Processes are in the Eye of the Beholder

Leslie Lamport

December 25, 1994
revised January 16, 1996

Author's Abstract

A two-process algorithm is shown to be equivalent to an N-process one, illustrating the insubstantiality of processes. A formal equivalence proof in TLA (the Temporal Logic of Actions) is sketched.

Contents

1 Introduction

Processes are often taken to be the fundamental building blocks of concurrency. A concurrent algorithm is traditionally represented as the composition of processes. We show by an example that processes are an artifact of how an algorithm is represented. The difference between a two-process representation and a four-process representation of the same algorithm is no more fundamental than the difference between $2+2$ and $1+1+1+1$.

Our example is a fifo ring buffer, pictured in Figure 1. The ith input value received on channel *in* is stored in $buf[i-1 \bmod N]$, until it is sent on channel *out*. Input and output may occur concurrently, but input is enabled only when the buffer is not full, and output is enabled only when the buffer is not empty.

Figure 2 shows a representation of the ring buffer as a two-process program in a CSP-like language [2]. (We ignore CSP's termination convention; the loops are assumed never to terminate.) The variables p and g record the number of values received on channel *in* by the *Receiver* process and sent on channel *out* by the *Sender* process, respectively. Declaring p and g to be internal means that their values are not externally visible, so a compiler is free to implement them any way it can, or to eliminate them entirely.

The intuitive meaning of this program should be clear to readers acquainted with CSP. We will not attempt to give a rigorous meaning to the program text. Programming languages evolved as a method of describing algorithms to compilers, not as a method for reasoning about them. We do not know how to write a completely formal proof that two programming-language representations of the ring buffer are equivalent. In Section 2, we represent the program formally in TLA, the Temporal Logic of Actions [5]. Figure 2 will serve only as an intuitive description of the TLA formula.

Figure 3 shows another representation of the ring buffer, where *IsNext*

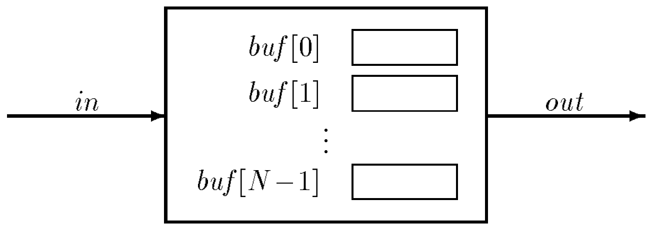

Figure 1: A ring buffer.

$$
\begin{array}{l}
in,\ out : \textbf{channel of } \mathit{Value} \\
\mathit{buf}\ \textbf{array}\ 0\,..\,N-1\ \textbf{of}\ \mathit{Value} \\
p,\ g : \textbf{internal}\ \mathit{Natural}\ \textbf{initially}\ 0 \\
\mathit{Receiver} :: * \begin{bmatrix} p - g \neq N & \rightarrow & in\,?\,\mathit{buf}[p \bmod N]; \\ & & p := p+1 \end{bmatrix} \\
\quad \| \\
\mathit{Sender} :: \quad * \begin{bmatrix} p - g \neq 0 & \rightarrow & out\,!\,\mathit{buf}[g \bmod N]; \\ & & g := g+1 \end{bmatrix}
\end{array}
$$

Figure 2: The ring buffer, represented in a CSP-like language.

$$
\begin{array}{l}
in,\ out : \textbf{channel of } \mathit{Value} \\
\mathit{buf}\ \textbf{array}\ 0\,..\,N-1\ \textbf{of}\ \mathit{Value} \\
pp,\ gg : \textbf{internal array}\ 0\,..\,N-1\ \textbf{of}\ \{0,\ 1\}\ \textbf{initially}\ 0 \\
\mathit{Buffer}(i\ :\ 0\,..\,N-1) :: \\
\quad * \begin{bmatrix} \mathit{empty}\colon\ \mathit{IsNext}(pp,\ i) & \rightarrow & in\,?\,\mathit{buf}[i]; \\ & & pp[i] := 1 - pp[i]\,; \\ \mathit{full}\colon\ \mathit{IsNext}(gg,\ i) & \rightarrow & out\,!\,\mathit{buf}[i]; \\ & & gg[i] := 1 - gg[i] \end{bmatrix}
\end{array}
$$

Figure 3: Another representation of the ring buffer.

p	$pp[0]$	$pp[1]$	$pp[2]$	$pp[3]$
0	$\boxed{0}$	0	0	0
1	1	$\boxed{0}$	0	0
2	1	1	$\boxed{0}$	0
3	1	1	1	$\boxed{0}$
4	$\boxed{1}$	1	1	1
5	0	$\boxed{1}$	1	1
6	0	0	$\boxed{1}$	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Figure 4: The correspondence between values of pp and p, for $N = 4$.

is defined by

$$IsNext(r, i) \triangleq \begin{array}{l} \textbf{if } i = 0 \textbf{ then } r[0] = r[N-1] \\ \qquad\qquad \textbf{else } r[i] \neq r[i-1] \end{array}$$

This is as an N-process program; the ith process, $Buffer(i)$, reads and writes $buf[i]$. Variables p and g of the two-process program are replaced by arrays pp and gg of bits. Array elements $pp[i]$ and $gg[i]$ are read and written by process $Buffer(i)$, and are read by process $Buffer(i+1 \bmod N)$.

The two programs are equivalent because the values assumed by pp and gg in the N-process program correspond directly to the values assumed by p and g in the two-process one. The correspondence between pp and p is shown in Figure 4 for $N = 4$. A boxed number in the $pp[i]$ column indicates that $IsNext(pp, i)$ equals TRUE. The correspondence between gg and g is the same.

It is not hard to argue informally that the two programs are equivalent. Formalizing this argument should be as straightforward as proving formally that $222 + 222$ equals $111 + 111 + 111 + 111$. But, even if straightforward, a completely formal proof of either result from first principles is not trivial. In Section 3, we sketch a formal TLA proof that the two versions of the ring buffer are equivalent.

2 The Algorithm in TLA

We now write the TLA formulas that describe the programs of Figures 2 and 3. The program texts do not tell us what liveness properties are assumed. To make the example more interesting, we assume no liveness properties for sending values on the *in* channel, but we require that every value received in the buffer be eventually sent on the *out* channel. For the two-process program, this means assuming fairness for the *Sender*, but not for the *Receiver*. For the N-process program, it means assuming fairness for the *full* action of each process, but not for the *empty* action.

The program texts also do not determine the grain of atomicity. For simplicity, we assume that an entire guarded command is a single atomic operation. Thus, evaluating a guard and executing the subsequent communication and assignment statements is taken to be an indivisible step.

We give an interleaving representation of the ring buffer—one in which sending and receiving are represented by distinct atomic actions. In Section 4, we describe how the specifications and proofs could be written in terms of a noninterleaving representation that allows values to be sent and received simultaneously.

We use the following notation: $\mathcal{N}$ is the set of natural numbers; $\mathcal{Z}_m$ is the set $\{0, \ldots, m-1\}$; square brackets denote function application; $[S \rightarrow T]$ is the set of functions with domain S and range a subset of T; $[i \in S \mapsto e]$ is the function f with domain S such that $f[i] = e$ for all $i \in S$; $[f \text{ EXCEPT } ![i] = e]$ is the function $\widehat{f}$ that is the same as f except $\widehat{f}[i] = e$; angle brackets enclose tuples; $t[i]$ is the ith component of tuple t, so $\langle v, w \rangle[2] = w$; and $S \setminus T$ is the set of elements in S that are not in T.

A TLA formula is an assertion about *behaviors*, which are sequences of states. Steps (pairs of successive states) in a behavior are described by *actions*, which are boolean-valued expressions containing primed and unprimed variables; unprimed variables refer to the old state and primed variables refer to the new state. To describe CSP-style communication, we represent a channel by a variable and represent the sending of a value by a change to that variable. We define $Channel(V)$ to be the set of legal values of a channel of type V, and $Comm(v, c)$ to be the action that represents communicating a value v on channel c. The actual definitions, given below, are irrelevant; we require only that a $Comm(v, c)$ action changes c, if $v \in V$ and $c \in Channel(V)$.

$$
\begin{aligned}
Channel(V) &\triangleq V \times \mathcal{Z}_2 \\
Comm(v, c) &\triangleq c' = \langle v, 1 - c[2] \rangle
\end{aligned}
$$

The TLA formula Π_2 that represents the two-process program is defined in Figure 5. We now explain that definition.

$$
\begin{array}{lcl}
\mathit{Type2} & \triangleq & \land\ p, g \in \mathcal{N} \\
 & & \land\ \mathit{buf} \in [\mathcal{Z}_N \rightarrow \mathit{Value}] \\
 & & \land\ \mathit{in}, \mathit{out} \in \mathit{Channel}(\mathit{Value}) \\
\mathit{UnB}(i) & \triangleq & [j \in \mathcal{Z}_N \setminus \{i\} \mapsto \mathit{buf}[j]] \\
\mathit{Rcv} & \triangleq & \land\ p - g \neq N \\
 & & \land\ p' = p + 1 \\
 & & \land\ \mathit{Comm}(\mathit{buf}'[p \bmod N], \mathit{in}) \\
 & & \land\ \text{UNCHANGED}\ \langle g, \mathit{out}, \mathit{UnB}(p \bmod N) \rangle \\
\mathit{Snd} & \triangleq & \land\ p - g \neq 0 \\
 & & \land\ g' = g + 1 \\
 & & \land\ \mathit{Comm}(\mathit{buf}[g \bmod N], \mathit{out}) \\
 & & \land\ \text{UNCHANGED}\ \langle p, \mathit{buf}, \mathit{in} \rangle \\
\Phi_2 & \triangleq & \land\ \Box \mathit{Type2} \\
 & & \land\ (p = 0) \land \Box[\mathit{Rcv}]_{\langle p, \mathit{buf}, \mathit{in} \rangle} \\
 & & \land\ (g = 0) \land \Box[\mathit{Snd}]_{\langle g, \mathit{out} \rangle} \land \mathrm{WF}_{\langle g, \mathit{out} \rangle}(\mathit{Snd}) \\
\Pi_2 & \triangleq & \exists\, p, g : \Phi_2
\end{array}
$$

Figure 5: The TLA formula Π_2 representing the two-process program.

A list of expressions bulleted by $\land$ denotes their conjunction; indentation is used to eliminate parentheses. If formula F is written as such a list, then $F.i$ is its ith conjunct—for example, $Rcv.2$ is $p' = p{+}1$. A similar convention is used for disjunctions.

The state predicate *Type*2 asserts that each variable has the correct type. (The array variable *buf* of the programming language representation becomes a variable whose value is a function.) The type declarations of the two-process program are represented by the TLA formula $\Box Type2$, which asserts that *Type*2 equals TRUE in all states of the behavior.

Action *Snd* describes a step of the *Sender* process; it can occur only when $p - g \neq 0$, and it increments g by 1, communicates $buf[g \bmod N]$ on channel *out*, and leaves p, *buf*, and *in* unchanged (UNCHANGED v is defined to equal $v' = v$). Similarly, action *Rcv* describes a step of the *Receiver* process. The conjunct $Rcv.3$ asserts that the value $buf'[p \bmod N]$ (the new value of $buf[p \bmod N]$) is communicated on channel *in*. The state function $UnB(i)$ is defined so that, if it is unchanged, then $buf[j]$ is unchanged for all $j \neq i$. Thus, *Rcv* asserts that the new value of $buf[p \bmod N]$ is the value communicated on channel *in*, and that $buf[j]$ remains unchanged for all $j \neq p \bmod N$.

Formula $\Phi_2.2$ describes the *Receiver* process. It asserts that p is initially 0, and that every step is a *Rcv* step or leaves p, *buf*, and *in* unchanged ($[A]_v$ is defined to equal $A \vee (v' = v)$). Steps that leave p, *buf*, and *in* unchanged represent steps of the *Receiver*'s environment—either steps of the *Sender* or steps of the entire program's environment. The conjunct $\Phi_2.3$ similarly represents the *Sender* process. The formula $\mathrm{WF}_{\langle g, out \rangle}(Snd)$ asserts weak fairness of the *Snd* action. In general, $\mathrm{WF}_v(A)$ asserts that if action $\langle A \rangle_v$ (defined to equal $A \wedge (v' \neq v)$) remains continuously enabled, then an $\langle A \rangle_v$ step must eventually occur.

Formula Φ_2 is the conjunction of the specifications of the two processes with the formula asserting type correctness. It describes the two-process program with p and g visible. The complete program specification Π_2 is obtained by hiding p and g. In logic, hiding means existential quantification; in temporal logic, flexible variables (distinct from rigid variables like N) are hidden with the temporal existential quantifier $\boldsymbol{\exists}$.

The conjunct $\Box Type2$ of Φ_2 makes type correctness an explicit part of the specification. We put type-correctness assumptions in our specifications to make them as much like Figures 2 and 3 as possible. However, to avoid errors, it is usually better to let type correctness be a consequence of the specification. We could rewrite Φ_2 as follows to eliminate the conjunct $\Box Type2$. The conjunct $\Box Type2.1$ is already redundant because it is implied by $\Phi_2.2 \wedge \Phi_2.3$. We can eliminate $\Box Type2.3$ by making $Type2.3$ part of the initial condition, since $Type2.3 \wedge \Phi_2.2 \wedge \Phi_2.3$ implies $\Box Type2.3$. (The proof requires the fact that $c \in Channel(V)$ and $Comm(v, c)$ imply $c' \in Channel(V)$.) We can eliminate $\Box Type2.2$ in the same way, if we modify *Rcv* so it leaves the domain of *buf* unchanged.

The TLA formula Π_N that represents the N-process program is defined in Figure 6. There are two things in this definition that merit further explanation. First, we introduce an array *ctl* to represent the control state. The value of $ctl[i]$ equals "empty" if control in process $Buffer(i)$ is at the point labeled *empty*, and it equals "full" if control is at *full*. Second, we introduce an action $NotProc(i)$ that has no obvious counterpart in Figure 3 or in Π_2. The specifications of the two processes in Figure 2 are especially simple because each variable is changed by an action of only one of the processes. For example, a step of the *Sender*'s environment can be characterized as any step that leaves g and *out* unchanged. We can think of g and *out* as belonging to the *Sender*. In the N-process program, $pp[i]$, $gg[i]$, and $ctl[i]$ belong to $Buffer(i)$. However, *in* and *out* don't belong to any single process; they can be changed by a step of any of the N pro-

$$
\begin{array}{lcl}
\mathit{TypeN} & \triangleq & \land\ pp, gg \in [\mathcal{Z}_N \to \mathcal{Z}_2] \\
 & & \land\ ctl \in [\mathcal{Z}_N \to \{\text{“empty”}, \text{“full”}\}] \\
 & & \land\ buf \in [\mathcal{Z}_N \to \mathit{Value}] \\
 & & \land\ in, out \in \mathit{Channel}(\mathit{Value}) \\
\mathit{Fill}(i) & \triangleq & \land\ ctl[i] = \text{“empty”} \\
 & & \land\ \mathit{IsNext}(pp, i) \\
 & & \land\ ctl' = [ctl \ \textsc{except}\ ![i] = \text{“full”}] \\
 & & \land\ pp' = [pp \ \textsc{except}\ ![i] = 1 - pp[i]] \\
 & & \land\ \mathit{Comm}(buf'[i], in) \\
 & & \land\ \textsc{unchanged}\ \langle gg, out, \mathit{UnB}(i) \rangle \\
\mathit{Empty}(i) & \triangleq & \land\ ctl[i] = \text{“full”} \\
 & & \land\ \mathit{IsNext}(gg, i) \\
 & & \land\ ctl' = [ctl \ \textsc{except}\ ![i] = \text{“empty”}] \\
 & & \land\ gg' = [gg \ \textsc{except}\ ![i] = 1 - gg[i]] \\
 & & \land\ \mathit{Comm}(buf[i], out) \\
 & & \land\ \textsc{unchanged}\ \langle pp, in, buf \rangle \\
\mathit{NotProc}(i) & \triangleq & \land\ \textsc{unchanged}\ \langle pp[i], gg[i], ctl[i], buf[i] \rangle \\
 & & \land\ \mathit{IsNext}(pp, i) \Rightarrow \textsc{unchanged}\ in \\
 & & \land\ \mathit{IsNext}(gg, i) \Rightarrow \textsc{unchanged}\ out \\
\mathit{varN} & \triangleq & \langle pp, gg, ctl, buf, in, out \rangle \\
\Phi_{\mathrm{N}} & \triangleq & \land\ \Box \mathit{TypeN} \\
 & & \land\ \forall\, i \in \mathcal{Z}_N \ :\ \land\ (pp[i] = gg[i] = 0) \land (ctl[i] = \text{“empty”}) \\
 & & \qquad\qquad\qquad \land\ \Box[\mathit{Fill}(i) \lor \mathit{Empty}(i) \lor \mathit{NotProc}(i)]_{\mathit{varN}} \\
 & & \qquad\qquad\qquad \land\ \mathrm{WF}_{\mathit{varN}}(\mathit{Empty}(i)) \\
\Pi_{\mathrm{N}} & \triangleq & \exists\, pp, gg, ctl \ :\ \Phi_{\mathrm{N}}
\end{array}
$$

Figure 6: The TLA formula Π_{N} representing the N-process program.

cesses. The variable *in* belongs to $Buffer(i)$ only when $IsNext(pp, i)$ equals TRUE, and *out* belongs to $Buffer(i)$ only when $IsNext(gg, i)$ equals TRUE. Action $NotProc(i)$ characterizes steps of $Buffer(i)$'s environment, which is allowed to change *in* when $IsNext(pp, i)$ equals FALSE, and to change *out* when $IsNext(gg, i)$ equals FALSE. The subscript in $\Box[\ldots]_{varN}$ allows steps of the entire program's environment that leave all the variables unchanged. It is semantically superfluous, since $NotProc(i)$ already allows such steps, but the syntax of TLA requires some subscript.

3 The Proof

We now give a hierarchically structured proof that Π_2 and Π_{N} are equivalent [4]. The proof is completely formal, meaning that each step is a mathematical formula. English is used only to explain the low-level reasoning. The entire proof could be carried down to a level at which each step follows from the simple application of formal rules, but such a detailed proof is more suitable for machine checking than human reading. Our complete proof, with "Q.E.D." steps and low-level reasoning omitted, appears in Appendix A.

The correctness of the algorithm rests on simple properties of integers and of the *mod* operator. We need the following lemma, where the bit array $Rep(m)$ used to represent the integer m is defined by

$$Rep(m) \;\triangleq\; [i \in \mathcal{Z}_N \mapsto \textbf{if } i < m \bmod 2N \leq i + N \textbf{ then } 1 \textbf{ else } 0]$$

The lemma is proved in Appendix B. We assume throughout that N is a positive integer.

Lemma 1 *If $m \in \mathcal{N}$ and $i \in \mathcal{Z}_N$, then*

1. $IsNext(Rep(m), i) \equiv (i = m \bmod N)$
2. $IsNext(Rep(m), i) \Rightarrow$
   $Rep(m+1) = [Rep(m) \text{ EXCEPT } ![i] = 1 - Rep(m)[i]]$

For temporal reasoning, we use the following TLA rules from Figure 5 of [5].

Theorem $\Pi_2 \equiv \Pi_N$

1a. $\Phi_2 \equiv \Phi_2^u$
 b. $\Phi_N \equiv \Phi_N^u$
2a. $\Phi_2^u \equiv \exists\, pp, gg, ctl : \Phi_2^h$
 b. $\Phi_N^u \equiv \exists\, p, g : \Phi_N^h$
3. $\Phi_2^h \equiv \Phi_N^h$
4. Q.E.D.
PROOF:

$$
\begin{array}{rcll}
\Pi_2 & \equiv & \exists\, p, g : \Phi_2^u & \text{step 1a and the definition of } \Pi_2 \\
 & \equiv & \exists\, p, g, pp, gg, ctl : \Phi_2^h & \text{step 2a} \\
 & \equiv & \exists\, p, g, pp, gg, ctl : \Phi_N^h & \text{step 3} \\
 & \equiv & \exists\, pp, gg, ctl, p, g : \Phi_N^h & \text{simple logic} \\
 & \equiv & \exists\, pp, gg, ctl : \Phi_N^u & \text{step 2b} \\
 & \equiv & \Pi_N & \text{step 1b and the definition of } \Pi_N
\end{array}
$$

Figure 7: The high-level structure of the proof.

(This version of TLA2 generalizes the one in [5].)

$$
\begin{array}{ll}
\text{STL2. } \vdash \Box F \Rightarrow F & \text{STL3. } \vdash \Box\Box F \equiv \Box F \\
\text{STL4. } \dfrac{F \Rightarrow G}{\Box F \Rightarrow \Box G} & \text{STL5. } \vdash \Box(F \wedge G) \equiv (\Box F) \wedge (\Box G) \\
\text{INV1. } \dfrac{I \wedge [N]_f \Rightarrow I'}{I \wedge \Box[N]_f \Rightarrow \Box I} & \text{INV2. } \vdash \Box I \Rightarrow (\Box[N]_f \equiv \Box[N \wedge I \wedge I']_f)
\end{array}
$$

$$
\text{TLA2. } \frac{P \wedge (\forall i \in S : [A_i]_{f_i}) \Rightarrow Q \wedge [B]_g}{\Box P \wedge (\forall i \in S : \Box[A_i]_{f_i}) \Rightarrow \Box Q \wedge \Box[B]_g}
$$

The high-level structure of the proof is shown in Figure 7. The proofs of steps 1–3, and the definitions of Φ_2^u, Φ_N^u, Φ_2^h, and Φ_N^h, are given in the following sections.

3.1 Step 1: Removing the Process Structure

Formulas Φ_2^u and Φ_N^u are defined in Figure 8. They can be thought of as uniprocess versions of the two algorithms. We obtained them by rewriting Φ_2 and Φ_N as formulas with a single next-state relation, instead of as the conjunction of processes.

Step 1a is proved as follows.

1a. $\Phi_2 \equiv \Phi_2^u$
 1a.1. $Type2 \Rightarrow ([Rcv]_{\langle p, buf, in\rangle} \wedge [Snd]_{\langle g, out\rangle} \equiv [Rcv \vee Snd]_{var2})$
 PROOF: Given below.

$$
\begin{array}{lcl}
var2 & \triangleq & \langle p, g, buf, in, out \rangle \\
\Phi_2^{\mathrm{u}} & \triangleq & \land \Box\, Type2 \\
& & \land\ (p = 0) \land (g = 0) \\
& & \land\ \Box[Rcv \lor Snd]_{var2} \\
& & \land\ \mathrm{WF}_{\langle g, out \rangle}(Snd) \\
\Phi_{\mathrm{N}}^{\mathrm{u}} & \triangleq & \land \Box\, TypeN \\
& & \land\ \forall\, i \in \mathcal{Z}_N : (pp[i] = gg[i] = 0) \land (ctl[i] = \text{“}\mathsf{empty}\text{”}) \\
& & \land\ \Box[\exists\, i \in \mathcal{Z}_N : Fill(i) \lor Empty(i)]_{varN} \\
& & \land\ \forall\, i \in \mathcal{Z}_N : \mathrm{WF}_{varN}(Empty(i))
\end{array}
$$

Figure 8: Formulas Φ_2^{u} and $\Phi_{\mathrm{N}}^{\mathrm{u}}$.

1a.2. $\Box\, Type2 \Rightarrow (\Box[Rcv]_{\langle p, buf, in \rangle} \land \Box[Snd]_{\langle g, out \rangle} \equiv \Box[Rcv \lor Snd]_{var2})$

PROOF: Step 1a.1 and rule TLA2.

1a.3. Q.E.D.

PROOF: Step 1a.2 and the definitions of Φ_2 and Φ_2^{u}.

Step 1a.1 is proved by showing that $Type2$ implies

$$
\begin{array}{lll}
\multicolumn{3}{l}{[Rcv]_{\langle p, buf, in \rangle} \land [Snd]_{\langle g, out \rangle}} \\
\equiv & \multicolumn{2}{l}{\text{by definition of } [A]_v} \\
& \multicolumn{2}{l}{\land\ Rcv \lor (\langle p, buf, in \rangle' = \langle p, buf, in \rangle)} \\
& \multicolumn{2}{l}{\land\ Snd \lor (\langle g, out \rangle' = \langle g, out \rangle)} \\
\equiv & \multicolumn{2}{l}{\text{by propositional logic}} \\
& \multicolumn{2}{l}{\lor\ Rcv \land (\langle g, out \rangle' = \langle g, out \rangle)} \\
& \multicolumn{2}{l}{\lor\ Snd \land (\langle p, buf, in \rangle' = \langle p, buf, in \rangle)} \\
& \multicolumn{2}{l}{\lor\ Rcv \land Snd} \\
& \multicolumn{2}{l}{\lor\ (\langle g, out \rangle' = \langle g, out \rangle) \land (\langle p, buf, in \rangle' = \langle p, buf, in \rangle)} \\
\equiv & \lor\ Rcv & Rcv \text{ implies } \langle g, out \rangle' = \langle g, out \rangle \\
& \lor\ Snd & Snd \text{ implies } \langle p, buf, in \rangle' = \langle p, buf, in \rangle \\
& \lor\ \text{FALSE} & Type2 \land Rcv \text{ implies } p' \neq p, \text{ and } Snd \text{ implies } p' = p \\
& \lor\ var2' = var2 & \langle v_1, \ldots, v_m \rangle' = \langle v_1, \ldots, v_m \rangle \text{ iff } (v_1' = v_1) \land \ldots \land (v_m' = v_m) \\
\equiv & [Rcv \lor Snd]_{var2} & \text{by definition of } [A]_v
\end{array}
$$

All of the nontemporal steps in our proof can be reduced to this kind of algebraic manipulation. From now on, we just sketch such proofs and leave the detailed calculations to the reader.

The proof of step 1b is similar to that of step 1a, but it is a bit more difficult because it requires an invariant $InvN$, which asserts that the arrays

pp and gg are representations of natural numbers.

$$InvN \triangleq (\exists\, m \in \mathcal{N} : pp = Rep(m)) \wedge (\exists\, m \in \mathcal{N} : gg = Rep(m))$$

1b. $\Phi_{\mathrm{N}} \equiv \Phi^{\mathrm{u}}_{\mathrm{N}}$

1b.1a. $\Phi_{\mathrm{N}} \Rightarrow \Box InvN$

b. $\Phi^{\mathrm{u}}_{\mathrm{N}} \Rightarrow \Box InvN$

PROOF: Described below.

1b.2. $TypeN \wedge InvN \Rightarrow$
$[\exists\, i \in \mathcal{Z}_N : Fill(i) \vee Empty(i)]_{varN} \equiv$
$\forall\, i \in \mathcal{Z}_N : [Fill(i) \vee Empty(i) \vee NotProc(i)]_{varN}$

PROOF: If $i \neq j$, then $TypeN$ implies that $Fill(i) \wedge Fill(j)$, $Empty(i) \wedge Empty(j)$, and $Fill(i) \wedge Empty(j)$ are all false; and $TypeN \wedge InvN$ implies $Fill(i) \Rightarrow NotProc(j)$ and $Empty(i) \Rightarrow NotProc(j)$. By Lemma 1.1, $TypeN \wedge InvN$ implies $(\forall\, i \in \mathcal{Z}_N : NotProc(i)) \equiv (varN' = varN)$.

1b.3. $\Box TypeN \wedge \Box InvN \Rightarrow$
$\Box[\exists\, i \in \mathcal{Z}_N : Fill(i) \vee Empty(i)]_{varN} \equiv$
$\forall\, i \in \mathcal{Z}_N : \Box[Fill(i) \vee Empty(i) \vee NotProc(i)]_{varN}$

PROOF: Step 1b.2 and rule TLA2.

1b.4 Q.E.D.

PROOF: Steps 1b.1 and 1b.3 and the definitions of Φ_{N} and $\Phi^{\mathrm{u}}_{\mathrm{N}}$.

Steps 1b.1a and 1b.1b are standard invariance properties; 1b.1a is proved as follows.

1b.1a $\Phi_{\mathrm{N}} \Rightarrow \Box InvN$

1b.1a.1 $TypeN \wedge (\forall\, i \in \mathcal{Z}_N : pp[i] = gg[i] = 0) \Rightarrow InvN$

PROOF: $Rep(0) = [i \in \mathcal{Z}_N \mapsto 0]$

1b.1a.2 $\wedge\ InvN$
$\wedge\ [TypeN \wedge (\forall\, i \in \mathcal{Z}_N : Fill(i) \vee Empty(i) \vee NotProc(i))]_{varN}$
$\Rightarrow InvN'$

1b.1a.2.1. $InvN \wedge TypeN \wedge (i \in \mathcal{Z}_N) \wedge Fill(i) \Rightarrow InvN'$

1b.1a.2.2. $InvN \wedge TypeN \wedge (i \in \mathcal{Z}_N) \wedge Empty(i) \Rightarrow InvN'$

1b.1a.2.3. $InvN \wedge TypeN \wedge (\forall\, i \in \mathcal{Z}_N : NotProc(i)) \Rightarrow InvN'$

1b.1a.2.4. $InvN \wedge (varN' = varN) \Rightarrow InvN'$

1b.1a.2.5. Q.E.D.

PROOF: Steps 1b.1a.2.1–1b.1a.2.4.

1b.1a.3. $\wedge\ TypeN \wedge (\forall\, i \in \mathcal{Z}_N : pp[i] = gg[i] = 0)$
$\wedge\ \Box TypeN$
$\wedge\ \Box[\forall\, i \in \mathcal{Z}_N : Fill(i) \vee Empty(i) \vee NotProc(i)]_{varN}$
$\Rightarrow \Box InvN$

PROOF: Steps 1b.1a.1 and 1b.1a.2 and rules INV1 and INV2.

1b.1a.4. Q.E.D.

PROOF: Step 1b.1a.3 and rule TLA2, since $(\forall\, i : [A_i]_v) \equiv [\forall\, i : A_i]_v$.

Steps 1b.1a.2.1 and 1b.1a.2.2 are proved using Lemma 1.2; steps 1b.1a.2.3 and 1b.1a.2.4 follow because their hypotheses imply $pp' = pp$ and $gg' = gg$. As indicated in the appendix, the proof of step 1b.1b is similar.

3.2 Step 2: Adding History Variables

Formulas Φ_2^{h} and $\Phi_{\mathrm{N}}^{\mathrm{h}}$ are defined in Figure 9, which also defines their safety parts, Φ_2^{hS} and $\Phi_{\mathrm{N}}^{\mathrm{hS}}$. We obtained Φ_2^{h} by adding pp, gg, and ctl as history variables to Φ_2^{u}; and we obtained $\Phi_{\mathrm{N}}^{\mathrm{h}}$ by adding p and g as history variables to $\Phi_{\mathrm{N}}^{\mathrm{u}}$. In general, adding an auxiliary variable a to a formula F means writing a formula F^a such that $F \equiv \boldsymbol{\exists}\, a : F^a$. A history variable is an auxiliary variable that records information from previous states. It is added by using the following lemma, which can be deduced from the results in [1]. Step 2 is easily proved by repeated application of this lemma.

Lemma 2 (History Variable) *If h and h' do not occur in Init, $\mathcal{A}_i$, $\mathcal{B}_j$, v, or f, and h' does not occur in g_i, for all $i \in I$ and $j \in J$, then*

$$
\begin{aligned}
& \mathit{Init} \;\wedge\; \Box[\exists\, i \in I : \mathcal{A}_i]_v \;\wedge\; (\forall\, j \in J : \mathrm{WF}_v(\mathcal{B}_j)) \\
& \quad \equiv \boldsymbol{\exists}\, h : \wedge\; \mathit{Init} \wedge (h = f) \\
& \qquad\qquad \wedge\; \Box[\exists\, i \in I : \mathcal{A}_i \wedge (h' = g_i)]_{\langle h, v\rangle} \\
& \qquad\qquad \wedge\; \forall\, j \in J : \mathrm{WF}_v(\mathcal{B}_j)
\end{aligned}
$$

3.3 Step 3: Equivalence of Φ_2^{h} and $\Phi_{\mathrm{N}}^{\mathrm{h}}$

In the two-process algorithm, p and g are the actual internal variables, while pp, gg, and ctl are history variables. The situation is reversed in the N-process algorithm. Step 3 involves showing that the history variables of one algorithm behave like the internal variables of the other. Its proof uses the following formulas, where Inv will be shown to be an invariant of both Φ_2^{h} and $\Phi_{\mathrm{N}}^{\mathrm{h}}$.

$$
\mathit{IsFull}(g, p, i) \;\stackrel{\Delta}{=}\; \exists\, m \in \mathcal{N} : (g \leq m < p) \wedge (i = m \bmod N)
$$

$$
\begin{aligned}
\mathit{Inv} \;\stackrel{\Delta}{=}\; & \wedge\; pp = \mathit{Rep}(p) \\
& \wedge\; gg = \mathit{Rep}(g) \\
& \wedge\; ctl = [i \in \mathcal{Z}_N \mapsto \textbf{if } \mathit{IsFull}(g, p, i) \textbf{ then } \text{“full”} \textbf{ else } \text{“empty”}] \\
& \wedge\; 0 \leq p - g \leq N
\end{aligned}
$$

The high-level structure of the proof is:

$$
\begin{array}{lcl}
\mathit{Init} & \triangleq & \land\ p = g = 0 \\
 & & \land\ pp = gg = [i \in \mathcal{Z}_N \mapsto 0] \\
 & & \land\ ctl = [i \in \mathcal{Z}_N \mapsto \text{“}\mathsf{empty}\text{”}] \\
\mathit{Type} & \triangleq & \mathit{Type2} \land \mathit{TypeN} \\
\mathit{var} & \triangleq & \langle pp, gg, ctl, p, g, \mathit{buf}, \mathit{in}, \mathit{out} \rangle
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{HRcv} & \triangleq & \land\ \mathit{Rcv} \\
 & & \land\ pp' = [pp \text{ EXCEPT } ![p \bmod N] = 1 - pp[p \bmod N]] \\
 & & \land\ ctl' = [ctl \text{ EXCEPT } ![p \bmod N] = \text{“}\mathsf{full}\text{”}] \\
 & & \land\ \text{UNCHANGED } gg \\
\mathit{HSnd} & \triangleq & \land\ \mathit{Snd} \\
 & & \land\ gg' = [gg \text{ EXCEPT } ![g \bmod N] = 1 - gg[g \bmod N]] \\
 & & \land\ ctl' = [ctl \text{ EXCEPT } ![g \bmod N] = \text{“}\mathsf{empty}\text{”}] \\
 & & \land\ \text{UNCHANGED } pp \\
\Phi_2^{\mathrm{hS}} & \triangleq & \land\ \Box \mathit{Type} \\
 & & \land\ \mathit{Init} \\
 & & \land\ \Box[\mathit{HRcv} \lor \mathit{HSnd}]_{var} \\
\Phi_2^{\mathrm{h}} & \triangleq & \Phi_2^{\mathrm{hS}} \land \mathrm{WF}_{\langle g, out \rangle}(\mathit{Snd})
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{HFill}(i) & \triangleq & \land\ \mathit{Fill}(i) \\
 & & \land\ p' = p + 1 \\
 & & \land\ \text{UNCHANGED } g \\
\mathit{HEmpty}(i) & \triangleq & \land\ \mathit{Empty}(i) \\
 & & \land\ g' = g + 1 \\
 & & \land\ \text{UNCHANGED } p \\
\Phi_{\mathrm{N}}^{\mathrm{hS}} & \triangleq & \land\ \Box \mathit{Type} \\
 & & \land\ \mathit{Init} \\
 & & \land\ \Box[\exists\, i \in \mathcal{Z}_N : \mathit{HFill}(i) \lor \mathit{HEmpty}(i)]_{var} \\
\Phi_{\mathrm{N}}^{\mathrm{h}} & \triangleq & \Phi_{\mathrm{N}}^{\mathrm{hS}} \land (\forall\, i \in \mathcal{Z}_N : \mathrm{WF}_{varN}(\mathit{Empty}(i)))
\end{array}
$$

Figure 9: Formulas Φ_2^{h} and $\Phi_{\mathrm{N}}^{\mathrm{h}}$.

3. $\Phi_2^{\mathrm{h}} \equiv \Phi_{\mathrm{N}}^{\mathrm{h}}$

3.1a. $\mathit{Type} \wedge \mathit{Inv} \Rightarrow (\mathit{HRcv} \equiv \exists\, i \in \mathcal{Z}_N : \mathit{HFill}(i))$

b. $\mathit{Type} \wedge \mathit{Inv} \Rightarrow (\mathit{HSnd} \equiv \exists\, i \in \mathcal{Z}_N : \mathit{HEmpty}(i))$

3.2. $[\mathit{Type} \wedge \mathit{Inv} \wedge (\mathit{HRcv} \vee \mathit{HSnd})]_{var} \equiv$
$[\mathit{Type} \wedge \mathit{Inv} \wedge (\exists\, i \in \mathcal{Z}_N : \mathit{HFill}(i) \vee \mathit{HEmpty}(i))]_{var}$

3.3a. $\Phi_2^{\mathrm{hS}} \Rightarrow \Box \mathit{Inv}$

b. $\Phi_{\mathrm{N}}^{\mathrm{hS}} \Rightarrow \Box \mathit{Inv}$

3.4. $\Phi_2^{\mathrm{hS}} \equiv \Phi_{\mathrm{N}}^{\mathrm{hS}}$

3.5. $\Box \mathit{Inv} \wedge \Phi_{\mathrm{N}}^{\mathrm{hS}} \Rightarrow (\mathrm{WF}_{\langle g, out\rangle}(\mathit{Snd}) \equiv (\forall\, i \in \mathcal{Z}_N : \mathrm{WF}_{varN}(\mathit{Empty}(i))))$

3.6. Q.E.D.

PROOF: Immediate from steps 3.3–3.5.

Steps 3.1 and 3.2 are (nontemporal) action formulas. They make it intuitively clear why the two transformed formulas are equivalent. Step 3.1a is proved as follows.

3.1a. $\mathit{Type} \wedge \mathit{Inv} \Rightarrow (\mathit{HRcv} \equiv \exists\, i \in \mathcal{Z}_N : \mathit{HFill}(i))$

3.1a.1 $\mathit{Type} \wedge \mathit{Inv} \Rightarrow (\mathit{HRcv} \equiv \mathit{HFill}(p \bmod N))$

3.1a.1.1. $\mathit{Type} \wedge \mathit{Inv} \Rightarrow ((p - g \neq N) \equiv (\mathit{ctl}[p \bmod N] = \text{“empty”}))$

PROOF: Arithmetic reasoning and the definition of *IsFull*.

3.1a.1.2. $\mathit{Type} \wedge \mathit{Inv} \Rightarrow \mathit{IsNext}(pp, p \bmod N)$

PROOF: Lemma 1.1.

3.1a.1.3. Q.E.D.

PROOF: Steps 3.1a.1.1 and 3.1a.1.2, and the definitions of *HRcv* and *HFill*.

3.1a.2 $\mathit{Type} \wedge \mathit{Inv} \Rightarrow (\mathit{HFill}(p \bmod N) \equiv (\exists\, i \in \mathcal{Z}_N : \mathit{HFill}(i)))$

PROOF: By Lemma 1.1, $\mathit{Type} \wedge \mathit{Inv}$ implies $\mathit{IsNext}(pp, p \bmod N)$ and $\neg \mathit{IsNext}(pp, i)$, if $i \in \mathcal{Z}_N$ and $i \neq (p \bmod N)$.

3.1a.3 Q.E.D.

PROOF: Steps 3.1a.1 and 3.1a.2.

As indicated in the appendix, the proof of 3.1b is analogous. Step 3.2 follows easily from step 3.1.

Step 3.3 asserts that *Inv* is an invariant of both formulas; its proof is a standard invariance argument.

3.3a. $\Phi_2^{\mathrm{hS}} \Rightarrow \Box \mathit{Inv}$

b. $\Phi_{\mathrm{N}}^{\mathrm{hS}} \Rightarrow \Box \mathit{Inv}$

3.3.1. $\mathit{Init} \Rightarrow \mathit{Inv}$

PROOF: $\mathit{Rep}(0)$ equals $[i \in \mathcal{Z}_N \mapsto 0]$ and $\mathit{IsFull}(0, 0, i) \equiv \text{FALSE}$, for all $i \in \mathcal{Z}_N$.

3.3.2a. $Inv \wedge [Type \wedge (HRcv \vee HSnd)]_{var} \Rightarrow Inv'$

b. $Inv \wedge [Type \wedge (\exists\, i \in \mathcal{Z}_N : HFill(i) \vee HEmpty(i))]_{var} \Rightarrow Inv'$

PROOF: Given below.

3.3.3. Q.E.D.

PROOF: Steps 3.3.1 and 3.3.2, and rules INV1 and INV2.

Step 3.3.2 asserts that the next-state actions leave Inv invariant. The proof of 3.3.2a is:

3.3.2a. $Inv \wedge [Type \wedge (HRcv \vee HSnd)]_{var} \Rightarrow Inv'$

3.3.2a.1 $Inv \wedge Type \wedge HRcv \Rightarrow Inv'$

PROOF: Assume $Inv \wedge Type \wedge HRcv$. Then $Inv.1'$ is immediate because $p' = p$ and $pp' = pp$; $Inv.2'$ follows from Lemma 1.2; $Inv.4'$ follows from $Inv.4$, since $HRcv$ implies $p' = p + 1$, $g' = g$, and $p - g \neq N$; and $Inv.3'$ holds because

$$\begin{aligned} IsFull(g', p', i) \;\equiv\;& \text{by definition of } HRcv \\ & IsFull(g, p+1, i) \\ \equiv\;& \text{by definition of } IsFull \\ & \exists\, m \in \mathcal{N} : (g \leq m < p+1) \wedge (i = m \bmod N) \\ \equiv\;& \text{by } Rcv.1 \text{ and } Inv.4 \\ & \textbf{if } i = p \bmod N \textbf{ then } \text{TRUE} \textbf{ else } IsFull(g, p, i) \end{aligned}$$

3.3.2a.2 $Inv \wedge Type \wedge HSnd \Rightarrow Inv'$

PROOF: Similar to the proof of 3.3.2a.1.

3.3.2a.3 $Inv \wedge (var' = var) \Rightarrow Inv'$

PROOF: Immediate.

3.3.2a.4 Q.E.D.

PROOF: Steps 3.3.2a.1–3.3.2a.3.

Step 3.3.2b follows from steps 3.3.2a and 3.2. This completes the proof of step 3.3.

Steps 3.4 and 3.5, assert the equivalence of the safety and liveness parts of the formulas, respectively. Step 3.4 follows from 3.3 and

$$\begin{aligned} &\Box Type \wedge \Box Inv \Rightarrow \\ &\quad \Box[HRcv \vee HSnd]_{var} \equiv \Box[\exists\, i \in \mathcal{Z}_N : HFill(i) \vee HEmpty(i)]_{var} \end{aligned}$$

which follows from step 3.2 and rule TLA2. Step 3.5 has the following high-level proof.

3.5. $\Box Inv \wedge \Phi_{\mathrm{N}}^{\mathrm{hS}} \Rightarrow (\mathrm{WF}_{\langle g, out\rangle}(Snd) \equiv (\forall\, i \in \mathcal{Z}_N : \mathrm{WF}_{varN}(Empty(i))))$

3.5.1. $\Box Inv \wedge \Phi_{\mathrm{N}}^{\mathrm{hS}} \Rightarrow$
$(\forall\, i \in \mathcal{Z}_N : \mathrm{WF}_{varN}(Empty(i))) \equiv \mathrm{WF}_{varN}(\exists\, i \in \mathcal{Z}_N : Empty(i))$

3.5.2. $\Box Inv \wedge \Box Type \wedge \Box[HRcv \vee HSnd]_{var} \Rightarrow$
$\mathrm{WF}_{\langle g, out \rangle}(Snd) \equiv \mathrm{WF}_{varN}(\exists\, i \in \mathcal{Z}_N : Empty(i))$

3.5.3. Q.E.D.

PROOF: Steps 3.5.1 and 3.5.2.

We first consider step 3.5.1. When writing TLA specifications, one often has to choose between asserting fairness of $A_1 \vee \ldots \vee A_m$ and asserting fairness of each action A_i. The choice becomes a matter of taste when the resulting specifications are equivalent. This is the case if, whenever one of the A_i becomes enabled, a step of no other A_j can occur before the next A_i step. For weak fairness, the equivalence is a consequence of the following result, which can be derived from the TLA proof rules of [5].

Lemma 3 *If*

$$\text{ENABLED}\,\langle \mathcal{A}_i \rangle_v \;\wedge\; \Box Inv \wedge \Box[\mathcal{N} \wedge \neg \mathcal{A}_i]_v \;\Rightarrow\; \Box\neg\text{ENABLED}\,\langle \mathcal{A}_j \rangle_v$$

for all $i, j \in S$ *with* $i \neq j$, *then*

$$\Box Inv \;\wedge\; \Box[\mathcal{N}]_v \;\Rightarrow\; (\mathrm{WF}_v(\exists\, i \in S : \mathcal{A}_i) \;\equiv\; (\forall\, i \in S : \mathrm{WF}_v(\mathcal{A}_i)))$$

We use this lemma to prove step 3.5.1.

3.5.1. $\Box Inv \wedge \Phi_{\mathrm{N}}^{\mathrm{hS}} \Rightarrow$
$(\forall\, i \in \mathcal{Z}_N : \mathrm{WF}_{varN}(Empty(i))) \equiv \mathrm{WF}_{varN}(\exists\, i \in \mathcal{Z}_N : Empty(i)))$

3.5.1.1. $\Phi_{\mathrm{N}}^{\mathrm{hS}} \Rightarrow \Box[\exists\, i \in \mathcal{Z}_N : Fill(i) \vee Empty(i)]_{varN}$

PROOF: TLA2, since $HFill(i) \Rightarrow Fill(i)$ and $HEmpty(i) \Rightarrow Empty(i)$.

3.5.1.2. $\wedge\ i \in \mathcal{Z}_N$
$\wedge\ IsNext(gg, i)$
$\wedge\ \Box(Inv \wedge Type)$
$\wedge\ \Box[(\exists\, j \in \mathcal{Z}_N : Fill(j) \vee Empty(j)) \wedge \neg Empty(i)]_{varN}$
$\Rightarrow \Box IsNext(gg, i)$

PROOF: By rules INV1 and INV2, since

$$Inv \wedge Type \wedge (Fill(j) \vee Empty(j)) \wedge \neg Empty(i)$$

implies $gg' = gg$, for all $i, j \in \mathcal{Z}_N$.

3.5.1.3. $\wedge\ (i, j \in \mathcal{Z}_N) \wedge (i \neq j)$
$\wedge\ \text{ENABLED}\,\langle Empty(i) \rangle_{varN}$
$\wedge\ \Box(Inv \wedge Type)$
$\wedge\ \Box[(\exists\, k \in \mathcal{Z}_N : Fill(k) \vee Empty(k)) \wedge \neg Empty(i)]_{varN}$
$\Rightarrow \Box\neg\text{ENABLED}\,\langle Empty(j) \rangle_{varN}$

PROOF: Step 3.5.1.2 and rule STL4, since ENABLED $\langle Empty(i) \rangle_{varN}$ implies $IsNext(gg, i)$, and Lemma 1.1 implies

$$Inv \wedge Type \wedge IsNext(gg, i) \Rightarrow \neg IsNext(gg, j)$$

for all $i, j \in \mathcal{Z}_N$ with $i \neq j$.

3.5.1.4. Q.E.D.

PROOF: Steps 3.5.1.1 and 3.5.1.3, and Lemma 3.

Finally, we prove 3.5.2, which completes the proof of the theorem.

3.5.2. $\Box Inv \wedge \Box Type \wedge \Box[HRcv \vee HSnd]_{var} \Rightarrow$
$\mathrm{WF}_{\langle g, out \rangle}(Snd) \equiv \mathrm{WF}_{varN}(\exists\, i \in \mathcal{Z}_N : Empty(i))$

3.5.2.1. $Inv \wedge Type \wedge [HRcv \vee HSnd]_{var} \Rightarrow$
$\langle Snd \rangle_{\langle g, out \rangle} \equiv \langle \exists\, i \in \mathcal{Z}_N : Empty(i) \rangle_{varN}$

PROOF: By steps 3.1b and 3.2, since $Inv \wedge Type \wedge [HRcv \vee HSnd]_{var}$ implies $\langle Snd \rangle_{\langle g, out \rangle} \equiv HSnd$, and

$$Inv \wedge Type \wedge [\exists\, i \in \mathcal{Z}_N : HFill(i) \vee HEmpty(i)]_{var}$$

implies $\langle \exists\, i \in \mathcal{Z}_N : Empty(i) \rangle_{varN} \equiv (\exists\, i \in \mathcal{Z}_N : HEmpty(i))$.

3.5.2.2. $\Box Inv \wedge \Box Type \wedge \Box[HRcv \vee HSnd]_{var} \Rightarrow$
$\Box\Diamond\langle Snd \rangle_{\langle g, out \rangle} \equiv \Box\Diamond\langle \exists\, i \in \mathcal{Z}_N : Empty(i) \rangle_{varN}$

PROOF: $\Box Inv \wedge \Box Type \wedge \Box[HRcv \vee HSnd]_{var}$

$\Rightarrow$ by 3.5.2.1 and rules STL5 and TLA2, since $\langle A \rangle_v \equiv \neg[\neg A]_v$
$\Box[\neg Snd]_{\langle g, out \rangle} \equiv \Box[\neg \exists\, i \in \mathcal{Z}_N : Empty(i)]_{varN}$

$\Rightarrow$ $\neg\Box[\neg Snd]_{\langle g, out \rangle} \equiv \neg\Box[\neg \exists\, i \in \mathcal{Z}_N : Empty(i)]_{varN}$

$\Rightarrow$ by rules STL3, STL4, and STL5
$\Box\neg\Box[\neg Snd]_{\langle g, out \rangle} \equiv \Box\neg\Box[\neg \exists\, i \in \mathcal{Z}_N : Empty(i)]_{varN}$

$\Rightarrow$ since $\Diamond \equiv \neg\Box\neg$
$\Box\Diamond\neg[\neg Snd]_{\langle g, out \rangle} \equiv \Box\Diamond\neg[\neg \exists\, i \in \mathcal{Z}_N : Empty(i)]_{varN}$

$\Rightarrow$ since $\langle A \rangle_v \equiv \neg[\neg A]_v$
$\Box\Diamond\langle Snd \rangle_{\langle g, out \rangle} \equiv \Box\Diamond\langle \exists\, i \in \mathcal{Z}_N : Empty(i)) \rangle_{varN}$

3.5.2.3 $\Box Inv \wedge \Box Type \wedge \Box[HRcv \vee HSnd]_{var} \Rightarrow$
$\Box\Diamond\neg \textsc{Enabled}\ \langle Snd \rangle_{\langle g, out \rangle} \equiv$
$\Box\Diamond\neg \textsc{Enabled}\ \langle \exists\, i \in \mathcal{Z}_N : Empty(i)) \rangle_{varN}$

PROOF: Rules STL2 (which implies $F \Rightarrow \Diamond F$), STL4, STL5, and TLA2, since by 3.5.2.1, $Inv \wedge Type \wedge [HRcv \vee HSnd]_{var}$ implies

$$\textsc{Enabled}\ \langle Snd \rangle_{\langle g, out \rangle} \equiv \textsc{Enabled}\ \langle \exists\, i \in \mathcal{Z}_N : Empty(i)) \rangle_{varN}$$

3.5.2.4 Q.E.D.

PROOF: Steps 3.5.2.2 and 3.5.2.3, since $\mathrm{WF}_v(A)$ is defined to equal $\Box\Diamond\neg \textsc{Enabled}\ \langle A \rangle_v \vee \Box\Diamond\langle A \rangle_v$.

4 Further Remarks

We have proved the equivalence of two different representations of the ring buffer. This is not just an intellectual exercise; the ability to transform an algorithm into a completely different form is important for applying formal methods to real systems. Going from the two-process version to the N-

process one reduces the internal state of each process from an unbounded number (p or g) to three bits ($pp[i]$, $gg[i]$, and $ctl[i]$). As explained in [3], such a transformation enables us to apply model checking to unbounded-state systems.

In retrospect, it is not surprising that programs with different numbers of processes can be equivalent. Multiprocess programs are routinely executed on single-processor computers by interleaving the execution of their processes. The transformation of Φ_2 and Φ_{N} to Φ_2^{u} and $\Phi_{\mathrm{N}}^{\mathrm{u}}$ can be viewed as a formal description of this interleaving.

Using an interleaving representation makes the proof of equivalence a bit simpler, but it is not necessary. The equivalence of noninterleaving representations can be proved as follows. Let *RcvNI* and *SndNI* be the actions obtained from *Rcv* and *Snd* by removing the UNCHANGED conjuncts and adding the conjunct UNCHANGED $UnB(p \bmod N)$ to *RcvNI*. Replacing *Rcv* and *Snd* with *RcvNI* and *SndNI* in the definition of Π_2 yields a noninterleaving representation of the two-process program. Similarly, we get a noninterleaving representation of the N-process program by replacing $Fill(i)$ and $Empty(i)$ with actions $FillNI(i)$ and $EmptyNi(i)$ that have no UNCHANGED conjuncts except the one for $UnB(i)$. In the proof of equivalence, formula Φ_2^{u} is changed by replacing its next-state action $Rcv \lor Snd$ with $Rcv \lor Snd \lor (RcvNI \land SndNI)$, and $\Phi_{\mathrm{N}}^{\mathrm{u}}$ is changed by replacing its next-state action with $\exists\, i \in \mathcal{Z}_N : Fill(i) \lor Empty(i) \lor (FillNI(i) \land EmptyNI(i))$. Formulas Φ_2^{h} and $\Phi_{\mathrm{N}}^{\mathrm{h}}$ are obtained by adding history variables to the new versions of Φ_2^{u} and $\Phi_{\mathrm{N}}^{\mathrm{u}}$. The proof of equivalence is the same as before, except we have to consider the next-state actions' extra disjuncts. These disjuncts represent the simultaneous sending and receiving of values.

Indivisible state changes are an abstraction; executing an operation of a real program takes time. In TLA, we can represent the concurrent execution of program operations either as successive steps, or as a single step. Which representation we choose is a matter of convenience, not philosophy. We have found that interleaving representations are usually, but not always, more convenient than noninterleaving ones for reasoning about algorithms.

A proof that two algorithms are equivalent can be turned into a derivation of one algorithm from the other. Our proof yields the following derivation, where each equivalence is obtained from the indicated proof step(s).

$$\begin{array}{lll} \Pi_2 & \equiv \ \boldsymbol{\exists} : p, g : \Phi_2^{\mathrm{u}} & \text{1a} \\ & \equiv \ \boldsymbol{\exists}\, p, g, pp, gg, ctl : \Phi_2^{\mathrm{h}} & \text{2a} \end{array}$$

$$
\begin{array}{lll}
\equiv & \exists\, p, g, pp, gg, ctl : \Phi_2^{\mathrm{h}} \land \Box Inv & \text{3.3a} \\
\equiv & \exists\, pp, gg, ctl, p, g : \Phi_{\mathrm{N}}^{\mathrm{h}} \land \Box Inv & \text{3.4 and 3.5} \\
\equiv & \exists\, pp, gg, ctl, p, g : \Phi_{\mathrm{N}}^{\mathrm{h}} & \text{3.3b} \\
\equiv & \exists\, p, g : \Phi_{\mathrm{N}}^{\mathrm{u}} & \text{2b} \\
\equiv & \exists\, p, g : \Phi_{\mathrm{N}}^{\mathrm{u}} \land \Box InvN & \text{1b.1b} \\
\equiv & \exists\, p, g : \Phi_{\mathrm{N}} \land \Box InvN & \text{1b.3} \\
\equiv & \Pi_{\mathrm{N}} & \text{1b.1a}
\end{array}
$$

Our derivation uses rules of logic to rewrite formulas. In process algebra [6], analogous transformations are performed by applying algebraic laws. It would be interesting to compare a process-algebraic proof of equivalence of the two ring-buffer programs with our TLA proof.

A Proof of the Theorem

Theorem $\Pi_2 \equiv \Pi_{\mathrm{N}}$

1a. $\Phi_2 \equiv \Phi_2^{\mathrm{u}}$

1a.1. $Type2 \Rightarrow ([Rcv]_{\langle p, buf, in \rangle} \land [Snd]_{\langle g, out \rangle} \equiv [Rcv \lor Snd]_{var2})$

1a.2. $\Box Type2 \Rightarrow (\Box[Rcv]_{\langle p, buf, in \rangle} \land \Box[Snd]_{\langle g, out \rangle} \equiv \Box[Rcv \lor Snd]_{var2})$

1b. $\Phi_{\mathrm{N}} \equiv \Phi_{\mathrm{N}}^{\mathrm{u}}$

1b.1a. $\Phi_{\mathrm{N}} \Rightarrow \Box InvN$

1b.1a.1 $TypeN \land (\forall\, i \in \mathcal{Z}_N : pp[i] = gg[i] = 0) \Rightarrow InvN$

1b.1a.2 $\land\ InvN$
$\land\ [TypeN \land (\forall\, i \in \mathcal{Z}_N : Fill(i) \lor Empty(i) \lor NotProc(i))]_{varN}$
$\Rightarrow InvN'$

1b.1a.3. $\land\ TypeN \land (\forall\, i \in \mathcal{Z}_N : pp[i] = gg[i] = 0)$
$\land\ \Box TypeN \land \Box[\forall\, i \in \mathcal{Z}_N : Fill(i) \lor Empty(i) \lor NotProc(i)]_{varN}$
$\Rightarrow \Box InvN$

1b.1b. $\Phi_{\mathrm{N}}^{\mathrm{u}} \Rightarrow \Box InvN$

1b.1b.1. $TypeN \land (\forall\, i \in \mathcal{Z}_N : pp[i] = gg[i] = 0) \Rightarrow InvN$

1b.1b.2. $\land\ InvN$
$\land\ [TypeN \land (\exists\, i \in \mathcal{Z}_N : Fill(i) \lor Empty(i)]_{varN}$
$\Rightarrow InvN'$

1b.1b.3. $\land\ TypeN \land (\forall\, i \in \mathcal{Z}_N : pp[i] = gg[i] = 0)$
$\land\ \Box TypeN \land \Box[\exists\, i \in \mathcal{Z}_N Fill(i) \lor Empty(i)]_{varN}$
$\Rightarrow InvN'$

1b.2. $TypeN \land InvN \Rightarrow$
$[\exists\, i \in \mathcal{Z}_N : Fill(i) \lor Empty(i)]_{varN} \equiv$
$\forall\, i \in \mathcal{Z}_N : [Fill(i) \lor Empty(i) \lor NotProc(i)]_{varN}$

1b.3. $\Box\, TypeN \land \Box\, InvN \Rightarrow$
$\Box[\exists\, i \in \mathcal{Z}_N : Fill(i) \lor Empty(i)]_{varN} \equiv$
$\forall\, i \in \mathcal{Z}_N : \Box[Fill(i) \lor Empty(i) \lor NotProc(i)]_{varN}$

2a. $\Phi_2^{\mathrm{u}} \equiv \boldsymbol{\exists}\, pp, gg, ctl : \Phi_2^{\mathrm{h}}$

b. $\Phi_{\mathrm{N}}^{\mathrm{u}} \equiv \boldsymbol{\exists}\, p, g : \Phi_{\mathrm{N}}^{\mathrm{h}}$

3. $\Phi_2^{\mathrm{h}} \equiv \Phi_{\mathrm{N}}^{\mathrm{h}}$

3.1a. $Type \land Inv \Rightarrow (HRcv \equiv \exists\, i \in \mathcal{Z}_N : HFill(i))$

3.1a.1 $Type \land Inv \Rightarrow (HRcv \equiv HFill(p \bmod N))$

3.1a.1.1. $Type \land Inv \Rightarrow ((p - g \neq N) \equiv (ctl[p \bmod N] = \text{“}\mathsf{empty}\text{”}))$

3.1a.1.2. $Type \land Inv \Rightarrow IsNext(pp, p \bmod N)$

3.1a.2 $Type \land Inv \Rightarrow (HFill(p \bmod N) \equiv (\exists\, i \in \mathcal{Z}_N : HFill(i))$

3.1b. $Type \land Inv \Rightarrow (HSnd \equiv \exists\, i \in \mathcal{Z}_N : HEmpty(i))$

3.1b.1 $Type \land Inv \Rightarrow (HSnd \equiv HEmpty(g \bmod N)$

3.1b.1.1. $Type \land Inv \Rightarrow ((p - g \neq 0) \equiv (ctl[g \bmod N] = \text{“}\mathsf{empty}\text{”}))$

3.1b.1.2. $Type \land Inv \Rightarrow IsNext(gg, g \bmod N)$

3.1b.1 $Type \land Inv \Rightarrow (HEmpty(p \bmod N) \equiv (\exists\, i \in \mathcal{Z}_N : HEmpty(i))$

3.2. $[Type \land Inv \land (HRcv \lor HSnd)]_{var} \equiv$
$[Type \land Inv \land (\exists\, i \in \mathcal{Z}_N : HFill(i) \lor HEmpty(i))]_{var}$

3.3a. $\Phi_2^{\mathrm{hS}} \Rightarrow \Box\, Inv$

b. $\Phi_{\mathrm{N}}^{\mathrm{hS}} \Rightarrow \Box\, Inv$

3.3.1. $Init \Rightarrow Inv$

3.3.2a. $Inv \land [Type \land (HRcv \lor HSnd)]_{var} \Rightarrow Inv'$

3.3.2a.1 $Inv \land Type \land HRcv \Rightarrow Inv'$

3.3.2a.2 $Inv \land Type \land HSnd \Rightarrow Inv'$

3.3.2a.3 $Inv \land (var' = var) \Rightarrow Inv'$

3.3.2b. $Inv \land [Type \land (\exists\, i \in \mathcal{Z}_N : HFill(i) \lor HEmpty(i))]_{var} \Rightarrow Inv'$

3.4. $\Phi_2^{\mathrm{hS}} \equiv \Phi_{\mathrm{N}}^{\mathrm{hS}}$

3.5. $\Box\, Inv \land \Phi_{\mathrm{N}}^{\mathrm{hS}} \Rightarrow (\mathrm{WF}_{\langle g, out \rangle}(Snd) \equiv (\forall\, i \in \mathcal{Z}_N : \mathrm{WF}_{varN}(Empty(i))))$

3.5.1. $\Box\, Inv \land \Phi_{\mathrm{N}}^{\mathrm{hS}} \Rightarrow$
$(\forall\, i \in \mathcal{Z}_N : \mathrm{WF}_{varN}(Empty(i))) \equiv \mathrm{WF}_{varN}(\exists\, i \in \mathcal{Z}_N : Empty(i))$

3.5.1.1. $\Phi_{\mathrm{N}}^{\mathrm{hS}} \Rightarrow \Box[\exists\, i \in \mathcal{Z}_N : Fill(i) \lor Empty(i)]_{varN}$

3.5.1.2. $\land\; i \in \mathcal{Z}_N$
$\land\; IsNext(gg, i)$
$\land\; \Box(Inv \land Type)$
$\land\; \Box[(\exists\, j \in \mathcal{Z}_N : Fill(j) \lor Empty(j)) \land \neg Empty(i)]_{varN}$
$\Rightarrow \Box\, IsNext(gg, i)$

3.5.1.3. $\wedge\ (i, j \in \mathcal{Z}_N) \wedge (i \neq j)$
$\wedge$ ENABLED $\langle Empty(i) \rangle_{varN}$
$\wedge\ \Box(Inv \wedge Type)$
$\wedge\ \Box[(\exists\, k \in \mathcal{Z}_N\ :\ Fill(k) \vee Empty(k)) \wedge \neg Empty(i)]_{varN}$
$\Rightarrow \Box\neg$ENABLED $\langle Empty(j) \rangle_{varN}$

3.5.2. $\Box Inv \wedge \Box Type \wedge \Box[HRcv \vee HSnd]_{var} \Rightarrow$
$\mathrm{WF}_{\langle g, out \rangle}(Snd) \equiv \mathrm{WF}_{varN}(\exists\, i \in \mathcal{Z}_N\ :\ Empty(i))$

3.5.2.1. $Inv \wedge Type \wedge [HRcv \vee HSnd]_{var} \Rightarrow$
$\langle Snd \rangle_{\langle g, out \rangle} \equiv \langle \exists\, i \in \mathcal{Z}_N : Empty(i) \rangle_{varN}$

3.5.2.2. $\Box Inv \wedge \Box Type \wedge \Box[HRcv \vee HSnd]_{var} \Rightarrow$
$\Box\Diamond\langle Snd \rangle_{\langle g, out \rangle} \equiv \Box\Diamond\langle \exists\, i \in \mathcal{Z}_N\ :\ Empty(i)) \rangle_{varN}$

3.5.2.3 $\Box Inv \wedge \Box Type \wedge \Box[HRcv \vee HSnd]_{var} \Rightarrow$
$\Box\Diamond\neg$ENABLED $\langle Snd \rangle_{\langle g, out \rangle} \equiv$
$\Box\Diamond\neg$ENABLED $\langle \exists\, i \in \mathcal{Z}_N\ :\ Empty(i)) \rangle_{varN}$

B Proof of Lemma 1

Lemma 1 *If $m \in \mathcal{N}$ and $i \in \mathcal{Z}_N$, then*

1. $IsNext(Rep(m), i) \equiv (i = m \bmod N)$

2. $IsNext(Rep(m), i) \Rightarrow$
   $Rep(m+1) = [Rep(m)$ EXCEPT $![i] = 1 - Rep(m)[i]]$

1. $Rep(m+1) = [Rep(m)$ EXCEPT $![m \bmod N] = 1 - Rep(m)[m \bmod N]]$

1.1. CASE: $(m+1) \bmod 2N = (m \bmod 2N) + 1$

PROOF: It suffices to prove that

$$Rep(m+1)[j] \ = \ \textbf{if}\ j = m \bmod N\ \textbf{then}\ 1 - Rep(m)[j]\ \textbf{else}\ Rep(m)[j]$$

for any $j \in \mathcal{Z}_N$. The proof follows.

1.1.1. $(j = m \bmod N) \equiv (j = m \bmod 2N) \vee (j + N = m \bmod 2N)$

PROOF: Simple number theory.

1.1.2. If $j = m \bmod N$, then
$(j < 1 + (m \bmod 2N) \leq j + N) \equiv \neg(j < m \bmod 2N \leq j + N)$

PROOF: Step 1.1.1 and simple arithmetic.

1.1.3. If $j \neq m \bmod N$ then
$(j < 1 + (m \bmod 2N) \leq j + N) \equiv (j < m \bmod 2N \leq j + N)$

PROOF: Step 1.1.1 and simple arithmetic.

1.1.4. Q.E.D.

PROOF: By 1.1.2, 1.1.3, and the definition of Rep.

1.2. CASE: $(m+1) \bmod 2N \neq (m \bmod 2N) + 1$

PROOF: The case assumption implies $m \bmod 2N = 2N - 1$, which implies

$$\begin{array}{lcl} Rep(m) & = & [i \in \mathcal{Z}_N \mapsto \textbf{if } i = N-1 \textbf{ then } 0 \textbf{ else } 1] \\ Rep(m+1) & = & [i \in \mathcal{Z}_N \mapsto 0] \end{array}$$

1.3. Q.E.D.

PROOF: Steps 1.1 and 1.2.

2. $IsNext(Rep(m), i) \equiv (i = m \bmod N)$

The proof is by induction on m.

2.1. CASE: $m = 0$

PROOF: $Rep(0) = [j \in \mathcal{Z}_N \mapsto 0]$ and $IsNext(Rep(0), i) \equiv (i = 0)$, for $i \in \mathcal{Z}_N$.

2.2. ASSUME: $IsNext(Rep(m), i) \equiv (i = m \bmod N)$

PROVE: $IsNext(Rep(m+1), i) \equiv (i = m + 1 \bmod N)$

The result is trivial if $N = 1$. We assume $N > 1$.

2.2.1. $(i = m \bmod N) \Rightarrow (IsNext(Rep(m+1), i) \equiv \neg IsNext(Rep(m), i))$

PROOF: $i = m \bmod N$ implies $IsNext(Rep(m+1), i)$

$$\begin{array}{ll} \equiv & \text{by definition of } IsNext \\ & \textbf{if } i = 0 \textbf{ then } Rep(m+1)[0] = Rep(m+1)[N-1] \\ & \phantom{\textbf{if } i = 0} \textbf{ else } Rep(m+1)[i] \neq Rep(m+1)[i-1] \\ \equiv & \text{by step 1} \\ & \textbf{if } i = 0 \textbf{ then } 1 - Rep(m)[0] = Rep(m)[N-1] \\ & \phantom{\textbf{if } i = 0} \textbf{ else } 1 - Rep(m)[i] \neq Rep(m)[i-1] \\ \equiv & \neg IsNext(Rep(m), i) \end{array}$$

2.2.2. $(i = m+1 \bmod N) \Rightarrow (IsNext(Rep(m+1), i) \equiv \neg IsNext(Rep(m), i))$

PROOF: $i = m + 1 \bmod N$ implies $IsNext(Rep(m+1), i)$

$$\begin{array}{ll} \equiv & \text{by definition of } IsNext \\ & \textbf{if } i = 0 \textbf{ then } Rep(m+1)[0] = Rep(m+1)[N-1] \\ & \phantom{\textbf{if } i = 0} \textbf{ else } Rep(m+1)[i] \neq Rep(m+1)[i-1] \\ \equiv & \text{by step 1, since } N > 1 \text{ implies } m + 1 \bmod N \neq m \bmod N \\ & \textbf{if } i = 0 \textbf{ then } Rep(m)[0] = 1 - Rep(m)[N-1] \\ & \phantom{\textbf{if } i = 0} \textbf{ else } Rep(m)[i] \neq 1 - Rep(m)[i-1] \\ \equiv & \neg IsNext(Rep(m), i) \end{array}$$

2.2.3. $(i \neq m \bmod N) \wedge (i \neq m + 1 \bmod N) \Rightarrow$
$IsNext(Rep(m+1), i) \equiv IsNext(Rep(m), i)$

PROOF: The hypothesis implies $IsNext(Rep(m+1), i)$

$\equiv$ by definition of *IsNext*

$$\textbf{if } i = 0 \textbf{ then } Rep(m+1)[0] = Rep(m+1)[N-1] \textbf{ else } Rep(m+1)[i] \neq Rep(m+1)[i-1]$$

$\equiv$ by step 1

$$\textbf{if } i = 0 \textbf{ then } Rep(m)[0] = Rep(m)[N-1] \textbf{ else } Rep(m)[i] \neq Rep(m)[i-1]$$

$\equiv$ $IsNext(Rep(m), i)$

2.2.4. Q.E.D.

PROOF: By 2.2.1–2.2.3 and the induction assumption.

2.3 Q.E.D.

PROOF: By steps 2.1 and 2.2 and mathematical induction.

3. $IsNext(Rep(m), i) \Rightarrow (Rep(m+1) = [Rep(m) \text{ EXCEPT } ![i] = 1 - Rep(m)[i]])$

   PROOF: Immediate from steps 1 and 2.

4. Q.E.D.

   PROOF: Steps 2 and 3.

References

[1] Martín Abadi, Leslie Lamport, and Stephan Merz. Refining specifications. To appear.

[2] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, August 1978.

[3] R. P. Kurshan and Leslie Lamport. Verification of a multiplier: 64 bits and beyond. In Costas Courcoubetis, editor, *Computer-Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 166–179, Berlin, June 1993. Springer-Verlag. Proceedings of the Fifth International Conference, CAV'93.

[4] Leslie Lamport. How to write a proof. Research Report 94, Digital Equipment Corporation, Systems Research Center, February 1993. To appear in *American Mathematical Monthly*.

[5] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[6] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, New York, 1980.