Conjoining Specifications

Martín Abadi and Leslie Lamport

3 November 1995

This is a revised version of SRC Research Report 118. It will appear in *ACM Transactions on Programming Languages and Systems*.

Contents

Conjoining Specifications

MARTÍN ABADI and LESLIE LAMPORT
Digital Equipment Corporation

We show how to specify components of concurrent systems. The specification of a system is the conjunction of its components' specifications. Properties of the system are proved by reasoning about its components. We consider both the decomposition of a given system into parts, and the composition of given parts to form a system.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Program Verification—*correctness proofs*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*specification techniques*

General Terms: Theory, Verification

Additional Key Words and Phrases: Composition, concurrent programming, decomposition, liveness properties, modular specification, safety properties, temporal logic

1. INTRODUCTION

Large systems are built from smaller parts. We present a method for deducing properties of a system by reasoning about its components. We show how to represent an individual component Π_i by a formula S_i so that the parallel composition usually denoted **cobegin** $\Pi_1 \parallel \ldots \parallel \Pi_n$ **coend** is represented by the formula $S_1 \wedge \ldots \wedge S_n$. Composition is conjunction.

We reduce composition to conjunction not for the sake of elegance, but because it is the best way we know to prove properties of composite systems. Rigorous reasoning requires logic, and hence a language of logical formulas. It does not require a conventional programming language for describing systems. We find it most convenient to regard programs and circuit descriptions as low-level specifications, and to represent them in the same logic used for higher-level specifications. The logic we use is TLA, the Temporal Logic of Actions [Lamport 1994]. We do not discuss here the important problem of translating from a low-level TLA specification to an implementation in a conventional language.

The idea of representing concurrent programs and their specifications as formulas in a temporal logic was first proposed by Pnueli [1981]. It was later observed that, if specifications allow "stuttering" steps that leave the state unchanged, then $S_l \Rightarrow S_h$ asserts that S_l implements S_h [Lamport 1983]. Hence, proving that a lower-level specification implements a higher-level one was reduced to proving a formula in

Authors' address: Systems Research Center, Digital Equipment Corporation, 130 Lytton Avenue, Palo Alto, CA 94301.

the logic. Still later, it was noticed that the formula $\exists x : S$ specifies the same system as S except with the variable x hidden [Abadi and Lamport 1991; Lamport 1989], and variable hiding became logical quantification. The idea of composition as conjunction has also been suggested [Abadi and Plotkin 1993; Abramsky and Jagadeesan 1994; Zave and Jackson 1993], but our method for reducing composition to conjunction is new.

To deduce useful properties of a component, we must specify its environment. No component will exhibit its intended behavior in the presence of a sufficiently hostile environment. For example, a combinational circuit will not produce an output in the intended range if some input line, instead of having a 0 or a 1, has an improper voltage level of 1/2. The specification of the circuit's environment must rule out such improper inputs.

How we reason about a composite system depends on how it was formed. Composite specifications arise in two ways: by *decomposing* a given system into smaller parts and by *composing* given parts to form a larger system. These two situations call for two methods of writing component specifications that differ in their treatment of the environment. This difference leads in turn to different proof rules.

When decomposing a specification, the environment of each component is assumed to be the other components, and is usually left implicit. To reason about a component, we must state what we are assuming about its environment, and then prove that this assumption is satisfied by the other components. The Decomposition Theorem of Section 4 provides the needed proof rule. It reduces the verification of a complex, low-level system to proving properties of a higher-level specification and properties of one low-level component at a time. Decomposing proofs in this way allows us to apply decision procedures to verifications that hitherto required completely hand-guided proofs [Kurshan and Lamport 1993].

When specifying a reusable component, without knowing precisely where it will be used, we must make explicit what it assumes of its environment. We therefore assert that the component satisfies a guarantee M only as long as its environment satisfies an assumption E. This assumption/guarantee property [Jones 1983] is denoted $E \overset{+}{\rhd} M$. To show that a composition of reusable components satisfies a specification S, we must prove a formula of the form $(E_1 \overset{+}{\rhd} M_1) \wedge \ldots \wedge (E_n \overset{+}{\rhd} M_n) \Rightarrow S$, where S may again be an assumption/guarantee property. We prove such a formula with the Composition Theorem of Section 5. This theorem allows us to reason about assumption/guarantee specifications using well-established, effective methods for reasoning about specifications of complete systems.

In the following section, we examine the issues that arise in decomposition and composition. Our discussion is informal, because we wish to show that these issues are fundamental, not artifacts of a particular formalism. We treat these topics formally in Sections 4 and 5. Section 3 covers the formal preliminaries. A comparison with related work appears in the conclusion. Proofs are relegated to the appendix.

2. AN INFORMAL OVERVIEW

2.1 Decomposing Complete Systems

A complete system is one that is self-contained; it may be observed, but it does not interact with the observer. A program is a complete system, provided we model inputs as being generated nondeterministically by the program itself.

As a tiny example of a complete system, we consider a program for computing a GCD (greatest common divisor), for which we have devised an informal programming-language notation. Statements within angle brackets are executed atomically; **loop-endloop** keywords enclose an infinite loop; **cobegin-coend** keywords enclose parallel statements, separated by $\|$; and semicolon has its usual meaning. When writing processes, we will also mark variables as output, input, or internal; a process cannot change its input variables or access the internal variables of another process.

$$
\begin{array}{l}
\textsc{Program GCD} \\
\quad \textbf{var} \;\; a \;\textbf{initially}\; 233344, \;\; b \;\textbf{initially}\; 233577899\,; \\
\quad \textbf{cobegin} \;\; \textbf{loop} \;\langle\; \textbf{if}\; a > b \;\textbf{then}\; a := a - b \;\rangle\; \textbf{endloop} \\
\qquad\qquad\qquad \| \\
\qquad\qquad\quad \textbf{loop} \;\langle\; \textbf{if}\; b > a \;\textbf{then}\; b := b - a \;\rangle\; \textbf{endloop} \;\; \textbf{coend}
\end{array}
$$

Program GCD satisfies the correctness property that eventually a and b become and remain equal to the gcd of 233344 and 233577899. We make no distinction between programs and properties, writing them all as TLA formulas. If formula M_{gcd} represents program GCD, and formula P_{gcd} represents the correctness property, then the program implements the property iff (if and only if) M_{gcd} implies P_{gcd}. Thus, correctness of program GCD is verified by proving $M_{gcd} \Rightarrow P_{gcd}$.

In hierarchical development, one decomposes the specification of a system into specifications of its parts. As explained in Section 4, the specification M_{gcd} of program GCD can be written as $M_a \wedge M_b$, where M_a asserts that a initially equals 233344 and is repeatedly decremented by the value of b whenever $a > b$, and where M_b is analogous. The formulas M_a and M_b are the specifications of two processes Π_a and Π_b. We can write Π_a and Π_b as

$$
\begin{array}{ll}
\textsc{Process}\; \Pi_a & \textsc{Process}\; \Pi_b \\
\quad \textbf{output var} \;\; a \;\textbf{initially}\; 233344\,; & \quad \textbf{output var} \;\; b \;\textbf{initially}\; 233577899\,; \\
\quad \textbf{input var} \;\; b\,; & \quad \textbf{input var} \;\; a\,; \\
\quad \textbf{loop} \;\langle\; \textbf{if}\; a > b \;\textbf{then}\; a := a - b \;\rangle & \quad \textbf{loop} \;\langle\; \textbf{if}\; b > a \;\textbf{then}\; b := b - a \;\rangle \\
\quad \textbf{endloop} & \quad \textbf{endloop}
\end{array}
$$

One decomposes a specification in order to refine the components separately. We can refine the GCD program, to remove simultaneous atomic accesses to both a and b, by refining process Π_a to

$$
\begin{array}{l}
\textsc{Process}\; \Pi_a^l \\
\quad \textbf{output var} \;\; a \;\textbf{initially}\; 233344\,; \\
\quad \textbf{internal var} \;\; ai\,; \\
\quad \textbf{input var} \;\; b\,; \\
\quad \textbf{loop} \;\langle\, ai := b \,\rangle\,; \;\; \textbf{if}\; \langle\, a > ai \,\rangle \;\textbf{then}\; \langle\, a := a - ai \,\rangle \;\textbf{endloop}
\end{array}
$$

and refining Π_b to the analogous process Π_b^l.

The composition of processes Π_a^l and Π_b^l correctly implements program GCD. This is expressed in TLA by the assertion that $M_a^l \wedge M_b^l$ implies $M_a \wedge M_b$, where M_a^l and M_b^l are the formulas representing Π_a^l and Π_b^l.

We would like to decompose the proof of $M_a^l \wedge M_b^l \Rightarrow M_a \wedge M_b$ into proofs of $M_a^l \Rightarrow M_a$ and $M_b^l \Rightarrow M_b$. These proofs would show that Π_a^l implements Π_a and that Π_b^l implements Π_b.

	initial state	37 *sent*	37 *acked*	4 *sent*	4 *acked*	19 *sent*	
$c.ack$:	0	0	1	1	0	0	...
$c.sig$:	0	1	1	0	0	1	...
$c.val$:	–	37	37	4	4	19	...

Fig. 1. The two-phase handshake protocol for a channel c.

Unfortunately, Π_a^l does not implement Π_a because, in the absence of assumptions about when its input b can change, Π_a^l can behave in ways that process Π_a cannot. Process Π_a can decrement a only by the current value of b, but Π_a^l can decrement a by a previous value of b if b changes between the assignment to ai and the assignment to a. Similarly, Π_b^l does not implement Π_b.

Process Π_a^l does correctly implement process Π_a in a context in which b does not change when $a > b$. This is expressed in TLA by the formula $E_a \wedge M_a^l \Rightarrow M_a$, where E_a asserts that b does not change when $a > b$. Similarly, $E_b \wedge M_b^l \Rightarrow M_b$ holds, for the analogous E_b. The Decomposition Theorem of Section 4.3 allows us to deduce $M_a^l \wedge M_b^l \Rightarrow M_a \wedge M_b$ from approximately the following hypotheses:

$$\begin{aligned} E_a \wedge M_a^l &\Rightarrow M_a \\ E_b \wedge M_b^l &\Rightarrow M_b \\ M_a \wedge M_b &\Rightarrow E_a \wedge E_b \end{aligned} \tag{1}$$

The third hypothesis holds because the composition of processes Π_a and Π_b does not allow a to change when $b > a$ or b to change when $a > b$.

Observe that E_a asserts only the property of Π_b^l needed to guarantee that Π_a^l implements Π_a. In a more complicated example, E_a will be significantly simpler than M_b^l, the full specification of Π_b^l. Verifying these hypotheses will therefore be easier than proving $M_a^l \wedge M_b^l \Rightarrow M_a \wedge M_b$ directly, since this proof requires reasoning about the specification $M_a^l \wedge M_b^l$ of the complete low-level program.

One cannot really deduce $M_a^l \wedge M_b^l \Rightarrow M_a \wedge M_b$ from the hypotheses (1). For example, (1) is trivially satisfied if E_a, E_b, M_a, and M_b all equal false; but we cannot deduce $M_a^l \wedge M_b^l \Rightarrow \mathsf{false}$ for arbitrary M_a^l and M_b^l. The precise hypotheses of the Decomposition Theorem are more complicated, and we must develop a number of formal concepts in order to state them. We also develop results that allow us to discharge these more complicated hypotheses by proving conditions essentially as simple as (1).

2.2 Composing Open Systems

An open system is one that interacts with an environment it does not control. In our examples, we consider systems that communicate by using a standard two-phase handshake protocol [Mead and Conway 1980] to send values over channels. The state of a channel c is described by three components: the value $c.val$ that is being sent, and two bits $c.sig$ and $c.ack$ used for synchronization. We let $c.snd$ denote the pair $\langle c.sig, c.val \rangle$. Figure 1 shows the sequence of states assumed in sending the sequence of values 37, 4, 19, ... The channel is ready to send when $c.sig = c.ack$. A value v is sent by setting $c.val$ to v and complementing $c.sig$. Receipt of the value is acknowledged by complementing $c.ack$.

We consider an N-element queue with input channel i and output channel o. It

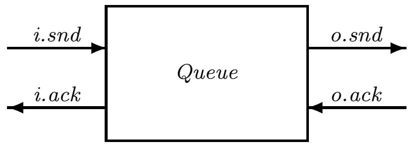


Fig. 2. A queue.

Process Queue

output var $i.ack$, $o.sig$ **initially** 0,
$o.val$;
internal var q **initially** $\langle\,\rangle$;
input var $i.sig$, $i.val$, $o.ack$;
cobegin

$$\textbf{loop}\ \left\langle \begin{array}{l} \textbf{if}\ (i.ack \neq i.sig) \wedge (|q| < N) \\ \quad \textbf{then}\ \ q := q \circ \langle i.val \rangle; \\ \qquad\qquad i.ack := 1 - i.ack \end{array} \right\rangle \textbf{endloop}$$

$$\|$$

$$\textbf{loop}\ \left\langle \begin{array}{l} \textbf{if}\ (o.ack = o.sig) \wedge (|q| > 0) \\ \quad \textbf{then}\ \ o.val := head(q); \\ \qquad\qquad q := tail(q); \\ \qquad\qquad o.sig := 1 - o.sig \end{array} \right\rangle \textbf{endloop}$$

coend

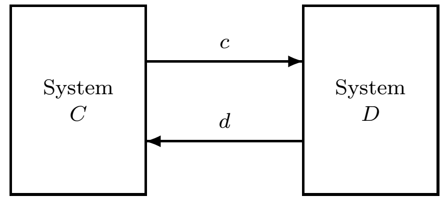
Fig. 3. A queue process.

is depicted in Figure 2. To describe the queue, we use the programming-language constructs introduced in Section 2.1; in particular, we write large atomic actions within angle brackets. We also introduce the following notation for finite sequences: $|\rho|$ denotes the length of sequence ρ, which equals 0 if ρ is empty; $Head(\rho)$ and $Tail(\rho)$ as usual denote the head (first element) and the tail of sequence ρ, if ρ is nonempty; and $\rho \circ \tau$ denotes the concatenation of sequences ρ and τ. Moreover, angle brackets are used to form sequences; so $\langle\,\rangle$ denotes the empty sequence, and $\langle e \rangle$ denotes the sequence with e as its only element. With this notation, the queue can be written as in Figure 3.

Let QM be the TLA formula that represents this queue process. It might seem natural to take QM as the specification of the queue. However, this specification would be difficult or impossible to implement because it states that the queue behaves properly even if the environment does not obey the communication protocol. For example, in a lower-level implementation, reading the input $o.ack$ and setting the outputs $o.sig$ and $o.val$ would be separate actions. If the environment changed $o.ack$ between these actions, the implementation could violate the requirement that it change $o.val$ only when $o.ack = o.sig$. This problem is not an artifact of our particular representation of the queue; actual hardware implementations of a queue can enter metastable states, consequently producing bizarre, unpredictable behavior, if their inputs are changed when they are not supposed to be [Mead and Conway 1980].

A specification of the queue should allow executions in which the queue performs correctly; it should not rule out bad behavior of the queue caused by the environment performing incorrectly. Such a specification can be written in the assumption/guarantee style, a generalization of the traditional pre/postcondition style for sequential programs. An assumption/guarantee specification asserts that the system provides a guarantee M if its environment satisfies an assumption E. For the queue, M is the formula QM, and E asserts that the environment obeys

Fig. 4. A simple example.



the communication protocol.

It is not obvious how to reason about the composition of systems described by assumption/guarantee specifications. The basic problem is illustrated by the simple case of two systems, one guaranteeing M_c assuming M_d, and the other guaranteeing M_d assuming M_c. Since each system guarantees to satisfy the other's environment assumption, we would like to conclude that their composition implements the specification $M_c \wedge M_d$ unconditionally, with no environment assumption. Can we? We attempt to answer this question by considering two simple examples, based on Figure 4.

In the first example:

—M_c^0 asserts that c always equals 0.
—M_d^0 asserts that d always equals 0.

We can implement these specifications with the following two processes.

Process Π_c	Process Π_d
output var c **initially** 0;	**output var** d **initially** 0;
input var d;	**input var** c;
loop $\langle c := d \rangle$ **endloop**	**loop** $\langle d := c \rangle$ **endloop**

Process Π_c guarantees M_c^0 assuming M_d^0, and process Π_d guarantees M_d^0 assuming M_c^0. Clearly, their composition leaves c and d unchanged, so it implements $M_c^0 \wedge M_d^0$.

In the second example:

—M_c^1 asserts that c eventually equals 1.
—M_d^1 asserts that d eventually equals 1.

The same processes Π_c and Π_d implement the specifications in this case too; process Π_c guarantees M_c^1 assuming M_d^1, and process Π_d guarantees M_d^1 assuming M_c^1. However, since their composition leaves c and d unchanged, it does not implement $M_c^1 \wedge M_d^1$.

Our conclusion in the first example does not depend on the particular choice of processes Π_c and Π_d. We can deduce directly from the assumption/guarantee specifications that the composition must implement $M_c^0 \wedge M_d^0$, because the first process to change its output variable would violate its guarantee before its assumption had been violated. This argument does not apply to the second example, because violating M_c^1 and M_d^1 are sins of omission that do not occur at any particular instant. A property that can be made false only by being violated at some instant is called a safety property [Alpern and Schneider 1985]. As the examples suggest, reasoning about the composition of assumption/guarantee specifications is easiest when assumptions are safety properties.

The argument that the composition should implement $M_c^0 \wedge M_d^0$ in the first example rests on the requirement that a process maintains its guarantee until after the environment violates its assumption. In other words, we interpret the assumption/guarantee specification as an assertion that the guarantee M can become false only after the assumption E becomes false. We write this assertion as the formula $E \stackrel{+}{\Rightarrow} M$. Section 5 discusses this form of specification.

Our rules for reasoning about the composition of assumption/guarantee specifications are embodied in the Composition Theorem of Section 5.2. With the Composition Theorem, we can prove that the conjunction of the assumption/guarantee specifications $M_c^0 \stackrel{+}{\Rightarrow} M_d^0$ and $M_d^0 \stackrel{+}{\Rightarrow} M_c^0$ implies $M_c^0 \wedge M_d^0$. We can also prove more substantial results—for example, that the composition of queues implements a larger queue. Verifying the hypotheses of the theorem requires reasoning only about complete systems, so the theorem allows us to handle assumption/guarantee specifications as easily as complete-system specifications.

3. PRELIMINARIES

3.1 TLA

3.1.1 *Review of the Syntax and Semantics.* A state is an assignment of values to variables. (Technically, our variables are the "flexible" variables of temporal logic that correspond to the variables of programming languages; they are distinct from the variables of first-order logic.) A behavior is an infinite sequence of states. Semantically, a TLA formula F is true or false of a behavior; we say that F is *valid*, and write $\models F$, iff it is true of every behavior. Syntactically, TLA formulas are built up from state functions using Boolean operators ($\neg$, $\wedge$, $\vee$, $\Rightarrow$ [implication], and $=$ [equivalence]) and the operators $'$, $\Box$, and $\boldsymbol{\exists}$, as described below.

A *state function* is like an expression in a programming language. Semantically, it assigns a value to each state—for example $3 + x$ assigns to state s three plus the value of the variable x in s. A *state predicate* is a Boolean-valued state function. An *action* is a Boolean-valued expression containing primed and unprimed variables. Semantically, an action is true or false of a pair of states, with primed variables referring to the second state—for example, $x + 1 > y'$ is true for $\langle s, t\rangle$ iff the value of $x + 1$ in s is greater than the value of y in t. A pair of states satisfying action $\mathcal{A}$ is called an $\mathcal{A}$ *step.* We say that $\mathcal{A}$ is *enabled* in state s iff there exists a state t such that $\langle s, t\rangle$ is an $\mathcal{A}$ step—for example, $(x > 0) \wedge (x + 1 > y')$ is enabled only in states where $x > 0$. The state predicate *Enabled* $\mathcal{A}$ is true for state s iff $\mathcal{A}$ is enabled in s. We write v' for the expression obtained by priming all the variables of the state function v, and $[\mathcal{A}]_v$ for $\mathcal{A} \vee (v' = v)$, so an $[\mathcal{A}]_v$ step is either an $\mathcal{A}$ step or a step that leaves v unchanged.

As usual in temporal logic, if F is a formula then $\Box F$ is a formula that means that F is always true, and $\Diamond F$, an abbreviation for $\neg\Box\neg F$, means that F is eventually true. In addition, if $\mathcal{A}$ is an action and v is a state function then $\Box[\mathcal{A}]_v$ is a formula; $\Diamond\langle\mathcal{A}\rangle_v$ is an abbreviation for $\neg\Box[\neg A]_v$. Using $\Box$ and "enabled" predicates, we can define fairness operators WF and SF. The *weak-fairness* formula $\mathrm{WF}_v(\mathcal{A})$ asserts of a behavior that either there are infinitely many $\mathcal{A}$ steps that change v, or there are infinitely many states in which such steps are not enabled. This can be written $(\Box\Diamond\langle\mathcal{A}\rangle_v) \vee (\Box\Diamond\neg \mathit{Enabled}\ \langle\mathcal{A}\rangle_v)$. The *strong-fairness* formula $\mathrm{SF}_v(\mathcal{A})$ asserts that either there are infinitely many $\mathcal{A}$ steps that change v, or there are only finitely many states in which such steps are enabled. This can be written

$(\Box\Diamond\langle\mathcal{A}\rangle_v) \vee (\Diamond\Box\neg \mathit{Enabled}\ \langle\mathcal{A}\rangle_v)$.

The formula $\exists x : F$ means essentially that there is some way of choosing a sequence of values for x such that the temporal formula F holds. We think of $\exists x : F$ as "F with x hidden" and call x an internal variable of $\exists x : F$. Both x and x' are bound by $\exists x$ in $\exists x : F$. If x is a tuple of variables $\langle x_1, \ldots, x_k\rangle$, we write $\exists x : F$ for $\exists x_1 : \ldots \exists x_k : F$.

The standard way of specifying a system in TLA is with a formula in the "canonical form" $\exists x : \mathit{Init} \wedge \Box[\mathcal{N}]_v \wedge L$, where Init is a predicate and L a conjunction of fairness conditions. This formula asserts that there exists a sequence of values for x such that (1) Init is true for the initial state, (2) every step of the behavior is an $\mathcal{N}$ step or leaves the state function v unchanged, and (3) L holds. For example, the specification M_{gcd} of the complete high-level GCD program is written in canonical form by taking[1]

$$\begin{array}{lcl}
\mathit{Init} & \triangleq & (a = 233344) \wedge (b = 233577899) \\
\mathcal{N} & \triangleq & \vee\ (a > b)\ \wedge\ (a' = a - b)\ \wedge\ (b' = b) \\
 & & \vee\ (b > a)\ \wedge\ (b' = b - a)\ \wedge\ (a' = a) \\
v & \triangleq & \langle a, b\rangle \\
L & \triangleq & \mathrm{WF}_v(\mathcal{N})
\end{array} \qquad (2)$$

Intuitively, a variable represents some part of the universe, and a behavior represents a possible complete history of the universe. A system Π is represented by a TLA formula M that is true for precisely those behaviors that represent histories in which Π is running. We make no formal distinction between systems, specifications, and properties; they are all represented by TLA formulas, which we usually call specifications.

3.1.2 *Interleaving and Noninterleaving Representations.* Let ξ and ψ be two objects, represented by the variables x and y, respectively. When representing a history of the universe as a behavior, we can describe concurrent changes to ξ and ψ either by a single simultaneous change to x and y, or by separate changes to x and y in some order. If the changes to ξ and ψ are directly linked, then it is usually most convenient to describe their concurrent change by a single change to both x and y. However, if the changes are independent, then we are free to choose whether or not to allow simultaneous changes to x and y. An *interleaving* representation is one in which such simultaneous changes are disallowed.

When changes to ξ and ψ are directly linked, we often think of x and y as output variables of a single component. An interleaving representation is then one in which simultaneous changes to output variables of different processes are disallowed. The absence of such simultaneous changes can be expressed as a TLA formula. For a system with n components in which v_i is the tuple of output variables of component i, interleaving is expressed by the formula

$$\mathit{Disjoint}(v_1, \ldots, v_n) \ \triangleq\ \bigwedge_{i \neq j} \Box[(v_i' = v_i) \vee (v_j' = v_j)]_{\langle v_i,\, v_j\rangle}$$

[1]We let a list of formulas bulleted with $\wedge$ or $\vee$ denote the conjunction or disjunction of the formulas, using indentation to eliminate parentheses. We also let $\Rightarrow$ have lower precedence than the other Boolean operators.

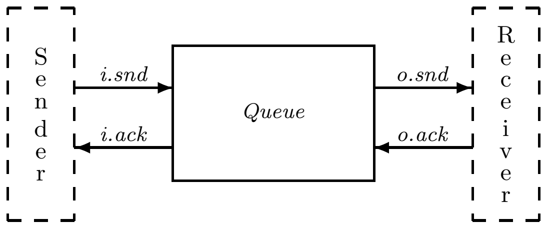


Fig. 5. The complete system of queue plus environment.

We have found that, in TLA, interleaving representations are usually easier to write and to reason about. Moreover, an interleaving representation is adequate for reasoning about a system if the system is modeled at a sufficiently fine grain of atomicity. However, as discussed below, TLA also works for noninterleaving representations. TLA does not mandate any particular method for representing systems. Indeed, one can write specifications that are intermediate between interleaving and noninterleaving representations.

3.1.3 *The Queue Example.* We now give a TLA specification of the queue of natural numbers of length N, which was described informally in Section 2.2 and illustrated in Figure 2. As in Section 2.2, we write $c.snd$ for the pair $\langle c.sig,\ c.val\rangle$ for a channel c; we also write c for the triple $\langle c.sig,\ c.ack,\ c.val\rangle$.

A channel is initially ready for sending, so the initial condition on wire c is the predicate $CInit(c)$ defined by

$$CInit(c) \;\triangleq\; (c.sig = c.ack = 0)$$

The operations of sending a value v and acknowledging receipt of a value on channel c are represented by the following $Send(\mathsf{v}, c)$ and $Ack(c)$ actions.

$$Send(\mathsf{v}, c) \;\triangleq\; \begin{array}{l} \land\ c.sig = c.ack \\ \land\ c.snd' = \langle 1 - c.sig,\ \mathsf{v}\rangle \\ \land\ c.ack' = c.ack \end{array} \qquad Ack(c) \;\triangleq\; \begin{array}{l} \land\ c.sig \neq c.ack \\ \land\ c.ack' = 1 - c.ack \\ \land\ c.snd' = c.snd \end{array}$$

To represent the queue as a complete system, we add an environment that sends arbitrary natural numbers over channel i and acknowledges receipt of values on channel o. The resulting complete system is shown in Figure 5.

The TLA formula CQ specifying the queue is defined in Figure 6. It has the canonical form $\exists x : Init \land \Box[\mathcal{N}]_v \land L$, where:

—x is the internal variable q, which represents the sequence of values received on the input channel i but not yet sent on the output channel o.

—$Init$ is written as the conjunction $Init_E \land Init_M$ of initial predicates for the environment and component. (We arbitrarily consider the initial conditions on a channel to be part of the sender's initial predicate.)

—$\mathcal{N}$ is the disjunction of two actions: $\mathcal{Q}_M$, describing the steps taken by the component, and $\mathcal{Q}_E \land (q' = q)$, describing steps taken by the environment (which leave q unchanged). Action $\mathcal{Q}_M$ is the disjunction of actions Enq and Deq. An Enq step acknowledges receipt of a value on i and appends the value to q; it is enabled only when q has fewer than N elements. A Deq step removes the first element of q and sends it on o. Action $\mathcal{Q}_E$ is the disjunction of Put, which

$$
\begin{array}{lcll}
Init_E & \triangleq & CInit(i) & \text{Environment Actions} \\
Put & \triangleq & (\exists v \in \mathsf{Nat} : Send(v, i)) \ \land \ (o' = o) & \\
Get & \triangleq & Ack(o) \ \land \ (i' = i) & \\
\mathcal{Q}_E & \triangleq & Get \ \lor \ Put & \\
\hline
Init_M & \triangleq & CInit(o) \ \land \ (q = \langle \, \rangle) & \text{Component Actions} \\
Enq & \triangleq & \begin{array}{l} \land \ |q| < N \\ \land \ Ack(i) \ \ \land \ \ (q' = q \circ \langle i.val \rangle) \\ \land \ o' = o \end{array} & \\
Deq & \triangleq & \begin{array}{l} \land \ |q| > 0 \\ \land \ Send(Head(q), o) \ \ \land \ \ (q' = Tail(q)) \\ \land \ i' = i \end{array} & \\
\mathcal{Q}_M & \triangleq & Enq \ \lor \ Deq & \\
\hline
ICL & \triangleq & \mathrm{WF}_{\langle i,\, o,\, q \rangle}(\mathcal{Q}_M) & \text{Complete-System Specification} \\
ICQ & \triangleq & \begin{array}{l} \land \ Init_E \ \land \ Init_M \\ \land \ \Box \begin{bmatrix} \lor \ \mathcal{Q}_E \ \land \ (q' = q) \\ \lor \ \mathcal{Q}_M \end{bmatrix}_{\langle i,\, o,\, q \rangle} \\ \land \ ICL \end{array} & \\
CQ & \triangleq & \boldsymbol{\exists}\, q : \ ICQ &
\end{array}
$$

Fig. 6. The specification CQ of the complete queue. (Formulas $CInit$, $Send$, and Ack are defined in the text.)

sends an arbitrary number on channel i, and Get, which acknowledges receipt of a number on channel o.

—v is the tuple $\langle i,\, o,\, q \rangle$ of all relevant variables.[2]

—L is the weak-fairness condition ICL, which is defined to be $\mathrm{WF}_{\langle i,\, o,\, q \rangle}(\mathcal{Q}_M)$, and asserts that a component step cannot remain forever possible without occurring. It can be shown that a logically equivalent specification is obtained if this condition is replaced with $\mathrm{WF}_{\langle i,\, o,\, q \rangle}(Enq) \land \mathrm{WF}_{\langle i,\, o,\, q \rangle}(Deq)$.

Formula CQ gives an interleaving representation of a queue; simultaneous steps by the queue and its environment are not allowed. Moreover, simultaneous changes to the two inputs $i.snd$ and $o.ack$ are disallowed, as are simultaneous changes to the two outputs $i.ack$ and $o.snd$. In Section 4, we describe a noninterleaving representation of the queue.

3.2 Implementation

A specification M^l implies a specification M iff every behavior that satisfies M^l also satisfies M; hence proving $M^l \Rightarrow M$ shows that the system Π^l represented by M^l implements the system or property Π represented by M. Note that if M^l is inconsistent (equivalent to false), then $M^l \Rightarrow M$ holds vacuously, but an inconsistent M^l does not represent any system Π^l.

The formula $M^l \Rightarrow M$ is proved by applying a handful of simple rules [Lamport 1994]. When M has the form $\boldsymbol{\exists}\, x : \widehat{M}$, a key step in the proof is finding a *refinement*

[2]Informally, we write $\langle i,\, o,\, q \rangle$ for the concatenation of the tuples i, o, and $\langle q \rangle$.

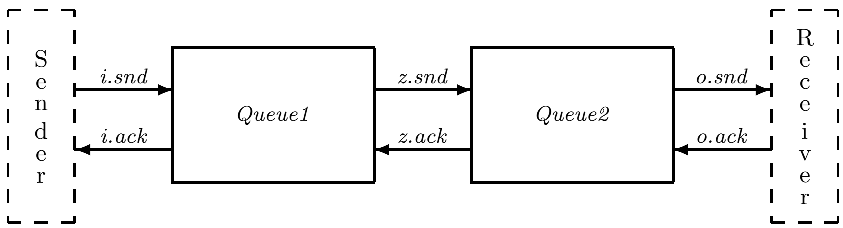


Fig. 7. A complete system containing two queues in series.

$$
\begin{array}{lcl}
ICDQ & \triangleq & \land\ Init_E \ \land\ Init_M^{[1]} \ \land\ Init_M^{[2]} \\
 & & \land\ \Box \begin{bmatrix} \lor\ \mathcal{Q}_E \ \land \langle q_1,\, q_2,\, z\rangle' = \langle q_1,\, q_2,\, z\rangle \\ \lor\ \mathcal{Q}_M^{[1]} \land \langle q_2,\, o\rangle' = \langle q_2,\, o\rangle \\ \lor\ \mathcal{Q}_M^{[2]} \land \langle q_1,\, i\rangle' = \langle q_1,\, i\rangle \end{bmatrix}_{\langle i,\, o,\, z,\, q_1,\, q_2\rangle} \\
 & & \land\ ICL^{[1]} \ \land\ ICL^{[2]} \\
CDQ & \triangleq & \exists\, q_1,\, q_2 : ICDQ
\end{array}
$$

Fig. 8. Specification of the complete double-queue system of Figure 7.

mapping—a tuple of state functions $\overline{x}$ such that M^l implies $\overline{\widehat{M}}$, where $\overline{\widehat{M}}$ is the formula obtained by substituting $\overline{x}$ for x (and therefore $(\overline{x})'$ for x') in $\widehat{M}$. Under reasonable assumptions, such a refinement mapping exists when $M^l \Rightarrow \exists\, x : \widehat{M}$ is valid [Abadi and Lamport 1991].

As an example, we show that the system composed of two queues in series, shown in Figure 7, implements a single larger queue. We first specify the composite queue. Let $F[e_1/v_1, \ldots, e_n/v_n]$ denote the result of (simultaneously) substituting each expression e_i for v_i in a formula F. For example, if *Get* is defined as in Figure 6, then $Get[z/i]$ equals $Ack(o) \land (z' = z)$. For any formula F, let

$$F^{[1]} \ \triangleq\ F[z/o,\, q_1/q] \qquad\qquad F^{[2]} \ \triangleq\ F[z/i,\, q_2/q]$$

In Figure 8, the specification *CDQ* of the complete system, consisting of the double queue and its environment, is defined in terms of the formulas from Figure 6. We think of the complete system as containing three components: the environment and the two queues. The initial condition is the conjunction of the initial conditions of each component. The next-state action consists of three disjuncts, representing actions of each of the three components that leave other components' variables unchanged. Finally, we take as the liveness condition the conjunction of the fairness conditions of the two queues.

We now show that the composite queue implements a $(2N+1)$-element queue. (The "+1" arises because the internal channel z acts as a buffer element.) The correctness condition is $CDQ \Rightarrow CQ^{[\mathrm{dbl}]}$, where $F^{[\mathrm{dbl}]}$ denotes $F[(2N+1)/N]$, for any formula F. This is proved by showing $ICDQ \Rightarrow \overline{ICQ^{[\mathrm{dbl}]}}$, with the refinement mapping defined by

$$\overline{q} \;\triangleq\; \textbf{if}\ z.sig = z.ack\ \textbf{then}\ q_1 \circ q_2 \ \textbf{else}\ q_1 \circ \langle z.val \rangle \circ q_2$$

The formula $ICDQ \Rightarrow \overline{ICQ^{[\mathrm{dbl}]}}$ can be proved by standard TLA reasoning of the kind described by Lamport [1994].

3.3 Conditional Implementation

Instead of proving that a specification M^l implements a specification M, we sometimes want to prove the weaker condition that M^l implements M assuming a formula G. In other words, we want to prove $G \Rightarrow (M^l \Rightarrow M)$, which is equivalent to $G \wedge M^l \Rightarrow M$. The formula G may express one or more of the following:

—A law of nature. For example, in a real-time specification, G might assert that time increases monotonically. If the current time is represented by the variable *now*, this assumption is expressed by the formula $(now \in \mathbf{R}) \wedge \Box[now' \in (now, \infty)]_{now}$, where $\mathbf{R}$ is the set of real numbers.

—An interface refinement, where G expresses the relation between a low-level tuple l of variables and its high-level representation as a tuple h of variables. For example, l might be a low-level interface representing the transmission of sequences of bits over a wire, and h could be the high-level interface in which the sending of seven successive bits is interpreted as the transmission of a single ASCII character.

—An assumption about how reality is translated into the formalism of behaviors. In particular, G may assert an interleaving assumption—for example, an assumption of the form $Disjoint(v_1, \ldots, v_n)$.

Conditional implementation, with an explicit formula G, is needed only for open systems. For a complete system, the properties expressed by G can easily be made part of the system specification. For example, the system can include a component that advances time. In contrast, it can be difficult to include G in the specification of an open system.

3.4 Safety and Closure

3.4.1 *Definition of Closure.* A finite sequence of states is called a finite behavior. For any formula F and finite behavior ρ, we say that ρ satisfies F iff ρ can be extended to an infinite behavior that satisfies F. For convenience, we say that the empty sequence $\langle\,\rangle$ satisfies every formula (even false).

A *safety property* is a formula that is satisfied by an infinite behavior σ iff it is satisfied by every prefix of σ [Alpern and Schneider 1985]. For any predicate *Init*, action $\mathcal{N}$, and state function v, the formula $Init \wedge \Box[\mathcal{N}]_v$ is a safety property. It can be shown that, for any TLA formula F, there is a TLA formula $\mathcal{C}(F)$, called the *closure of* F, such that a behavior σ satisfies $\mathcal{C}(F)$ iff every prefix of σ satisfies F. Formula $\mathcal{C}(F)$ is the strongest safety property such that $\models F \Rightarrow \mathcal{C}(F)$. Proposition 1 below implies that $\mathcal{C}(Init \wedge \Box[\mathcal{N}]_v \wedge L)$ equals $Init \wedge \Box[\mathcal{N}]_v$ when L is the conjunction of suitable fairness properties.

3.4.2 *Machine Closure.* When writing a specification in the form $Init \wedge \Box[\mathcal{N}]_v \wedge L$, we expect L to constrain infinite behaviors, not finite ones. Formally, this means that the closure of $Init \wedge \Box[\mathcal{N}]_v \wedge L$ should be $Init \wedge \Box[\mathcal{N}]_v$. A pair of properties (P, L) is called *machine closed* iff $\mathcal{C}(P \wedge L)$ equals P [Abadi and Lamport 1991]. (We often say informally that $P \wedge L$ is machine closed.)

Proposition 1 below, which we have already proved [Abadi and Lamport 1994], shows that we can use fairness properties to write machine-closed specifications. The proposition relies on the following definition: an action $\mathcal{A}$ is a *subaction* of a safety property P iff for every finite behavior $\rho = \langle r_0, \ldots, r_n \rangle$, if ρ satisfies P and $\mathcal{A}$ is enabled in state r_n, then there exists a state r_{n+1} such that $\langle r_0, \ldots, r_{n+1} \rangle$ satisfies P and $\langle r_n, r_{n+1} \rangle$ is an $\mathcal{A}$ step. It follows from this definition of subaction that, if $\mathcal{A}$ implies $\mathcal{N}$, then $\mathcal{A}$ is a subaction of $Init \wedge \Box[\mathcal{N}]_v$.

PROPOSITION 1. *If P is a safety property and L is the conjunction of a countable number of formulas of the form $\mathrm{WF}_w(\mathcal{A})$ and/or $\mathrm{SF}_w(\mathcal{A})$ such that $\mathcal{A} \wedge (w' \neq w)$ is a subaction of P, then (P, L) is machine closed.*

3.4.3 *Closure and Hiding.* Several of our results have hypotheses of the form $\mathcal{C}(M_1) \wedge \ldots \wedge \mathcal{C}(M_n) \Rightarrow \mathcal{C}(M)$. The obvious first step in proving such a formula is to compute the closures $\mathcal{C}(M_1), \ldots, \mathcal{C}(M_n)$, and $\mathcal{C}(M)$. We can use Proposition 1 to compute the closure of a formula with no internal variables. When there are internal variables, the following proposition allows us to reduce the proof of $\mathcal{C}(M_1) \wedge \ldots \wedge \mathcal{C}(M_n) \Rightarrow \mathcal{C}(M)$ to the proof of a formula in which the closures can be computed with Proposition 1.

PROPOSITION 2. *Let x, x_1, $\ldots$, x_n be tuples of variables such that for each i, no variable in x_i occurs in M or in any M_j with $i \neq j$.*

$$\textit{If } \models \bigwedge_{i=1}^{n} \mathcal{C}(M_i) \;\Rightarrow\; \exists\, x : \mathcal{C}(M), \textit{ then } \models \bigwedge_{i=1}^{n} \mathcal{C}(\exists\, x_i : M_i) \;\Rightarrow\; \mathcal{C}(\exists x : M).$$

Proofs are in the appendix.

Some of our results also have hypotheses of the form $\mathcal{C}(M_1) \wedge \ldots \wedge \mathcal{C}(M_n) \Rightarrow E$, where we expect E to be a safety property. If we can verify that E is a safety property, so $\models E = \mathcal{C}(E)$, then we can apply Proposition 2. When E has internal variables, we can often use Proposition 2 of [Abadi and Lamport 1991] to verify that E is a safety property.

3.5 Additional Temporal Operators

We now define some additional temporal operators. Although they can be expressed in terms of the primitive TLA operations $'$, $\Box$, and $\exists$, we define them semantically.

3.5.1 *The + Operator.* The formula E_{+v} asserts that, if the temporal formula E ever becomes false, then the state function v stops changing. More precisely, a behavior σ satisfies E_{+v} iff either σ satisfies E, or there is some n such that (1) E holds for the first n states of σ and (2) v never changes from the $(n+1)$st state on. When E is a safety property in canonical form, it is easy to write E_{+v} explicitly:

PROPOSITION 3. *If x is a tuple of variables none of which occurs in v, and s is a variable that does not occur in Init, $\mathcal{N}$, w, v, or x, and*

$$\begin{array}{lll} \widehat{Init} & \triangleq & (Init \wedge (s=0)) \vee (\neg Init \wedge (s=1)) \\ \widehat{\mathcal{N}} & \triangleq & \vee\, (s=0) \wedge \vee\, (s'=0) \wedge (\mathcal{N} \vee (w'=w)) \\ & & \phantom{\vee\, (s=0) \wedge{}} \vee\, (s'=1) \wedge \neg(\mathcal{N} \vee (w'=w)) \\ & & \vee\, (s=1) \wedge (s'=1) \wedge (v'=v) \end{array}$$

then $\models (\exists\, x : Init \wedge \Box[\mathcal{N}]_w)_{+v} \;=\; \exists\, x, s : \widehat{Init} \wedge \Box[\widehat{\mathcal{N}}]_{\langle w,\, v,\, s\rangle}$.

We need to reason about $+$ only to verify hypotheses of the form $\models \mathcal{C}(E)_{+v} \wedge \mathcal{C}(M^l) \Rightarrow \mathcal{C}(M)$ in our Decomposition and Composition Theorems. We can verify such a hypothesis by first applying the observation that $\mathcal{C}(E)_{+v}$ equals $\mathcal{C}(E_{+v})$ and using Proposition 3 to calculate E_{+v}. However, this approach is necessary only for noninterleaving specifications. Proposition 4 below provides a way of proving these hypotheses for interleaving specifications without having to calculate E_{+v}.

3.5.2 *The $-\!\triangleright$ Operator.* For temporal formulas E and M, the formula $E -\!\triangleright M$ asserts that M holds at least as long as E does [Abadi and Plotkin 1993]. More precisely $E -\!\triangleright M$ is true of a behavior σ iff $E \Rightarrow M$ is true of σ and, for every finite prefix ρ of σ, if E is true of ρ then M is true of ρ. It follows from this definition of $-\!\triangleright$ that $E -\!\triangleright M$ equals $(\mathcal{C}(E) -\!\triangleright \mathcal{C}(M)) \wedge (E \Rightarrow M)$. The operator $-\!\triangleright$ acts much like ordinary implication. In fact, $\models E -\!\triangleright M$ is equivalent to $\models E \Rightarrow M$. Of course, it is not in general true that $\models (E -\!\triangleright M) = (E \Rightarrow M)$.

3.5.3 *The $\stackrel{+}{-\!\triangleright}$ Operator.* As we observed in the introduction, we interpret the specification that M is guaranteed under assumption E as the formula $E \stackrel{+}{-\!\triangleright} M$, which means that M holds at least one step longer than E does. More precisely, $E \stackrel{+}{-\!\triangleright} M$ is true of a behavior σ iff $E \Rightarrow M$ is true of σ and, for every $n \geq 0$, if E holds for the first n states of σ, then M holds for the first $n{+}1$ states of σ. It follows from this definition of $\stackrel{+}{-\!\triangleright}$ that $E \stackrel{+}{-\!\triangleright} M$ equals $(\mathcal{C}(E) \stackrel{+}{-\!\triangleright} \mathcal{C}(M)) \wedge (E \Rightarrow M)$.

The formula $E \stackrel{+}{-\!\triangleright} M$ is stronger than $E -\!\triangleright M$, which asserts that M holds as long as E does. It can be shown that, if E is a safety property, then $E \stackrel{+}{-\!\triangleright} M$ equals $(M -\!\triangleright E) -\!\triangleright M$. We prove in the appendix that, if E and M are both safety properties and v is a tuple of variables containing all free variables of M, then $E \stackrel{+}{-\!\triangleright} M$ equals $E_{+v} -\!\triangleright M$.

3.5.4 *The $\perp$ Operator.* The specification M of a component can be made false only by a step that changes the component's output variables. In an interleaving representation, we do not allow a single step to change output variables of two different components. Hence, if E and M are specifications of separate components, we expect that no step will make both E and M false. More precisely, we expect E and M to be orthogonal ($\perp$), where $E \perp M$ is true of a behavior σ iff there is no $n \geq 0$ such that E and M are both true for the first n states of σ and both false for the first $n{+}1$ states of σ. It can be shown that $E \perp M$ equals $\mathcal{C}(E) \perp \mathcal{C}(M)$, and that, if E and M are safety properties, then $E \perp M$ equals $(E \wedge M) \stackrel{+}{-\!\triangleright} (E \vee M)$.

If no step falsifies both E and M, and M remains true as long as E does, then M must remain true at least one step longer than E does. Hence, $E \perp M$ implies the equivalence of $E -\!\triangleright M$ and $E \stackrel{+}{-\!\triangleright} M$. In fact, we prove in the appendix that

$(E \stackrel{+}{\rightarrowtriangle} M) = (E \rightarrowtriangle M) \wedge (E \perp M)$ is valid. From this and the relation between $\stackrel{+}{\rightarrowtriangle}$ and $+$, we can derive:

PROPOSITION 4. *If E, M, and R are safety properties, and v is a tuple of variables containing all variables that occur free in M, then $\models E \wedge R \Rightarrow M$ and $\models R \Rightarrow E \perp M$ imply $\models E_{+v} \wedge R \Rightarrow M$.*

This proposition enables us to use orthogonality to remove $+$ from proof obligations. To apply the proposition, we must prove the orthogonality of component specifications. We do this for interleaving specifications with the following result.

PROPOSITION 5. *If*

$$\begin{aligned} &\models \mathcal{C}(E) \;=\; \mathit{Init}_E \;\wedge\; \Box[\mathcal{N}_E]_{\langle x,\, e\rangle} \\ &\models \mathcal{C}(M) \;=\; \mathit{Init}_M \;\wedge\; \Box[\mathcal{N}_M]_{\langle y,\, m\rangle} \end{aligned}$$

then

$$\models (\exists x : \mathit{Init}_E \vee \exists y : \mathit{Init}_M) \wedge \mathit{Disjoint}(e, m) \;\Rightarrow\; \mathcal{C}(\boldsymbol{\exists} x : E) \perp \mathcal{C}(\boldsymbol{\exists} y : M)$$

4. DECOMPOSING A COMPLETE SPECIFICATION

4.1 Specifying a Component

Let us consider how to write the specification M of one component of a larger system. We assume that the free variables of the specification can be partitioned into tuples m of output variables and e of input variables; the component changes the values of the variables of m only. (A more general situation is discussed below.) The specification of a component has the same form $\boldsymbol{\exists} x : \mathit{Init} \wedge \Box[\mathcal{N}]_v \wedge L$ as that of a complete system. For a component specification:

—v is the tuple $\langle x, m, e\rangle$.

—*Init* describes the initial values of the component's output variables m and internal variables x.

—$\mathcal{N}$ should allow two kinds of steps—ones that the component performs and ones that its environment performs. Steps performed by the component, which change its output variables m, are described by an action $\mathcal{N}_m$. In an interleaving representation, the component's inputs and outputs cannot change simultaneously, so $\mathcal{N}_m$ implies $e' = e$. In a noninterleaving representation, $\mathcal{N}_m$ does not constrain the value of e', so the variables of e do not appear primed in $\mathcal{N}_m$. In either case, we are specifying the component but not its environment, so we let the environment do anything except change the component's output variables or internal variables. In other words, the environment is allowed to perform any step in which $\langle m, x\rangle'$ equals $\langle m, x\rangle$. (Below, we describe more general specifications in which an environment action can change x.) Therefore, $\mathcal{N}$ should equal $\mathcal{N}_m \vee (\langle m, x\rangle' = \langle m, x\rangle)$.

—L is the conjunction of fairness conditions, each of the form $\mathrm{WF}_{\langle m, x\rangle}(\mathcal{A})$ or $\mathrm{SF}_{\langle m, x\rangle}(\mathcal{A})$. For an interleaving representation, which by definition does not allow steps that change both e and m, the subscripts $\langle m, x\rangle$ and $\langle e, m, x\rangle$ yield equivalent fairness conditions.

This leads us to write M in the form

$$M \;\triangleq\; \exists x : \mathit{Init} \;\wedge\; \Box[\mathcal{N}_m \vee (\langle m, x\rangle' = \langle m, x\rangle)]_{\langle e,\, m,\, x\rangle} \;\wedge\; L \tag{3}$$

By simple logic, (3) is equivalent to

$$M \;\triangleq\; \exists\, x : \mathit{Init} \;\wedge\; \Box[\mathcal{N}_m]_{\langle m,\, x\rangle} \;\wedge\; L \tag{4}$$

For the specification M_a of process Π_a in the GCD example, x is the empty tuple (there is no internal variable), the input variable e is b, the output variable m is a, and

$$\begin{array}{lcl} \mathit{Init}_a & \triangleq & a = 233344 \\ \mathcal{N}_a & \triangleq & (a > b) \;\wedge\; (a' = a - b) \;\wedge\; (b' = b) \\ M_a & \triangleq & \mathit{Init}_a \;\wedge\; \Box[\mathcal{N}_a]_a \;\wedge\; \mathrm{WF}_a(\mathcal{N}_a) \end{array} \tag{5}$$

For the specification M_a^l of the low-level process Π_a^l, the tuple x is $\langle ai, pca\rangle$, where pca is an internal variable that tells whether control is at the beginning of the loop or after the assignment to ai. The specification has the form

$$M_a^l \;\triangleq\; \exists\, ai, pca : \mathit{Init}_a^l \;\wedge\; \Box[\mathcal{N}_a^l]_{\langle a,\, ai,\, pca\rangle} \;\wedge\; \mathrm{WF}_{\langle a,\, ai,\, pca\rangle}(\mathcal{N}_a^l) \tag{6}$$

for appropriate initial condition Init_a^l and next-state action $\mathcal{N}_a^l$. The specifications M_b and M_b^l are similar.

In our queue example, we can write the specifications of both the queue and its environment as separate components in the form (4). For the queue component, the tuple m of output variables is $\langle i.ack, o.snd\rangle$, the tuple e of input variables is $\langle i.snd, o.ack\rangle$, and the specification is

$$\begin{array}{lcl} \mathit{IQM} & \triangleq & \mathit{Init}_M \;\wedge\; \Box[\mathcal{Q}_M]_{\langle i.ack,\, o.snd,\, q\rangle} \;\wedge\; \mathit{ICL} \\ \mathit{QM} & \triangleq & \exists\, q : \mathit{IQM} \end{array} \tag{7}$$

The specification of the environment as a separate component is

$$\mathit{QE} \;\triangleq\; \mathit{Init}_E \;\wedge\; \Box\,[\mathcal{Q}_E]_{\langle i.snd,\, o.ack\rangle} \tag{8}$$

We have provided specifications of the queue and its environment in an interleaving representation. A noninterleaving representation of the queue can be obtained by modifying its specification as follows.

—Change the *Enq* and *Deq* actions so they do not constrain the values of $i.snd'$ or $o.ack'$.

—Define an action *DeqEnq* that simultaneously enqueues an input value and dequeues an output value, and change the definition of $\mathcal{Q}_M$ to have *DeqEnq* as an additional disjunct.

The resulting specification QM^{ni} is given in Figure 9. It is a noninterleaving specification because it allows a step that changes i and o simultaneously. A noninterleaving representation of the queue's environment can be obtained in a similar fashion.

In describing the component's next-state action $\mathcal{N}$, we required that an environment action not change the component's internal variables. One can also write a

$$
\begin{array}{lcl}
Init_M & \triangleq & CInit(o) \;\wedge\; (q = \langle\,\rangle) \\
Enq^{ni} & \triangleq & \wedge\; |q| < N \\
 & & \wedge\; Ack(i) \;\wedge\; (q' = q \circ \langle i.val \rangle) \\
 & & \wedge\; o.snd' = o.snd \\
Deq^{ni} & \triangleq & \wedge\; |q| > 0 \\
 & & \wedge\; Send(Head(q), o) \;\wedge\; (q' = Tail(q)) \\
 & & \wedge\; i.ack' = i.ack \\
DeqEnq^{ni} & \triangleq & \wedge\; (|q| > 0) \;\wedge\; Send(Head(q), o) \\
 & & \wedge\; Ack(i) \\
 & & \wedge\; q' = Tail(q) \circ \langle i.val \rangle \\
\mathcal{Q}_M^{ni} & \triangleq & Enq^{ni} \;\vee\; Deq^{ni} \;\vee\; DeqEnq^{ni} \\
IQM^{ni} & \triangleq & Init_M \;\wedge\; \Box[\mathcal{Q}_M^{ni}]_{\langle i.ack,\, o.snd,\, q\rangle} \;\wedge\; \mathrm{WF}_{\langle i.ack,\, o.snd,\, q\rangle}(\mathcal{Q}_M^{ni}) \\
QM^{ni} & \triangleq & \exists\, q : IQM^{ni}
\end{array}
$$

Fig. 9. A noninterleaving representation of the queue component.

specification in which the component records environment actions by changing its own internal variables. In this case, $\mathcal{N}$ will not equal $\mathcal{N}_m \vee (\langle m, x\rangle' = \langle m, x\rangle)$, but may just imply $(e' = e) \vee (m' = m)$. The resulting formula will not be a pure interleaving specification because environment actions can change the component's variables, but no action can change both the component's and the environment's output variables. We have not explored this style of specification.

We have been assuming that the visible variables of the component's specification can be partitioned into tuples m of output variables and e of input variables. To see how to handle a more general case, let μ_M be the action $m' \neq m$, let v equal $\langle e, m\rangle$, and observe that $[\mathcal{N}_M]_{\langle m, x\rangle}$ equals $[\mathcal{N}_M \vee (\neg\mu_M \wedge (x' = x))]_{\langle v, x\rangle}$. A μ_M step is one that is attributed to the component, since it changes the component's output variables. When the tuple v of variables is not partitioned into input and output variables, we define an action μ_M that specifies what steps are attributed to the component, and we write the component's next-state action in the form $\mathcal{N}_M \vee (\neg\mu_M \wedge (x' = x))$. All our results for separate input and output variables can be generalized by writing the next-state action in this form. However, for simplicity, we consider only the special case.

4.2 Conjoining Components to Form a Complete System

In Section 3.1, we describe how to specify a complete system. In Section 4.1, we describe how to specify an individual component of a system. A complete system is the composition of its components. Composing two systems means constructing a universe in which they are both running. If formulas M_1 and M_2 represent the two systems, then $M_1 \wedge M_2$ represents their composition, since a behavior represents a possible history of a universe containing both systems iff it satisfies both M_1 and M_2. Thus, in principle, composition is conjunction. We now show that composition is conjunction in practice as well.

For composition to be conjunction, the conjunction of the specifications of all components should be equivalent to the specification of the complete system. For example, the conjunction of the specifications QM of the queue and QE of its environment should be equivalent to the specification CQ of the complete system shown in Figure 5. Recall that

$$\begin{aligned} QE &= Init_E \;\wedge\; \Box[\mathcal{Q}_E]_{\langle i.snd,\, o.ack\rangle} \\ QM &= \exists\, q : Init_M \;\wedge\; \Box[\mathcal{Q}_M]_{\langle i.ack,\, o.snd,\, q\rangle} \;\wedge\; ICL \\ CQ &= \exists\, q : \wedge\; Init_E \;\wedge\; Init_M \\ &\qquad\quad \wedge\; \Box\begin{bmatrix} \vee\; \mathcal{Q}_E \;\wedge\; (q' = q) \\ \vee\; \mathcal{Q}_M \end{bmatrix}_{\langle i,\, o,\, q\rangle} \\ &\qquad\quad \wedge\; ICL \end{aligned}$$

We deduce the equivalence of $QE \wedge QM$ and CQ from the following result, by substituting QE for M_1 and QM for M_2. (In this case, x_1 is the empty tuple $\langle\,\rangle$, so $\widehat{x}_2$ equals $\langle\,\rangle$ and $\widehat{x}'_2 = \widehat{x}_2$ equals true.)

Proposition 6. *Let $m_1, \ldots, m_n$, $x_1, \ldots, x_n$ be tuples of variables, and let*

$$\begin{gathered} m \;\triangleq\; \langle m_1, \ldots, m_n\rangle \qquad\qquad x \;\triangleq\; \langle x_1, \ldots, x_n\rangle \\ \widehat{x}_i \;\triangleq\; \langle x_1, \ldots, x_{i-1},\, x_{i+1}, \,\ldots, x_n\rangle \\ M_i \;\triangleq\; \exists\, x_i : Init_i \;\wedge\; \Box[\mathcal{N}_i]_{\langle m_i,\, x_i\rangle} \;\wedge\; L_i \end{gathered}$$

If, for all i, $j = 1, \ldots, n$ with $i \neq j$:

(1) no variable of x_j occurs free in x_i or M_i,

(2) m includes all free variables of M_i, and

(3) $\models \mathcal{N}_i \Rightarrow (m'_j = m_j)$

then

$$\models \bigwedge_{i=1}^{n} M_i \;=\; \exists\, x : \bigwedge_{i=1}^{n} Init_i \;\wedge\; \Box[\bigvee_{i=1}^{n} \mathcal{N}_i \wedge (\widehat{x}'_i = \widehat{x}_i)]_{\langle m,\, x\rangle} \;\wedge\; \bigwedge_{i=1}^{n} L_i$$

In this proposition, the third hypothesis asserts that component i leaves the variables of other components unchanged, so M_i is an interleaving representation of component i. Hence, M_i implies $Disjoint(m_i, m_j)$, for each $j \neq i$, and $\bigwedge_{i=1}^{n} M_i$ implies $Disjoint(m_1, \ldots, m_n)$, as expected for an interleaving representation of the complete system.

In the GCD example, we apply this proposition to the formula M_a of (5) and the analogous formula M_b. We immediately get that $M_a \wedge M_b$ is equivalent to a formula that is the same as M_{gcd}, defined by (2), except with $\mathrm{WF}_{\langle a,\, b\rangle}(\mathcal{N}_a) \wedge \mathrm{WF}_{\langle a,\, b\rangle}(\mathcal{N}_b)$ instead of $\mathrm{WF}_{\langle a,\, b\rangle}(\mathcal{N})$. It can be shown that these two fairness conditions are equivalent; hence $M_a \wedge M_b$ is equivalent to M_{gcd}.

For another example of decomposition, we consider the system of Figure 7, which consists of two queues in series together with an environment. This system can be decomposed into three components with the following specifications.

$$\begin{aligned} &\text{1st queue:} && \exists\, q_1 : Init_M^{[1]} \;\wedge\; \Box[\mathcal{Q}_M^{[1]} \wedge (o' = o)]_{\langle i.ack,\, z.snd,\, q_1\rangle} \;\wedge\; ICL^{[1]} \\ &\text{2nd queue:} && \exists\, q_2 : Init_M^{[2]} \;\wedge\; \Box[\mathcal{Q}_M^{[2]} \wedge (i' = i)]_{\langle z.ack,\, o.snd,\, q_2\rangle} \;\wedge\; ICL^{[2]} \\ &\text{environment:} && Init_E \;\wedge\; \Box[\mathcal{Q}_E \wedge (z' = z)]_{\langle i.snd,\, o.ack\rangle} \end{aligned}$$

To obtain an interleaving representation, we have conjoined $o' = o$ to $\mathcal{Q}_M^{[1]}$ in the first queue's next-state action, because $\mathcal{Q}_M^{[1]}$ does not mention o. Similarly, we have conjoined $i' = i$ to the second queue's next-state action, and $z' = z$ to the environment's. It follows from Proposition 6 that the conjunction of these three specifications equals the specification CDQ of the complete system, defined in Figure 8.

The third hypothesis of Proposition 6 is satisfied only by interleaving representations. For arbitrary representations, a straightforward calculation shows

$$\models \bigwedge_{i=1}^{n} M_i \;=\; \exists\, x : \begin{array}{l} \wedge\ \bigwedge_{i=1}^{n} Init_i \\ \wedge\ \Box[\bigwedge_{i=1}^{n}(\mathcal{N}_i \vee \langle m_i,\, x_i\rangle' = \langle m_i,\, x_i\rangle)]_{\langle m,\, x\rangle} \\ \wedge\ \bigwedge_{i=1}^{n} L_i \end{array} \tag{9}$$

assuming only the first hypothesis of the proposition. The right-hand side has the expected form for a noninterleaving specification, since it allows $\mathcal{N}_i \wedge \mathcal{N}_j$ steps for $i \neq j$. Hence, composition is conjunction for noninterleaving representations too.

4.3 The Decomposition Theorem

4.3.1 *The Basic Theorem.* Consider a complete system decomposed into components Π_i. We would like to prove that this system is implemented by a lower-level one, consisting of components Π_i^l, by proving that each Π_i^l implements Π_i. Let M_i be the specification of Π_i and M_i^l be the specification of Π_i^l. We must prove that $\bigwedge_{i=1}^n M_i^l$ implies $\bigwedge_{i=1}^n M_i$. This implication is trivially true if M_i^l implies M_i, for all i. However, as we saw in the GCD example, M_i^l need not imply M_i.

Even when $M_i^l \Rightarrow M_i$ does not hold, we need not reason about all the lower-level components together. Instead, we prove $E_i \wedge M_i^l \Rightarrow M_i$, where E_i includes just the properties of the other components assumed by component i, and is usually much simpler than $\bigwedge_{k \neq i} M_k^l$. Proving $E_i \wedge M_i^l \Rightarrow M_i$ involves reasoning only about component i, not about the entire lower-level system.

In propositional logic, to deduce that $\bigwedge_{i=1}^n M_i^l$ implies $\bigwedge_{i=1}^n M_i$ from $\bigwedge_{i=1}^n (E_i \wedge M_i^l \Rightarrow M_i)$, we may prove that $\bigwedge_{k=1}^n M_k^l$ implies E_i for each i. However, proving this still requires reasoning about $\bigwedge_{k=1}^n M_k^l$, the specification of the entire lower-level system. The following theorem shows that we need only prove that E_i is implied by $\bigwedge_{k=1}^n M_k$, the specification of the higher-level system—a formula usually much simpler than $\bigwedge_{k=1}^n M_k^l$.

Proving $E_i \wedge M_i^l \Rightarrow M_i$ and $(\bigwedge_{k=1}^n M_k) \Rightarrow E_i$ for each i and deducing $(\bigwedge_{i=1}^n M_i^l) \Rightarrow (\bigwedge_{i=1}^n M_i)$ is circular reasoning, and is not sound in general. Such reasoning would allow us to deduce $(\bigwedge_{i=1}^n M_i^l) \Rightarrow (\bigwedge_{i=1}^n M_i)$ for any M_i^l and M_i—simply let E_i equal M_i. To break the circularity, we need to add some $\mathcal{C}$'s and one hypothesis: if E_i is ever violated then, for at least one additional step, M_i^l implies M_i. This hypothesis is expressed formally as $\models \mathcal{C}(E_i)_{+v} \wedge \mathcal{C}(M_i^l) \;\Rightarrow\; \mathcal{C}(M_i)$, for some v; the hypothesis is weakest when v is taken to be the tuple of all relevant variables. Our proof rule is:

THEOREM 1 (DECOMPOSITION THEOREM). *If, for* $i = 1, \ldots, n$,

(1) $\models \bigwedge_{j=1}^{n} \mathcal{C}(M_j) \;\Rightarrow\; E_i$

(2) (a) $\models \mathcal{C}(E_i)_{+v} \wedge \mathcal{C}(M_i^l) \;\Rightarrow\; \mathcal{C}(M_i)$

(b) $\models E_i \wedge M_i^l \;\Rightarrow\; M_i$

then $\models \bigwedge_{i=1}^{n} M_i^l \;\Rightarrow\; \bigwedge_{i=1}^{n} M_i$.

This theorem is a corollary of the Composition Theorem of Section 5.2 below.

In the GCD example, we want to use the theorem to prove $M_a^l \wedge M_b^l \Rightarrow M_a \wedge M_b$. (The component specifications are described in Section 4.1.) The abstract environment specification E_a asserts that b can change only when $a < b$, and that a is not changed by steps that change b. Thus,

$$E_a \;\triangleq\; \Box[(a < b) \wedge (a' = a)]_b$$

The definition of E_b is analogous. We let v be $\langle a, b \rangle$.

In general, the environment and component specifications can have internal variables. The theorem also allows them to contain fairness conditions. However, the first hypothesis asserts that the E_i are implied by safety properties. In practice, this means that the theorem can be applied only when the E_i are safety properties. The examples of Section 2.2 lead us to expect such a restriction. Moreover, if the E_i have internal variables, we expect them to be simple history-determined variables [Abadi and Lamport 1994], so Proposition 2 of [Abadi and Lamport 1991] can be used to prove that the E_i are safety properties.

4.3.2 *Verifying the Hypotheses.* We now discuss how one verifies the hypotheses of the Decomposition Theorem, illustrating the method with the GCD example.

To prove the first hypothesis, one first eliminates the closure operators and existential quantifiers by using Propositions 1 and 2 and Proposition 2 of [Abadi and Lamport 1991]. This reduces the hypothesis to a condition of the form

$$\models \bigwedge_{i=1}^{n} (Init_i \wedge \Box[\mathcal{N}_i]_{v_i}) \Rightarrow E_i \tag{10}$$

For interleaving representations, we can then use Proposition 6 to write $\bigwedge_{i=1}^{n}(Init_i \wedge \Box[\mathcal{N}_i]_{v_i})$ in canonical form. For noninterleaving representations, we apply (9). In either case, the proof of (10) is an implementation proof of the kind discussed in Section 3.2.

For the GCD example, the first hypothesis asserts that $\mathcal{C}(M_a) \wedge \mathcal{C}(M_b)$ implies E_a and E_b. This differs from the third hypothesis of (1) in Section 2.1 because of the $\mathcal{C}$'s. To verify the hypothesis, we can apply Proposition 1 to show that $\mathcal{C}(M_a)$ and $\mathcal{C}(M_b)$ are obtained by simply deleting the fairness conditions from M_a and M_b. Since $\mathcal{N}_b$ implies $(a < b) \wedge (a' = a)$, it is easy to see that $\mathcal{C}(M_b)$ implies E_a. It is equally easy to see that $\mathcal{C}(M_a)$ implies E_b. (In more complicated examples, E_i will not follow from $\mathcal{C}(M_j)$ for any single j.)

To prove part (a) of the second hypothesis, we first eliminate the $+$. For noninterleaving representations, this must be done with Proposition 3, as described in Section 3.5.1. For interleaving representations, we can apply Propositions 4 and 5, as described in Section 3.5.4. In either case, we can prove the resulting formula by first using Proposition 2 to eliminate quantifiers, using Proposition 1 to compute closures, and then performing a standard implementation proof with a refinement mapping.

Part (b) of the hypothesis also calls for a standard implementation proof, for which we use the same refinement mapping as in the proof of (a). Since E_i implies $\mathcal{C}(E_i)_{+v}$ and M_i^l implies $\mathcal{C}(M_i^l)$, we can infer from part (a) that $E_i \wedge M_i^l$ implies $\mathcal{C}(M_i)$. Thus proving part (b) requires verifying only the liveness part of M_i.

For the GCD example, we verify the two parts of the second hypothesis by proving $\mathcal{C}(E_a)_{+\langle a,\, b\rangle} \wedge \mathcal{C}(M_a^l) \Rightarrow \mathcal{C}(M_a)$ and $E_a \wedge M_a^l \Rightarrow M_a$; the proofs of the corresponding conditions for M_b are similar. We first observe that the initial condition of E_a is true, and that, since M_a^l is an interleaving representation, its next-state action $\mathcal{N}_a^l$ implies that no step changes both a and b, so $\mathcal{C}(M_a^l)$ implies $\mathit{Disjoint}(a,\, b)$. Hence, applying Propositions 4 and 5, we reduce our task to proving $\mathcal{C}(E_a) \wedge \mathcal{C}(M_a^l) \Rightarrow \mathcal{C}(M_a)$ and $E_a \wedge M_a^l \Rightarrow M_a$. Applying Proposition 2 to remove the quantifier from $\mathcal{C}(M_a^l)$ and Proposition 1 to remove the $\mathcal{C}$'s, we reduce proving $\mathcal{C}(E_a) \wedge \mathcal{C}(M_a^l) \Rightarrow \mathcal{C}(M_a)$ to proving

$$E_a \wedge \mathit{Init}_a^l \wedge \Box[\mathcal{N}_a^l]_{\langle a,\, ai,\, pca\rangle} \Rightarrow \mathit{Init}_a \wedge \Box[\mathcal{N}_a]_a \tag{11}$$

Using simple logic and (11), we reduce proving $E_a \wedge M_a^l \;\Rightarrow\; M_a$ to proving

$$E_a \;\wedge\; \mathit{Init}_a^l \wedge \Box[\mathcal{N}_a^l]_{\langle a,\, ai,\, pca\rangle} \wedge \mathrm{WF}_{\langle a,\, ai,\, pca\rangle}(\mathcal{N}_a^l) \;\Rightarrow\; \mathrm{WF}_a(\mathcal{N}_a) \tag{12}$$

We can use Proposition 6 to rewrite the left-hand sides of (11) and (12) in canonical form. The resulting conditions are in the usual form for a TLA implementation proof.

In summary, by applying our propositions in a standard sequence, we can use the Decomposition Theorem to reduce decompositional reasoning to ordinary TLA reasoning. This reduction may seem complicated for so trivial an example as the GCD program. However, it will be insignificant compared to the complexity of the complete proof in any realistic example, such as the one by Kurshan and Lamport [1993], discussed below.

4.3.3 *The General Theorem.* We sometimes need to prove the correctness of systems defined inductively. At induction stage $N+1$, the low- and high-level specifications are defined as the conjunctions of k copies of low- and high-level specifications of stage N, respectively. For example, a 2^{N+1}-bit multiplier is sometimes implemented by combining four 2^N-bit multipliers. We want to prove by induction on N that the stage N low-level specification implements the stage N high-level specification. For such a proof, we need a more general decomposition theorem whose conclusion at stage N can be used in proving the hypotheses at stage $N+1$. The appropriate theorem is:

THEOREM 2 (GENERAL DECOMPOSITION THEOREM). *If, for $i = 1, \ldots, n$,*

(1) $\models \mathcal{C}(E) \wedge \bigwedge_{j=1}^{n} \mathcal{C}(M_j) \;\Rightarrow\; E_i$

(2) (a) $\models \mathcal{C}(E_i)_{+v} \wedge \mathcal{C}(M_i^l) \;\Rightarrow\; \mathcal{C}(M_i)$
(b) $\models E_i \wedge M_i^l \;\Rightarrow\; M_i$

(3) v is a tuple of variables including all the free variables of M_i

then

(a) $\models \mathcal{C}(E)_{+v} \wedge \bigwedge_{j=1}^{n} \mathcal{C}(M_j^l) \;\Rightarrow\; \bigwedge_{j=1}^{n} \mathcal{C}(M_j)$ *and*

(b) $\models E \wedge \bigwedge_{j=1}^{n} M_j^l \;\Rightarrow\; \bigwedge_{j=1}^{n} M_j$.

Conclusion (b) of this theorem has the same form as hypothesis 2(b), with M_i^l and M_i replaced with conjunctions. To make the corresponding hypothesis 2(a) follow from conclusion (a), it suffices to prove $\bigwedge_{j=1}^{n} \mathcal{C}(M_j) \Rightarrow \mathcal{C}(\bigwedge_{j=1}^{n} M_j)$, since $\mathcal{C}(\bigwedge_{j=1}^{n} M_j^l) \Rightarrow \bigwedge_{j=1}^{n} \mathcal{C}(M_j^l)$ is always true.

The General Decomposition Theorem has been applied to the verification of an inductively defined multiplier circuit [Kurshan and Lamport 1993].

It can be shown that both versions of our decomposition theorem provide complete rules for verifying that one composition implies another. However, this result is of no significance. Decomposition can simplify a proof only if the proof can be decomposed, in the sense that each M_i^l implements the corresponding M_i under a simple environment assumption E_i. Our theorems are designed to handle those proofs that can be decomposed.

5. COMPOSING ASSUMPTION/GUARANTEE SPECIFICATIONS

5.1 The Form of an Assumption/Guarantee Specification

An assumption/guarantee specification asserts that a system guarantees M under the assumption that its environment satisfies E. As we saw in Section 2.2, this specification is expressed by the formula $E \stackrel{+}{-\!\!\triangleright} M$, which means that, for any n, if the environment satisfies E through "time" n, then the system must satisfy M through "time" $n{+}1$.

Perhaps the most obvious form for an assumption/guarantee specification is $E \Rightarrow M$. The formula $E \Rightarrow M$ is weaker than $E \stackrel{+}{-\!\!\triangleright} M$, since it allows behaviors in which M is violated before E. However, an implementation could exploit this extra freedom only by predicting in advance that the environment will violate E. A system does not control its environment, so it cannot predict what the environment will do. The specifications $E \Rightarrow M$ and $E \stackrel{+}{-\!\!\triangleright} M$ therefore allow the same implementations. We take $E \stackrel{+}{-\!\!\triangleright} M$ to be the form of assumption/guarantee specifications because this form leads to the simpler rules for composition.

As discussed in Section 2.2, composition works best when environment assumptions are safety properties. It can be shown that $E \stackrel{+}{-\!\!\triangleright} M$ is equivalent to $\mathcal{C}(E) \stackrel{+}{-\!\!\triangleright} (\mathcal{C}(M) \wedge (E \Rightarrow M))$, so we can in principle convert any assumption/guarantee specification to one whose assumption is a safety property. (A similar observation appears in our earlier work [Abadi and Lamport 1993, Theorem 1].) However, this equivalence is of intellectual interest only. In practice, we write the environment assumption as a safety property and the system's fairness guarantee as the conjunction of properties $E_L \Rightarrow \mathrm{WF}_v(\mathcal{A})$ and $E_L \Rightarrow \mathrm{SF}_v(\mathcal{A})$, where E_L is an environment fairness assumption. We can apply Proposition 1 to show that the resulting specification is machine closed because, if (P, L) is machine closed and L implies R, then (P, R) is also machine closed [Abadi and Lamport 1994, Proposition 3].

5.2 The Composition Theorem

Suppose we are given n devices, each with an assumption/guarantee specification $E_j \stackrel{+}{\rhd} M_j$. To verify that the composition of these devices implements a higher-level assumption/guarantee specification $E \stackrel{+}{\rhd} M$, we must prove $\bigwedge_{j=1}^{n}(E_j \stackrel{+}{\rhd} M_j) \Rightarrow (E \stackrel{+}{\rhd} M)$. We use the following theorem:

THEOREM 3 (COMPOSITION THEOREM). *If, for* $i = 1, \ldots, n$,

(1) $\models \mathcal{C}(E) \wedge \bigwedge_{j=1}^{n} \mathcal{C}(M_j) \;\Rightarrow\; E_i$

(2) (a) $\models \mathcal{C}(E)_{+v} \wedge \bigwedge_{j=1}^{n} \mathcal{C}(M_j) \;\Rightarrow\; \mathcal{C}(M)$

(b) $\models E \wedge \bigwedge_{j=1}^{n} M_j \;\Rightarrow\; M$

then $\models \bigwedge_{j=1}^{n}(E_j \stackrel{+}{\rhd} M_j) \;\Rightarrow\; (E \stackrel{+}{\rhd} M)$.

This theorem also allows us to prove conditional implementation results of the form $G \wedge \bigwedge_{j=1}^{n}(E_j \stackrel{+}{\rhd} M_j) \Rightarrow (E \stackrel{+}{\rhd} M)$; we just let M_1 equal G and E_1 equal true, since $\mathsf{true} \stackrel{+}{\rhd} G$ equals G. For interleaving specifications, we can in general prove only conditional implementation, where G includes disjointness conditions asserting that the outputs of different components do not change simultaneously.

The hypotheses of the Composition Theorem are similar to those of the Decomposition Theorem, and they are proved in much the same way. The major difference is that, for interleaving specifications, the orthogonality condition $\mathcal{C}(E) \perp \mathcal{C}(M)$ does not follow from the form of the component specifications, but requires explicit disjointness assumptions.

Observe that the hypotheses have the form $\models P \wedge \bigwedge_{j=1}^{n} Q_j \;\Rightarrow\; R$. Each formula $P \wedge \bigwedge_{j=1}^{n} Q_j$ has the form of the specification of a complete system, with component specifications P, Q_1, ..., Q_n. Thus, each hypothesis asserts that a complete system satisfies a property R. In other words, the theorem reduces reasoning about assumption/guarantee specifications to the kind of reasoning used for complete-system specifications.

Among the corollaries of the Composition Theorem are ones that allow us to prove that a lower-level specification implies a higher-level one. The simplest such result has, as its conclusion, $\models (E \stackrel{+}{\rhd} M^l) \Rightarrow (E \stackrel{+}{\rhd} M)$. This condition expresses the correctness of the refinement of a component with a fixed environment assumption.

COROLLARY 1. *If E is a safety property and*

(a) $\models E_{+v} \wedge \mathcal{C}(M^l) \Rightarrow \mathcal{C}(M)$

(b) $\models E \wedge M^l \Rightarrow M$

then $\models (E \stackrel{+}{\rhd} M^l) \Rightarrow (E \stackrel{+}{\rhd} M)$.

5.3 The Queue Example

The assumption/guarantee specification of the queue of Figure 2 is $QE \stackrel{+}{\rhd} QM$, where QM and QE are defined in (7) and (8) of Section 4.1. We now compose two queues, as shown in Figure 7. The specifications of these queues are obtained from $QE \stackrel{+}{\rhd} QM$ by substitution; they are $QE^{[1]} \stackrel{+}{\rhd} QM^{[1]}$ and $QE^{[2]} \stackrel{+}{\rhd} QM^{[2]}$. We want to show that their composition implements the $(2N+1)$-element queue specified by $QE^{[\mathrm{dbl}]} \stackrel{+}{\rhd} QM^{[\mathrm{dbl}]}$. The obvious thing to try to prove is

$$(QE^{[1]} \stackrel{+}{\rhd} QM^{[1]}) \wedge (QE^{[2]} \stackrel{+}{\rhd} QM^{[2]}) \;\Rightarrow\; (QE^{[\mathrm{dbl}]} \stackrel{+}{\rhd} QM^{[\mathrm{dbl}]}) \qquad (13)$$

We could prove this had we used a noninterleaving representation of the queue. However, (13) is not valid for an interleaving representation, for the following reason. The specification of the first queue does not mention o, and that of the second queue does not mention i. The conjunction of the two specifications allows an enqueue action of the first queue and a dequeue action of the second queue to happen simultaneously, a step that changes $i.ack$ and $o.snd$ simultaneously. But, in an interleaving representation, the $(2N+1)$-element queue's guarantee does not allow such a step, so (13) must be invalid. Another problem with (13) is that the conjunction of the component queues' specifications allows a step that changes $z.snd$ and $o.ack$ simultaneously. Such a step satisfies the $(2N+1)$-element queue's environment assumption $QE^{[\mathrm{dbl}]}$, which does not mention z, so (13) asserts that the next step must satisfy its guarantee $QM^{[\mathrm{dbl}]}$. However, a step that changes both $z.snd$ and $o.ack$ violates the second component queue's environment assumption $QE^{[2]}$, permitting the component queue to make arbitrary changes to $o.snd$ in the next step. A similar problem is caused by simultaneous changes to $i.snd$ and $z.ack$.

We already faced the problem of disallowing simultaneous changes to different components' outputs in Section 4.2, where we decomposed an interleaving specification of a $(2N+1)$-element queue. There, the solution was to strengthen the next-state actions of the component queues and of the environment. This solution cannot be used if we want to compose preexisting specifications without modifying them. In this case, we prove that the composition implements the larger queue under the assumption that the outputs of two different components do not change simultaneously. Thus, we prove

$$G \wedge (QE^{[1]} \stackrel{+}{\rhd} QM^{[1]}) \wedge (QE^{[2]} \stackrel{+}{\rhd} QM^{[2]}) \;\Rightarrow\; (QE^{[\mathrm{dbl}]} \stackrel{+}{\rhd} QM^{[\mathrm{dbl}]}) \qquad (14)$$

where G is the formula

$$G \triangleq \mathit{Disjoint}(\langle i.snd,\, o.ack\rangle,\, \langle z.snd,\, i.ack\rangle,\, \langle o.snd,\, z.ack\rangle)$$

The proof is outlined in Figure 10.

6. CONCLUSION

We have developed a method for describing components of concurrent systems as TLA formulas. We have shown how to describe a complete system as the conjunction of component specifications and how to describe an open system as a formula $E \stackrel{+}{\rhd} M$, where E and M are specifications of an environment component and a system component, respectively. Although the idea of reducing programming concepts to logic is old, our approach is new. Our style of writing specifications is direct and, we believe, practical.

1. $\mathcal{C}(QE^{[\mathrm{dbl}]}) \wedge \mathcal{C}(G) \wedge \mathcal{C}(QM^{[1]}) \wedge \mathcal{C}(QM^{[2]}) \;\Rightarrow\; QE^{[1]} \wedge QE^{[2]}$
   PROOF: We use Propositions 2 and 1 to remove the quantifiers and closure operators from the left-hand side of the implication. The resulting formula then asserts that a complete system, consisting of the safety parts of the two queues (with their internal state visible) together with the environment, implements $QE^{[1]}$ and $QE^{[2]}$. The proof of this formula is straightforward.
2. $\mathcal{C}(QE^{[\mathrm{dbl}]})_{+\langle i,\, o,\, z\rangle} \wedge \mathcal{C}(QM^{[1]}) \wedge \mathcal{C}(G) \wedge \mathcal{C}(QM^{[2]}) \;\Rightarrow\; \mathcal{C}(QM^{[\mathrm{dbl}]})$
   2.1. $\mathcal{C}(G) \wedge \mathcal{C}(QM^{[1]}) \wedge \mathcal{C}(QM^{[2]}) \;\Rightarrow\; \mathcal{C}(QE^{[\mathrm{dbl}]}) \perp \mathcal{C}(QM^{[\mathrm{dbl}]})$
      2.1.1. $\mathcal{C}(IQM^{[1]}) \wedge \mathcal{C}(IQM^{[2]}) \;\Rightarrow\; \exists\, q_1, q_2 : \mathit{Init}_M^{[1]} \wedge \mathit{Init}_M^{[2]}$
         PROOF: Follows easily from Proposition 1 and the definitions.
      2.1.2. $\mathcal{C}(QM^{[1]}) \wedge \mathcal{C}(QM^{[2]}) \;\Rightarrow\; \exists\, q_1, q_2 : \mathit{Init}_M^{[1]} \wedge \mathit{Init}_M^{[2]}$
         PROOF: 2.1.1 and Proposition 2 (since any predicate is a safety property).
      2.1.3. Q.E.D.
         PROOF: 2.1.2, the definition of G, and Proposition 5 (since disjointness is a safety property).
   2.2. $\mathcal{C}(QE^{[\mathrm{dbl}]}) \wedge \mathcal{C}(G) \wedge \mathcal{C}(QM^{[1]}) \wedge \mathcal{C}(QM^{[2]}) \;\Rightarrow\; \mathcal{C}(QM^{[\mathrm{dbl}]})$
      PROOF: We use Propositions 2 and 1 to remove the quantifiers and closures from the formula. The resulting formula is proved when proving the safety part of step 3.
   2.3. Q.E.D.
      PROOF: 2.1, 2.2, and Proposition 4.
3. $QE^{[\mathrm{dbl}]} \wedge G \wedge QM^{[1]} \wedge QM^{[2]} \;\Rightarrow\; QM^{[\mathrm{dbl}]}$
   PROOF: A direct calculation shows that the left-hand side of the implication implies CDQ, the complete-system specification of the double queue. We already observed in Section 3.2 that CDQ implements $CQ^{[\mathrm{dbl}]}$, which equals $QE^{[\mathrm{dbl}]} \wedge QM^{[\mathrm{dbl}]}$.
4. Q.E.D.
   PROOF: 1–3 and the Composition Theorem, substituting

$$\begin{array}{llll} M_1 \leftarrow G & M_2 \leftarrow QM^{[1]} & M_3 \leftarrow QM^{[2]} & M \leftarrow QM^{[\mathrm{dbl}]} \\ E_1 \leftarrow \mathsf{true} & E_2 \leftarrow QE^{[1]} & E_3 \leftarrow QE^{[2]} & E \leftarrow QE^{[\mathrm{dbl}]} \end{array}$$

Fig. 10. Proof sketch of (14).

We have also provided rules for proving properties of large systems by reasoning about their components. The Composition and Decomposition Theorems are rather simple, yet they allow fairness properties and hiding. They were preceded by results in a long list of publications, described next.

Like ours, most previous composition theorems were strong, in the sense that they could handle circularities for safety properties. Our approach differs from earlier ones in its general treatment of fairness and hiding. The first strong composition theorem we know is that of Misra and Chandy [1981], who considered safety properties of processes communicating by means of CSP primitives. They wrote assumption/guarantee specifications as Hoare triples containing assertions about history variables. Pandya and Joseph [1991] extended this approach to handle some liveness properties. Pnueli [1984] was the first to use temporal logic to write assumption/guarantee specifications. He had a strong composition theorem for safety properties with no hiding. To handle liveness, he wrote assumption/guarantee specifications with implication instead of $\stackrel{+}{\triangleright}$, so he did not obtain a strong composition theorem. Stark [1985] also wrote assumption/guarantee specifications as implications of temporal formulas and required that circularity be avoided. Our earlier work [Abadi and Lamport 1993] was semantic, in a more complicated model with agents. It lacked practical proof rules for handling fairness and hiding. Collette [1993] adapted this work to Unity. Abadi and Plotkin [1993] used a propositional logic with agents, and considered only safety properties.

Most previous papers were concerned only with composition of assumption/guarantee specifications, and lacked an analog of our Decomposition Theorem. An exception is the work of Berthet and Cerny [1988], who used decomposition in proving safety properties for finite-state automata.

So far, we have applied our Composition Theorem only to toy examples. Formal reasoning about systems is still rare, and it generally occurs on a case-by-case basis. When the specification of a component is used only to verify a specific system, there is no need for a general assumption/guarantee specification. For most practical applications, decomposition suffices. When decomposition does not suffice, the Composition Theorem makes reasoning about open systems almost as easy as reasoning about complete ones.

We have used our Decomposition Theorem with no difficulty on a few toy examples. However, we believe that its biggest payoff will be for systems that are too complex to verify easily by hand. The theorem makes it possible for decision procedures to do most of the work in verifying a system, even when these procedures cannot be applied to the whole system because its state space is very large or unbounded. This approach is currently being pursued in one substantial example: the mechanical verification of a multiplier circuit using a combination of TLA reasoning and mechanical verification with COSPAN [Kurshan and Lamport 1993]. Because it eliminates reasoning about the complete low-level system, the Decomposition Theorem is the key to this division of labor.

Acknowledgments

Karlis Cerans, Stephan Merz, Yuan Yu, and anonymous referees provided helpful comments on an earlier version.

REFERENCES

ABADI, M. AND LAMPORT, L. 1994. An old-fashioned recipe for real time. *ACM Trans. Program. Lang. Syst. 16,* 5 (Sept.), 1543–1571.

ABADI, M. AND LAMPORT, L. 1993. Composing specifications. *ACM Trans. Program. Lang. Syst. 15,* 1 (Jan.), 73–132.

ABADI, M. AND LAMPORT, L. 1991. The existence of refinement mappings. *Theor. Comput. Sci. 82,* 2 (May), 253–284.

ABADI, M. AND PLOTKIN, G. 1993. A logical view of composition. *Theor. Comput. Sci. 114,* 1 (June), 3–30.

ABRAMSKY, S. AND JAGADEESAN, R. 1994. Games and full completeness for multiplicative linear logic. *J. Symb. Logic 59,* 2 (June), 543–574.

ALPERN, B. AND SCHNEIDER, F. B. 1985. Defining liveness. *Inf. Process. Lett. 21,* 4 (Oct.), 181–185.

BERTHET, C. AND CERNY, E. 1988. An algebraic model for asynchronous circuits verification. *IEEE Trans. Comput. 37,* 7 (July), 835–847.

COLLETTE, P. 1993. Application of the composition principle to Unity-like specifications. In *TAPSOFT'93: Theory and Practice of Software Development*, M.-C. Gaudel and J.-P. Jouannaud, Eds. Lecture Notes in Computer Science, vol. 668. Springer-Verlag, Berlin, 230–242.

JONES, C. B. 1983. Specification and design of (parallel) programs. In *Information Processing 83: Proceedings of the IFIP 9th World Congress*, R. E. A. Mason, Ed. North-Holland, Amsterdam, 321–332.

KURSHAN, R. P. AND LAMPORT, L. 1993. Verification of a multiplier: 64 bits and beyond. In *Computer-Aided Verification*, C. Courcoubetis, Ed. Lecture Notes in Computer Science, vol. 697. Springer-Verlag, Berlin, 166–179. Proceedings of the 5th International Conference, CAV'93.

LAMPORT, L. 1994. The temporal logic of actions. *ACM Trans. Program. Lang. Syst. 16,* 3 (May), 872–923.

LAMPORT, L. 1989. A simple approach to specifying concurrent systems. *Commun. ACM 32,* 1 (Jan.), 32–45.

LAMPORT, L. 1983. What good is temporal logic? In *Information Processing 83: Proceedings of the IFIP 9th World Congress*, R. E. A. Mason, Ed. North-Holland, Amsterdam, 657–668.

MEAD, C. AND CONWAY, L. 1980. *Introduction to VLSI Systems*. Addison-Wesley, Reading, Mass., chap. 7.

MISRA, J. AND CHANDY, K. M. 1981. Proofs of networks of processes. *IEEE Trans. Softw. Eng. SE-7,* 4 (July), 417–426.

PANDYA, P. K. AND JOSEPH, M. 1991. P-A logic—a compositional proof system for distributed programs. *Distrib. Comput. 5,* 1, 37–54.

PNUELI, A. 1984. In transition from global to modular temporal reasoning about programs. In *Logics and Models of Concurrent Systems*, K. R. Apt, Ed. NATO ASI Series, Springer-Verlag, Berlin, 123–144.

PNUELI, A. 1981. The temporal semantics of concurrent programs. *Theor. Comput. Sci. 13*, 45–60.

STARK, E. W. 1985. A proof technique for rely/guarantee properties. In *Foundations of Software Technology and Theoretical Computer Science*, S. N. Maheshwari, Ed. Lecture Notes in Computer Science, vol. 206. Springer-Verlag, Berlin, 369–391.

ZAVE, P. AND JACKSON, M. 1993. Conjunction as composition. *ACM Trans. Softw. Eng. Method. 2,* 4 (Oct.), 379–411.

Received December 1993; revised July 1994 and January 1995; accepted January 1995

APPENDIX

An appendix to this article is available in electronic form (PostScript™). Any of the following methods may be used to obtain it; or see the inside back cover of a current issue for up-to-date instructions.

—By anonymous ftp from **acm.org**, file **[pubs.journals.toplas.append]p1532.ps**

—Send electronic mail to **mailserve@acm.org** containing the line
send [anonymous.pubs.journals.toplas.append]p1532.ps

—By *Gopher* from **acm.org**

—By anonymous ftp from **ftp.cs.princeton.edu**, file **pub/toplas/append/p1532.ps**

—Hardcopy from *Article Express*, for a fee: phone 800-238-3458, fax 201-216-8526, or write P.O. Box 1801, Hoboken NJ 07030; and request ACM-TOPLAS-APPENDIX-1532.

THIS DOCUMENT IS THE APPENDIX TO THE FOLLOWING PAPER:

Conjoining Specifications

MARTÍN ABADI and LESLIE LAMPORT
Digital Equipment Corporation

ACM Transactions on Programming Languages and Systems 17, 3 (May, 1995)

We now prove our propositions and theorems. Section A introduces some definitions and notation required for the proofs, and explains our structured proof notation. The proofs are in Section B.

APPENDIX A. DEFINITIONS

A1 Additional Semantic Notions

As before, $\circ$ denotes concatenation of sequences, and angle brackets $\langle\,\rangle$ are used to form sequences. We write $\sigma|_n$ for the finite behavior consisting of the first n states of a behavior σ. In particular, $\sigma|_0$ is the empty sequence $\langle\,\rangle$, which satisfies every formula. We write σ_n for the nth state of behavior σ, so σ equals $\langle \sigma_1, \sigma_2, \ldots \rangle$. When σ is finite, we write $last(\sigma)$ for its last state, and $|\sigma|$ for its length.

We let $[\![e]\!]$ denote the meaning of an expression e. When e is a state function, $[\![e]\!]$ is a mapping from states to values; in the special case when e is a state predicate, $[\![e]\!]$ is a mapping from states to truth values. When e is an action, $[\![e]\!]$ is a mapping from pairs of states to truth values. When e is a temporal formula, $[\![e]\!]$ is a mapping from behaviors to truth values. We extended this mapping to finite behaviors by letting $[\![e]\!](\rho) = \mathsf{true}$ iff $[\![e]\!](\sigma) = \mathsf{true}$ for some σ that extends ρ. In all cases, we let $u \models e$ mean $[\![e]\!](u) = \mathsf{true}$. If F is a temporal formula and σ a behavior, then $\sigma \models \mathcal{C}(F)$ iff $\sigma|_n \models F$ for all n. Hence, $[\![\mathcal{C}(F)]\!](\rho) = [\![F]\!](\rho)$ for any finite behavior ρ.

If s and t are states and x is a tuple of variables, we write $s =_x t$ when s and t are identical except possibly for the value they assign to the tuple x. In other words, $s =_x t$ iff $[\![y]\!](s) = [\![y]\!](t)$ for every variable y not in the tuple x. We extend this notion to behaviors, and write $\sigma =_x \tau$ iff $\sigma_n =_x \tau_n$ for all $n > 0$.

The *stutter-free version* of a behavior is the behavior obtained by removing from it all finite repetitions of states; thus, the stutter-free version of $\sigma \circ \langle s, s\rangle \circ \tau$ equals the stutter-free version of $\sigma \circ \langle s\rangle \circ \tau$. Two behaviors are *stuttering equivalent* iff they have the same stutter-free version. Every TLA formula F is *invariant under stuttering*, in the sense that $[\![F]\!](\sigma) = [\![F]\!](\tau)$ for any two stuttering-equivalent behaviors σ and τ. More generally, $[\![F]\!](\sigma) = [\![F]\!](\tau)$ if there is a behavior $\widehat{\tau}$ stuttering equivalent to σ such that $[\![y]\!](\widehat{\tau}_n) = [\![y]\!](\tau_n)$ for all $n > 0$ and all variables

---

y occurring free in F.

We write $\sigma \simeq_x \tau$ when $\widehat{\sigma} =_x \widehat{\tau}$ for some $\widehat{\sigma}$ and $\widehat{\tau}$ stuttering equivalent to σ and τ, respectively. If F is a TLA formula and σ a behavior, we let $[\![\exists\, x : F]\!](\sigma) = \mathsf{true}$ iff there exists a behavior τ such that $\sigma \simeq_x \tau$ and $[\![F]\!](\tau) = \mathsf{true}$. Equivalently, since F is invariant under stuttering, $[\![\exists\, x : F]\!](\sigma) = \mathsf{true}$ iff there exist behaviors $\widehat{\sigma}$ and $\widehat{\tau}$ such that $\widehat{\sigma}$ is stuttering equivalent to σ, $\widehat{\sigma} =_x \widehat{\tau}$, and $[\![F]\!](\widehat{\tau}) = \mathsf{true}$.

An operator H on formulas is *superdiagonal* iff $\models A \Rightarrow H(A)$ for all A in its domain. For example, $\mathcal{C}$ is superdiagonal. As usual, an operator H is *monotonic* iff $\models A \Rightarrow B$ implies $\models H(A) \Rightarrow H(B)$ for all A and B. Antimonotonicity is defined similarly, with the second implication reversed.

A2 Proof Notation

Reliable reasoning about specifications depends on the correctness of the underlying logical proofs. Even a minor error, such as the omission of a hypothesis in a proposition, could allow one to "prove" the correctness of an incorrect implementation. To avoid such errors, we provide detailed, hierarchically structured proofs.

In our proof notation, the theorem to be proved is statement $\langle 0\rangle 1$. The proof of statement $\langle i\rangle j$ is either an ordinary paragraph-style proof or the sequence of statements $\langle i{+}1\rangle 1$, $\langle i{+}1\rangle 2$, ... and their proofs. (The absence of a proof means that the statement follows easily from definitions, previous statements, and assumptions.) Within a proof, $\langle k\rangle l$ denotes the most recent statement with that number. A statement has the form

Assume: *Assump* Prove: *Goal*

which is abbreviated to *Goal* if there is no assumption. The assertion Q.E.D. in statement number $\langle i+1\rangle k$ of the proof of statement $\langle i\rangle j$ denotes the goal of statement $\langle i\rangle j$. The statement

Case: *Assump*

is an abbreviation for

Assume: *Assump* Prove: Q.E.D.

Within the proof of statement $\langle i\rangle j$, assumption $\langle i\rangle$ denotes that statement's assumption, and $\langle i\rangle{:}k$ denotes the assumption's k^{th} item.

We recommend that proofs be read hierarchically, from the top level down. To read the proof of a long level-k step: (i) read the level-$(k{+}1)$ statements that comprise its proof, together with the proof of the final Q.E.D. step (which is usually a short paragraph), and (ii) read the proof of each level-$(k{+}1)$ step, in any desired order.

APPENDIX B. PROOFS

Results are organized in groups that roughly correspond to their subject and to the position of the corresponding discussion in the text.

Our proofs employ many lemmas. We omit the proofs of some of the simpler ones. We also omit the proof of Proposition 1, which is given in [Abadi and Lamport 1994].

B1 Properties of $\mathrel{-\!\!\rhd}$ and $\stackrel{+}{\mathrel{-\!\!\rhd}}$

The proofs of most of these properties are straightforward and are omitted. Some of the basic arguments about $\mathrel{-\!\!\rhd}$ can be found in [Abadi and Plotkin 1993].

Lemma 1. *If P, Q, and R are safety properties, then*
1. $P \mathrel{-\!\!\triangleright} Q$ *and* $P \mathrel{\stackrel{+}{-\!\!\triangleright}} Q$ *are safety properties.*
2. $\models P \Rightarrow (Q \mathrel{-\!\!\triangleright} R)$ *if and only if* $\models P \wedge Q \Rightarrow R$.

Lemma 2. *For any properties P and Q,*
1. $\models (P \mathrel{-\!\!\triangleright} Q) = (\mathcal{C}(P) \mathrel{-\!\!\triangleright} \mathcal{C}(Q)) \wedge (P \Rightarrow Q)$
2. $\models (P \mathrel{\stackrel{+}{-\!\!\triangleright}} Q) = (\mathcal{C}(P) \mathrel{\stackrel{+}{-\!\!\triangleright}} \mathcal{C}(Q)) \wedge (P \Rightarrow Q)$

Lemma 3. *For any properties P and Q,*
1. $\models P \wedge (P \mathrel{-\!\!\triangleright} Q) \Rightarrow Q$
2. $\models P \wedge (P \mathrel{\stackrel{+}{-\!\!\triangleright}} Q) \Rightarrow Q$

Lemma 4. *If P and Q are safety properties, then*

$$\models (P \mathrel{-\!\!\triangleright} Q) \wedge (Q \mathrel{-\!\!\triangleright} P) \;\Rightarrow\; ((P \vee Q) \mathrel{-\!\!\triangleright} (P \wedge Q))$$

Lemma 5. *If P_i and Q_i are safety properties, for $i = 1, \ldots, n$, then*

$$\models \bigwedge_{i=1}^{n} (P_i \mathrel{\stackrel{+}{-\!\!\triangleright}} Q_i) \;\Rightarrow\; ((\bigwedge_{i=1}^{n} P_i) \mathrel{\stackrel{+}{-\!\!\triangleright}} (\bigwedge_{i=1}^{n} Q_i))$$

Lemma 6. *If P is a safety property and Q is any property, then*

$$\models (P \mathrel{\stackrel{+}{-\!\!\triangleright}} Q) = ((Q \mathrel{-\!\!\triangleright} P) \mathrel{-\!\!\triangleright} Q)$$

Lemma 7.
Assume*:* 1. *P, Q, and R are safety properties.*
2. $\models Q \wedge R \Rightarrow P$
Prove*:* $\models (P \mathrel{\stackrel{+}{-\!\!\triangleright}} Q) \Rightarrow (R \mathrel{\stackrel{+}{-\!\!\triangleright}} Q)$

$\langle 1\rangle$1. $\models (Q \mathrel{-\!\!\triangleright} R) \Rightarrow (Q \mathrel{-\!\!\triangleright} P)$
$\langle 2\rangle$1. $\models Q \wedge (Q \mathrel{-\!\!\triangleright} R) \Rightarrow R$
Proof: Lemma 3(1).
$\langle 2\rangle$2. $\models Q \wedge (Q \mathrel{-\!\!\triangleright} R) \Rightarrow (Q \wedge R)$
Proof: $\langle 2\rangle$1 and propositional logic.
$\langle 2\rangle$3. $\models Q \wedge (Q \mathrel{-\!\!\triangleright} R) \Rightarrow P$
Proof: $\langle 2\rangle$2 and assumption $\langle 0\rangle$:2.
$\langle 2\rangle$4. Q.E.D.
Proof: $\langle 2\rangle$3, assumption $\langle 0\rangle$:1, and Lemma 1(2).
$\langle 1\rangle$2. $\models (P \mathrel{\stackrel{+}{-\!\!\triangleright}} Q) \wedge (Q \mathrel{-\!\!\triangleright} P) \Rightarrow Q$
Proof: Assumption $\langle 0\rangle$:1, Lemma 6, and Lemma 3(1).
$\langle 1\rangle$3. $\models (P \mathrel{\stackrel{+}{-\!\!\triangleright}} Q) \wedge (Q \mathrel{-\!\!\triangleright} R) \Rightarrow Q$
Proof: $\langle 1\rangle$2 and $\langle 1\rangle$1.
$\langle 1\rangle$4. $\models (P \mathrel{\stackrel{+}{-\!\!\triangleright}} Q) \Rightarrow ((Q \mathrel{-\!\!\triangleright} R) \mathrel{-\!\!\triangleright} Q)$
Proof: $\langle 1\rangle$3, assumption $\langle 0\rangle$:1, and Lemma 1.
$\langle 1\rangle$5. Q.E.D.
Proof: $\langle 1\rangle$4, assumption $\langle 0\rangle$:1, and Lemma 6.

B2 Closure and Existential Quantification

These results are useful for reasoning about the closure of a quantified formula. This reasoning can be difficult because $\mathcal{C}$ and $\exists$ do not commute.

LEMMA 8. *For any property M and tuple of variables x,*

$$\models \mathcal{C}(\exists\, x : \mathcal{C}(M)) = \mathcal{C}(\exists\, x : M)$$

$\langle 1\rangle 1.$ $\models \mathcal{C}(\exists\, x : M) \Rightarrow \mathcal{C}(\exists\, x : \mathcal{C}(M))$
  PROOF: $\mathcal{C}$ is superdiagonal and both $\mathcal{C}$ and $\exists x$ are monotonic.
$\langle 1\rangle 2.$ $\models \mathcal{C}(\exists\, x : \mathcal{C}(M)) \Rightarrow \mathcal{C}(\exists\, x : M)$
  $\langle 2\rangle 1.$ $\models M \Rightarrow \exists\, x : M$
    PROOF: $\exists$ is superdiagonal.
  $\langle 2\rangle 2.$ $\models \mathcal{C}(M) \Rightarrow \mathcal{C}(\exists\, x : M)$
    PROOF: $\langle 2\rangle 1$ and the monotonicity of $\mathcal{C}$.
  $\langle 2\rangle 3.$ $\models (\exists\, x : \mathcal{C}(M)) \Rightarrow \mathcal{C}(\exists\, x : M)$
    PROOF: $\langle 2\rangle 2$, since x does not occur free in $\mathcal{C}(\exists\, x : M)$.
  $\langle 2\rangle 4.$ Q.E.D.
    PROOF:$\langle 2\rangle 3$ and the monotonicity and idempotence of $\mathcal{C}$.
$\langle 1\rangle 3.$ Q.E.D.

LEMMA 9.
ASSUME*: x_i is a tuple of variables, and no variable in x_i occurs free in M_j, for all $i, j \in \{1, \ldots, n\}$ with $i \neq j$.*
PROVE*:* $\models \bigwedge_i \mathcal{C}(\exists\, x_i : M_i) \Rightarrow \mathcal{C}(\exists\, x_1, \ldots, x_n : \bigwedge_i \mathcal{C}(M_i))$

The proof is by induction on n, setting apart the cases for $n = 1$ and $n = 2$.
$\langle 1\rangle 1.$ CASE: $n = 1$
  PROOF: Immediate from Lemma 8.
$\langle 1\rangle 2.$ CASE: n=2
  LET: $A \triangleq \mathcal{C}(\exists\, x_1, x_2 : \mathcal{C}(M_1) \wedge \mathcal{C}(M_2))$
  $\langle 2\rangle 1.$ $\models \mathcal{C}(M_1) \wedge \mathcal{C}(M_2) \Rightarrow A$
    PROOF: Predicate logic, since $\mathcal{C}$ is superdiagonal.
  $\langle 2\rangle 2.$ $\models \mathcal{C}(M_1) \Rightarrow (\mathcal{C}(M_2) \mathrel{-\!\!\triangleright} A)$
    PROOF: $\langle 2\rangle 1$ and Lemma 1(2).
  $\langle 2\rangle 3.$ $\models M_1 \Rightarrow (\mathcal{C}(M_2) \mathrel{-\!\!\triangleright} A)$
    PROOF: $\langle 2\rangle 2$, since $\mathcal{C}$ is superdiagonal.
  $\langle 2\rangle 4.$ $\models (\exists\, x_1 : M_1) \Rightarrow (\mathcal{C}(M_2) \mathrel{-\!\!\triangleright} A)$
    PROOF: $\langle 2\rangle 3$ and the hypothesis that no variable of x_1 occurs free in M_2.
  $\langle 2\rangle 5.$ $\models \mathcal{C}(\exists\, x_1 : M_1) \Rightarrow (\mathcal{C}(M_2) \mathrel{-\!\!\triangleright} A)$
    PROOF: $\langle 2\rangle 4$ and the monotonicity and idempotence of $\mathcal{C}$, since A is closed by definition and $\mathcal{C}(M_2) \mathrel{-\!\!\triangleright} A$ is closed by Lemma 1(1).
  $\langle 2\rangle 6.$ $\models \mathcal{C}(M_2) \Rightarrow (\mathcal{C}(\exists\, x_1 : M_1) \mathrel{-\!\!\triangleright} A)$
    PROOF: $\langle 2\rangle 5$ and two applications of Lemma 1(2)
  $\langle 2\rangle 7.$ $\models M_2 \Rightarrow (\mathcal{C}(\exists\, x_1 : M_1) \mathrel{-\!\!\triangleright} A)$
    PROOF: $\langle 2\rangle 6$, since $\mathcal{C}$ is superdiagonal.
  $\langle 2\rangle 8.$ $\models (\exists\, x_2 : M_2) \Rightarrow (\mathcal{C}(\exists\, x_1 : M_1) \mathrel{-\!\!\triangleright} A)$
    PROOF: $\langle 2\rangle 7$ and predicate logic.
  $\langle 2\rangle 9.$ $\models \mathcal{C}(\exists\, x_2 : M_2) \Rightarrow (\mathcal{C}(\exists\, x_1 : M_1) \mathrel{-\!\!\triangleright} A)$
    PROOF: $\langle 2\rangle 8$, Lemma 1(1), and the monotonicity and idempotence of $\mathcal{C}$.
  $\langle 2\rangle 10.$ Q.E.D.
    PROOF: $\langle 2\rangle 9$ and Lemma 1(2).
$\langle 1\rangle 3.$ CASE: $n > 2$

ASSUME: $\models \bigwedge_{i=1}^{n-1} \mathcal{C}(\exists\, x_i : M_i) \Rightarrow \mathcal{C}(\exists\, x_1 \ldots x_{n-1} : \bigwedge_{i=1}^{n-1} \mathcal{C}(M_i))$
PROVE: $\models \bigwedge_{i=1}^{n} \mathcal{C}(\exists\, x_i : M_i) \Rightarrow \mathcal{C}(\exists\, x_1 \ldots x_n : \bigwedge_{i=1}^{n} \mathcal{C}(M_i))$
PROOF: $\bigwedge_{i=1}^{n} \mathcal{C}(\exists\, x_i : M_i)$
$\Rightarrow\ \mathcal{C}(\exists\, x_1 \ldots x_{n-1} : \bigwedge_{i=1}^{n-1} \mathcal{C}(M_i)) \wedge \mathcal{C}(\exists\, x_n : M_n)$
by assumption $\langle 1 \rangle$
$=\ \mathcal{C}(\exists\, x_1 \ldots x_{n-1} : \mathcal{C}(\bigwedge_{i=1}^{n-1} \mathcal{C}(M_i))) \wedge \mathcal{C}(\exists\, x_n : M_n)$
a conjunction of safety properties is a safety property
$\Rightarrow\ \mathcal{C}(\exists\, x_1 \ldots x_n : \mathcal{C}(\bigwedge_{i=1}^{n-1} \mathcal{C}(M_i)) \wedge \mathcal{C}(M_n))$
by $\langle 1 \rangle 2$
$=\ \mathcal{C}(\exists\, x_1 \ldots x_n : \bigwedge_{i=1}^{n} \mathcal{C}(M_i))$
a conjunction of safety properties is a safety property

$\langle 1 \rangle 4$. Q.E.D.

PROPOSITION 2.
ASSUME: 1. *x_i is a tuple of variables, and no variable in x_i occurs free in M or M_j, for all i, $j \in \{1, \ldots, n\}$ with $i \neq j$*
2. $\models \bigwedge_{i=1}^{n} \mathcal{C}(M_i) \Rightarrow \exists\, x : \mathcal{C}(M)$
PROVE: $\models \bigwedge_{i=1}^{n} \mathcal{C}(\exists\, x_i : M_i) \Rightarrow \mathcal{C}(\exists\, x : M)$

PROOF: $\bigwedge_{i=1}^{n} \mathcal{C}(\exists\, x_i : M_i)$
$\Rightarrow\ \mathcal{C}(\exists\, x_1 \ldots x_n : \bigwedge_{i=1}^{n} \mathcal{C}(M_i))$
by Lemma 9 and assumption $\langle 0 \rangle$:1
$\Rightarrow\ \mathcal{C}(\exists\, x_1 \ldots x_n : \exists\, x : \mathcal{C}(M))$
by assumption $\langle 0 \rangle$:2 and the monotonicity of $\exists$ and $\mathcal{C}$
$=\ \mathcal{C}(\exists\, x : \mathcal{C}(M))$
by assumption $\langle 0 \rangle$:1
$=\ \mathcal{C}(\exists\, x : M)$
by Lemma 8.

B3 Properties of +

LEMMA 10. *For any state function f, if P is a safety property, then P_{+f} is a safety property.*

PROOF: By the definition of safety properties, it suffices to:
ASSUME: 1. P a safety property.
2. $\forall n : \sigma|_n \models P_{+f}$
PROVE: $\sigma \models P_{+f}$
$\langle 1 \rangle 1$. CASE: $\forall n : \sigma|_n \models P$
PROOF: Assumption $\langle 0 \rangle$:1.
$\langle 1 \rangle 2$. CASE: $\exists n : \neg(\sigma|_n \models P)$
$\langle 2 \rangle 1$. Choose the largest m such that $\sigma|_m \models P$.
PROOF: m exists since $\sigma|_0 \models P$ is true for any σ and P.
$\langle 2 \rangle 2$. $\forall n > m : [\![f]\!](\sigma_n) = [\![f]\!](\sigma_{m+1})$
PROOF: $\langle 2 \rangle 1$, assumption $\langle 0 \rangle$:2, and the definition of P_{+f}.
$\langle 2 \rangle 3$. Q.E.D.
PROOF: $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, and the definition of P_{+f}.
$\langle 1 \rangle 3$. Q.E.D.

LEMMA 11.
ASSUME: 1. *P and Q are safety properties.*
2. *the tuple x includes all the free variables of Q.*
PROVE: $\models (P_{+x} \mathrel{-\!\triangleright} Q) \;=\; (P \overset{+}{\mathrel{-\!\triangleright}} Q)$

$\langle 1\rangle$1. $\models (P_{+x} \mathrel{-\!\triangleright} Q) \;\Rightarrow\; (P \overset{+}{\mathrel{-\!\triangleright}} Q)$

By assumption $\langle 0\rangle$:1, Lemma 1(1), and the definition of $\overset{+}{\mathrel{-\!\triangleright}}$, it suffices to:
ASSUME: 1. For all n, $\sigma|_n \models (P_{+x} \mathrel{-\!\triangleright} Q)$
2. $\sigma|_{n-1} \models P$
PROVE: $\sigma|_n \models Q$

$\langle 2\rangle$1. $\sigma|_n \models P_{+x}$
PROOF: By assumption $\langle 1\rangle$:2 and the definition of P_{+x}.

$\langle 2\rangle$2. Q.E.D.
PROOF: $\langle 2\rangle$1 and assumption $\langle 1\rangle$:1.

$\langle 1\rangle$2. $\models (P \overset{+}{\mathrel{-\!\triangleright}} Q) \;\Rightarrow\; (P_{+x} \mathrel{-\!\triangleright} Q)$

By assumption $\langle 0\rangle$:1, Lemmas 10 and 1(1), and the definition of $\mathrel{-\!\triangleright}$, it suffices to:
ASSUME: 1. For all n, $\sigma|_n \models (P \overset{+}{\mathrel{-\!\triangleright}} Q)$
2. $\sigma|_n \models P_{+x}$
PROVE: $\sigma|_n \models Q$

$\langle 2\rangle$1. Choose $m \leq n$ such that
1. $\sigma|_m \models P$
2. $\forall p : m < p \leq n \Rightarrow [\![x]\!](\sigma_p) = [\![x]\!](\sigma_{m+1})$

PROOF: Assumption $\langle 1\rangle$:2.

$\langle 2\rangle$2. $\sigma|_{m+1} \models Q$
PROOF: $\langle 2\rangle$1.1 and assumption $\langle 1\rangle$:1.

$\langle 2\rangle$3. Q.E.D.
PROOF: $\langle 2\rangle$2, $\langle 2\rangle$1.2, and assumption $\langle 0\rangle$:2, since Q is invariant under stuttering.

$\langle 1\rangle$3. Q.E.D.

LEMMA 12.
ASSUME: 1. *P, Q, and R are safety properties.*
2. $\models R_{+f} \wedge P \Rightarrow Q$
PROVE: $\models (R \overset{+}{\mathrel{-\!\triangleright}} P) \Rightarrow (R \overset{+}{\mathrel{-\!\triangleright}} Q)$

$\langle 1\rangle$1. $\models R \overset{+}{\mathrel{-\!\triangleright}} R_{+f}$

By assumption $\langle 0\rangle$:1 and Lemmas 10 and 1, it suffices to:
ASSUME: $\sigma|_n \models R$
PROVE: $\sigma|_{n+1} \models R_{+f}$

$\langle 2\rangle$1. $\sigma|_{n+1} \circ \langle \sigma_{n+1}, \sigma_{n+1}, \ldots\rangle \models R_{+f}$
PROOF: The definition of R_{+f}.

$\langle 2\rangle$2. Q.E.D.

$\langle 1\rangle$2. Q.E.D.

PROOF: $(R \stackrel{+}{\triangleright} P) \;\Rightarrow\; (R \stackrel{+}{\triangleright} P) \land (R \stackrel{+}{\triangleright} R_{+f})$
by $\langle 1\rangle 1$
$\Rightarrow\; (R \stackrel{+}{\triangleright} (P \land R_{+f}))$
by assumption $\langle 0\rangle$:1 and Lemmas 5 and 10
$\Rightarrow\; (R \stackrel{+}{\triangleright} Q)$
by assumption $\langle 0\rangle$:2 and monotonicity of $\stackrel{+}{\triangleright}$ in its second argument.

LEMMA 13.
ASSUME*: No variable of the tuple x occurs free in v.*
PROVE*:* $\models (\exists x : P_{+v}) = (\exists x : P)_{+v}$

$\langle 1\rangle 1.$ $\models (\exists x : P_{+v}) \Rightarrow (\exists x : P)_{+v}$
ASSUME: $\sigma \models (\exists x : P_{+v})$
PROVE: $\sigma \models (\exists x : P)_{+v}$
$\langle 2\rangle 1.$ Choose $\widehat{\sigma}$ such that $\widehat{\sigma} \simeq_x \sigma$ and $\widehat{\sigma} \models P_{+v}$.
PROOF: Assumption $\langle 1\rangle$ and the definition of $\exists$.
$\langle 2\rangle 2.$ CASE: $\widehat{\sigma} \models P$
$\langle 3\rangle 1.$ $\sigma \models (\exists x : P)$
PROOF: $\langle 2\rangle 1$ and case assumption $\langle 2\rangle$.
$\langle 3\rangle 2.$ Q.E.D.
PROOF: $\langle 3\rangle 1$ and the definition of $(\ldots)_{+v}$.
$\langle 2\rangle 3.$ CASE: There exists $\widehat{\rho}$ and $\widehat{\tau}$ such that $\widehat{\sigma} = \widehat{\rho} \circ \widehat{\tau}$, $\widehat{\rho} \models P$, and $\widehat{\tau} \models \Box[\mathsf{false}]_v$.
$\langle 3\rangle 1.$ Choose ρ and τ such that $\sigma = \rho \circ \tau$, $\rho \simeq_x \widehat{\rho}$, and $\tau \simeq_x \widehat{\tau}$.
PROOF: $\langle 2\rangle 1$ and case assumption $\langle 2\rangle$.
$\langle 3\rangle 2.$ $\rho \models \exists x : P$
PROOF: $\langle 3\rangle 1$ (which asserts $\rho \simeq_x \widehat{\rho}$) and case assumption $\langle 2\rangle$ (which asserts $\widehat{\rho} \models P$).
$\langle 3\rangle 3.$ $\tau \models \Box[\mathsf{false}]_v$
PROOF: $\langle 3\rangle 1$ (which asserts $\tau \simeq_x \widehat{\tau}$), case assumption $\langle 2\rangle$ (which asserts $\widehat{\tau} \models \Box[\mathsf{false}]_v$), and assumption $\langle 0\rangle$.
$\langle 3\rangle 4.$ Q.E.D.
PROOF: $\langle 3\rangle 1$ (which asserts $\sigma = \rho \circ \tau$), $\langle 3\rangle 2$, $\langle 3\rangle 3$, and the definition of $(\ldots)_{+v}$.
$\langle 2\rangle 4.$ Q.E.D.
PROOF: $\langle 2\rangle 1$, $\langle 2\rangle 2$, $\langle 2\rangle 3$, and the definition of $(\ldots)_{+v}$.
$\langle 1\rangle 2.$ $\models (\exists x : P)_{+v} \Rightarrow (\exists x : P_{+v})$
ASSUME: $\sigma \models (\exists x : P)_{+v}$
PROVE: $\sigma \models (\exists x : P_{+v})$
$\langle 2\rangle 1.$ CASE: $\sigma \models (\exists x : P)$
PROOF: Immediate, since $\models P \Rightarrow P_{+v}$ and $\exists$ is monotonic.
$\langle 2\rangle 2.$ CASE: There exist ρ and τ such that $\sigma = \rho \circ \tau$, $\rho \models \exists x : P$, and $\tau \models \Box[\mathsf{false}]_v$.
$\langle 3\rangle 1.$ Choose $\widehat{\rho}$ such that $\widehat{\rho} \simeq_x \rho$ and $\widehat{\rho} \models P$.
PROOF: Case assumption $\langle 2\rangle$ and the definition of $\exists$.
$\langle 3\rangle 2.$ $\widehat{\rho} \circ \tau \models P_{+v}$
PROOF: $\langle 3\rangle 1$, case assumption $\langle 2\rangle$ (which asserts $\tau \models \Box[\mathsf{false}]_v$), and the definition of $(\ldots)_{+v}$.

$\langle 3\rangle 3$. Q.E.D.

PROOF: $\langle 3\rangle 1$, $\langle 3\rangle 2$, and case assumption $\langle 2\rangle$, which imply $\widehat{\rho} \circ \tau \simeq_x \sigma$.

$\langle 2\rangle 3$. Q.E.D.

PROOF: $\langle 2\rangle 1$, $\langle 2\rangle 2$, and the definition of $(\ldots)_{+v}$.

$\langle 1\rangle 3$. Q.E.D.

LEMMA 14. *If s is a variable that does not occur in Init, $\mathcal{N}$, w, or v, and*

$$\begin{aligned} \widehat{Init} &\triangleq (Init \wedge (s = 0)) \vee (\neg Init \wedge (s = 1)) \\ \widehat{\mathcal{N}} &\triangleq \vee\, (s = 0) \wedge \vee\, (s' = 0) \wedge (\mathcal{N} \vee (w' = w)) \\ &\qquad\qquad\qquad\;\; \vee\, (s' = 1) \wedge \neg(\mathcal{N} \vee (w' = w)) \\ &\quad\;\; \vee\, (s = 1) \wedge (s' = 1) \wedge (v' = v) \end{aligned}$$

then $\models (Init \wedge \Box[\mathcal{N}]_w)_{+v} \;=\; \exists s : \widehat{Init} \wedge \Box[\widehat{\mathcal{N}}]_{\langle w,\, v,\, s\rangle}$.

$\langle 1\rangle 1$. $\models (Init \wedge \Box[\mathcal{N}]_w)_{+v} \Rightarrow \exists s : \widehat{Init} \wedge \Box[\widehat{\mathcal{N}}]_{\langle w,\, v,\, s\rangle}$

ASSUME: $\sigma \models (Init \wedge \Box[\mathcal{N}]_w)_{+v}$

PROVE: $\sigma \models \exists s : \widehat{Init} \wedge \Box[\widehat{\mathcal{N}}]_{\langle w,\, v,\, s\rangle}$

LET: $\widehat{\sigma}$ be the behavior such that $\widehat{\sigma} =_s \sigma$ and, for all $n > 0$:

$$[\![s]\!](\widehat{\sigma}_n) \;\triangleq\; \textbf{if } \sigma|_n \models (Init \wedge \Box[\mathcal{N}]_w) \textbf{ then } 0 \textbf{ else } 1$$

$\langle 2\rangle 1$. $\widehat{\sigma} \models \widehat{Init} \wedge \Box[\widehat{\mathcal{N}}]_{\langle w,\, v,\, s\rangle}$

$\langle 3\rangle 1$. $\widehat{\sigma} \models \widehat{Init}$

PROOF: The definitions of $\widehat{\sigma}$ and $\widehat{Init}$, assumption $\langle 1\rangle$, and the hypothesis that s does not occur in *Init*.

$\langle 3\rangle 2$. $\widehat{\sigma} \models \Box[\widehat{\mathcal{N}}]_{\langle w,\, v,\, s\rangle}$

$\langle 4\rangle 1$. CASE: $\sigma \models Init \wedge \Box[\mathcal{N}]_w$

$\langle 5\rangle 1$. $\widehat{\sigma} \models \Box[(s = 0) \wedge (s' = 0) \wedge (\mathcal{N} \vee (w' = w))]_{\langle w,\, v,\, s\rangle}$

PROOF: The definition of $\widehat{\sigma}$, case assumption $\langle 4\rangle$, and the hypothesis that s does not occur in $\mathcal{N}$ or w.

$\langle 5\rangle 2$. Q.E.D.

PROOF: $\langle 5\rangle 1$ and the definition of $\widehat{\mathcal{N}}$.

$\langle 4\rangle 2$. CASE: $\sigma \not\models Init$

$\langle 5\rangle 1$. $\sigma \models \Box\,[\mathsf{false}]_v$

PROOF: Case assumption $\langle 4\rangle$, assumption $\langle 1\rangle$, and the definition of $(\ldots)_{+v}$.

$\langle 5\rangle 2$. $\widehat{\sigma} \models \Box\,(s = 1) \wedge \Box\,[\mathsf{false}]_v$.

PROOF: $\langle 5\rangle 1$ and the definition of $\widehat{\sigma}$.

$\langle 5\rangle 3$. Q.E.D.

PROOF: $\langle 5\rangle 2$ and the definition of $\widehat{\mathcal{N}}$.

$\langle 4\rangle 3$. CASE: $\sigma \models Init$ and $\sigma \not\models Init \wedge \Box[\mathcal{N}]_w$.

$\langle 5\rangle 1$. Choose ρ and τ with $|\rho| > 0$ such that

1. $\sigma = \rho \circ \tau$,
2. $\rho \models Init \wedge \Box[\mathcal{N}]_w$
3. $\rho \circ \langle \tau_1 \rangle \not\models \Box[\mathcal{N}]_w$
4. $\tau \models \Box\,[\mathsf{false}]_v$

PROOF: Case assumption $\langle 4\rangle$, assumption $\langle 1\rangle$, and the definition of $(\ldots)_{+v}$.

$\langle 5\rangle 2$. Choose $\widehat{\rho}$ and $\widehat{\tau}$ such that $\widehat{\sigma} = \widehat{\rho} \circ \widehat{\tau}$, $\widehat{\rho} =_s \rho$, and $\widehat{\tau} =_s \tau$.
PROOF: The definition of $\widehat{\sigma}$ and $\langle 5\rangle 1.1$.
$\langle 5\rangle 3$. $\widehat{\rho} \models \Box[(s = 0) \wedge (s' = 0) \wedge (\mathcal{N} \vee (w' = w))]_{\langle w, v, s\rangle}$
PROOF: The definition of $\widehat{\sigma}$, $\langle 5\rangle 1.2$, $\langle 5\rangle 2$, and the hypothesis that s does not occur in $\mathcal{N}$ or w.
$\langle 5\rangle 4$. $\langle \mathit{last}(\widehat{\rho}), \widehat{\tau}_1\rangle \models (s = 0) \wedge (s' = 1) \wedge \neg\mathcal{N} \wedge (w' \neq w)$
PROOF: The definition of $\widehat{\sigma}$, $\langle 5\rangle 1.3$, $\langle 5\rangle 2$, and the hypothesis that s does not occur in $\mathcal{N}$ or w.
$\langle 5\rangle 5$. $\widehat{\tau} \models \Box\,(s = 1) \wedge \Box\,[\mathsf{false}]_v$
PROOF: The definition of $\widehat{\sigma}$,, $\langle 5\rangle 1.4$, $\langle 5\rangle 2$, and the hypothesis that s does not occur in v.
$\langle 5\rangle 6$. Q.E.D.
PROOF: $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$, and $\langle 5\rangle 5$, and the definition of $\widehat{\mathcal{N}}$.
$\langle 4\rangle 4$. Q.E.D.
PROOF: $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, assumption $\langle 1\rangle$, and the definition of $(\ldots)_{+v}$.
$\langle 3\rangle 3$. Q.E.D.
$\langle 2\rangle 2$. Q.E.D.
PROOF: The definition of $\widehat{\sigma}$, $\langle 2\rangle 1$, and the definition of $\boldsymbol{\exists}$.

$\langle 1\rangle 2$. $\models \boldsymbol{\exists} s : \widehat{\mathit{Init}} \wedge \Box[\widehat{\mathcal{N}}]_{\langle w, v, s\rangle} \Rightarrow (\mathit{Init} \wedge \Box[\mathcal{N}]_w)_{+v}$
ASSUME: $\sigma \models \boldsymbol{\exists} s : \widehat{\mathit{Init}} \wedge \Box[\widehat{\mathcal{N}}]_{\langle w, v, s\rangle}$
PROVE: $\sigma \models (\mathit{Init} \wedge \Box[\mathcal{N}]_w)_{+v}$
$\langle 2\rangle 1$. Choose $\widehat{\sigma}$ such that $\widehat{\sigma} \simeq_s \sigma$ and $\widehat{\sigma} \models \widehat{\mathit{Init}} \wedge \Box[\widehat{\mathcal{N}}]_{\langle w, v, s\rangle}$.
PROOF: Assumption $\langle 1\rangle$ and the definition of $\boldsymbol{\exists}$.
$\langle 2\rangle 2$. $\widehat{\sigma} \models (\mathit{Init} \wedge \Box[\mathcal{N}]_w)_{+v}$
$\langle 3\rangle 1$. CASE: $\widehat{\sigma} \models \Box\,(s = 0)$
$\langle 4\rangle 1$. $\widehat{\sigma} \models (\mathit{Init} \wedge \Box[\mathcal{N}]_w)$
PROOF: $\langle 2\rangle 1$, case assumption $\langle 3\rangle$, and the definitions of $\widehat{\mathit{Init}}$ and $\widehat{\mathcal{N}}$.
$\langle 4\rangle 2$. Q.E.D.
PROOF: $\langle 4\rangle 1$, since the operator $(\ldots)_{+v}$ is superdiagonal.
$\langle 3\rangle 2$. CASE: $\widehat{\sigma} \models \Box\,(s = 1)$
$\langle 4\rangle 1$. $\widehat{\sigma} \models \Box\,[\mathsf{false}]_v$
PROOF: $\langle 2\rangle 1$, the definition of $\widehat{\mathcal{N}}$, and case assumption $\langle 3\rangle$.
$\langle 4\rangle 2$. Q.E.D.
PROOF: $\langle 4\rangle 1$ and the definition of $(\ldots)_{+v}$.
$\langle 3\rangle 3$. CASE: $\widehat{\sigma} \not\models \Box\,(s = 0)$ and $\widehat{\sigma} \not\models \Box\,(s = 1)$
$\langle 4\rangle 1$. Choose $\widehat{\rho}$ and $\widehat{\tau}$ with $|\widehat{\rho}| > 0$ such that
1. $\widehat{\sigma} = \widehat{\rho} \circ \widehat{\tau}$
2. $\widehat{\rho} \models \Box\,(s = 0)$
3. $\widehat{\tau} \models \Box\,(s = 1)$,

PROOF: Case assumption $\langle 3\rangle$, $\langle 2\rangle 1$, and the definitions of $\widehat{\mathit{Init}}$ and $\widehat{\mathcal{N}}$.
$\langle 4\rangle 2$. $\widehat{\rho} \models (\mathit{Init} \wedge \Box[\mathcal{N}]_w)$
PROOF: $\langle 2\rangle 1$, $\langle 4\rangle 1.2$, and the definitions of $\widehat{\mathit{Init}}$ and $\widehat{\mathcal{N}}$.
$\langle 4\rangle 3$. $\widehat{\tau} \models \Box\,[\mathsf{false}]_v$
PROOF: $\langle 2\rangle 1$, $\langle 4\rangle 1.3$, and the definition of $\widehat{\mathcal{N}}$.
$\langle 4\rangle 4$. Q.E.D.

Proof: $\langle 4\rangle 1.1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 3\rangle$, and the definition of $(\ldots)_{+v}$.

$\langle 3\rangle 4$. Q.E.D.

$\langle 2\rangle 3$. Q.E.D.

Proof: $\langle 2\rangle 1$, $\langle 2\rangle 2$, and the hypothesis that s does not occur in *Init*, $\mathcal{N}$, w, or v.

$\langle 1\rangle 3$. Q.E.D.

Proposition 3. *If x is a tuple of variables none of which occurs in v, and s is a variable that does not occur in Init, $\mathcal{N}$, w, v, or x, and*

$$\begin{array}{lll}\widehat{Init} & \triangleq & (Init \wedge (s = 0)) \vee (\neg Init \wedge (s = 1)) \\ \widehat{\mathcal{N}} & \triangleq & \vee\ (s = 0) \wedge \vee\ (s' = 0) \wedge (\mathcal{N} \vee (w' = w)) \\ & & \vee\ (s' = 1) \wedge \neg(\mathcal{N} \vee (w' = w)) \\ & & \vee\ (s = 1) \wedge (s' = 1) \wedge (v' = v)\end{array}$$

then $\models (\exists\, x : Init \wedge \Box[\mathcal{N}]_w)_{+v} \;=\; \exists\, x, s : \widehat{Init} \wedge \Box[\widehat{\mathcal{N}}]_{\langle w,\, v,\, s\rangle}$.

Proof: Follows immediately from Lemmas 13 and 14.

B4 Properties of $\perp$

Lemma 15.

1. *For any properties P and Q, $\models P \perp Q = \mathcal{C}(P) \perp \mathcal{C}(Q)$.*
2. *If P and Q are safety properties, then $\models P \perp Q = (P \wedge Q) \overset{+}{\Rightarrow} (P \vee Q)$.*

Lemma 16. *For any properties P and Q,*

$$\models (P \overset{+}{\Rightarrow} Q) \;=\; (P \Rightarrow Q) \wedge (P \perp Q)$$

$\langle 1\rangle 1$. Case: P and Q safety properties

$\langle 2\rangle 1$. $\models (P \overset{+}{\Rightarrow} Q) \Rightarrow (P \Rightarrow Q) \wedge (P \perp Q)$

$\langle 3\rangle 1$. $\models (P \overset{+}{\Rightarrow} Q) \Rightarrow (P \Rightarrow Q)$

Proof: Obvious from the definitions of $\Rightarrow$ and $\overset{+}{\Rightarrow}$.

$\langle 3\rangle 2$. $\models (P \overset{+}{\Rightarrow} Q) \Rightarrow (P \perp Q)$

Proof: Lemma 15(2), since $\overset{+}{\Rightarrow}$ is monotonic in its second argument and antimonotonic in its first.

$\langle 3\rangle 3$. Q.E.D.

$\langle 2\rangle 2$. $\models (P \Rightarrow Q) \wedge (P \perp Q) \;\Rightarrow\; (P \overset{+}{\Rightarrow} Q)$

$\langle 3\rangle 1$. $\models (P \Rightarrow Q) \wedge (P \perp Q) \wedge (Q \Rightarrow P) \;\Rightarrow\; Q$

Proof:

$$\begin{array}{ll} & (P \perp Q) \wedge (P \Rightarrow Q) \wedge (Q \Rightarrow P) \\ = & (((P \vee Q) \Rightarrow (P \wedge Q)) \Rightarrow (P \vee Q)) \wedge (P \Rightarrow Q) \wedge (Q \Rightarrow P) \\ & \quad \text{case assumption } \langle 1\rangle\text{, Lemma 15(2), and Lemma 6} \\ \Rightarrow & (((P \vee Q) \Rightarrow (P \wedge Q)) \Rightarrow (P \vee Q)) \wedge ((P \vee Q) \Rightarrow (P \wedge Q)) \\ & \quad \text{by Lemma 4} \\ \Rightarrow & (P \vee Q) \wedge ((P \vee Q) \Rightarrow (P \wedge Q)) \\ & \quad \text{by Lemma 3(1)} \\ \Rightarrow & Q \\ & \quad \text{by Lemma 3(1).}\end{array}$$

$\langle 3\rangle 2$. $\models (P \Rightarrow Q) \wedge (P \perp Q) \;\Rightarrow\; ((Q \Rightarrow P) \Rightarrow Q)$

Proof: $\langle 3\rangle 1$ and Lemma 1(2).

$\langle 3\rangle 3.$ Q.E.D.

PROOF: $\langle 3\rangle 2$, case assumption $\langle 1\rangle$, and Lemma 6.

$\langle 1\rangle 2.$ Q.E.D.

PROOF:
$$
\begin{aligned}
P \overset{+}{\rightarrow\!\!\triangleright} Q &= (P \Rightarrow Q) \wedge (\mathcal{C}(P) \overset{+}{\rightarrow\!\!\triangleright} \mathcal{C}(Q)) \\
&\qquad \text{by Lemma 2(2)} \\
&= (P \Rightarrow Q) \wedge (\mathcal{C}(P) \rightarrow\!\!\triangleright \mathcal{C}(Q)) \wedge (\mathcal{C}(P) \perp \mathcal{C}(Q)) \\
&\qquad \text{by } \langle 1\rangle 1 \\
&= (P \Rightarrow Q) \wedge (\mathcal{C}(P) \rightarrow\!\!\triangleright \mathcal{C}(Q)) \wedge (P \perp Q) \\
&\qquad \text{by Lemma 15(1)} \\
&= (P \rightarrow\!\!\triangleright Q) \wedge (P \perp Q) \\
&\qquad \text{by Lemma 2(1).}
\end{aligned}
$$

PROPOSITION 4.

ASSUME: 1. *P, Q, and R are safety properties.*
2. $\models P \wedge Q \Rightarrow R$
3. $\models Q \Rightarrow P \perp R$
4. *the tuple x contains all the free variables of R.*

PROVE: $\models P_{+x} \wedge Q \Rightarrow R$

$\langle 1\rangle 1.$ $\models Q \Rightarrow (P \rightarrow\!\!\triangleright R)$

PROOF: Assumptions $\langle 0\rangle$:1 and $\langle 0\rangle$:2, and Lemma 1(2).

$\langle 1\rangle 2.$ $\models Q \Rightarrow (P \overset{+}{\rightarrow\!\!\triangleright} R)$

PROOF: $\langle 1\rangle 1$, assumption $\langle 0\rangle$:3, and Lemma 16.

$\langle 1\rangle 3.$ $\models Q \Rightarrow (P_{+x} \rightarrow\!\!\triangleright R)$

PROOF: $\langle 1\rangle 2$, assumptions $\langle 0\rangle$:1 and $\langle 0\rangle$:4, and Lemma 11.

$\langle 1\rangle 4.$ Q.E.D.

PROOF: $\langle 1\rangle 3$ and Lemma 1(2).

LEMMA 17.

LET: $E \triangleq \mathit{Init}_E \wedge \Box[\mathcal{N}_E]_{\langle x,\, e\rangle}$
$M \triangleq \mathit{Init}_M \wedge \Box[\mathcal{N}_M]_{\langle y,\, m\rangle}$

PROVE: $\models ((\boldsymbol{\exists}\, x : \mathit{Init}_E) \vee (\boldsymbol{\exists}\, y : \mathit{Init}_M)) \wedge \mathit{Disjoint}(e, m) \Rightarrow$
$\mathcal{C}(\boldsymbol{\exists}\, x : E) \perp \mathcal{C}(\boldsymbol{\exists}\, y : M)$

PROOF: By definition of $\perp$, it suffices to prove the following, for all $n \geq 0$:

ASSUME: 1. $\sigma \models ((\exists\, x : \mathit{Init}_E) \vee (\exists\, y : \mathit{Init}_M))$
2. $\sigma \models \mathit{Disjoint}(e, m)$
3. $\sigma|_n \models \mathcal{C}(\boldsymbol{\exists}\, x : E) \wedge \mathcal{C}(\boldsymbol{\exists}\, y : M)$

PROVE: $\sigma|_{n+1} \models \mathcal{C}(\boldsymbol{\exists}\, x : E) \vee \mathcal{C}(\boldsymbol{\exists}\, y : M)$

$\langle 1\rangle 1.$ CASE: $n = 0$

$\langle 2\rangle 1.$ CASE: $\sigma \models (\exists\, x : \mathit{Init}_E)$

$\langle 3\rangle 1.$ Choose a state s such that $s =_x \sigma_1$ and $s \models \mathit{Init}_E$.

PROOF: Case assumption $\langle 2\rangle$.

$\langle 3\rangle 2.$ $\langle s, s, s, \ldots\rangle \models E$

PROOF: $\langle 3\rangle 1$ and the definition of E.

$\langle 3\rangle 3.$ $\langle \sigma_1, s, s, s, \ldots\rangle \models \boldsymbol{\exists}\, x : E$

PROOF: $\langle 3\rangle 1$, $\langle 3\rangle 2$, and the definition of $\boldsymbol{\exists}$.

$\langle 3\rangle 4.$ $\sigma|_1 \models \boldsymbol{\exists} x : E$

PROOF: $\langle 3\rangle 3$.

$\langle 3\rangle 5.$ Q.E.D.

$\langle 2\rangle 2.$ CASE: $\sigma \models (\exists\, x : \mathit{Init}_M)$

PROOF: The proof is the same as the proof of $\langle 2\rangle 1$, with M substituted for E and y substituted for x.

$\langle 2\rangle 3.$ Q.E.D.

PROOF: $\langle 2\rangle 1$, $\langle 2\rangle 2$, and assumption $\langle 0\rangle$:1.

$\langle 1\rangle 2.$ CASE: $n > 0$

$\langle 2\rangle 1.$ $([\![e]\!](\sigma_n) = [\![e]\!](\sigma_{n+1})) \;\vee\; ([\![m]\!](\sigma_n) = [\![m]\!](\sigma_{n+1}))$

PROOF: Assumption $\langle 0\rangle$:2.

$\langle 2\rangle 2.$ CASE: $[\![e]\!](\sigma_n) = [\![e]\!](\sigma_{n+1})$

$\langle 3\rangle 1.$ Choose ρ such that:

1. $\rho \simeq_x \sigma|_n$
2. $\rho \models E$

PROOF: Assumption $\langle 0\rangle$:3, since $\eta \models \mathcal{C}(P)$ iff $\eta \models P$, for any property P and finite behavior η.

LET: t be the state such that $t =_x \sigma_{n+1}$ and $[\![x]\!](t) = [\![x]\!](\mathit{last}(\rho))$.

$\langle 3\rangle 2.$ $\rho \circ \langle t\rangle \models E$

PROOF: $\langle 3\rangle 1.2$, case assumption $\langle 2\rangle$, and the definitions of t and E.

$\langle 3\rangle 3.$ $\sigma|_{n+1} \simeq_x \rho \circ \langle t\rangle$

PROOF: $\langle 3\rangle 1.1$ and the definition of t.

$\langle 3\rangle 4.$ $\sigma|_{n+1} \models \boldsymbol{\exists} x : E$

PROOF: $\langle 3\rangle 2$ and $\langle 3\rangle 3$.

$\langle 3\rangle 5.$ Q.E.D.

PROOF: $\langle 3\rangle 4$.

$\langle 2\rangle 3.$ CASE: $[\![m]\!](\sigma_n) = [\![m]\!](\sigma_{n+1})$

PROOF: The proof is the same as the proof of $\langle 2\rangle 2$, with m, M, and y substituted for e, E, and x, respectively.

$\langle 2\rangle 4.$ Q.E.D.

PROOF: $\langle 2\rangle 1$, $\langle 2\rangle 2$, and $\langle 2\rangle 3$.

$\langle 1\rangle 3.$ Q.E.D.

PROPOSITION 5.

ASSUME: 1. $\models \mathcal{C}(E) = \mathit{Init}_E \wedge \Box\,[\mathcal{N}_E]_{\langle x,\, e\rangle}$

2. $\models \mathcal{C}(M) = \mathit{Init}_M \wedge \Box\,[\mathcal{N}_M]_{\langle y,\, m\rangle}$

PROVE: $\models ((\boldsymbol{\exists}\, x : \mathit{Init}_E) \vee (\boldsymbol{\exists}\, y : \mathit{Init}_M)) \wedge \mathit{Disjoint}(e, m) \Rightarrow \mathcal{C}(\boldsymbol{\exists}\, x : E) \perp \mathcal{C}(\boldsymbol{\exists}\, y : M)$

PROOF: Follows from Lemma 17, with $\mathcal{C}(E)$ substituted for E and $\mathcal{C}(M)$ substituted for M, and Lemma 8.

B5 Composition as Conjunction

PROPOSITION 6. *Let $m_1, \ldots, m_n$, $x_1, \ldots, x_n$ be tuples of variables, and let*

$$m \;\triangleq\; \langle m_1, \ldots, m_n\rangle \qquad\qquad x \;\triangleq\; \langle x_1, \ldots, x_n\rangle$$

$$\widehat{x}_i \;\triangleq\; \langle x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\rangle$$

$$M_i \;\triangleq\; \boldsymbol{\exists}\, x_i : \mathit{Init}_i \;\wedge\; \Box[\mathcal{N}_i]_{\langle m_i,\, x_i\rangle} \;\wedge\; L_i$$

ASSUME: *For all i, j with $i \neq j$:*

1. *no variable of x_j occurs free in x_i or M_i.*

2. *m includes all free variables of M_i.*
3. $\models \mathcal{N}_i \Rightarrow (m'_j = m_j)$

PROVE: $\models \bigwedge_{i=1}^{n} M_i \;=\; \exists x : \bigwedge_{i=1}^{n} \mathit{Init}_i \;\wedge\; \Box[\bigvee_{i=1}^{n} \mathcal{N}_i \wedge (\widehat{x}'_i = \widehat{x}_i)]_{\langle m,\, x\rangle} \;\wedge\; \bigwedge_{i=1}^{n} L_i$

PROOF: The hypotheses remain true and the conclusion is unchanged if we remove from m_j any variable that appears in x_j. (Assumption 2 remains true because, by assumption 1, the variable removed cannot occur free in M_i.) Therefore, without loss of generality, we can strengthen assumption 1 to:

ASSUME: 1(a). The variables in x_j do not occur free in M_i, and are distinct from the variables in x_i and m_j.

The proof is by induction on n, with the cases for $n = 1$ and $n = 2$ proved separately.

$\langle 1\rangle 1.$ CASE: $n = 1$

PROOF: This case follows immediately from the definition of M_i.

$\langle 1\rangle 2.$ CASE: $n = 2$

LET: $$\mathcal{N}_H \triangleq \begin{bmatrix} \vee\, \mathcal{N}_1 \wedge (x'_2 = x_2) \\ \vee\, \mathcal{N}_2 \wedge (x'_1 = x_1) \end{bmatrix}_{\langle m,\, x\rangle}$$
$$\mathcal{N}_U \triangleq \begin{bmatrix} \vee\, \mathcal{N}_1 \wedge (x'_2 = x_2) \\ \vee\, \mathcal{N}_2 \wedge (x'_1 = x_1) \\ \vee\, \mathcal{N}_1 \wedge \mathcal{N}_2 \end{bmatrix}_{\langle m,\, x\rangle}$$
$$H \triangleq \exists x_1, x_2 : \mathit{Init}_1 \wedge \mathit{Init}_2 \wedge \Box \mathcal{N}_H \wedge L_1 \wedge L_2$$
$$U \triangleq \mathit{Init}_1 \wedge \mathit{Init}_2 \wedge \Box \mathcal{N}_U \wedge L_1 \wedge L_2$$

PROVE: $\models M_1 \wedge M_2 = H$

$\langle 2\rangle 1.$ $\models M_1 \wedge M_2 \;=\; \exists x_1, x_2 : U$

LET: $$\mathcal{N}_V \triangleq \begin{bmatrix} \wedge\, \mathcal{N}_1 \vee (\langle m_1,\, x_1\rangle' = \langle m_1,\, x_1\rangle) \\ \wedge\, \mathcal{N}_2 \vee (\langle m_2,\, x_2\rangle' = \langle m_2,\, x_2\rangle) \end{bmatrix}_{\langle m,\, x\rangle}$$
$$V \triangleq \mathit{Init}_1 \wedge \mathit{Init}_2 \wedge \Box \mathcal{N}_V \wedge L_1 \wedge L_2$$

$\langle 3\rangle 1.$ $\models \mathcal{N}_V = \mathcal{N}_U$

PROOF: Assumption $\langle 0\rangle$:3, which implies
$$\models \mathcal{N}_2 \wedge (\langle m_1,\, x_1\rangle' = \langle m_1,\, x_1\rangle) = \mathcal{N}_2 \wedge (x'_1 = x_1)$$
$$\models \mathcal{N}_1 \wedge (\langle m_2,\, x_2\rangle' = \langle m_2,\, x_2\rangle) = \mathcal{N}_1 \wedge (x'_2 = x_2)$$

$\langle 3\rangle 2.$ $\models V = U$

PROOF: $\langle 3\rangle 1$ and the definitions of V and U.

$\langle 3\rangle 3.$ $\models [\mathcal{N}_1]_{\langle m_1,\, x_1\rangle} \wedge [\mathcal{N}_2]_{\langle m_2,\, x_2\rangle} = \mathcal{N}_V$

PROOF: The definition of m and x.

$\langle 3\rangle 4.$ $\models M_1 \wedge M_2 \;=\; \exists x_1, x_2 : V$

PROOF: $\langle 3\rangle 3$ and assumption $\langle 0\rangle$:1(a), since $\Box$ distributes over $\wedge$.

$\langle 3\rangle 5.$ Q.E.D.

PROOF: $\langle 3\rangle 2$ and $\langle 3\rangle 4$.

$\langle 2\rangle 2.$ $\models H \Rightarrow M_1 \wedge M_2$

PROOF: $\langle 2\rangle 1$, since $\models \mathcal{N}_H \Rightarrow \mathcal{N}_U$.

$\langle 2\rangle 3.$ $\models M_1 \wedge M_2 \Rightarrow H$

ASSUME: $\sigma \models M_1 \wedge M_2$

PROVE: $\sigma \models H$

$\langle 3\rangle 1.$ Choose τ such that $\tau \simeq_{\langle x_1,\, x_2\rangle} \sigma$ and $\tau \models U$.

PROOF: τ exists by assumption $\langle 2\rangle$, $\langle 2\rangle 1$, and the definition of $\exists$.

LET: η be the behavior such that, for all $n > 0$:

$$\begin{array}{lcl} \eta_{2n-1} & \triangleq & \tau_n \\ \eta_{2n} & \triangleq & \textbf{if } [\![x_1]\!](\tau_n) = [\![x_1]\!](\tau_{n+1}) \text{ or } [\![x_2]\!](\tau_n) = [\![x_2]\!](\tau_{n+1}) \\ & & \quad \textbf{then } \tau_n \\ & & \quad \textbf{else } \text{the state such that } [\![x_1]\!](\eta_{2n}) = [\![x_1]\!](\tau_{n+1}) \\ & & \qquad\quad \text{and } \eta_{2n} =_{x_1} \tau_n. \end{array}$$

(η is the same as τ except that each step is split in two. A step that changes both x_1 and x_2 is split into a step that changes only x_1 followed by one that leaves x_1 unchanged. For a step that leaves x_1 or x_2 unchanged, a stuttering step is added.)

$\langle 3\rangle 2$. For all $n > 0$, if $[\![x_1]\!](\tau_n) \neq [\![x_1]\!](\tau_{n+1})$ and $[\![x_2]\!](\tau_n) \neq [\![x_2]\!](\tau_{n+1})$ then $\langle \tau_n, \tau_{n+1}\rangle$ is an $\mathcal{N}_1 \wedge \mathcal{N}_2 \wedge (m' = m)$ step.

ASSUME: $[\![x_1]\!](\tau_n) \neq [\![x_1]\!](\tau_{n+1})$ and $[\![x_2]\!](\tau_n) \neq [\![x_2]\!](\tau_{n+1})$.

PROVE: $\langle \tau_n, \tau_{n+1}\rangle$ is an $\mathcal{N}_1 \wedge \mathcal{N}_2 \wedge (m' = m)$ step.

$\langle 4\rangle 1$. $\langle \tau_n, \tau_{n+1}\rangle$ is an $\mathcal{N}_U$ step.

PROOF: $\langle 3\rangle 1$ (which asserts $\tau \models U$) and the definition of U.

$\langle 4\rangle 2$. $\langle \tau_n, \tau_{n+1}\rangle$ is an $\mathcal{N}_1 \wedge \mathcal{N}_2$ step.

PROOF: Assumption $\langle 3\rangle$, $\langle 4\rangle 1$, and the definition of $\mathcal{N}_U$.

$\langle 4\rangle 3$. Q.E.D.

PROOF: $\langle 4\rangle 2$ and assumption $\langle 0\rangle$:3.

$\langle 3\rangle 3$. For all $n > 0$, $\langle \eta_n, \eta_{n+1}\rangle$ is an $\mathcal{N}_H$ step.

LET: $k \triangleq (n+1) \text{ div } 2$

$\langle 4\rangle 1$. CASE: $[\![x_1]\!](\tau_k) = [\![x_1]\!](\tau_{k+1})$ or $[\![x_2]\!](\tau_k) = [\![x_2]\!](\tau_{k+1})$
(In this case, $\langle \eta_n, \eta_{n+1}\rangle$ is a step of τ or a stutter.)

$\langle 5\rangle 1$. $\langle \eta_n, \eta_{n+1}\rangle = \langle \tau_k, \tau_{k+1}\rangle$ or $\eta_n = \eta_{n+1}$.

PROOF: The definition of η and case assumption $\langle 4\rangle$.

$\langle 5\rangle 2$. $[\![x_1]\!](\eta_n) = [\![x_1]\!](\eta_{n+1})$ or $[\![x_2]\!](\eta_n) = [\![x_2]\!](\eta_{n+1})$.

PROOF: $\langle 5\rangle 1$ and case assumption $\langle 4\rangle$.

$\langle 5\rangle 3$. $\langle \eta_n, \eta_{n+1}\rangle$ is an $\mathcal{N}_U$ step.

PROOF: $\langle 5\rangle 1$, $\langle 3\rangle 1$ (which asserts $\tau \models U$), and the definition of U.

$\langle 5\rangle 4$. Q.E.D.

PROOF: $\langle 5\rangle 2$ and $\langle 5\rangle 3$, since $\models \mathcal{N}_U \wedge ((x_1' = x_1) \vee (x_2' = x_2)) \Rightarrow \mathcal{N}_H$.

$\langle 4\rangle 2$. CASE: $n = 2k-1$, $[\![x_1]\!](\tau_k) \neq [\![x_1]\!](\tau_{k+1})$, and
$[\![x_2]\!](\tau_k) \neq [\![x_2]\!](\tau_{k+1})$.
(In this case, $\langle \eta_n, \eta_{n+1}\rangle$ is a step that changes only x_1.)

$\langle 5\rangle 1$. $\eta_n = \tau_k$, $[\![x_1]\!](\eta_{n+1}) = [\![x_1]\!](\tau_{k+1})$, and $\eta_{n+1} =_{x_1} \tau_k$

PROOF: The definition of η and case assumption $\langle 4\rangle$.

$\langle 5\rangle 2$. $\langle \tau_k, \tau_{k+1}\rangle$ is an $\mathcal{N}_1 \wedge \mathcal{N}_2 \wedge (m' = m)$ step.

PROOF: $\langle 3\rangle 2$ and case assumption $\langle 4\rangle$.

$\langle 5\rangle 3$. $[\![m]\!](\eta_n) = [\![m]\!](\tau_k)$ and $[\![m]\!](\eta_{n+1}) = [\![m]\!](\tau_{k+1})$

PROOF: $\langle 5\rangle 1$ implies $[\![m]\!](\eta_n) = [\![m]\!](\tau_k)$, $\langle 5\rangle 1$ and assumption $\langle 0\rangle$:1(a) (which implies that no variable in x_1 occurs in m_1 or m_2) imply $[\![m]\!](\eta_{n+1}) = [\![m]\!](\tau_k)$, and $\langle 5\rangle 2$ implies $[\![m]\!](\tau_k) = [\![m]\!](\tau_{k+1})$.

$\langle 5\rangle 4$. $[\![x_1]\!](\eta_n) = [\![x_1]\!](\tau_k)$ and $[\![x_1]\!](\eta_{n+1}) = [\![x_1]\!](\tau_{k+1})$

PROOF: $\langle 5\rangle 1$.

$\langle 5\rangle 5$. $\langle m, x_1\rangle$ contains all variables free in $\mathcal{N}_1$.

PROOF: Assumption $\langle 0\rangle$:2 and the definition of M_1.

$\langle 5\rangle 6.$ $\langle \eta_n, \eta_{n+1}\rangle$ is an $\mathcal{N}_1$ step

PROOF: $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$, and $\langle 5\rangle 5$.

$\langle 5\rangle 7.$ $\langle \eta_n, \eta_{n+1}\rangle$ is an $x_2' = x_2$ step.

PROOF: $\langle 5\rangle 1$ and $\langle 0\rangle$:1(a), which implies that x_1 and x_2 have no variable in common.

$\langle 5\rangle 8.$ Q.E.D.

PROOF: $\langle 5\rangle 6$ and $\langle 5\rangle 7$, since $\models \mathcal{N}_1 \wedge (x_2' = x_2) \Rightarrow \mathcal{N}_H$.

$\langle 4\rangle 3.$ CASE: $n = 2k$, $[\![x_1]\!](\tau_k) \neq [\![x_1]\!](\tau_{k+1})$, and
$[\![x_2]\!](\tau_k) \neq [\![x_2]\!](\tau_{k+1})$.
(In this case, $\langle \eta_n, \eta_{n+1}\rangle$ is a step that leaves x_1 unchanged.)

$\langle 5\rangle 1.$ $\eta_{n+1} = \tau_{k+1}$, $[\![x_1]\!](\eta_n) = [\![x_1]\!](\tau_{k+1})$, and $\eta_n =_{x_1} \tau_k$.

PROOF: The definition of η and case assumption $\langle 4\rangle$.

$\langle 5\rangle 2.$ $\langle \tau_k, \tau_{k+1}\rangle$ is an $\mathcal{N}_2$ step.

PROOF: $\langle 3\rangle 2$ and case assumption $\langle 4\rangle$.

$\langle 5\rangle 3.$ $[\![\langle m, x_2\rangle]\!](\eta_n) = [\![\langle m, x_2\rangle]\!](\tau_k)$ and
$[\![\langle m, x_2\rangle]\!](\eta_{n+1}) = [\![\langle m, x_2\rangle]\!](\tau_{k+1})$.

PROOF: $\langle 5\rangle 1$ and assumption $\langle 0\rangle$:1(a), which implies that x_1 has no variable in common with x_2 or m.

$\langle 5\rangle 4.$ $\langle m, x_2\rangle$ contains all variables free in $\mathcal{N}_2$.

PROOF: Assumption $\langle 0\rangle$:2 and the definition of M_2.

$\langle 5\rangle 5.$ $\langle \eta_n, \eta_{n+1}\rangle$ is an $\mathcal{N}_2$ step

PROOF:$\langle 5\rangle 2$, $\langle 5\rangle 3$, and $\langle 5\rangle 4$.

$\langle 5\rangle 6.$ $\langle \eta_n, \eta_{n+1}\rangle$ is an $x_1' = x_1$ step.

PROOF: $\langle 5\rangle 1$.

$\langle 5\rangle 7.$ Q.E.D.

PROOF:$\langle 5\rangle 5$ and $\langle 5\rangle 6$, since $\models \mathcal{N}_2 \wedge (x_1' = x_1) \Rightarrow \mathcal{N}_H$.

$\langle 4\rangle 4.$ Q.E.D.

PROOF: $\langle 4\rangle 1$, $\langle 4\rangle 2$, and $\langle 4\rangle 3$.

$\langle 3\rangle 4.$ $\eta \models \mathit{Init}_1 \wedge \mathit{Init}_2$

$\langle 4\rangle 1.$ $\tau \models \mathit{Init}_1 \wedge \mathit{Init}_2$

PROOF: The definition of U and $\langle 3\rangle 1$ (which asserts $\tau \models U$).

$\langle 4\rangle 2.$ $\eta_1 = \tau_1$

PROOF: The definition of η.

$\langle 4\rangle 3.$ Q.E.D.

PROOF: $\langle 4\rangle 1$ and $\langle 4\rangle 2$, since $[\![P]\!](\rho) = [\![P]\!](\rho_1)$ for any predicate P and behavior ρ.

$\langle 3\rangle 5.$ $\eta \models L_1 \wedge L_2$

$\langle 4\rangle 1.$ $\eta \models L_1$

$\langle 5\rangle 1.$ $\tau \models L_1$

PROOF: $\langle 3\rangle 1$ and the definition of U.

$\langle 5\rangle 2.$ For all $n > 0$:

1. $\eta_{2n-1} = \tau_n$
2. $[\![\langle m, x_1\rangle]\!](\eta_{2n}) = [\![\langle m, x_1\rangle]\!](\tau_n)$ or
$[\![\langle m, x_1\rangle]\!](\eta_{2n}) = [\![\langle m, x_1\rangle]\!](\tau_{n+1})$

PROOF: Part 1 follows from the definition of η. Part 2 follows from the definition of η and $\langle 3\rangle 2$, which implies $[\![m]\!](\tau_n) = [\![m]\!](\tau_{n+1})$ when the **if** condition in the definition is false.

$\langle 5\rangle 3$. Choose $\widehat{\eta}$ such that $\widehat{\eta}_{2n-1} = \tau_n$ and

$$\widehat{\eta}_{2n} \;=\; \textbf{if}\;\; [\![\langle m,\, x_1\rangle]\!](\eta_{2n}) = [\![\langle m,\, x_1\rangle]\!](\tau_n) \;\;\textbf{then}\;\; \tau_n \;\;\textbf{else}\;\; \tau_{n+1}$$

for all n. Then $\widehat{\eta}$ is stuttering equivalent to τ, and, for all n, $[\![\langle m,\, x_1\rangle]\!](\widehat{\eta}_n) = [\![\langle m,\, x_1\rangle]\!](\eta_n)$.

PROOF: $\langle 5\rangle 2$.

$\langle 5\rangle 4$. $\widehat{\eta} \models L_1$

PROOF: $\langle 5\rangle 1$ and $\langle 5\rangle 3$, since TLA formulas are invariant under stuttering.

$\langle 5\rangle 5$. $[\![L_1]\!](\widehat{\eta}) = [\![L_1]\!](\eta)$

PROOF: $\langle 5\rangle 3$, since $\langle m,\, x_1\rangle$ contains all variables occurring free in L_1, by assumption $\langle 0\rangle$:2 and the definition of M_1.

$\langle 5\rangle 6$. Q.E.D.

PROOF: $\langle 5\rangle 4$ and $\langle 5\rangle 5$.

$\langle 4\rangle 2$. $\eta \models L_2$

$\langle 5\rangle 1$. $\tau \models L_2$

PROOF: $\langle 3\rangle 1$ and the definition of U.

$\langle 5\rangle 2$. $\eta \simeq_{x_1} \tau$

PROOF: The definition of η.

$\langle 5\rangle 3$. Q.E.D.

PROOF: $\langle 5\rangle 1$, $\langle 5\rangle 2$, and assumption $\langle 0\rangle$:1(a), which implies that x_1 does not occur free in L_2.

$\langle 4\rangle 3$. Q.E.D.

$\langle 3\rangle 6$. $\eta \models H$

PROOF: $\langle 3\rangle 3$, $\langle 3\rangle 4$, and $\langle 3\rangle 5$, and the definition of H.

$\langle 3\rangle 7$. $\eta \simeq_{\langle x_1,\, x_2\rangle} \sigma$

PROOF: $\langle 3\rangle 1$, which asserts $\tau \simeq_{\langle x_1,\, x_2\rangle} \sigma$, and the definition of η, which implies $\eta \simeq_{x_1} \tau$.

$\langle 3\rangle 8$. Q.E.D.

PROOF: $\langle 3\rangle 6$, $\langle 3\rangle 7$, and the definition of H.

$\langle 2\rangle 4$. Q.E.D.

$\langle 1\rangle 3$. CASE: $n > 2$, and the theorem holds with p substituted for n, for all $p < n$.

LET:
$$\begin{array}{lcl} mm & \triangleq & \langle m_1, \ldots, m_{n-1}\rangle \\ xx & \triangleq & \langle x_1, \ldots, x_{n-1}\rangle \\ \widehat{xx}_i & \triangleq & \langle x_1, \ldots\, x_{i-1},\, , x_{i+1},\, \ldots, x_{n-1}\rangle \end{array}$$

PROOF:

$$\begin{array}{rcl} \bigwedge_{i\leq n} M_i & = & (\bigwedge_{i\leq n-1} M_i) \wedge M_n \\ & & \quad \text{by propositional logic} \\ & = & \wedge\; \exists\, xx : \wedge \bigwedge_{i\leq n-1} \mathit{Init}_i \\ & & \qquad\qquad \wedge\; \Box\,[\bigvee_{i\leq n-1} \mathcal{N}_i \wedge (\widehat{xx}'_i = \widehat{xx}_i)]_{\langle mm,\, xx\rangle} \\ & & \qquad\qquad \wedge \bigwedge_{n-1} L_i \\ & & \wedge\; M_n \\ & & \quad \text{by case assumption } \langle 1\rangle\text{, with } n-1 \text{ substituted for } p \end{array}$$

$$= \exists xx, x_n : \wedge \bigwedge Init_i \\ \wedge \Box \begin{bmatrix} \vee \wedge \bigvee_{i \le n-1} \mathcal{N}_i \wedge (\widehat{xx}_i' = \widehat{xx}_i) \\ \quad \wedge (x_n' = x_n) \\ \vee \mathcal{N}_n \wedge (\widehat{x}_n' = \widehat{x}_n) \end{bmatrix}_{\langle mm, xx, m_n, x_n \rangle} \\ \wedge \bigwedge L_i$$

by case assumption $\langle 1 \rangle$, with 2 substituted for p

$$= \exists\, x : \bigwedge_{i=1}^{n} Init_i \;\wedge\; \Box[\bigvee_{i=1}^{n} \mathcal{N}_i \wedge (\widehat{x}_i' = \widehat{x}_i)]_{\langle m, x \rangle} \;\wedge\; \bigwedge_{i=1}^{n} L_i$$

$\langle 1 \rangle 4$. Q.E.D.

Proof: $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, and mathematical induction.

B6 Decomposition and Composition

Theorem 1 is an immediate consequence of Theorem 2. The proof of Theorem 2 assumes Theorem 3, but Theorem 2 is not used in the proof of Theorem 3 or of any lemma, so there is no circularity.

Theorem 2.

Assume: *For $i = 1, \ldots, n$:*

1. $\models \mathcal{C}(E) \wedge \bigwedge_{j=1}^{n} \mathcal{C}(M_j) \;\Rightarrow\; E_i$
2. *a.* $\models \mathcal{C}(E_i)_{+v} \wedge \mathcal{C}(M_i^l) \;\Rightarrow\; \mathcal{C}(M_i)$
   b. $\models E_i \wedge M_i^l \;\Rightarrow\; M_i$
3. *v is a tuple of variables including all the free variables of M_i.*

Prove: *a.* $\models \mathcal{C}(E)_{+v} \wedge \bigwedge_{j=1}^{n} \mathcal{C}(M_j^l) \;\Rightarrow\; \bigwedge_{j=1}^{n} \mathcal{C}(M_j)$

b. $\models E \wedge \bigwedge_{j=1}^{n} M_j^l \;\Rightarrow\; \bigwedge_{j=1}^{n} M_j$

$\langle 1 \rangle 1$. For any E, E_i, M_i^l, and M_i satisfying assumptions $\langle 0 \rangle$:1–3, and all $i = 1, \ldots, n$: $\models (\bigwedge_{j=1}^{n} M_j^l) \Rightarrow (E \stackrel{+}{\rhd} M_i)$

$\langle 2 \rangle 1$. For $j = 1, \ldots, n$: $\models M_j^l \Rightarrow (E_i \stackrel{+}{\rhd} M_j)$

$\langle 3 \rangle 1$. For $i = 1, \ldots, n$: $\models \mathcal{C}(M_i^l) \Rightarrow (\mathcal{C}(E_i) \stackrel{+}{\rhd} \mathcal{C}(M_i))$

Proof: Assumption $\langle 0 \rangle$:2(a), Lemma 1(2), assumption $\langle 0 \rangle$:3, and Lemma 11.

$\langle 3 \rangle 2$. For $i = 1, \ldots, n$: $\models M_i^l \Rightarrow (E_i \Rightarrow M_i)$

Proof: Assumption $\langle 0 \rangle$:2(b).

$\langle 3 \rangle 3$. Q.E.D.

Proof: $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, and Lemma 2(2).

$\langle 2 \rangle 2$. For $i = 1, \ldots, n$: $\models (\bigwedge_{j=1}^{n} (E_j \stackrel{+}{\rhd} M_j)) \Rightarrow (E \stackrel{+}{\rhd} M_i)$

Proof: The Composition Theorem (Theorem 3), with M_i substituted for M, where hypothesis 1 of the Composition Theorem follows from assumption $\langle 0 \rangle$:1, and hypotheses 2(a) and 2(b) are vacuous when M_i is substituted for M.

$\langle 2 \rangle 3$. Q.E.D.

Proof: $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, and propositional reasoning.

$\langle 1 \rangle 2$. Conclusion (a) holds.

⟨2⟩1. For $i = 1, \ldots, n$: $\models (\bigwedge_{j=1}^{n} \mathcal{C}(M_j^l)) \Rightarrow (\mathcal{C}(E) \stackrel{+}{\rhd} \mathcal{C}(M_i))$

PROOF: ⟨1⟩1, substituting $\mathcal{C}(E)$ for E, $\mathcal{C}(M_j^l)$ for M_j^l, and $\mathcal{C}(M_i)$ for M_i. Since $\mathcal{C}$ is idempotent, this instantiation changes only assumption 2(b), which becomes $\models \mathcal{C}(E_i) \wedge \mathcal{C}(M_i^l) \Rightarrow \mathcal{C}(M_i)$. This assumption follows from 2(a), since $\models P \Rightarrow P_{+v}$, for any P.

⟨2⟩2. For $i = 1, \ldots, n$: $\models (\bigwedge_{j=1}^{n} \mathcal{C}(M_j^l)) \Rightarrow (\mathcal{C}(E)_{+v} \multimap\!\rhd \mathcal{C}(M_i))$

PROOF: ⟨2⟩1, assumption ⟨0⟩:3, and Lemma 11.

⟨2⟩3. Q.E.D.

PROOF: ⟨2⟩2 and Lemma 1(2) (conjoining over all i).

⟨1⟩3. Conclusion (b) holds.

⟨2⟩1. $\models E \wedge (\bigwedge_{j=1}^{n} M_j^l) \Rightarrow M_i$

PROOF: ⟨1⟩1 and Lemma 2(2).

⟨2⟩2. Q.E.D.

PROOF: ⟨2⟩1 (conjoining over all i).

⟨1⟩4. Q.E.D.

LEMMA 18.

ASSUME: *For $i = 1, \ldots, n$:*

0. *M_i is a safety property.*
1. *E and E_i are safety properties.*
2. $\models (E \wedge \bigwedge_{j=1}^{n} M_j) \;\Rightarrow\; E_i$

PROVE: $\models (\bigwedge_{j=1}^{n} (E_j \stackrel{+}{\rhd} M_j)) \;\Rightarrow\; (E \stackrel{+}{\rhd} (\bigwedge_{j=1}^{n} M_j))$

PROOF: $\bigwedge_{j=1}^{n} (E_j \stackrel{+}{\rhd} M_j))$

$\Rightarrow\; (\bigwedge_{j=1}^{n} E_j) \stackrel{+}{\rhd} (\bigwedge_{j=1}^{n} M_j)$

Lemma 5 and assumptions ⟨0⟩:0 and ⟨0⟩:1

$\Rightarrow\; E \stackrel{+}{\rhd} (\bigwedge_{j=1}^{n} M_j)$

assumption ⟨0⟩:2 and Lemma 7, substituting E for R, $\bigwedge_{j=1}^{n} E_j$ for P, and $\bigwedge_{j=1}^{n} M_j$ for Q.

THEOREM 3.

ASSUME: *For $i = 1, \ldots, n$:*

1. $\models \mathcal{C}(E) \wedge \bigwedge_{j=1}^{n} \mathcal{C}(M_j) \;\Rightarrow\; E_i$
2. *a.* $\models \mathcal{C}(E)_{+v} \wedge \bigwedge_{j=1}^{n} \mathcal{C}(M_j) \;\Rightarrow\; \mathcal{C}(M)$

   b. $\models E \wedge \bigwedge_{j=1}^{n} M_j \;\Rightarrow\; M$

PROVE: $\models \bigwedge_{j=1}^{n} (E_j \stackrel{+}{\rhd} M_j) \;\Rightarrow\; (E \stackrel{+}{\rhd} M)$

$\langle 1 \rangle 1.$ $\models (\bigwedge_{j=1}^{n}(\mathcal{C}(E_j) \stackrel{+}{\rhd} \mathcal{C}(M_j))) \;\Rightarrow\; (\mathcal{C}(E) \stackrel{+}{\rhd} (\bigwedge_{j=1}^{n} \mathcal{C}(M_j)))$

PROOF: Assumption $\langle 0 \rangle$:1 and Lemma 18, since $\models E_i \;\Rightarrow\; \mathcal{C}(E_i)$ (because $\mathcal{C}$ is superdiagonal).

$\langle 1 \rangle 2.$ $\models (\mathcal{C}(E) \stackrel{+}{\rhd} (\bigwedge_{j=1}^{n} \mathcal{C}(M_j))) \;\Rightarrow\; (\mathcal{C}(E) \stackrel{+}{\rhd} \mathcal{C}(M))$

PROOF: Assumption $\langle 0 \rangle$:2(a) and Lemma 12.

$\langle 1 \rangle 3.$ $\models (\bigwedge_{j=1}^{n}(E_j \stackrel{+}{\rhd} M_j)) \;\Rightarrow\; (E \Rightarrow M)$

$\langle 2 \rangle 1.$ $\models \mathcal{C}(E) \wedge (\bigwedge_{j=1}^{n}(\mathcal{C}(E_j) \stackrel{+}{\rhd} \mathcal{C}(M_j))) \;\Rightarrow\; \bigwedge_{j=1}^{n} \mathcal{C}(M_j)$

PROOF: $\langle 1 \rangle 1$ and Lemma 3(2).

$\langle 2 \rangle 2.$ $\models E \wedge (\bigwedge_{j=1}^{n}(E_j \stackrel{+}{\rhd} M_j)) \;\Rightarrow\; \bigwedge_{j=1}^{n} \mathcal{C}(M_j)$

PROOF: $\langle 2 \rangle 1$, since $\models E \Rightarrow \mathcal{C}(E)$ (because $\mathcal{C}$ is superdiagonal) and $\models (E_j \stackrel{+}{\rhd} M_j) \Rightarrow (\mathcal{C}(E_j) \stackrel{+}{\rhd} \mathcal{C}(M_j))$ by Lemma 2.

$\langle 2 \rangle 3.$ $\models E \wedge (\bigwedge_{j=1}^{n}(E_j \stackrel{+}{\rhd} M_j)) \;\Rightarrow\; \bigwedge_{j=1}^{n} E_j$

PROOF: $\langle 2 \rangle 2$ and assumption $\langle 0 \rangle$:1, since $\mathcal{C}$ is superdiagonal.

$\langle 2 \rangle 4.$ $\models E \wedge (\bigwedge_{j=1}^{n}(E_j \stackrel{+}{\rhd} M_j)) \;\Rightarrow\; \bigwedge_{j=1}^{n} M_j$

PROOF: $\langle 2 \rangle 3$ and Lemma 3(2).

$\langle 2 \rangle 5.$ $\models E \wedge (\bigwedge_{j=1}^{n}(E_j \stackrel{+}{\rhd} M_j)) \;\Rightarrow\; M$

PROOF: $\langle 2 \rangle 4$ and assumption $\langle 0 \rangle$:2(b).

$\langle 2 \rangle 6.$ Q.E.D.

PROOF: $\langle 2 \rangle 5$.

$\langle 1 \rangle 4.$ Q.E.D.

PROOF: $\langle 1 \rangle 1$ and $\langle 1 \rangle 2$, which imply

$$\models \bigwedge_{j=1}^{n} (\mathcal{C}(E_j) \stackrel{+}{\rhd} \mathcal{C}(M_j)) \;\Rightarrow\; (\mathcal{C}(E) \stackrel{+}{\rhd} \mathcal{C}(M))$$

$\langle 1 \rangle 3$, and Lemma 2(2).

Index