The Existence of Refinement Mappings

Martín Abadi and Leslie Lamport
Digital Equipment Corporation
Systems Research Center

Abstract

Refinement mappings are used to prove that a lower-level specification correctly implements a higher-level one. We consider specifications consisting of a state machine (which may be infinite-state) that specifies safety requirements and an arbitrary supplementary property that specifies liveness requirements. A refinement mapping from a lower-level specification $\mathbf{S_1}$ to a higher-level one $\mathbf{S_2}$ is a mapping from $\mathbf{S_1}$'s state space to $\mathbf{S_2}$'s state space that maps steps of $\mathbf{S_1}$'s state machine steps to steps of $\mathbf{S_2}$'s state machine and maps behaviors allowed by $\mathbf{S_1}$ to behaviors allowed by $\mathbf{S_2}$. We show that, under reasonable assumptions about the specifications, if $\mathbf{S_1}$ implements $\mathbf{S_2}$, then by adding auxiliary variables to $\mathbf{S_1}$ we can guarantee the existence of a refinement mapping. This provides a completeness result for a practical hierarchical specification method.

1 Introduction

1.1 Specifications

A system may be specified at many levels of abstraction, from a description of its highest-level properties to a description of its implementation in terms of microcode and circuitry. We address the problem of proving that a lower-level specification is a correct implementation of a higher-level one.

Unlike simple programs, which can be specified by input/output relations, complex systems can be adequately specified only by describing the behaviors that they may exhibit. We consider specification methods in which a behavior is represented by a sequence of states and a system is specified by a set of permitted behaviors.

A specification should describe only the *externally visible* components of a system's state. However, it is often helpful to describe its behavior in terms of unobservable *internal* components. For example, a natural way to specify a queue includes a description of the sequence of elements currently in the queue, and that sequence is not externally visible. Although internal components are mentioned, the specification prescribes the behavior of only the externally visible components. The system may exhibit the externally visible behavior

$$\langle\langle e_0,\ e_1,\ e_2,\ \dots\rangle\rangle$$

where e_i is a state of the externally visible component, if there exist states y_i of the internal component such that the *complete* behavior

$$\langle\langle (e_0,y_0),\ (e_1,y_1),\ (e_2,y_2),\ \dots\rangle\rangle$$

is permitted by the specification. (We use $\langle\langle\ \rangle\rangle$ to denote a sequence.)

A specification may allow steps in which only the internal state component changes—for example, a sequence

$$\langle\langle (e_0,y_0),\ (e_1,y_1),\ (e_1,y_1'),\ (e_1,y_1''),\ (e_2,y_2),\ \dots\rangle\rangle$$

Such internal steps are not externally visible, so the sequence of external states $\langle\langle e_0, e_1, e_1, e_1, e_2, \dots\rangle\rangle$ should be equivalent to the sequence $\langle\langle e_0, e_1, e_2, \dots\rangle\rangle$ obtained by removing the "stuttering" steps from e_1 to e_1. Let $\Gamma(\langle\langle e_0, e_1, \dots\rangle\rangle)$ be the set of all sequences obtained from $\langle\langle e_0, e_1, \dots\rangle\rangle$ by repeating states and deleting repeated states—that is, by adding and removing stuttering. We consider only specifications in which a sequence $\langle\langle e_0, e_1, \dots\rangle\rangle$ is allowed only if all sequences in $\Gamma(\langle\langle e_0, e_1, \dots\rangle\rangle)$ are allowed. Such specifications are said to be *invariant under stuttering.*

The behaviors permitted by a specification can be described as the set of sequences satisfying a safety and a liveness property [AS86, Lam77]. Intuitively, a safety property asserts that something bad does not happen and a liveness property asserts that something good does eventually happen. In specifying a queue, the safety property might assert that the sequence of elements removed from the queue is an initial prefix of the sequence of elements added to the queue. The liveness property might assert that an operation of putting an element into the queue is eventually completed if the queue is not full, and an operation of removing an element from the queue is eventually completed if the queue is not empty. (What operations are in progress and what elements they are adding to or have removed from the queue would be described by the externally visible state.)

We are concerned with specifications in which the safety property is described by an "abstract" nondeterministic program; a behavior satisfies the property if it can be generated by the program. Liveness conditions are described either directly by writing axioms or indirectly by placing fairness constraints on the abstract program. In a specification of a queue, the program describes the sequence of actions by which an element is added to or removed from the sequence of queued elements, ensuring the safety property that the correct elements are removed from the queue. Additional fairness constraints assert that certain actions must eventually occur, ensuring the liveness property that operations that should complete eventually do complete.

Many proposed specification methods involve writing programs and fairness conditions in this way [LS84, Lam83, LT87]. (Some methods do not consider liveness at all and just specify safety properties with programs.)

To describe specifications formally, we represent a program by a state machine (whose set of states may be infinite) and we represent the fairness constraints by an arbitrary *supplementary* condition. For our results, it does not matter if the supplementary condition specifies a liveness property.

1.2 Proving That One Specification Implements Another

A specification $\mathbf{S_1}$ implements a specification $\mathbf{S_2}$ if every externally visible behavior allowed by $\mathbf{S_1}$ is also allowed by $\mathbf{S_2}$. To prove that $\mathbf{S_1}$ implements $\mathbf{S_2}$, it suffices to prove that if $\mathbf{S_1}$ allows the behavior

$$\langle\langle (e_0, z_0),\ (e_1, z_1),\ (e_2, z_2),\ \ldots \rangle\rangle$$

where the z_i are internal states, then there exist internal states y_i such that $\mathbf{S_2}$ allows

$$\langle\langle (e_0, y_0),\ (e_1, y_1),\ (e_2, y_2),\ \ldots \rangle\rangle$$

In general, each y_i can depend upon the entire sequence $\langle\langle (e_0, z_0), (e_1, z_1), (e_2, z_2), \ldots \rangle\rangle$, and proving the existence of the y_i may be quite difficult. The proof is easier if each y_i depends only upon e_i and z_i, so there exists a function f such that $(e_i, y_i) = f(e_i, z_i)$. The proof becomes easier still if f maps steps of $\mathbf{S_1}$'s state machine into (possibly stuttering) steps of $\mathbf{S_2}$'s state machine. In this case, verifying that $\langle\langle f(e_0, z_0), f(e_1, z_1), f(e_2, z_2), \ldots \rangle\rangle$ satisfies the safety property of $\mathbf{S_2}$ involves reasoning about states and pairs of states, not about sequences. Such a mapping f is called a *refinement mapping*.

In the example of a queue, the internal state y_i of specification $\mathbf{S_2}$ might describe the sequence of elements currently in the queue, and the internal state z_i of specification $\mathbf{S_1}$ might describe the contents of an array that implements the queue. To prove that $\mathbf{S_1}$ implements $\mathbf{S_2}$, one would construct a refinement mapping f such that $f(e_i, z_i) = (e_i, y_i)$, where y_i describes the state of the queue that is represented by the contents of the array described by state z_i.

Several methods for proving that one specification implements another are based upon finding a refinement mapping [LS84, Lam83]. In practice, if $\mathbf{S_1}$ implements $\mathbf{S_2}$, then these methods usually can prove that the implementation is correct—usually, but not always. The methods fail if the refinement mapping does not exist. Three reasons why the mapping might not exist are:

- $\mathbf{S_2}$ may specify an internal state with "historical information" not needed by $\mathbf{S_1}$. For example, suppose $\mathbf{S_2}$ requires that the system display any arbitrary number of least-significant bits of a clock, so its internal state includes an unbounded clock value. This specification is implemented by a lower-level specification $\mathbf{S_1}$ that alternately displays zero and one, with no internal state. A refinement mapping does not exist because there is no way to define the internal state of the clock as a function of its low-order bit.

- $\mathbf{S_2}$ may specify that a nondeterministic choice is made before it has to be. For example, consider two specifications $\mathbf{S_1}$ and $\mathbf{S_2}$ for a system that displays ten nondeterministically chosen values in sequence. Suppose $\mathbf{S_2}$ requires that all values be chosen before any is displayed, while $\mathbf{S_1}$ requires each value to be chosen as it is displayed. Both specifications describe the same externally visible behaviors, so each implements the other. However, $\mathbf{S_2}$ requires the internal state to contain *all* ten values before any is displayed, while $\mathbf{S_1}$ does not specify any internal state, so no refinement mapping is possible.

- $\mathbf{S_2}$ may "run slower" than $\mathbf{S_1}$. For example, let $\mathbf{S_1}$ and $\mathbf{S_2}$ both specify clocks in which hours and minutes are externally visible and seconds are internal. Suppose that in $\mathbf{S_2}$ each step increments the clock by one second, while in $\mathbf{S_1}$ each step increments the clock by ten seconds. Both specifications allow the same externally visible behaviors. A complete behavior $\langle\langle s_0, s_1, s_2, \ldots \rangle\rangle$ specified by $\mathbf{S_1}$ may produce an externally visible change every six steps. For any mapping f, the sequence $\langle\langle f(s_0), f(s_1), f(s_2), \ldots \rangle\rangle$ may also produce an externally visible change every six steps. This is not allowed by $\mathbf{S_2}$, which requires fifty-nine internal steps for every externally visible one. Hence, no refinement mapping can prove that $\mathbf{S_1}$ implements $\mathbf{S_2}$.

If a refinement mapping does not exist, it can often be made to exist by adding *auxiliary variables* to the lower-level specification. An auxiliary variable is an internal state component that is added to a specification without affecting the externally visible behavior. The three situations described above in which refinement mappings cannot be found are handled as follows:

- Historical information missing from the internal state specified by $\mathbf{S_1}$ can be provided by adding a *history variable*—a well-known form of auxiliary variable that merely records past actions [Owi75].

- If $\mathbf{S_2}$ requires that a nondeterministic choice be made before it has to be, then $\mathbf{S_1}$ can be modified so the choice is made sooner by adding a *prophecy variable*. A prophecy variable is a new form of auxiliary variable that is the mirror image of a history variable—its formal definition is almost the same as the definition of a history variable with past and future interchanged, but there is an asymmetry due to behaviors having a beginning but not necessarily an end.

- If $\mathbf{S_2}$ runs slower than $\mathbf{S_1}$, then an auxiliary variable must be added to $\mathbf{S_1}$ to slow it down. We will define prophecy variables in such a way that they can perform this slowing.

Our main result states that, under three hypotheses about the specifications, if $\mathbf{S_1}$ implements $\mathbf{S_2}$ then one can add auxiliary history and prophecy variables to $\mathbf{S_1}$ to form an equivalent specification $\mathbf{S_1^{hp}}$ and find a refinement mapping from $\mathbf{S_1^{hp}}$ to $\mathbf{S_2}$. The three hypotheses, and their intuitive meanings, are:

$\mathbf{S_1}$ *is machine closed.* Machine closure means that the supplementary property (the one normally used to specify liveness requirements) does not specify any safety property not already specified by the state machine. In other words, the state machine does as much of the specifying as possible.

$\mathbf{S_2}$ *has finite invisible nondeterminism.* This denotes that, given any finite portion of an externally visible behavior allowed by $\mathbf{S_2}$, there are only a finite number of possible choices for its internal state component.

$\mathbf{S_2}$ *is internally continuous.* A specification is internally continuous if we can show that it does not allow a particular complete behavior by examining the behavior's externally visible part (which may be infinite) and some finite portion of the complete behavior.

We will show by examples why these three hypotheses are needed.

We will prove that any safety property has a specification with finite invisible nondeterminism, any specification of a safety property is internally continuous, and any property has a machine-closed specification. Therefore, our result implies that if the specifications are written properly and $\mathbf{S_2}$ specifies only a safety property then one can ensure that a refinement mapping exists. We will also show that, even when $\mathbf{S_2}$ is not internally continuous, a refinement mapping exists to show that $\mathbf{S_1}$ satisfies the safety property specified by $\mathbf{S_2}$. Therefore, by writing specifications properly, refinement mappings can always be used to prove the safety property of a specification if not its liveness property. We do not know if anything can be said about proving arbitrary liveness properties.

In this paper, proofs are just sketched. Detailed proofs as well as some additional discussion can be found in [AL88].

2 Preliminaries

2.1 Sequences

We now define some useful notations for sequences. In these definitions, σ denotes the sequence $\langle\langle s_0, s_1, s_2, \ldots\rangle\rangle$ and τ denotes the sequence $\langle\langle t_0, t_1, t_2, \ldots\rangle\rangle$. These sequences may be finite or infinite. If σ is finite, we let $\|\sigma\|$ denote its length and $last(\sigma)$ denote its last element, so $\|\langle\langle s_0, \ldots, s_{m-1}\rangle\rangle\| = m$ and $last(\langle\langle s_0, \ldots, s_{m-1}\rangle\rangle) = s_{m-1}$. An infinite sequence is said to be *terminating* if it is of the form $\langle\langle s_0, s_1, \ldots, s_n, s_n, s_n, \ldots\rangle\rangle$—in other words, if it reaches a final state that it repeats forever.

As usual, a mapping on elements is extended to a mapping on sequences of elements by defining $g(\sigma)$ to equal $\langle\langle g(s_0), g(s_1), \ldots\rangle\rangle$, and to a mapping on sets of elements by defining $g(S)$ to equal $\{g(s) : s \in S\}$.

The sequence σ is said to be *stutter-free* if, for each i, either $s_i \neq s_{i+1}$ or the sequence is infinite and $s_i = s_j$ for all $j \geq i$. Thus, a nonterminating sequence is stutter-free iff (if and only if) it never stutters, and a terminating sequence is stutter-free iff it stutters only after reaching its final state. We define $\natural\sigma$ to be the stutter-free sequence obtained by replacing every maximal finite subsequence $s_i, s_{i+1}, \ldots, s_j$ of identical elements with the single element s_i. For example,

$$\natural\langle\langle 0,1,1,2,2,2,3,3,3,3,4\ldots\rangle\rangle = \langle\langle 0,1,2,3,4,\ldots\rangle\rangle$$

We define $\sigma \simeq \tau$ to mean that $\natural\sigma = \natural\tau$, and we define $\Gamma\sigma$ to be the set $\{\tau : \tau \simeq \sigma\}$. If S is a set of sequences, $\Gamma(S)$ is the set $\{\Gamma\sigma : \sigma \in S\}$. A set of sequences S is *closed under stuttering* if $S = \Gamma(S)$. Thus, S is closed under stuttering iff for every pair of sequences σ, τ with $\sigma \simeq \tau$, if $\sigma \in P$ then $\tau \in P$.

We use "$\cdot$" to denote concatenation of sequences—that is, if $\|\sigma\| = m$, then $\sigma \cdot \tau = \langle\langle s_0, \ldots, s_{m-1}, t_0, t_1, \ldots\rangle\rangle$. If $\|\sigma\| \geq m$, we let $\sigma|_m$ denote $\langle\langle s_0, s_1, \ldots, s_{m-1}\rangle\rangle$.

For any set Σ, let Σ^ω denote the set of all infinite sequences of elements in Σ. An infinite sequence $\langle\langle \sigma_0, \sigma_1, \sigma_2, \ldots\rangle\rangle$ of sequences in Σ^ω is said to *converge* to the sequence σ in Σ^ω iff for all $m \geq 0$ there exists an $n \geq 0$ such that $\sigma_i|_m = \sigma|_m$ for all $i \geq n$. In this case, we define $\lim \sigma_i$ to be σ. This definition of convergence gives rise to a topology on Σ^ω. We now recall some other definitions.

Let σ be an element in Σ^ω and let S be a subset of Σ^ω. We say that σ is a *limit point* of S iff there exist elements σ_i in S such that $\lim \sigma_i = \sigma$. The set S is *closed* iff S contains all its limit points. The *closure* of S, denoted $\overline{S}$, consists of all limit points of S; it is the smallest closed set containing S.

2.2 Properties

We can only say that one specification implements another if we are given a correspondence between the externally visible states of the two specifications. For example, if $\mathbf{S_2}$ asserts that the initial value of a particular register is the integer -3 and $\mathbf{S_1}$ asserts that the register's initial value is the sequence of bits 1111100, then we can't say whether or not $\mathbf{S_1}$ implements $\mathbf{S_2}$ without knowing how to interpret a sequence of bits as an integer. In general, to decide if $\mathbf{S_1}$ implements $\mathbf{S_2}$, we must know how to interpret an externally visible state of $\mathbf{S_1}$ as an externally visible state of $\mathbf{S_2}$. Given such an interpretation, we can translate $\mathbf{S_1}$ into a specification with the same set of externally visible states as $\mathbf{S_2}$. Thus, there is no loss of generality in requiring that $\mathbf{S_1}$ and $\mathbf{S_2}$ have the same set of externally visible states.

We therefore assume that all specifications under consideration have the same fixed set Σ_E of externally visible states.

A *state space* Σ is a subset of $\Sigma_E \times \Sigma_I$ for some set Σ_I of internal states. We let Π_E be the obvious projection mapping from $\Sigma_E \times \Sigma_I$ onto Σ_E. The set Σ_E itself is considered to be a state space for which Π_E is the identity mapping.

If Σ is a state space, then a *Σ-behavior* is an element of Σ^ω. A Σ_E-behavior is called an *externally visible behavior.* A *Σ-property* P is a set of Σ-behaviors that is closed under stuttering. A Σ_E-property is called an *externally visible property.* If P is a Σ-property, then $\Pi_E(P)$ is a set of externally visible behaviors but is not necessarily an externally visible property because it need not be closed under stuttering. The externally visible property *induced* by a Σ-property P is defined to be the set $\Gamma(\Pi_E(P))$.

If Σ is clear from context or is irrelevant, we use the terms *behavior* and *property* instead of Σ-behavior and Σ-property. We sometimes add the adjective "complete", as in "complete behavior", to distinguish behaviors and properties from externally visible behaviors and properties.

A property P that is closed ($P = \overline{P}$) is called a *safety property.* Intuitively, a safety property is one asserting that something bad does not happen. To see that our formal definition of a safety property as a closed set captures this intuitive meaning, observe that if something bad happens, then it must happen within some finite period of time. Thus, P is a safety property iff, for any sequence σ not in P, one can tell that σ is not in P by looking at some finite prefix $\sigma|_i$ of σ. In other words, $\sigma \notin P$ iff there exists an i such that for all τ if $\tau|_i = \sigma|_i$ then $\tau \notin P$. Hence, $\sigma \in P$ iff for all i there exists a $\tau_i \in P$ such that $\tau_i|_i = \sigma|_i$. But $\lim \tau_i = \sigma$, which implies that $\sigma \in \overline{P}$; thus, $\sigma \in P$ iff $\sigma \in \overline{P}$. Therefore, P satisfies the intuitive definition of a safety property only if $P = \overline{P}$.

Even though we do not use the formal definition, it is interesting to note that a Σ-property L can be defined to be a liveness property iff it is dense in Σ^ω—in other words, if $\overline{L} = \Sigma^\omega$. This means that L is a liveness property iff any finite sequence of elements in Σ can be extended to a behavior in L. In a topological space, every set can be written as the intersection of a closed set and a dense set, so any property P can be written as $M \cap L$, where M is a safety property and L is a liveness property. Moreover, M can be taken to be $\overline{P}$.

2.3 Specifications

A *state machine* is a triple (Σ, F, N) where

- Σ is a state space. (Recall that this means $\Sigma \subseteq \Sigma_E \times \Sigma_I$ for some set Σ_I of internal states.)
- F, the set of *initial* states, is a subset of Σ.
- N, the *next-state relation*, is a subset of $\Sigma \times \Sigma$. (Elements of N are denoted by pairs of states enclosed in angle brackets, like $\langle s, t\rangle$.)

The *(complete) property generated by* a state machine (Σ, F, N) consists of all infinite sequences $\langle\langle s_0, s_1, \ldots\rangle\rangle$ such that $s_0 \in F$ and, for all $i \geq 0$, either $\langle s_i, s_{i+1}\rangle \in N$ or $s_i = s_{i+1}$. This set is closed under stuttering, so it is a Σ-property. The *externally visible property generated by* a state machine is the externally visible property induced by its complete property.

We now show that the complete property P generated by a state machine is a safety property. This requires proving that if $\lim \sigma_i = \sigma$ and each $\sigma_i \in P$, then $\sigma \in P$. For any behavior $\tau = \langle\langle s_0, s_1, \ldots\rangle\rangle$ and any $j \geq 0$, let τ^j be the terminating behavior $\langle\langle s_0, s_1, \ldots, s_j, s_j, s_j, \ldots\rangle\rangle$. Then τ is in P iff each τ^j is in P. Since $\lim \sigma_i = \sigma$, each σ^j equals $(\sigma_i)^j$ for some i. Since each σ_i is in P, each $(\sigma_i)^j$ is in P, which implies that σ is also in P. Hence, P is closed, so the complete property generated by a state machine is a safety property. However, we will show in Section 3 that its externally visible property need not be a safety property.

A state machine (Σ, F, N) is a familiar type of nondeterministic automaton, where F is the set of starting states and N describes the possible state transitions. (However, remember that Σ may be an infinite set.) The set of sequences generated (or accepted) by such an automaton is usually defined to be the set A of all sequences starting with a state in F and progressing by making transitions allowed by N. However, we also allow stuttering transitions, so we have defined the property generated by the state machine to be $\Gamma(A)$ together with all terminating sequences obtained from finite prefixes of behaviors in $\Gamma(A)$ by infinite stuttering.

A *specification* $\mathbf{S}$ is a four-tuple (Σ, F, N, L), where (Σ, F, N) is a state machine and L is a Σ-property, called the *supplementary property* of the specification. The property M generated by the state machine (Σ, F, N) is called the *machine property* of $\mathbf{S}$. The *(complete) property defined by* $\mathbf{S}$ is defined to be $M \cap L$, and the *externally visible property defined by* $\mathbf{S}$ is defined to be $\Gamma(\Pi_E(M \cap L))$, the externally visible property induced by $M \cap L$.

State machines are easier to work with than arbitrary sets of sequences, so one would like to specify a property purely in terms of state machines. However, the complete property generated by a state machine is a safety property. The supplementary property of a specification is needed to introduce liveness requirements. However, if we were to place no additional requirement on our specifications, we could use the supplementary property to do all the specifying. To see why this leads to trouble, let $\mathbf{S_2}$ be a specification consisting of any arbitrary state machine that generates an externally visible safety property O together with the trivial supplementary property that contains all behaviors. Define $\mathbf{S_1}$ to be the specification with state space Σ_E whose state machine is the trivial one that generates all Σ_E-behaviors and whose supplementary property is O. Obviously $\mathbf{S_1}$ implements $\mathbf{S_2}$. The existence of a refinement mapping from $\mathbf{S_1}$ to $\mathbf{S_2}$ implies that $\mathbf{S_1}$'s state machine implements $\mathbf{S_2}$'s state machine. However, $\mathbf{S_1}$ has the trivial state machine and no internal state. Auxiliary variables are added to a specification's state machine without affecting or being affected by the supplementary property. (This is what makes the addition of auxiliary variables practical.) No sound method of adding auxiliary variables can transform the trivial machine into one that implements a completely arbitrary state machine. Therefore, we need some constraint on the supplementary property.

In practice, we specify a desired property P by writing P as the intersection $M \cap L$ of a safety property M and a liveness property L. We try to construct L so that it does not specify any safety property, meaning that it does not rule out any finite behavior. More precisely, we try to choose L to be a liveness property such that any finite sequence of states generated by the state machine is the prefix of a behavior in P. For our results, it is not necessary that L be a liveness property; we need only require that L does not specify any safety property not implied by M. To express this requirement formally, we say that a specification $\mathbf{S}$ having machine property M and supplementary property L is *machine closed* iff $M = \overline{M \cap L}$.

The following lemma implies that, for a machine-closed specification, we can ignore the supplementary property and consider only the state machine when we are interested in finite portions of behaviors.

Lemma 1 *If $M = \overline{P}$, then every prefix of a behavior in M is the prefix of a behavior in P.*

Proof Given $\sigma \in M$ and $m > 0$, we must find $\tau \in P$ with $\sigma|_m = \tau|_m$. Choose $\sigma_i \in P$ with $\lim \sigma_i = \sigma$, choose n such that $\sigma_i|_m = \sigma|_m$ for all $i \geq n$, and let $\tau = \sigma_n$. ∎

The converse of this lemma is also true, but we will not need it.

2.4 Refinement Mappings

A specification $\mathbf{S_1}$ *implements* a specification $\mathbf{S_2}$ iff the externally visible property induced by $\mathbf{S_1}$ is a subset of the externally visible property induced by $\mathbf{S_2}$. In other words, $\mathbf{S_1}$ implements $\mathbf{S_2}$ iff every externally visible behavior allowed by $\mathbf{S_1}$ is also allowed by $\mathbf{S_2}$.

A *refinement mapping* from a specification $\mathbf{S_1} = (\Sigma_1, F_1, N_1, L_1)$ to a specification $\mathbf{S_2} = (\Sigma_2, F_2, N_2, L_2)$ is a mapping $f : \Sigma_1 \to \Sigma_2$ such that

R1. For all $s \in \Sigma_1$: $\Pi_E(f(s)) = \Pi_E(s)$. (f preserves the externally visible state component.)

R2. $f(F_1) \subseteq F_2$. (f takes initial states into initial states.)

R3. If $\langle s, t\rangle \in N_1$ then $\langle f(s), f(t)\rangle \in N_2$ or $f(s) = f(t)$. (A state transition allowed by N_1 is mapped by f into a [possibly stuttering] transition allowed by N_2.)

R4. $f(P_1) \subseteq L_2$, where P_1 is the property defined by $\mathbf{S_1}$. (f maps behaviors allowed by $\mathbf{S_1}$ into behaviors that satisfy $\mathbf{S_2}$'s supplementary property.)

Conditions R1–R3 are local, meaning that they can be checked by reasoning about states or pairs of states rather than about behaviors. Condition R4 is not local, but checking it is simplified by the fact that f is not an arbitrary mapping on sequences, but is obtained from a mapping on states. Thus, one can apply local methods like well-founded induction to prove R4.

Proposition 1 *If there exists a refinement mapping from $\mathbf{S_1}$ to $\mathbf{S_2}$, then $\mathbf{S_1}$ implements $\mathbf{S_2}$.*

Proof For $i = 1, 2$, let $\mathbf{S_i} = (\Sigma_i, F_i, N_i, L_i)$, let M_i be the machine property of $\mathbf{S_i}$, and let f be a refinement mapping from $\mathbf{S_1}$ to $\mathbf{S_2}$. Conditions R2 and R3 imply that $f(M_1) \subseteq M_2$, and R4 then implies $f(M_1 \cap L_1) \subseteq M_2 \cap L_2$. Condition R1 implies that $\Pi_E(M_1 \cap L_1) \subseteq \Pi_E(M_2 \cap L_2)$. ∎

3 Finite Invisible Nondeterminism

The machine property M of a specification is a safety property. However, the property that is really being specified by the specification's state machine is the externally visible property $\Gamma(\Pi_E(M))$ induced by M. The following example shows that this externally visible property is not necessarily a safety property.

Let Σ_E be the set $\mathbf{N}$ of natural numbers, and define the state machine (Σ, F, N) by:

- Σ equals $\Sigma_E \times \mathbf{N}$.
- F equals $\{(0,0)\}$.
- N is the union of the following two sets:
  - $\{\langle(0,0),(1,n)\rangle : n \in \mathbf{N}\}$,
  - $\{\langle(m,n+1),(m+1,n)\rangle : m,n \in \mathbf{N}\}$.

A stutter-free behavior of this machine starts in state $(0,0)$, goes to state $(1,n)$ for some arbitrary $n \geq 0$, then goes through the sequence of states $(2, n-1)$, $(3, n-2)$, ..., $(n-i+1, i)$ for some $i \geq 0$, and terminates (stutters forever) in the state $(n-i+1, i)$.

The set of externally visible behaviors induced by this state machine consists of all sequences obtainable by stuttering from a sequence σ_n of the form $\langle\langle 0,1,2,\ldots,n,n,n,\ldots\rangle\rangle$. This set is not closed, because $\lim \sigma_n = \langle\langle 0,1,2,3,\ldots\rangle\rangle$, and $\langle\langle 0,1,2,3,\ldots\rangle\rangle$ is not in the set. The externally visible property specified by the state machine is the conjunction of two properties:

1. The set of all behaviors that start in state 0 and change state only by adding 1 to the previous state.
2. The set of terminating behaviors.

The first property is a safety property, but the second is a liveness property; their intersection is neither a safety nor a liveness property.

The purpose of a specification is to specify an externally visible property. We feel that the externally visible property specified by state machine should be a safety property, so we want to restrict the class of allowed state machines.

The reason the externally visible property defined by the state machine in our example is not a safety property can be traced to the existence of infinitely many state transitions $\langle(0,0),(1,n)\rangle$ that correspond to the same externally visible transition $\langle 0,1\rangle$. It is this type of infinite invisible nondeterminism that allows the introduction of liveness into the externally visible property of a state machine. To ensure that a state machine specifies only safety properties, we must restrict it to having finite invisible nondeterminism.

Instead of defining the concept of finite invisible nondeterminism for a state machine, it is more general to define it for a property. A state machine is defined to have finite invisible nondeterminism iff the property it generates does.

Definition 1 *Let P be a property and O its induced externally visible property. We say that P is* fin *(for* finitely invisibly nondeterministic*) iff for all $\eta \in O$ and all $n \geq 0$ the set*

$$\{\natural(\sigma|_m) : (m > 0) \wedge (\sigma \in P) \wedge (\Pi_E(\natural(\sigma|_m)) \simeq \eta|_n)\}$$

is finite. (In other words, every finite prefix $\eta|_n$ of a behavior η in O is the externally visible part of only finitely many finite stutter-free prefixes $\natural(\sigma|_m)$ of behaviors in P.) *We say that a specification is fin iff the complete property of the specification is fin.*

If a property M is fin then every stronger property P is also fin. (Property P is stronger than property M iff $P \subseteq M$.) In our main result, instead of requiring that the state machine of $\mathbf{S_2}$ is fin, we make the weaker assumption that $\mathbf{S_2}$ is fin. This is strictly weaker only if $\mathbf{S_2}$ is not machine closed, since a machine-closed specification is fin iff its state machine is fin.

The following proposition asserts that the externally visible property of a fin state machine is a safety property. It is a simple corollary of the subsequent lemma, which will be used later as well.

Proposition 2 *If a safety property P is fin, then the externally visible property $\Gamma(\Pi_E(P))$ that it induces is also a safety property.*

Lemma 2 (Closure and nondeterminism) *Let property P be fin and let O be the externally visible property that it induces. If δ is a limit point of O then there is a limit point ρ of P such that $\Pi_E(\rho) \simeq \delta$.*

Proof Let Θ_n equal $\{\natural(\sigma|_m) : (m > 0) \wedge (\sigma \in P) \wedge (\Pi_E(\natural(\sigma|_m)) \simeq \delta|_n)\}$. Since P is fin, the set Θ_n is finite. The definitions of δ and O imply that Θ_n is nonempty. Let $\sigma \preceq \tau$ denote that σ is a prefix of τ. For all n and all $\theta \in \Theta_{n+1}$ there exists $\theta' \in \Theta_n$ such that $\theta' \preceq \theta$. König's Lemma [Knu73, pages 381–383] then implies that there is an infinite sequence $\rho_1 \preceq \rho_2 \preceq \rho_3 \preceq \ldots$ with each $\rho_i \in \Theta_i$. Extend the length of the ρ_i by stuttering if necessary so they keep getting longer, extend each ρ_i to a behavior $\rho'_i \in P$, and let $\rho = \lim \rho'_i$. ∎

For a state machine to be fin, it may not make an infinite nondeterministic choice unless all but a finite part of that choice is immediately revealed in the externally visible state. We can weaken our definition by requiring only that the choice eventually be revealed. Formally, this means defining a property P with induced externally visible property O to be fin iff for every η in O and $n \geq 0$ there exists an $n' \geq n$ such that the set

$$\{\natural(\sigma|_m) : (m > 0) \wedge (\sigma \in P) \wedge (\Pi_E(\natural(\sigma|_m)) \simeq \eta|_n) \\ \wedge\ \exists m' : (\Pi_E(\natural(\sigma|_{m'})) \simeq \eta|_{n'})\}$$

is finite. However, using this weaker definition of finite invisible nondeterminism would require somewhat more powerful prophecy variables and would complicate our proofs, so we will stick with our original definition.

4 Safety Properties

Alpern and Schneider [AS87] and others have observed in the finite-state case that there is a correspondence between state machines and externally visible safety properties. We extend their results to the infinite-state case for state machines with finite invisible nondeterminism. We also prove a result that allows us to apply our completeness theorem to safety properties even when the internal continuity hypothesis is not satisfied.

Proposition 2 implies that the externally visible property generated by a fin state machine is a safety property. We now prove the converse.

Proposition 3 *Every externally visible safety property can be generated by a state machine with finite invisible nondeterminism.*

Proof Given a closed Σ_E-property O, the required state machine machine is constructed by defining the set of states to consist of all pairs $(e_i, \langle\langle e_0, e_1, \ldots, e_i\rangle\rangle)$ such that $\langle\langle e_0, e_1, \ldots, e_i\rangle\rangle$ is a prefix of a sequence in O, defining the starting states to be ones with internal components of length one, and defining the next-state relation so the machine can go from state $(e_i, \langle\langle e_0, \ldots, e_i\rangle\rangle)$ only to state $(e_{i+1}, \langle\langle e_0, \ldots, e_i, e_{i+1}\rangle\rangle)$ for some e_{i+1}. ∎

If specification $\mathbf{S_2}$ is not internally continuous, it is possible for it to be implemented by a specification $\mathbf{S_1}$ without there being a refinement mapping from $\mathbf{S_1}$ to $\mathbf{S_2}$. (Internal continuity will be defined formally in Section 6.) However, since safety properties are internally continuous, we would expect to be able to prove that the externally visible machine property of $\mathbf{S_1}$ implements the externally visible machine property of $\mathbf{S_2}$. Combined with our main theorem, the following result shows that this is always possible if the state machine of $\mathbf{S_1}$ is machine closed and the machine property of $\mathbf{S_2}$ is fin.

Theorem 1 (Separate safety proofs) *Let $P_1 = M_1 \cap L_1$ and $P_2 = M_2 \cap L_2$, where the L_i are arbitrary properties and the M_i are safety properties; and let O_i and O_i^M be the externally visible properties induced by P_i and M_i, respectively. If $M_1 = \overline{P_1}$, M_2 is fin, and $O_1 \subseteq O_2$, then $O_1^M \subseteq O_2^M$.*

Proof For any set Q of behaviors, $\Gamma(\overline{Q}) \subseteq \overline{\Gamma(Q)}$ and $\Pi_E(\overline{Q}) \subseteq \overline{\Pi_E(Q)}$. From this and the hypothesis that $M_1 = \overline{P_1}$, we can prove that $O_1^M \subseteq \overline{O_1}$, which implies $O_1^M \subseteq \overline{O_2}$. The monotonicity of Π_E, Γ, and closure imply $\overline{O_2} \subseteq \overline{O_2^M}$, and Proposition 2 implies that $\overline{O_2^M} = O_2^M$. ▌

5 Auxiliary Variables

Although in practice refinement mappings usually exist, they do not always exist. To construct a refinement mapping, it may be necessary to add auxiliary variables. We now formally define two types of auxiliary variables: the well-known *history variable* and the new *prophecy variable*. These auxiliary variables are added to a specification's state machine; the supplementary property is essentially left unchanged.

5.1 History Variables

Adding a history variable means augmenting the state space with an additional component Σ_H and modifying the state machine in such a way that this additional component records past information but does not affect the behavior of the original state components. Formally, a specification $\mathbf{S}^h = (\Sigma^h, F^h, N^h, L^h)$ is said to be *obtained from* the specification $\mathbf{S} = (\Sigma, F, N, L)$ *by adding a history variable* iff the following five conditions are satisfied. In these conditions, we identify $(\Sigma_E \times \Sigma_I) \times \Sigma_H$ with $\Sigma_E \times (\Sigma_I \times \Sigma_H)$ (so H1 implies that Σ^h is a state space), and we let $\Pi_{[H]}$ be the obvious projection mapping from $\Sigma \times \Sigma_H$ onto Σ. (In the intuitive explanation, we say that a Σ^h-behavior σ *simulates* the Σ-behavior $\Pi_{[H]}(\sigma)$.)

H1. $\Sigma^h \subseteq \Sigma \times \Sigma_H$ for some set Σ_H.

H2. $\Pi_{[H]}(F^h) = F$. (A state in Σ is an initial state of $\mathbf{S}$ iff it is the first component of an initial state of $\mathbf{S}^h$.)

H3. If $\langle (s,h), (s',h') \rangle \in N^h$ then $\langle s, s' \rangle \in N$ or $s = s'$. (Every step of $\mathbf{S}^h$'s state machine simulates a [possibly stuttering] step of $\mathbf{S}$'s state machine.)

H4. If $\langle s, s' \rangle \in N$ and $(s,h) \in \Sigma^h$ then there exists $h' \in \Sigma_H$ such that $\langle (s,h), (s',h') \rangle \in N^h$. (From any state, $\mathbf{S}^h$'s state machine can simulate any possible step of $\mathbf{S}$'s state machine.)

H5. $L^h = \Pi_{[H]}^{-1}(L)$. (A Σ^h-behavior is in L^h iff the Σ-behavior that it simulates is in L.)

The following result shows that adding a history variable leaves an implementation essentially unchanged.

Proposition 4 (Soundness of history variables) *If* $\mathbf{S}^h$ *is obtained from* $\mathbf{S}$ *by adding a history variable, then the two specifications define the same externally visible property.*

Proof Let M and M^h be the specifications' machine properties, P and P^h their complete properties, and O and O^h the induced externally visible properties. Conditions H2 and H3 imply $\Pi_{[H]}(M^h) \subseteq M$ and condition H5 implies $\Pi_{[H]}(P^h) \subseteq P$, proving that $O^h \subseteq O$. To prove $O \subseteq O^h$, we must show that $P \subseteq \Pi_{[H]}(P^h)$, which is done by using H2–H4 to construct from any sequence $\langle\langle s_0, s_1, \ldots \rangle\rangle \in P$ a sequence $\langle\langle (s_0,h_0), (s_1,h_1), \ldots \rangle\rangle \in M^h$, and using H5 to show that this sequence is also in L^h. ▌

5.2 Simple Prophecy Variables

A prophecy variable is the dual of a history variable; its definition is almost that of a history variable with time running backwards. Intuitively, whereas a history variable records past behavior, a prophecy variable guesses future behavior. Using notation similar to that used in defining history variables, we define a specification $\mathbf{S}^p = (\Sigma^p, F^p, N^p, L^p)$ to be *obtained from* $\mathbf{S} = (\Sigma, F, N, L)$ *by adding a prophecy variable* iff the following conditions are satisfied. (Conditions P2′ and P4′ will be replaced in Section 5.3.)

P1. $\Sigma^p \subseteq \Sigma \times \Sigma_P$ for some set Σ_P.

P2′. $F^p = \Pi_{[P]}^{-1}(F)$. (This is the expected correspondence between the initial states of the two specifications.)

P3. If $\langle (s,p), (s',p') \rangle \in N^p$ then $\langle s, s' \rangle \in N$ or $s = s'$. (Every step of $\mathbf{S}^p$'s state machine simulates a [possibly stuttering] step of $\mathbf{S}$'s state machine.)

P4′. If $\langle s, s' \rangle \in N$ and $(s',p') \in \Sigma^p$ then there exists $p \in \Sigma_P$ such that $\langle (s,p), (s',p') \rangle \in N^p$. (From every state in Σ^p, the state machine of $\mathbf{S}^p$ can take a backwards step that simulates any possible backwards step of $\mathbf{S}$'s state machine. This is the time-reversed version of condition H4.)

P5. $L^p = \Pi_{[P]}^{-1}(L)$. (The supplementary property of $\mathbf{S}^p$ is the set of behaviors that simulate behaviors in the supplementary property of $\mathbf{S}$.)

P6. For all $s \in \Sigma$ the set $\Pi_{[P]}^{-1}(s)$ is finite and nonempty. (To every state of $\mathbf{S}$ there corresponds some nonzero finite number of states of $\mathbf{S}^p$.)

Condition P6 is the only one not corresponding to any condition for history variables. It is needed because time reversal is asymmetric—all behaviors have initial states but only terminating behaviors have final states. The second example below indicates why it is needed.

We now give two examples to illustrate the definition of prophecy variables. We mention only the state machines; the supplementary property can be taken to be the trivial one containing all behaviors.

For our first example, we take a state machine that nondeterministically generates an integer between 0 and 9. The machine counts up by one until it either decides to stop or else reaches 9, at which point it stutters forever. The set Σ_E of externally visible states is the set $\mathbf{N}$ of natural numbers, and the internal state component is a Boolean that becomes true when the final value is reached. (The Boolean values are written $\mathbf{t}$ and $\mathbf{f}$.)

- $\Sigma = \mathbf{N} \times \{\mathbf{t}, \mathbf{f}\}$.
- $F = \{(0, \mathbf{f})\}$.
- N is the union of the following two sets:
  - $\{\langle (i-1, \mathbf{f}), (i, \mathbf{f}) \rangle : 0 < i < 10\}$,
  - $\{\langle (i, \mathbf{f}), (i, \mathbf{t}) \rangle : i \in \mathbf{N}\}$.

The set of stutter-free behaviors generated by this state machine consists of all sequences of the forms

$$\langle\langle (0,\mathbf{f}), (1,\mathbf{f}), \ldots, (n,\mathbf{f}), (n,\mathbf{t}), (n,\mathbf{t}), (n,\mathbf{t}), \ldots \rangle\rangle$$

and

$$\langle\langle (0,\mathbf{f}), (1,\mathbf{f}), \ldots, (n,\mathbf{f}), (n,\mathbf{f}), (n,\mathbf{f}), \ldots \rangle\rangle$$

with $0 \leq n < 10$.

We now add a prophecy variable whose value is a natural number. This variable "predicts" how many more nonstuttering steps the state machine will take at most. The precise definition of the new state machine is:

- Σ^p is the union of the following two sets:
  - $\{(i, \mathbf{f}, j) : 0 \leq i,\ 0 \leq j, \text{ and } i + j < 10\}$,
  - $\{(i, \mathbf{t}, 0) : 0 \leq i < 10\}$.
- $F^p = \{(0, \mathbf{f}, j) \in \Sigma^p\}$.
- N^p is the union of the following two sets:
  - $\{\langle (i-1, \mathbf{f}, j+1), (i, \mathbf{f}, j) \rangle \in \Sigma^p \times \Sigma^p\}$,
  - $\{\langle (i, \mathbf{f}, 0), (i, \mathbf{t}, 0) \rangle \in \Sigma^p \times \Sigma^p\}$.

The reader can check that the conditions P1–P4′ and P6 given above are satisfied. (Condition P5 is satisfied if L and L^p are the trivial properties that contain all behaviors.) Observe that although condition P4 is satisfied, condition H4 is not. The state machine can take a backwards step from the state $(6, \mathbf{f}, 0)$ but not a forward step.

The state machine (Σ^p, F^p, N^p) is deterministic. The only stutter-free behaviors starting from the state $(0, \mathbf{f}, n)$ are of the forms

$$\langle\langle (0,\mathbf{f},n), (1,\mathbf{f},n-1), \ldots, (n,\mathbf{f},0), (n,\mathbf{t},0), (n,\mathbf{t},0), \ldots \rangle\rangle$$

and

$$\langle\langle (0,\mathbf{f},n), (1,\mathbf{f},n-1), \ldots, (i,\mathbf{f},n-i), (i,\mathbf{f},n-i), \ldots \rangle\rangle$$

with $0 \leq i \leq n$. The set of externally visible behaviors generated by the two state machines is the same; the stutter-free behaviors have the form $\langle\langle 0, 1, \ldots, n, n, n, \ldots \rangle\rangle$ for some n less than 10. State machine (Σ, F, N) decides nondeterministically when it is going to stop counting, while in state machine (Σ^p, F^p, N^p) this choice is made by the initial value of the prophecy variable.

As our second example, replace "10" by "∞" in the definitions of the two state machines. Conditions P1–P4′ still hold, but P6 does not; for each state $(i, \mathbf{f})$ of Σ there are an infinite number of states $(i, \mathbf{f}, j)$ in Σ^p. The externally visible stutter-free behaviors of (Σ^p, F^p, N^p) consist of sequences of the form $\langle\langle 0, 1, \ldots, n, n, n, \ldots \rangle\rangle$ for any natural number n. The state machine (Σ, F, N) generates all these behaviors plus the additional behavior $\langle\langle 0, 1, 2, 3, \ldots \rangle\rangle$ that never terminates. Because the finiteness condition P6 is not satisfied, adding the auxiliary variable changed the specification by ruling out this nonterminating behavior—effectively adding a liveness condition.

We can use our last example to indicate why we need the hypothesis of finite invisible nondeterminism for our completeness result. Let $\mathbf{S_2}$ be the specification consisting of the state machine (Σ^p, F^p, N^p) we just constructed (the one with "10" replaced by "∞") and the trivial supplementary property containing all Σ^p-behaviors. Let $\mathbf{S_1}$ be the specification with state machine (Σ, F, N) and supplementary property L consisting of all terminating behaviors. Both specifications define the same set of externally visible behaviors—all behaviors obtainable by stuttering from ones of the form $\langle\langle 0, 1, \ldots, n, n, n \rangle\rangle$. To construct a refinement mapping, we would have to add to $\mathbf{S_1}$ a prophecy variable that "guesses" the value of the last component of a state of Σ^p. However, no such prophecy variable can be constructed that satisfies P6, since for any starting state of $\mathbf{S_1}$ there are an infinite number of corresponding starting states of $\mathbf{S_2}$.

The complete property P_2 defined by this specification $\mathbf{S_2}$ is a safety property, and we will see that this implies that $\mathbf{S_2}$ is internally continuous. Moreover, specification $\mathbf{S_1}$ is machine closed. Nevertheless, adding auxiliary variables to $\mathbf{S_1}$ will not allow us to construct a refinement mapping to prove that it implements $\mathbf{S_2}$. Our completeness theorem does not apply because P_2 is not fin.

In this example, the prophecy variable we wanted to add would not satisfy P6. However, the supplementary property happened to ensure that adding the prophecy variable did not change the externally visible behavior. If we were to replace P6 by the weaker requirement that $\mathbf{S^P}$ have the same externally visible property as $\mathbf{S}$, then we could find a refinement mapping. However, this requirement is precisely what we had to prove in the first place—namely, that $\mathbf{S_1}$ implements $\mathbf{S_2}$.

5.3 Prophecy Variables That Add Stuttering

We now generalize our definition of a prophecy variable to allow it to introduce stuttering. Condition P2′ asserts that a state $(s, p) \in \Sigma^p$ is an initial state of $\mathbf{S^P}$'s state machine iff s is an initial state of $\mathbf{S}$'s state machine. We relax this condition by requiring only that such a state (s, p) be reachable from an initial state by steps that simulate stuttering steps. Formally, we replace P2′ by:

P2. (a) $\Pi_{[P]}(F^p) \subseteq F$.

(b) For all $(s,p) \in \Pi_{[P]}^{-1}(F)$ there exist $p_0, p_1, \ldots, p_n = p$ such that $(s, p_0) \in F^p$ and, for $0 \leq i < n$, $\langle (s, p_i), (s, p_{i+1}) \rangle \in N^p$.

Similarly, we relax condition P4′ by allowing $\mathbf{S^p}$'s state machine to simulate the step in **S**'s state machine from state s to state s' by a sequence of $n+1$ steps, the last n of which simulate stuttering steps. The precise condition that replaces P4′ is:

P4. If $\langle s, s' \rangle \in N$ and $(s', p') \in \Sigma^p$ then there exist p, $p'_0, \ldots, p'_{n-1}$, $p'_n = p'$ such that $\langle (s,p), (s', p'_0) \rangle \in N^p$ and, for $0 \leq i < n$, $\langle (s', p'_i), (s', p'_{i+1}) \rangle \in N^p$.

As with history variables, the addition of prophecy variables leaves an implementation essentially unchanged.

Proposition 5 (Soundness of prophecy variables) *If $\mathbf{S^p}$ is obtained from **S** by adding a prophecy variable, then the two specifications define the same externally visible property.*

Proof Let M and M^p be the specifications' machine properties, P and P^p their complete properties, and O and O^p the induced externally-visible properties. The proof that $O^p \subseteq O$ is identical to the proof of the corresponding condition for history variables in Proposition 4. To prove that $O \subseteq O^p$, we must prove that $P \subseteq \Pi_{[P]}(P^p)$. Given $\sigma = \langle\langle s_0, s_1, \ldots \rangle\rangle \in P$, we find $\tau \in P^p$ with $\Pi_{[P]}(\tau) \simeq \sigma$ as follows. Define a graph whose set of nodes is $\Sigma^p \times \mathbf{N}$ with an edge between $((s_i, p), i)$ and $((s_j, p'), i+1)$ iff $(s_i, p) = (s_{i+1}, p')$ or there exist $p_0, p_1, \ldots, p_n = p' \in \Sigma_P$ such that $\langle (s_i, p), (s_{i+1}, p_0) \rangle \in N^p$ and, for all $0 \leq k < n$, $\langle (s_{i+1}, p_k), (s_{i+1}, p_{k+1}) \rangle \in N^p$. The subgraph reachable from nodes of the form $((s_0, p), 0)$ is acyclic, and P6 implies that it has finite branching and a finite set of sources. An induction proof based on P4 implies that for all $n \geq 0$ and all $(s_n, p_n) \in \Sigma^p$ there exist elements $p_0, \ldots, p_{n-1}$ in Σ_P such that $\langle\langle ((s_0, p_0), 0), \ldots, ((s_n, p_n), n) \rangle\rangle$ is a path in this graph. König's Lemma implies the existence of an infinite path $\langle\langle ((s_0, p_0), 0), ((s_1, p_1), 1), \ldots \rangle\rangle$. Let $\rho = \langle\langle (s_0, p_0), (s_1, p_2), \ldots \rangle\rangle$, and construct τ from ρ by using the definition of the graph to fill in the internal steps of $\mathbf{S^p}$ between each (s_i, p_i) and (s_{i+1}, p_{i+1}), and using P2 to fill in the steps at the beginning. The construction guarantees that $\tau \in M^p$, and P5 implies that $\tau \in L^p$. ▮

6 Internal Continuity

We now define internal continuity, which appears in our third hypothesis. But first, we give an example that indicates why the hypothesis is needed for our completeness theorem.

Let $\Sigma_E = \mathbf{N}$, let η_i be the terminating sequence $\langle\langle 0, 1, \ldots, i, i, i, \ldots \rangle\rangle$, and let η be the nonterminating sequence $\langle\langle 0, 1, 2, \ldots \rangle\rangle$. Let $\langle\langle e_0, e_1, \ldots \rangle\rangle \times x$ denote the sequence $\langle\langle (e_0, x), (e_1, x), \ldots \rangle\rangle$. We construct a specification $\mathbf{S_2}$ that defines the property whose stutter-free sequences consist of all sequences $\eta_i \times \mathbf{t}$ together with the sequence $\eta \times \mathbf{f}$. Formally, $\mathbf{S_2} = (\Sigma_2, F_2, N_2, L_2)$, where

- $\Sigma_2 = \mathbf{N} \times \{\mathbf{t}, \mathbf{f}\}$. (The internal component is a Boolean.)
- $F_2 = \{(0, \mathbf{t}), (0, \mathbf{f})\}$. (Behaviors start with their visible components equal to 0.)
- $N_2 = \{\langle (i, b), (i+1, b) \rangle\}$. (The external component is incremented by 1 and the internal component remains constant.)
- L_2 consists of all behaviors *except* ones of the form $\sigma \times \mathbf{f}$ with σ terminating and $\sigma \times \mathbf{t}$ with σ nonterminating.

The externally visible property O_2 defined by $\mathbf{S_2}$ consists of the behaviors η_i, the behavior η, and all behaviors obtained from them by stuttering. Specification $\mathbf{S_2}$ is fin and machine closed.

The externally visible property O_2 is also defined by the simpler specification $\mathbf{S_1} = (\Sigma_1, F_1, N_1, L_1)$, where

- $\Sigma_1 = \Sigma_E = \mathbf{N}$. (There is no internal component.)
- $F_1 = \{0\}$. (All behaviors start at 0.)
- $N_1 = \{\langle i, i+1 \rangle\}$. (The state is incremented by 1.)
- $L_1 = \Sigma_1^\omega$ (the trivial property that allows all behaviors).

Obviously, $\mathbf{S_1}$ implements $\mathbf{S_2}$. Let $\mathbf{S_1^p} = (\Sigma_1^p, F_1^p, N_1^p, L_1^p)$ be any specification obtained from $\mathbf{S_1}$ by adding a prophecy variable. We now show that there does not exist a refinement mapping from $\mathbf{S_1^p}$ to $\mathbf{S_2}$; in fact there does not exist any mapping from Σ_1^p to Σ_2 that proves that $\mathbf{S_1^p}$ implements $\mathbf{S_2}$.

Let P_1^p be the property defined by $\mathbf{S_1^p}$. We show by contradiction that there does not exist any mapping $f : \Sigma_1^p \to \Sigma_2$ such that (i) $\Pi_E(f(i,p)) = i$ and (ii) $f(P_1^p) \subseteq P_2$. For each i let $\eta'_i \in P_1^p$ be a behavior with $\Pi_{[P]}(\eta'_i) \simeq \eta_i$. Moreover, P5 implies that we can choose η'_i to have no repeated nonfinal states, meaning that for $j < i$ and $k > 1$, there is no segment $\langle\langle (j, p_1), (j, p_2), \ldots, (j, p_k) \rangle\rangle$ of η'_i with $p_1 = p_k$. By (i), we then have that for every i and m with $i < m$ there is an l such that $\Pi_E(\eta'_m|_l) \simeq \eta_i|_{i+1}$. Moreover, P6 and the absence of repeated nonfinal states imply that for each i there is an integer $\pi(i) > i$ such that $l \leq \pi(i)$ for all such m. We can choose π so that $\pi(i+1) \geq \pi(i)$ for all i.

For any n, the set $\{\eta'_j|_{\pi(n)}\}$ is finite (by P6). Therefore, we can inductively construct the sequence θ_n of length $\pi(n)$ such that θ_n is a prefix of infinitely many of the η'_j and is also a prefix of θ_{n+1}. Let $\eta' = \lim \theta_n$; then $\Pi_E(\eta') \simeq \eta$. Since each θ_n is a prefix of some η'_j, clearly η' is in the machine property of $\mathbf{S_1^p}$. Property P5 then implies that $\eta' \in P_1^p$. By definition of η'_i, assumption (ii) implies that $f(\eta'_i) \simeq \eta_i \times \mathbf{t}$, which implies that $f(\eta') \simeq \eta \times \mathbf{t}$. We then have $\eta' \in P_1^p$ and $f(\eta') \notin P_2$, which contradicts assumption (ii).

This proof can be extended to the case where $\mathbf{S_1}$ is replaced by any specification $\mathbf{S_1^h}$ obtained from it by adding a history variable. We just replace η with any behavior allowed by $\mathbf{S_1^h}$ that simulates it, and replace η_i with an initial prefix of this new η. Thus, first adding a history variable still does not allow one to construct the refinement mapping.

The problem with specification $\mathbf{S_2}$ is that $\eta \times \mathbf{t}$ is not in P_2 even though $\Pi_E(\eta \times \mathbf{t})$ is in O_2 and any finite portion of $\eta \times \mathbf{t}$ is the same as the corresponding portion of some behavior $\eta_i \times \mathbf{t}$ in P_2. The sequence $\eta \times \mathbf{t}$ is not in P_2 even though we cannot tell that it isn't by looking either at its externally visible component or at any finite part of the complete behavior. To rule out this possibility, we must add to our completeness theorem the hypothesis that P_2 is *internally continuous.*

Definition 2 *A Σ-property P with induced externally visible property O is* internally continuous *iff, for any Σ-behavior σ, if $\Pi_E(\sigma) \in O$ and $\sigma \in \overline{P}$, then $\sigma \in P$. A specification is internally continuous iff the (complete) property it defines is internally continuous.*

Suppose $P = M \cap L$ and $M = \overline{P}$. Then $\lim \sigma_i = \sigma$ for $\sigma_i \in P$ iff $\sigma \in M$. It follows from this that, for a machine-closed specification, internal continuity is equivalent to the condition that a complete behavior is allowed iff it is generated by the state machine and its externally visible component is allowed. In particular, safety properties are internally continuous.

Since the machine property M is closed, if $\lim \sigma_i = \sigma$ for $\sigma_i \in M \cap L$, then $\sigma \in L$ iff $\sigma \in M \cap L$. This implies that if L is internally continuous, then $M \cap L$ is internally continuous. Hence, for any specification, if the supplementary property is internally continuous, then the specification is internally continuous. The converse is not true, since if M is the empty property, then $M \cap L$ is internally continuous for any L.

Any specification can be made internally continuous by adding to L all sequences σ in M such that $\Pi_E(\sigma) \in O$. Expanding L in this way obviously adds no new externally visible behaviors, so the resulting specification is equivalent to the original one. The expansion could introduce infinite internal nondeterminism, but not if M is fin.

7 Completeness Results

We can now prove our main result.

Theorem 2 (Completeness) *If the machine-closed specification $\mathbf{S_1}$ implements the internally continuous, fin specification $\mathbf{S_2}$, then there is a specification $\mathbf{S_1^h}$ obtained from $\mathbf{S_1}$ by adding a history variable and a specification $\mathbf{S_1^{hp}}$ obtained from $\mathbf{S_1^h}$ by adding a prophecy variable such that there exists a refinement mapping from $\mathbf{S_1^{hp}}$ to $\mathbf{S_2}$.*

Proof For $i = 1, 2$, let $\mathbf{S_i} = (\Sigma_i, F_i, N_i, L_i)$, and let P_i be the complete property of $\mathbf{S_i}$. Define $\mathbf{S_1^h} = (\Sigma_1^h, F_1^h, N_1^h, L_1^h)$, where: Σ_1^h is the set of pairs (s, h) such that h is a finite prefix of a behavior in P_1 ending in s; F_1^h is the set of states with $\|h\| = 1$; $\langle (s,h), (s',h') \rangle \in N_1^h$ iff $h' = h \cdot \langle\langle s' \rangle\rangle$; and L_1^h is determined by H5. Conditions H2 and H4 follow from the machine closure of $\mathbf{S_1}$ and Lemma 1, and the other conditions are proved using only the definition of $\mathbf{S_1^h}$. Define $\mathbf{S_1^{hp}} = (\Sigma_1^{hp}, F_1^{hp}, N_1^{hp}, L_1^{hp})$, where: Σ_1^{hp} is the set of triples (s,h,p) such that p is an initial prefix of a stutter-free behavior in P_2 such that p has the same externally visible behavior as h; F_1^{hp} is the set of states with $(s,h) \in F_1^h$ and $\|p\| = 1$; $\langle (s,h,p), (s',h',p') \rangle \in N_1^{hp}$ iff either (a) $p' = p \cdot \langle\langle \mathit{last}(p') \rangle\rangle$ and either $\langle (s,h), (s',h') \rangle \in N_1^h$ or $(s,h) = (s',h')$, or (b) $p' = p$ and $\langle (s,h),(s',h') \rangle \in N_1^h$; and L_2^{hp} is determined by P5. Conditions P1–P5 follow from the definition of $\mathbf{S_1^{hp}}$, and the proof of P6 requires also the hypotheses that $\mathbf{S_1}$ implements $\mathbf{S_2}$ and that $\mathbf{S_2}$ is fin. Define $f : \Sigma_1^{hp} \to \Sigma_2$ by $f((s,h,p)) = \mathit{last}(p)$. Then R1–R3 follow from the definition of $\mathbf{S_1^{hp}}$ and R4 follows from the hypotheses that $\mathbf{S_1}$ implements $\mathbf{S_2}$ and $\mathbf{S_2}$ is internally continuous. ▮

The converse of this completeness theorem is not true. For instance, no matter how pathological a specification is, we can use the identity refinement mapping to prove that it implements itself.

The hypotheses of the internal continuity and finite invisible nondeterminism of $\mathbf{S_2}$ can be removed from our completeness result by generalizing the definition of a prophecy variable—namely, by replacing condition P6 with the explicit requirement that the externally visible behaviors of $\mathbf{S^p}$ be the same a those of $\mathbf{S}$. This result is proved by defining $\mathbf{S_1^h}$ as in the proof of Theorem 2, defining $\mathbf{S_1^{hp}}$ so that its states are 4-tuples (s,h,n,τ) with $(s,h) \in \Sigma_1^h$, τ a complete behavior allowed by $\mathbf{S_2}$, and $\Pi_E(h) \simeq \Pi_E(\tau|_n)$, and defining the refinement mapping by letting $f((s,h,n,\tau))$ be the n^{th} element of τ. However, the condition that replaces P6 asserts that specification $\mathbf{S^p}$ implements $\mathbf{S}$, which is precisely the type of condition we are trying to prove in the first place. This generalization of Theorem 2 is therefore of little practical value, so we will not bother to state it and prove it formally.

There is one simple way to strengthen the completeness theorem that is of some interest. The specification $\mathbf{S_2}$ is fin and internally continuous iff the property P_2 that it defines is fin and internally continuous. We can weaken the hypothesis by requiring only that there exist a fin and internally continuous property P_2' contained in P_2 that induces the same externally-visible property as P_2. Let $\mathbf{S_2'}$ be the specification obtained from $\mathbf{S_2}$ by replacing L_2 with $L_2 \cap P_2'$. The correctness of this result follows easily from Theorem 2 by replacing $\mathbf{S_2}$ with $\mathbf{S_2'}$.

8 Whence and Whither?

Refinement mappings are not new. They form the basis of the methods advocated by Lam and Shankar [LS84] and by us [Lam83], and they are used by Lynch and Tuttle [LT87] to prove that one automaton implements another. However, none of this work addresses the issue of completeness. Jonsson [Jon87] did prove a completeness result similar to ours, but for a smaller class of specifications.

Complete methods for checking that a program implements a specification, without constructing refinement mappings, have been developed. Some of the most general are those of Alpern and Schneider [AS87], Manna and Pnueli [MP87], and Vardi [Var87]. Their methods differ from our approach in at least two important ways:

- They do not consider behaviors with different amounts of "stuttering" to be equivalent, so their definition of what constitutes a correct implementation is weaker than ours.

- They require taking the negation of specifications. In practice, the negation of a specification may be hard to construct and hard to understand.

Because of these differences, the methods may not offer practical alternatives to the use of refinement mappings for proving correctness.

Our exposition has been purely semantic. We have considered specifications, but not the languages in which they are expressed. We proved the existence of refinement mappings, but said nothing about whether they are expressible in any language. We do not know what languages can describe the necessary auxiliary variables and resulting refinement mappings.

Our results also raise the question of what properties can be described by specifications that are fin and internally continuous. If the specification language is expressive enough, then all properties can be defined by specifications without internal state, which are trivially fin and internally continuous. At the other extreme, one can easily invent artificially impoverished languages that do not allow any fin or internally continuous specifications. The question becomes interesting only for interesting specification languages, such as various forms of temporal logic. In addition, recall that the hypotheses of our completeness result can be weakened by requiring only that $\mathbf{S}_2$'s complete property be equivalent to a fin and internally continuous subproperty. This raises the more general question of what expressible properties have equivalent fin and continuous subproperties.

Acknowledgements

The first example we saw that demonstrated the inadequacy of history variables is due to Herlihy and Wing [HW87]. The introduction of prophecy variables was based on a suggestion of Jim Saxe. We also wish to thank Pierre Wolper for making clear whence our ideas came and Gordon Plotkin for making clear that whither they will lead is not an easy question, since he couldn't answer it on the spot.

References

[AL88] Martín Abadi and Leslie Lamport. *The Existence of Refinement Mappings.* Research Report SRC28, Digital Equipment Corporation, Systems Research Center, April 1988.

[AS86] Bowen Alpern and Fred B. Schneider. *Recognizing Safety and Liveness.* Technical Report TR86-727, Department of Computer Science, Cornell University, January 1986.

[AS87] Bowen Alpern and Fred Schneider. Proving boolean combinations of deterministic properties. In *Proceedings of the Second Symposium on Logic in Computer Science*, pages 131–137, IEEE, June 1987.

[HW87] M.P. Herlihy and J.M. Wing. Axioms for concurrent objects. In *Proceedings of the Fourteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 13–26, ACM, Munich, January 1987.

[Jon87] Bengt Jonsson. *Compositional Verification of Distributed Systems.* PhD thesis, Uppsala University, 1987.

[Knu73] Donald E. Knuth. *Fundamental Algorithms.* Volume 1 of *The Art of Computer Programming*, Addison-Wesley, Reading, Massachusetts, second edition, 1973.

[LS84] Simon S. Lam and A. Udaya Shankar. Protocol verification via projections. *IEEE Transactions on Software Engineering*, SE-10(4):325–342, July 1984.

[Lam77] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, March 1977.

[Lam83] Leslie Lamport. Specifying concurrent program modules. *ACM Transactions on Programming Languages and Systems*, 5(2):190–222, April 1983.

[LT87] Nancy Lynch and Mark Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the Sixth Symposium on the Principles of Distributed Computing*, pages 137–151, ACM, August 1987.

[MP87] Zohar Manna and Amir Pnueli. Specification and verification of concurrent programs by $\forall$-automata. In *Proceedings of the Fourteenth Symposium on the Principles of Programming Languages*, pages 1–12, ACM, January 1987.

[Owi75] S. Owicki. *Axiomatic Proof Techniques for Parallel Programs.* PhD thesis, Cornell University, August 1975.

[Var87] Moshe Vardi. Verification of concurrent programs: the automata-theoretic framework. In *Proceedings of the Second Symposium on Logic in Computer Science*, pages 167–176, IEEE, June 1987.