# Control Predicates Are Better Than Dummy Variables For Reasoning About Program Control

LESLIE LAMPORT
Digital Equipment Corporation

When explicit control predicates rather than dummy variables are used, the Owicki-Gries method for proving safety properties of concurrent programs can be strengthened, making it easier to construct the required program annotations.

Categories and Subject Descriptors: D2.4 [**Software Enginering**]: Program Verification—*correctness proofs*; F3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs

General terms: Verification
Additional Key Words and Phrases: concurrent programming, invariance, safety properties, non-interference, Owicki-Gries method

## 1. INTRODUCTION

The Owicki-Gries method, an extension to concurrent programs of Floyd's method [3] for proving partial correctness of sequential programs, was developed independently by Owicki and Gries [11] and by us [8]. These two presentations of the method differed in two ways. First, Owicki and Gries used a conventional structured programming language while we used a flowchart language. This was a purely syntactic difference. The second, more significant difference, involved how control information is represented.

In the Owicki-Gries method, as in Floyd's method, a program is annotated by attaching assertions to control points. The major part of the proof involves showing the invariance of the annotation [7]. In Floyd's method, the assertions mention only the program's variables. However, for concurrent programs, the assertions attached to one process must also refer to the control state of other processes—that is, they must be functions of the values of other processes' "program counters". The presentations in [11] and [8] differed in how dependence on the control state was expressed. In [11], Owicki and Gries avoided explicit mention of the control state by using dummy variables[1]—variables introduced only for the proof—to encode control information. In [8], we used control predicates—assertions that explicitly

[1]They have also been called "auxiliary variables", "ghost variables", and "thought variables".

Author's address: Systems Research Center, Digital Equipment Corporation, 130 Lytton Ave., Palo Alto, CA 94301.
Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1987 ACM 0164-0925/87/0400-0001 $00.75

mention the control state.

Since control predicates can be simulated by dummy variables, it appears that choosing between the two approaches is purely a matter of taste. We have preferred to use control predicates both for aesthetic reasons and because they are necessary for certain extensions of the method [10]. However, when applying the standard Owicki-Gries method, there seems to be no basic difference between the two approaches.

In this paper, we show that there is a real difference between control predicates and dummy variables. Although dummy variables can represent the control state, the implicit nature of this representation limits their utility. The use of explicit control predicates allows a strengthening of the ordinary Owicki-Gries method that makes it easier to write annotations.

Our strengthening of the Owicki-Gries method eliminates a well-known weakness in the original method. Assertional methods for proving safety properties involve proving the invariance of an assertion. In the Ashcroft method [1], one writes a single global assertion; in the Owicki-Gries method, the global assertion is decomposed into an annotation of the program. It often happens that when the global invariant used in an Ashcroft-method proof is decomposed in the obvious way, the original Owicki-Gries method cannot prove its invariance; a different and often more complicated annotation must be used. This is not the case with the strengthened version. If the Ashcroft method can prove invariance of a global assertion, then the strengthened Owicki-Gries method can prove the invariance of the corresponding annotation.

Strengthening the Owicki-Gries method makes it easier to construct proofs; it does not change what can be proved. The global invariant used in an Ashcroft-style proof can always be translated into a proof with the original Owicki-Gries method by simply attaching the global invariant to all control points, though of course this defeats the whole purpose of the method, which is to decompose the invariant. Moreover, even though the original Owicki-Gries method fails on one simple decomposition of the invariant, there may be another equally simple decomposition for which it does work. What we claim is that using the strengthened method requires less cleverness than using the original method. Finding the proper annotation to prove a property of a concurrent program is a difficult art; anything that makes the task easier should be welcome.

Section 2 examines two simple algorithms. The first illustrates the Ashcroft and Owicki-Gries methods and shows why control predicates can permit a simpler program annotation than dummy variables. However, it does not convincingly demonstrate the need for control predicates because an extra lemma allows the same proof to be written with dummy variables. In Section 2.5, another algorithm is considered, and a proof using control predicates is given that cannot be so easily rewritten as one using dummy variables.

To simplify the exposition, we consider $n$-process programs of the form

$$\textbf{cobegin}\ \Pi_0\ \Box\ \ldots\ \Box\ \Pi_{n-1}\ \textbf{coend}$$

with each process $\Pi_i$ consisting of a sequence of statements

$$\langle S_1 \rangle ; \langle S_2 \rangle ; \ldots ; \langle S_k \rangle$$

where the angle brackets denote atomic operations. The atomic statements $\langle S_i \rangle$

$\alpha_i$: $\langle\, x_i := \mathit{true} \,\rangle$;
$\beta_i$: $\langle\, \textbf{when } \neg x_{i\oplus 1} \textbf{ do } \mathit{skip} \,\rangle$
$cs_i$: $\langle$ critical section $\rangle$
$\delta_i$: $\langle\, x_i := \mathit{false} \,\rangle$

Fig. 1. A simple algorithm—process $i$'s program.

are either ordinary assignment statements or statements of the form

$$\langle\, \textbf{when } b \textbf{ do } S \,\rangle$$

This is a synchronization primitive that causes the process to wait until the boolean expression $b$ is true, whereupon it executes $S$ as an atomic action. Thus, the semaphore operation $P(s)$ can be represented as

$$\langle\, \textbf{when } s > 0 \textbf{ do } s := s - 1 \,\rangle$$

Since we are concerned only with safety properties, it does not matter whether one assumes any fairness properties of the **when** statement. However, it is important that the evaluation of $b$ and, if it evaluates to *true*, the subsequent execution of $S$ are a single atomic action.

By restricting attention to such "straight-line processes", we avoid some irrelevant issues raised by branching and looping constructs. These constructs are discussed in Section 3.4.

## 2. EXAMPLES

### 2.1 A Simple Example

We begin with a simple algorithm containing two processes, numbered 0 and 1. The program for each process $i$ is shown in Figure 1, where $\oplus$ denotes addition modulo 2. This algorithm is a simplified version of a popular mutual exclusion protocol. (In simplifying it, we have eliminated almost all semblance of a real mutual exclusion algorithm.) We assume that process $i$'s critical section statement does not modify $x_i$ or $x_{i\oplus 1}$.

The property to be proved for this program is that both processes are not simultaneously at their critical sections. For any label $\lambda$, let $at(\lambda)$ be the control predicate that is true if and only if the process's control is at the point labeled $\lambda$. We must prove that $\neg(at(cs_0) \wedge at(cs_1))$ is always true.

In any assertional method, one shows that an assertion $P$ is always true by exhibiting a global assertion $I$ such that:

(1) $I$ is true of the initial state.

(2) $I$ implies $P$.

(3) $I$ is invariant—that is, any program step executed with $I$ true leaves it true.

In our example, $P$ is the assertion $\neg(at(cs_0) \wedge at(cs_1))$.

### 2.2 The Ashcroft Method

In the Ashcroft method, one simply writes the global assertion $I$ as a single formula. For our example, let $I$ be the assertion

$$\neg(at(cs_0) \wedge at(cs_1)) \;\wedge\; \bigwedge_{i=0,1} (at(\beta_i) \vee at(cs_i)) \Rightarrow x_i \qquad (1)$$

where $\Rightarrow$ denotes logical implication. Initially, both processes are at control point $\alpha_i$ and $I$ is trivially true, so condition 1 holds. Condition 2 is obvious, so we need prove only condition 3—the invariance of $I$.

The invariance of $I$ means that executing any atomic action of the program starting with $I$ true leaves $I$ true. Let $\langle \lambda \rangle$ denote the atomic statement with label $\lambda$. To prove the invariance of $I$ we must prove $\{I\}\langle \lambda \rangle\{I\}$ for every atomic program statement $\lambda$, where $\{P\}\langle \lambda \rangle\{Q\}$ is the Hoare logic formula asserting that if $\langle \lambda \rangle$ is executed with $P$ true, then $Q$ will be true after its execution [4]. (By definition of atomicity, an atomic action can be executed only if it terminates.) Note that, unlike the Hoare logic ordinarily used for sequential programs, we allow pre- and postconditions to contain control predicates.

Verifying $\{I\}\langle \lambda \rangle\{I\}$ for each atomic operation $\langle \lambda \rangle$ in the program of Figure 1 is easy. There are four atomic operations in each process, so there are eight formulas to check. However, since the two processes are identical except for the value of $i$, the corresponding operations in both processes can be handled together, leaving only four formulas to verify. We will verify $\{I\}\langle \beta_i \rangle\{I\}$, which is the most interesting one; the reader can check the others.

Statement $\langle \beta_i \rangle$ can be executed only if $at(\beta_i)$ is true, so it follows directly from the meaning of $\{P\}\langle \lambda \rangle\{Q\}$ that $\{I\}\langle \beta_i \rangle\{I\}$ holds if and only if $\{I \wedge at(\beta_i)\}\langle \beta_i \rangle\{I\}$ does. Since $at(cs_i)$ must be true after $\langle \beta_i \rangle$ is executed, instead of proving $\{I \wedge at(\beta_i)\}\langle \beta_i \rangle\{I\}$ we may just as well prove the stronger condition $\{I \wedge at(\beta_i)\}\langle \beta_i \rangle\{I \wedge at(cs_i)\}$. Simple logical manipulation shows that

$$\begin{aligned} I \wedge at(\beta_i) &\equiv at(\beta_i) \wedge x_i \wedge [(at(\beta_{i\oplus 1}) \vee at(cs_{i\oplus 1})) \Rightarrow x_{i\oplus 1}] \\ I \wedge at(cs_i) &\equiv at(cs_i) \wedge x_i \wedge \neg at(cs_{i\oplus 1}) \wedge (at(\beta_{i\oplus 1}) \Rightarrow x_{i\oplus 1}) \end{aligned}$$

We must therefore show

$$\begin{aligned} &\{at(\beta_i) \wedge x_i \wedge [(at(\beta_{i\oplus 1}) \vee at(cs_{i\oplus 1})) \Rightarrow x_{i\oplus 1}]\}\ \langle \beta_i \rangle \\ &\qquad \{at(cs_i) \wedge x_i \wedge \neg at(cs_{i\oplus 1}) \wedge (at(\beta_{i\oplus 1}) \Rightarrow x_{i\oplus 1})\} \end{aligned} \tag{2}$$

Executing $\langle \beta_i \rangle$ does not change the value of any program variable or of the control state of process $i \oplus 1$, so the only part of the postcondition that is not immediately obvious is $\neg at(cs_{i\oplus 1})$. Statement $\langle \beta_i \rangle$ can be executed only when $x_{i\oplus 1}$ equals *false*, and the precondition implies that, in this case, $at(cs_{i\oplus 1})$ must also be false. Hence, after executing $\langle \beta_i \rangle$, $at(cs_{i\oplus 1})$ is false, which proves (2). Formal proof rules for deriving this kind of formula are given in Section 3.1.

## 2.3 The Strengthened Owicki-Gries Method

In the Owicki-Gries method, the invariant is written as a program annotation. An annotation in which, for every $\lambda$, assertion $I^\lambda$ is attached to control point $\lambda$ represents the assertion

$$\bigwedge_{\lambda} (at(\lambda) \Rightarrow I^\lambda) \tag{3}$$

To reformulate the proof above in the Owicki-Gries method, using control predicates, we must write the invariant (1) in the form of (3). Using the equivalence

$$\neg(at(cs_0) \wedge at(cs_1)) \equiv \bigwedge_{i=0,1} (at(cs_i) \Rightarrow \neg at(cs_{i\oplus 1}))$$

$$\begin{array}{rl} & \alpha_i\text{: } \langle\, x_i := \mathit{true} \,\rangle; \\ \{x_i\} & \beta_i\text{: } \langle\, \textbf{when } \neg x_{i\oplus 1} \textbf{ do } \mathit{skip} \,\rangle \\ \{x_i \wedge \neg at(cs_{i\oplus 1})\} & cs_i\text{: } \langle\, \text{critical section} \,\rangle \\ & \delta_i\text{: } \langle\, x_i := \mathit{false} \,\rangle \end{array}$$

Fig. 2. Annotation of process $i$'s program.

(1) can be written as

$$\bigwedge_{i=0,1} (at(\beta_i) \Rightarrow x_i) \wedge (at(cs_i) \Rightarrow (x_i \wedge \neg at(cs_{i\oplus 1})))$$

This assertion is expressed as the program annotation of Figure 2.

In the original Owicki-Gries method, the invariance of the assertion $I$ defined by (3) is proved by verifying the following two conditions for each atomic statement $\langle\, \lambda \,\rangle$, where $\lambda^+$ denotes the control point immediately following $\langle\, \lambda \,\rangle$.

*Sequential Correctness*: $\{I^\lambda\}\langle\, \lambda \,\rangle\{I^{\lambda^+}\}$

*Interference Freedom*: For every control point $\nu$ in a different process from $\lambda$: $\{I^\nu \wedge I^\lambda\}\langle\, \lambda \,\rangle\{I^\nu\}$

Sequential correctness asserts that executing $\langle\, \lambda \,\rangle$ starting with $I^\lambda$ (the assertion attached to $\lambda$) true makes $I^{\lambda^+}$ true. Interference freedom asserts that, for each control point $\nu$ in a different process, executing $\langle\, \lambda \,\rangle$ starting with both $I^\lambda$ and $I^\nu$ (the assertion attached to $\nu$) true leaves $I^\nu$ true. Since execution of $\langle\, \lambda \,\rangle$ is possible only when control is at $\lambda$, and that execution leaves the process's control at $\lambda^+$ and does not change the control point of any other process, these two conditions imply $\{I\}\langle\, \lambda \,\rangle\{I\}$. Thus, proving these conditions for every statement $\langle\, \lambda \,\rangle$ proves the invariance of $I$.

Proving sequential correctness for all the atomic actions in a process involves a standard Floyd-method verification of that process's annotation. For our example annotation of Figure 2, proving sequential correctness for $\langle\, \beta_i \,\rangle$ requires proving the following verification condition:

$$\{x_i\}\,\langle\, \beta_i \,\rangle\,\{x_i \wedge \neg at(cs_{i\oplus 1})\} \tag{4}$$

This cannot be proved. Looking only at process $i$, there is no reason why an execution of $\langle\, \beta_i \,\rangle$ starting with $x_i$ true should finish with $at(cs_{i\oplus 1})$ false.

The Owicki-Gries method can be strengthened by allowing the use of the other process's annotation in proving sequential correctness. To prove sequential correctness for a statement $\langle\, \lambda \,\rangle$ of one process, we may assume, as the precondition for $\langle\, \lambda \,\rangle$, not just that $I^\lambda$ is true but that the assertion $I$ defined by the entire annotation is true. In particular, we can assume that each other process is at a control point whose attached assertion is true. Let $I_j$ be the assertion determined by process $j$'s annotation, so

$$I_j \;\equiv\; (at(\beta_j) \Rightarrow x_j) \wedge (at(cs_j) \Rightarrow (x_j \wedge \neg at(cs_{j\oplus 1})))$$

When proving sequential correctness for $\langle\, \beta_i \,\rangle$, we may assume the truth of $I_{i\oplus 1}$, the annotation of process $i \oplus 1$. Therefore, instead of proving (4), we need prove only the weaker condition:

$$\{x_i \wedge I_{i\oplus 1}\}\,\langle\, \beta_i \,\rangle\,\{x_i \wedge \neg at(cs_{i\oplus 1})\} \tag{5}$$

$$\begin{array}{rl} & \alpha_i\colon \ \langle\, x_i := \mathit{true} \,\rangle; \\ \{x_i\} & \beta_i\colon \ \langle\, \textbf{when } \neg x_{i\oplus 1} \textbf{ do } acs_i := \mathit{true} \,\rangle \\ \{x_i \wedge \neg acs_{i\oplus 1}\} & cs_i\colon \ \langle\, \text{critical section; } acs_i := \mathit{false} \,\rangle \\ & \delta_i\colon \ \langle\, x_i := \mathit{false} \,\rangle \end{array}$$

Fig. 3. Annotation of process $i$ with dummy variables.

This condition can be verified, since $I_{i\oplus 1}$ implies that if $x_{i\oplus 1}$ is false (the only case in which $\langle\, \beta_i \,\rangle$ can be executed) then $at(cs_{i\oplus 1})$ must also be false. In fact, except for lacking the obvious hypothesis $at(\beta_i)$, the precondition of (5) is the same as that of (2), and the postcondition of (5) is part of the postcondition of (2).

Sequential correctness for the other atomic operations is easily verified, and the only nontrivial interference-freedom condition to be proved is that executing $\langle\, \beta_i \,\rangle$ does not falsify the assertion attached to $cs_{i\oplus 1}$. This involves verifying

$$\{x_{i\oplus 1} \wedge \neg at(cs_i) \wedge x_i\}\,\langle\, \beta_i \,\rangle\,\{x_{i\oplus 1} \wedge \neg at(cs_i)\}$$

which is true because $\langle\, \beta_i \,\rangle$ cannot be executed when $x_{i\oplus 1}$ is true. (The formula $\{P\}\langle\, \lambda \,\rangle\{Q\}$ asserts that every possible execution of $\langle\, \lambda \,\rangle$ starting from a state in which $P$ is true terminates with $Q$ true, so it is vacuously valid if $\langle\, \lambda \,\rangle$ cannot be executed when $P$ is true.)

### 2.4 The Owicki-Gries Method with Dummy Variables

Let us now try to reformulate the proof above using dummy variables instead of control predicates. The first problem we encounter is that our correctness condition, that $\neg(at(cs_0) \wedge at(cs_1))$ is always true, is a control predicate. We therefore have to introduce a dummy boolean variable $acs_i$ to represent the control predicate $at(cs_i)$, where $acs_i$ is set true by $\langle\, \beta_i \,\rangle$ and set false by $\langle\, cs_i \,\rangle$. This leads to the annotated program of Figure 3.[2]

Let us consider the proof of sequential correctness for statement $\langle\, \beta_i \,\rangle$. The verification condition corresponding to (5) is

$$\{x_i \wedge I_{i\oplus 1}\}\,\langle\, \beta_i \,\rangle\,\{x_i \wedge \neg acs_{i\oplus 1}\} \tag{6}$$

where $I_{i\oplus 1}$ is the assertion

$$(at(\beta_{i\oplus 1}) \Rightarrow x_{i\oplus 1}) \wedge [at(cs_{i\oplus 1}) \Rightarrow (x_{i\oplus 1} \wedge \neg acs_i)]$$

that corresponds to the annotation of Figure 3 for process $i\oplus 1$. We cannot verify (6). The assertion $I_{i\oplus 1}$ implies that $at(cs_{i\oplus 1})$ is false when $x_{i\oplus 1}$ is false; it does not imply that $acs_{i\oplus 1}$ is false when $x_{i\oplus 1}$ is false. Even though we introduced the variable $acs_{i\oplus 1}$ to represent the control predicate $at(cs_{i\oplus 1})$, they are formally different. The implication $at(cs_{i\oplus 1}) \Rightarrow x_{i\oplus 1}$ can be obtained directly from the annotation of process $i \oplus 1$. The implication $acs_{i\oplus 1} \Rightarrow x_{i\oplus 1}$, which is needed to prove (6), is not obtainable directly from the annotation.

There are two ways to correct this problem. The first is to attach to each control point of the program the additional assertion $acs_{i\oplus 1} \Rightarrow x_{i\oplus 1}$. (More precisely, this assertion is conjoined with each of the assertions in the annotation, including the

[2]A proof in the style of [11] requires additional dummy variables; for simplicity, the following argument uses control predicates instead.

$$\begin{array}{rl}
 & \alpha_i\!: \ \langle\, x := i \ \rangle \\
 & \beta_i\!: \ \langle\, \textbf{when } y = -1 \textbf{ do } \textit{skip}\, \rangle \\
 & \gamma_i\!: \ \langle\, y := i \ \rangle \\
\{x = i \Rightarrow y \neq -1\} & \delta_i\!: \ \langle\, \textbf{when } x = i \textbf{ do } \textit{skip}\, \rangle \\
\{P_i\} & cs_i\!: \ \langle\, \text{critical section}\, \rangle \\
\{P_i\} & \epsilon_i\!: \ \langle\, y := -1 \ \rangle
\end{array}$$

Fig. 4. Process $i$ of another mutual exclusion algorithm.

implicit assertion *true* at control points $\alpha_i$ and $\delta_i$.) The resulting annotation can then be verified with the original Owicki-Gries method.

One can always convert an Ashcroft-method proof to a proof in the original Owicki-Gries method with dummy variables by strengthening the assertions. Indeed, this can be done quite trivially by attaching the global invariant to every control point, replacing control predicates with dummy variables. However, the whole point of the Owicki-Gries method is to break the large global assertion of Ashcroft's method into the simpler local assertions of the annotation, making the invariant easier to understand and to verify. If this requires more complicated local assertions, then the Owicki-Gries method may not offer any advantage. In our example, most people would probably prefer the Ashcroft proof to the Owicki-Gries proof with the extra assertion $acs_{i\oplus 1} \Rightarrow x_{i\oplus 1}$ added to all control points.

The second way to fix the problem is to prove a lemma stating that, if the program is started in a proper initial state, then $acs_{i\oplus 1} \Rightarrow x_{i\oplus 1}$ is always true. Such a lemma is easily proved with the original Owicki-Gries method. This is the better approach because, in the spirit of the Owicki-Gries method, it breaks the proof into small parts. The use of such lemmas is described by Schneider in [12]. However, while possible in this case, an Ashcroft method proof cannot always be converted by a simple lemma to an Owicki-Gries method proof with dummy variables. In the next section, an example is given in which the use of dummy variables instead of control points forces one to use a different annotation.

### 2.5 Another Example

Our second example is a highly simplified version of a mutual exclusion protocol used in [6]. It is an $n$-process program, with processes numbered 0 through $n-1$, where process $i$ is given in Figure 4 with its annotation. The shared variables $x$ and $y$ are of type integer, with $y$ initially equal to $-1$. The assertion $P_i$ in the annotation of process $i$ is defined to equal

$$\forall j \neq i : (\neg at(cs_j)) \wedge [(at(\gamma_j) \vee at(\delta_j)) \Rightarrow x \neq j]$$

With the ordinary Owicki-Gries method, proving sequential correctness of this annotation for statement $\langle\, \delta_i \,\rangle$ requires proving the following condition:

$$\{x = i \Rightarrow y \neq -1\}\, \langle\, \delta_i \,\rangle\, \{P_i\}$$

This is not directly provable, since the postcondition asserts (among other things) that in no other process $j$ is control at control point $cs_j$, which cannot be inferred from the precondition. However, in the strengthened method, we are allowed to assume in the precondition that the assertion determined by every other process

$j$'s annotation is true. Letting $I_j$ denote this assertion, so

$$I_j \equiv [at(\delta_j) \Rightarrow (x = j \Rightarrow y \neq -1)] \wedge [at(cs_j) \Rightarrow P_j] \wedge [at(\epsilon_j) \Rightarrow P_j]$$

it suffices to prove the weaker condition

$$\{(x = i \Rightarrow y \neq -1) \wedge at(\delta_i) \wedge \textstyle\bigwedge_{j \neq i} I_j\}\langle\, \delta_i \,\rangle\{P_i\}$$

This formula follows from the observation that $at(\delta_i) \wedge I_j$ implies that, if $at(cs_j)$ is true, then $x \neq i$ and statement $\langle\, \delta_i \,\rangle$ cannot be executed.

The verification of the other sequential correctness conditions and of interference freedom is straightforward and is left to the reader.

In this example, the proof of sequential correctness for $\langle\, \delta_i \,\rangle$ requires assuming that, if another process $j$ is at control point $cs_j$, then the attached assertion $P_j$ is true. However, sequential correctness for $\langle\, \delta_i \,\rangle$ proves that $P_i$ is true when process $i$ reaches control point $cs_i$. Thus, we are using an induction argument, showing that if every other process that has already reached control point $cs_j$ did so with $P_j$ true, then $P_i$ will be true when process $i$ reaches $cs_i$.

In the previous example, the information contained in the annotation of another process needed to prove sequential correctness could be established separately as a simple lemma. We now indicate why this is not the case here. In the sequential correctness proof, the information obtained from the annotation of process $j$ is exactly the result we are trying to prove for process $i$. Assuming the truth of the assertion $I_j$ in the sequential correctness proof for process $i$ is analogous to assuming, in the proof of some theorem by mathematical induction, that the desired result is true for all $j < i$ and proving that it is true for $i$. Trying to replace the assumption that $I_j$ holds for $j \neq i$ by a lemma would be like trying to replace the induction assumption that the theorem is true for all $j < i$ by a lemma, which cannot be done because proving the lemma is equivalent to proving the original theorem.

The correctness of the annotation of Figure 4 cannot be proved with the original form of the Owicki-Gries method, and thus this proof cannot be translated into one using dummy variables instead of control predicates. A different annotation is required when dummy variables are used.

In writing a proof of this algorithm for the original version of [6], we were unable to find a simple annotation that could be proved invariant with the original Owicki-Gries method, and we were forced to introduce the extended method to give a simple proof. Afterwards, J. Misra discovered a proof as simple as ours using dummy variables and the original Owicki-Gries method [2]. We do not know if it is always possible to construct a simple proof with the ordinary Owicki-Gries method, but we do know that it is not always easy.

## 3. THE FORMALISM

The discussion of the examples in the preceding section included an informal explanation of how one applies the Owicki-Gries method using control predicates in the annotation. In this section, we develop a formalism that justifies our informal reasoning. For now, we continue to consider only simple straight-line multiprocess programs. Section 3.4 discusses the extension of the formalism to other control structures.

We continue using the notation that $\lambda$ and $\lambda^+$ denote the control points at and immediately after the atomic statement $\langle \lambda \rangle$. In the examples, $\lambda$ was a unique label written in front of the statement. However, the formalism does not require statement labels; any naming scheme could be used.

### 3.1 Hoare Logic with Control Predicates

To prove that a program $\Pi$ leaves invariant a global assertion $I$, one must prove the Hoare logic formula $\{I\}\langle \lambda \rangle\{I\}$ for every (atomic) statement $\langle \lambda \rangle$ of $\Pi$. (This can be viewed as either the definition of invariance or an application of the Decomposition Principle of [10].)

The presence of control predicates in $P$ and $Q$ makes the formulas $\{P\}\langle \lambda \rangle\{Q\}$ fundamentally different from ordinary Hoare triples. The *Control Predicate Hoare Logic* (CPHL) for reasoning about these formulas is therefore different from ordinary Hoare logic. Consider the statement $\alpha_i$: $\langle x := i \rangle$ from the program of Figure 4. If the assertion $P$ does not mention the variable $x$, then the ordinary Hoare formula $\{P\}x := i\{P\}$ is valid, but the CPHL formula $\{P\}\langle \alpha_i \rangle\{P\}$ need not be valid. For example, even though the predicate $at(\alpha_j)$ does not mention $x$, the formula $\{at(\alpha_j)\}\langle \alpha_i \rangle\{at(\alpha_j)\}$ is valid only if $j \neq i$; it is invalid when $j = i$ because executing $\langle \alpha_i \rangle$ makes $at(\alpha_i)$ false.

CPHL subsumes ordinary Hoare logic through the following rule.

*Subsumption Rule*: For the statement $\lambda$: $\langle S \rangle$, the validity of the ordinary Hoare logic formula $\{P\}S\{Q\}$ (where $P$ and $Q$ do not contain control predicates) implies the validity of the CPHL formula $\{P\}\langle \lambda \rangle\{Q\}$.

Using the subsumption rule, we can derive the following CPHL rule from ordinary Hoare logic:

**when** *Rule*: For the statement $\lambda$: $\langle$ **when** $b$ **do** $S \rangle$, the validity of the ordinary Hoare logic formula $\{P\}S\{Q\}$ implies the validity of $\{P \vee \neg b\}\langle \lambda \rangle\{Q\}$.

Given the axioms and rules of ordinary Hoare logic, the subsumption rule captures the semantics of atomic language constructs. Ordinary Hoare logic also has rules that are independent of the language constructs. These rules, as listed below, are included in CPHL. (They differ from the corresponding rules of ordinary Hoare logic only in allowing control predicates in the pre- and postconditions.)

*Rule of Consequence*: If $\{P\}\langle \lambda \rangle\{Q\}$, $P' \Rightarrow P$, and $Q \Rightarrow Q'$, then $\{P'\}\langle \lambda \rangle\{Q'\}$.
*Disjunction Rule*: If $\{P\}\langle \lambda \rangle\{Q\}$ and $\{P'\}\langle \lambda \rangle\{Q'\}$, then $\{P \vee P'\}\langle \lambda \rangle\{Q \vee Q'\}$.
*Conjunction Rule*: If $\{P\}\langle \lambda \rangle\{Q\}$ and $\{P'\}\langle \lambda \rangle\{Q'\}$, then $\{P \wedge P'\}\langle \lambda \rangle\{Q \wedge Q'\}$.

Thus far, all our CPHL rules are derived from ordinary Hoare logic rules. Reasoning about control predicates requires the following additional axiom and rule. Their soundness is self-evident. Recall that $\lambda^+$ denotes the control point immediately following statement $\langle \lambda \rangle$.

*Noninterference Axiom*: If $\nu$ is a control point in a different process from $\langle \lambda \rangle$, then $\{at(\nu)\}\,\langle \lambda \rangle\,\{at(\nu)\}$
*Locality Rule*: If $\{P \wedge at(\lambda)\}\,\langle \lambda \rangle\,\{Q \vee \neg at(\lambda^+)\}$ then $\{P\}\langle \lambda \rangle\{Q\}$.

Since $\{true\}\langle \lambda \rangle\{true\}$ always holds (by the Subsumption Rule), the Locality Rule implies $\{true\}\langle \lambda \rangle\{at(\lambda^+)\}$. The Rule of Consequence then implies the following general result.

*Control Theorem*: $\{at(\lambda)\}\,\langle\,\lambda\,\rangle\,\{at(\lambda^+)\}$

In addition to these rules and axioms, we need axioms for proving simple formulas about state predicates. For example, we must be able to prove that, if $\nu$ and $\mu$ are different control points in the same process, then $at(\nu) \wedge at(\mu) \equiv \mathit{false}$. Such axioms are given in [7] for a more complicated language; we do not consider them here.

Observe that CPHL has no equivalent to the Rule of Composition of ordinary Hoare logic—the rule for reasoning about the "**;**" construction. The semantics of the "**;**" are given by the Control Theorem, together with the implicit rule for calculating $\lambda^+$. (For example, in the program of Figure 4, we know that $\alpha_i^+ = \beta_i$.) As we shall see, it is characteristic of CPHL that flow of control is specified by relations among control predicates rather than by the special inference rules of ordinary Hoare logic.

As an illustration of how the rules of CPHL are applied, we sketch the formal proof of (5) from our first example. By the Rule of Consequence and the definition of $I_{i\oplus 1}$, it suffices to prove

$$\{x_i \wedge (at(cs_{i\oplus 1}) \Rightarrow x_{i\oplus 1})\}\,\langle\,\beta_i\,\rangle\,\{x_i \wedge \neg at(cs_{i\oplus 1})\}$$

Expressing the precondition as a disjunction and applying the Disjunction Rule reduces the problem to proving the following two conditions:

$$\{x_i \wedge x_{i\oplus 1}\}\ \langle\,\beta_i\,\rangle\ \{x_i \wedge \neg at(cs_{i\oplus 1})\} \tag{7}$$

$$\{x_i \wedge \neg at(cs_{i\oplus 1})\}\ \langle\,\beta_i\,\rangle\ \{x_i \wedge \neg at(cs_{i\oplus 1})\} \tag{8}$$

Formula (7) follows from the Rule of Consequence and the formula

$$\{x_{i\oplus 1}\}\langle\,\beta_i\,\rangle\{\mathit{false}\}$$

which is a consequence of the **when** Rule (with *false* substituted for both $P$ and $Q$).

To prove (8), we apply the Conjunction Rule to break it into the two conditions:

$$\{x_i\}\ \langle\,\beta_i\,\rangle\ \{x_i\}$$

$$\{\neg at(cs_{i\oplus 1})\}\ \langle\,\beta_i\,\rangle\ \{\neg at(cs_{i\oplus 1})\}$$

The first follows from the proof rule for the **when** statement. To prove the second, we use the equivalence

$$\neg at(cs_{i\oplus 1}) \equiv at(\alpha_{i\oplus 1}) \vee at(\beta_{i\oplus 1}) \vee at(\delta_{i\oplus 1}) \vee at(\delta^+_{i\oplus 1})$$

and the Disjunction Rule, and we apply the Noninterference Axiom four times.

### 3.2 The Strengthened Owicki-Gries Method

We assume an $n$-process program $\Pi$ with processes $\Pi_0, \ldots, \Pi_{n-1}$. We let $\nu \in \Pi$ mean that $\nu$ is a control point of $\Pi$, and similarly $\nu \in \Pi_i$ means that $\nu$ is a control point of process $\Pi_i$.

In the Owicki-Gries method, the invariant $I$ has the form

$$\bigwedge_{\nu \in \Pi} at(\nu) \Rightarrow I^\nu \tag{9}$$

where $I^{\nu}$ is the assertion attached to control point $\nu$. Let $I_j$ denote $\bigwedge_{\nu \in \Pi_j} at(\nu) \Rightarrow I^{\nu}$, the assertion represented by the annotation of process $\Pi_j$. If $\langle\, \lambda \,\rangle$ is a statement of process $\Pi_i$, then

$$at(\lambda) \wedge I \;\equiv\; at(\lambda) \wedge I^{\lambda} \wedge \bigwedge_{j \neq i} I_j$$
$$at(\lambda^{+}) \wedge I \;\equiv\; at(\lambda^{+}) \wedge I^{\lambda^{+}} \wedge \bigwedge_{j \neq i} I_j$$

Thus, by the Locality Rule and the Rule of Consequence, to prove the invariance condition $\{I\}\langle\, \lambda \,\rangle\{I\}$ it suffices to prove:

$$\{I^{\lambda} \wedge \textstyle\bigwedge_{j \neq i} I_j\}\,\langle\, \lambda \,\rangle\,\{I^{\lambda^{+}} \wedge \textstyle\bigwedge_{j \neq i} I_j\} \tag{10}$$

In the standard Owicki-Gries method, one applies the Conjunction Rule to break the verification of (10) into two parts:

$$\{I^{\lambda}\}\;\langle\, \lambda \,\rangle\;\{I^{\lambda^{+}}\} \tag{11}$$
$$\forall j \neq i\; \forall \nu \in \Pi_j :\; \{I^{\lambda} \wedge (at(\nu) \Rightarrow I^{\nu})\}\;\langle\, \lambda \,\rangle\;\{at(\nu) \Rightarrow I^{\nu}\} \tag{12}$$

Condition (11) is sequential correctness for $\langle\, \lambda \,\rangle$. To verify (12), we write $at(\nu) \Rightarrow I^{\nu}$ as $I^{\nu} \vee \neg at(\nu)$ and apply the Disjunction Rule and the Rule of Consequence to decompose it into the problem of verifying the following two conditions:

$$\forall j \neq i\; \forall \nu \in \Pi_j :\; \{I^{\lambda} \wedge I^{\nu}\}\;\langle\, \lambda \,\rangle\;\{I^{\nu}\} \tag{13}$$
$$\forall j \neq i\; \forall \nu \in \Pi_j :\; \{\neg at(\nu)\}\;\langle\, \lambda \,\rangle\;\{\neg at(\nu)\} \tag{14}$$

Condition 13 is interference freedom. Since $\neg at(\nu) \equiv \bigvee_{\mu \in \Pi_j, \mu \neq \nu} at(\mu)$ (because control must be somewhere in process $j$), formula (14) follows from the Disjunction Rule and the Noninterference Axiom.

Formulas (11) and (13) represent the sequential correctness and interference freedom conditions of the standard Owicki-Gries method. Since our goal is to prove the invariance of $I$, it is easy to see that we can weaken these conditions (by strengthening their preconditions) as follows:

*Weak Sequential Correctness*: $\{I^{\lambda} \wedge \bigwedge_{j \neq i} I_j\}\,\langle\, \lambda \,\rangle\,\{I^{\lambda^{+}}\}$
*Weak Interference Freedom*: $\forall j \neq i\; \forall \nu \in \Pi_j$ :
$$\{I^{\lambda} \wedge I^{\nu} \wedge at(\nu) \wedge \textstyle\bigwedge_{k \neq i,j} I_k\}\,\langle\, \lambda \,\rangle\,\{I^{\nu}\}$$

It is this weak sequential correctness condition that we used in our two examples. The weak interference freedom condition is weaker than (13) because, to prove that executing the statement $\langle\, \lambda \,\rangle$ of process $i$ leaves invariant the assertion $I^{\nu}$ attached to process $j$, we are allowed to use the additional hypothesis that, for any third process $k$, the assertion $I_k$ defined by the annotation of process $k$ is true.

We did not need the weak interference freedom condition in our two examples. (Indeed, except for the extra hypothesis $at(\nu)$, it is the same as the original condition (13) when there are only two processes, as in our first example.) In most of the concurrent algorithms that we have studied, safety properties can be proved by considering the processes two at a time, so the stronger postcondition employed in the weak interference freedom condition does not help. However, as the examples indicate, the weak sequential correctness condition is very useful.

### 3.3 Equivalence to the Ashcroft Method

We now show that the strengthened Owicki-Gries method is as powerful as the Ashcroft method. More precisely, we prove that, given an assertion $I$ of the form (9), the CPHL formula $\{I\}\langle\,\lambda\,\rangle\{I\}$ that must be verified (for all $\lambda$) with the Ashcroft method is provable if and only if the weak sequential correctness and interference freedom conditions for $\lambda$ are provable. The proof assumes the ability to prove simple logical equivalences among predicates. This means that, barring some pathological weakness in the ability to reason about predicates, an annotation can be proved correct with the strengthened Owicki-Gries method if and only if the corresponding global assertion can be proved invariant with the Ashcroft method.

We showed above that the two weak verification conditions of the extended Owicki-Gries method imply the Ashcroft method condition $\{I\}\langle\,\lambda\,\rangle\{I\}$; we now show the converse. Recall that $I_j \equiv \bigwedge_{\nu\in\Pi_j}(at(\nu) \Rightarrow I^\nu)$, so $I \equiv \bigwedge_j I_j$. Our proof is based upon the equivalence

$$I_j \equiv \bigvee_{\nu\in\Pi_j} (at(\nu) \wedge I^\nu) \tag{15}$$

which follows from the observation that $\left(\bigvee_{\nu\in\Pi_j} at(\nu)\right) \equiv \mathit{true}$ and, for any $\nu, \mu \in \Pi_j$ with $\nu \neq \mu$: $at(\nu) \wedge at(\mu) \equiv \mathit{false}$.

Assume $\{I\}\langle\,\lambda\,\rangle\{I\}$. From the Control Theorem, the Conjunction Rule, and the observation that $I_i \wedge at(\lambda) \equiv I^\lambda \wedge at(\lambda)$, we infer

$$\{I^\lambda \wedge at(\lambda) \wedge \textstyle\bigwedge_{j\neq i} I_j\}\,\langle\,\lambda\,\rangle\,\{I\} \tag{16}$$

The weak sequential correctness condition now follows from the Locality Rule and the Rule of Consequence.

To prove the validity of the weak noninterference condition, we use (15) to substitute for $I_j$ and apply the distributive law for the logical operators to rewrite (16) as

$$\{\textstyle\bigvee_{\nu\in\Pi_j} I^\lambda \wedge at(\lambda) \wedge I^\nu \wedge at(\nu) \wedge \bigwedge_{k\neq i,j} I_k\}\,\langle\,\lambda\,\rangle\,\{I\}$$

The weak noninterference condition now follows from the Locality Rule and the Rule of Consequence.

### 3.4 Other Control Structures

To indicate how sequential control structures are handled, we consider first the **while** statement. Suppose a process contains

**while** $\beta{:}\langle\,b\,\rangle$ **do** $\sigma{:}S$ **od**; $\gamma$: ...

where $\beta$, $\sigma$, and $\gamma$ are labels and $S$ is any sequence of statements. The angle brackets indicate that the evaluation of the expression $b$ is a single atomic action. The evaluation of $b$ is one of the atomic operations of the program; to prove the invariance of an assertion $I$, we must show this evaluation leaves $I$ true. In other words, we must prove the CPHL formula $\{I\}\langle\,\beta\,\rangle\{I\}$, where $\langle\,\beta\,\rangle$ denotes the evaluation of the condition in the **while** statement.

Ordinary Hoare logic includes only formulas $\{P\}S\{Q\}$ in which $S$ is a complete statement; it has no such formulas as $\{I\}\langle\,\beta\,\rangle\{I\}$ where $\langle\,\beta\,\rangle$ is a **while**-statement test. The need for these formulas indicates that the Owicki-Gries method is a generalization of Floyd's method, which has a proof rule for flowchart "test" boxes, rather than Hoare's method, which doesn't. (The generalized Hoare logic of concurrency, described in [10], does not have these Floyd-like rules.)

The proof rule for the **while** test $\langle \beta \rangle$ is complicated by the fact that, after its execution, control is either at $\sigma$ or at $\gamma$. Hence, there is no unique successor control point $\beta^+$. It is useful to define the control predicate $\mathit{after}(\lambda)$ to be true if and only if control is immediately after the statement or test $\langle \lambda \rangle$. For an assignment or **when** statement, $\mathit{after}(\lambda) \equiv \mathit{at}(\lambda^+)$. However, for the **while** statement above, $\mathit{after}(\beta) \equiv \mathit{at}(\sigma) \vee \mathit{at}(\gamma)$. We generalize the Locality Rule and the Control Theorem to more general control structures by replacing $\mathit{at}(\lambda^+)$ with $\mathit{after}(\lambda)$. All our rules for reasoning about concurrent programs, including the strengthened Owicki-Gries method for proving invariance, then remain valid, where we define

$$I^{\beta^+} \;\equiv\; (\mathit{at}(\sigma) \Rightarrow I^{\sigma}) \wedge (\mathit{at}(\gamma) \Rightarrow I^{\gamma})$$

when $\langle \beta \rangle$ is the **while** test above.

To enable us to prove CPHL formulas for the atomic action $\beta$, we need the following axiom:

**while** *Test Axiom*: If $P$ contains no control predicates, then

$$\{P\}\,\langle \beta \rangle\,\{(\mathit{at}(\sigma) \wedge P \wedge b) \vee (\mathit{at}(\gamma) \wedge P \wedge \neg b)\}$$

This axiom does not completely define the semantics of the **while** statement; additional axioms are needed to specify the flow of control. We already mentioned one such axiom: $\mathit{after}(\beta) \equiv \mathit{at}(\sigma) \vee \mathit{at}(\gamma)$. This asserts that, after executing the test, control goes to either $\sigma$ or $\gamma$. We also need to specify that, after executing $S$, control goes back to $\beta$. Define $\mathit{after}(S)$ to be $\mathit{after}(\lambda)$, where $\lambda{:}\,\langle S_n \rangle$ is the last statement in the list $S$ of atomic statements. The axiom $\mathit{after}(S) \equiv \mathit{at}(\beta)$ asserts that control loops back to $\beta$ after executing the body of the **while** statement. The semantics of the **while** statement are captured by the **while** Test Axiom and these two axioms about control predicates.

Other sequential control structures are handled similarly. For example, consider the statement

**if** $\beta{:}\langle b \rangle$ **then** $\sigma{:}S$ **fi** ; $\gamma{:}\ \ldots$

The axiom for the test $\langle \beta \rangle$ in this statement is identical to the **while** Test Axiom above. The flow of control axioms are: $\mathit{after}(\beta) \equiv \mathit{at}(\sigma) \vee \mathit{at}(\gamma)$ and $\mathit{after}(S) \equiv \mathit{at}(\gamma)$.

Observe that the only difference in the axioms for the **while** and **if** statements are in the axiom for $\mathit{after}(S)$. This reflects the fact that the only difference between the two statements is that, after executing $S$, the **while** loops back to the beginning and the **if** continues to the following statement. In CPHL, flow of control is described by relations among control predicates, not by special inference rules.

One can also extend the Owicki-Gries method to programs having any process structure that can be expressed with nested **cobegin** statements. In this case, the interference freedom condition must be generalized by letting $\nu$ range over all control points in concurrently active processes. (These control points are determined syntactically.) Control predicate axioms assert that control is at the beginning of each clause (process) when it is at the beginning of the **cobegin**, and control is at the point immediately following the **coend** when it is at the end of each clause. Care must also be exercised in defining $I^{\lambda}$ and $I^{\lambda^+}$ for the control points immediately before and after the **cobegin** when applying the method.

## 4. DISCUSSION

We have shown how the Owicki-Gries method can be strengthened by using weaker sequential correctness and interference freedom conditions. The significant change is the weaker sequential correctness condition, which permits the use of information from other processes' annotations. This strengthening is useful only when control predicates appear in the annotation; it is of no benefit if the control predicates are replaced by dummy variables, as in the method originally advocated by Owicki and Gries. Unlike the original Owicki-Gries method, the strengthened version has the property that it works for any annotation that represents an invariant assertion.

When expressed formally, the weak sequential correctness and interference freedom conditions are more complicated than the original ones (11) and (13). However, this is a welcome complication because it adds hypotheses to the precondition of a Hoare formula. In practice, one adds only those extra hypotheses that are useful. (Formally, this means applying the Rule of Consequence.)

The significant distinction between control predicates and dummy variables is not between predicates and variables, but between control and "dummy". When proving properties of concurrent programs, one must reason about the control state. Although dummy variables can be used to represent the control state, the lack of a formal connection between these variables and the control predicates that they represent limits their utility.

As mentioned in [9], control predicates can be viewed as implicit variables. (We prefer the term "implicit" to "dummy" or "auxiliary" because these variables represent a part of the program state that is just as real as that represented by ordinary variables; they differ from ordinary variables only in that the programming language provides no explicit mechanism for representing their values.) Relations among control predicates, such as $after(\beta) \equiv at(\sigma) \vee at(\gamma)$, become aliasing relations among these variables. Our Control Predicate Hoare Logic can be obtained by extending the ordinary Hoare logic to handle aliasing relations (as in [9]) and assertions containing implicit variables.

Considering control predicates to be implicit variables can provide a more elegant formal justification of the Owicki-Gries method, but it does not change the way the method is used to reason about specific programs. This formal approach works best with the generalized Hoare logic of concurrency. It provides one of the techniques used in [5] to define a formal semantics for concurrent programming languages.

### ACKNOWLEDGMENTS

I wish to thank Fred Schneider for his help in clarifying the distinction between CPHL and ordinary Hoare logic and Edsger Dijkstra for communicating Misra's correctness proof of the algorithm of Figure 4.

### REFERENCES

1. Ashcroft, E. Proving assertions about parallel programs. *J. Comput. Systm. Sci. 10* (Jan. 1975), 110–135.
2. Dijkstra, E. W. Misra's proof for Lamport's mutual exclusion. Nov. 1985. EWD948.

3. FLOYD, R. W. Assigning meanings to programs. In *Proceedings of the Symposium on Applied Math., Vol. 19* (1967), American Mathematical Society, pp. 19–32.
4. HOARE, C. An axiomatic basis for computer programming. *Commun. ACM 12*, 10 (Oct. 1969), 576–583.
5. LAMPORT, L. An axiomatic semantics of concurrent programming languages. In *Logics and Models of Concurrent Systems* (Berlin, 1985), K. R. Apt, Ed., Springer-Verlag, pp. 77–122.
6. LAMPORT, L. A fast mutual exclusion algorithm. 1985. Submitted for publication.
7. LAMPORT, L. The 'Hoare logic' of concurrent programs. *Acta Inf. 14*, 1 (1980), 21–37.
8. LAMPORT, L. Proving the correctness of multiprocess programs. *IEEE Trans. Softw. Eng. SE-3*, 2 (March 1977), 125–143.
9. LAMPORT, L., AND SCHNEIDER, F. B. Constraints: a uniform approach to aliasing and typing. In *Proceedings of the Twelfth ACM Symposium on Principles of Programming Languages* (New Orleans, Jan. 1985), ACM SIGACT-SIGPLAN.
10. LAMPORT, L., AND SCHNEIDER, F. B. The "Hoare logic" of CSP, and all that. *ACM Trans. Program. Lang. Syst. 6*, 2 (Apr. 1984), 281–296.
11. OWICKI, S., AND GRIES, D. An axiomatic proof technique for parallel programs. *Acta Inf. 6*, 4 (1976), 319–340.
12. SCHNEIDER, F. B., AND ANDREWS, G. Concepts for concurrent programming. In *Current Trends in Concurrency* (1986), J. W. de Bakker, W. P. de Roever, and G. Rozenberg, Eds., Springer-Verlag.


Received July 1986; revised January 1987; accepted February 1987