Pretending Atomicity

Leslie Lamport
Fred Schneider*

TR 89-1005
May 1989

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*This research is supported in part by the Office of Naval Research under contract N00014-86-D-0092, the National Science Foundation under Grant No. CCR-8701103, and Digital Equipment Corporation. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not reflect the views of these agencies.

Pretending Atomicity

Leslie Lamport

Digital Equipment Corporation
Systems Research Center
Palo Alto, California 94301

Fred B. Schneider*

Department of Computer Science
Cornell University
Ithaca, New York 14853

May 14, 1989

ABSTRACT

We present a theorem for deriving properties of a concurrent program by reasoning about a simpler, coarser-grained version. The theorem generalizes a result that Lipton proved for partial correctness and deadlock-freedom. Our theorem applies to all safety properties.

*Supported in part by the Office of Naval Research under contract N00014-86-K-0092, the National Science Foundation under Grant No. CCR-8701103, and Digital Equipment Corporation. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not reflect the views of these agencies.

Contents

1 Introduction

To specify a concurrent program, one must specify what its atomic actions are. If $x := x + 1$ is executed as a single atomic action, then

$$\textbf{cobegin } x := x + 1 \,\square\, x := x + 1 \textbf{ coend}$$

increments x by 2; if each read and store of x is a separate atomic action, then it increments x by 1 or 2.

We specify that a statement is executed as a single atomic action by enclosing it in angle brackets. For example, $\langle\, x := x+1 \,\rangle$ is a statement that is executed as one atomic action. A statement $x := x+1$ in which each read and store of x is a separate atomic action can be written as

$$\langle\, t := x \,\rangle;\ \langle\, t := t+1 \,\rangle;\ \langle\, x := t \,\rangle$$

where t is a new variable that is local to the process and represents an "accumulator".

Representing a program using fewer atomic actions simplifies reasoning about it. One way to reduce the number of atomic actions in a program is to combine two or more atomic actions into a single larger one. This is often done by pretending that a statement is atomic if its execution contains at most one access (read or write) of a shared variable, tacitly applying what we will call the *single-action rule*. For the example above, applying this rule would allow

$$\langle\, t := x \,\rangle;\ \langle\, t := t+1 \,\rangle$$

to be combined into the single atomic action $\langle\, t := x + 1 \,\rangle$.

The single-action rule cannot always be applied. For example, it would imply that any operation can be considered atomic in a single-process program, because no variable is shared. This would mean that a property of the program

$$\langle\, y := x + 1 \,\rangle;\ \langle\, x := y \,\rangle \tag{1}$$

could be established by proving it for the program

$$\langle\, y := x + 1;\ x := y \,\rangle \tag{2}$$

This reasoning is wrong. The following property holds for the second program but not the first.

> If the program is started in a state with $x = y$, then $x = y$ holds in all states reached during execution.

Execution of (1) reaches an intermediate state in which $x \neq y$—a state that does not occur when executing (2).

In this paper, we derive a general rule for combining atomic actions. It includes a correct version of the single-action rule as a corollary. Our rule applies only to safety properties, which include partial correctness, mutual exclusion, and deadlock-freedom, but not to liveness properties, such as termination and starvation-freedom. A safety property asserts that "something bad does not happen", so if it is violated, then it is violated by a finite portion of a (possibly infinite) execution of the program.

The idea of combining atomic actions is probably as old as the study of concurrent algorithms. To our knowledge, the single-action rule was first mentioned in print by Owicki and Gries [10], where it was informally claimed for partial correctness properties. In [9], Lipton formally proved a closely related theorem for partial correctness and deadlock-freedom. However, Lipton was primarily concerned with semaphore operations, and it was not widely recognized that the single-action rule is a corollary of his results. Doeppner [4] extended Lipton's partial-correctness result to a somewhat larger class of safety properties. In this paper, we extend Lipton's and Doeppner's results to a more general class of safety properties.

2 Lipton's Theorem

Before describing our result, we give an informal review of Lipton's work [9]. The hypotheses of his main theorem involve commutativity relations between atomic actions. We begin by defining these relations, departing somewhat from Lipton's original notation.

Henceforth, we refer to atomic actions simply as actions. Formally, an action α is a set of pairs of program states, where $(t, u) \in \alpha$ means that executing α in state t can produce state u. We say that α is *enabled* in state t iff (if and only if) there is a state u such that $(t, u) \in \alpha$. We write $t \xrightarrow{\alpha} u$ to denote that (t, u) is an element of α. For example, a semaphore operation $P(sem)$ is represented by an action α that is enabled in state t iff control is at that operation and the value of sem is positive. For this action, $t \xrightarrow{\alpha} u$ holds iff (i) α is enabled in state t and (ii) state u is the same as t, except that control is after the semaphore operation and the value of sem is one less than its value in t.

The program state includes control information, in addition to the values of program variables. Thus, two instances of a statement $\langle\, x := x + 1 \,\rangle$ in a program are different actions because they have different effects on the control components of the state.

If α and β are actions, then $\alpha\beta$ is defined to be the action such that $t \xrightarrow{\alpha\beta} u$ iff there exists a v such that $t \xrightarrow{\alpha} v$ and $v \xrightarrow{\beta} u$. An action ρ *right commutes* with an action α iff $t \xrightarrow{\rho\alpha} u$ implies $t \xrightarrow{\alpha\rho} u$, for every pair of states t, u. In other words, ρ right commutes with α means that if it is possible to execute first ρ then α, then it is possible to produce the same state by executing first α then ρ. Similarly, λ *left commutes* with α iff $t \xrightarrow{\alpha\lambda} u$ implies $t \xrightarrow{\lambda\alpha} u$ for every pair of states t, u. Thus, ρ right commutes with λ iff λ left commutes with ρ. Two actions *commute* iff each one left commutes and right commutes with the other.

The hypotheses of Lipton's main theorem involve commutativity between actions in different processes. An action ρ in a process is called a *right mover* iff it right commutes with the actions of every other process. An action λ is a *left mover* iff it left commutes with the actions in every other process.

Lipton observed that, if semaphore operations are represented as atomic actions, then P actions are right movers and V actions are left movers. To see that P actions are right movers, let ρ be a $P(sem)$ action, let λ be an action in another process, and assume that executing ρ then λ from state t can produce state u. There are three cases to consider.

- λ does not access the semaphore *sem*. In this case, ρ can obviously be executed after λ to produce the same state u.

- λ is another $P(sem)$ action. Executing the two $P(sem)$ actions in either order must produce the same state.

- λ is a $V(sem)$ action. In this case, executing λ from state t produces a state with $sem > 0$, so ρ can then be executed to produce state u. (Note that ρ does not left commute with λ because, in a state with $sem = 0$, it is possible to execute a $V(sem)$ followed by a $P(sem)$, but not a $P(sem)$ followed by a $V(sem)$.)

Similar reasoning shows that every V action is a left mover.

To combine actions, Lipton introduced the notion of *reducing* a program by a statement. Let S be a sequence $\langle S_1\rangle; \langle S_2\rangle; \ldots; \langle S_k\rangle$ of statements in a program Π. Program Π reduced by S, denoted Π/S, is the program obtained from Π by replacing S with the single atomic statement $\langle S_1; \ldots; S_k\rangle$. Lipton proved the following result.

Lipton's Theorem *Let Π be a program and S have the form $\langle S_1\rangle; \langle S_2\rangle; \ldots; \langle S_k\rangle$, where, for some i:*

1. *$S_1, \ldots, S_{i-1}$ are right movers.*
2. *$S_{i+1}, \ldots, S_k$ are left movers.*
3. *From any program state in which execution of S has begun but not terminated, it is possible, by executing only actions in S, to reach a state in which S has terminated.*

Then, programs Π and Π/S satisfy the same partial correctness and deadlock-freedom properties.

The single-action rule asserts that, if S contains at most one access to a shared variable, then we can prove a property of program Π by proving it for Π/S. If an action α does not access any variable that is accessed by any other process, then α is both a left and a right mover. Letting $\langle S_i\rangle$ be the single statement in S that accesses a shared variable (or any statement if S does not access a shared variable), Lipton's Theorem implies the single-action rule for reasoning about partial correctness and deadlock freedom—except that the single-action rule does not require hypothesis 3. We will show that hypothesis 3 is not needed in Lipton's Theorem for partial correctness properties, so the single-action rule is valid for partial correctness.

Partial correctness relates initial and final states, but makes no assertion about states in which control is inside S. Doeppner extended Lipton's result to a more general class of safety properties that also assert nothing when control is within S. A precise statement of Doeppner's result is given below.

To use Lipton's Theorem (or Doeppner's extension), one usually performs many reductions to decrease the number of separate actions in a program. We now show that these reductions can all be done at once. Let S and S' be two disjoint sequences of statements. We show that if S and S' both satisfy the hypotheses of Lipton's Theorem, then $(\Pi/S)/S'$, which equals $(\Pi/S')/S$, and Π satisfy the same partial correctness and deadlock-freedom properties. Since S satisfies the hypotheses, Π/S and Π satisfy the same properties. An action that left or right commutes with every action of S in program Π must left or right commute with $\langle S\rangle$ in program Π/S. Therefore, if S' satisfies the hypotheses of Lipton's Theorem in program Π, then it also satisfies these hypotheses in Π/S. Hence, a second application of Lipton's Theorem shows that $(\Pi/S)/S'$ and Π satisfy the same partial correctness and deadlock-freedom properties. Generalizing to an arbitrary number of reductions is obvious.

3 A General Reduction Theorem

We begin by defining the concepts needed to formalize the notion of reduction. Then, in Section 3.2, we state a generalization of Lipton's Theorem; its proof is in the appendix. We derive Doeppner's result as a corollary, and use it to prove Lipton's Theorem. The section closes with an example of the use of our theorem.

3.1 Definitions

3.1.1 Programs

Thus far, we have viewed a program Π as a set of states and a set of actions. (Recall that an [atomic] action is a set of pairs of states.) However, what matters for safety properties is not the set of actions, but the program's *next-state* relation, which is the union of all the program's actions. For example, replacing the single program action

$$\langle\, x := |x| + 1 \,\rangle$$

by the pair of actions

$$\textbf{if}\ \langle\, x \geq 0 \rightarrow x := x + 1 \,\rangle \,\square\, \langle\, x < 0 \rightarrow x := -x + 1 \,\rangle\ \textbf{fi}$$

yields an equivalent program.

We therefore formally define a program Π to consist of a set of states and a single action $\widehat{\Pi}$, where $\widehat{\Pi}$ is the next-state relation. (The next-state relation, being the union of actions, is itself an action.) Observe that, although the specification of a program usually describes its possible starting states, we do not include any special starting or terminating states in our formal definition—they are irrelevant to our results.

3.1.2 Histories

A ***history*** of Π is a finite, nonempty sequence $t_0, \ldots, t_n$ of states such that $t_{i-1} \xrightarrow{\widehat{\Pi}} t_i$, for $0 < i \leq n$. This history represents a partial execution (possibly complete) of Π, starting in state t_0 and reaching state t_n. Only such finite partial executions need be considered when proving safety properties, even of nonterminating programs, since a safety property is, by definition, one that is satisfied by an infinite execution iff it is satisfied by every finite prefix [1].

3.1.3 Commutativity

Recall that an action ρ right commutes with an action λ (and λ left commutes with ρ) iff $t \xrightarrow{\rho\lambda} u$ implies $t \xrightarrow{\lambda\rho} u$ for all states t and u. It follows ¿from this definition that, if ρ equals the union of actions ρ_i and λ equals the union of actions λ_j, then ρ right commutes with λ if every ρ_i right commutes with every λ_j.

If there are no states s, t, and u such that $s \xrightarrow{\rho} t \xrightarrow{\alpha} u$, so α cannot be executed immediately after ρ, then ρ right commutes with α. Hence, if ρ is an action in a process of a concurrent program, then ρ right commutes with every action in that process, except the action immediately following it. Hypothesis 1 of Lipton's theorem is therefore equivalent to the hypothesis that $S_1, \ldots, S_{i-1}$ right commute with every program action not in S. Similarly, an action left commutes with every action in the same process except the action immediately preceding it.

For any action α, we define $\stackrel{\alpha}{\Longrightarrow}$ to be the reflexive, transitive closure of $\xrightarrow{\alpha}$. Thus, $t \stackrel{\alpha}{\Longrightarrow} u$ iff $t = u$ or there exists a state v such that $t \xrightarrow{\alpha} v \stackrel{\alpha}{\Longrightarrow} u$. In other words, $t \stackrel{\alpha}{\Longrightarrow} u$ iff it is possible to go from state t to state u by "executing" action α zero or more times. We adopt the usual convention of writing $t \stackrel{\alpha}{\Longrightarrow} v \stackrel{\beta}{\Longrightarrow} u$ to denote that $t \stackrel{\alpha}{\Longrightarrow} v$ and $v \stackrel{\beta}{\Longrightarrow} u$ hold.

3.1.4 Predicates and Safety Properties

A *predicate* is a Boolean-valued function on the set of states. The value $Q(t)$ of predicate Q on state t is written $t \models Q$. An action α is defined to leave predicate Q *invariant* iff $t \models Q$ implies $u \models Q$ whenever $t \xrightarrow{\alpha} u$. It follows from this definition that, if α equals the union of actions α_i, then α leaves Q invariant iff every α_i does. Note that if $t \models Q$ implies that α cannot be executed in state t, so there is no state u such that $t \xrightarrow{\alpha} u$, then α trivially leaves Q invariant. Thus, if U is the predicate asserting that α is enabled, then α leaves $\neg U$ invariant.

If *Init* and Q are predicates, then a program Π satisfies the temporal logic formula $Init \Rightarrow \Box Q$ iff the following holds: for any history $t_0, \ldots, t_n$ of Π, if $t_0 \models Init$ then $t_i \models Q$, for $0 \leq i \leq n$. This property is equivalent to

$$\text{For all states } t \text{ and } u\text{: if } t \stackrel{\widehat{\Pi}}{\Longrightarrow} u \text{ and } t \models Init, \text{ then } u \models Q.$$

Properties of the form $Init \Rightarrow \Box Q$ are proved with the Owicki-Gries method [10] and similar assertional methods [2,6]. Moreover, by adding auxiliary variables to the program, any safety property can be expressed in this form.

3.1.5 Operations

The notion of a statement is meaningful only in the context of a programming language. To make our results independent of any language, we will define reduction with respect to *operations* rather than statements. The intuitive view is that an operation S consists of a collection of related actions from a single process. Actions are "related" iff, from the time the first action of S is executed until the entire operation completes, the process can execute actions only from S. Executing the first action of S moves control inside S, and executing the last action moves control outside S. Only actions of S can move control inside or outside of S.

Formally, an operation S of program Π consists of a subset $\widehat{S}$ of the next-state relation $\widehat{\Pi}$ together with a predicate $\mathcal{E}(S)$ (where $\mathcal{E}$ stands for *external*), such that $\widehat{\Pi} - \widehat{S}$ leaves both $\mathcal{E}(S)$ and $\neg\mathcal{E}(S)$ invariant. Being subsets of $\widehat{\Pi}$, an action, $\widehat{S}$ and $\widehat{\Pi} - \widehat{S}$ are themselves actions. This formal definition corresponds to the intuitive view above, where $\widehat{S}$ is the union of the actions constituting S, and $\mathcal{E}(S)$ is the predicate asserting that control is outside S.[1]

We now define what it means for an operation to be atomic. We could define A to be atomic iff $\mathcal{E}(A)$ holds in all states. However, we want Π and Π/S to satisfy the same properties, so we want them to have the same set of states; this means that Π/S may contain states in which $\mathcal{E}(\langle S\rangle)$ is false even though it has $\langle S\rangle$ as an atomic action. Therefore, we adopt the more general definition that an operation A of program Π is atomic iff $\mathcal{E}(A)$ is left invariant by $\widehat{\Pi}$. Consequently, if A is atomic, then control will remain outside A throughout any history that starts in a state with control outside A.

Observe that the concept of a process is not used in our formal definition of an operation, and nothing prevents actions of different processes from being part of a single operation. For example, a matching pair of communication statements in a CSP program can be represented by a single atomic operation [8].

[1] In the notation of [5], $\mathcal{E}(S) = at(S) \vee \neg in(S)$.

3.1.6 Sequential Composition

Our reduction theorem involves the sequential composition $T; U$ of operations T and U. Composition is usually defined for statements in a programming language. A precise definition for sequential composition of operations is complicated. However, the composition $T; U$ has the expected meaning if (i) control cannot be inside both T and U, and (ii) any execution of $T; U$ consists of a (possibly null) sequence of executions of T followed by a (possibly null) sequence of executions of U. For example, in the statement

if b **then** $T; U_1$
    else U_2
fi

the **then** and **else** clauses together define a single operation $T; U$, where the operation U is defined by $\widehat{U} = \widehat{U_1} \cup \widehat{U_2}$ and $\mathcal{E}(U) = \mathcal{E}(U_1) \wedge \mathcal{E}(U_2)$. By our definition of atomicity, if each U_i is atomic, then U is atomic.

For a general definition of the sequential composition of operations, we must use $\mathcal{E}(T)$, $\mathcal{E}(U)$, $\widehat{T}$, and $\widehat{U}$ to characterize when operation $T; U$ is defined and, when it is defined, what $\widehat{T; U}$ and $\mathcal{E}(T; U)$ are. Such a definition is complicated; the only simple part is that when $T; U$ is defined, $\widehat{T; U}$ equals $\widehat{T} \cup \widehat{U}$. Therefore, instead of giving a formal definition, we just list in the appendix properties of sequential composition that we require.

If T is null, meaning that $\widehat{T}$ is the empty set and $\mathcal{E}(T)$ is identically true, then $T; U$ equals U. Similarly, if U is null, then $T; U$ equals T.

3.1.7 Possible Termination

Hypothesis 3 of Lipton's Theorem asserts that it is possible for S to terminate from any state in which control is inside S. Control being inside S means that $\neg\mathcal{E}(S)$ holds. Termination of S means reaching a state in which $\mathcal{E}(S)$ holds. Thus, Lipton's hypothesis 3 asserts that, for every state t, if $t \models \neg\mathcal{E}(S)$ then there exists a state u such that $t \overset{\widehat{S}}{\Longrightarrow} u$ and $u \models \mathcal{E}(S)$.

3.2 The Reduction Theorem and Corollaries

3.2.1 Reduction

The purpose of our reduction theorem is to justify pretending that an operation is atomic. To define what this pretense means, we first define the operation $\langle S \rangle$ for an arbitrary operation S in a program Π. This requires defining action $\widehat{\langle S\rangle}$ and predicate $\mathcal{E}(\langle S\rangle)$. We define $\mathcal{E}(\langle S\rangle)$ to equal $\mathcal{E}(S)$. Our definition of $\widehat{\langle S\rangle}$ should assert that $t \xrightarrow{\widehat{\langle S\rangle}} u$ iff a complete execution of S can take state t to state u. A "complete execution" is one that starts with control outside S and ends as soon as control leaves S. We define $\widehat{\langle S\rangle}$ to consist of all pairs (t, u) such that $t \models \mathcal{E}(S)$, $u \models \mathcal{E}(S)$, and there exist states $t_0, \ldots, t_n$, with $0 < n$, such that

$$t = t_0 \xrightarrow{\widehat{S}} t_1 \xrightarrow{\widehat{S}} \ldots \xrightarrow{\widehat{S}} t_{n-1} \xrightarrow{\widehat{S}} t_n = u$$

and $t_i \models \neg\mathcal{E}(S)$ for $0 < i < n$.

For any action α, define $t \underset{S}{\overset{\alpha}{\Longrightarrow}} u$ to mean that there exist states $t_0, \ldots, t_n$, with $0 \leq n$, such that

$$t = t_0 \xrightarrow{\alpha} t_1 \xrightarrow{\alpha} \ldots \xrightarrow{\alpha} t_{n-1} \xrightarrow{\alpha} t_n = u$$

and $t_i \models \neg\mathcal{E}(S)$ for $0 < i < n$. Then, $t \underset{S}{\overset{\alpha}{\Longrightarrow}} u$ implies $t \overset{\alpha}{\Longrightarrow} s$. If $u \models \neg\mathcal{E}(S)$, then $t \underset{S}{\overset{\alpha}{\Longrightarrow}} u$ and $u \underset{S}{\overset{\alpha}{\Longrightarrow}} v$ imply $t \underset{S}{\overset{\alpha}{\Longrightarrow}} v$.

To see the relation between the two actions $\widehat{S}$ and $\widehat{\langle S\rangle}$, suppose $t \models \mathcal{E}(S)$ and $u \models \mathcal{E}(S)$. The definition of $\widehat{\langle S\rangle}$ implies that $t \underset{S}{\overset{\widehat{S}}{\Longrightarrow}} u$ iff $t \xrightarrow{\widehat{\langle S\rangle}} u$ or $t = u$. This in turn implies that $t \overset{\widehat{S}}{\Longrightarrow} u$ iff $t \overset{\widehat{\langle S\rangle}}{\Longrightarrow} u$.

We can now formally define program Π/S. We want Π/S to be the program obtained by replacing S by an atomic action, so Π/S is defined to have the same set of states as Π and to have its next-state relation $\widehat{\Pi/S}$ equal to $(\widehat{\Pi} - \widehat{S}) \cup \widehat{\langle S\rangle}$. To show that $\langle S\rangle$ is an atomic operation of Π/S, we must show that $\widehat{\Pi/S}$ leaves $\mathcal{E}(\langle S\rangle)$ invariant. By definition of what it means for S to be an operation of Π, action $\widehat{\Pi} - \widehat{S}$ leaves $\mathcal{E}(S)$ invariant. By definition of $\langle S\rangle$, action $\widehat{\langle S\rangle}$ leaves $\mathcal{E}(S)$ invariant. Therefore, $(\widehat{\Pi} - \widehat{S}) \cup \widehat{\langle S\rangle}$, which equals $\widehat{\Pi/S}$, leaves invariant $\mathcal{E}(S)$, which equals $\mathcal{E}(\langle S\rangle)$.

The useful part of the reduction theorem states that, for certain operations S, if a safety property is satisfied by Π/S then it is satisfied by Π. The converse, that a safety property is satisfied by Π/S if it is satisfied by Π, is true for any S.

Lemma 1 *If Init* $\Rightarrow \Box Q$ *is satisfied by program* Π *then it is satisfied by program* Π/S.

Proof of Lemma

1. For any states t and u, if $t \overset{\widehat{\Pi}}{\Longrightarrow} u$ and $t \models \mathit{Init}$, then $u \models Q$.
   Proof: By the hypothesis that Π satisfies $\mathit{Init} \Rightarrow \Box Q$.
2. For any states t and u, if $t \overset{\widehat{\Pi/S}}{\Longrightarrow} u$ then $t \overset{\widehat{\Pi}}{\Longrightarrow} u$.
   Proof: By definition of reduction, since $\widehat{\Pi/S} - \widehat{\langle S \rangle} \subseteq \widehat{\Pi}$ and $v \overset{\langle S \rangle}{\longrightarrow} w$ implies $v \overset{\widehat{S}}{\Longrightarrow} w$.
3. For any states t and u, if $t \overset{\widehat{\Pi/S}}{\Longrightarrow} u$ and $t \models \mathit{Init}$ then $u \models Q$.
   Proof: By 1 and 2.
4. Program Π/S satisfies $\mathit{Init} \Rightarrow \Box Q$.
   Proof: By 3 and the definition of what it means for Π/S to satisfy $\mathit{Init} \Rightarrow \Box Q$.

End Proof of Lemma

3.2.2 The Reduction Theorem and a Corollary

We now state our reduction theorem, which is proved in the appendix, and derive a corollary.

Reduction Theorem *Let Π be a program, Init and Q be predicates, and S be an operation of Π having the form $R; \langle A \rangle; L$, where*

0. *Init implies $\mathcal{E}(S)$.*

1. (a) *Action $\widehat{R}$ right commutes with action $\widehat{\Pi} - \widehat{S}$.*

   (b) *For all states t and u: if $t \overset{\widehat{R}}{\Longrightarrow} u$ and $t \models (Q \wedge \mathcal{E}(S))$ then $u \models (Q \vee \mathcal{E}(S))$.*

2. (a) *Action $\widehat{L}$ left commutes with action $\widehat{\Pi} - \widehat{S}$.*

   (b) *For all states t and u: if $t \overset{\widehat{L}}{\Longrightarrow} u$ and $t \models (\neg Q \wedge \neg\mathcal{E}(S))$ then $u \models (\neg Q \vee \neg\mathcal{E}(S))$.*

3. *For all states t: if $t \models (\neg Q \wedge \mathcal{E}(R; \langle A \rangle) \wedge \neg\mathcal{E}(S))$ then there exists a state u such that $t \overset{\widehat{L}}{\Longrightarrow} u$ and $u \models \mathcal{E}(S)$.*[2]

Then, Init $\Rightarrow \Box Q$ is satisfied by Π iff it is satisfied by Π/S.

[2] $\mathcal{E}(R; \langle A \rangle) \wedge \neg\mathcal{E}(S)$ asserts that control is either inside L or at its entry point.

Observe that hypothesis 1(b) holds if R leaves Q invariant, and hypothesis 2(b) holds if L leaves $\neg Q$ invariant. Thus, both of these hypotheses hold if R and L do not change any part of the state on which Q depends.

The conclusion of our reduction theorem asserts that if Q holds throughout the execution of Π/S then it holds throughout the execution of Π. Weaker hypotheses lead to the weaker conclusion that, in the execution of Π, predicate Q holds only when control is external to S, giving a result obtained by Doeppner [4].

Corollary (Doeppner) *Let Π be a program and S have the form $R; \langle A \rangle; L$, where*

0. *Init implies $\mathcal{E}(S)$.*

1. *Action $\widehat{R}$ right commutes with action $\widehat{\Pi} - \widehat{S}$.*

2. *Action $\widehat{L}$ left commutes with action $\widehat{\Pi} - \widehat{S}$.*

Then, $Init \Rightarrow \Box(Q \vee \neg\mathcal{E}(S))$ is satisfied by Π iff $Init \Rightarrow \Box Q$ is satisfied by Π/S.

Proof of Corollary

1. $Init \Rightarrow \Box(Q \vee \neg\mathcal{E}(S))$ is satisfied by Π iff it is satisfied by Π/S.
   Proof: Apply the Reduction Theorem with $Q \vee \neg\mathcal{E}(S)$ substituted for Q. Hypotheses 0, 1(a), and 2(a) of the theorem follow from hypotheses 0–2 of the corollary. Hypothesis 1(b) of the theorem holds trivially because $(Q \vee \neg\mathcal{E}(S)) \vee \mathcal{E}(S)$ is identically true. Hypothesis 2(b) of the theorem holds vacuously because $\neg(Q \vee \neg\mathcal{E}(S)) \wedge \neg\mathcal{E}(S)$ is identically false. Hypothesis 3 also holds vacuously because $\neg(Q \vee \neg\mathcal{E}(S)) \wedge \mathcal{E}(R; \langle A \rangle) \wedge \neg\mathcal{E}(S)$ is identically false.
2. Π/S satisfies $Init \Rightarrow \Box\mathcal{E}(S)$.
   Proof: By hypothesis 0, since $\widehat{\Pi/S}$ leaves $\mathcal{E}(S)$, which equals $\mathcal{E}(\langle S \rangle)$, invariant.
3. Π/S satisfies $Init \Rightarrow \Box(Q \vee \neg\mathcal{E}(S))$ iff it satisfies $Init \Rightarrow \Box Q$.
   Proof: Follows from 2 and the definition of what it means for Π/S to satisfy a formula of the form $Init \Rightarrow \Box P$.

End Proof of Corollary

The corollary provides a correct statement of the single-action rule. The incorrect version of the rule asserts that if the reduced program satisfies a property then the original program does. The correct version asserts that if the reduced program satisfies a property $Init \Rightarrow \Box Q$, then the original program satisfies the related property $Init \Rightarrow \Box(Q \vee \neg\mathcal{E}(S))$. Only if $\neg\mathcal{E}(S)$ implies Q does the original program satisfy the same property as the reduced program.

3.2.3 Deriving Lipton's Theorem

We now derive Lipton's Theorem from the corollary. Lipton's Theorem concerns partial correctness and deadlock freedom properties. We consider each of them separately.

The partial correctness property $\{Pre\}\Pi\{Post\}$ can be expressed in the form $Init \Rightarrow \Box Q$ by letting $Init$ be the predicate asserting that control is at the beginning of Π and Pre holds, and letting Q be $Term \Rightarrow Post$, where $Term$ is the predicate asserting that Π has terminated—that is, $Term$ asserts that control is at the end of the program. Since control at the end of Π implies that $\mathcal{E}(S)$ holds, $\neg\mathcal{E}(S)$ implies Q, so $Q \vee \neg\mathcal{E}(S)$ is equivalent to Q. Hence, the corollary implies that, under the hypotheses of Lipton's Theorem, Π satisfies $\{Pre\}\Pi\{Post\}$ iff Π/S does. This proves Lipton's Theorem for partial correctness. Moreover, we have strengthened this part of Lipton's Theorem by eliminating hypothesis 3. In so doing, we have shown that the single-action rule is valid for partial correctness properties.

We next show that the deadlock-freedom part of Lipton's Theorem follows ¿from the corollary. A program is deadlocked iff it has not terminated and no program action is enabled. Program Π has terminated iff program Π/S has. Thus, we need show only that an action of Π is always enabled iff an action of Π/S is always enabled. Let $Init$ be the predicate asserting that control is at the beginning of Π and let DF_Π be the predicate asserting that some action of Π is enabled. Similarly, define $DF_{\Pi/S}$ to assert that some action of Π/S is enabled. The conclusion of Lipton's Theorem states, in our notation, that Π satisfies $Init \Rightarrow \Box DF_\Pi$ iff Π/S satisfies $Init \Rightarrow \Box DF_{\Pi/S}$. We use the corollary to show that this conclusion is implied by the hypotheses of Lipton's Theorem.

1. Π satisfies $Init \Rightarrow \Box(DF_{\Pi/S} \vee \neg\mathcal{E}(S))$ iff Π/S satisfies $Init \Rightarrow \Box DF_{\Pi/S}$.
   Proof: Apply the Corollary with $DF_{\Pi/S}$ substituted for Q.
2. $DF_{\Pi/S} \vee \neg\mathcal{E}(S)$ implies DF_Π.
   2.1. $DF_{\Pi/S}$ implies DF_Π.
   Proof: By definition of Π/S, if an action of Π/S is enabled then an action of Π must be enabled.

2.2. $\neg\mathcal{E}(S)$ implies DF_Π.

Proof: By hypothesis 3 of Lipton's Theorem.

3. DF_Π implies $DF_{\Pi/S} \vee \neg\mathcal{E}(S)$.

Proof: It suffices to prove that DF_Π and $\mathcal{E}(S)$ imply $DF_{\Pi/S}$. For this, it suffices to prove that for any state t, if there exists a state u such that $t \models \mathcal{E}(S)$ and $t \xrightarrow{\widehat{\Pi}} u$, then there exists a state v such that $t \xrightarrow{\widehat{\Pi/S}} v$.

Since $t \xrightarrow{\widehat{\Pi}} u$, either $t \xrightarrow{\widehat{\Pi}-\widehat{S}} u$ or else $t \xrightarrow{\widehat{S}} u$. If $t \xrightarrow{\widehat{\Pi}-\widehat{S}} u$, then we can let v equal u. Assume that $t \xrightarrow{\widehat{S}} u$. If $u \models \neg\mathcal{E}(S)$, then hypothesis 3 of Lipton's Theorem implies that there exists a state v such that $v \models \mathcal{E}(S)$ and $u \xRightarrow{\widehat{S}} v$. If $u \models \mathcal{E}(S)$, then let v equal u. In either case, $t \xRightarrow{\widehat{S}} v$, $t \models \mathcal{E}(S)$, and $v \models \mathcal{E}(S)$, so $t \xrightarrow{\langle S \rangle} v$.

4. Π satisfies $Init \Rightarrow \Box DF_\Pi$ iff Π/S satisfies $Init \Rightarrow \Box DF_{\Pi/S}$.

Proof: By 1, since 2 and 3 imply $DF_\Pi = DF_{\Pi/S} \vee \neg\mathcal{E}(S)$.

The single-action rule is not valid for deadlock freedom. For example, let Π be the single-process program

$$\langle\, x := 0 \text{ or } 1 \,\rangle;\ \langle\, \textbf{await } x = 0 \,\rangle$$

where the assignment nondeterministically sets x to 0 or 1, and the **await** delays forever if $x = 1$. Since every variable is local, a naive single-action rule would assert that this program is equivalent to

$$\langle\, x := 0 \text{ or } 1;\ \textbf{await } x = 0 \,\rangle$$

which, by our definition of $\langle S \rangle$, is equivalent to

$$\langle\, x := 0 \,\rangle$$

The reduced program is deadlock free, but the original program is not—it deadlocks if the assignment statement sets x to 1.

One might be able to find an alternate definition of $\langle S \rangle$ that makes the single-action rule valid for deadlock freedom. However, we believe that such a definition would be unnatural, and unlikely to be of any practical use.

3.3 An Example

Program Π_1 of Figure 1 is a two-process concurrent program, where *head* and *tail* are the usual operators on sequences, and $\circ$ denotes concatenation.

Program Π_1

```
variables
  inp    : infinite sequence of value;
  out    : sequence of value;
  buf    : array [0...N - 1] of value;
  x, y   : value;
  fp, fc : Natural;
  cobegin
    Producer: loop
                D_p: ⟨ x,inp := head(inp), tail(inp) ⟩;
                A_p: ⟨ await (fp - fc) < N ⟩;
                B_p: ⟨ buf[fp mod N] := x ⟩;
                C_p: ⟨ fp := fp + 1 ⟩
              end loop
  □
    Consumer: loop
                A_c: ⟨ await (fp - fc) > 0 ⟩;
                B_c: ⟨ y := buf[fc mod N] ⟩;
                C_c: ⟨ fc := fc + 1 ⟩;
                D_c: ⟨ out := out ∘ y ⟩
              end loop
  coend
```

Figure 1: A simple producer/consumer program.

Using a bounded buffer, a producer process communicates an infinite sequence of values to a consumer process. The safety property of interest is that the sequence of values *out* received by the consumer is a prefix of the initial value of the sequence *inp*. This property is formulated as $Init \Rightarrow \Box Q$, where

- *Init* asserts that *buf* is empty, *inp* has some initial value inp_{init}, $fp = fc = 0$, and $at(D_p)$ and $at(A_c)$ hold, where $at(\xi)$ is a predicate that is true iff control is at action ξ.

- Q asserts that *out* is an initial prefix of inp_{init}.

To prove that Π_1 satisfies this property, the Reduction Theorem is applied twice. First, Program Π_1 is reduced by $A_p; B_p; C_p$, resulting in a program where the producer has only two actions—D_p and $\langle A_p; B_p; C_p\rangle$. Then, that program is reduced by $A_c; B_c; C_c$, resulting in a final program having just four atomic actions. As we observed at the end of Section 2, these two reductions can be done at once. This is because a consumer action left (right) commutes with each of the actions A_p, B_p, and C_p iff it left (right) commutes with the single action $\langle A_p; B_p; C_p\rangle$.

For the first reduction, the theorem is applied with A_p for R, B_p for $\langle A\rangle$, and C_p for L. We now show that the four hypotheses of the theorem are satisfied.

Hypothesis 0. *Init* implies $\mathcal{E}(A_p; B_p; C_p)$.

> *Proof*: This follows from the definition of *Init* and $\mathcal{E}$, because *Init* implies $at(D_p)$, and $at(D_p)$ implies that control is external to $A_p; B_p; C_p$.

Hypothesis 1. (a) Action $\widehat{R}$ right commutes with action $\widehat{\Pi} - \widehat{S}$, where S is $A_p; B_p; C_p$.

> (b) For all states t and u, if $t \xrightarrow{\widehat{A_p}} u$ and $t \models (Q \wedge \mathcal{E}(S))$ then $u \models (Q \vee \mathcal{E}(S))$.
>
> 1. $\widehat{A_p}$ right commutes with $\widehat{D_p}$.
>
>   *Proof*: $\widehat{D_p}$ cannot be executed immediately after $\widehat{A_p}$.
>
> 2. $\widehat{A_p}$ right commutes with $\widehat{A_c}$, $\widehat{B_c}$, and $\widehat{D_c}$.
>
>   *Proof*: Actions $\widehat{A_p}$ and $\widehat{A_c}$ commute because neither modifies any variable accessed by the other, and $\widehat{A_p}$ commutes with $\widehat{B_c}$ and with $\widehat{D_c}$ for the same reason.

3. $\widehat{A_p}$ right commutes with $\widehat{C_c}$.

3.1. If $s \xrightarrow{\widehat{A_p}\widehat{C_c}} t$ and $s \xrightarrow{\widehat{C_c}\widehat{A_p}} t'$, then $t = t'$.

Proof: From the definitions of $\widehat{A_c}$ and $\widehat{C_c}$.

3.2. If it is possible to execute first $\widehat{A_p}$ then $\widehat{C_c}$ on a state s, then it is also possible to execute first $\widehat{C_c}$ then $\widehat{A_p}$ on s.

Proof: It is possible to execute $\widehat{A_p}$ then $\widehat{C_c}$ on s iff

$$s \models at(A_p) \wedge at(C_c) \wedge (fp - fc < N) \tag{3}$$

It is possible to execute $\widehat{C_c}$ then $\widehat{A_p}$ on s iff

$$s \models at(A_p) \wedge at(C_c) \wedge (fp - (fc + 1) < N) \tag{4}$$

Obviously, (3) implies (4).

3.3. If $s \xrightarrow{\widehat{A_p}\widehat{C_c}} t$ then $s \xrightarrow{\widehat{C_c}\widehat{A_p}} t$

Proof: By 3.1 and 3.2.

4. Hypothesis 1(a) holds.

Proof: By 1, 2, and 3, since $\widehat{\Pi} - \widehat{S}$ equals the union of $\widehat{D_p}$, $\widehat{A_c}$, $\widehat{B_c}$, $\widehat{C_c}$, and $\widehat{D_c}$.

5. Hypothesis 1(b) holds.

Proof: Action $\widehat{A_p}$ does not modify any part of the state on which Q depends, so it leaves Q invariant.

Hypothesis 2. (a) Action $\widehat{C_p}$ left commutes with action $\widehat{\Pi} - \widehat{S}$.

(b) For all states t and u, if $t \xRightarrow{\widehat{C_p}} u$ and $t \models (\neg Q \wedge \neg \mathcal{E}(S))$ then $u \models (\neg Q \vee \neg \mathcal{E}(S))$.

Proof: The proof of this is similar to the proof of hypothesis 1. The key step in the proof that C_p left commutes with A_c is the observation that (i) it is possible to execute A_c then C_p on a state s iff $s \models (at(A_c) \wedge at(C_p) \wedge (fp - fc > 0))$, and (ii) it is possible to execute C_p then A_c on s iff $s \models (at(A_c) \wedge at(C_p) \wedge (fp - fc \geq 0))$. Hypothesis 2(b) holds because action C_p does not change any part of the state on which Q depends, so it leaves $\neg Q$ invariant.

Hypothesis 3. For all states t: if $t \models (\neg Q \wedge at(C_p))$ then C_p can terminate from t.

Proof: C_p can terminate from any state t for which $t \models (at(C_p))$.

The justification of the second reduction is similar to that of the first with p and c subscripts interchanged. We must prove that $\widehat{A_c}$ left commutes and $\widehat{C_c}$ right commutes with the four actions $\widehat{A_p}$, $\widehat{B_p}$, $\widehat{C_p}$, and $\widehat{D_p}$. (Recall that this implies that they left and right commute with $\widehat{\langle A_p; B_p; C_p \rangle}$.) Proving the symmetric versions of statements 0–3 in the proof of the first reduction allows our theorem to be applied to the second reduction. We omit the proofs. (Note that the proofs of the commutativity relations between $\widehat{A_c}$ and $\widehat{C_p}$, and between $\widehat{C_c}$ and $\widehat{A_p}$ appeared in the proof of the first reduction.)

4 Constraints

We can replace the unbounded integer variables *fp* and *fc* of Program Π_1 by integers modulo $2N$, to obtain producer/consumer program Π_2 of Figure 2. Program Π_2 can be viewed as an implementation of Π_1 in which the "left-most bits" of *fp* and *fc* have been eliminated. We would, therefore, expect to be able to reduce Π_2 to a program with only four atomic actions, just as we reduced Π_1. Unfortunately, we cannot. The action pairs $\widehat{A_p}$, $\widehat{C_c}$ and $\widehat{A_c}$, $\widehat{C_p}$ of Π_2 do not satisfy the required commutativity relations. For example, if t is a state in which $fp = fc$, then there are states u and v such that $t \xrightarrow{\widehat{A_p}} u \xrightarrow{\widehat{C_c}} v$, but no state u' such that $t \xrightarrow{\widehat{C_c}} u' \xrightarrow{\widehat{A_p}} v$ because $-1 \bmod 2N$ equals $2N - 1$, which is greater than or equal to N. (Executing $\widehat{C_c}$ when $fp = fc$ disables $\widehat{A_p}$.) Thus, $\widehat{A_p}$ does not right commute with $\widehat{C_c}$.

Program Π_2 admits "irreducible" histories—ones that are not equivalent to any of the reduced program's histories. However, these irreducible histories are irrelevant because they cannot arise when Π_2 is started in a "proper" initial state. The property we want to prove is $Init \Rightarrow \Box Q$, which asserts that Q is always true for any execution started in a state satisfying the predicate *Init*, and it turns out that there is no irreducible history beginning with a state that satisfies *Init*. For example, histories containing a state in which $fp = fc$ and both $\widehat{A_p}$ and $\widehat{C_c}$ are enabled, so $\widehat{A_p}$ does not right commute with $\widehat{C_c}$, are irrelevant because such a state cannot be reached when Program Π_2 is started with *Init* true.

We will dispense with these irrelevant histories by modifying Π_2 to eliminate them.[3] We constrain the program by a predicate I to eliminate histories

[3] We could define these histories out of existence by including the initial state in the formal definition of a program, but this would complicate our definitions without making it any easier to actually prove properties of programs.

Program Π_2

variables

$$
\begin{array}{lll}
\textit{inp} & : & \textbf{infinite sequence of } \textit{value}; \\
\textit{out} & : & \textbf{sequence of } \textit{value}; \\
\textit{buf} & : & \textbf{array } [0 \ldots N-1] \textbf{ of } \textit{value}; \\
x,\, y & : & \textit{value}; \\
\textit{fp},\, \textit{fc} & : & \{0 \ldots 2N-1\};
\end{array}
$$

cobegin

$$
\begin{array}{ll}
\text{Producer:} & \textbf{loop} \\
 & \quad D_p\text{: } \langle\, x, \textit{inp} := \textit{head}(\textit{inp}), \textit{tail}(\textit{inp})\,\rangle; \\
 & \quad A_p\text{: } \langle\, \textbf{await } (\textit{fp} - \textit{fc}) \bmod 2N < N\,\rangle; \\
 & \quad B_p\text{: } \langle\, \textit{buf}[\textit{fp} \bmod N] := x\,\rangle; \\
 & \quad C_p\text{: } \langle\, \textit{fp} := \textit{fp} + 1 \bmod 2N\,\rangle \\
 & \textbf{end loop}
\end{array}
$$

$\square$

$$
\begin{array}{ll}
\text{Consumer:} & \textbf{loop} \\
 & \quad A_c\text{: } \langle\, \textbf{await } (\textit{fp} - \textit{fc}) \bmod 2N > 0\,\rangle; \\
 & \quad B_c\text{: } \langle\, y := \textit{buf}[\textit{fc} \bmod N]\,\rangle; \\
 & \quad C_c\text{: } \langle\, \textit{fc} := \textit{fc} + 1 \bmod 2N\,\rangle; \\
 & \quad D_c\text{: } \langle\, \textit{out} := \textit{out} \circ y\,\rangle \\
 & \textbf{end loop}
\end{array}
$$

coend

Figure 2: Another simple producer/consumer program.

in which I becomes false [7]. If the original program satisfies $\mathit{Init} \Rightarrow \Box I$, then only irrelevant histories are eliminated.

For an action α and a predicate I, define $\alpha|_I$ (read α *constrained by* I) to be the action $\{(s,t) \in \alpha : (s \models I) \wedge (t \models I)\}$. Thus, $s \xrightarrow{\alpha|_I} t$ iff $s \xrightarrow{\alpha} t$ and I holds in states s and t. For a program Π we define $\Pi|_I$ to be the program whose states are the states of Π that satisfy I, and whose next-state relation is $\widehat{\Pi}|_I$. If S is an operation of Π, then $S|_I$ is the operation of $\Pi|_I$ such that $\widehat{S|_I}$ equals $\widehat{S}|_I$ and $\mathcal{E}(S|_I)$ equals $\mathcal{E}(S)$ with its domain restricted to the states of $\Pi|_I$.

The next-state relation $\widehat{\Pi|_I}$ is enabled only in states satisfying I, and $\widehat{\Pi|_I}$ can produce only states satisfying I. The histories of $\Pi|_I$ consist of the histories of Π in which all states satisfy I. This implies that every history of $\Pi|_I$ is a history of Π.

Suppose that $\mathit{Init} \Rightarrow \Box I$ holds for a program Π. Then, I is true for all states in any history of Π beginning in a state with *Init* true. Therefore, any history of Π beginning with *Init* true is also a history of $\Pi|_I$. If Π satisfies $\mathit{Init} \Rightarrow \Box I$, then Π satisfies $\mathit{Init} \Rightarrow \Box Q$ iff $\Pi|_I$ does. The property $\mathit{Init} \Rightarrow \Box I$ can be proved by ordinary assertional methods. Usually, I is an invariant of Π.

To define the predicate I for Π_2, we first define a function Ψ_p on the set of program states:

$$\Psi_p \;=\; \begin{cases} 1 & \text{if } at(B_p) \vee at(C_p) \\ 0 & \text{otherwise} \end{cases}$$

We define Ψ_c similarly, replacing p by c. The predicate I is defined to equal

$$\Psi_c \leq (\mathit{fp} - \mathit{fc}) \bmod 2N \leq N - \Psi_p$$

That I is an invariant of Π can be established in the usual way. It is also easy to check that *Init* implies I. Therefore, to prove that $\mathit{Init} \Rightarrow \Box Q$ is satisfied by Π_2, we need to show only that it is satisfied by $\Pi_2|_I$.

We can now apply our Reduction Theorem to $\Pi_2|_I$, reducing it first by $A_p|_I; B_p|_I; C_p|_I$ and then by $A_c|_I; B_c|_I; C_c|_I$. The proof is almost identical to that for Π_1 given above. The major difference is in the proof that $\widehat{A_p|_I}$ right commutes with $\widehat{C_c|_I}$. As in step 3.2 above, we must show that if it is possible to execute $\widehat{A_p|_I}$ followed by $\widehat{C_c|_I}$ from a state t, then it is also possible to execute $\widehat{C_c|_I}$ followed by $\widehat{A_p|_I}$ from t. It is possible to execute

$\widehat{A_p|_I}$ followed by $\widehat{C_c|_I}$ from t iff

$$t \models I \wedge at(A_p) \wedge at(C_c) \wedge ((fp - fc) \bmod 2N < N) \tag{5}$$

and it is possible to execute $\widehat{C_c|_I}$ followed by $\widehat{A_p|_I}$ from t iff

$$t \models I \wedge at(A_p) \wedge at(C_c) \wedge ((fp - (fc + 1)) \bmod 2N < N) \tag{6}$$

Since $I \wedge at(A_p) \wedge at(C_c)$ implies that $1 \leq fp - fc \bmod 2N \leq N$, it follows that (5) implies (6).

5 Discussion

We have given a reduction theorem for proving that a safety property of the form $Init \Rightarrow \Box Q$ holds for a program Π if it holds for the coarser-grained program Π/S. In general, a reduction theorem allows one to conclude that Π satisfies a property $\mathcal{P}$ if Π/S satisfies a related property $\mathcal{P}'$. It is proved by showing that for any history Σ of Π, there is a corresponding history Σ' of Π/S such that Σ satisfies $\mathcal{P}$ if Σ' satisfies $\mathcal{P}'$. The history Σ' is derived from Σ by commuting actions and completing or eliminating any unfinished execution of S. Hypotheses about commutativity and the possible termination of L make it possible to derive Σ'. Additional hypotheses may be needed to guarantee that if Σ' satisfies $\mathcal{P}'$ then Σ satisfies $\mathcal{P}$. In our reduction theorem, these are hypotheses 1(b) and 2(b).

A reduction theorem is tailored to a particular class of properties. We chose the hypotheses of our reduction theorem to be as weak as possible for properties of the form $Init \Rightarrow \Box Q$. Lipton considered partial correctness and deadlock-freedom properties, and Doeppner considered properties closely related to partial correctness. We do not know of a similar reduction theorem for liveness properties. We do know that such a theorem would need different hypotheses. For example, the hypotheses of Lipton's Theorem are satisfied if S equals $P(sem); V(sem)$, in which case $\langle S \rangle$ leaves sem unchanged. Suppose a program Π contains a process that repeatedly executes S. Then Π/S might satisfy a progress property that is not satisfied by Π because the repeated decrementing and incrementing of sem prevents some other process from making progress. Thus, the hypotheses of Lipton's Theorem are not sufficient for deriving liveness properties.

Back [3] does give a reduction theorem for total correctness—the conjunction of partial correctness (a safety property) and termination (a liveness property). However, his hypotheses involve commutativity relations between actions outside S, so the theorem is not closely related to either our reduction theorem or Lipton's.

Appendix: Proof of the Reduction Theorem

Our proof relies on the following properties of sequential composition and atomic operations, where S equals $T; U$.

SC1. For any action α, if $v \underset{S}{\overset{\alpha}{\Longrightarrow}} w$, then there exists a state x such that $v \underset{T}{\overset{\alpha-\widehat{U}}{\Longrightarrow}} x \underset{U}{\overset{\alpha-\widehat{T}}{\Longrightarrow}} w$.
[When executing S: first, actions in T or not in S are executed until control exits T; then, actions in U or not in S are executed until control exits S.]

SC2. $\mathcal{E}(S)$ implies $\mathcal{E}(T) \wedge \mathcal{E}(U)$.
[If control is external to S, then it is external to its components T and U.]

SC3. $\neg\mathcal{E}(T) \wedge \neg\mathcal{E}(U)$ is identically false.
[Control cannot be internal to both T and U.]

SC4. $\neg\mathcal{E}(T)$ implies that $\widehat{U}$ is not enabled.
[U cannot be executed when control is internal to T.]

SC5. If U is an atomic operation and $v \xrightarrow{\widehat{U}} w$ then $w \models \mathcal{E}(S)$.
[When control exits U, control is external to S; and control exits an atomic action when it is executed.]

Lemma 2 (a) *Let α and ρ be actions such that ρ right commutes with $\alpha - \rho$. For states t and u, if $t \overset{\alpha}{\Longrightarrow} u$ then there exists a state v such that $t \overset{\alpha-\rho}{\Longrightarrow} v \overset{\rho}{\Longrightarrow} u$.*

(b) *Let α and λ be actions such that λ left commutes with $\alpha - \lambda$. For states t and u, if $t \overset{\alpha}{\Longrightarrow} u$ then there exists a state v such that $t \overset{\lambda}{\Longrightarrow} v \overset{\alpha-\lambda}{\Longrightarrow} u$.*

Proof of Lemma
We prove part (a); the proof of part (b) is similar. The hypothesis asserts that

$$t = t_0 \xrightarrow{\alpha} t_1 \xrightarrow{\alpha} \ldots \xrightarrow{\alpha} t_{n-1} \xrightarrow{\alpha} t_n = u \tag{7}$$

for some states t_i, with $0 \leq n$. If $w \xrightarrow{\alpha} x$, then either $w \xrightarrow{\rho} x$ or $w \xrightarrow{\alpha-\rho} x$. By the right-commutativity hypothesis, if $w \xrightarrow{\rho} x \xrightarrow{\alpha-\rho} y$, then there exists x' such that $w \xrightarrow{\alpha-\rho} x' \xrightarrow{\rho} y$. Thus, by repeatedly replacing $\xrightarrow{\rho} x \xrightarrow{\alpha-\rho}$ with

$\xrightarrow{\alpha-\rho} x' \xrightarrow{\rho}$, we can deduce from (7) the existence of k and of states t'_i such that

$$t = t'_0 \xrightarrow{\alpha-\rho} t'_1 \xrightarrow{\alpha-\rho} \ldots \xrightarrow{\alpha-\rho} t'_k \xrightarrow{\rho} \ldots \xrightarrow{\rho} t'_n = u$$

This implies $t \overset{\alpha-\rho}{\Longrightarrow} v \overset{\rho}{\Longrightarrow} u$, where $v = t'_k$.

End Proof of Lemma

Lemma 3 *Assume hypotheses* 0–3 *of the Reduction Theorem and the additional hypotheses that, for states t and u:*

4. $t \models \mathcal{E}(S)$
5. $u \models \mathcal{E}(S)$
6. $t \overset{\widehat{\Pi}}{\Longrightarrow} u$

Then $t \overset{\widehat{\Pi/S}}{\Longrightarrow} u$.

Proof of Lemma

We prove by induction on n that, for any states t and u, if there exist states $t_0, \ldots, t_n$ such that

$$t = t_0 \xrightarrow{\widehat{\Pi}} t_1 \xrightarrow{\widehat{\Pi}} \ldots \xrightarrow{\widehat{\Pi}} t_{n-1} \xrightarrow{\widehat{\Pi}} t_n = u \qquad (8)$$

then $t \overset{\widehat{\Pi/S}}{\Longrightarrow} u$. The base case $n = 0$ is trivial, since then $t = u$ and the relation $\overset{\widehat{\Pi/S}}{\Longrightarrow}$ is reflexive.

We now prove the induction step, assuming $n > 0$. Assume states t_0, $\ldots$, t_n satisfying (8) exist. The proof that $t \overset{\widehat{\Pi/S}}{\Longrightarrow} u$ is split into two cases, depending upon whether or not $t_i \models \mathcal{E}(S)$ holds for some $0 < i < n$.

1. If $t_i \models \mathcal{E}(S)$ holds for some $0 < i < n$, then $t \overset{\widehat{\Pi/S}}{\Longrightarrow} u$.

   Proof: Since $t \overset{\widehat{\Pi}}{\Longrightarrow} t_i$ and $t_i \overset{\widehat{\Pi}}{\Longrightarrow} u$, the induction hypothesis implies $t \overset{\widehat{\Pi/S}}{\Longrightarrow} t_i$ and $t_i \overset{\widehat{\Pi/S}}{\Longrightarrow} u$. Thus, $t \overset{\widehat{\Pi/S}}{\Longrightarrow} u$ holds by transitivity of $\overset{\widehat{\Pi/S}}{\Longrightarrow}$.

2. If $t_i \models \neg\mathcal{E}(S)$ holds for all $0 < i < n$, then $t \overset{\widehat{\Pi/S}}{\Longrightarrow} u$.

   2.1. Choose a state v such that $t \underset{R;\langle A\rangle}{\overset{\widehat{\Pi}-\widehat{L}}{\Longrightarrow}} v \underset{L}{\overset{\widehat{\Pi}-\widehat{R;\langle A\rangle}}{\Longrightarrow}} u$.

   Proof: State v exists by SC1, since $t \overset{\widehat{\Pi}}{\Longrightarrow} u$ by hypothesis 6, so $t \underset{S}{\overset{\widehat{\Pi}}{\Longrightarrow}} u$ by the antecedent of 2.

2.2. Choose a state w such that $t \xRightarrow[R]{(\widehat{\Pi}-\widehat{L})-\widehat{\langle A\rangle}} w \xRightarrow[\langle A\rangle]{(\widehat{\Pi}-\widehat{L})-\widehat{R}} v \xRightarrow[L]{\widehat{\Pi}-\widehat{R;\langle A\rangle}} u$.

Proof: State w exists by 2.1 and SC1.

2.3. $w \xrightarrow{\langle A\rangle} v$ or $w \xrightarrow{\widehat{\Pi}-\widehat{S}} v$ or $w = v$.

2.3.1. $w \models \mathcal{E}(\langle A\rangle)$

Proof: By hypothesis 4 and SC2, $t \models \mathcal{E}(\langle A\rangle)$; proof step 2.2 implies $t \xRightarrow{\widehat{\Pi}} w$; and $\widehat{\Pi}$ leaves $\mathcal{E}(\langle A\rangle)$ invariant by definition of atomicity.

2.3.2. Choose states $w_0, \ldots, w_m$, for $0 \leq m$, such that $w = w_0 \xrightarrow{(\widehat{\Pi}-\widehat{L})-\widehat{R}} w_1 \ldots w_{m-1} \xrightarrow{(\widehat{\Pi}-\widehat{L})-\widehat{R}} w_m = v$ and $w_j \models \neg\mathcal{E}(\langle A\rangle)$ for $0 < j < m$.

Proof: By 2.2.

2.3.3. $w_j \models \mathcal{E}(\langle A\rangle)$ for $0 < j < m$.

Proof: By 2.3.1 and the definition of atomicity.

2.3.4. $m \leq 1$

Proof: By 2.3.2 ($w_j \models \neg\mathcal{E}(\langle A\rangle)$ for $0 < j < m$) and 2.3.3.

2.3.5. $w \xrightarrow{\langle A\rangle} v$ or $w \xrightarrow{\widehat{\Pi}-\widehat{S}} v$ or $w = v$

Proof: By 2.3.2 and 2.3.4, since $(\widehat{\Pi}-\widehat{L})-\widehat{R} = (\widehat{\Pi}-\widehat{S}) \cup \widehat{\langle A\rangle}$.

2.4. If $w \xrightarrow{\langle A\rangle} v$ then there exist states x and y such that $t \xRightarrow{\widehat{\Pi}-\widehat{S}} x \xRightarrow{R} w \xrightarrow{\langle A\rangle} v \xRightarrow{L} y \xRightarrow{\widehat{\Pi}-\widehat{S}} u$.

Proof: Step 2.2 and the antecedent imply $t \xRightarrow{(\widehat{\Pi}-\widehat{L})-\widehat{\langle A\rangle}} w \xrightarrow{\langle A\rangle} v \xRightarrow{\widehat{\Pi}-\widehat{R;\langle A\rangle}} u$. The existence of x follows from hypothesis 1(a) and part (a) of Lemma 2, and the existence of y follows from hypothesis 2(a) and part (b) of Lemma 2, since $((\widehat{\Pi}-\widehat{L})-\widehat{\langle A\rangle})-\widehat{R}$ and $(\widehat{\Pi}-\widehat{R;\langle A\rangle})-\widehat{L}$ both equal $\widehat{\Pi}-\widehat{S}$.

2.5. If $w \xrightarrow{\widehat{\Pi}-\widehat{S}} v$ or $w = v$ then there exist states x and y such that $t \xRightarrow{\widehat{\Pi}-\widehat{S}} x \xRightarrow{R} v \xRightarrow{L} y \xRightarrow{\widehat{\Pi}-\widehat{S}} u$.

Proof: Step 2.2 and the antecedent imply $t \xRightarrow{(\widehat{\Pi}-\widehat{L})-\widehat{\langle A\rangle}} w \xRightarrow{\widehat{\Pi}-\widehat{S}} v \xRightarrow{\widehat{\Pi}-\widehat{R;\langle A\rangle}} u$. This implies $t \xRightarrow{(\widehat{\Pi}-\widehat{L})-\widehat{\langle A\rangle}} v \xRightarrow{\widehat{\Pi}-\widehat{R;\langle A\rangle}} u$, since $\widehat{\Pi}-\widehat{S} \subseteq (\widehat{\Pi}-\widehat{L})-\widehat{\langle A\rangle}$. The existence of x follows from hypothesis 1(a) and part (a) of Lemma 2, and the existence of y follows from hypothesis 2(a) and part (b) of Lemma 2, since $((\widehat{\Pi}-\widehat{L})-\widehat{\langle A\rangle})-\widehat{R}$ and $(\widehat{\Pi}-\widehat{R;\langle A\rangle})-\widehat{L}$ both equal $\widehat{\Pi}-\widehat{S}$.

2.6. Choose x and y such that $t \stackrel{\widehat{\Pi}-\widehat{S}}{\Longrightarrow} x \stackrel{R}{\Longrightarrow} w \stackrel{\langle A \rangle}{\longrightarrow} v \stackrel{L}{\Longrightarrow} y \stackrel{\widehat{\Pi}-\widehat{S}}{\Longrightarrow} u$ or $t \stackrel{\widehat{\Pi}-\widehat{S}}{\Longrightarrow} x \stackrel{R}{\Longrightarrow} v \stackrel{L}{\Longrightarrow} y \stackrel{\widehat{\Pi}-\widehat{S}}{\Longrightarrow} u$.

Proof: By 2.3, 2.4, and 2.5.

2.7. $t \stackrel{\widehat{\Pi/S}}{\Longrightarrow} x$ and $y \stackrel{\widehat{\Pi/S}}{\Longrightarrow} u$.

Proof: By 2.6, since $(\widehat{\Pi} - \widehat{S}) \subseteq \widehat{\Pi/S}$.

2.8. $x \stackrel{\widehat{\langle S \rangle}}{\Longrightarrow} y$

2.8.1. $x \models \mathcal{E}(S)$

Proof: By hypothesis 4 and 2.6, since every action of $\widehat{\Pi} - \widehat{S}$ leaves $\mathcal{E}(S)$ invariant.

2.8.2. $y \models \mathcal{E}(S)$

Proof: By hypothesis 5 and 2.6, since every action of $\widehat{\Pi} - \widehat{S}$ leaves $\neg\mathcal{E}(S)$ invariant.

2.8.3. $x \stackrel{\widehat{\langle S \rangle}}{\Longrightarrow} y$

Proof: By 2.6 (which implies $x \stackrel{\widehat{S}}{\Longrightarrow} y$), 2.8.1, and 2.8.2, and the definition of $\langle S \rangle$.

2.9. $t \stackrel{\Pi/S}{\Longrightarrow} u$

Proof: By 2.6, 2.7, and 2.8, since $\widehat{\langle S \rangle} \subseteq \widehat{\Pi/S}$.

3. $t \stackrel{\widehat{\Pi/S}}{\Longrightarrow} u$

Proof: By 1 and 2.

End Proof of Lemma

Lemma 4 *Assume hypotheses* 0–3 *of the Reduction Theorem, and the additional hypotheses that, for states t and u:*

4. Π/S *satisfies* $Init \Rightarrow \Box Q$
5. $t \models Init$
6. $t \stackrel{\widehat{\Pi}}{\Longrightarrow} u$
7. $u \models \mathcal{E}(S)$

Then $u \models Q$.

Proof of Lemma

1. $t \models \mathcal{E}(S)$

Proof: $t \models Init$ by hypothesis 5, and $Init \Rightarrow \mathcal{E}(S)$ by hypothesis 0 of the Reduction Theorem.

2. $t \overset{\widehat{\Pi/S}}{\Longrightarrow} u$

   Proof: By hypotheses 6 and 7, and Lemma 3.

3. $u \models Q$

   Proof: By 1, 2, and hypothesis 4.

End Proof of Lemma

Proof of Theorem

The "only if" part follows from Lemma 1. To prove the "if" part, it suffices to assume, for states t and u:

4. Π/S satisfies $Init \Rightarrow \Box Q$

5. $t \models Init$

6. $t \overset{\widehat{\Pi}}{\Longrightarrow} u$

and show that $u \models Q$.

The proof considers separately the cases $u \models \mathcal{E}(S)$ and $u \models \neg\mathcal{E}(S)$. The second case is further split into the cases $u \models \mathcal{E}(R; \langle A \rangle)$ and $u \models \neg\mathcal{E}(R; \langle A \rangle)$, yielding a total of three separate cases.

1. If $u \models \mathcal{E}(S)$ then $u \models Q$.

   Proof: By Lemma 4.

2. If $u \models (\mathcal{E}(R; \langle A \rangle) \wedge \neg\mathcal{E}(S))$ then $u \models Q$.

   Proof: The proof is by contradiction. We assume that $u \models \neg Q$.

   2.1. Choose a state v such that $u \overset{\widehat{L}}{\Longrightarrow} v$ and $v \models \mathcal{E}(S)$.

   Proof: State v exists by the assumption that $u \models \neg Q$, the antecedent of 2, and hypothesis 3.

   2.2. $t \overset{\widehat{\Pi}}{\Longrightarrow} v$

   Proof: By 2.1 and assumption 6, which asserts that $t \overset{\widehat{\Pi}}{\Longrightarrow} u$.

   2.3. $v \models Q$

   Proof: By 2.2 and Lemma 4, since $v \models \mathcal{E}(S)$ by 2.1. (Substitute v for u in the lemma.)

   2.4. $u \models (\neg Q \wedge \neg\mathcal{E}(S))$

   Proof: By the assumption that $u \models \neg Q$ and the antecedent of 2.

   2.5. $v \models (\neg Q \vee \neg\mathcal{E}(S))$

   Proof: By 2.1, 2.4, and hypothesis 2(b), substituting u for t and v for u.

   2.6. Contradiction.

   Proof: 2.3, 2.5, and 2.1 ($v \models \mathcal{E}(S)$).

3. If $u \models (\neg\mathcal{E}(R;\langle A\rangle) \wedge \neg\mathcal{E}(S))$ then $u \models Q$.

3.1. $t \models \mathcal{E}(S)$

Proof: By hypotheses 5 and 0.

3.2. Choose state v such that $t \stackrel{\widehat{\Pi}}{\Longrightarrow} v \underset{S}{\stackrel{\widehat{\Pi}}{\Longrightarrow}} u$ and $v \models \mathcal{E}(S)$.

Proof: Hypothesis 6 asserts the existence of states t_i such that $t = t_0 \xrightarrow{\widehat{\Pi}} t_1 \xrightarrow{\widehat{\Pi}} \ldots \xrightarrow{\widehat{\Pi}} t_k = u$. Let v be the last t_i such that $t_i \models \mathcal{E}(S)$. By 3.1, t_i exists.

3.3. Choose state w such that $v \underset{R;\langle A\rangle}{\stackrel{\widehat{\Pi}-\widehat{L}}{\Longrightarrow}} w \underset{L}{\stackrel{\widehat{\Pi}-\widehat{R;\langle A\rangle}}{\Longrightarrow}} u$.

Proof: By SC1, since 3.2 asserts that $v \underset{S}{\stackrel{\widehat{\Pi}}{\Longrightarrow}} u$, and S equals $(R;\langle S\rangle);L$.

3.4. If $w \neq u$ then $w \xrightarrow{\widehat{\Pi}-\widehat{S}} u$ and $w \models \neg\mathcal{E}(R;\langle A\rangle)$.

3.4.1. Choose states $w_0, \ldots, w_n$ such that $w = w_0 \xrightarrow{\widehat{\Pi}-\widehat{R;\langle A\rangle}} w_1 \ldots w_{n-1} \xrightarrow{\widehat{\Pi}-\widehat{R;\langle A\rangle}} w_n = u$ and $w_j \models \neg\mathcal{E}(L)$ for $0 < j < n$.

Proof: The states w_j exist by 3.3, which asserts that $w \underset{L}{\stackrel{\widehat{\Pi}-\widehat{R;\langle A\rangle}}{\Longrightarrow}} u$.

3.4.2. $u \models \neg\mathcal{E}(R;\langle A\rangle)$

Proof: Antecedent of 3.

3.4.3. $w_j \models \neg\mathcal{E}(R;\langle A\rangle)$ for $0 \leq j \leq n$

Proof: For $j = n$, this follows from 3.4.2 (since $w_n = u$). For $j < n$, it follows by induction since $\widehat{\Pi} - \widehat{R;\langle A\rangle}$ leaves $\mathcal{E}(R;\langle A\rangle)$ invariant.

3.4.4. $0 \leq n \leq 1$

Proof: By 3.4.1 ($w_j \models \neg\mathcal{E}(L)$ for $0 < j < n$). By 3.4.3, $w_j \models$ *false* for $0 < j < n$, since $\neg\mathcal{E}(L) \wedge \neg\mathcal{E}(R;\langle A\rangle) =$ *false* by SC3.

3.4.5. $w \xrightarrow{\widehat{\Pi}-\widehat{R;\langle A\rangle}} u$

Proof: By 3.4.1 and 3.4.4, since $w \neq u$ (the antecedent of 3.4) implies $n \neq 0$.

3.4.6. $w \xrightarrow{\widehat{\Pi}-\widehat{S}} u$

Proof: By 3.4.3, $w \models \neg\mathcal{E}(R;\langle A\rangle)$. By SC4, this implies $\widehat{L}$ is not enabled in state w. Since $\widehat{S} = \widehat{R;\langle A\rangle} \cup \widehat{L}$, 3.4.5 then implies $w \xrightarrow{\widehat{\Pi}-\widehat{S}} u$.

3.4.7. Proof statement 3.4 holds.

Proof: By 3.4.3 and 3.4.6, since $w = w_0$.

3.5. $v \overset{\widehat{\Pi}-\widehat{L}}{\underset{R;\langle A\rangle}{\Longrightarrow}} u$

Proof: By 3.3, which asserts $v \overset{\widehat{\Pi}-\widehat{L}}{\underset{R;\langle A\rangle}{\Longrightarrow}} w$, and 3.4, since $\widehat{\Pi}-\widehat{S} \subseteq \widehat{\Pi}-\widehat{L}$

3.6. Choose state x such that $v \overset{(\widehat{\Pi}-\widehat{L})-\widehat{\langle A\rangle}}{\underset{R}{\Longrightarrow}} x \overset{(\widehat{\Pi}-\widehat{L})-\widehat{R}}{\underset{\langle A\rangle}{\Longrightarrow}} u$.

Proof: From 3.5 by SC1.

3.7. If $x \neq u$ then $x \xrightarrow{\widehat{\Pi}-\widehat{S}} u$.

3.7.1. $t \overset{\widehat{\Pi}}{\Longrightarrow} x$

Proof: By 3.2 and 3.6.

3.7.2. $x \models \mathcal{E}(\langle A\rangle)$

Proof: 3.1 and SC2 imply $t \models \mathcal{E}(\langle A\rangle)$, and 3.7.1 and the definition of atomicity then imply $x \models \mathcal{E}(\langle A\rangle)$.

3.7.3. $x \xrightarrow{(\widehat{\Pi}-\widehat{L})-\widehat{R}} u$

Proof: By 3.6, there exist states $x_0, \ldots, x_p$ such that $x = x_0 \xrightarrow{(\widehat{\Pi}-\widehat{L})-\widehat{R}} x_1 \ldots x_{p-1} \xrightarrow{(\widehat{\Pi}-\widehat{L})-\widehat{R}} x_p = u$ and $x_j \models \neg\mathcal{E}(\langle A\rangle)$ for $0 < j < p$. By 3.7.2 and the definition of atomicity, $x_j \models \mathcal{E}(\langle A\rangle)$ for $0 \leq j \leq p$. Hence, $p \leq 1$, and since $x \neq u$ (by the antecedent of 3.7), $p = 1$.

3.7.4. $x \xrightarrow{\widehat{\Pi}-\widehat{S}} u$

Proof: Since $u \models \neg\mathcal{E}(R;\langle A\rangle)$ (by the antecedent of 3), SC5 implies that if $x \xrightarrow{\alpha} u$, then $\alpha \not\subseteq \widehat{\langle A\rangle}$. Hence, 3.7.3 implies $x \xrightarrow{\widehat{\Pi}-\widehat{S}} u$, since $((\widehat{\Pi}-\widehat{L})-\widehat{R})-\widehat{\langle A\rangle}$ equals $\widehat{\Pi}-\widehat{S}$.

3.8. $v \overset{(\widehat{\Pi}-\widehat{S})\cup\widehat{R}}{\Longrightarrow} u$

Proof: 3.6 and 3.7 imply $v \overset{(\widehat{\Pi}-\widehat{L})-\widehat{\langle A\rangle}}{\Longrightarrow} u$, and $(\widehat{\Pi}-\widehat{S})\cup\widehat{R} = (\widehat{\Pi}-\widehat{L})-\widehat{\langle A\rangle}$.

3.9. Choose state y such that $v \overset{\widehat{\Pi}-\widehat{S}}{\Longrightarrow} y \overset{\widehat{R}}{\Longrightarrow} u$.

Proof: By 3.8 and Lemma 2.

3.10. $y \models \mathcal{E}(S)$

Proof: From 3.9, since $v \models \mathcal{E}(S)$ by 3.2, and $\widehat{\Pi}-\widehat{S}$ leaves $\mathcal{E}(S)$ invariant.

3.11. $y \models Q$

Proof: Since $t \overset{\widehat{\Pi}}{\Longrightarrow} v$ by 3.2 and $v \overset{\widehat{\Pi}-\widehat{S}}{\Longrightarrow} y$ by 3.9, we have $t \overset{\widehat{\Pi}}{\Longrightarrow} y$. Also, $y \models \mathcal{E}(S)$ by 3.10. Hence, Lemma 4, substituting y for u, implies $y \models Q$.

3.12. $u \models Q$

Proof: By 3.9, 3.10, and 3.11, substituting y for t in hypothesis 1(b) implies $u \models (Q \vee \mathcal{E}(S))$. The antecedent of 3 asserts that $u \models \neg\mathcal{E}(S)$.

4. $u \models Q$

Proof: By 1, 2, and 3.

End Proof of Theorem

References

[1] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.

[2] E. A. Ashcroft. Proving assertions about parallel programs. *Journal of Computer and System Sciences*, 10:110–135, January 1975.

[3] R. J. R. Back. *Refining Atomicity in Parallel Algorithms.* Reports on Computer Science and Mathematics Ser. A, No 57, Swedish University of Åbo, February 1988.

[4] Thomas W. Doeppner, Jr. Parallel program correctness through refinement. In *Fourth Annual ACM Symposium on Principles of Programming Languages*, pages 155–169, ACM, January 1977.

[5] Leslie Lamport. The 'Hoare logic' of concurrent programs. *Acta Informatica*, 14(1):21–37, 1980.

[6] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, March 1977.

[7] Leslie Lamport and Fred B. Schneider. Constraints: a uniform approach to aliasing and typing. In *Proceedings of the Twelfth ACM Symposium on Principles of Programming Languages*, pages 205–216, ACM SIGACT-SIGPLAN, New Orleans, January 1985.

[8] Leslie Lamport and Fred B. Schneider. The "Hoare logic" of CSP, and all that. *ACM Transactions on Programming Languages and Systems*, 6(2):281–296, April 1984.

[9] Richard J. Lipton. Reduction: a method of proving properties of parallel programs. *Communications of the ACM*, 18(12):717–721, December 1975.

[10] Susan Owicki and David Gries. An axiomatic proof technique for parallel programs. *Acta Informatica*, 6(4):319–340, 1976.