $I$ is also extended to families $\Gamma$ of formulas in $L$.

**Definition 1.3.**

(i) $I(A) = \{(s, s') | s \models A, \text{then } s' \models A\}.$

(ii) $I(\Gamma) = \bigcap_{A \in \Gamma} I(A).$

The idea is that if $(s, s') \notin I(A)$, then this state change "interferes" with the truth of $A$ at $s$. Each $I(\Gamma)$ is a subset of $S \times S$. The least subset is $\{(s, s)\}$: clearly, a null change cannot interfere with the truth of any formula. One representation of this set is $I(\emptyset)$. The largest subset is $S \times S$: representatives of this set include $I(\emptyset)$, $I(\text{tt})$ and $I(\text{ff})$ where tt is true and ff false.

We let $\Rightarrow$ be an invariant implication between sets of formulas.

**Definition 1.4.** $\Gamma \Rightarrow \Delta$ iff $I(\Gamma) \subseteq I(\Delta)$

The following lemma provides a characterization of $\Rightarrow$. Some notation: if $\Gamma$ is a set of formulas, then we let $\neg \Gamma = \{-A | A \in \Gamma\}$ with the convention that $\neg \emptyset = \{\text{ff}\}$. Moreover, we assume that

$\Gamma \models B$ iff $\forall s \in S. \text{if } \forall A \in \Gamma, s \models A, \text{then } s \models B.$

**Lemma 1.5.** $\Gamma \Rightarrow \Delta$ iff $\forall B \in A \forall A' \subseteq \Gamma. (\Gamma' \models B \text{ or } \neg (\Gamma - \Gamma') \models \neg B).$

**Proof.** Suppose $\Gamma \Rightarrow \Delta$. However, for some $B \in \Delta$, $\Gamma' \subseteq \Gamma$, not $(\Gamma' \models B)$ and not $(\neg (\Gamma - \Gamma') \models \neg B)$. Then there is an $s$ such that $\forall A \in \Gamma', s \models A$ and $s \not\models B$, and an $s'$ such that $\forall A \in \neg (\Gamma - \Gamma'), s' \not\models A$ and $s' \models \neg B$. Hence, $(s', s) \notin I(\Delta)$. But consider any $A \in \Gamma$. If $A \in \Gamma'$, then $(s', s) \in I(A)$. If $A \notin \Gamma'$, then also $(s', s) \in I(A)$ since $s' \not\models A$.

Suppose $\forall B \in \Delta \forall A' \subseteq \Gamma. (\Gamma' \models B \text{ or } \neg (\Gamma - \Gamma') \models \neg B)$. However, $(s, s') \in I(\Gamma)$ and $(s, s') \notin I(\Delta)$. Then there is a $B \in \Delta$ such that $s \models B$ and $s \not\models B$. Let $\Gamma_0 = \{C | C \in \Gamma$ and $s \models C\}$. Then either $\Gamma_0 \models B$ or $\neg (\Gamma - \Gamma_0) \models \neg B$. Suppose $\Gamma_0 \models B$. Since $s \not\models B$, there is an $A \in \Gamma_0$ such that $s' \not\models A$. But then $(s, s') \notin I(A)$, contradicting $(s, s') \in I(\Gamma)$. 

Otherwise, $\neg(\Gamma - \Gamma_0) \models \neg B$. Then, as $s \models B$, there is an $A \in \Gamma - \Gamma_0$ such that $s \models A$. But this contradicts the definition of $\Gamma_0$. □

Some useful consequences of this lemma are the following facts where we write $\Sigma, D$ for $\Sigma \cup \{D\}$.

**Fact 1.6.** (i) $\Rightarrow$ is reflexive and transitive.

(ii) If $\Delta \subseteq \Gamma$, then $\Gamma \Rightarrow \Delta$.

(iii) If $\models A \iff B$ and $\Gamma \Rightarrow \Delta$, then $\Gamma, A \Rightarrow \Delta, B$.

(iv) $\Gamma, A, B \Rightarrow \Gamma, A \land B$.

This logic of invariants will play a central role in the Hoare logic of Section 3. A pc $\sigma$ involves a sequence of state changes represented by adjacent pairs of states, $(\sigma(i), \sigma(i+1))$. Hence, pcs may be classified by sets of invariants. However, a pc involves two sorts of state change; one is the result of the environment, the other the result of a program action. Consequently, a more discriminating classification of pcs is in terms of the invariance properties of their environment and their program state changes.

**Definition 1.7.** (i) $E[\Gamma] = \{\sigma | \exists p. \sigma \in [p] \text{ and } \forall i. \text{ if } \sigma(\lambda i) = E, \text{ then } (\sigma(i), \sigma(i+1)) \in I(\Gamma)\}$.

(ii) $P[\Gamma] = \{\sigma | \exists p. \sigma \in [p] \text{ and } \forall i. \text{ if } \sigma(\lambda i) = P, \text{ then } (\sigma(i), \sigma(i+1)) \in I(\Gamma)\}$.

So $E[\Gamma]$ is the set of pcs with the property that all of its environment changes are $\Gamma$ invariant. $P[\Gamma]$ is similar except for program changes.

2. Semantics for an extended Hoare triple

Using pcs and invariants we develop, in this section, semantics of an extended Hoare logic for Owicki-Gries's parallel programming language. First consider the meaning of the usual Hoare asserted program $\{A\} p \{B\}$ where $A, B \in L$, the first-order language interpreted over the state set $S$. Relative to this interpretation $\{A\} p \{B\}$ is true (which we write as $\models \{A\} p \{B\}$) if whenever $A$ is true of an initial state of an actual computation from $p$, then if the computation terminates, $B$ is then true. We recast this in terms of potential and actual computations. Two auxiliary notions are expressed in the following definition.

**Definition 2.1.** For $A \in L$,

(i) $O[A] = \{\sigma | \exists p. \sigma \in [p] \text{ and } \sigma(o) \models A\}$;

(ii) $A[A] = \{\sigma | \exists p. \sigma \in [p] \text{ and } \forall j. \text{ if } \sigma(\pi j) = \wedge, \text{ then } \exists i < j. \sigma(\pi i) = \wedge \text{ and } \sigma(i) \models A\}$. 
Generalization of Owicki–Gries's Hoare logic

The pc $\sigma$ is an element of $O[A]$ when $A$ holds of its initial state. Count $\sigma$ from $p$ as terminating if, for some $i$, $\sigma(\pi i) = \land$, in which case $p$ has exhausted its potential computational behaviour in $\sigma$. So $\sigma \in \Lambda[A]$ if it fails to terminate or if $A$ is true of the initial terminating state, the first $s_i$ such that $\sigma(\pi i) = \land$. Therefore, in terms of computations (recall that $A[p]$ is the set of actual computations from $p$) we have the following definition.

**Definition 2.2.** $\models \{A\} p \{B\}$ iff $O[A] \cap A[p] \subseteq \Lambda[B]$.

The customary difficulties with Definition 2.2 for building Hoare logics for parallel programs are twofold. First, there is the presence of $A[p]$. Second, there is the abstraction from properties of the intermediate states of computations. At the level of proof rules, Owicki–Gries's solution was to observe that a Hoare triple should not just be viewed as an independent object but as the conclusion of a proof outline, containing intermediate assertions. This extra information is about the intermediate states. Two proof outlines can be composed in parallel if they do not interfere with each other: the assertions of one including the intermediaries are invariants with respect to the subproofs of the indivisible subprograms of the other [9]. Consider using pcs to interpret the programs in two such proof outlines. Then interference freedom amounts to saying that the environment invariants of one are program invariants of the other. We are now going to show this.

Suppose we replace $A[p]$ with $[p]$ in Definition 2.2. The result, as it stands, is not very satisfactory because of the unlimited interference to $p$'s behaviour allowed from the environment. A means of controlling the interference is needed. A natural suggestion is to include environment assumptions in the Hoare notation, and in particular a set of environment invariants. As a first step, we let $\{I, A\} p \{B\}$ have the meaning

$$E[I] \cap O[A] \cap [p] \subseteq \Lambda[B]$$

The set $I$ restricts the changes allowed from the environment. To compose this Hoare triple in parallel with another, say $\{\Sigma, C\} \ q \ \{D\}$, according to Owicki–Gries we must know two things:

(i) that $I(\Sigma)$ “contains” the assertions used in the proof of $\{A\} p \{B\}$ ($\{C\} q \{D\}$);

(ii) that each member of $I(\Sigma)$ is invariant with respect to the subproofs of the indivisible subprograms of $q(p)$.

In fact, (i) is too detailed: all we need to know is that $\Gamma \Rightarrow B$ ($\Sigma \Rightarrow D$). And (ii) just means that $I(\Sigma)$ is a set of program invariants over the pcs in $E[\Sigma] \cap O[C] \cap [q]$ ($E[I] \cap O[A] \cap [p]$). This suggests that we must include program invariants in the specification. So we introduce the extended Hoare triple $\{I', A\} p \{B, \Delta\}$ whose meaning is given by the following definition.
Definition 2.3. \( \models \{ \Gamma, A \} p \{ B, \Delta \} \) iff \( E[\Gamma] \cap O[A] \cap \parallel p \parallel \subseteq \Lambda[B] \cap P[\Delta] \).

The analysis of when two extended Hoare triples can be composed in parallel is justified by the next lemma.

Lemma 2.4. If

(i) \( \Gamma \Rightarrow B \) and \( \models \{ \Gamma, A \} p \{ B, \Sigma \cup \Delta \} \), and

(ii) \( \Sigma \Rightarrow D \) and \( \models \{ \Sigma, C \} q \{ D, \Gamma \cup \Delta \} \),

then \( \models \{ \Gamma \cup \Sigma, A \wedge C \} p \parallel q \{ B \wedge D, \Delta \} \).

Proof. Suppose \( \sigma \in E[\Gamma \cup \Sigma] \cap O[A \wedge C] \cap \parallel p \parallel \parallel q \parallel \). Then

\[
\sigma = \langle p_0\|q_0, s_0 \rangle \xrightarrow{l_0} \langle p_1\|q_1, s_1 \rangle \xrightarrow{l_1} \cdots .
\]

So \( \sigma \) can be decomposed into \( \sigma_1 \) from \( p_0 \) and \( \sigma_2 \) from \( q_0 \) with the same sequences of states. First we show that \( \sigma_1 \in E[\Gamma] \) and \( \sigma_2 \in E[\Sigma] \). Suppose not. Then let \( \sigma_1', \sigma_2' \) be the maximum prefixes of \( \sigma_1 \) and \( \sigma_2 \) which are in \( E[\Gamma] \) and \( E[\Sigma] \). Suppose the lengths of \( \sigma_1', \sigma_2' \) are the same. Then

\[
\sigma_1' = \langle p_0, s_0 \rangle \xrightarrow{k_0} \cdots \xrightarrow{k_n} \langle p_{n+1}, s_{n+1} \rangle
\]

and

\[
\sigma_2' = \langle q_0, s_0 \rangle \xrightarrow{k_0} \cdots \xrightarrow{k_n} \langle q_{n+1}, s_{n+1} \rangle.
\]

Clearly, then both \( \sigma_1(\lambda n + 1) = \sigma_2(\lambda n + 1) = E \). Hence, \( \sigma(\lambda n + 1) = E \). But as \( \sigma_1' \) is a maximal prefix of \( \sigma_1 \) in \( E[\Gamma] \), we have \((s_{n+1}, s_{n+2}) \notin I(\Gamma)\) which contradicts \( \sigma \in E[\Gamma \cup \Sigma] \). Otherwise, without loss of generality, suppose the length of \( \sigma_1' \) is greater than the length of \( \sigma_2' \), and suppose \( \sigma_2' \) is as above. Then \( k_{n+1}' = E \). Clearly, \( \sigma_1'(\lambda n + 1) = P \) for otherwise \( \sigma(\lambda n + 1) = E \) and so \( \sigma \notin E[\Gamma \cup \Sigma] \). But \( \sigma_1' \in E[\Gamma] \cap O[A] \cap \parallel p \parallel \) and so \( \sigma_1' \in P[\Sigma \cup \Delta] \). But then \((s_{n+1}, s_{n+2}) \notin I(\Sigma) \). And so \( \sigma_2' \) is not the maximal prefix of \( \sigma_2 \) in \( E[\Sigma] \). Hence, \( \sigma_1 \in E[\Gamma] \) and \( \sigma_2 \in E[\Sigma] \).

Now we show \( \sigma \in \Lambda[B \wedge D] \cap P[\Delta] \). Suppose \( \sigma \) terminates and its first terminating state is \( \sigma(i) \). Then both \( \sigma_1 \) and \( \sigma_2 \) terminate. Suppose \( \sigma_1 \) terminates before \( \sigma_2 \). Then let \( \sigma_1 \)'s first terminating state be \( \sigma_1(j) \). Clearly, then \( \sigma_2 \)'s is \( \sigma_2(i) \), and \( j < i \). By (i), \( \sigma_1(j) \models B \) and all labels beyond \( j \) in \( \sigma_1 \) are \( E \) labels. Hence, as \( \Gamma \Rightarrow B \) and \( \sigma_1 \in E[\Gamma] \) for all defined \( j' > j \), we have \( \sigma_1(j') \models B \). Hence, \( \sigma_1(j) \models B \). By (ii), \( \sigma_2(i) \models D \). So \( \sigma(i) \models B \wedge D \), that is, \( \sigma \in \Lambda[B \wedge D] \). The proof is reciprocal if \( \sigma_2 \) terminates before \( \sigma_1 \). Finally, clearly \( \sigma \in P[\Delta] \) as \( \sigma_1 \in P[\Sigma \cup \Delta] \) and \( \sigma_2 \in P[\Gamma \cup \Delta] \).

Lemma 2.4 rationalizes Owicki–Gries's discussion of parallel composition. First, note that \( p \) and \( q \) may themselves be parallel programs: so \( \parallel \parallel \) can be treated as a binary operator like sequential composition. Next, interference freedom can be understood semantically, independent of proof outlines and assertions appearing therein. At the proof level it means we can offer a syntax-directed Hoare rule for parallel. There are further benefits of making explicit features that are implicit in Owicki-Gries. Once \( \{ \Gamma, A \} p \{ B, \Delta \} \) is known, it can be used again in other proofs.
of parallel programs: the system in [9] lacks this explicit portability. Moreover, the extended Hoare triple fits in with the proposal [6] to explicitly include noninterference in program specifications. In fact, the extended triple is very similar to the proposal in [6]: \( \Gamma \) is just a set of rely conditions while \( \Delta \) consists of guarantee conditions.

A special case of \( \{ \Gamma, A \} p \{ B, \Delta \} \) is when \( \Gamma = L \) and \( \Delta = \emptyset \). For then it is assumed that the environment does not interfere with \( p \), the only environment actions allowed do not involve state change; and nothing needs to be known about the invariance of the program actions. The meaning of this special case coincides with the usual Hoare triple’s meaning as given by Definition 2.2.

**Fact 2.5.** \( \models \{ L, A \} p \{ B, \emptyset \} \text{ iff } \models \{ A \} p \{ B \} \).

Hence, the Hoare triple introduced in Definition 2.3 is an extension of the standard triple.

### 3. The extended Hoare logic

The generalization of Owicki-Gries’s Hoare logic below employs the triples \( \{ \Gamma, A \} p \{ B, \Delta \} \) whose intended meaning was given in the previous section. We assume on the basis of Fact 2.5 that \( \{ A \} p \{ B \} \) abbreviates \( \{ L, A \} p \{ B, \emptyset \} \). Let \( X \) be a set of variables which appear in \( p \) only in assignments \( x := t \) with \( x \in X \); so, as in [9], \( X \) is a set of auxiliary variables for \( p \). Let \( p \setminus X \) be the program obtained by replacing all assignments of \( p \) to variables in \( X \) with \( A \) and transforming any \( \text{await } t \) then \( q \) into \( q \) when \( q = \wedge \) or \( q = y := t \). The rules are as follows.

\[
\begin{align*}
\wedge & & \{ \Gamma, A \} \wedge \{ A, \Delta \} \\
\text{if} & & \begin{array}{c}
\Gamma \Rightarrow A \\
A \equiv C \land \forall B \in \Delta, A \land B \models B^t
\end{array} \quad \{ \Gamma, A \} x := t \{ C, \Delta \} \\
\text{while} & & \begin{array}{c}
\Gamma \Rightarrow A \\
\{ \Gamma, A \land D \} p \{ A, \Delta \}
\end{array} \quad \text{while } D \text{ do } p \{ A \land \neg D, \Delta \}
\end{align*}
\]

\[
\begin{align*}
\text{await} & & \begin{array}{c}
\Gamma \Rightarrow A \\
\{ A \land D \} p \{ C \}
\end{array} \quad \forall B \in \Delta, \{ A \land D \land B \} p \{ B \}
\end{align*}
\]

\[
\begin{align*}
\text{\text{\|}} & & \begin{array}{c}
\Gamma \Rightarrow C \\
\{ \Gamma, A \} p \{ C, \Sigma \lor \Delta \}
\end{array} \quad \Sigma \Rightarrow E \quad \{ \Sigma, B \} q \{ E, \Gamma \lor \Delta \}
\end{align*}
\]

\[
\begin{align*}
\text{AV} & & \begin{array}{c}
\{ A \} p \{ B \}
\end{array} \quad A, B \text{ do not contain free occurrences of } x \in X
\end{align*}
\]
Valid as special cases are the usual Hoare rules for while programs. They are obtained by setting \( r = L \) and \( A = 0 \): for instance, since \( L \equiv B \) for any \( B \in L \), \( A = A \) and \( \Delta = \emptyset \), we can conclude \( \{ A \} x := t \{ A \} \) from the assignment rule. Notice that, as required, each formula in \( \Delta \) is invariant with respect to the subproofs of indivisible subprograms of a program \( p \): so \( \Delta \) is set in the assignment and await rules. The parallel rule depends on interlocking of invariants: the environment invariants of the one are program invariants of the other proof. This guarantees, as we noted in the previous section, a (generalization of) interference freedom in the sense of Owicki-Gries. A major disadvantage of both the original Owicki-Gries system and its reformulation here is the need for the auxiliary variable rule. For example, it is essential to the proof of \( \{ x = 0 \} x := x + 1 \Rightarrow \{ x = 2 \} \). The consequence rule appeals to the two consequence relations \( \Rightarrow \) and \( \Rightarrow \) defined in Section 1.

The following theorem shows that the system is sound with respect to the interpretation of the previous section. Let \( \vdash \{ \Gamma, A \} p \{ B, A \} \) mean that \( \{ \Gamma, A \} p \{ B, A \} \) is provable in the system above.

**Theorem 3.1.** If \( \vdash \{ \Gamma, A \} p \{ B, A \} \), then \( \models \{ \Gamma, A \} p \{ B, A \} \).

**Proof.** By induction on the structure of \( p \). We give two sample cases. The central case, \( p = q \rightarrow r \) is covered by Lemma 2.4. The base cases are \( p = \Lambda \) and \( p = x := t \).

**p = x := t.** Suppose \( \Gamma \Rightarrow A \), \( A = B \), and, for all \( C \in \Delta \), \( A \wedge C = C \). Suppose \( \sigma \in E[\Gamma] \cap O[A] \cap \{ p \} \). If \( \sigma \) terminates, then

\[
\sigma = \langle p, s_0 \rangle \rightarrow \cdots \rightarrow \langle p, s_i \rangle \rightarrow \langle \wedge, s_i[t/x] \rangle \rightarrow \cdots .
\]

As \( \Gamma \Rightarrow A \) and \( \sigma \in E[\Gamma] \cap O[A] \), we have \( s_i \models A \). But also \( s_i \models B \) since \( A \models B \). Hence, \( s_i[t/x] \models B \), and \( \sigma \in A \{ B \} \). Moreover, for \( C \in \Delta \), if \( s_i \models A \wedge C \), then \( s_i \models C \) and \( s_i[t/x] \models C \). Thus \( \sigma \in P[\Delta] \). If \( \sigma \) does not terminate, then it only contains \( E \) labels and so \( \sigma \in A \{ B \} \cap P[\Delta] \).

**p = q \rightarrow r.** Suppose \( \{ \Gamma, A \} q \{ B, A \} \) and \( \{ \Gamma, B \} q \{ C, A \} \). Consider \( \sigma \in E[\Gamma] \cap O[A] \cap \{ q ; r \} \). If \( \sigma \) terminates, then

\[
\sigma = \langle q, r, s_0 \rangle \rightarrow \cdots \rightarrow \langle q, r, s_i \rangle \rightarrow \langle r, s_{i+1} \rangle \rightarrow \cdots
\]

Clearly, we can extract the pcs \( \sigma_1, \sigma_2 \) with \( \sigma_1 \) from \( q \) and \( \sigma_2 \) from \( r \):

\[
\sigma_1 = \langle q, s_0 \rangle \rightarrow \cdots \rightarrow \langle q, s_i \rangle \rightarrow \langle \wedge, s_{i+1} \rangle ,
\]

\[
\sigma_2 = \langle r, s_{i+1} \rangle \rightarrow \cdots \rightarrow \langle r, s_{n+1} \rangle \rightarrow \langle \wedge, s_{n+2} \rangle .
\]
Then $\sigma_1 \in A[B] \cap P[\Delta]$ since $\sigma_1 \in E[\Gamma] \cap O[A] \cap [q]$. And $\sigma_2 \in A[C] \cap P[\Delta]$ since $\sigma_2 \in E[\Gamma] \cap O[B] \cap [r]$. So $\sigma \in A[D] \cap P[\Delta]$. The case for nonterminating $\sigma$ is similar. \qed

The system is complete relative to Owicki-Gries's system (which is Cook complete [2, 8]). Forgetting the different styles of systems, Owicki-Gries's system is (barring one or two inessentials) the subsystem where $T \Rightarrow A$ is understood as $A \in \Gamma$. Let $\vdash \{A\} p \{B\}$ mean $\{A\} p \{B\}$ is provable in Owicki-Gries's system.

**Theorem 3.2.** If $\vdash \{A\} p \{B\}$, then $\vdash \{A\} p \{B\}$.

**Proof.** The result follows almost trivially from Owicki's preassertion function, $\text{pre}$, (cf. [8]; see also [2]) and the characterization of interference freedom in Section 2. For from $\text{pre}$ we get

\[
\vdash \{A\} p \{B\} \text{ implies } \vdash \{\text{pre}(p) \cup \Gamma, A\} p \{B, \emptyset\},
\]

and interference freedom gives the statement: If $\vdash \{A\} p \{B\}$ and $\vdash \{C\} q \{D\}$ are interference-free, then

\[
\vdash \{\text{pre}(p) \cup \{B\}, A\} p \{B, \text{pre}(q) \cup \{D\}\} \text{ and }
\vdash \{\text{pre}(q) \cup \{D\}, C\} q \{D, \text{pre}(p) \cup \{B\}\}. \qed
\]

A pertinent criticism of Owicki-Gries's system is its unsuitability for program development. In contrast, the generalization offered here does allow for development in the spirit of [6]. We illustrate this with an example, a generalization of Owicki-Gries's array searching algorithm (cf. [9]) with which it should be compared. The two features that aid development are the inclusion of noninterference conditions in the specification and the logic of invariants.

**Example 3.3.** Let $a[i]$, $1 \leq i \leq m$ be an array. Find the least member, if there is one, satisfying the predicate $P$ (where $P$ does not mention $a$). Let this member be $x$, otherwise, if there is not one, let $x = m + 1$. So we want a program FINDP such that $\{\text{tt}\}$ FINDP $\{A\}$, where

\[
A \equiv 1 \leq x \leq m + 1 \land \forall i. 1 \leq i \leq x, \neg P(a[i]) \land x \leq m \rightarrow P(a[x]).
\]

We expect FINDP to be of the form: INIT; SEARCH where INIT initializes and SEARCH searches. In SEARCH we use a number of concurrent programs, say $r \approx m$: for simplicity let $r$ divide $m$. Call each one SEARCH$(j)$, $1 \leq j \leq r$. Each SEARCH$(j)$ scans the array squares $j, n + j, 2n + j, \ldots, m + j$ looking for $x$. So assume
each $\text{SEARCH}(j)$ has a “private” variable $x_j$ for searching. We want termination of $\text{SEARCH}(j)$ to occur when

(i) $P(a[x_j])$ or
(ii) $x_j > m$ or
(iii) $\text{SEARCH}(k)$ has found that $P(a[x_k])$ for $x_k < x_j$.

So we introduce a further “private” variable $y_j$ for each $\text{SEARCH}(j)$ which initially is set to $m + j$ and if $P(a[x_j])$ holds, then $y_j$ is reduced to $x_j$. Hence, the termination condition for $\text{SEARCH}(j)$ is $x_j \geq \min\{y_1, \ldots, y_r\}$. Finally, $x$ will be the value of $\min\{y_1, \ldots, y_r\}$ when each $\text{SEARCH}(j)$ terminates.

The idea of the formalism $\{\Gamma, A\} p \{B, A\}$ is that we develop $p$ subject to the assumptions $\Gamma$ about the environment and assumptions $A$ we make about $p$. Consider $\text{SEARCH}(j)$; then,

(i) as $x_j, y_j$ are “private”, the environment cannot affect their values or increase $y_k, k \neq j$
(ii) the program $\text{SEARCH}(j)$ cannot affect the variables $x_n$ and $y_n$ (for $n \neq j$) and it does not increase the initial value of $y_j$.

Thus the environment and program assumptions for $\text{SEARCH}(j)$ are given by the sets:

\[
\Gamma_j = \bigcup_{n \neq j} \{y_n = i | 1 \leq i \leq m + r\} \cup \{x_j = i | 1 \leq i \leq m + r\}
\]

\[
\Delta_j = \bigcup_{n \neq j} \{y_n = i | 1 \leq i \leq m + r\} \cup \bigcup_{n \neq j} \{x_n = i | 1 \leq i \leq m + r\}
\]

\[
\cup \{y_j \leq i | 1 \leq i \leq m + r\}.
\]

$\text{SEARCH}(j)$ will be a loop with invariant $C_j$ (compare $A$):

- $A_j = x_j \mod r = j \land \forall k. 1 \leq k \leq x_j \land k \mod r = j \rightarrow \neg P(a[k])$;
- $B_j = y_j \leq m \rightarrow P(a[y_j]) \land y_j \leq m + j$;
- $C_j = A_j \land B_j$.

So $\text{SEARCH}(j)$ is

while $x_j < \min\{y_1, \ldots, y_r\}$ do
if $P(a[x_j])$ then $y_j := x_j$
else $x_j := x_j + r$

Then $\{\Gamma_j, C_j\} \text{SEARCH}(j) \{C_j \land x_j \geq \min\{y_1, \ldots, y_r\}, \Delta_j\}$ follows from

(a) $\{\Gamma_j, x_j < y_j \land C_j \land P(a[x_j])\} y_j := x_j \{C_j, \Delta_j\}$,
(b) $\{\Gamma_j, x_j < y_j \land C_j \land \neg P(a[x_j])\} x_j := x_j + r \{C_j, \Delta_j\}$.

And (a) depends on

$\Gamma_j \Rightarrow x_j < y_j \land C_j \land P(a[x_j])$;

(b) depends on

$\Gamma_j \Rightarrow x_j < y_j \land C_j \land \neg P(a[x_j])$. 
Application of the if rule depends on \( F_j \Rightarrow x_j < y_j \land C_j \) and, finally, applying the while rule depends on \( F \Rightarrow C_j \).

SEARCH is the program \( \text{SEARCH}(1) \parallel \cdots \parallel \text{SEARCH}(r) \). By repeated application of the parallel rule (where \( \land = \land_{1 \leq i \leq r} \)) we find

\[
\{ \bigcup F_i, \land C_i \} \text{SEARCH} \{ \land C_i \land x_i \geq \min \{ y_1, \ldots, y_r \}, \emptyset \}.
\]

So, by the Con rule, as \( L \Rightarrow \bigcup F_i \),

\[
\{ \land C_i \} \text{SEARCH} [ \land C_i \land x_i \geq \min \{ y_1, \ldots, y_r \}].
\]

Then \( x \) is set to \( \min \{ y_1, \ldots, y_r \} \). Note that

\[
\land C_i \land x_i \geq \min \{ y_1, \ldots, y_r \} \models A_x \min \{ y_1, \ldots, y_r \}.
\]

And INIT is \( x_j := j ; y_j := m + j \) for \( 1 \leq i \leq r \). So, finally,

\[
\{ \tau \} \text{INIT} ; \text{SEARCH} ; x := \min \{ y_1, \ldots, y_r \} \{ A \}.
\]

Acknowledgment

I would like to thank Roland Backhouse for suggesting various improvements of an earlier version, and Dorothy McKie for typing the manuscript.

References