Refinement and Projection of Relational Specifications*

Simon S. Lam

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

A. Udaya Shankar

Department of Computer Science and
Institute for Advanced Computer Studies,
University of Maryland,
College Park, Maryland 20742

Abstract. A relational specification consists of a state transition system and a set of fairness assumptions. The state transition system is specified using two basic constructs: *state formulas* that respresent sets of states, and *event formulas* that represent sets of state transitions. We present a theory of *refinement* of relational specifications. Several refinement relations between specifications are defined. To illustrate our concepts and methods, three specifications of the alternating-bit protocol are given. We also apply the theory to explain "auxiliary variables." Other applications of the theory to protocol verification, composition, and conversion are discussed. Our approach is compared with the approaches of other authors.

Key words: Specification, refinement, protocols, distributed systems, temporal logic.

CONTENTS

* The work of Simon S. Lam was supported by National Science Foundation under grant no. NCR-8613338 and by a grant from the Texas Advanced Research Program. The work of A. Udaya Shankar was supported by National Science Foundation under grant no. ECS-8502113 and grant no. NCR-8904590. This paper is an abbreviated version of [Lam & Shankar 88].

1. Introduction

The concepts of *state* and *state transition* are fundamental to many formalisms for the specification and analysis of systems. For example, state-transition models are widely used for the specification of communication protocols in practice [IBM 80, ISO 85, Sabnani 88, West 78]. Most engineers and programmers are familiar with the concepts of state and state tranition and, in fact, many prefer working with these concepts [Piatkowski 86].

During the course of our research on modeling communication network protocols and time-dependent distributed systems, a formalism for specifying them as state transition systems has evolved [Shankar & Lam 84, 87a, 87b, Lam & Shankar 87]. Our formalism has been developed with the hope that protocol engineers will find our notation and the accompanying proof method to be easy to learn because states and state transitions are represented explicitly. But instead of individual states and state transitions, we work with *sets* of states and state transitions; these sets are specified using the language of predicate logic.

Our proof method is based upon a fragment of linear-time temporal logic [Chandy & Misra 88, Lamport 83a, Manna & Pnueli 84, Owicki & Lamport 82, Pnueli 86]. Motivated by examples in communication network protocols, we introduce several extensions to the body of work cited above. First, we introduce a "leads-to" proof rule designed for message-passing networks with unreliable channels. Second, we advocate the approach of stating fairness assumptions explicitly for individual events as part of a specification, noting that for many specifications only *some* events need be fairly scheduled. In general, to facilitate implementation, a specification includes fairness assumptions that are as weak as possible. While this idea is not new (see [Pnueli 86]), our definition of the *allowed behaviors* of a specification differs from Pnueli's definition of *fair computations* in that an allowed behavior may not be "maximal." Our definition is motivated by the specification of program modules to satisfy interfaces [Lynch & Tuttle 87, Lam & Shankar 87].

A *relational specification* consists of a state transition system given in the relational notation together with a set of fairness assumptions. We present a theory of refinement of relational specifications. Let A and B denote specifications. Several relations between A and B are defined. Other authors have defined similar relations between specifications: *A implements B*, *A simulates B*, etc. Every one of these relations is a form of the *subset* relation with the following meaning: every externally visible behavior allowed by A is also allowed by B [Lam & Shankar 84, Lamport 85, Lynch & Tuttle 87]. (There are differences in how behaviors and visible behaviors are defined.) Our definition of *A is a well-formed refinement of B* in this paper is essentially the same.

For many applications, we found the subset relation, as informally defined above, to be too strong to be useful. In refining a specification B to a specification A, it is seldom the case that every progress property of B must be preserved in A. In most cases, only *some* specific progress properties of B are to be preserved. In this paper, we provide a variety of relations between specifications. The *well-formed refinement* relation is the strongest. In our experience, it is seldom used. The *refinement* relation is the weakest. In our experience, it is always used. In [Shankar & Lam 87b], a weaker form of the refinement relation, called *conditional refinement*, is also defined for use in a stepwise refinement heuristic.

In refining a specification B to a specification A, if some state variables in B are replaced by new state variables in A, our approach is to find an "invariant" that specifies the relation between the new and old state variables. The approach of other authors is to find a special mapping from the states of A to the states of B [Abadi & Lamport 88, Lynch & Tuttle 87]. The relation to be found is the *same* in each approach. The approaches differ mainly in how such a relation is represented. (Instead of a special mapping, we use an invariant, auxiliary variables and a projection mapping.) To illustrate our method and to compare it with those of other authors, we present, in Section 8 below, three specifications of the alternating-bit protocol.

Since the specification and proof of communication networks with unreliable channels is an important application domain of ours, we have extended the theory of refinement to include the refinement of messages. Furthermore, for messages represented by a tuple of message fields, the concept of auxiliary variables is extended to include the use of *auxiliary fields* in messages.

2. Relational Notation

We consider state transition systems that are defined by a pair (S,T), where S is a countable set of states and T is a binary relation on S. T is a set of state transitions, each of which is represented by an ordered pair of states. Given (S,T) and an initial condition on the system state, a sequence of states $<s_0, s_1, \cdots>$ is said to be a *path* if s_0 satisfies the initial condition and, for $i \geq 0$, (s_i, s_{i+1}) is in T.

In the relational notation, a state transition system is specified by a set of state variables, $\mathbf{v} = \{v_1, v_2, \cdots\}$, a set of events, $e_1, e_2, \cdots$, and an initial condition, to be defined below. For every state variable, there is a specified domain of allowed values. The system state is represented by the set of values assumed by the state variables. The state space S of the system is the cartesian product of the state variable domains. The binary relation T is defined by the set of events (see below).

Parameters may be used for defining groups of related events, as well as groups of related system properties. Let $\mathbf{w}$ denote a set of parameters, each with a specified domain of allowed values.

Let $\mathbf{v}'$ denote the set of variables $\{v' : v \in \mathbf{v}\}$. In specifying an event, we use $\mathbf{v}$ and $\mathbf{v}'$ to denote, respectively, the system state before and after an event occurrence. Instead of a programming language, the language of predicate logic is used for specifying events. Such a language consists of a set of symbols for variables, constants, functions and predicates, and a set of formulas defined over the symbols [Manna & Waldinger 1985]. We assume that there is a known interpretation that assigns meanings to all of the function symbols and predicate symbols, and values to all of the constant symbols that we use. As a result, the truth value of a formula can be determined if values are assigned to its free variables.

The set of variables in our language is $\mathbf{v} \cup \mathbf{v}' \cup \mathbf{w}$. We will use two kinds of formulas: A formula whose free variables are in $\mathbf{v} \cup \mathbf{w}$ is called a *state formula*. A formula whose free variables are in $\mathbf{v} \cup \mathbf{v}' \cup \mathbf{w}$ is called an *event formula*.

A state formula can be evaluated to be true or false for each system state by adopting this convention: if a parameter occurs free in a state formula, it is assumed to be universally quantified over the parameter domain.

We say that a system state s satisfies a state formula P if and only if (iff) P evaluates to true for s. A state formula P represents the *set* of system states that satisfy P. In particular, the initial condition of the system is specified by a state formula. A system state that satisfies the initial condition is called an *initial state*. In the following, the letters P, Q, R and I are used to denote state formulas.

Events are specified by event formulas. Each event (formula) defines a *set* of system state transitions. (These sets may overlap.) The union of these sets over all events defines the binary relation T of the transition system. Some examples of event definitions are shown below:

$$e_1 \equiv v_1>2 \wedge v_2' \in \{1, 2, 5\}$$

$$e_2 \equiv v_1>v_2 \wedge v_1+v_2'=5$$

where " $\equiv$ " denotes "is defined by." In each definition, the event name is given on the left-hand side and the event formula is given on the right-hand side. For convenience, we sometimes use the same symbol to denote the name of an event as well as the event formula that defines it. The context where the symbol appears will determine what it means.

Convention. Given an event formula e, for every state variable v in $\mathbf{v}$, if v' is not a free variable of e then each occurrence of the event e does not change the value of v; that is, the conjunct $v'=v$ is implicit in the event formula.

For example, consider a system with two state variables v_1 and v_2. Let e_2 above be an event of the system. Note that v_1' is not a free variable of e_2. By the above convention, the event formula that defines e_2 is in fact $v_1>v_2 \wedge v_1+v_2'=5 \wedge v_1'=v_1$.

If a parameter occurs free in an event definition, then the system has an event defined for each value in the domain of the parameter. For example, consider

$$e_3(m) \equiv v_1>v_2 \wedge v_1+v_2'=m$$

where m is a parameter. A parameterized event is a convenient way to specify a group of related events.

We next introduce the notion of the enabling condition of an event. An event (formula) defines a set of ordered pairs (s, s'), where $s \in S$ and $s' \in S$. Let the ordered pairs shown in Figure 1 be those defined by some event. The *enabling condition* of this event is defined by the three states shown inside the shaded area of Figure 1. (A formal definition is given below.)

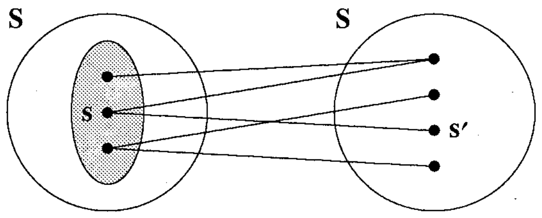

Figure 1. An illustration of an event formula and its enabling condition.

An event can occur only when the system state satisfies the enabling condition of the event. In any system state, more than one event may be enabled. The choice of the next event to occur from the set of enabled events is nondeterministic.[1] When an event occurs, we assume that the state variables of the system are updated in one *atomic* step.

Formally, the enabling condition of an event formula e, to be denoted by $enabled(e)$, is given by

$$enabled(e) \equiv \exists \mathbf{v}'[e]$$

which is a state formula. Consider the following event formula as an example:

$$e_4 \equiv v_1 > v_2 \wedge v_1' = 1 \wedge v_2' = 0$$

Suppose the domain of each state variable is the set of natural numbers. We have

$$enabled(e_4) \equiv \exists v_1' \exists v_2' [e_4]$$

which is $v_1 > v_2$, because the expression $\exists v_1' \exists v_2' [v_1' = 1 \wedge v_2' = 0]$ is true.

For readability, we write many event formulas in the following *separable* form:

$$e \equiv guard \wedge action$$

where *guard* is a state formula and *action* is an event formula. We must keep in mind that for *guard* to be logically equivalent to $enabled(e)$, the two conjuncts in the separable form must satisfy the condition:

$$guard \Rightarrow \exists \mathbf{v}'[action]$$

Otherwise, part of the enabling condition of e is specified by *action*.

In summary, the relational notation has two basic constructs: state formulas and event formulas. A state formula defines a set of system states. An event formula defines a set of system state transitions.

As an example, consider the following model of an object moving in two-dimensional space. Imagine that the object is an airplane flying from Austin to Dallas. There are two state variables: y is the horizontal distance from Austin along a straight line between the two cities, and z is the altitude of the airplane. The domain of y is $\{0, 1, \cdots, N\}$ such that $y=0$ indicates that the airplane is at Austin airport and $y=N$ indicates that the airplane is at Dallas airport. The domain of z is the set of natural numbers such that $z=0$ indicates that the airplane is on the ground. The initial condition is $y=0 \wedge z=0$, indicating that the airplane is on the ground at Austin airport. The events are defined as follows:

TakeOff	$\equiv$	$y=0 \wedge z=0 \wedge y'=1 \wedge 10 \le z' \le 20$
Landing	$\equiv$	$y=N-1 \wedge 10 \le z \le 20 \wedge y'=N \wedge z'=0$
Fly	$\equiv$	$1 \le y \le N-2 \wedge 10 \le z \le 20 \wedge y'=y+1$
FlyHigher	$\equiv$	$1 \le y \le N-1 \wedge 10 \le z < 20 \wedge z'=z+1$
FlyLower	$\equiv$	$1 \le y \le N-1 \wedge 10 < z \le 20 \wedge z'=z-1$

3. System Specification

A system can be specified in many ways, in many notations. In this paper, we consider two related approaches. In the first approach, a system is specified by defining a state space, and giving a set of requirements each of which is an assertion of a desired system property. Two classes of system properties are of interest to us in this paper, namely, safety properties and progress properties. In particular,

[1]The choice is not strictly nondeterministic if the system specification includes fairness assumptions for some events (see Section 3 below).

we will use safety assertions of the form,

P is invariant

and progress assertions of the form,[2]

P *leads-to* Q

where P and Q are state formulas for the state space defined.

In the second approach, a specification consists of a state transition system given in the relational notation.[3] The two approaches are related in the following sense: A specification consisting of a state transition system *implements* a specification consisting of a set of requirements iff all of the requirements are properties of the state transition system.

We use the airplane example in Section 2 to illustrate the two classes of system properties. A safety requirement of the airplane example may be stated as follows: The airplane is in a specified portion of the air space if it is not at one of the airports. This is formalized by the assertion

$y \neq 0 \wedge y \neq N \Rightarrow 10 \leq z \leq 20$ is invariant

Safety properties of a state transition system are determined by its finite paths. The following definitions apply to state transition systems specified in any notation.

Definition. A system state s is *reachable* iff there is a finite path from an initial state to s.

Definition. P is invariant for a state transition system iff every reachable state of the system satisfies P.

If a parameter occurs free in the state formula P, then P is invariant for a state transition system iff, for every allowed value of the parameter, P is invariant for the system.

A progress requirement of the airplane example may be stated as follows: The airplane, initially at Austin, eventually arrives at Dallas. This is formalized by the assertion

$y=0 \wedge z=0$ *leads-to* $y=N \wedge z=0$

Let us first define the meaning of P *leads-to* Q for a sequence of states $\sigma = \langle s_0, s_1, \cdots \rangle$. The sequence may be finite or infinite.

Definition. P *leads-to* Q for σ iff the following holds: if some state s_i in σ satisfies P then there is a state s_j in σ ($j \geq i$) that satisfies Q.

Before defining 'P *leads-to* Q for a specification,' we next introduce the concept of fairness and two fairness criteria.

Note that a state transition system specified in the relational notation has the additional concept of named events. In any system state, several events may be enabled. Strict nondeterminism in choosing the event to occur next allows the possibility that some events never occur even though they are enabled continuously (or infinitely often). In the airplane example, for instance, there are many infinite paths that correspond to system executions in which the event *Fly* is continuously enabled but never occurs.

[2]In writing assertions containing *leads-to*, we adopt the convention that its binding power is weaker than any of the logical connectives in state formulas.

[3] and some fairness assumptions. The meaning of fairness is introduced below and can be ignored for now. In general, a specification may consist of a state transition system together with a set of of safety requirements and progress requirements. Such a general approach subsumes both approaches herein. See [Lam & Shankar 87, Shankar & Lam 87b].

Such unfair behavior is undesirable and should be disallowed by the system specification.

One way to disallow certain unfair behaviors is to refine the state transition system specification to include an event scheduler. In the airplane example, for instance, we can keep a count of the number of times *FlyHigher* and *FlyLower* have occurred since the last occurrence of *Fly* or *TakeOff*; also, *FlyHigher* and *FlyLower* are disabled whenever the count exceeds some threshold value. Such a specification requires an additional state variable and various modifications of event actions. The specification, moreover, leaves little flexibility to system implementors who might have a different, yet better, solution to the event scheduling problem.

Instead of specifying event scheduling explicitly, fairness assumptions can be included as part of a system specification. Different criteria of fairness abound in the literature. The one that we use most often is *weak fairness*, also called *justice* [Manna & Pnueli 1984]. Informally, the meaning of an event having weak fairness is the following: if the event is continuously enabled in a system execution, it eventually occurs. (A more precise definition is given below. The *strong fairness* criterion is also defined below and used in subsequent sections.) For example, the airplane specification satisfies the progress requirement stated above, if the events *TakeOff*, *Fly* and *Landing* are scheduled in such a way that each event has *weak fairness*; the other events in the example do not have to be fairly scheduled.

Generally, to satisfy a given set of progress requirements, only some of the events in a specification need to be fairly scheduled. To facilitate implementation, a specification should include fairness assumptions that are as weak as possible. Very often, fairness assumptions can be weakened by defining a new event to be the disjunction of a set of events already defined, e.g.,

$$e_3 \equiv \exists m\, [e_3(m)]$$

$$e_5 \equiv e_1 \vee e_2$$

and showing that only the new event needs to be fairly scheduled, and not the individual events in the set.

For events defined by a parameterized formula, such as $e_3(m)$, we have to be especially careful. Suppose the domain of m is infinite. In this case, fair scheduling of $e_3(m)$ for every allowed value of m may or may not be a physically meaningful assumption.

At this point, we note that the set of paths is not adequate for defining fairness criteria for events. Specifically, each state transition in a path, say (s_i, s_j), may be due to the occurrence of any of several events. Therefore, the path may correspond to many possible system executions.

To define fairness criteria for events, we represent a system execution by a path in which each transition is labeled by the name of the event whose occurrence caused the transition. We refer to such a labeled path as a *behavior*.

Consider an event e and an infinite behavior β, we define two fairness criteria:

- Event e has *weak fairness* for infinite behavior β iff e either occurs infinitely often or is disabled infinitely often in β. [4]

[4] The following definition is equivalent: Event e has weak fairness for infinite behavior β iff for some state s_k in β and for all $i \geq k$, if state s_i satisfies *enabled(e)* then there is some state s_j in β $(j \geq k)$ such that the transition from s_j to s_{j+1} is labeled e.

- Event e has *strong fairness* for infinite behavior β iff the following holds: e occurs infinitely often in β if it is enabled infinitely often in β.

It is easy to see that if β satisfies the strong fairness criterion for event e, then it also satisfies the weak fairness criterion for event e. (The converse does not hold.) The weak and strong fairness criteria are the only ones used in this paper. Thus, when we say that a specification includes a fairness assumption for event e, we mean that event e has either weak fairness or strong fairness.

For a given specification consisting of a state transition system and a set of fairness assumptions, the *allowed behaviors* of the specification are defined as follows: [5]

- A finite behavior of the state transition system is an allowed behavior of the specification iff every event that has a fairness assumption in the specification is disabled in the last state of the behavior.
- An infinite behavior of the state transition system is an allowed behavior of the specification iff every fairness assumption of the specification holds for the behavior.

Definition. P *leads-to* Q for a behavior β iff P *leads-to* Q for the sequence of states in β.

Definition. P *leads-to* Q for a specification iff P *leads-to* Q for every allowed behavior of the specification.

If a parameter occurs free in P or Q, then P *leads-to* Q for a specification iff, for every allowed value of the parameter, P *leads-to* Q for the specification.

In Section 4 below, we present some inference rules for proving the two kinds of assertions introduced above. Before doing that, we digress and discuss the use of *auxiliary variables* in a specification. That is, some of the state variables in $\mathbf{v}$ may be auxiliary variables, which are needed for specification and verification only, and do not have to be included in an actual implementation (this notion will be made more precise in Section 7). For example, an auxiliary variable may be needed to record the history of certain event occurrences.[6] Informally, a subset of variables in $\mathbf{v}$ can be considered auxiliary if they do not affect the enabling condition of any event nor do they affect the update of any state variable that is not auxiliary [Owicki & Gries 76]. To state the above condition precisely, let $\mathbf{u}$ be a proper subset of $\mathbf{v}$, and $\mathbf{u}' = \{v' : v \in \mathbf{u}\}$. The state variables in $\mathbf{u}$ can be considered auxiliary if, for every event e of the system, the following holds:

$$e \Rightarrow \forall \mathbf{u} \exists \mathbf{u}' [e]$$

To see why auxiliary variables do not have to be included in an actual implementation of a specification, suppose there is an observer and it can only see nonauxiliary variables. The above condition ensures that the set of 'observable behaviors' of the specification is the same whether or not auxiliary variables are part of the system state. We will elaborate on this explanation in Section 7 after the theory of refinement and projection has been presented.

[5] This definition is motivated by the specification of a program module to satisfy its interfaces; specifically, some interface events may be controlled by the environment and not controllable by the module [Lynch & Tuttle 87, Lam & Shankar 87]. For this reason, an allowed behavior is not necessarily *maximal*, as in [Manna & Pnueli 84].

[6] What we call auxiliary variables here are also known as history variables. Abadi and Lamport [1988] defined another kind of auxiliary variables called prophecy variables.

4. Proof Rules

Consider a specification consisting of a state transition system given in the relational notation and a set of fairness assumptions. Let e denote an arbitrary event and *Initial* denote a state formula specifying the initial condition.

Notation. For an arbitrary state formula Q, we use Q' to denote the formula obtained by replacing every free state variable v in Q by v'.

Invariance rules: For a given specification, P is invariant if one of the following holds:

- $\mathit{Initial} \Rightarrow P$ and, for all e, $P \wedge e \Rightarrow P'$
- for some R, R is invariant and $R \Rightarrow P$

In applying the first invariance rule, if I is invariant for the specification, we can replace $P \wedge e \Rightarrow P'$ in the rule with $I \wedge I' \wedge P \wedge e \Rightarrow P'$. Also, we follow this convention: For p and q being formulas with free variables, $p \Rightarrow q$ is logically valid iff $p \Rightarrow q$ is logically valid for all values of the free variables.

For convenience, if P is invariant for a specification, we refer to the formula P as an *invariant property* of the specification or, simply, an invariant.

Definition: For a given specification in which event e_i has weak fairness, P *leads-to* Q *via* e_i iff

(i) $P \wedge e_i \Rightarrow Q'$,

(ii) for all e, $P \wedge e \Rightarrow P' \vee Q'$, and

(iii) $P \Rightarrow \mathit{enabled}(e_i)$ is invariant.

Definition: For a given specification in which event e_i has strong fairness, P *leads-to* Q *via* e_i iff

(i) $P \wedge e_i \Rightarrow Q'$,

(ii) for all e, $P \wedge e \Rightarrow P' \vee Q'$, and

(iii) P *leads-to* $Q \vee \mathit{enabled}(e_i)$.

If I is invariant for the specification, it can be used to strengthen the antecedent of every logical implication in the above definitions; that is, replace P by $I \wedge I' \wedge P$. Also, if the event formula defining e_i has a free parameter, then P *leads-to* Q *via* e_i holds iff each part of the applicable definition holds for every allowed value of the free parameter.

Leads-to rules: For a given specification, P *leads-to* Q if one of the following holds:

- $P \Rightarrow Q$ is invariant [Implication]
- for some event e_i that has fairness, P *leads-to* Q *via* e_i [Event]
- for some R, P *leads-to* R and R *leads-to* Q [Transitivity]
- $P = \exists m \in M[P(m)]$ and $\forall m \in M : P(m)$ *leads-to* Q [Disjunction]

Note that there are actually two Event rules, one for events that have weak fairness and one for events that have strong fairness, to be referred to as the *weak* and *strong* Event rules, respectively. In the Disjunction rule, m denotes a parameter with domain M; also, m does not occur free in Q. A special case of the Disjunction rule is the following: P *leads-to* Q if P_1 *leads-to* Q, P_2 *leads-to* Q, and $P = P_1 \vee P_2$.

What we have presented above is a fragment of linear-time temporal logic.[7] In the next section, we consider message-passing networks with unreliable channels. An additional leads-to rule is presented there. But before doing so, we state two lemmas that are needed in Section 6. (Proofs of the lemmas can be found in [Lam & Shankar 88].)

Lemma 1: For a given specification, P_0 *leads-to* $(Q \vee P_2)$ if

(i) P_0 *leads-to* $(Q \vee P_1)$, and

(ii) P_1 *leads-to* $(Q \vee P_2)$

Lemma 2: For a given specification, if I is invariant and $P \wedge I$ *leads-to* Q, then P *leads-to* Q.

5. Distributed Systems

The relational notation and proof method introduced above are not dependent on whether a distributed or centralized system is being specified. The relevant assumption we have made is that event actions are atomic; consequently, concurrent actions in different modules of a system are modeled by interleaving them in any order.

We next consider distributed systems that are *message-passing* networks. In particular, the network topology is a directed graph whose nodes are called *entities* and whose arcs are called *channels*. For each channel i, there is a state variable that represents the channel state, given by the sequence of messages travelling along the channel. Errors that can occur to messages travelling along a channel are specified by introducing events whose occurrences can update the channel state.[8] The events of a channel can access (read or update) only the channel state variable and auxiliary state variables of the system.

Each entity in a distributed system is specified by a set of state variables and events. Every nonauxiliary state variable in the set is assumed to be local to the entity; that is, it can only be accessed by events of the entity.[9] Entity events can also access auxiliary state variables of the system, as well as state variables representing channels that are connected to the entity.

For clarity in writing specifications, channel state variables are accessed by entities only via send and receive primitives that are defined for the channels. For example, let z_i be a state variable

[7] Essentially a derivative of the work in [Manna & Pnueli 84, Pnueli 86]. A proof that the invariance and leads-to rules are sound is straightforward and is omitted. The reader is also referred to [Chandy & Misra 88] for a comprehensive treatment of proof rules.

[8] If a channel is assumed to be error-free, then no event is introduced for the channel and it behaves like a FIFO queue. If messages travelling along a channel can be duplicated and arbitrarily reordered, it might appear that representing the channel state by a bag of messages is more appropriate. It is easy to see, however, that there is no loss of generality in representing the channel state by a sequence of messages.

[9] Actually, a nonauxiliary state variable of one entity can be read by another entity provided that the value read affects the update of auxiliary variables only.

representing channel i that is a channel with unbounded capacity. Let m denote a message. Define

$$Send_i(m) \equiv \mathbf{z}_i' = \mathbf{z}_i @ m$$

$$Rec_i(m) \equiv \mathbf{z}_i = m @ \mathbf{z}_i'$$

where @ denotes the concatenation operator. Note that $Rec_i(m)$ is *false* if $\mathbf{z}_i$ is empty. Primitives for channels with a finite capacity can be similarly defined. Such a primitive is simply an event formula that has m, $\mathbf{z}_i$ and $\mathbf{z}_i'$ as free variables; thus the names $Send_i(m)$ and $Rec_i(m)$ are introduced primarily to improve the readability of events in a system specification.

For a given message m, an event whose occurrence sends message m along a specified channel is called a *send event* of m. An event whose occurrence receives message m from the channel is called a *receive event* of m. If a message symbol m occurs free in a formula that defines a send event or a receive event for the channel, the domain of m is assumed to be known; for notational brevity, it will not be explicitly shown.

To prove that a message-passing network has useful progress properties, we need two assumptions. First, we assume that the system specification includes "adequate" receive events in the following sense:

Receive events assumption: For message m that is sent along channel i, let $\{e_h(m)\}$ denote the set of receive events of m. For every message and every channel, the set of receive events in the specification satisfies

$$m = Head(\mathbf{z}_i) \Rightarrow \exists h\, [enabled(e_h(m))] \text{ is invariant}$$

for the specification, where $\mathbf{z}_i$ is the channel state variable, *Head* is a function whose value is the first element of $\mathbf{z}_i$ if $\mathbf{z}_i$ is not null; otherwise, *Head* returns a null value (that is not an allowed message value).

For distributed systems with unreliable channels that can lose or reorder messages, a second assumption is needed, namely: the unreliable channels have some minimal progress property. (Otherwise, the channels may be so unreliable that they do not really exist.) Such an assumption should be as weak as possible such that it can be satisfied by most physical communication links. The following is adapted from [Hailpern & Owicki 83]:

Channel progress assumption: If messages in a set M are sent infinitely often along a channel, then they are received infinitely often from the channel.

Informally, if messages in set M are sent repeatedly along a channel, one of them is eventually received. The channel progress assumption can be viewed as a fairness assumption. For a system with unreliable channels, an infinite behavior is an allowed behavior of the specification only if the channel progress assumption holds for the behavior.

Before stating another rule for proving leads-to assertions, we define a new *leads-to-via* relation between state formulas. In the following definition, m denotes a message, e_r denotes a receive event of messages in set M for a given channel, and $count(M)$ denotes an auxiliary variable whose value indicates the total number of times messages in M have been sent along the channel since the beginning of system execution.

Definition: For a given specification, P *leads-to* Q *via* M iff

(i) for all e_r, $\forall m \in M\ [P \wedge e_r(m) \Rightarrow Q']$,

(ii) for all e, $P \wedge e \Rightarrow P' \vee Q'$, and

(iii) $P \wedge count(M) \geq k$ *leads-to* $Q \vee count(M) \geq k+1$

Given the channel progress assumption, we have the following leads-to rule (in addition to the ones presented in Section 4).

Leads-to rule: For a given specification, P *leads-to* Q if

• for some M, P *leads-to* Q *via* M [Message]

We next give a few general observations about the use of our notation and proof method for specifying distributed systems.

First, each nonauxiliary state variable in a distributed system, other than channel state variables, is local to some entity and can be accessed by any event of that entity. Suppose we want to refine the entity into a network of entities. To do so, we may have to make some of the nonauxiliary state variables of the entity into auxiliary variables and also introduce new state variables (more on refinement in the next section).

Second, specific applications of our proof rules may be very simple for events that access a small subset of state variables. In applying the first invariance rule, for instance, if none of the free state variables in P is updated by event e then $P \wedge e \Rightarrow P'$ is trivially satisfied. While most events in a distributed system access a small subset of state variables, the above observation is applicable to any system specification. Note that information on the subset of state variables accessed by an event is available from the syntax of the event definition.

Lastly, in applying leads-to rules to prove a progress property, we must be careful to avoid circular reasoning. A good practice is to present the proof as a sequence of leads-to properties: $L_0, L_1, \cdots, L_m$. Suppose L_0 is the desired property. To prove L_i in the sequence by a leads-to rule, or a lemma, that uses another progress property L_j, we require that L_i precedes L_j in the sequence.

6. Refinement and Projection of Relational Specifications

Throughout this section, we consider specifications A and B, each consisting of a state transition system and a set of fairness assumptions. We introduce two relations between A and B: *A is a refinement of B*, and *A is a well-formed refinement of B*. Before defining what they mean, we mention two possible applications for motivation: First, A is the specification of a multifunction communication protocol and B is the specification of a smaller protocol that implements just one of the functions of A [Shankar & Lam 83]. Second, A is the specification of a program module and B is the specification of its user interface.[10]

The refinement relations are useful for composing system specifications, as well as for constructing proofs of system properties, in a hierarchical fashion. (We will elaborate on applications in Section

[10] Actually, a small extension to the theory presented in this section is needed for interfaces; see [Lam & Shankar 87].

9.) In general, we proceed as follows. Suppose B is given or is specified first, and some desirable properties have been proved for B. We would like to derive A from B such that some or all of the desirable properties proved for B are guaranteed, by satisfying certain conditions, to be properties of A. That is, they do not have to be proved again for A.

To define the refinement relation, let V_A and V_B denote the state variable sets of A and B respectively. Specifically, let V_A be the set $\{v_1, v_2, \cdots, v_n\}$ and V_B the subset $\{v_1, v_2, \cdots, v_m\}$, where $m \leq n$. That is, in deriving A from B, every state variable in B is kept as a state variable in A with the same name and the same domain of values. (This is not a restriction, as we shall see, because such a state variable can be made into an auxiliary variable in A.) Since V_B is a subset of V_A there is a projection mapping from the states of A to the states of B, defined as follows: those states in A having the same values for $\{v_1, v_2, \cdots, v_m\}$ are mapped to the same state in B. We further require that every parameter in B is a parameter in A with the same name and same domain of values. Given the above requirements, any state formula, say P, of B is a state formula of A and can be interpreted directly for A without any translation. The interpretation is this one: if state t of B satisfies P then any state of A whose image is t under the projection mapping satisfies P.

For clarity, we assume that A and B have finite sets of state variables and parameters. The domain of a state variable (or parameter) may be countably infinite.

Let $\{a_i\}$ denote the set of events of A, and $\{b_k\}$ the set of events of B. We first provide conditions for an event in specification A to be a refinement of events in specification B. An informal explanation then follows.

Event a_i in A is a refinement of events in B if, for some invariant R_A of A, one of the following holds:

$$R_A \wedge a_i \Rightarrow \exists k\, [b_k] \qquad \text{(event refinement condition)}$$

$$R_A \wedge a_i \Rightarrow v_1'=v_1 \wedge v_2'=v_2 \wedge \cdots \wedge v_m'=v_m \qquad \text{(null image condition)}$$

Very often, a_i is the refinement of a single event in B. In this case, to check that a_i satisfies the event refinement condition, it is sufficient to show that, for some b_k, either $a_i \Rightarrow b_k$ or $R_A \wedge a_i \Rightarrow b_k$.

Informally, the meaning of event a_i being a refinement of events in B is the following: For every state transition defined by a_i that is observable in the state space of B, the same observable state transition is defined in B. More precisely, if a_i can take A from state s_1 to s_2 then there is some event b_k that can take B from state t_1 to t_2, where t_1 and t_2 are the images of s_1 and s_2 respectively under the projection mapping. This condition can be relaxed by introducing an invariant property R_A of A, in which case the condition has to hold only for each (s_1, s_2) pair such that s_1 and s_2 satisfy R_A. Note that the invariant R_A introduced will have to be proved separately to be a property of A.

The null image condition says the following: Event a_i is a refinement of events in B if none of its state transitions are observable in B under the projection mapping, namely, $t_1 = t_2$ for all s_1 and s_2 reachable in A such that (s_1, s_2) is defined by a_i. This can be checked very simply by noting that the action of a_i does not update any state variable belonging to V_B.

Suppose specification B is given, and specification A is to be derived from B. The invariant properties needed to guarantee events in A to be refinements of events in B often arise naturally in the following manner. Suppose we want to replace a state variable x in B by two state variables y and z in A; also, x is to become an auxiliary variable in A. To prove $a_i \Rightarrow b_k$, where b_k may contain x as a free variable, a state formula specifying the relation between x, y and z in A must be included as a conjunct of R_A. Note that this relation encodes the same information as the multi-valued possibilities mapping of Lynch and Tuttle [1987]. For the special case of the relation being a function, the function is just like the state functions used by Lamport [1983a]. We provide an example to illustrate this observation.

Example. Let x be a state variable of B. Its domain is the set of natural numbers. The following event is defined in B:

$$b_1 \equiv even(x) \wedge x'=x+1$$

where $even(x)$ is true iff x is an even number. In deriving A from B, suppose we introduce a variable y with domain $\{0,1\}$ to replace x, and the following event is defined:

$$a_1 \equiv y=0 \wedge y'=1 \wedge x'=x+1$$

Event a_1 is a refinement of b_1 given that $y=x \bmod 2$ is invariant for A. Note that x can be made into an auxiliary variable of A so that it does not have to be included in an actual implementation of A.

We next consider the refinement of messages. Let M be a set of messages that can be sent along a channel in B. In deriving A from B, the message set M can be refined as follows: Each message m in M is refined to a nonempty set N_m of messages in A. For two distinct messages i and j in M, we require $N_i \cap N_j = \emptyset$. For message m in B and message n in A, we say that n is a refinement of m, or m is the image of n, if and only if $n \in N_m$. Let N denote the set of messages that can be sent along the same channel in A,

$$N = \bigcup_{m \in M} N_m \bigcup N_{new}$$

where N_{new} is a set of new messages, if any. Such new messages are not observable in the state space of B and are said to have a *null image* in B. Note that the receive events assumption must be satisfied by specification A for all messages in N.

Example. In B, the message set for some channel consists of the message *ack* only. In A, the message set for the same channel is refined to $\{ack0,\ ack1,\ nak\}$, such that *ack* is the image of *ack*0 and *ack*1, and the new message *nak* has a null image.

If N is different from M for a channel, then the channel state variables in A and B have different domains for the same channel. The projection mapping from channel states in A to channel states in B is defined as follows [Lam & Shankar 84]: Let $y = <n_1, n_2, \cdots>$ be a sequence of messages representing a channel state in A. The image of the channel state, denoted by $image(y)$, is the sequence obtained by replacing each message in y by its image in B and deleting null images from the resulting sequence.

Given the above definition of projection mapping for channel states, a state formula of B, say P, can be interpreted for A as before, namely: state s of A satisfies P iff the image of s in B satisfies P. In this case, however, a translation between the message sets N and M is needed to interpret state formulas of B for A.

For a channel with message set N in A and message set M in B, let y denote the channel state variable in A, and x the channel state variable in B. The send and receive primitives in B are

$$Send(m) \equiv \mathrm{x}'=\mathrm{x}@m \text{ and } Rec(m) \equiv \mathrm{x}=m@\mathrm{x}'$$

The primitives for the same channel in A are

$$Send(n) \equiv \mathrm{y}'=\mathrm{y}@n \text{ and } Rec(n) \equiv \mathrm{y}=n@\mathrm{y}'$$

For send and receive events in A to be refinements of events in B, it is necessary that send and receive primitives in A are refinements of send and receive primitives in B for the same channel. To show that such send and receive primitives satisfy the event refinement condition, let x be an auxiliary state variable of A. For every message n in N with a nonnull image m in M, add the conjunct $\mathrm{x}'=\mathrm{x}@m$ to the formula defining $Send(n)$, and the conjunct $\mathrm{x}'=Tail(\mathrm{x})$ to the formula defining $Rec(n)$. It is easy to see that $\mathrm{x}=image(\mathrm{y})$ is an invariant of A, and that this invariant property ensures that the send and receive primitives satisfy the event refinement condition. Note that the relation between x and y defined by the invariant encodes the same information as the projection mapping defined between channel states in A and channel states in B.

Let $Initial_A$ and $Initial_B$ be state formulas defining the initial conditions of specifications A and B respectively.

Definition: A is a refinement of B if and only if every event in A is a refinement of events in B and $Initial_A \Rightarrow Initial_B$.

We say that B is an image of A under the projection mapping if and only if A is a refinement of B; that is, the relation *image* is the inverse of the relation *refinement* by definition. In some applications, we are first given A, and B is to be derived from A. For example, let A be some multifunction communication protocol. To prove that A has certain desirable properties, the following approach may be taken: Derive from A, single-function protocols that are images of A. Prove that these single-function protocols have the desirable properties, and infer that A has the same properties by the lemmas and theorems presented below.

Recall that P is a state formula of specification B iff every free variable of P is either in V_B or is a parameter.

Theorem 1: Let specification A be a refinement of specification B. If P is invariant for B then P is invariant for A, where P is an arbitrary state formula of B.

Proof: Let R_B denote a state formula that satisfies the first invariance rule for B and $R_B \Rightarrow P$. Let R_A be an invariant of A that makes events of A satisfy the event refinement condition or the null image condition. First, from $Initial_A \Rightarrow Initial_B$ and $Initial_B \Rightarrow R_B$, we have $Initial_A \Rightarrow R_B$. Second, for every event a_i of A that satisfies the event refinement condition, we have

$$\begin{aligned} R_B \wedge R_A \wedge a_i &\Rightarrow R_B \wedge (\exists k\,[b_k]) \\ &\Rightarrow \exists k\,[R_B \wedge b_k] \\ &\Rightarrow R_B{}' \end{aligned}$$

For every event a_i of A that satisfies the null image condition, we have

$$R_B \wedge R_A \wedge a_i \Rightarrow R_B \wedge v_1'=v_1 \wedge v_2'=v_2 \wedge \cdots \wedge v_m'=v_m$$
$$\Rightarrow R_B'$$

Thus, R_B is invariant for A by the first invariance rule. We know that $R_B \Rightarrow P$ holds for A, and the proof is complete by the second invariance rule.

Q.E.D.

For a given specification, the following property

for all event e, $P \wedge e \Rightarrow P' \vee Q'$

is called *P unless Q*, which is a safety property [Chandy & Misra 1988]. If A is a refinement of B, the following lemma says that every *unless* property of B is also an *unless* property of A.

Lemma 3: Let specification A be a refinement of specification B. If P *unless* Q holds for B then P *unless* Q holds for A, where P and Q are arbitrary state formulas of B.

A proof of Lemma 3 is immediate by applying the event refinement and null image conditions. We next consider leads-to properties of B and provide various sufficient conditions for some, or all, of these properties to be properties of A. We first state a useful lemma, which is the PSP theorem in [Chandy & Misra 88].

Lemma 4: For a given specification, if P *unless* Q holds and Q_1 *leads−to* Q_2, then

$P \wedge Q_1$ *leads−to* $Q \vee (P \wedge Q_2)$ for the specification.

A proof of Lemma 4 is given in [Lam & Shankar 88]. (Note that our inference rules and fairness assumptions are different from those of Chandy and Misra [1988].)

Suppose we have proved that P *leads−to* Q for B. The proof may be direct, by an application of the Implication rule or the weak Event rule, or it may consist of a sequence of leads-to properties. We present below various conditions under which we can infer P *leads−to* Q for A, where P and Q are state formulas of B.

First, if the proof is by an application of the Implication rule, we immediately have P *leads−to* Q for A by Theorem 1. Next, consider the weak Event rule. Suppose P *leads−to* Q *via* b_j for B, where b_j has weak fairness. To guarantee that P *leads−to* Q for A, we may apply Lemma 5, 6 or 7, where P and Q are assumed to be known. Or we may apply Lemma 8, where P and Q can be arbitrary state formulas of B. Proofs of these lemmas are given in [Lam & Shankar 88]. In the balance of this section, R_A denotes some invariant of A.

Lemma 5: Let specification A be a refinement of specification B, and b_j an event that has fairness in B. If P *leads−to* Q *via* b_j for B, then P *leads−to* Q for A if there is some event in A, denoted by a_j, that has weak fairness, is a refinement of b_j, and

$R_A \wedge P \Rightarrow Q \vee enabled(a_j)$

The last condition in Lemma 5 can be weakened if event a_j has the following **noninterference property** in A: for all event $a_i, i \neq j$,

$$R_A \wedge enabled(a_j) \wedge a_i \Rightarrow enabled(a_j)'.$$

Lemma 6: Let specification A be a refinement of specification B, and b_j an event that has fairness in B. If P *leads-to* Q *via* b_j for B, then P *leads-to* Q for A if there is some event in A, denoted by a_j, that has weak fairness and the noninterference property, is a refinement of b_j, and

$$R_A \wedge P \textit{ leads-to } Q \vee enabled(a_j) \text{ for } A.$$

However, if event a_j has strong fairness in A, the noninterference property is not needed. We have the following result.

Lemma 7: Let specification A be a refinement of specification B, and b_j an event that has fairness in B. If P *leads-to* Q *via* b_j for B, then P *leads-to* Q for A if there is some event in A, denoted by a_j, that has strong fairness, is a refinement of b_j, and

$$R_A \wedge P \textit{ leads-to } Q \vee enabled(a_j) \text{ for } A.$$

Now, suppose we want to guarantee that if P *leads-to* Q *via* b_j for B, then P *leads-to* Q for A, for arbitrary state formulas P and Q. We need conditions that do not make use of P or Q.

SWF Condition: For event b_j that has weak fairness in B, an event in A, denoted by a_j, is a well-formed refinement of b_j if

- a_j is a refinement of b_j
- $R_A \wedge enabled(b_j) \Rightarrow enabled(a_j)$, and
- a_j has weak fairness

The conditions in SWF are simple and easy to use. It has been our experience, in specifying communication protocols and concurrency control protocols, that many events can be refined to satisfy SWF. Such an event is said to be a strongly well-formed refinement. But sometimes, SWF cannot be satisfied, or b_j has strong fairness in B. We provide a second condition:

WF Condition: For event b_j that has fairness (weak or strong) in B, an event in A, denoted by a_j, is a well-formed refinement of b_j if

- a_j is a refinement of b_j
- $R_A \wedge enabled(b_j)$ *leads-to* $enabled(a_j)$ for A, and
- either a_j has weak fairness and the noninterference property
  or a_j has strong fairness

Lemma 8: Let specification A be a refinement of specification B, and b_j an event that has weak fairness in B. If P *leads-to* Q *via* b_j for B, then P *leads-to* Q for A if there is some event in A that is a well-formed refinement of b_j.

Note that Lemmas 5-7 are proved for an event b_j that has fairness (weak or strong) in B. Lemma 8 is proved for an event b_j that has weak fairness in B; the more general result for an event b_j that has strong fairness in B is included in Theorem 2 below. For some applications, it is desirable that *every* leads-to property of B is a leads-to property of A. A sufficient condition is the following:

Definition: Specification A is a *well-formed refinement* of specification B (or B is a *well-formed image* of A) if and only if

- A is a refinement of B, and
- for every event b_j that has fairness (weak or strong) in B, there is an event in A that is a well-formed refinement of b_j.

Theorem 2: Let specification A be a well-formed refinement of specification B. If P *leads−to* Q for B then P *leads−to* Q for A, where P and Q are arbitrary state formulas of B.

A proof of Theorem 2 is given in [Lam & Shankar 88]. As an example, let us now consider a refinement of the airplane specification in Section 2. Let the state variables y and z be augmented by a third state variable x, with domain over all integers, so that we will be reasoning about trajectories of the airplane in 3-dimensional space. Initially, x=0. Five events, labeled by *, are defined in terms of events of the 2-variable specification as follows:

$$\mathit{TakeOff}^* \equiv \mathit{TakeOff} \wedge x=0 \wedge -10 \le x' \le 10$$
$$\mathit{Landing}^* \equiv \mathit{Landing} \wedge -10 \le x \le 10 \wedge x'=0$$
$$\mathit{Fly}^* \equiv \mathit{Fly} \wedge -10 \le x \le 10$$
$$\mathit{FlyHigher}^* \equiv \mathit{FlyHigher} \wedge -10 \le x \le 10$$
$$\mathit{FlyLower}^* \equiv \mathit{FlyLower} \wedge -10 \le x \le 10$$

It is easy to see that the above events are refinements of corresponding events in the 2-variable specification. Add the following two events:

$$\mathit{FlyLeft} \equiv 1 \le y \le N-1 \wedge 10 \le z \le 20 \wedge -10 < x \le 10 \wedge x'=x-1$$
$$\mathit{FlyRight} \equiv 1 \le y \le N-1 \wedge 10 \le z \le 20 \wedge -10 \le x < 10 \wedge x'=x+1$$

The two new events are also refinements because they are null-image events whose occurrences are not observable in the 2-variable specification. Hence, the new 3-variable specification is a refinement of the 2-variable specification. Like the 2-variable specification, the events *TakeOff**, *Fly** and *Landing** have weak fairness. It is easy to show that each original event and its refinement satisfy the SWF condition given the following invariant requirement:

$$R_A \equiv (y=0 \wedge z=0 \Rightarrow x=0) \wedge (1 \le y \le N-1 \Rightarrow -10 \le x \le 10)$$

The above is easily shown to be an invariant of the 3-variable specification. Thus invariant and progress properties of the 2-variable specification are also properties of the 3-variable specification. Once proved for the 2-variable specification, they do not have to be proved again for the 3-variable specification.

In summary, we have given several conditions to ensure that some or all of the properties of specification B are properties of specification A. Of these conditions, the well-formed refinement relation between two specifications is the strongest. (Its semantics is essentially the same as the *simulation* or *implementation* relation of other authors [Abadi & Lamport 88, Lamport 83b, Lamport 85, Lynch & Tuttle 87].) For some applications, such a condition may be too strong to be useful. For these applications, it may be enough to ensure that only safety properties of B are preserved in A, or safety properties and some *specific* progress properties of B are preserved in A. In this case, only those events that are needed in the proof of the desired progress properties of B have to satisfy the WF or SWF condition. In fact, the weaker conditions in Lemmas 5-7 can be used instead.

In general, the SWF condition should be regarded as a 'shortcut.' For a given event, the SWF condition is checked first. If it is too strong and cannot be easily satisfied, then one of the weaker conditions is used.

In refining the events of B to get events of A, the event refinement and null image conditions are generally easy to satisfy. However, if some state variables in B are replaced by new state variables in A (and made into auxiliary variables in A) then finding an invariant that specifies the relation between the old and new state variables may be nontrivial. This problem is the same as finding a multi-valued possibilities mapping from the states of A to the states of B, as in [Lynch & Tuttle 87].

7. Auxiliary Variables

We can now give a rigorous explanation of auxiliary variables. Consider a specification A consisting of a state transition system and some fairness assumptions. Let the initial condition of A be denoted by $Initial_A$ and its events by $\{a_i\}$. Suppose the set $\mathbf{v}$ of state variables of A is partitioned into two sets, $\mathbf{u}$ and $\mathbf{x}$, such that an observer can only see the state variables in $\mathbf{x}$. In this case, the observable behaviors of A are behaviors of a specification C derived from A as follows: The state variables of C are the ones in $\mathbf{x}$. The initial condition of C is

$$Initial_C \equiv \exists \mathbf{u}[Initial_A]$$

The events of C are defined by

$$c_i \equiv \forall \mathbf{u} \exists \mathbf{u}'[a_i]$$

for every event a_i of A that does not have a null image in the state space of C. Event c_i has a fairness assumption in C if and only if event a_i has the same fairness assumption in A. [11]

Suppose the state variables in $\mathbf{u}$ have been shown to be auxiliary variables of specification A. In Section 3, we assert that auxiliary variables, introduced for specification and verification, do not have to be included in an actual implementation. The meaning of the assertion is this: instead of implementing specification A, we implement specification C which does not have the auxiliary variables in $\mathbf{u}$. (Note that $\mathbf{x}$ may contain other auxiliary variables that are not in $\mathbf{u}$.) We next discuss how properties of C are related to properties of A.

[11] Specifically, it is assumed that there is no event of A that has fairness in A and a null image in C. (We cannot think of a reason for having such events.) Given a finite number of such events, this assumption is not necessary. It is made to simplify the proof of Theorem 4 in [Lam & Shankar 88].

By the definition of auxiliary variables, every event a_i of A that does not have a null image in the state space of C satisfies $a_i \Rightarrow c_i$, which is a form of the event refinement condition. Also, we have

$$\begin{aligned} Initial_A &\Rightarrow \exists \mathbf{u}[Initial_A] \\ &= Initial_C \end{aligned}$$

Thus specification C is an image of specification A under the projection mapping. Additionally, we have

$$\begin{aligned} enabled(c_i) &= \exists \mathbf{x}' \forall \mathbf{u} \exists \mathbf{u}'[a_i] \\ &\Rightarrow \forall \mathbf{u} \exists \mathbf{x}' \exists \mathbf{u}'[a_i] \\ &= \forall \mathbf{u} \exists \mathbf{v}'[a_i] \end{aligned}$$

Since variables in **u** do not occur free in $enabled(c_i)$, we have

$$\forall \mathbf{u}(enabled(c_i) \Rightarrow \exists \mathbf{v}'[a_i])$$

which is, by our convention,

$$\begin{aligned} enabled(c_i) &\Rightarrow \exists \mathbf{v}'[a_i] \\ &= enabled(a_i) \end{aligned}$$

Thus, if event a_i has weak fairness, c_i and a_i satisfy the SWF condition. If event a_i has strong fairness, c_i and a_i satisfy the WF condition. And we have the following result:

Corollary 1: Specification C is a well-formed image of specification A.

From Corollary 1 and the results in Section 6, we know that properties of C such as, P is invariant, P *unless* Q and P *leads-to* Q, are also properties of A, where P and Q are arbitrary state formulas of C (that is, variables in **u** do not occur free in P or Q).

Actually, we know more about specification C than what is in Corollary 1. For specifications A and C given above, we have the following results.

Theorem 3: Let p denote an arbitrary state formula of A. If p is invariant for A, then $\exists \mathbf{u}[p]$ is invariant for C.

Corollary 2: P is invariant for C if and only if P is invariant for A, where P is an arbitrary state formula of C.

Theorem 4: P *leads-to* Q for C if and only if P *leads-to* Q for A, where P and Q are arbitrary state formulas of C.

Proofs of the above results are given in [Lam & Shankar 88]. Let p and q denote arbitrary formulas of A. It can be easily shown that if p *unless* q holds for A then $\exists \mathbf{u}[p]$ *unless* $\exists \mathbf{u}[q]$ holds for C. (See Lemma 11 in [Lam & Shankar 88].) However, if p *leads-to* q for A, it does not follow that $\exists \mathbf{u}[p]$ *leads-to* $\exists \mathbf{u}[q]$ for C. (There are counterexamples.)

Auxiliary variables play an important role in our methodology for refining specifications. Let us revisit a scenario considered in Section 6. In the process of refining a specification B, suppose we want to replace a state variable x by two new state variables y and z. In our methodology, we first derive a specification A to be a refinement of B. Specification A has all three state variables x, y and z. The events of A are then refined such that state variable x is an auxiliary variable of A. Lastly, specification C without the auxiliary variable x is derived from A as a well-formed image. A nontrivial example can be found in Section 8 where three specifications of the alternating-bit protocol are given.

We next consider the special case of channel state variables. In modeling communication protocols, the messages that are sent along a channel can be represented by a set of message types [Shankar & Lam 87a]. Each message type is a tuple, for example, $(data, cn)$. Each element of the tuple, called a message field, has a specified domain of allowed values. In the above example, the domain of $data$ is a set of allowed sequences of bits; the domain of cn may be the set $\{0,1\}$. The set of messages represented by a message type is the cartesian product of the domains of its message fields.

In specifying communication protocols, it is sometimes convenient to use *auxiliary message fields*. For example, consider the message type $(data, cn, n)$ where n is a natural number. Think of n as the unbounded sequence number of a message while cn is the corresponding cyclic sequence number that is actually implemented. (Unbounded sequence numbers are needed for specification and proofs.) Since unbounded sequence numbers are not practically implementable, the message field n should be auxiliary in the same sense as an auxiliary variable.

Adding a new field, such as n, to a message type, such as $(data, cn)$, changes the domain of the channel state variable $\mathbf{z}_i$. To ensure that the new field is auxiliary, in the above sense, we can use the following reasoning. Imagine that the channel state variable consists of two variables $\mathbf{z}_i$ and $\mathbf{u}_i$, where $\mathbf{z}_i$ represents the channel state and $\mathbf{u}_i$ represents the sequence of n message fields associated with messages of type $(data, cn)$ in $\mathbf{z}_i$. The message field n is auxiliary iff $\mathbf{u}_i$ is an auxiliary variable of the system specification; informally, $\mathbf{u}_i$ does not affect the enabling condition of any event nor does it affect the update of any nonauxiliary state variable.

8. Specification Examples

To illustrate the various concepts and results presented in this paper, we give three specifications of the alternating-bit protocol, each consisting of a state transition system and a set of fairness assumptions. Specification AB_1 uses state variables and message fields with unbounded domains (i.e., natural numbers). We prove that AB_1 has the desired safety and progress properties of the alternating-bit protocol. (Applications of the leads-to Message rule are illustrated in the proof.) Specification AB_2 is derived by adding binary-valued state variables and message fields to AB_1. We prove that AB_2 is a well-formed refinement of AB_1. Therefore, safety and progress properties proved for AB_1 are also properties of AB_2. Furthermore, we show that those state variables and message fields with unbounded domains are *auxiliary* in AB_2. Specification AB_3 is obtained from AB_2 by deleting the auxiliary state variables and message fields. AB_3 is a well-formed image of AB_2. AB_3 is most suitable for implementation because it is the smallest (i.e., its sender has four states, its receiver has two states, and only modulo-2 sequence numbers are used in its messages).

For all three specifications of the alternating-bit protocol, consider the network configuration in Figure 2, where entity 1 is the sender and entity 2 is the receiver of data blocks. Assume that the channels are lossy; that is, if the channel state variable $\mathbf{z}_i$ is not null, then a loss event is enabled whose occurrence deletes an arbitrary message in $\mathbf{z}_i$. (Recall that the channel progress assumption has to be satisfied.) Initially, both channels are empty.

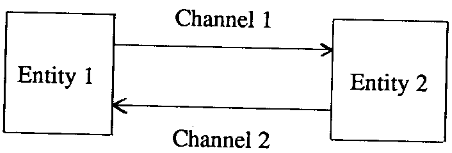

Figure 2. Network topology.

Notation. Let *guard* be a state formula and *action* an event formula such that $guard \Rightarrow enabled(action)$. Let $\mathbf{y}$ be the subset of state variables such that the variables $\{y' : y \in \mathbf{y}\}$ occur free in *action*. Define the event formula

$$guard \rightarrow action \;\equiv\; (guard \wedge action) \vee (\neg guard \wedge \mathbf{y}'=\mathbf{y})$$

Note that if *guard* is false, none of the state variables in $\mathbf{y}$ is updated by the above event formula.

Specification AB_1

Let *DATA* denote the set of data blocks that can be sent in this protocol. Let *natural* denote the set of natural numbers {0, 1, $\cdots$ }. Entity 1 sends only one type of messages, namely, $(D, data, n)$ where D is a constant denoting the name of the message type, the domain of the message field *data* is *DATA*, and the domain of the message field n is *natural*. Entity 2 sends only one type of messages, namely, (ACK, n) where *ACK* is a constant denoting the name of the message type, and the domain of the message field n is *natural*. Below, we use a Pascal-like notation to define state variables and their domains. We use *empty* to denote a constant not in *DATA*.

- Entity 1 state variables

  produced: sequence of *DATA*, initially null.
  s: *natural*, initially 0.
  sendbuff: $DATA \cup empty$, initially *empty*.

- Entity 1 events

$$Produce(data) \equiv \quad sendbuff{=}empty \wedge produced'{=}produced@data \wedge s'{=}s+1 \wedge sendbuff'{=}data$$

$$SendD \equiv \quad sendbuff{\neq}empty \wedge Send_1(D, sendbuff, s-1)$$

$$RecACK(n) \equiv \quad Rec_2(ACK, n) \wedge ((sendbuff{\neq}empty \wedge s{=}n) \rightarrow sendbuff'{=}empty)$$

- Entity 2 state variables

  consumed: sequence of *DATA*, initially null.
  r: *natural*, initially 0.

- Entity 2 event

$$\begin{aligned} RecD(data, n) \equiv \quad & Rec_1(D, data, n) \\ & \wedge Send_2(ACK, r) \\ & \wedge (r=n \rightarrow (consumed'=consumed@data \wedge r'=r+1)) \end{aligned}$$

Note that the events $RecACK(n)$ and $RecD(data, n)$ satisfy the receive events assumption in Section 5. (The enabling condition of $RecACK(n)$ is simply $Head(\mathbf{z}_2)=(ACK, n)$.) The desired invariant property of the alternating-bit protocol is I_0 below.

Notation. For a sequence *seq*, we use $|seq|$ to denote the length of the sequence. For channel state variable $\mathbf{z}_1$, we use $<D, n>$ to denote a sequence of zero or more copies of the $(D, produced(n), n)$ message, where $produced(n)$ is the nth element of *produced* for $n=0,1,\cdots$. For channel state variable $\mathbf{z}_2$, we use $<ACK, n>$ to denote a sequence of zero or more copies of the (ACK, n) message.

Invariant properties:

$I_0 \equiv$ *consumed* is a prefix of *produced*

$I_1 \equiv |produced|=s \wedge |consumed|=r$

$I_2 \equiv (sendbuff=empty \wedge r=s)$
$\quad \vee (sendbuff=produced(s-1) \wedge (r=s \vee r=s-1))$

$I_3 \equiv sendbuff=empty \Rightarrow \mathbf{z}_1=<D, r-1> \wedge \mathbf{z}_2=<ACK, r>$

$I_4 \equiv sendbuff \neq empty \wedge s=r+1 \Rightarrow \mathbf{z}_1=<D, r-1>@<D, r> \wedge \mathbf{z}_2=<ACK, r>$

$I_5 \equiv sendbuff \neq empty \wedge s=r \Rightarrow \mathbf{z}_1=<D, r-1> \wedge \mathbf{z}_2=<ACK, r-1>@<ACK, r>$

Let $I \equiv I_0 \wedge I_1 \wedge I_2 \wedge I_3 \wedge I_4 \wedge I_5$. It is straightforward to show that I satisfies the first invariance rule. (In applying the rule, keep in mind that the loss event of each channel is in the set of events.)

The desired progress property of the alternating-bit protocol is L_0 below. To prove L_0, five additional leads-to properties are given. For brevity, we use $(D, n-1)$ to denote the message $(D, produced(n-1), n-1)$.

Progress properties:

$L_0 \equiv sendbuff \neq empty \wedge s=n$ *leads-to* $sendbuff=empty \wedge s=n$

$L_1 \equiv sendbuff \neq empty \wedge s=n \wedge r=n-1$ *leads-to* $sendbuff \neq empty \wedge s=n \wedge r=n$

$L_2 \equiv sendbuff \neq empty \wedge s=n \wedge r=n$ *leads-to* $sendbuff=empty \wedge s=n \wedge r=n$

$L_3 \equiv$ $sendbuff{\neq}empty \wedge s{=}n \wedge r{=}n{-}1 \wedge count(D,n{-}1){\geq}k$
$leads{-}to\ count(D,n{-}1){\geq}k{+}1 \vee (sendbuff{\neq}empty \wedge s{=}n \wedge r{=}n)$

$L_4 \equiv$ $sendbuff{\neq}empty \wedge s{=}n \wedge r{=}n \wedge count(ACK,n){\geq}l$
$leads{-}to\ count(ACK,n){\geq}l{+}1 \vee (sendbuff{=}empty \wedge s{=}n \wedge r{=}n)$

$L_5 \equiv$ $sendbuff{\neq}empty \wedge s{=}n \wedge r{=}n \wedge count(ACK,n){\geq}l \wedge count(D,n{-}1){\geq}k$
$leads{-}to\ count(D,n{-}1){\geq}k{+}1 \vee count(ACK,n){\geq}l{+}1 \vee (sendbuff{=}empty \wedge s{=}n \wedge r{=}n)$

Proof of L_0:

Assume that *SendD* has weak fairness in specification AB_1.

L_3 holds via event *SendD*.
L_1 holds via message set $\{(D, n{-}1)\}$, using L_3.

L_5 holds via event *SendD*.
L_4 holds via message set $\{(D, n{-}1)\}$, using L_5.
L_2 holds via message set $\{(ACK, n)\}$, using L_4.

By Implication, Transitivity and Disjunction rules on L_1 and L_2, we get

$$sendbuff{\neq}empty \wedge s{=}n \wedge (r{=}n \vee r{=}n{-}1)\ leads{-}to\ sendbuff{=}empty \wedge s{=}n$$

L_0 follows from the above property and I_2 by Lemma 2.

Q.E.D.

From the above proof, we see that specification AB_1 requires a fairness assumption for event *SendD* only. The other events do not have to be fairly scheduled. (Of course, the channel progress assumption is needed and it can be viewed as a fairness assumption.)

Specification AB_2

AB_2 is derived from AB_2 by adding binary-valued state variables *cs* and *cr*. The state variables *s* and *r* are made into auxiliary variables. Also, a modulo-2 sequence number field *cn* is added to each message type. The message field *n* is also made auxilary.

- Entity 1 state variables

  produced, *s*, and *sendbuff* as in AB_1.
  cs: $\{0, 1\}$, initially 0.

- Entity 1 events

  $Produce^*(data) \equiv$ $Produce(data) \wedge cs'{=}(cs{+}1) \bmod 2$

  $SendD^* \equiv$ $sendbuff{\neq}empty$
  $\wedge Send_1(D, sendbuff, s{-}1, (cs{-}1) \bmod 2)$

$$RecACK^*(n, cn) \equiv \quad Rec_2(ACK, n, cn)$$
$$\wedge((sendbuff \neq empty \wedge cs=cn) \rightarrow sendbuff'=empty)$$

- Entity 2 state variables

  consumed and r as in AB_1.
  cr: {0, 1}, initially 0.

- Entity 2 event

$$RecD^*(data, n, cn) \equiv Rec_1(D, data, n, cn)$$
$$\wedge Send_2(ACK, r, cr)$$
$$\wedge(cr=cn \rightarrow$$
$$(consumed'=consumed@data \wedge r'=r+1 \wedge cr'=(cr+1) \bmod 2))$$

Note that the events *RecACK** and *RecD** satisfy the receive events assumption in Section 5. In order for AB_2 to be a well-formed refinement of AB_1, we require that *SendD** has weak fairness.

Proposition: AB_2 is a well-formed refinement of AB_1.

Proof: Event *SendD** is a well-formed refinement of *SendD* because it satisfies the SWF condition. Since *SendD* is the only event that has a fairness assumption in AB_1, it is sufficient to prove that the other events in AB_2 satisfy the event refinement condition. (Note that the initial condition of AB_2 implies the initial condition of AB_1.)

Clearly, *Produce** (*data*) satisfies the event refinement condition because *Produce* (*data*) is one of its conjuncts. For $RecACK^*(n, cn)$ to be a refinement of $RecACK(n)$, it is sufficient that the following is invariant:

$$R_0 \equiv \quad Head(z_2)=(ACK, n, cn) \wedge sendbuff \neq empty \wedge cs=cn \Rightarrow s=n$$

For $RecD^*(data, n, cn)$ to be a refinement of $RecD(data, n)$, it is sufficient that the following is invariant:

$$R_1 \equiv \quad Head(z_1)=(D, data, n, cn) \wedge cr=cn \Rightarrow r=n$$

Our proof that R_0 and R_1 are invariant for AB_2 is as follows. Define

$$R_2 \equiv \quad cs=s \bmod 2 \wedge cr=r \bmod 2$$
$$R_3 \equiv \quad (D, data, n, cn) \in z_1 \Rightarrow cn=n \bmod 2$$
$$R_4 \equiv \quad (ACK, n, cn) \in z_2 \Rightarrow cn=n \bmod 2$$

We first prove the following for AB_2 (proofs are given below):

(1) $R_2 \wedge R_3 \wedge R_4$ is invariant

(2) $I \wedge R_2 \wedge R_3 \wedge R_4 \Rightarrow R_0 \wedge R_1$

Next, we show that (1) and (2) imply that $R_0 \wedge R_1$ is invariant for AB_2, as follows. Let e^* denote an event in AB_2 and e the corresponding event in AB_1.

$I \wedge R_2 \wedge R_3 \wedge R_4 \wedge e^* \Rightarrow I \wedge R_0 \wedge R_1 \wedge e^*$ (from (2))

$\Rightarrow I \wedge e$ (by event refinement condition)

$\Rightarrow I'$ (I is invariant for AB_1)

Also, the initial condition of AB_2 implies the initial condition of AB_1 which satisfies I. Thus, I is invariant for AB_2 by the first invariance rule. From (2) above, $R_0 \wedge R_1$ is invariant for AB_2 by the second invariance rule.

Note that because of message refinement, the interpretation of I_3, I_4 and I_5 (conjuncts of I) requires a translation from messages in AB_1 to messages in AB_2, using the projection mapping for channel states defined in Section 6. Specifically, given the projection mapping, $<D, n>$ denotes a sequence of zero or more copies of the $(D, produced(n), n, i)$ message, while $<ACK, n>$ denotes a sequence of zero or more copies of the (ACK, n, i) message, where i is a parameter with domain $\{0, 1\}$.

To complete a proof of the proposition, we give proofs of (1) and (2) assumed above.

Proof of (1): Each of the conjuncts in $R_2 \wedge R_3 \wedge R_4$ satisfies the first invariance rule for AB_2, as follows:

The initial condition of AB_2 satisfies R_2.
$R_2 \wedge e \Rightarrow R_2'$ holds for $e = Produce^*(data)$ and $e = RecD^*(data, n, cn)$.
R_2 is not affected by any other event.

The initial condition of AB_2 satisfies R_3.
$R_2 \wedge R_3 \wedge SendD^* \Rightarrow R_3'$.
$R_3 \wedge RecD^*(data, n, cn) \Rightarrow R_3'$.
R_3 is not affected by any other event.

The initial condition of AB_2 satisfies R_4.
$R_2 \wedge R_4 \wedge SendACK^* \Rightarrow R_4'$.
$R_4 \wedge RecACK^*(n, cn) \Rightarrow R_4'$.
R_4 is not affected by any other event.

Proof of (2): Specifically, we prove that $I_2 \wedge I_4 \wedge I_5 \wedge R_2 \wedge R_3 \Rightarrow R_1$ and $I_2 \wedge I_4 \wedge I_5 \wedge R_2 \wedge R_4 \Rightarrow R_0$ hold for specification AB_2. We give below a detailed derivation of the latter; a derivation of the former is similar.

Assume the antecedent of R_0, namely, $Head(z_2)=(ACK, n, cn) \wedge sendbuff \neq empty \wedge cs=cn$. From I_2, I_4, I_5 and R_4, we know that X, Y or Z holds, where

$X \equiv s=r+1 \wedge Head(z_2)=(ACK, r, r \bmod 2)$

$Y \equiv s=r \wedge Head(z_2)=(ACK, r-1, (r-1) \bmod 2)$

$Z \equiv s=r \wedge Head(z_2)=(ACK, r, r \bmod 2)$

From R_2, we have $s \bmod 2 = cn$, which implies $\neg X \wedge \neg Y$. Hence Z holds, which implies $s=r=n$. And the consequent of R_0 holds.

Q.E.D.

Since AB_2 is a well-formed refinement of AB_1, the invariant and progress properties proved above for AB_1 are also properties of AB_2.

Now, consider the state variables s and r, and the message field n in each message type of specification AB_2. They are not practically implementable because their domains are unbounded. It is easy to see that the events of AB_2 satisfy the condition for auxiliary variables for s and r, i.e., their values affect neither the enabling condition nor the updates of the other state variables for each event. To see that the same condition is satisfied for message field n, rewrite the receive events of AB_2 in the following form:

$$RecACKs(cn) \equiv \exists n\ [RecACK^*(n, cn)]$$

$$RecDs(data, cn) \equiv \exists n\ [RecD^*(data, n, cn)]$$

The above receive events satisfy the receive events assumption in Section 5. Note that the safety properties of AB_2 do not depend on how the receive events are represented. Because none of the receive events has a fairness assumption, the progress properties of AB_2 also do not depend on how the receive events are represented. It is now easy to see that the events of AB_2 satisfy the auxiliary variable conditon for the message field n.

Specification AB_3

AB_3 is derived from AB_2 by deleting the auxiliary variables s and r and the auxiliary message field n, in the manner described in Section 7. By Corollary 1, AB_3 is a well-formed image of AB_2.

- Entity 1 state variables

  sendbuff and cs as in AB_2.

- Entity 1 events

$$Produce^{**}(data) \equiv sendbuff{=}empty \wedge cs'{=}(cs+1) \bmod 2 \wedge sendbuff'{=}data$$

$$SendD^{**} \equiv sendbuff{\neq}empty \wedge Send_1(D, sendbuff, (cs-1) \bmod 2)$$

$$RecACK^{**}(cn) \equiv Rec_2(ACK, cn) \wedge ((sendbuff{\neq}empty \wedge cs{=}cn) \rightarrow sendbuff'{=}empty)$$

- Entity 2 state variables

  cr as in AB_2.

- Entity 2 event

$$RecD^{**}(data, cn) \equiv Rec_1(D, data, cn) \wedge Send_2(ACK, cr) \wedge (cr{=}cn \rightarrow cr'{=}(cr+1) \bmod 2)$$

Specification AB_3 includes a weak fairness assumption for event $SendD^{**}$. Note that the events $RecACK^{**}(cn)$ and $RecD^{**}(data, cn)$ satisfy the receives events assumption in Section 5. Invariant and progress properties of AB_3 are inferred from those of AB_2 by applying Theorems 3 and 4 in Section 7.

We first apply Theorem 3. From $\exists s, r, produced, consumed\ [I \wedge R_0 \wedge R_1 \wedge R_2 \wedge R_3 \wedge R_4]$, we infer that the following state formulas are invariant for AB_3:

$$sendbuff{=}empty \Rightarrow cr{=}cs$$

$$sendbuff{=}empty \Rightarrow z_1{=}{<}D, cr-1{>} \wedge z_2{=}{<}ACK, cr{>}$$

$$sendbuff{\neq}empty \wedge cs{=}(cr+1) \bmod 2 \Rightarrow z_1{=}{<}D, cr-1{>}@{<}D, cr{>} \wedge z_2{=}{<}ACK, cr{>}$$

$$sendbuff{\neq}empty \wedge cs{=}cr \Rightarrow z_1{=}{<}D, cr-1{>} \wedge z_2{=}{<}ACK, cr-1{>}@{<}ACK, cr{>}$$

where

$<D, cr-1>$ denotes a sequence of zero or more copies of the $(D, data, (cr-1) \bmod 2)$ message for some $data$ in $DATA$,

$<D, cr>$ denotes a sequence of zero or more copies of the $(D, sendbuff, cr)$ message,

$<ACK, cr-1>$ denotes a sequence of zero or more copies of the $(ACK, (cr-1) \bmod 2)$ message, and

$<ACK, cr>$ denotes a sequence of zero or more copies of the (ACK, cr) message.

The following progress property of AB_1 can be derived from L_0 by applying the Disjunction theorem in [Chandy & Misra 88],

$$sendbuff{\neq}empty\ leads{-}to\ sendbuff{=}empty$$

Applying Theorem 4, the above progress property is a property of AB_3.

9. Discussions and Related Work

The basic constructs for specifying systems in the relational notation are *state formulas* and *event formulas*. A state formula defines a set of system states. An event formula defines a set of system state transitions. Additionally, parameters may be used for defining groups of related events, as well as groups of related system properties. We believe that our notation is easy to learn because states and state transitions are represented explicitly. Our objective is to retain much of the intuitive appeal of state-transition models (such as communicating finite state machines), but none of their limitations. Our proof method was also designed to use a minimal amount of notation with the goal that it will be accessible to protocol engineers.

The v' notation for specifying events as formulas in $v \cup v'$ is not unique to our work. The same notational device is used by various other authors [Lamport 83a, Hehner 84, Scheid & Holtsberg 88].

Prototypes of the relational notation and the proof method presented in this paper were described in [Shankar & Lam 84, Shankar & Lam 87a]. Our proof method is based upon a fragment of linear-time temporal logic [Chandy & Misra 88, Lamport 83a, Owicki & Lamport 82, Manna & Pnueli 84, Pnueli 86]. Motivated by examples in communication protocols, we introduced two small extensions to the body of work cited above. First, we defined the *P leads–to Q via M* relation. The resulting leads-to Message rule for unreliable channels is a very useful one for communication protocols.

Second, we advocate the approach of stating fairness assumptions explicitly for individual events as part of a specification, noting that for many systems not all events need be fairly scheduled. This contrasts with the approach of a blanket assumption that all events in a system are fairly scheduled according to some criterion. While this approach is not new (see [Pnueli 86]), our definition of the *allowed behaviors* of a specification differs from the definition of *fair computations* of [Pnueli 86] in that an allowed behavior may not be 'maximal.' Specifically, every fair computation of Pnueli [1986] is an allowed behavior in our model but not vice versa. Our definition of allowed behaviors is motivated by the specification of interfaces of program modules [Lynch & Tuttle 87, Lam & Shankar 87].

We refer to a state transition system given in the relational notation together with a set of fairness assumptions as a *relational specification*. In Section 6, we present a theory of refinement and projection of relational specifications. The theory has been adapted to relational specifications from our earlier work on protocol projections. The relation *A is a well-formed refinement of B*, for two specifications A and B, is by definition the inverse of the relation *B is a well-formed image of A* introduced in [Lam & Shankar 84].

Other authors have defined similar relations between specifications: *A implements B*, *A simulates B*, *A satisfies B*, etc. [Abadi & Lamport 88, Lamport 83b, Lamport 85, Lynch & Tuttle 87]. Informally, the meaning of every one of these relations is the following: every externally visible behavior allowed by A is also allowed by B. (There are some differences in how behaviors and observable behaviors are defined.) Our definition of *A is a well-formed refinement of B* is essentially the same. We differ from the others in how the above definition is applied. First, instead of using it directly, we introduced the relational notation as a specification formalism. In our experience, the event refinement, WF and SWF conditions, expressed in the relational notation, are very convenient to use. Second, if some state variables in B are replaced by new state variables in A, our approach is to find an invariant that specifies the relation between the new and old state variables. The approach of the other authors is to find a mapping from the states of A to the states of B. The relation to be found is the same one in each approach. The approaches differ in how the relation is represented. In [Abadi & Lamport 88], the existence of such mappings is addressed.

For many applications, we found the above relations, *well-formed refinement*, *implements*, etc., to be too strong to be useful. In refining a specification B to A, it is seldom the case that every progress property of B must be preserved in A. In most cases, only *some* specific progress properties of B are to be preserved. Our conditions given in Section 6 are designed for such use. In our theory, the *refinement* relation between two specifications is the weakest. (Only safety properties of B are preserved.) It is always used. The *well-formed refinement* relation is the strongest. It is seldom used.

Chandy and Misra [1988] defined the relation *A is a superposition of B*. In their approach, A is obtained from transforming B by repeated applications of two rules. This approach is attractive because the rules are syntactic and are thus very easy to use. But because the rules are syntactic, the class of specifications that can be derived by applying these rules is much smaller than the class that can be derived as well-formed refinements. Specifically, it is easy to see that if *A is a superposition of B* then *A is a well-formed refinement of B*. The converse does not hold.

While the relational notation and proof method in this paper are applicable to state transition systems in general, their development has been motivated primarily by protocol systems. The ideas and methods in this paper have been applied to the specification and verification of several nontrivial protocols, which are briefly described below.

The first application was the verification of a version of the High-level Data Link Control (HDLC) protocol standard with functions of connection management and full-duplex data transfer. Instead of verifying such a multifunction protocol in its entirety, smaller image protocols were obtained by projection and then verified [Shankar & Lam 83]. Properties of the multifunction protocol were inferred from properties of the image protocols.

Murphy and Shankar demonstrated how a complete transport protocol with functions of connection management and full-duplex data transfer can be composed from protocols specified for the individual functions. Because the multifunction protocol is a refinement of instances of the single-function protocols, safety properties of the single-function protocols are preserved in the multifunction protocol. Proofs of progress properties of the multifunction protocol were obtained in a hierarchical manner [Murphy & Shankar 87, Murphy & Shankar 88].

The well-formed image relation between specifications was also applied to the protocol conversion problem. Suppose a converter (translator) is interposed between two entities, say E_1 and E_2, that implement different communication protocols, say A_1 and A_2, respectively. Whenever, the converter receives an A_1 message sent by entity E_1, it translates the message to an A_2 message which is delivered to E_2. (The converter may delete the message instead of translating it.) Similarly, A_2 messages sent by E_2 are translated into A_1 messages which are delivered to E_1. The well-formed image relation was used to define what it means for a protocol converter to achieve interoperability between E_1 and E_2 [Calvert & Lam 90, Lam 88].

The theory presented in this paper has already been extended in several ways. We mention two of them below.

First, in deriving a specification A from specification B, we found that it is preferable to go through a succession of intermediate specifications, B_1, B_2, $\cdots$. To facilitate such a heuristic search, we defined a weaker form of the refinement relation, called *conditional refinement*. A stepwise refinement heuristic was developed based upon conditional refinement. The heuristic was applied to the specification of sliding window protocols for the transport layer where channels can lose, duplicate, and reorder messages, and the protocols use cyclic sequence numbers [Shankar 86, Shankar & Lam 87b]. It was also applied to the specification of connection management protocols for the transport layer [Murphy & Shankar 87, Murphy & Shankar 88].

Second, an extension to our theory herein is also needed to specify interfaces and implementations of program modules. To get simple conditions for composing modules, we impose a hierarchical relationship between modules that interact via an interface. The theory extension was to define what it means for a program module to *offer an upper interface* to a user, and to *use a lower interface* offered by another program module. It was applied to prove that specifications of two database implementations, based upon a two-phase locking protocol and a multi-version timestamp protocol, satisfy a serializable interface specification [Lam & Shankar 87].

Acknowledgements

This paper is an abbreviated version of [Lam & Shankar 88]. The final presentation of this paper has benefited greatly from the constructive criticisms and diligence of the anonymous referees of *IEEE Transactions on Software Engineering*, who reviewed [Lam & Shankar 88]. We are also grateful to Ken Calvert, Mohamed Gouda, Leslie Lamport, Jayadev Misra, Ambuj Singh, and Thomas Woo for their helpful comments.

REFERENCES

[Abadi & Lamport 88] M. Abadi and L. Lamport, "The Existence of Refinement Mappings," Technical Report, Digitial Systems Research Center, Palo Alto, California, August 1988.

[Calvert & Lam 90] K. L. Calvert and S. S. Lam, "Formal Methods for Protocol Conversion," to appear in *IEEE Journal on Selected Areas in Communications*, January 1990.

[Chandy & Misra 88] K.M. Chandy and J. Misra, *Parallel Program Design: A Foundation*, Addison-Wesley, Reading, MA, 1988.

[Hailpern & Owicki 83] B.T. Hailpern and S. Owicki, "Modular Verification of Computer Communication Protocols," *IEEE Transactions on Communications*, Vol. COM-31, No. 1, January 1983.

[Hehner 84] E.C.R Hehner, "Predicative Programming, Part I and Part II," *Communications of the ACM*, Vol. 27, No. 2, February 1984.

[IBM 80] IBM Corporation, Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic, IBM Form No. SC32-3112-2, 1980.

[ISO 85] ISO/TC97/SC21/WG16-1 N422 Estelle — A Formal Description Technique Based on an Extended State Transition Model, February 1985.

[Lam 88] S. S. Lam, "Protocol Conversion," *IEEE Transactions on Software Engineering*, Vol. 14, No. 3, March 1988.

[Lam & Shankar 84] S. S. Lam and A. U. Shankar, "Protocol Verification via Projections," *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 4, July 1984.

[Lam & Shankar 87] S. S. Lam and A. U. Shankar, "Specifying Implementations to Satisfy Interfaces:

A State Transition System Approach,'' presented at the 26th Lake Arrowhead Workshop on *How will we specify concurrent systems in the year 2000?*, September 1987; full version available as Technical Report TR-88-30, Department of Computer Sciences, University of Texas at Austin, August 1988 (revised June 1989).

[Lam & Shankar 88] S. S. Lam and A. U. Shankar, ``A Relational Notation for State Transition Systems,'' Technical Report TR-88-21, Department of Computer Sciences, The University of Texas at Austin, May 1988 (Second Revision, August 1989).

[Lamport 83a] L. Lamport, ``What Good is Temporal Logic?'' *Proceedings Information Processing 83*, IFIP, 1983.

[Lamport 83b] L. Lamport, ``Specifying Concurrent Program Modules,'' *ACM TOPLAS*, Vol. 5, No. 2, April 1983.

[Lamport 85] L. Lamport, ``What it means for a concurrent program to satisfy a specification: Why no one has specified priority,'' *Proceedings of the 12th ACM Symposium on Principles of Programming Languages*, New Orleans, January 1985.

[Lynch & Tuttle 87] N.A. Lynch and M.R. Tuttle, ``Hierarchical Correctness Proofs for Distributed Algorithms,'' *Proceedings of the ACM Symposium on Principles of Distributed Computing*, Vancouver, B.C., August 1987.

[Manna & Pnueli 84] Z. Manna and A. Pnueli, ``Adequate Proof Principles for Invariance and Liveness Properties of Concurrent Programs,'' *Science of Computer Programming*, Vol. 4, 1984.

[Manna & Waldinger 85] Z. Manna and R. Waldinger, *The Logical Basis for Computer Programming*, Addison-Wesley, Reading, MA, 1985.

[Murphy & Shankar 87] S.L. Murphy and A.U. Shankar, ``A Verified Connection Management Protocol for the Transport Layer,'' *Proceedings ACM SIGCOMM '87 Workshop*, Stowe, Vermont, August 1987.

[Murphy & Shankar 88] S.L. Murphy and A.U. Shankar, ``Service Specification and Protocol Construction for the Transport Layer,'' *Proceedings ACM SIGCOMM '88 Symposium*, Stanford University, August 1988.

[Owicki & Gries 76] S. Owicki and D. Gries, ``Verifying Properties of Parallel Programs: An Axiomatic Approach,'' *Communications of the ACM*, Vol. 19, No. 5, May 1976.

[Owicki & Lamport 82] S. Owicki and L. Lamport, ``Proving Liveness Properties of Concurrent Systems,'' *ACM TOPLAS*, Vol. 4, No. 3, 1982.

[Piatkowski 86] T. F. Piatkowski, "The State of The Art in Protocol Engineering," *Proceedings ACM Sigcomm '86 Symposium*, Stowe, Vermont, 1986.

[Pnueli 86] A. Pnueli, "Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends," in *Current Trends in Concurrency: Overviews and Tutorials*, J.W, deBakker et al. (ed.), LNCS 224, Springer Verlag, 1986.

[Sabnani 88] K. Sabnani, "An Algorithmic Procedure for Protocol Verification," *IEEE Transactions on Communications*, Vol. 36, No. 8, August 1988.

[Scheid & Holtsberg 88] J. Scheid and S. Holtsberg, *Ina Jo Specification Language Reference Manual*, System Development Group, Unisys Corp., Santa Monica, CA, September 1988.

[Shankar 86] A.U. Shankar, "Verified Data Transfer Protocols with Variable Flow Control," *ACM Transactions on Computer Systems*, Vol. 7, No. 3, August 1989; an abbreviated version appears in *Proceedings ACM SIGCOMM '86*, Stowe, Vermont, August 1986.

[Shankar & Lam 83] A.U. Shankar and S.S. Lam, "An HDLC Protocol Specification and its Verification Using Image Protocols," *ACM TOCS*, Vol. 1, No. 4, November 1983.

[Shankar & Lam 84] A.U. Shankar and S.S. Lam, "Time-dependent communication protocols," in *Tutorial: Principles of Communication and Networking Protocols*, S.S. Lam (ed.), IEEE Computer Society, 1984.

[Shankar & Lam 87a] A.U. Shankar and S.S. Lam, "Time-dependent distributed systems: proving safety, liveness, and real-time properties," *Distributed Computing*, Vol. 2, No. 2, 1987.

[Shankar & Lam 87b] A.U. Shankar and S.S. Lam, "A Stepwise Refinement Heuristic for Protocol Construction," Technical Report CS-TR-1812, Department of Computer Science, University of Maryland, March 1987 (revised March 1989).

[West 78] C.H. West, "A General Technique for Communications Protocol Validation," *IBM Journal of Research and Development*, Vol. 22, July 1978.