66

Composing Specifications

Martín Abadi and Leslie Lamport

October 10, 1990

Systems Research Center

DEC's business and technology objectives require a strong research program. The Systems Research Center (SRC) and three other research laboratories are committed to filling that need.

SRC began recruiting its first research scientists in 1984—their charter, to advance the state of knowledge in all aspects of computer systems research. Our current work includes exploring high-performance personal computing, distributed computing, programming environments, system modelling techniques, specification technology, and tightly-coupled multiprocessors.

Our approach to both hardware and software research is to create and use real systems so that we can investigate their properties fully. Complex systems cannot be evaluated solely in the abstract. Based on this belief, our strategy is to demonstrate the technical and practical feasibility of our ideas by building prototypes and using them as daily tools. The experience we gain is useful in the short term in enabling us to refine our designs, and invaluable in the long term in helping us to advance the state of knowledge about those systems. Most of the major advances in information systems have come through this strategy, including time-sharing, the ArpaNet, and distributed personal computing.

SRC also performs work of a more mathematical flavor which complements our systems research. Some of this work is in established fields of theoretical computer science, such as the analysis of algorithms, computational geometry, and logics of programming. The rest of this work explores new ground motivated by problems that arise in our systems research.

DEC has a strong commitment to communicating the results and experience gained through pursuing these activities. The Company values the improved understanding that comes with exposing and testing our ideas within the research community. SRC will therefore report results in conferences, in professional journals, and in our research report series. We will seek users for our prototype systems among those with whom we have common research interests, and we will encourage collaboration with university researchers.

Robert W. Taylor, Director

Composing Specifications

Martín Abadi and Leslie Lamport

October 10, 1990

A preliminary version of this report appeared in the proceedings of the REX Workshop on Stepwise Refinement of Distributed Systems, held at Mook, the Netherlands, in May 1989 [dBdRR90].

Authors' Abstract

A rigorous modular specification method requires a proof rule asserting that if each component behaves correctly in isolation, then it behaves correctly in concert with other components. Such a rule is subtle because a component need behave correctly only when its environment does, and each component is part of the others' environments. We examine the precise distinction between a system and its environment, and provide the requisite proof rule when modules are specified with safety and liveness properties.

Contents

1 Introduction **1**

1.1 States versus Actions 6
1.2 System versus Environment 7
1.3 Specifying the System and its Environment 8
1.4 Composition and Proof 9
1.5 Semantics versus Logic 10

2 The Semantic Model **10**

3 Realizability **14**

3.1 Safety Properties 14
3.2 Realizability of Arbitrary Properties 15
3.2.1 Definitions 15
3.2.2 Discussion of the Definitions 17
3.2.3 Some Basic Propositions 18

4 The Form of a Specification **19**

4.1 The Form of a Complete Program 20
4.1.1 The Parts of a Complete Program 20
4.1.2 The Progress Property 21
4.2 The Form of a Partial Program 23
4.2.1 The Parts of a Partial Program 23
4.2.2 Hiding the Internal State 26
4.3 The Normal Form of a Specification 27
4.4 An Overly Normal Form 30

5 Composing Specifications **31**

5.1 The Composition of Specifications 31
5.1.1 Assumptions about the States 32
5.1.2 Assumptions about the Agents 33
5.2 Implementing One Specification by Another 34
5.2.1 Definition 34
5.2.2 Proving That One Specification Implements Another . 35
5.3 The Main Theorem 38
5.3.1 A Precise Statement of the Composition Principle . . 38
5.3.2 The Hypotheses of the Theorem and Proposition . . . 41
5.3.3 The Hypotheses of the Proof Rule 43

6 Concluding Remarks **46**

Appendix: Proofs **47**
Proposition 1 51
Proposition 2 56
Proposition 3 56
Proposition 4 57
Proposition 5 57
Proposition 6 59
Proposition 7 61
Proposition 8 64
Proposition 9 65
Proposition 10 67
Theorem 1 69
Corollary 72
Proposition 11 72
Proposition 12 74
Theorem 2 75

Acknowledgements **79**

Glossary **81**

References **85**

Index **89**

1 Introduction

In the transition-axiom method, concurrent systems are specified by combining abstract programs and temporal logic [Lam89]. The method permits a hierarchical approach in which the composition of lower-level specifications is proved to implement a higher-level specification. In [AL91], we described how to prove that one specification implements another. Here, we examine how to compose specifications. We work at the semantic level, independent of any particular specification language or logic. Thus, our results can be applied to a number of approaches besides the transition-axiom method—for example, to Lam and Shankar's method of projections [LS84a], and to the I/O automata of Lynch and Tuttle [LT87].

Composition makes sense only for systems that interact with their environments. Such a system will behave properly only if its environment does. A Pascal program may behave quite improperly if a `read(x)` statement receives from the I/O system a value not allowed by the type of `x`. A circuit may exhibit bizarre behavior if, instead of a 0 or a 1, an input line provides a "1/2"—that is, if the input line has an improper voltage level. A proper specification of an interactive system $\boldsymbol{\Pi}$ asserts that the system guarantees a property M only under the assumption that its environment satisfies some property E.

The fundamental problem of composing specifications is to prove that a composite system satisfies its specification if all its components satisfy their specifications. Consider a system $\boldsymbol{\Pi}$ that is the composition of systems $\boldsymbol{\Pi}_1, \ldots, \boldsymbol{\Pi}_n$. We must prove that $\boldsymbol{\Pi}$ guarantees a property M under an environment assumption E, assuming that each $\boldsymbol{\Pi}_i$ satisfies a property M_i under an environment assumption E_i. Observe that:

1. We expect $\boldsymbol{\Pi}$ to guarantee M only because of the properties guaranteed by its components. Therefore, we must be able to infer that $\boldsymbol{\Pi}$ guarantees M from the assumption that each $\boldsymbol{\Pi}_i$ guarantees M_i.

2. The component $\boldsymbol{\Pi}_i$ guarantees M_i only under the assumption that its environment satisfies E_i; and $\boldsymbol{\Pi}_i$'s environment consists of $\boldsymbol{\Pi}$'s environment together with all the other components $\boldsymbol{\Pi}_j$. We must therefore be able to infer E_i from the environment assumption E and the component guarantees M_j.

These observations lead to the following principle.

Composition Principle *Let* $\boldsymbol{\Pi}$ *be the composition of* $\boldsymbol{\Pi}_1, \ldots, \boldsymbol{\Pi}_n$*, and let the following conditions hold.*

1. $\boldsymbol{\Pi}$ *guarantees* M *if each component* $\boldsymbol{\Pi}_i$ *guarantees* M_i.

2. *The environment assumption* E_i *of each component* $\boldsymbol{\Pi}_i$ *is satisfied if the environment of* $\boldsymbol{\Pi}$ *satisfies* E *and every* $\boldsymbol{\Pi}_j$ *satisfies* M_j.

3. *Every component* $\boldsymbol{\Pi}_i$ *guarantees* M_i *under environment assumption* E_i.

Then $\boldsymbol{\Pi}$ *guarantees* M *under environment assumption* E.

The reasoning embodied by the Composition Principle is circular. To prove that every E_i holds, we assume that every M_i holds; but M_i holds only under the assumption that E_i holds. So, it is not surprising that the principle is not always valid. We will show that the principle is valid under suitably weak hypotheses, and that it provides a satisfactory rule for composing specifications.

Before embarking on a rigorous development of the Composition Principle, we consider some examples. We begin with partial-correctness specifications of sequential programs. The Hoare triple $\{P\}\boldsymbol{\Pi}\{Q\}$ can be viewed as an assertion that $\boldsymbol{\Pi}$ guarantees M under environment assumption E, where M asserts that $\boldsymbol{\Pi}$ terminates only when Q is true, and E asserts that $\boldsymbol{\Pi}$ is started (by some action of the environment) only when P is true. The Composition Principle is valid for such specifications, and it is the basis for the standard composition rules of Hoare logic. For example, consider the following rule, where $\boldsymbol{\Pi}$ is the sequential composition $\boldsymbol{\Pi}_1;\boldsymbol{\Pi}_2$ of $\boldsymbol{\Pi}_1$ and $\boldsymbol{\Pi}_2$.

$$\frac{P \Rightarrow P_1,\ \ \{P_1\}\boldsymbol{\Pi}_1\{Q_1\},\ \ Q_1 \Rightarrow P_2,\ \ \{P_2\}\boldsymbol{\Pi}_2\{Q_2\},\ \ Q_2 \Rightarrow Q}{\{P\}\boldsymbol{\Pi}\{Q\}}$$

The hypotheses of this rule imply the three conditions of the Composition Principle:

1. $Q_2 \Rightarrow Q$: If $\boldsymbol{\Pi}_2$ guarantees M_2, then $\boldsymbol{\Pi}$ guarantees M.

2. $P \Rightarrow P_1$: If the environment of $\boldsymbol{\Pi}$ satisfies E, then the environment assumption E_1 of $\boldsymbol{\Pi}_1$ is satisfied.

   $Q_1 \Rightarrow P_2$: If $\boldsymbol{\Pi}_1$ guarantees M_1, then the environment assumption E_2 of $\boldsymbol{\Pi}_2$ is satisfied.

3. $\{P_i\}\boldsymbol{\Pi}_i\{Q_i\}$: $\boldsymbol{\Pi}_i$ guarantees M_i under environment assumption E_i.

The principle's conclusion, that $\boldsymbol{\Pi}$ satisfies M under environment assumption E, is the conclusion $\{P\}\boldsymbol{\Pi}\{Q\}$ of the proof rule.

We now consider reactive systems [HP85]. The interaction of a reactive system with its environment cannot be expressed simply by pre- and postconditions. For example, suppose the environment passes values to the system through a register r. If reading and writing r are not atomic operations, then the system and its environment must obey a protocol to insure the correct passing of values. If the environment does not obey the protocol, then the system could read r while it is being written and obtain completely arbitrary values—for example, values with incorrect types. The system can therefore be expected to guarantee a property M only under the assumption that the environment obeys a communication protocol, and such a protocol cannot be specified simply in terms of a precondition.

When we try to extend the Composition Principle beyond simple partial-correctness properties, we find that its validity depends on the precise nature of the properties being guaranteed and assumed. Consider the situation depicted in Figure 1, where a split wire indicates that the same value is sent to two different destinations. Suppose $\boldsymbol{\Pi}_1$ and $\boldsymbol{\Pi}_2$ have the following specifications.

- $\boldsymbol{\Pi}_1$ guarantees that it never sends a "1" on *out1*, assuming that its environment never sends it a "2" on *in1*.

- $\boldsymbol{\Pi}_2$ guarantees that it never sends a "2" on *out2*, assuming that its environment never sends it a "1" on *in2*.

System $\boldsymbol{\Pi}_1$'s guarantee M_1, that it never sends a "1" on its output wire, implies $\boldsymbol{\Pi}_2$'s assumption E_2, that its environment never sends it a "1". Similarly, $\boldsymbol{\Pi}_2$'s guarantee M_2 implies $\boldsymbol{\Pi}_1$'s environment assumption E_1. Hence, condition 2 of the Composition Principle holds. We deduce from the principle that if each component $\boldsymbol{\Pi}_i$ guarantees M_i under assumption E_i, then their composition $\boldsymbol{\Pi}$ guarantees the property M, that it never sends a "1" on *out1* and never sends a "2" on *out2*. (There is no environment assumption E because $\boldsymbol{\Pi}$ has no inputs, so its behavior is independent of its environment.) This deduction is valid. For example, suppose $\boldsymbol{\Pi}_1$ does nothing unless it receives a "2" on *in1*, whereupon it sends a "1" on *out1*; and $\boldsymbol{\Pi}_2$ behaves symmetrically. Each $\boldsymbol{\Pi}_i$ then guarantees M_i under assumption E_i, and the composite system $\boldsymbol{\Pi}$, which does nothing, guarantees M.

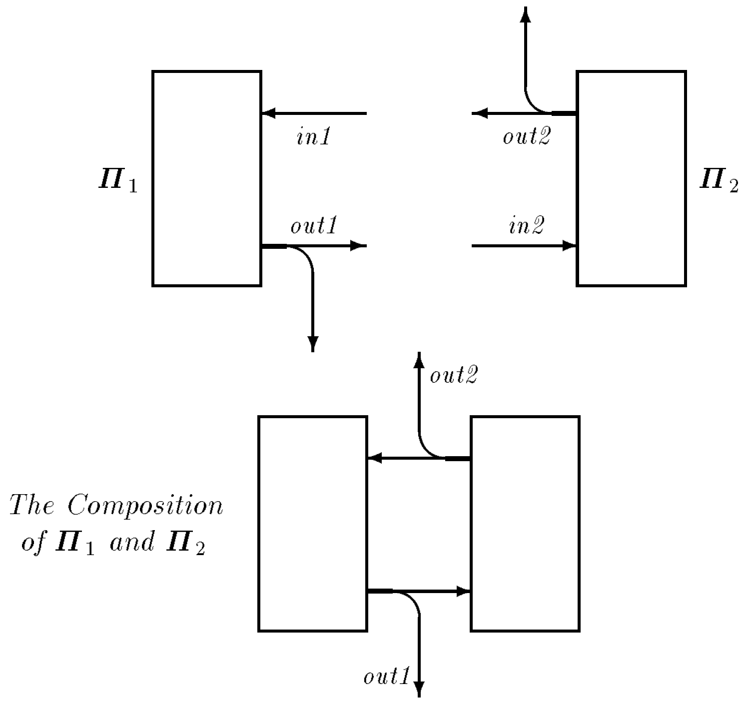

Figure 1: Composing Systems

Now consider what happens if we modify these specifications by replacing "never" with "eventually", obtaining:

- $\boldsymbol{\Pi}_1$ guarantees that it eventually sends a "1" on *out1*, assuming that its environment eventually sends it a "2" on *in1*.
- $\boldsymbol{\Pi}_2$ guarantees that it eventually sends a "2" on *out2*, assuming that its environment eventually sends it a "1" on *in2*.

Again, the property M_i guaranteed by each $\boldsymbol{\Pi}_i$ implies that the other's environment assumption E_j is satisfied. This time, the Composition Principle leads to the conclusion that $\boldsymbol{\Pi}$ guarantees eventually to send a "1" on *out1* and eventually to send a "2" on *out2*. This conclusion is invalid. The two systems described above, which send the appropriate output only after receiving the appropriate input, satisfy the modified specifications. Their composition, which does nothing, does not fulfill the guarantee implied by the Composition Principle.

Replacing "never" with "eventually" changed the guarantees M_i and the environment assumptions E_i from safety properties to liveness properties. Intuitively, a safety property asserts that something bad does not happen, while a liveness property asserts that something good eventually does happen. (Safety and liveness are defined formally in Section 2.) For most methods of describing and composing systems, the Composition Principle is valid if all guarantees and assumptions are safety properties. Various special cases of this result have appeared, in different guises. Its most familiar incarnation is in the inference rules for partial-correctness specifications; the guarantees and assumptions of such specifications are safety properties. The Composition Principle for safety properties is also embodied in a proof rule of Misra and Chandy [MC81] for processes communicating by means of CSP primitives.

Specifications that involve only safety properties are not very satisfying, since any safety property is satisfied by a system that does nothing. Liveness properties must be added to rule out trivial implementations. Pnueli [Pnu84], considering a different class of programs, gave a more general proof rule than that of Misra and Chandy. Pnueli's rule handles liveness properties, but unlike our Composition Principle, it requires an explicit induction step. Stark [Sta85] proposed another general proof rule. Stark's method handles liveness properties at the cost of requiring the discovery of a set of auxiliary assertions that explicitly break the circularity of the Composition Principle.

Our main result, Theorem 2 of Section 5.3, provides a formal statement of the Composition Principle. Its main hypothesis is that the environment assumptions are safety properties. The properties guaranteed by the system and its components need not be safety properties; they can include liveness. Theorem 1 of Section 4.3 shows that any specification satisfying a certain reasonable hypothesis is equivalent to a specification whose environment assumption is a safety property. These theorems are the fruit of a detailed examination of the distinction between a system and its environment, presented in Sections 3 and 4.

Our Composition Principle is extremely general. It does not assume any particular language or logic for writing specifications. It applies equally to specifications of Ada programs, microcode, and digital circuits. Formalizing our result in such generality requires concepts that may seem odd to readers accustomed to language-based models of computation. The rest of Section 1 introduces these concepts and relates them to other approaches that some readers may find more familiar. Precise definitions appear in Section 2.

A glossary of notation and conventions appears at the end.

1.1 States versus Actions

The popular approaches to specification are based on either states or actions. In a state-based approach, an execution of a system is viewed as a sequence of states, where a state is an assignment of values to some set of components. An action-based approach views an execution as a sequence of actions. These different approaches are, in some sense, equivalent. An action can be modeled as a state change, and a state can be modeled as an equivalence class of sequences of actions. However, the two approaches have traditionally taken very different formal directions. State-based approaches are often rooted in logic, a specification being a formula in some logical system. Action-based approaches have tended to use algebra, a specification being an object that is manipulated algebraically. Milner's CCS is the classic example of an algebraic formalism [Mil80].

State-based and action-based approaches also tend to differ in practice. To specify keyboard input using an action-based approach, the typing of a single character might be represented as a single action. In a state-based approach, it might have to be represented by two separate state changes: the key is first depressed and then released. An action-based representation often appears simpler—pressing a key is one action instead of two state changes. But this simplicity can be deceptive. A specification in which typing a character is a single action does not provide for the real situation in which a second key is depressed before the first is released. We have no reason to expect actions to be simpler than states for accurately describing real systems. We have found that a state-based approach forces a close examination of how the real system is represented in the model, helping to avoid oversimplification. On the other hand, there are circumstances in which oversimplified models are useful.

We adopt a state-based approach and use the term "action" informally to mean a state change.

1.2 System versus Environment

We view a specification as a formal description of the interface between the system and its environment. A state completely describes the state of the interface at some instant.

It is necessary to distinguish actions performed by the system from ones performed by the environment. For example, consider the specification of a clock circuit whose output is an increasing sequence of values; the circuit does not change the clock value until the environment has acknowledged reading it. The specification might include state components *clock* and *ack*, with a correct behavior consisting of a sequence of actions that alternately increment *clock* and complement *ack*.

Now, consider an "anti-clock", which is a circuit that assumes its environment (the rest of the circuit) provides a clock. The anti-clock issues acknowledgements and expects the environment to change the clock. The clock and anti-clock both display the same sequence of states—that is, the same sequence of *clock* and *ack* values—but they are obviously different systems. To distinguish them, we must specify not only what state changes may occur, but also which state changes are performed by the system and which by the environment.

An action-based formalism could simply partition the actions into system and environment actions. Formalisms based on joint system/environment actions require more subtle distinctions, such as between "internal" and "external" nondeterminism, or between the $\sqcap$ and $\Box$ operators of CSP [Hoa85].

In a state-based formalism, the easiest way to distinguish system actions from environment actions is to partition the state components into input and output components and require that the values of an input and an output component cannot both change at once. We can then declare that changes to output components are performed by the system and changes to input components are performed by the environment.

This method of partitioning the state components is not as flexible as we would like. For example, we might want to specify an individual assignment statement $x := x + 1$ as a system whose environment is the rest of the program in which it appears. Since x can be modified by other parts of the program, it is both an input and an output component for this system. In general, we want to allow module boundaries to be orthogonal to process boundaries [Lam84], so modules need not communicate only by means of simple input and output variables.

Instead of partitioning state components, we assume that each state change is performed by some "agent" and partition the set of agents into environment agents and system agents. A system execution is modeled as a *behavior*, which is a sequence of alternating states and agents, each agent being responsible for the change into the next state.

1.3 Specifying the System and its Environment

The specification of a system $\boldsymbol{\Pi}$ asserts that $\boldsymbol{\Pi}$ guarantees a property M under the assumption that its environment satisfies some property E. We will formally define a property to be a set of behaviors, so an execution of $\boldsymbol{\Pi}$ satisfies property P if and only if the behavior (a sequence of states and agents) that represents the execution is an element of P. The specification of $\boldsymbol{\Pi}$ is the property $E \Rightarrow M$, which is the set of all behaviors that are in M or not in E. A behavior satisfies this specification if it satisfies M or fails to satisfy E. The system $\boldsymbol{\Pi}$ satisfies the specification $E \Rightarrow M$ if all behaviors representing executions of $\boldsymbol{\Pi}$ are elements of $E \Rightarrow M$.

It is important to realize that E is an assumption about the environment, not a constraint placed on it. The environment cannot be constrained or controlled by the system. The system cannot prevent the user from depressing two keys at the same time. We can include in E the assumption that the user does not press two keys at once, but this means that the system guarantees to behave properly only if the user presses one key at a time. A specification that requires the user not to press two keys at once cannot be implemented unless the system can control what the user does with his fingers. This distinction between assumption and requirement is central to our results and is addressed formally in Section 3.

Our definition of a property as a set of behaviors means that we can determine whether or not a system satisfies a specification by examining each possible system execution by itself, without having to examine the set of all possible executions at once. For example, we can specify that the system's average response time be less than one millisecond in any execution containing at least 10,000 requests, where the average is over all responses in a single execution. However, we cannot specify an average response time where the average is over all possible executions.

1.4 Composition and Proof

In a modular specification method, one proves that the composition of lower-level systems implements a higher-level one. Section 5.2 explains how the refinement-mapping method described in [AL91] can be used to prove that a specification of the form $E \Rightarrow M$ implements a higher-level specification of the same form.

In our approach, composition is conjunction. Therefore, the composition of two systems with specifications $E_1 \Rightarrow M_1$ and $E_2 \Rightarrow M_2$ satisfies their conjunction, $(E_1 \Rightarrow M_1) \wedge (E_2 \Rightarrow M_2)$. To prove that this composition implements a specification $E \Rightarrow M$, we first use the Composition Principle to show that it satisfies the specification $E \Rightarrow M_1 \wedge M_2$. We can then use the method described in [AL91] to prove that $E \Rightarrow M_1 \wedge M_2$ implements $E \Rightarrow M$.

Theorem 2 (our formal statement of the Composition Principle) and Proposition 12 of Section 5.3 allow us to conclude that if $E \wedge M_2$ implies the environment assumption E_1, and $E \wedge M_1$ implies the environment assumption E_2, then the composition of systems satisfying $E_1 \Rightarrow M_1$ and $E_2 \Rightarrow M_2$ is a system satisfying $E \Rightarrow M_1 \wedge M_2$. The circularity of such a deduction was already observed in the examples based on Figure 1. Those examples had E identically true, $E_1 = M_2$, and $E_2 = M_1$; and the Composition Principle permitted us to deduce $M_1 \wedge M_2$ from $M_1 \Rightarrow M_2$ and $M_2 \Rightarrow M_1$. Theorem 2 and Proposition 12 imply that this apparently absurd deduction is valid, the major hypothesis being that E, E_1, and E_2 are safety properties. Theorem 1 of Section 4 shows that this is a reasonable hypothesis.

Our Composition Principle applies in cases where $E \Rightarrow M$ excludes behaviors allowed by the specifications $E_i \Rightarrow M_i$, so $E \Rightarrow M$ cannot be deduced logically from the properties $E_i \Rightarrow M_i$. The principle is sound because the excluded behaviors do not correspond to executions produced by any components satisfying $E_i \Rightarrow M_i$—for example, behaviors in which the environment chooses to violate E_i only after the component has violated M_i. Thus, the Composition Principle can be valid despite its apparent logical circularity.

1.5 Semantics versus Logic

In the transition-axiom method, a specification is a logical formula that describes a set of behaviors. Instead of stating our results for the particular temporal logic on which transition axioms are based, we take a more general semantic view in which a specification *is* a set of behaviors. The relation between logic and semantics is indicated by the following list of logical formulas and their corresponding semantic objects. The symbols P and Q denote formulas (logical view) and their corresponding sets of behaviors (semantic view), and Υ denotes the set of all behaviors.

Logic	Semantics	Logic	Semantics
$\neg P$	$\Upsilon - P$	$\models P$	$P = \Upsilon$
$P \wedge Q$	$P \cap Q$	$\models P \Rightarrow Q$	$P \subseteq Q$
$P \Rightarrow Q$	$(\Upsilon - P) \cup Q$		

Our semantic model is described in the following section.

2 The Semantic Model

We now define the semantic concepts on which our results are based. Most of these concepts have appeared before, so they are described only briefly; the reader can consult the cited sources for more complete discussions.

States

A *state* is an element of a nonempty set $\mathbf{S}$ of states. Except where stated otherwise, we assume that $\mathbf{S}$ is fixed. A *state predicate*, sometimes called an $\mathbf{S}$-*predicate*, is a subset of the set $\mathbf{S}$ of states.

We think of an element of $\mathbf{S}$ as representing the state, at some instant, of the relevant universe—that is, of the interfaces of all the systems under consideration. A specification should describe only what is externally visible, so elements of $\mathbf{S}$ represent only the state of the interfaces and not of any internal mechanisms.

Agents

We assume a nonempty set $\mathbf{A}$ of *agents*. If μ is a set of agents, then $\neg\mu$ denotes the set $\mathbf{A} - \mu$ of agents. An *agent set* μ is a subset of $\mathbf{A}$ such that neither μ nor $\neg\mu$ is empty. This terminology may seem confusing, since an arbitrary set of agents is not the same as an agent set. The empty set of agents $\emptyset$ and the full set of agents $\mathbf{A}$ turn out to be anomalous for uninteresting technical reasons; sometimes we unobtrusively exclude these anomalous cases by considering only agent sets.

We think of the elements of $\mathbf{A}$ as the entities responsible for changing the state. A specification describes what it means for a set of agents μ to form a correctly operating system—in other words, what it means for a behavior to be correct when the agents in μ are considered to form the system and the agents in $\neg\mu$ are considered to form the environment.

In describing a system, the particular agent that performs an action is not important; what matters is whether the agent belongs to the system or the environment. Thus, if we are dealing with a single specification, we could assume just two agents, a system agent and an environment agent, as was done by Barringer, Kuiper, and Pnueli in [BKP86] and by us in [ALW89]. However, for composing specifications, one needs more general sets of agents, as introduced in [Lam83a] (where agents were called "actions").

It may help the reader to think of the agents as elementary circuit components or individual machine-language instructions. However, the actual identity of the individual agents never matters.

Behaviors

A *behavior prefix* is a sequence

$$s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \ldots \tag{1}$$

where each s_i is a state and each α_i is an agent, and the sequence is either infinite or else ends in a state s_m for some $m \geq 0$. A *behavior* is an infinite behavior prefix. If σ is the behavior prefix (1), then $\mathbf{s}_i(\sigma)$ denotes s_i and $\mathbf{a}_i(\sigma)$ denotes α_i. For a behavior σ, we let $\sigma|_m$ denote the finite prefix of σ ending with the m^{th} state $\mathbf{s}_m(\sigma)$, for $m \geq 0$. We sometimes use the term $\mathbf{S}$-behavior to indicate that the states in the behavior are elements of $\mathbf{S}$.

A behavior represents a possible complete history of the relevant universe, starting at some appropriate time. As usual in state-based approaches, we adopt an interleaving semantics, in which the evolution of the universe is broken into atomic actions (state changes), and concurrent actions are considered to happen in some arbitrary order. A step $s_{i-1} \xrightarrow{\alpha_i} s_i$ of a behavior denotes an action in which agent α_i changes the state of the universe from s_{i-1} to s_i. Steps in our formalism correspond to the actions of action-based formalisms.

Stuttering-Equivalence

If μ is any set of agents, then a *μ-stuttering step* is a sequence $s \xrightarrow{\alpha} s$ with $\alpha \in \mu$. If σ is a behavior prefix, then $\natural_\mu \sigma$ is defined to be the behavior prefix obtained from σ by replacing every maximal (finite or infinite) sequence $s \xrightarrow{\alpha_1} s \xrightarrow{\alpha_2} s \ldots$ of μ-stuttering steps with the single state s. Two behavior prefixes σ and τ are said to be *μ-stuttering-equivalent*, written $\sigma \simeq_\mu \tau$, iff (if and only if) $\natural_\mu \sigma = \natural_\mu \tau$. When μ equals $\mathbf{A}$, we write $\sigma \simeq \tau$ instead of $\sigma \simeq_{\mathbf{A}} \tau$ and *stuttering-equivalent* instead of $\mathbf{A}$*-stuttering-equivalent*. If σ is a finite behavior prefix, then $\widehat{\sigma}$ is defined to be some arbitrary behavior such that $\widehat{\sigma} \simeq \sigma$ and $\widehat{\sigma}|_m = \sigma$ for some m. (The precise choice of $\widehat{\sigma}$, which involves choosing which agents perform the infinite number of stuttering steps that must be added to σ, does not matter.)

A state describes the state of the entire relevant universe, and a stuttering step does not change the state, so a stuttering step has no observable effect. Therefore, two behaviors that are stuttering-equivalent should be indistinguishable. A useful way to think about stuttering is to imagine that a state in $\mathbf{S}$ describes only the observable parts of the universe, and that there are also unobservable, internal state components of the various objects that make up the universe. A stuttering step represents a step in which some object changes only its internal state. As explained in [Lam83b] and [Lam89], considering stuttering-equivalent behaviors to be equivalent allows the hierarchical decomposition of specifications by refining the grain of atomicity.

If σ is a finite behavior prefix, then $\widehat{\sigma}$ is obtained from σ by adding an infinite number of stuttering steps. The behavior $\widehat{\sigma}$ represents a history of the universe in which all externally observable activity ceases after a finite number of steps. (For example, a computer that has halted continues to take stuttering steps because its internal clock keeps ticking.)

Properties

A *property* P is a set of behaviors that is closed under stuttering-equivalence, meaning that for any behaviors σ and τ, if $\sigma \simeq \tau$ then $\sigma \in P$ iff $\tau \in P$. We sometimes call P an $\mathbf{S}$-property to indicate that it is a set of $\mathbf{S}$-behaviors. A state predicate I is considered to be the property such that $\sigma \in I$ iff $\mathbf{s}_0(\sigma) \in I$. For properties P and Q, we define $P \Rightarrow Q$ to be the property $(\neg P) \cup Q$, where $\neg$ denotes complementation in the set of all behaviors. In formulas, $\Rightarrow$ has lower precedence than $\cap$, so $P \cap Q \Rightarrow R$ denotes $(P \cap Q) \Rightarrow R$.

A property P is a *safety* property iff it satisfies the following condition: a behavior σ is in P iff $\widehat{\sigma|_m} \in P$ for all $m \geq 0$. A property P is a *liveness* property iff every finite behavior prefix is a prefix of a behavior in P. With a standard topology on the set of behaviors, a property is a safety property iff it is closed, and it is a liveness property iff it is dense [AS85]. It follows from elementary results of topology that every property is the conjunction of a safety property and a liveness property. The closure of a property P in this topology, written $\overline{P}$, is the smallest safety property containing P.

Property P is a safety property iff every behavior not in P has a finite prefix that is not in P. Hence, a safety property is one that is finitely refutable. For any state predicate I, the property I depends only on the initial state, so it is a safety property. A property P is a liveness property iff every finite behavior prefix can be completed to a behavior in P. Hence, a liveness property is one that is never finitely refutable. Alpern and Schneider [AS85] discussed these definitions in more detail.

For properties P and Q, we define $P \mathrel{-\!\!\triangleright} Q$ to be the set of all behaviors σ such that $\widehat{\sigma|_m} \in \overline{P} \Rightarrow \overline{Q}$ for all $m \geq 0$. Thus, $P \mathrel{-\!\!\triangleright} Q$ is the safety property asserting that $\overline{Q}$ cannot become false before $\overline{P}$ does. It follows from the definition that $\overline{Q} \subseteq (P \mathrel{-\!\!\triangleright} Q) \subseteq (\overline{P} \Rightarrow \overline{Q})$, for any properties P and Q.

The specification of a system is the property consisting of all behaviors (histories of the relevant universe) in which the system is considered to perform correctly.

μ-Abstractness

If μ is a set of agents, then two behaviors σ and τ are *μ-equivalent* iff, for all $i \geq 0$:

- $\mathbf{s}_i(\sigma) = \mathbf{s}_i(\tau)$
- $\mathbf{a}_{i+1}(\sigma) \in \mu$ iff $\mathbf{a}_{i+1}(\tau) \in \mu$.

A set P of behaviors is *μ-abstract* iff, for any behaviors σ and τ that are μ-equivalent, $\sigma \in P$ iff $\tau \in P$.

Two behaviors are μ-equivalent iff they would be the same if we replaced every agent in μ by a single agent, and every agent not in μ by a different single agent. A reasonable specification of a system does not describe which agent performs an action, only whether the action is performed by a system or an environment agent. Thus, if μ is the set of system agents, then the specification should not distinguish between μ-equivalent behaviors, so it should be a μ-abstract property.

3 Realizability

A specification of a system is a property P consisting of all behaviors in which the system performs correctly. Whether a behavior is allowed by the specification may depend upon the environment's actions as well as the system's actions. This dependence upon what the environment does is unavoidable, since the system cannot be expected to perform in a prescribed fashion if the environment does not behave correctly. However, the ability to specify the environment as well as the system gives us the ability to write specifications that constrain what the environment is allowed to do. Such a specification would require the system to control (or predict) what the environment will do; it would be unimplementable because the environment is precisely the part of the universe that the system *cannot* control.

A specification should assert that the system performs properly if the environment does; it should not assert that the environment performs properly. For example, assume that the environment is supposed to decrement some state component x. A specification (property) P asserting that the environment must decrement x would not be implementable because given any system, there is a possible universe containing the system whose behavior is not in P—namely one in which the environment never decrements x. Hence, no system can satisfy the specification P. A specification of the system should allow all behaviors in which the environment never decrements x.

A specification that is unimplementable because it constrains the environment's actions is called *unrealizable*. (A specification may be unimplementable for other reasons that do not concern us here—for example, because it requires the system to compute a noncomputable function.) We now define precisely what realizability means, and explore some of its implications for specifications. The definitions are almost identical to the ones in [AL91].

3.1 Safety Properties

A safety property is finitely refutable, so if a behavior does not satisfy the property, then we can tell who took the step that violated it. More precisely, if P is a safety property and a behavior σ is not in P, then there is some number $m \geq 0$ such that $\widehat{\sigma|_m}$ is not in P. If m is the smallest such number, then we can say that P was violated by the agent that performed the m^{th} step of σ, assuming $m > 0$. A safety property is defined to constrain only the system iff the property can be violated only by system agents.

We now formalize this definition. For any property P and behavior σ, let $V(P, \sigma)$ equal the smallest nonnegative integer m such that $\widehat{\sigma|_m}$ is not in P. (We leave $V(P, \sigma)$ undefined if there is no such m.) If μ is an agent set, then a safety property P *constrains at most* μ iff for all behaviors σ, if $\sigma \notin P$ then $V(P, \sigma) > 0$ and $\mathbf{a}_{V(P,\sigma)}(\sigma) \in \mu$.

3.2 Realizability of Arbitrary Properties

3.2.1 Definitions

To understand the general concept of realizability, it helps to think of a behavior as the outcome of a two-person infinite game played by the system and the environment. The environment chooses the initial state, and then the environment and the system alternate moves to produce the behavior, with the environment taking the first move. An environment move consists of adding any finite number of steps performed by environment agents (possibly zero steps); a system move consists of doing nothing or adding one step performed by a system agent. (A similar class of games was studied by Morton Davis [Dav64].) The system wins the game iff the resulting behavior prefix satisfies the specification or is finite. (Our informal discussion is simplified by considering the system to win games with finite outcomes, which do not correspond to the infinite behaviors of our formalism.) A specification is said to be *realizable* iff the system has a winning strategy—that is, iff the system can always win no matter what moves the environment makes.

A specification is realizable if it has enough behaviors so that the system can win even if the environment plays as well as it can. A specification may also contain behaviors that are outcomes of games in which the environment had a chance to win but played badly and lost. A correct implementation can never allow such behaviors to occur because it can't count on the environment playing badly. The *realizable part* of a specification is defined to consist only of those behaviors in which the environment never had a chance to win. An implementation that satisfies the specification can produce only behaviors in the realizable part. Hence, two specifications have the same implementations iff they have the same realizable parts. Two such specifications are said to be *equirealizable*. We can replace a specification with an equirealizable one without changing the class of real systems that are being specified.

The formal definitions of these concepts is based on the definition of a strategy, which is a rule by which the system determines its next move. More precisely, a strategy is a partial function that determines the system's next step as a function of the behavior up to that point. It suffices to consider deterministic strategies, since the set of behaviors that result from a nondeterministic strategy is the union of the sets of behaviors produced by some set of deterministic strategies. In the following definitions, μ is an arbitrary agent set.

- A *μ-strategy* f is a partial function from the set of finite behavior prefixes to $\mu \times \mathbf{S}$. (Intuitively, $f(\sigma) = (\alpha, s)$ means that, if the system gets to move after play has produced σ, then it adds $\xrightarrow{\alpha} s$. If $f(\sigma)$ is undefined, then the system chooses not to move.)

- A *μ-outcome* of a μ-strategy f is a behavior σ such that for all $m > 0$, if $\mathbf{a}_m(\sigma) \in \mu$ then $f(\sigma|_{m-1}) = (\mathbf{a}_m(\sigma), \mathbf{s}_m(\sigma))$. A μ-outcome σ is *fair* iff $\mathbf{a}_{m+1}(\sigma) \in \mu$ or $\sigma|_m$ is not in the domain of f for infinitely many values of m. (A μ-outcome of f is one in which all the μ-moves were produced by the strategy f. It is fair iff it could have been obtained by giving the system an infinite number of chances to move.)

- If f is a μ-strategy, then $\mathcal{O}_\mu(f)$ is the set of all fair μ-outcomes of f.

- The *μ-realizable* part of a set P of behaviors, denoted $\mathcal{R}_\mu(P)$, is the union of all sets $\mathcal{O}_\mu(f)$ such that f is a μ-strategy and $\mathcal{O}_\mu(f) \subseteq P$. (Intuitively, $\mathcal{R}_\mu(P)$ is the set of fair outcomes that can be produced by correct implementations of P.) We show in Proposition 1 below that $\mathcal{R}_\mu(P)$ is a property if P is.

- A property P is *μ-realizable* iff $\mathcal{R}_\mu(P)$ is nonempty. (A μ-realizable property is one that has a correct implementation.)

- Properties P and Q are *μ-equirealizable* iff $\mathcal{R}_\mu(P) = \mathcal{R}_\mu(Q)$. (Equirealizable properties have the same correct implementations.)

- A property P is *μ-receptive* iff $\mathcal{R}_\mu(P) = P$. (A μ-receptive property includes only behaviors that can be produced by correct implementations.)

Stark studied a generalization of receptiveness, which he called *local $\mathcal{D}$-consistency* in his thesis [Sta84]. The special case corresponding to our definition of receptiveness was not considered in the thesis, but did appear in his unpublished thesis proposal. Dill independently developed the notion of receptiveness and introduced its name [Dil88]. In [ALW89], a concept of realizability was defined in which $\mathcal{O}_\mu(f)$ included all outcomes, rather than just fair ones. By eliminating unfair outcomes, we are preventing the environment from ending the game by taking an infinite number of steps in a single move. Allowing such an infinite move, in which the environment prevents the system from ever taking another step, would produce a game that does not correspond to the kind of autonomous system that we are concerned with here. Our concept of realizability is similar but not identical to fair realizability as defined in [ALW89]. The difference between these two concepts is described below.

3.2.2 Discussion of the Definitions

The set $\mathcal{O}_\mu(f)$ is not in general a property; it can contain a behavior σ and not contain a behavior σ' that is stuttering-equivalent to σ. Moreover, since the strategy f chooses specific agents, the set $\mathcal{O}_\mu(f)$ is not μ-abstract. However, our definitions do insure that $\mathcal{R}_\mu$ preserves invariance under stuttering and μ-abstractness.

Proposition 1 *For every agent set μ, if P is a property then $\mathcal{R}_\mu(P)$ is a property, and if P is μ-abstract then $\mathcal{R}_\mu(P)$ is μ-abstract.*

The proofs of this and of our other results appear in the appendix.

Our definition of strategies allows them to depend upon the presence or absence of stuttering. In other words, if f is a μ-strategy, then $f(\sigma)$ and $f(\tau)$ can be different for two stuttering-equivalent prefixes σ and τ. This seems to contradict our assertion that stuttering-equivalent behaviors should be indistinguishable. If we think of a stuttering step as representing an externally unobservable step of some object, then the system should certainly not be able to detect stuttering actions performed by the environment. Define f to be *invariant under $\neg\mu$-stuttering* iff $\sigma \simeq_{\neg\mu} \tau$ implies $f(\sigma) = f(\tau)$, for all finite behavior prefixes σ and τ. It would be more natural to add to the definition of a μ-strategy f the requirement that f be invariant under $\neg\mu$-stuttering. The following proposition shows that we could restrict ourselves to such strategies, and could even add the further requirement that the strategies be total functions.

Proposition 2 *For any agent set μ and any property P, let $\mathcal{S}_\mu(P)$ be the subset of $\mathcal{R}_\mu(P)$ consisting of the union of all sets $\mathcal{O}_\mu(f)$ contained in P such that f is a total μ-strategy that is invariant under $\neg\mu$-stuttering. Then every behavior in $\mathcal{R}_\mu(P)$ is stuttering-equivalent to a behavior in $\mathcal{S}_\mu(P)$.*

We could thus define $\mathcal{R}_\mu(P)$ to be the closure of $\mathcal{S}_\mu(P)$ under stuttering-equivalence. Taking this closure would be necessary even if one of the two conditions—totality or invariance under $\neg\mu$-stuttering—were dropped from the definition of $\mathcal{S}_\mu(P)$. It is therefore more convenient to allow arbitrary strategies in the definition of $\mathcal{R}_\mu(P)$.

Although we could restrict ourselves to μ-strategies that are invariant under $\neg\mu$-stuttering, requiring strategies to be invariant under all stuttering, as in the definition of "fair realizability" of [ALW89], would materially change our definitions. A result in Stark's unpublished thesis proposal suggests that this restriction would not change the definition of realizability; but the following example shows that it would alter the definition of receptiveness. Let P be the property consisting of all behaviors containing infinitely many nonstuttering steps. With the definitions used here, P equals its μ-realizable part. With the definition in [ALW89], the "fairly μ-realizable" part of P would consist of only those behaviors containing infinitely many nonstuttering μ steps. (This example demonstrates that a conjecture of Broy et al. [BDDW91] is false.)

System stuttering steps represent ones in which the system changes only its internal state, so allowing a μ-strategy to depend upon μ-stuttering steps is equivalent to allowing the strategy to depend upon the system's internal state. More precisely, suppose that the state includes some "variable" that the property P does not depend on. Then adding the requirement that a μ-strategy be invariant under stuttering does not change the definition of $\mathcal{R}_\mu(P)$. (This can be proved by showing that if a μ-strategy f is invariant under $\neg\mu$-stuttering, then one can modify f to obtain an "equivalent" strategy f' that is invariant under all stuttering; f' takes a step that changes only the extra variable whenever f takes a stuttering step.) By allowing a strategy to depend upon stuttering steps, we obviate the need to rely upon internal state for our definitions.

3.2.3 Some Basic Propositions

We now state some results about realizability. The first asserts that $\mathcal{R}_\mu$ is monotonic.

Proposition 3 *For any properties P and Q and any agent set μ, if $P \subseteq Q$ then $\mathcal{R}_\mu(P) \subseteq \mathcal{R}_\mu(Q)$.*

The next proposition asserts that the realizable part of a property is receptive.

Proposition 4 *For any property P and agent set μ, $\mathcal{R}_\mu(\mathcal{R}_\mu(P)) = \mathcal{R}_\mu(P)$.*

The next result provides a useful representation of the realizable part of a property.

Proposition 5 *For any property P and agent set μ, $\mathcal{R}_\mu(P) = \overline{\mathcal{R}_\mu(P)} \cap P$.*

The next result indicates that "constrains at most" and receptiveness are essentially the same for safety properties.

Proposition 6 *For any nonempty safety property P and any agent set μ, property P constrains at most μ iff P is μ-receptive.*

Proposition 4 asserts that the μ-realizable part $\mathcal{R}_\mu(P)$ of a property P is μ-receptive. Hence, Proposition 6 implies that, if $\overline{\mathcal{R}_\mu(P)}$ is a nonempty safety property, then it constrains at most μ. The following result generalizes this to the case when $\mathcal{R}_\mu(P)$ is not a safety property.

Proposition 7 *For any agent set μ, if P is a μ-realizable property then $\overline{\mathcal{R}_\mu(P)}$ constrains at most μ.*

In general, the realizable part of a property is not expressible in terms of simpler operations on properties. Proposition 6 describes a simple case in which $\mathcal{R}_\mu(P)$ equals P. Since *true* $\Rightarrow Q$ and *true* $\mathrel{-\!\triangleright} Q$ both equal Q, the following proposition generalizes the "only if" part of Proposition 6.

Proposition 8 *Let μ be an agent set, I a state predicate, P a safety property that constrains at most $\neg\mu$, and Q a safety property that constrains at most μ. Then $\mathcal{R}_\mu(I \cap P \Rightarrow Q)$ equals $I \cap P \mathrel{-\!\triangleright} Q$.*

4 The Form of a Specification

Our Composition Principle applies only to specifications of the form $E \Rightarrow M$, where E is a safety property. In this section, we explain why specifications can and should be written in this way. Before considering general specifications, we first examine a particular class of specifications—programs. A program is a specification that is sufficiently detailed so a system that satisfies it can be generated automatically. Typically, a system satisfying the specification is generated by compiling the program and executing the resulting code on a computer.

4.1 The Form of a Complete Program

We start by considering complete programs. In formal models of complete programs, there are no environment actions, only system actions. Input occurs through initial values of variables or by executing a nondeterministic *input* statement in the program. (An *input* statement is nondeterministic because the program text and the execution of the program up to that point do not determine the input value.) Thus, a complete program is a specification in which every agent in $\mathbf{A}$ is a system agent. Since we want the specification to be $\mathbf{A}$-abstract, it does not matter what agents perform the steps of a behavior, so we can ignore the agents and consider a behavior to be a sequence of states.

4.1.1 The Parts of a Complete Program

A complete program is defined by four things:

set of states A state provides an "instantaneous picture" of the execution status of the program. It is determined by such things as the values of variables, the loci of control of processes, and the messages in transit—the details depending upon the programming language.

initial predicate The initial predicate I is a state predicate that specifies the set of valid starting states of the program. Recall that the predicate I (a set of states) is interpreted as the property consisting of all behaviors whose starting state is in I.

next-state relation The next-state relation $\mathcal{N}$ is a set of pairs of states that describes the state transitions allowed by the program, where $(s, t) \in \mathcal{N}$ iff executing one step of the program starting in state s can produce the new state t. It is described explicitly by the program text and the assumptions about what actions are considered to be atomic. The next-state relation $\mathcal{N}$ determines a property $\mathcal{TA}(\mathcal{N})$, defined by $\sigma \in \mathcal{TA}(\mathcal{N})$ iff $\mathbf{s}_i(\sigma) = \mathbf{s}_{i+1}(\sigma)$ or $(\mathbf{s}_i(\sigma), \mathbf{s}_{i+1}(\sigma)) \in \mathcal{N}$, for all $i \geq 0$. In other words, $\mathcal{TA}(\mathcal{N})$ is the set of all behaviors in which each nonstuttering step is allowed by the next-state relation $\mathcal{N}$.

progress property The next-state relation specifies what state changes *may* occur, but it does not require that any state changes actually do occur. The progress property L specifies what must occur. A common type of progress property is one asserting that if some state change is allowed by the next-state relation, then some state change must occur.

Formally, the program is the property $I \cap \mathcal{TA}(\mathcal{N}) \cap L$. Note that I and $\mathcal{TA}(\mathcal{N})$, and hence $I \cap \mathcal{TA}(\mathcal{N})$, are safety properties.

All assertional methods of reasoning about concurrent programs are based on a description of the program in terms of a set of states, an initial predicate, and a next-state relation. By now, these methods should be familiar enough that there is no need for us to discuss those parts of the program. Progress properties are less well understood and merit further consideration.

4.1.2 The Progress Property

Assertional methods that deal with liveness properties need some way of specifying the program's progress property. The requirement that the program be executable in practice constrains the type of progress property that can be allowed. The initial state and the computer instructions executed by a program are derived from the program's code, which specifies the next-state relation. The progress property should constrain the eventual scheduling of instructions, but not which instructions are executed. For the program to be executable in practice, the state transitions that it may perform must be determined by the initial state and the next-state relation alone; they must not be constrained by the progress property.

As an example, consider the simple next-state relation pictured in Figure 2, where the program state consists of the value of the single variable x. Assume that the initial predicate asserts that x equals 0. The property asserting that $x = 3$ holds at some time during execution, usually written $\Diamond(x = 3)$, is a liveness property. However, for the program to satisfy this property, it must not make the state transition from $x = 0$ to $x = 1$ allowed by the next-state relation. Thus, if $\Diamond(x = 3)$ were the program's progress property, a compiler would have to deduce that the transition from $x = 0$ to $x = 1$, which is permitted by the next-state relation, must not occur.

The condition that the progress property L does not further constrain the initial state or the next-state relation is expressed formally by the following conditions, which are all equivalent.

- For every finite behavior prefix ρ with $\widehat{\rho}$ in $I \cap \mathcal{TA}(\mathcal{N})$, there exists a behavior σ in $I \cap \mathcal{TA}(\mathcal{N}) \cap L$ such that ρ is a prefix of σ.
- $I \cap \mathcal{TA}(\mathcal{N}) \;=\; \overline{I \cap \mathcal{TA}(\mathcal{N}) \cap L}$

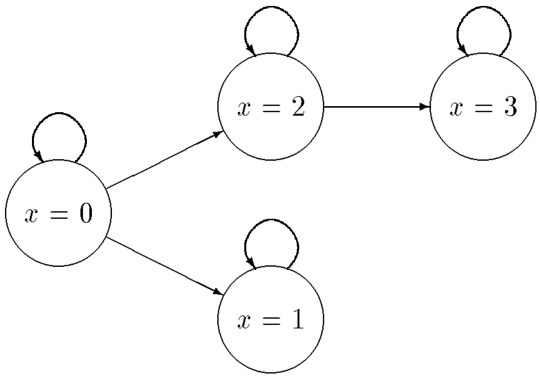

Figure 2: A simple next-state relation.

- If Q is any safety property, then $I \cap \mathcal{TA}(\mathcal{N}) \cap L \subseteq Q$ iff $I \cap \mathcal{TA}(\mathcal{N}) \subseteq Q$.

The last condition asserts that the safety properties satisfied by the program are completely determined by the initial predicate and the next-state relation; in other words, the progress property does not add any safety properties.

We define a pair (M, P) of properties to be *machine-closed* iff $M = \overline{P}$. (The term "machine-closed" was introduced in [AL91].) Machine closure of (M, P) means that P does not imply any safety properties not implied by M. So, if L is a progress property, we expect the pair $(I \cap \mathcal{TA}(\mathcal{N}), I \cap \mathcal{TA}(\mathcal{N}) \cap L)$ to be machine-closed. When this condition is satisfied, we sometimes informally write that the progress property L or the program is machine-closed. To our knowledge, all the progress assumptions that have been proposed for programs are machine-closed.

A program's progress property is usually called a fairness condition. There have been few attempts to give a general definition of fairness. Manna and Pnueli [MP87] define a class of "fairness" properties that is independent of any next-state relation, but they provide no justification for their terminology. Apt, Francez, and Katz [AFK88] discuss three "fairness criteria"; one of them is machine-closure, which they call "feasibility".

Most of the progress properties that have been proposed can be stated as fairness conditions on program actions—for example, the condition that certain state transitions cannot be enabled forever without occurring. These progress properties are not all generally considered to be fairness properties. In particular, the property asserting that the entire program never stops if some step can be executed is machine-closed, but multiprocess programs satisfying only this progress assumption are generally called unfair. We believe that machine-closure provides the proper definition of a progress property, and that any distinction between fairness properties and progress properties is probably language-dependent and not fundamental.

4.2 The Form of a Partial Program

A partial program is part of a larger program. It may be a single process in a CSP program, or a single assignment statement in a Pascal program. It should be possible to implement the partial program independently of the rest of the program, which constitutes its environment. Such an implementation might be very inefficient—as, for example, if each assignment statement of a Pascal program were compiled independently without knowing the types of the variables—but it should be possible. Actions may be taken either by the partial program or by the rest of the program, which constitutes the partial program's environment.

4.2.1 The Parts of a Partial Program

The following modifications of the parts that define a program are needed to handle partial programs.

set of states The complete state cannot be determined from the text of the partial program. For example, there is no way of knowing what variables are introduced in other parts of the complete program. There are two ways to define the set of states $\mathbf{S}$ for a partial program.

- $\mathbf{S}$ is the set of states defined by the complete program. Since the complete program is not known, $\mathbf{S}$ is not known, so the meaning of the partial program depends upon a fixed but unknown set of states.
- $\mathbf{S}$ includes all possible program variables and other state components. The meaning of the partial program is defined in terms of a known set of states, but it is a very "large" set of states, since it must accommodate all possible complete programs.

Both approaches lead to equivalent formalisms. Here, we find the first assumption most convenient, and we take $\mathbf{S}$ to be the unknown set of states of the larger program. The partial program modifies only those components of the state explicitly mentioned; the environment can modify any part of the state.

agent set We use agents to distinguish the actions performed by the partial program from the ones performed by its environment. Program steps are taken by agents in μ, environment steps by agents in $\neg\mu$. We don't care which agents in μ or in $\neg\mu$ take the steps, so it suffices to distinguish only μ steps and $\neg\mu$ steps.

initial predicate In our "realization game", the environment chooses the initial state. The initial condition must therefore become part of the environment specification, so it disappears from the program.

next-state relation The next-state relation $\mathcal{N}$ now constrains only the state transitions performed by the program, not the ones performed by the environment. It describes the property $\mathcal{TA}_\mu(\mathcal{N})$, which is defined by $\sigma \in \mathcal{TA}_\mu(\mathcal{N})$ iff $\mathbf{a}_{i+1}(\sigma) \in \mu$ implies $\mathbf{s}_i(\sigma) = \mathbf{s}_{i+1}(\sigma)$ or $(\mathbf{s}_i(\sigma), \mathbf{s}_{i+1}(\sigma)) \in \mathcal{N}$, for all $i \geq 0$. The next-state relation must be defined in such a way that any part of the state not explicitly mentioned is left unchanged.

This leaves the question of what is the appropriate modification to the machine-closure condition for progress properties. Recall that machine-closure was derived from the requirement that a complete program be implementable in practice. Ignoring the initial predicate, machine-closure asserts that any finite execution satisfying the next-state relation can be completed to an execution satisfying the next-state relation and the progress property. We similarly require that the partial program be implementable in practice, except now we have the additional requirement that it be implementable without knowing its environment. In other words, the implementation must work regardless of what the environment does. We therefore require that given any finite behavior prefix in which the program's actions satisfy the next-state relation, there is a strategy that the program can play from that point on and "win"—that is, produce a behavior satisfying the next-state relation and the progress property.

The formal expression of this condition is statement (a) in the following proposition, when $\mathcal{TA}_\mu(\mathcal{N})$ is substituted for M. Statement (b) is a useful variant of (a), and (c) is a reformulation of (a) in terms of topology and receptiveness.

Proposition 9 *For any agent set μ, safety property M, and arbitrary property L, the following three conditions are equivalent:*

(a) For every finite behavior ρ such that $\widehat{\rho} \in M$, there exist a μ-strategy f with $\mathcal{O}_\mu(f) \subseteq M \cap L$ and a behavior $\sigma \in \mathcal{O}_\mu(f)$ with ρ a prefix of σ.

(b) *For every finite behavior ρ such that $\widehat{\rho} \in M$, there exist a μ-strategy f with $\mathcal{O}_\mu(f) \subseteq M \cap L$ and a behavior $\sigma \in \mathcal{O}_\mu(f)$ with ρ stuttering-equivalent to a prefix of σ.*

(c) *The pair $(M,\ M \cap L)$ is machine-closed, and $M \cap L$ is μ-receptive.*

We define a pair of properties (M, P) to be *μ-machine-realizable* iff it is machine-closed and P is μ-receptive. The generalization to partial programs of the machine-closure condition on a progress property L is that the pair $(\mathcal{TA}_\mu(\mathcal{N}),\ \mathcal{TA}_\mu(\mathcal{N}) \cap L)$ be μ-machine-realizable, where $\mathcal{N}$ is the program's next-state relation. In this case, we say informally that L is machine-realizable.

To illustrate the difference between progress properties of partial and complete programs, let $L_\mathcal{A}$ be the property asserting that if some program action $\mathcal{A}$ is infinitely often enabled, then that action must occur infinitely often. More formally, let $\mathcal{A}$ be a subset of the next-state relation $\mathcal{N}$, define $\mathcal{A}$ to be enabled in a state s iff there exists a state t with $(s, t) \in \mathcal{A}$, and define $L_\mathcal{A}$ to be the property such that $\sigma \in L_\mathcal{A}$ iff either $\mathcal{A}$ is enabled in state $\mathbf{s}_i(\sigma)$ for only finitely many values of i, or else $(\mathbf{s}_i(\sigma), \mathbf{s}_{i+1}(\sigma)) \in \mathcal{A}$ for infinitely many values of i. The property $L_\mathcal{A}$ is the usual *strong fairness* requirement for action $\mathcal{A}$. Strong fairness is a reasonable progress property for a complete program, since it is machine-closed.

Now, suppose that $L_\mathcal{A}$ is the progress property of a partial program. When playing the "realization game", the environment can play infinitely many moves in which it adds two states—one in which $\mathcal{A}$ is enabled followed by one in which it is not enabled. (Such environment moves are "legal" because the partial program's safety property $\mathcal{TA}_\mu(\mathcal{N})$ allows any steps by the environment.) The program never has a chance to take an $\mathcal{A}$ step because it never gets to play a move when $\mathcal{A}$ is enabled. Thus, the resulting outcome does not satisfy the property $L_\mathcal{A}$, so $L_\mathcal{A}$ is not a machine-realizable progress property. In fact, it is not even realizable. This losing outcome corresponds to a physical situation in which the environment changes the state so fast that $\mathcal{A}$ never stays enabled long enough for the program to react in time to perform an $\mathcal{A}$ action.

To obtain a machine-realizable progress property, let $\mathcal{N}'$ be a next-state relation asserting that $\mathcal{A}$ is never disabled. Formally, $(s, t) \in \mathcal{N}'$ iff $\mathcal{A}$ is not enabled in s or is enabled in t. The property $\mathcal{TA}_{\neg\mu}(\mathcal{N}')$ asserts that the environment never disables $\mathcal{A}$. The progress property $\mathcal{TA}_{\neg\mu}(\mathcal{N}') \Rightarrow L_\mathcal{A}$ is machine-realizable. In the realization game, the environment loses if it ever disables $\mathcal{A}$, since doing so ensures that $\mathcal{TA}_{\neg\mu}(\mathcal{N}')$ will be false, making

$\mathcal{TA}_{\neg\mu}(\mathcal{N}') \Rightarrow L_{\mathcal{A}}$ true. The program can therefore always win the game by taking an $\mathcal{A}$ step whenever it gets to move with $\mathcal{A}$ enabled.

4.2.2 Hiding the Internal State

Another important concept introduced when considering partial programs is *hiding*. Variables and other state components that are local to the partial program should be hidden—meaning that they are modified only by the program and do not conflict with similarly-named components in the environment. In our approach, hiding is effected by existential quantification over state components.

Existential Quantification Existential quantification is defined formally as follows. Let $\mathbf{X}$ denote a set of values, let $\Pi_{\mathbf{S}}$ and $\Pi_{\mathbf{X}}$ denote the projection functions from $\mathbf{S} \times \mathbf{X}$ to $\mathbf{S}$ and $\mathbf{X}$, respectively, and let $\mathbf{x}$ be an abbreviation for $\Pi_{\mathbf{X}}$. We extend $\Pi_{\mathbf{S}}$ to a mapping from $\mathbf{S}\times\mathbf{X}$-behaviors to $\mathbf{S}$-behaviors by letting $\Pi_{\mathbf{S}}(\sigma)$ be the behavior such that, $\mathbf{a}_i(\Pi_{\mathbf{S}}(\sigma)) = \mathbf{a}_i(\sigma)$ and $\mathbf{s}_i(\Pi_{\mathbf{S}}(\sigma)) = \Pi_{\mathbf{S}}(\mathbf{s}_i(\sigma))$ for all i. For any $\mathbf{S} \times \mathbf{X}$-property P, we define $\exists \mathbf{x} : P$ to be the $\mathbf{S}$-property such that σ is in $\exists \mathbf{x} : P$ iff there exists an $\mathbf{S} \times \mathbf{X}$-behavior σ' in P with $\Pi_{\mathbf{S}}(\sigma') \simeq \sigma$.

Intuitively, $\mathbf{S}\times\mathbf{X}$ is a set of states in which $\mathbf{S}$ is the externally observable component and $\mathbf{X}$ is the component internal to the program. The property $\exists \mathbf{x} : P$ is obtained from P by hiding the $\mathbf{x}$-component. We use the notation "$\exists \mathbf{x}$" for this hiding operator because it obeys the logical rules of existential quantification when properties are expressed as formulas in an appropriate logic [Lam90]. As usual, $\exists$ binds more weakly than other operators.

Hiding with Existential Quantification Let $\mathcal{N}$ be the next-state relation of the program and L its progress property. When there is an internal state component, $\mathcal{N}$ is a set of pairs of elements of $\mathbf{S} \times \mathbf{X}$—in other words, a subset of $(\mathbf{S} \times \mathbf{X}) \times (\mathbf{S} \times \mathbf{X})$—and L is an $\mathbf{S} \times \mathbf{X}$-property. Formally, the program is the property $\exists \mathbf{x} : P \cap \mathcal{TA}_{\mu}(\mathcal{N}) \cap L$, where P is the $\mathbf{S} \times \mathbf{X}$-property asserting that the $\mathbf{x}$-component of the state has the correct initial value and is not changed by the environment. The correct initial value of the state's $\mathbf{x}$-component is specified by an initial $\mathbf{S}\times\mathbf{X}$-predicate $I_{\mathbf{x}}$. (Remember that the initial value of the $\mathbf{S}$-component is described by the environment specification.) The assertion that the environment leaves the $\mathbf{x}$-component unchanged is $\mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{x}})$, where $\mathcal{U}_{\mathbf{x}}$ is the next-state relation consisting of all pairs $((s,x), (s',x'))$ such that $x = x'$. The program is then the property

$$\exists \mathbf{x} : I_{\mathbf{x}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{x}}) \cap \mathcal{TA}_{\mu}(\mathcal{N}) \cap L \tag{2}$$

Since we want the program to be machine-realizable, it is natural to ask under what conditions the specification (2) is machine-realizable. Machine-realizability is defined for a pair of properties (M, P), where M is the program's safety property and P is the complete specification, which in this case equals (2). We expect the safety property M to be

$$\exists \mathbf{x} : I_{\mathbf{x}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{x}}) \cap \mathcal{TA}_{\mu}(\mathcal{N}) \tag{3}$$

This is not always a safety property, but it turns out to be a safety property for ordinary specifications written in a "reasonable" way—meaning that the next-state relation is not using the internal state component $\mathbf{x}$ to encode progress properties. For the precise condition under which (3) is a safety property, see Proposition 2 of [AL91]. A sufficient condition for (M, P) to be μ-machine-realizable is given by the following result.

Proposition 10 *Let μ be an agent set, let $\mathbf{x}$ be the projection function from $\mathbf{S} \times \mathbf{X}$ to $\mathbf{X}$, and let $I_{\mathbf{x}}$ be an $\mathbf{S} \times \mathbf{X}$-predicate, $\mathcal{N}$ a next-state relation on $\mathbf{S} \times \mathbf{X}$, and L an $\mathbf{S} \times \mathbf{X}$-property. Let M equal* (3) *and let P equal* (2)*. Assume that:*

- *(a) For all $s \in \mathbf{S}$ there exists $x \in \mathbf{X}$ such that $(s, x) \in I_{\mathbf{x}}$.*
- *(b) The pair $(\mathcal{TA}_{\mu}(\mathcal{N}),\ \mathcal{TA}_{\mu}(\mathcal{N}) \cap (I_{\mathbf{x}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{x}}) \Rightarrow L))$ is μ-machine-realizable.*
- *(c) M is a safety property.*

Then (M, P) is μ-machine-realizable.

This proposition remains valid if, in hypothesis (b), $\mathcal{TA}_{\mu}(\mathcal{N})$ is replaced by $(I_{\mathbf{x}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{x}})) \mathrel{-\!\!\triangleright} \mathcal{TA}_{\mu}(\mathcal{N})$, which equals $\mathcal{R}_{\mu}(\mathcal{U}_{\mathbf{x}} \Rightarrow \mathcal{TA}_{\mu}(\mathcal{N}))$ by Proposition 8.

4.3 The Normal Form of a Specification

The specification of a system is written as a property of the form $E \Rightarrow M$, asserting that the system guarantees property M under the assumption that the environment satisfies property E. In the transition-axiom approach [Lam83a, Lam89], E and M are written as abstract partial programs, using next-state relations and progress properties. Since the environment makes the first move in our realization game, the initial predicate must be included with E; the abstract program M has no initial predicate—except on its internal, hidden state. (Intuitively, we are assuming that the system has control of the initial values only of its internal state, not of the externally visible state.) We therefore write our specification in the canonical form

$$I \cap E_S \cap E_L \;\Rightarrow\; M_S \cap M_L \tag{4}$$

where I is an initial predicate, E_S is a safety property constraining only $\neg\mu$, and M_S is a safety property constraining only μ.

If the system property M were written as an executable program, then we would expect the pair $(M_S, M_S \cap M_L)$ to be machine-realizable. However, M is an abstract program that is meant to specify *what* the system is allowed to do, not *how* it does it. Requiring the abstract program to be executable in practice—that is, capable of being transformed into executable code by a real compiler—is too restrictive, leading to overly complex and overly restrictive specifications. It is not clear whether requiring the abstract program to be executable in principle—that is, to be machine-realizable—is too restrictive. If $(M_S,\, M_S \cap M_L)$ is not machine-realizable, then it allows behaviors that cannot be achieved in practice. Most of the specifications we have seen are machine-realizable. But allowing unachievable behaviors causes no harm, as long as the specification is realizable. Allowing some unachievable behaviors may yield a simpler specification. For example, the simplicity of the specification of a serializable database in [Lam89] results from its not being machine-closed, hence not machine-realizable. We have too little experience writing specifications to know if this example is an anomaly or if others will arise. We therefore do not assume machine-realizability of the pair $(M_S,\, M_S \cap M_L)$.

The situation is different for the environment property E. Progress assumptions about the environment seem to be unusual. A specification usually requires that the system eventually do something after the environment has taken some action, but seldom does it assume that the environment must take that action. Thus, E_L should generally be identically true, so the pair (E_S, E_S) will be $\neg\mu$-machine-realizable if E_S constrains at most $\neg\mu$. In a transition-axiom specification, E_S has the form $\mathcal{TA}_{\neg\mu}(\mathcal{N})$, which does constrain at most $\neg\mu$.

Even if a specification does include a nontrivial progress assumption E_L about the environment, we believe that it may be reasonable to require the pair $(E_S, E_S \cap E_L)$ to be $\neg\mu$-machine-realizable. The intent of the specification $E \Rightarrow M$ is that the system should win the realization game by making M true, not by making E false. The machine-realizability condition means that so long as the environment maintains E_S, it can ensure that $E_S \cap E_L$ will be true; hence, the system can never win by forcing E to be false. A specification in which $(E_S, E_S \cap E_L)$ is not $\neg\mu$-machine-realizable seems likely to be incorrect, in the sense that it does not capture the intent of its author.

If the environment assumption is machine-realizable, then there is no need for an environment progress assumption because the property E_L can be incorporated into the system's progress property. This is stated formally by the following theorem.

Theorem 1 *If I is a state predicate, $(E_S, E_S \cap E_L)$ is $\neg\mu$-machine-realizable, M_S is a safety property, and M_L is any property, then*

$$I \cap E_S \cap E_L \;\Rightarrow\; M_S \cap M_L$$

and

$$I \cap E_S \;\Rightarrow\; M_S \cap (E_L \Rightarrow M_L)$$

are μ-equirealizable.

The abstract programs describing the system and the environment may contain hidden, internal state components, in which case the specification involves existential quantification. We now consider how Theorem 1 can be applied in the presence of quantification.

Since environment specifications tend to be simple, we suspect that variables internal to the environment can usually be confined to E_S, allowing E to be written as $(\exists \mathbf{x} : E_S) \cap E_L$, so the theorem can be applied. In any case, the following approach can always be used to eliminate existential quantification from E. The laws of ordinary predicate logic imply that, if x is not free in M or P, then $P \Rightarrow ((\exists x : E) \Rightarrow M)$ is equivalent to $P \Rightarrow \forall x : (E \Rightarrow M)$, which in turn is valid iff $P \Rightarrow (E \Rightarrow M)$ is valid. Similar reasoning about quantification over state components allows us to replace $(\exists \mathbf{x} : E) \Rightarrow M$ by $E \Rightarrow M$, if we require that no implementation P use $\mathbf{x}$. (Implementation is discussed in Section 5.2.)

Existential quantification in the system's description M is handled by the following generalization of Theorem 1, in which the $\mathbf{S}$-predicate E_L is identified with the $\mathbf{S} \times \mathbf{X}$-predicate $\Pi_{\mathbf{S}}^{-1}(E_L)$.

Corollary *Let μ be any agent set, let $\mathbf{x}$ be the projection function from $\mathbf{S} \times \mathbf{X}$ to $\mathbf{X}$, let I be an $\mathbf{S}$-predicate, let $(E_S, E_S \cap E_L)$ be a $\neg\mu$-machine-realizable pair of $\mathbf{S}$-properties, and let M_S and M_L be $\mathbf{S} \times \mathbf{X}$-properties such that $\exists \mathbf{x} : M_S$ is a safety property. Then*

$$I \cap E_S \cap E_L \;\Rightarrow\; \exists \mathbf{x} : M_S \cap M_L$$

and

$$I \cap E_S \;\Rightarrow\; \exists \mathbf{x} : M_S \cap (E_L \Rightarrow M_L)$$

are μ-equirealizable.

4.4 An Overly Normal Form

Theorem 1 permits us to take a specification of the form (4) and move the environment's progress property to the right of the implication. But, can we always write the specification in the form (4) in the first place? The answer is that not only can we, but we don't even need the left-hand side of the implication. Propositions 5 and 7 imply that the realizable part of any realizable property P can be written as $M_S \cap M_L$, where M_S is a safety property that constrains only μ. (Just take M_S to be $\overline{\mathcal{R}_\mu(P)}$ and M_L to be P.) In fact, we can choose the pair $(M_S, M_S \cap M_L)$ to be μ-machine-realizable. (The μ-machine-realizability of $(\overline{\mathcal{R}_\mu(P)}, P)$ follows from Propositions 4 and 5.)

We can go still further in finding a representation of the realizable part of a property. It can be shown that any safety property that constrains at most μ can be written in the form

$$\exists \mathbf{x} : I_{\mathbf{x}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{x}}) \cap \mathcal{TA}_\mu(\mathcal{N})$$

for some initial predicate $I_{\mathbf{x}}$ satisfying hypothesis (a) of Proposition 10 and some next-state relation $\mathcal{N}$. (This result is a simple generalization of Proposition 3 of [AL91].) Thus, the μ-realizable part of any property P can be written in the form $\exists \mathbf{x} : M_S \cap M_L$, where M_S has the form $I_{\mathbf{x}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{x}}) \cap \mathcal{TA}_\mu(\mathcal{N})$ and the pair $(\exists \mathbf{x} : M_S, \exists \mathbf{x} : M_S \cap M_L)$ is μ-machine-realizable.

The ability to write a specification in this form seems to imply that there is no need to write an explicit assumption about the environment. Why write a specification of the form $E \Rightarrow M$ when we can simply write M? One answer is that separating the environment assumption E from the guarantee M allows us to take advantage of the Composition Principle. Another answer lies in the practical matter of what the specification looks like. If we eliminate the explicit environment assumption, then that assumption appears implicitly in the property M describing the system. Instead of M describing only the behavior of the system when the environment behaves correctly, M must also allow arbitrary behavior when the environment behaves incorrectly. Eliminating E makes M too complicated, and it is not a practical alternative to writing specifications in the form $E \Rightarrow M$.

To be useful, a specification must be understandable. Theorems that assert the existence of a specification in a certain form are of no practical interest because they prove only that the specification exists, not that it is understandable. On the other hand, a result like Theorem 1 that provides a simple way to rewrite an existing specification can be of practical interest because the rewritten specification will be understandable if the original one is.

Although it seems impractical in general to write $E \Rightarrow M$ without an explicit environment assumption, it is practical if M is a safety property. In this case, Proposition 8 shows that $E \Rightarrow M$ is equivalent to the system guarantee $E \mathrel{-\!\!\triangleright} M$. In fact, this is precisely the form of specification that has been used to develop composition principles for safety properties [MC81, Pnu84].

5 Composing Specifications

Our main result is a formal statement of the Composition Principle stated informally in the introduction. Before stating this result, we must explain how specifications are composed and what it means for one specification to implement another. For convenience, we restrict our attention to the composition of two systems. The generalization to an arbitrary number of systems is straightforward, and is described after the statement of our main theorem.

5.1 The Composition of Specifications

Consider two systems $\boldsymbol{\Pi}_1$ and $\boldsymbol{\Pi}_2$, and their composition, shown schematically in Figure 3. The "wires" *inp*, *mid*, and *out* denote state components, and μ_1 and μ_2 are the systems' agent sets. If S_1 and S_2 are the specifications of the two systems, what is the specification of their composition? Each S_i is the property consisting of all histories of the universe (behaviors)

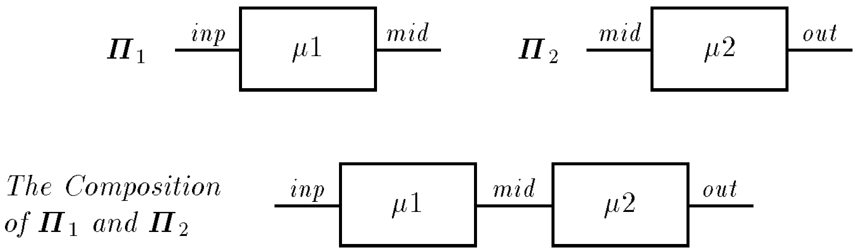

Figure 3: The composition of two systems.

in which component i functions correctly. A history of the universe is one in which both components function correctly iff it is in both S_1 and S_2. Thus, the specification of the composition of the two systems is simply $S_1 \cap S_2$. This simple semantics of composition as intersection rests on the two basic assumptions, discussed below, that Π_1 and Π_2 refer to the same states, and that μ_1 and μ_2 are disjoint.

5.1.1 Assumptions about the States

In composing the two systems Π_1 and Π_2 of Figure 3, we combined the two "wires" labeled *mid* into a single "wire". When two specifications are written as logical formulas, a state-component variable like *mid* that appears in both formulas is considered to represent the same state component. In some situations, this use of names to identify state components in the two systems is natural—for example, if the "systems" are the assignment statements $mid := inp + 1$ and $out := 2 * mid$. In other situations, there may be no connection between the names used in the two specifications, so renaming is necessary. For example, if the systems are circuits, Π_1's wire labeled *mid* might have been labeled *out*, and Π_2's wire labeled *mid* might have been labeled *inp*. In that case, the specification of the composite system in Figure 3 would be $S_1|^{out}_{mid} \cap S_2|^{inp}_{mid}$, where $S_1|^{out}_{mid}$ is obtained by substituting *mid* for *out* in the formula for S_1.

It is this kind of renaming that allows us to make do with the single operator $\cap$ for composing properties instead of having a multitude of different composition operators. For example, two programming-language statements can be combined by parallel composition or by sequential composition (";"). Simple intersection of their specifications provides parallel composition; sequential composition is obtained by first renaming components of their control states in such a way that control is at the end of one statement iff it is at the beginning of the other, then taking the intersection of the resulting specifications.

Even with the proper choice of state-component names, we can write the composition as the intersection $S_1 \cap S_2$ only if S_1 and S_2 are both $\mathbf{S}$-properties—that is, only if they have the same set of states $\mathbf{S}$. But looking at the two systems separately, we would not expect *out* to be a state component of $\boldsymbol{\Pi}_1$ or *inp* to be a state component of $\boldsymbol{\Pi}_2$. The two specifications might have to be modified to use the same set of states. This would be done by expanding S_1's state to include an *out* component, modifying S_1 to prohibit μ_1 agents from changing *out*, and allowing $\neg\mu_1$ agents to change *out* freely—making the analogous change to S_2 too.

The simplicity of representing all forms of composition as intersection is therefore somewhat illusory. We need renaming and state expansion as well. (By adopting the approach mentioned in Section 4.2.1 of having a single universal set of states, state expansion can be avoided at the expense of additional renaming.) Moreover, we might want some state components of the composed system to be hidden—for example, the component *mid* in Figure 3. This requires the use of existential quantification, as described in Section 4.2.2. Still, we feel that the ability to reduce composition to the well-understood operation of intersection—or, in the corresponding logical view, to conjunction—is a significant benefit of our approach.

5.1.2 Assumptions about the Agents

In drawing Figure 3, we have made a subtle assumption about the agent sets μ_1 and μ_2. Suppose we want to compose two copies of the $\boldsymbol{\Pi}_1$ without renaming, so the *inp* state components of the two copies would be identified (the two *inp* "wires" would be connected), as would the *mid* state components. The discussion so far might lead one to write the resulting specification as $S_1 \cap S_1$. But this is obviously wrong, since $S_1 \cap S_1$ equals S_1. The simple intersection of S_1 with itself, without renaming, yields a specification of system $\boldsymbol{\Pi}_1$, not of the composition of two separate copies of $\boldsymbol{\Pi}_1$.

A property S specifies what it means for a particular agent set μ to perform correctly. Making a separate copy of S means replacing μ by a different agent set. Let $S|^{\mu}_{\mu_i}$ denote the property obtained by substituting μ_i for μ in the formula describing S. The property $S|^{\mu}_{\mu_1} \cap S|^{\mu}_{\mu_2}$ specifies a system in which the agent sets μ_1 and μ_2 each behave like the agent set μ in the specification S—in other words, a system in which each μ_i is a separate copy of the original system μ.

By drawing separate, nonoverlapping boxes for $\boldsymbol{\Pi}_1$ and $\boldsymbol{\Pi}_2$ in Figure 3, we have tacitly assumed that their agent sets μ_1 and μ_2 are disjoint. As we have seen in the extreme case when S_1 equals S_2, the intersection $S_1 \cap S_2$ does not represent the expected composition of separate systems unless $\mu_1 \cap \mu_2$ is the empty set of agents.

5.2 Implementing One Specification by Another

5.2.1 Definition

A system's specification S describes the set of all behaviors in which the system is considered to behave correctly. For a system specified by S' to satisfy specification S, every behavior it allows must be in S. Thus, the system specified by S' satisfies the specification S if $S' \subseteq S$. Eliminating the phrase "the system specified by", we can say that specification S' implements S if $S' \subseteq S$.

While sufficient, the condition $S' \subseteq S$ is stronger than strictly necessary for S' to implement S. We view S' as a prescription for building an implementation, and we say that S' implements S iff every real system built according to the specification S' satisfies S. It is not necessary for every behavior in S' to be in S, just for every behavior that can be generated by a real implementation of S' to be in S. The set of behaviors that can be generated by a real implementation of S' is included in the realizable part of S', so we define S' implements S to mean $\mathcal{R}_\mu(S') \subseteq S$.

We expect "implements" to be transitive, meaning that if S'' implements S', and S' implements S, then S'' implements S. Proving transitivity requires showing that $\mathcal{R}_\mu(S'') \subseteq S'$ and $\mathcal{R}_\mu(S') \subseteq S$ imply $\mathcal{R}_\mu(S'') \subseteq S$. This implication is valid because, by Propositions 3 and 4, $\mathcal{R}_\mu(S'') \subseteq S'$ implies $\mathcal{R}_\mu(S'') \subseteq \mathcal{R}_\mu(S')$.

We now return to the composition of systems. Let S_1 and S_2 be specifications of systems with agent sets μ_1 and μ_2, respectively. Any real implementation that satisfies S_i will satisfy $\mathcal{R}_{\mu_i}(S_i)$, so combining an implementation of S_1 with an implementation of S_2 produces a system whose set of behaviors is contained in $\mathcal{R}_{\mu_1}(S_1) \cap \mathcal{R}_{\mu_2}(S_2)$. Thus, to prove that the composition of a system specified by S_1 and one specified by S_2 implements a specification S, it suffices to prove

$$\mathcal{R}_{\mu_1}(S_1) \cap \mathcal{R}_{\mu_2}(S_2) \;\subseteq\; S \tag{5}$$

If (5) holds, then the following proposition allows us to infer the stronger result $\mathcal{R}_{\mu_1}(S_1) \cap \mathcal{R}_{\mu_2}(S_2) \subseteq \mathcal{R}_{\mu_1 \cup \mu_2}(S)$.

Proposition 11 *For any disjoint pair of agent sets μ_1 and μ_2, and any properties P_1 and P_2, the property $\mathcal{R}_{\mu_1}(P_1) \cap \mathcal{R}_{\mu_2}(P_2)$ is $\mu_1 \cup \mu_2$-receptive.*

Proposition 11 implies that $\mathcal{R}_{\mu_1}(S_1) \cap \mathcal{R}_{\mu_2}(S_2) \subseteq \mathcal{R}_{\mu_1 \cup \mu_2}(S_1 \cap S_2)$. This in turn implies that condition (5) is weaker than $\mathcal{R}_{\mu_1 \cup \mu_2}(S_1 \cap S_2) \subseteq S$, which is what we would have to prove to show that $S_1 \cap S_2$ implements S.

The hypothesis that μ_1 and μ_2 are disjoint is necessary in Proposition 11. In particular, the conclusion does not hold if $\mu_1 = \mu_2$, because the intersection of two μ-receptive properties is not necessarily μ-receptive.

5.2.2 Proving That One Specification Implements Another

We now comment briefly on how one can prove in practice that a specification S' of the form $E' \Rightarrow M'$ implements a specification S of the form $E \Rightarrow M$. If S' is not μ-receptive (equal to its realizable part), then deriving an explicit formula for $\mathcal{R}_\mu(S')$ is likely to be very difficult. (If it were easy, then we would have written $\mathcal{R}_\mu(S')$ instead of S' in the first place.) Therefore, unless we can apply some general theorem—like Theorem 2 of Section 5.3 below—to prove that S' implements S, we will have to prove that $S' \subseteq S$.

Specification S' has environment assumption E', while S has environment assumption E. If the system specified by S' is to satisfy the specification S, it must do so assuming only that the environment satisfies E. Therefore, E' must be equal to or weaker than E—that is, we must have $E \subseteq E'$. Since $E \subseteq E'$ implies $(E' \Rightarrow M') \subseteq (E \Rightarrow M')$, if the implementation satisfies $E' \Rightarrow M'$ then it also satisfies $E \Rightarrow M'$. Therefore, it suffices to prove $(E \Rightarrow M) \subseteq (E \Rightarrow M')$.

By elementary set theory, $(E \Rightarrow M') \subseteq (E \Rightarrow M)$ is equivalent to $E \cap M' \subseteq E \cap M$.[1] Whereas $E \Rightarrow M$ consists of all behaviors in which the system behaves correctly in the face of arbitrary environment behavior, $E \cap M$ consists of only those behaviors in which both the environment and system behave correctly. In the transition-axiom approach, E is an abstract partial program describing the environment and M is an abstract partial program describing the system, so $E \cap M$ defines the complete program obtained by composing these two partial programs. Similarly, $E \cap M'$ describes a

[1] This equivalence was pointed out to us by Amir Pnueli.

complete program. Therefore, proving $E \cap M' \subseteq E \cap M$ requires proving that one complete program implements another.

Proving that one program implements another is a problem that has been addressed extensively in earlier work. The basic transition-axiom approach is described in [Lam89], and a formal basis along with a completeness result can be found in [AL91]. We briefly sketch this approach.

The specification $E \cap M$ can be written in the form

$$\exists \mathbf{x} : I \cap \mathcal{TA}_{\neg\mu}(\mathcal{N}_E) \cap \mathcal{TA}_{\mu}(\mathcal{N}_M) \cap L$$

where I is an initial predicate, $\mathcal{N}_E$ and $\mathcal{N}_M$ are next-state relations describing the environment and system actions, respectively, and L is a progress property—all with set of states $\mathbf{S} \times \mathbf{X}$. (Here, $\mathbf{X}$ consists of the system's internal state components; as we observed in Section 4.3, we can make the environment's internal variables visible.) We can write I as a logical formula on the state variables, $\mathcal{N}_E$ and $\mathcal{N}_M$ as relations between old and new state values, and L as a formula in some temporal logic. Similarly, $E \cap M'$ can be written in the form $\exists \mathbf{y} : I' \cap \mathcal{TA}_{\neg\mu}(\mathcal{N}'_E) \cap \mathcal{TA}_{\mu}(\mathcal{N}'_M) \cap L'$, with a set of internal states $\mathbf{Y}$. Moreover, $\mathcal{N}_E$ and $\mathcal{N}'_E$ will be essentially the same relations, depending only on the externally visible state (including the environment's internal state components). To prove that $E \cap M'$ implements $E \cap M$, we construct a *refinement mapping* f from $\mathbf{S} \times \mathbf{Y}$ to $\mathbf{S} \times \mathbf{X}$ that satisfies the following four conditions.

1. *f preserves the* $\mathbf{S}$*-component.* In other words, for all $(s, y) \in \mathbf{S} \times \mathbf{Y}$, there is some $x \in \mathbf{X}$ such that $f(s, y) = (s, x)$.

   In practice, a set of states is defined by a collection of state components. Let $e_1, \ldots, e_m$ denote the components defining $\mathbf{S}$, so an element s of $\mathbf{S}$ is an m-tuple $(e_1(s), \ldots, e_m(s))$; let $x_1, \ldots, x_n$ and $y_1, \ldots, y_p$ denote the similar components defining $\mathbf{X}$ and $\mathbf{Y}$. To specify the refinement mapping f, one must define functions $f_1, \ldots, f_n$ such that $f(s, y) = (s, (f_1(s, y), \ldots, f_n(s, y)))$. The f_j can be described by formulas having the components e_i and y_k as free variables. For example, the formula $e_1 + 4y_2$ denotes the function g such that $g(s, y) = e_1(s) + 4y_2(y)$.

2. *f takes initial states to initial states.* The formal condition is $f(I') \subseteq I$.

   To explain what this condition means in practice, we first make the following definition. For any formula H with free variables $e_1, \ldots, e_m$

and $x_1, \ldots, x_n$, define $f^*(H)$ to be the formula obtained by substituting f_j for x_j, for $j = 1, \ldots, n$. This defines $f^*(H)$ to be a formula with free variables $e_1, \ldots, e_m$ and $y_1, \ldots, y_p$. The semantic condition $f(I') \subseteq I$ is expressed in the logical framework as $\models I' \Rightarrow f^*(I)$, which is a formula "about" the implementation. In most cases, this condition is easy to check.

3. *f maps $\mathcal{N}'_M$ steps into $\mathcal{N}_M$ steps or stuttering steps.* Formally, we require that if (s, y) is any state reachable from a state in I' by a sequence of $\mathcal{N}'_E$ and $\mathcal{N}'_M$ steps, then $((s, y), (t, z)) \in \mathcal{N}'_M$ implies $(f(s, y), f(t, z)) \in \mathcal{N}_M$ or $f(s, y) = f(t, z)$.

   In practice, verifying this condition involves finding an $\mathbf{S} \times \mathbf{Y}$-predicate P such that $I \subseteq P$ and P is left invariant by $\mathcal{N}'_E$ and $\mathcal{N}'_M$, meaning that $(s, y) \in P$ and $((s, y), (t, z)) \in \mathcal{N}'_E \cup \mathcal{N}'_M$ imply $(t, z) \in P$. One then proves $old.P \wedge \mathcal{N}'_M \Rightarrow f^*(\mathcal{N}_M \vee \mathcal{I})$, where $old.P$ is the formula asserting that P is true in the first state of a step, and $\mathcal{I}$ is the identity relation. Finding an invariant P and proving its invariance is exactly what one does in a proof by the Owicki-Gries method [LS84b, OG76], so the method for proving this condition generalizes the standard method for proving invariance properties of concurrent programs.

4. *f maps behaviors that satisfy $I' \cap \mathcal{TA}_{\neg\mu}(\mathcal{N}'_E) \cap \mathcal{TA}_{\mu}(\mathcal{N}'_M) \cap L'$ into behaviors that satisfy L.* The formal condition is $f(I' \cap \mathcal{TA}_{\neg\mu}(\mathcal{N}'_E) \cap \mathcal{TA}_{\mu}(\mathcal{N}'_M) \cap L') \subseteq L$.

   Translated into the logical framework, the formula to be verified becomes $I' \wedge \mathcal{TA}_{\neg\mu}(\mathcal{N}'_E) \wedge \mathcal{TA}_{\mu}(\mathcal{N}'_M) \wedge L' \Rightarrow f^*(L)$. This formula asserts that the abstract program described by $I' \wedge \mathcal{TA}_{\neg\mu}(\mathcal{N}'_E) \wedge \mathcal{TA}_{\mu}(\mathcal{N}'_M) \wedge L'$ satisfies the property $f^*(L)$, which is generally a liveness property. Thus, verification of this condition is tantamount to proving that a program satisfies a liveness property, which can be done with the method of [OL82] when L and L' are expressed as temporal logic formulas.

Condition 3 is weaker in two ways than the corresponding condition R3 in the definition of a refinement mapping in [AL91]. First, condition 3 applies only to μ steps, while condition R3 applies to all steps. The weaker condition is sufficient because $\neg\mu$ steps, which are taken by the environment, are essentially the same in both $E \cap M'$ and $E \cap M$. (The formalism of [AL91] did not include agents and made no distinction between system and environment steps.) Second, condition 3 applies only to steps taken from a reachable state, while R3 applies to steps taken from any state. The weaker condition was not needed in [AL91], where history variables were used to eliminate unreachable states.

Theorem 2 of [AL91] asserts the existence of a refinement mapping under certain reasonable assumptions about the specifications, providing a completeness theorem for the proof method. In general, obtaining the refinement mapping may require adding two auxiliary variables to the lower-level specification: a history variable used to record past actions, and a prophecy variable used to predict future ones. Our limited experience indicates that prophecy variables are almost never needed and, with condition 3 rather than R3, history variables are seldom needed. Although our experience with this method for verifying concurrent systems is limited, we have good reason to believe that these mappings can be constructed in practice, because refinement mappings are essentially abstraction functions of the kind that have been used for years to prove that one data type implements another [Hoa72].

5.3 The Main Theorem

5.3.1 A Precise Statement of the Composition Principle

Having discussed composition and implementation, we come to the problem of proving that the composition of specifications S_1 and S_2 implements a specification S. As we observed in (5) of Section 5.2, we must prove $\mathcal{R}_{\mu_1}(S_1) \cap \mathcal{R}_{\mu_2}(S_2) \subseteq S$. One might attempt to prove this with the refinement-mapping method of Section 5.2.2. Since we cannot expect to construct the realizable part of a specification, we would have to prove the stronger result that $S_1 \cap S_2$ implements S. However, S has the form $E \Rightarrow M$ and each S_i has the form $E_i \Rightarrow M_i$. The refinement-mapping method proves that a specification of the form $E \Rightarrow M'$ implements $E \Rightarrow M$, but $S_1 \cap S_2$ is not in this form. A simple refinement mapping won't work; we need the Composition Principle.

We now restate the Composition Principle, for the case $n = 2$, in terms of our formal definitions. The principle's premises are that system $\boldsymbol{\Pi}$ is the composition of systems $\boldsymbol{\Pi}_1$ and $\boldsymbol{\Pi}_2$, the specification of $\boldsymbol{\Pi}$ is $E \Rightarrow M$, and the specification of each $\boldsymbol{\Pi}_i$ is $E_i \Rightarrow M_i$. As we have already indicated, E, E_1, and E_2 must be safety properties. Also needed are some additional assumptions that are natural consequences of our method of writing specifications—assumptions that we disregard for now, but add as hypotheses of the theorem and discuss afterwards. The hypotheses of the principle consist of three conditions:

1. *$\boldsymbol{\Pi}$ guarantees M if each component $\boldsymbol{\Pi}_i$ guarantees M_i.*

   Formally, this condition asserts that $M_1 \cap M_2 \subseteq M$. It can be satisfied automatically by taking M to be $M_1 \cap M_2$. We can therefore simplify the Composition Principle by eliminating M, and letting the conclusion assert that $\boldsymbol{\Pi}$ satisfies $E \Rightarrow M_1 \cap M_2$. To show that $\boldsymbol{\Pi}$ satisfies the specification $E \Rightarrow M$, one proves that $E \Rightarrow M_1 \cap M_2$ implements $E \Rightarrow M$, using the refinement-mapping method described in Section 5.2.2.

2. *The environment assumption E_i of each component $\boldsymbol{\Pi}_i$ is satisfied if the environment of $\boldsymbol{\Pi}$ satisfies E and every $\boldsymbol{\Pi}_j$ satisfies M_j.*

   This condition asserts that $E \cap M_1 \cap M_2 \subseteq E_1$ and $E \cap M_1 \cap M_2 \subseteq E_2$, two assertions that can be combined as $E \cap M_1 \cap M_2 \subseteq E_1 \cap E_2$.

3. *Every component $\boldsymbol{\Pi}_i$ guarantees M_i under environment assumption E_i.*

   This condition simply asserts that each component $\boldsymbol{\Pi}_i$ satisfies its specification $E_i \Rightarrow M_i$.

The Composition Principle's conclusion asserts that $\boldsymbol{\Pi}$ satisfies the specification $E \Rightarrow M_1 \cap M_2$. (Remember that we have replaced M by $M_1 \cap M_2$.) When the principle is formulated in terms of specifications rather than systems, condition 3 disappears and the conclusion states that the composition of the components' specifications implements the system's specification. The Composition Principle then becomes the proof rule:

$$\frac{E \cap M_1 \cap M_2 \subseteq E_1 \cap E_2}{\mathcal{R}_{\mu_1}(E_1 \Rightarrow M_1) \cap \mathcal{R}_{\mu_2}(E_2 \Rightarrow M_2) \;\subseteq\; E \Rightarrow M_1 \cap M_2} \tag{6}$$

Unfortunately, this rule is not valid. To obtain a valid rule, we must replace its hypothesis with a stronger one.

Rule (6) appears unreasonably circular because it allows one to assume M_i in proving the environment assumption E_i that is necessary for component $\boldsymbol{\Pi}_i$ to guarantee M_i. This suggests that we strengthen the hypothesis by disallowing the use of M_i in proving E_i, obtaining the rule:

$$\frac{E \cap M_2 \subseteq E_1, \quad E \cap M_1 \subseteq E_2}{\mathcal{R}_{\mu_1}(E_1 \Rightarrow M_1) \cap \mathcal{R}_{\mu_2}(E_2 \Rightarrow M_2) \;\subseteq\; E \Rightarrow M_1 \cap M_2} \tag{7}$$

This rule is indeed valid. However, it would be wrong to attribute the invalidity of (6) to simple circularity. Rule (7) is also circular, and it would be incorrect without the additional assumptions that we have been ignoring. For example, suppose we could take $E_1 = E_2 = M_1 = M_2 = P$ for some safety property P. Both hypotheses of (7) then reduce to the tautology $E \cap P \subseteq P$; and each $E_i \Rightarrow M_i$ becomes $P \Rightarrow P$, an identically true specification satisfied by any system. Rule (7) would then yield the ridiculous conclusion that the composition of any two systems satisfies the specification $E \Rightarrow P$.

Not only is (7) valid despite its circularity, but there is an even stronger valid rule that looks just as circular as (6). The way to strengthen (7) is suggested by a closer examination of its hypotheses. The first hypothesis asserts that E and M_2 imply E_1. Property E_1 is assumed to be a safety property, and any safety property that is implied by M_2 is implied by $\overline{M_2}$. We would therefore expect $E \cap M_2$ to imply E_1 only if $E \cap \overline{M_2}$ does. Similarly, $E \cap M_1$ should imply E_2 only if $E \cap \overline{M_1}$ does. Proposition 12 shows that the following inference rules are indeed valid—again, under certain natural assumptions.

$$\frac{E \cap M_2 \subseteq E_1}{E \cap \overline{M_2} \subseteq E_1} \qquad \frac{E \cap M_1 \subseteq E_2}{E \cap \overline{M_1} \subseteq E_2} \tag{8}$$

We can thus replace M_1 and M_2 by their closures in the hypotheses of (7). But we can do even more. In the hypothesis, we can actually assume both $\overline{M_1}$ and $\overline{M_2}$ when proving E_1 and E_2. In other words, rule (6) is valid if, in the hypothesis, we replace M_1 and M_2 by $\overline{M_1}$ and $\overline{M_2}$. Thus, one can assume $\overline{M_i}$ when proving the assumption E_i that is necessary for $\boldsymbol{\Pi}_i$ to guarantee M_i.

We now state our precise results. The hypotheses of the proposition and the theorem are discussed later. The proposition asserts the first rule of (8); the second rule is obtained by the obvious substitutions. In the theorem, we have strengthened the proof rule's conclusion by replacing $E \Rightarrow M_1 \cap M_2$ with its realizable part.

Proposition 12 *If μ_1, μ_2, and $\mu_1 \cup \mu_2$ are agent sets and E, E_1, and M_2 are properties such that:*

1. *$E = I \cap P$ where*
   - *(a) I is a state predicate.*
   - *(b) P is a safety property that constrains at most $\neg(\mu_1 \cup \mu_2)$.*
2. *E_1 is a safety property.*

3. $\mu_1 \cap \mu_2 = \emptyset$

4. M_2 *is a* μ_2*-abstract property.*

Then the rule of inference

$$\frac{E \cap M_2 \subseteq E_1}{E \cap \overline{M_2} \subseteq E_1}$$

is sound.

Theorem 2 *If* μ_1*,* μ_2*, and* $\mu_1 \cup \mu_2$ *are agent sets and* E*,* E_1*,* E_2*,* M_1*, and* M_2 *are properties such that:*

1. $E = I \cap P$*,* $E_1 = I_1 \cap P_1$*, and* $E_2 = I_2 \cap P_2$*, where*

   (a) I*,* I_1*, and* I_2 *are state predicates.*

   (b) P*,* P_1*, and* P_2 *are safety properties that constrain at most* $\neg(\mu_1 \cup \mu_2)$*,* $\neg\mu_1$*, and* $\neg\mu_2$*, respectively.*

2. $\overline{M_1}$ *and* $\overline{M_2}$ *constrain at most* μ_1 *and* μ_2*, respectively.*

3. $\mu_1 \cap \mu_2 = \emptyset$

Then the rule of inference

$$\frac{E \cap \overline{M_1} \cap \overline{M_2} \ \subseteq\ E_1 \cap E_2}{\mathcal{R}_{\mu_1}(E_1 \Rightarrow M_1) \cap \mathcal{R}_{\mu_2}(E_2 \Rightarrow M_2) \ \subseteq\ \mathcal{R}_{\mu_1 \cup \mu_2}(E \Rightarrow M_1 \cap M_2)}$$

is sound.

The theorem handles the composition of two systems. It has an obvious generalization to the composition of n systems, for any $n \geq 2$.

$$\frac{E \cap \overline{M_1} \cap \ldots \cap \overline{M_n} \ \subseteq\ E_1 \cap \ldots \cap E_n}{\begin{array}{l}\mathcal{R}_{\mu_1}(E_1 \Rightarrow M_1) \cap \ldots \cap \mathcal{R}_{\mu_n}(E_n \Rightarrow M_n) \\ \qquad\qquad \subseteq\ \mathcal{R}_{\mu_1 \cup \ldots \cup \mu_n}(E \Rightarrow M_1 \cap \ldots \cap M_n)\end{array}}$$

This rule can be derived from the theorem by using Proposition 11.

5.3.2 The Hypotheses of the Theorem and Proposition

We now discuss the theorem's three numbered hypotheses—which imply the first three hypotheses of the proposition—and the proposition's fourth hypothesis.

1. It is not hard to show that any safety property E' can be written as $I' \cap P'$, where I' is a state predicate and P' is a safety property that constrains at most ν, for some set of agents ν. If E' specifies the environment of a system with agent set μ, then ν should equal $\neg\mu$. Therefore, hypothesis 1 will be satisfied if the environment assumptions E, E_1, and E_2 are safety properties. Theorem 1 allows us to rewrite a specification so its environment assumption is a safety property.

   Observe that a system implemented by components with agent sets μ_1 and μ_2 should have $\mu_1 \cup \mu_2$ as its agent set. But, the higher-level specification $E \Rightarrow M$ we are ultimately trying to verify may be written in terms of an agent set μ rather than $\mu_1 \cup \mu_2$. In this case, we must perform a renaming operation, substituting $\mu_1 \cup \mu_2$ for μ, before applying the theorem.

2. In the transition-axiom approach, each M_i has the form

   $$\exists \mathbf{x} : I_{\mathbf{x}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{x}}) \cap \mathcal{TA}_{\mu}(\mathcal{N}) \cap L$$

   and we expect $\overline{M_i}$ to equal $\exists \mathbf{x} : I_{\mathbf{x}} \cap \mathcal{TA}_{\neg\mu_i}(\mathcal{U}_{\mathbf{x}}) \cap \mathcal{TA}_{\mu_i}(\mathcal{N})$, in which case hypothesis 2 is satisfied.

3. As we mentioned in Section 5.1.2, this hypothesis means that the two components are distinct. They need be distinct only at the current level of abstraction; their implementations could contain common parts. For example, the two components might specify distinct program procedures, while their implementations both invoke a common subprocedure. We can consider the subprocedure to be executed by different agents depending upon which procedure invoked it. Alternatively, we can generalize our notion of implementation to allow renaming of agents. In practice, this hypothesis seems to be a petty nuisance of the formalism, not a real concern.

4. When we write M_2 directly, either as an abstract program or by any sort of logical formula, individual agents are not mentioned. The only

reference to agents is through the symbol "μ_2", so M_2 is automatically μ_2-abstract.

5.3.3 The Hypotheses of the Proof Rule

We now show how one verifies the hypothesis $E \cap \overline{M_1} \cap \overline{M_2} \subseteq E_1 \cap E_2$ of the theorem's proof rule, using the systems $\boldsymbol{\Pi}_1$ and $\boldsymbol{\Pi}_2$ of Figure 3 as a generic example.

Each of the "wires" *inp*, *mid*, and *out* will have an associated protocol that the systems on its two ends are expected to obey. For each wire w, let I^w be the initial condition for the wire, let L^w_μ be the property asserting that the agent set μ correctly executes the protocol for the system on the left side of the wire, and let R^w_μ be the corresponding property for the right side of the wire.

For example, suppose the state component *mid* consists of a register r and two booleans *rdy* and *ack*, and that the following popular hardware protocol is used to pass values from a sender on the left to a receiver on the right. (Initially, the values of *rdy* and *ack* are equal.)

send: **begin loop**	*receive*: **begin loop**
await $rdy = ack$;	**await** $rdy \neq ack$;
write r;	**read** r;
$rdy := \neg rdy$	$ack := \neg ack$
end loop	**end loop**

Let $\mathcal{N}_{send}$ and $\mathcal{N}_{receive}$ be the next-state relations of the sender's and receiver's programs. For this protocol, I^{mid} is the initial predicate $rdy = ack$, the property L^{mid}_μ equals $\mathcal{TA}_\mu(\mathcal{N}_{send})$, and R^{mid}_μ equals $\mathcal{TA}_\mu(\mathcal{N}_{receive})$. Data is properly transferred from $\boldsymbol{\Pi}_1$ to $\boldsymbol{\Pi}_2$ across *mid* in every behavior satisfying the property $I^{mid} \cap L^{mid}_{\mu_1} \cap R^{mid}_{\mu_2}$.

We will not assume any particular protocols for *inp*, *mid*, and *out*. However, we can ignore any liveness properties the protocols might require, since these properties cannot appear in the environment assumptions. Therefore, we assume that L^w_μ and R^w_μ are safety properties, for each wire w.

In addition to specifying the mechanism by which values are sent over wire w, the protocol properties L^w_μ and R^w_μ can also specify what values are sent. Thus, it is reasonable to suppose that these protocol properties include any assumptions that a system makes about its environment. A system's environment assumption then asserts that the environment obeys its side of the protocol for each wire over which the system and environment communicate. The initial conditions for these wires must also be part of the environment assumption, since the environment is responsible for the initial values of all externally visible components. For the composition in Figure 3, we then get the following environment assumptions.

$$\begin{aligned}
E_1 &= I^{inp} \cap L^{inp}_{\neg\mu_1} \cap I^{mid} \cap R^{mid}_{\neg\mu_1} \\
E_2 &= I^{mid} \cap L^{mid}_{\neg\mu_2} \cap I^{out} \cap R^{out}_{\neg\mu_2} \\
E &= I^{inp} \cap L^{inp}_{\neg(\mu_1 \cup \mu_2)} \cap I^{mid} \cap \mathcal{TA}_{\neg(\mu_1\cup\mu_2)}(\mathcal{U}_{mid}) \cap I^{out} \cap R^{out}_{\neg(\mu_1\cup\mu_2)}
\end{aligned}$$

We have included in E the assumption that the composite system's environment does not affect *mid*.

We cannot prove the hypothesis of the theorem without knowing something about $\boldsymbol{\Pi}_1$, $\boldsymbol{\Pi}_2$, and the wires. The assumptions we will make, and their justifications, are listed below.

A1. For any wire w and agent sets ν_1 and ν_2:

(a) $\mathcal{TA}_{\nu_1}(\mathcal{U}_w) \cap L^w_{\nu_2} \subseteq L^w_{\nu_1\cup\nu_2}$

(b) $\mathcal{TA}_{\nu_1}(\mathcal{U}_w) \cap R^w_{\nu_2} \subseteq R^w_{\nu_1\cup\nu_2}$

Property $L^w_{\nu_2}$ asserts that agents in ν_2 obey the left-side protocol for wire w. Actions that do not affect the wire cannot disobey the protocol. Hence, if agents in ν_2 obey the protocol and agents in ν_1 do not affect w, then agents in $\nu_1 \cup \nu_2$ obey the protocol. Part (a) can be derived formally from three assumptions: (i) $L^w_{\nu_1\cup\nu_2}$ equals $L^w_{\nu_2}|^{\nu_2}_{\nu_1\cup\nu_2}$ (the property obtained by substituting $\nu_1 \cup \nu_2$ for ν_2 in $L^w_{\nu_2}$), (ii) $L^w_{\nu_2}$ constrains at most ν_2, and (iii) $L^w_{\nu_2}$ depends only on the w-component of the state. Part (b) has a similar justification.

A2. (a) $\overline{M_1} \subseteq \mathcal{TA}_{\mu_1}(\mathcal{U}_{out})$

(b) $\overline{M_2} \subseteq \mathcal{TA}_{\mu_2}(\mathcal{U}_{inp})$

Figure 3 assumes that $\boldsymbol{\Pi}_1$ does not affect *out* and $\boldsymbol{\Pi}_2$ does not affect *inp*. Formally, these assumptions are $M_1 \subseteq \mathcal{TA}_{\mu_1}(\mathcal{U}_{out})$ and $M_2 \subseteq \mathcal{TA}_{\mu_1}(\mathcal{U}_{inp})$, which imply A2 because $\mathcal{TA}_{\mu_1}(\mathcal{U}_{out})$ and $\mathcal{TA}_{\mu_2}(\mathcal{U}_{inp})$ are safety properties.

A3. $\overline{M_1} \cap \overline{M_2} \subseteq L^{mid}_{\mu_1} \cap R^{mid}_{\mu_2}$

For the composite system to work properly, $\boldsymbol{\Pi}_1$ and $\boldsymbol{\Pi}_2$ must cooperate to guarantee that the protocol condition $L^{mid}_{\mu_1} \cap R^{mid}_{\mu_2}$ is satisfied. Hence, $M_1 \cap M_2$ must be a subset of $L^{mid}_{\mu_1} \cap R^{mid}_{\mu_2}$. Since $L^{mid}_{\mu_1} \cap R^{mid}_{\mu_2}$ is a safety property, we expect it to contain $M_1 \cap M_2$ only if it contains $\overline{M_1} \cap \overline{M_2}$. This is an expectation, not a logical necessity. If we could always deduce A3 from $M_1 \cap M_2 \subseteq L_{\mu_1}^{mid} \cap R_{\mu_2}^{mid}$, then proof rule (6) would be valid.

With these assumptions, we can verify $E \cap \overline{M_1} \cap \overline{M_2} \subseteq E_1 \cap E_2$, the hypothesis of the theorem's proof rule. We prove that $E \cap \overline{M_1} \cap \overline{M_2}$ is a subset of E_1; proving that it is a subset of E_2 is similar. Since E_1 is the conjunction of four properties, there are four inclusions to verify.

1. $E \cap \overline{M_1} \cap \overline{M_2} \subseteq I^{inp}$
   Proof: The definition of E implies that it is a subset of I^{inp}.
2. $E \cap \overline{M_1} \cap \overline{M_2} \subseteq L_{\neg\mu_1}^{inp}$
   Proof: This is proved by the following sequence of steps.
   2.1. $\overline{M_2} \subseteq \mathcal{TA}_{\mu_2}(\mathcal{U}_{inp})$
   Proof: By A2(b).
   2.2. $E \subseteq L_{\neg(\mu_1 \cup \mu_2)}^{inp}$
   Proof: By definition of E.
   2.3. $E \cap \overline{M_2} \subseteq L_{\neg\mu_1}^{inp}$
   Proof: By 2.1, 2.2, and A1(a), substituting μ_2 for ν_1 and $\neg(\mu_1 \cup \mu_2)$ for ν_2, since the hypothesis that μ_1 and μ_2 are disjoint implies $\mu_2 \cup \neg(\mu_1 \cup \mu_2) = \neg\mu_1$.
3. $E \cap \overline{M_1} \cap \overline{M_2} \subseteq I^{mid}$
   Proof: By definition of E.
4. $E \cap \overline{M_1} \cap \overline{M_2} \subseteq R_{\neg\mu_1}^{mid}$
   4.1. $\overline{M_1} \cap \overline{M_2} \subseteq R_{\mu_2}^{mid}$
   Proof: By A3.
   4.2. $E \subseteq \mathcal{TA}_{\neg(\mu_1 \cup \mu_2)}(\mathcal{U}_{mid})$
   Proof: By definition of E.
   4.3. $E \cap \overline{M_1} \cap \overline{M_2} \subseteq R_{\neg\mu_1}^{mid}$
   Proof: By 4.1, 4.2, and A1(b), substituting $\neg(\mu_1 \cup \mu_2)$ for ν_1 and μ_2 for ν_2, using the disjointness of μ_1 and μ_2.

This completes our justification of the hypothesis $E \cap \overline{M_1} \cap \overline{M_2} \subseteq E_1 \cap E_2$ for the composition of Figure 3. It was based on assumptions derived from the figure, with no assumptions about what $\boldsymbol{\Pi}_1$ and $\boldsymbol{\Pi}_2$ are supposed to do. This example is therefore quite general, since *mid* represents all state components involved in communication between $\boldsymbol{\Pi}_1$ and $\boldsymbol{\Pi}_2$, while *inp* and *out* represent the state components with which $\boldsymbol{\Pi}_1$ and $\boldsymbol{\Pi}_2$ interact with the rest of their environments. The only real assumption implicit in the figure is that the composite system's environment does not modify any state component that is accessed by both $\boldsymbol{\Pi}_1$ and $\boldsymbol{\Pi}_2$. Removing this assumption means that communication over *mid* involves a three-party protocol, requiring an additional property T^{mid}_{μ} to be satisfied by the third party. (This would be represented pictorially by adding a third end to wire *mid* that is not connected to anything in Figure 3.) Correct transfer of data over wire *mid* then requires $L^{mid}_{\mu_1} \cap R^{mid}_{\mu_2} \cap T^{mid}_{\neg(\mu_1 \cup \mu_2)}$ to hold. Our argument can be modified to handle the more general case.

6 Concluding Remarks

We have approached the problem of composing specifications from a purely semantic point of view. A formal specification method can use a language and logic based on this semantics. Our Theorem 2 would appear as a proof rule in the logic. We have touched lightly on logical issues in our discussion, mentioning what form some logical formulas might take. Some concluding remarks about language and logic are in order.

The semantic form of our specifications suggests the general style of a specification language. Safety properties are expressed by describing a next-state relation, and progress properties are expressed either directly in some form of temporal logic, or with fairness conditions that can be translated into temporal logic.

There are obvious desiderata for a specification language: it should be expressive, readable, concise, etc. There are also more precise attributes that the specification logic must have. Clearly, we want all the sets of behaviors expressed to be properties, meaning that they are closed under stuttering-equivalence. Another simple attribute of a logic is *explicitness*, meaning that whether or not a behavior satisfies a formula F depends only on the values assumed during the behavior by the state components that are free variables of F. Explicitness is necessary if existential quantification is to have its expected meaning, but it poses a surprisingly serious constraint on how specifications are written. For example, consider a formula F that specifies the assignment statement $x := x + 1$. If this formula is to assert that executing the assignment statement does not change y, then explicitness requires that y (and every other variable that is not changed) be free variables of F. A practical language must allow one to write the formula F so that y is a free variable of F even though it does not appear in the text.

Closure under stuttering-equivalence and explicitness may seem esoteric to readers accustomed to popular, simple semantics of programs. In a typical semantics, the formula specifying a program is satisfied only by behaviors in which each step corresponds to the execution of a program action—for example, this is the natural way to write a semantics using the "next-time" temporal operator. However, composition cannot be conjunction in such a semantics. For example, consider two completely noninteracting programs, with separate sets of variables, described by formulas F and G. A behavior of their composition is obtained by interleaving actions from the two programs. But such an interleaved behavior does not satisfy F, since it contains steps that do not represent actions of that program, nor does it satisfy G. Thus, the composition of the two programs is not described by the formula $F \wedge G$. Closure under stuttering-equivalence and explicitness are needed for composition to be conjunction even in the trivial case of noninteracting programs.

Many styles of specification have been proposed, ranging from abstract axioms in a specific logic to abstract programs in a specific language. Most of these styles can be adapted to our semantics, so they can make use of our results. However, these specification styles have usually been based on a particular semantic theory, and that underlying theory might have to be modified. Thus, one can still specify properties with CSP programs, but the traditional failure-set semantics of CSP [Hoa85] would have to be revisited. We are now investigating a transition-axiom method based on the temporal logic of actions [Lam90].

Appendix: Proofs

This appendix contains the proofs of all propositions and theorems stated above. Also included are lemmas, which are used in the proofs but which are not mentioned in the main text. The proofs have been carried out to an excruciating level of detail, in a hierarchical style that is explained below. The reader may feel that we have given long, tedious proofs of obvious assertions. However, what he has not seen are the many equally obvious assertions that we discovered to be wrong only by trying to write similarly long, tedious proofs. We believe very strongly that reasoning must be carried out to this level of detail to avoid mistakes. Without these detailed proofs, we would have little confidence in the correctness of our results.

The proofs employ the following definitions and notations.

- We make all functions total by defining $f(x)$ to equal $\perp$ when x is not in the domain of f.

- If ρ is a finite behavior prefix, α an agent, and s a state, then $\rho \cdot (\alpha, s)$ is the behavior prefix obtained by concatenating $\xrightarrow{\alpha} s$ to the end of ρ.
- The length of a finite behavior prefix ρ, denoted $|\rho|$, is defined by $|s_0 \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_m} s_m| = m$.
- We extend the definition of $\rho|_m$, previously defined only for a behavior ρ, in the obvious way when ρ is a finite behavior prefix and $0 \leq m \leq |\sigma|$. (Thus $\sigma|_0$ is a prefix of length 0, consisting of a single state.)
- For a finite behavior prefix ρ, the state $\rho_{\mathbf{s}}$ is defined to equal $\mathbf{s}_{|\rho|}(\rho)$; and, when $|\rho| > 0$, the agent $\rho_{\mathbf{a}}$ is defined to equal $\mathbf{a}_{|\rho|}(\rho)$.
- A mapping f from behavior prefixes to behavior prefixes is *monotone* iff for all behavior prefixes σ and τ, if σ is a prefix of τ then $f(\sigma)$ is prefix of $f(\tau)$. Observe that if f is monotone, then $\lim_{m\to\infty} f(\sigma|_m)$ exists for any behavior σ.
- If f is a μ-strategy, then a finite behavior prefix ρ is said to *end according to* μ, f iff (i) $|\rho| = 0$, or (ii) $\rho_{\mathbf{a}} \notin \mu$, or (iii) $f(\rho|_{|\rho|-1}) = (\rho_{\mathbf{a}}, \rho_{\mathbf{s}})$. Note that a behavior τ is a μ-outcome of f iff every finite prefix of τ ends according to μ, f.
- If f is a μ-strategy, then a finite behavior prefix ρ is said to be a *partial μ-outcome* of f iff every prefix of ρ (including ρ itself) ends according to μ, f.

The proofs are written in a hierarchical style. A structured proof consists of a preamble followed by a sequence of statements, each with its own proof. A proof that uses a case split will have a separate proof for each case.

The preamble describes the assumptions that are to be made, the desired conclusion, and why this conclusion implies the result to be proved. It may also contain an informal description of the proof. The proof statement or statements that assert the preamble's desired conclusion are indicated by boxed statement numbers. The preamble is omitted if the assumptions and conclusion are obvious.

A sufficiently simple proof is not structured, being written in the customary paragraph style. Some proof statements serve only to make definitions and require no proof.

Lemma 1 *For any agent set μ, if (i) f is a μ-strategy, (ii) $\sigma \in \mathcal{O}_\mu(f)$, and (iii) $\sigma' \simeq \sigma$, then there exists a μ-strategy f' such that (iv) $\sigma' \in \mathcal{O}_\mu(f')$ and (v) every behavior in $\mathcal{O}_\mu(f')$ is stuttering-equivalent to some behavior in $\mathcal{O}_\mu(f)$.*

Proof of Lemma 1
We assume μ is an agent set, f a μ-strategy, $\sigma \in \mathcal{O}_\mu(f)$, and $\sigma' \simeq \sigma$; and we construct the required f'. We will define f' so it "tries to produce" σ' if that is still possible; otherwise it tries to act like f. This means that f' has to switch from trying to produce σ' to acting like f if the environment causes the behavior

to diverge from σ'. Our formal definition will be driven by the need for f' to make this switch smoothly. We will first define a mapping Ξ on behavior prefixes such that Ξ maps prefixes of σ' to prefixes of σ, and $\Xi(\rho)$ is stuttering-equivalent to ρ for any behavior prefix ρ. We will then define $f'(\rho)$ to equal $f(\Xi(\rho))$ if ρ is not a prefix of σ'.

1. For any finite behavior prefix ρ, define the finite behavior prefix $\Xi(\rho)$ inductively as follows.

   if $|\rho| = 0$ **then** $\Xi(\rho) = \rho$

   if $\rho = \theta \cdot (\alpha, s)$ **then if** ρ is a prefix of σ'
   then $\Xi(\rho)$ is the smallest prefix of σ that is stuttering-equivalent to ρ
   else $\Xi(\rho) = \Xi(\theta) \cdot (\alpha, s)$

2. For any finite behavior prefix ρ that is not a prefix of σ', $\Xi(\rho) = \Xi(\rho|_k) \cdot (\mathbf{a}_{k+1}(\rho), \mathbf{s}_{k+1}(\rho)) \cdots (\rho_{\mathbf{a}}, \rho_{\mathbf{s}})$, where k is the smallest natural number such that $\rho|_{k+1}$ is not a prefix of σ'.
   Proof: From 1, by a simple induction on $|\rho| - k$.
3. For any behavior τ, define $\Xi(\tau)$ as follows.

   if $\tau = \sigma'$ **then** $\Xi(\tau) = \sigma$
   else $\Xi(\tau) = \lim_{m\to\infty} \Xi(\tau|_m)$

   Then $\Xi(\tau)$ is a behavior.
   Proof: If $\tau = \sigma'$, then $\Xi(\tau)$ is a behavior because σ is. If $\tau \neq \sigma'$, then 2 implies that $\lim_{m\to\infty} \Xi(\tau|_m)$ exists and is infinite.
4. $\Xi(\tau) \simeq \tau$ for any behavior τ.
   Proof: If $\tau = \sigma'$, then the result follows from 3 and the hypothesis that $\sigma' \simeq \sigma$. If $\tau \neq \sigma'$, then 2 and 3 imply $\Xi(\tau) = \Xi(\tau|_k) \cdot (\mathbf{a}_{k+1}(\tau), \mathbf{s}_{k+1}(\tau)) \cdot (\mathbf{a}_{k+2}(\tau), \mathbf{s}_{k+2}(\tau)) \cdots$, where $\tau|_k$ is a prefix of σ' or $k = 0$. The result now follows from 1, which implies $\Xi(\tau|_k) \simeq \tau|_k$.
5. For any finite behavior prefix ρ, define $f'(\rho)$ as follows.

   if $\rho = \sigma'|_k$ **then if** $\mathbf{a}_{k+1}(\sigma') \in \mu$ **then** $f'(\rho) = (\mathbf{a}_{k+1}(\sigma'), \mathbf{s}_{k+1}(\sigma'))$
   else $f'(\rho) = \perp$

   if ρ is not a prefix of σ' **then** $f'(\rho) = f(\Xi(\rho))$

   Then f' is a μ-strategy.
   Proof: Follows from the hypothesis that f is a μ-strategy.
6. $\boxed{6.}$ $\sigma' \in \mathcal{O}_\mu(f')$
   Proof: It follows from the definition of f' that σ' is a μ-outcome of f'. It is a fair outcome because $\mathbf{a}_{k+1}(\sigma') \notin \mu$ implies $f'(\sigma'|_k) = \perp$, so if there are only finitely many μ actions in σ', then f' is undefined on infinitely many prefixes of σ'.
7. If τ is a fair μ-outcome of f', then $\Xi(\tau)$ is a fair μ-outcome of f.
   Proof: If $\tau = \sigma'$, then $\Xi(\tau) = \sigma$, and σ is a fair μ-outcome of f by hypothesis. We assume that τ is a fair μ-outcome of f' and $\tau \neq \sigma'$, and we prove that $\Xi(\tau)$ is a fair μ-outcome of f.

7.1. Choose k to be the largest natural number j such that $\tau|_j = \sigma'|_j$, or -1 if there is no such j. Let l equal $|\Xi(\tau|_k)|$ if $k \geq 0$, or -1 if $k = -1$. For all $i \geq 0$: (i) if $k \geq 0$ then $\Xi(\tau)|_{l+i} = \Xi(\tau|_k) \cdot (\mathbf{a}_{k+1}(\tau), \mathbf{s}_{k+1}(\tau)) \cdots (\mathbf{a}_{k+i}(\tau), \mathbf{s}_{k+i}(\tau))$, and (ii) if $k = -1$ then $\Xi(\tau)|_i = \tau|_i$.

Proof: The existence of k follows from the hypothesis that $\tau \neq \sigma'$. Case (i) follows by induction on i from 2 and 3. Case (ii) follows from 1, 2 (where the k of step 2 is 0), and 3.

7.2. $f(\Xi(\tau)|_{l+i}) = f'(\tau|_{k+i})$ for all $i > 0$.

Proof: By 7.1 and 5.

7.3. Every finite prefix of $\Xi(\tau)$ ends according to μ, f.

Proof: Let m be any natural number. We show that $\Xi(\tau)|_m$ ends according to μ, f. The proof is split into three cases.

<u>Case 7.3A</u>. $m \leq l$

In this case, $k \geq 0$. We have $\Xi(\tau)|_m$ is a prefix of $\Xi(\tau)|_l$, which equals $\Xi(\tau|_k)$ (by definition of l in 7.1, since $k \geq 0$), which in turn is a prefix of σ. Hence, $\Xi(\tau)|_m$ ends according to μ, f by the assumption that σ is in $\mathcal{O}_\mu(f)$.

<u>Case 7.3B</u>. $m = 1 + l$

If $m = 0$, then the result is trivial because any sequence of length 0 ends according to μ, f. Assume $m > 0$, so $m = 1 + l$ implies that $l \geq 0$, which implies $k \geq 0$. If $\mathbf{a}_m(\Xi(\tau)) \in \neg\mu$, then the result is trivial. It therefore suffices to prove $\mathbf{a}_m(\Xi(\tau)) \in \neg\mu$. Intuitively, this holds because only the environment can make the behavior diverge from σ'. Formally, we assume that $\mathbf{a}_m(\Xi(\tau)) \in \mu$ and prove $\tau|_{k+1} = \sigma'|_{k+1}$, which contradicts the definition of k in 7.1.

7.3B.1. $\mathbf{a}_m(\Xi(\tau)) = \mathbf{a}_{k+1}(\tau)$

Proof: By 7.1 and the assumption that $m = 1 + l$.

7.3B.2. $f'(\tau|_k) = (\mathbf{a}_{k+1}(\tau), \mathbf{s}_{k+1}(\tau))$

Proof: By 7.3B.1 and the assumptions that $\mathbf{a}_m(\Xi(\tau)) \in \mu$ and that τ is a μ-outcome of f'.

7.3B.3. $f'(\tau|_k) = (\mathbf{a}_{k+1}(\sigma'), \mathbf{s}_{k+1}(\sigma'))$

Proof: By 5 (the definition of f'), since $\tau|_k$ is a prefix of σ' and 7.3B.2 implies that $f'(\tau|_k)$ is defined.

$\boxed{\text{7.3B.4.}}$ $\tau|_{k+1} = \sigma'|_{k+1}$

Proof: By 7.3B.2 and 7.3B.3, since $\tau|_k = \sigma'|_k$ by 7.1.

<u>Case 7.3C</u>. $m > 1 + l$

7.3C.1. $\Xi(\tau)|_m = \Xi(\tau)|_{m-1} \cdot (\mathbf{a}_{k+m-l}(\tau), \mathbf{s}_{k+m-l}(\tau))$

Proof: By applying 7.1 twice, substituting $m - l - 1$ and $m - l$ for i.

7.3C.2. $f(\Xi(\tau)|_{m-1}) = f'(\tau|_{k+m-l-1})$.

Proof: By 7.2 with $m - l - 1$ substituted for i, since the hypothesis $m > 1 + l$ implies $i > 0$.

$\boxed{\text{7.3C.3.}}$ $\Xi(\tau)|_m$ ends according to μ, f.

Proof: By 7.3C.1, 7.3C.2, and the hypothesis that τ is a μ-outcome of f'.

$\boxed{7.4.}$ $\Xi(\tau)$ is a fair μ-outcome of f.

Proof: By 7.3, $\Xi(\tau)$ is a μ-outcome of f. We now show that it is fair. By 7.1, $\Xi(\tau)$ has infinitely many μ actions iff τ does. By 7.2, f is undefined on infinitely many prefixes of $\Xi(\tau)$ iff f' is undefined on infinitely many prefixes of τ. Hence, $\Xi(\tau)$ is fair because τ is assumed to be a fair μ-outcome of f'.

$\boxed{8.}$ Every behavior in $\mathcal{O}_\mu(f')$ is stuttering-equivalent to a behavior in $\mathcal{O}_\mu(f)$.

Proof: By 4 and 7.

End proof of Lemma 1

Lemma 2 *For any agent set μ, if (i) f is a μ-strategy, (ii) $\sigma \in \mathcal{O}_\mu(f)$, and (iii) σ' is μ-equivalent to σ, then there exists a μ-strategy f' such that (iv) $\sigma' \in \mathcal{O}_\mu(f')$ and (v) every behavior in $\mathcal{O}_\mu(f')$ is μ-equivalent to some behavior in $\mathcal{O}_\mu(f)$.*

Proof of Lemma 2

The proof is almost identical to that of Lemma 1, except with $\simeq$ replaced by μ-equivalence. The definition of Ξ in step 1 becomes:

if $|\rho| = 0$ **then** $\Xi(\rho) = \rho$

if $\rho = \theta \cdot (\alpha, s)$ **then if** ρ is a prefix of σ'
then $\Xi(\rho)$ is the prefix of σ of the same length as ρ
else $\Xi(\rho) = \Xi(\theta) \cdot (\alpha, s)$

The proof becomes a bit simpler because l equals k in step 7.1. We omit the details.

End Proof of Lemma 2

Proposition 1 *For every agent set μ, if P is a property then $\mathcal{R}_\mu(P)$ is a property, and if P is μ-abstract then $\mathcal{R}_\mu(P)$ is μ-abstract.*

Proof of Proposition 1

The first part of the proposition, that if P is a property then $\mathcal{R}_\mu(P)$ is also a property, follows immediately from Lemma 1 and the definitions. The second part, that if P is μ-abstract then $\mathcal{R}_\mu(P)$ is also μ-abstract, follows from Lemma 2.

End Proof of Proposition 1

Lemma 3 *For any agent set μ, if (i) f is a μ-strategy and (ii) $\sigma \in \mathcal{O}_\mu(f)$, then there exists a behavior σ' and a total μ-strategy f' that is invariant under $\neg\mu$-stuttering such that (iii) $\sigma' \simeq \sigma$, (iv) $\sigma' \in \mathcal{O}_\mu(f')$, and (v) every behavior in $\mathcal{O}_\mu(f')$ is stuttering-equivalent to a behavior in $\mathcal{O}_\mu(f)$.*

Proof of Lemma 3

We assume that f is a μ-strategy and $\sigma \in \mathcal{O}_\mu(f)$, and we will construct the required f' and σ'. Instead of using σ and f directly, for technical reasons we will construct a new behavior ϕ in step 1 by adding an infinite number of $\neg\mu$-stuttering steps to

σ, and will use Lemma 1 to obtain a strategy g that produces ϕ. The behavior σ' will be obtained (in step 8) by replacing $\neg\mu$-stuttering steps in ϕ by μ-stuttering steps. We will construct f' (in step 5) so it tries to produce σ' and, failing that, to simulate g. To make f' total, we will define it to stutter when g would be undefined. This requires f' to interpret μ-stuttering steps produced in this way as if they were $\neg\mu$-stuttering steps—an interpretation performed by the mapping Λ, defined in step 3. The behavior prefix $\Lambda(\rho)$ will be obtained from ρ by replacing μ-stuttering steps with $\neg\mu$-stuttering steps if either that will lead to a prefix of ϕ, or those μ-stuttering steps were produced by f' because g was undefined. To make f' invariant under $\neg\mu$-stuttering, we will define f' in terms of Ξ, the mapping obtained by removing $\neg\mu$-stuttering steps and then applying Λ.

1. Choose a behavior ϕ such that $\phi \simeq \sigma$ and ϕ contains infinitely many $\neg\mu$-stuttering steps, and choose a μ-strategy g such that $\phi \in \mathcal{O}_\mu(g)$ and every behavior in $\mathcal{O}_\mu(g)$ is stuttering-equivalent to a behavior in $\mathcal{O}_\mu(f)$.
   Proof: The existence of ϕ follows from the assumption that μ is an agent set, so $\neg\mu$ is nonempty. The existence of g follows from Lemma 1.
2. Choose agents β_μ in μ and $\beta_{\neg\mu}$ in $\neg\mu$.
   Proof: Since μ is an agent set, μ and $\neg\mu$ are nonempty.
3. For any finite behavior prefix ρ, define $\Lambda(\rho)$ as follows.

$$
\begin{array}{l}
\textbf{if } |\rho| = 0 \textbf{ then } \Lambda(\rho) = \rho \\[4pt]
\textbf{if } \rho = \theta \cdot (\alpha, s) \\
\quad \textbf{then if } \ (\Lambda(\theta) = \phi|_k) \wedge (s = \theta_{\mathbf{s}} = \mathbf{s}_k(\phi) = \mathbf{s}_{k+1}(\phi)) \wedge \\
\qquad\qquad\quad (\alpha \in \mu) \wedge (\mathbf{a}_{k+1}(\phi) \in \neg\mu) \\
\qquad\qquad\ \text{where } k = |\Lambda(\theta)| \\
\qquad\qquad \textbf{then } \Lambda(\rho) = \phi|_{k+1} \\
\qquad\qquad \textbf{else } \ \textbf{if } (s = \theta_{\mathbf{s}}) \wedge (\alpha \in \mu) \wedge (g(\Lambda(\theta)) = \bot) \\
\qquad\qquad\qquad\quad \textbf{then } \Lambda(\rho) = \Lambda(\theta) \cdot (\beta_{\neg\mu}, s) \\
\qquad\qquad\qquad\quad \textbf{else } \ \Lambda(\rho) = \Lambda(\theta) \cdot (\alpha, s)
\end{array}
$$

   For any behavior τ, define $\Lambda(\tau)$ to equal $\lim_{m\to\infty} \Lambda(\tau|_m)$.
   Proof: The mapping Λ is monotone on finite behavior prefixes, so the limit exists.
4. For any any behavior prefix τ, let $\Xi(\tau) = \Lambda(\natural_{\neg\mu}(\tau))$. Then $\tau \simeq \Xi(\tau)$.
   Proof: By definition, $\natural_{\neg\mu}\tau \simeq \tau$. It follows from 3 (the definition of Λ) by induction on the length of ρ that $\Lambda(\rho) \simeq \rho$ for any finite behavior prefix ρ. Since $\Lambda(\tau)$ equals $\lim_{m\to\infty} \Lambda(\tau|_m)$ (by 3), this implies that $\Lambda(\tau) \simeq \tau$ for any behavior τ. Hence, $\Xi(\tau) \simeq \tau$.
5. For any finite behavior prefix ρ, define $f'(\rho)$ as follows.

$$
\begin{array}{l}
\textbf{if } \ (\Xi(\rho) = \phi|_k) \wedge (\mathbf{s}_{k+1}(\phi) = \mathbf{s}_k(\phi)) \wedge (\mathbf{a}_{k+1}(\phi) \in \neg\mu) \\
\quad \text{where } k = |\Xi(\rho)|, \text{ or} \\
\quad\ g(\Xi(\rho)) = \bot \\
\textbf{then } f'(\rho) = (\beta_\mu, \rho_{\mathbf{s}}) \\
\textbf{else } \ f'(\rho) = g(\Xi(\rho))
\end{array}
$$

   Then f' is a total μ-strategy that is invariant under $\neg\mu$ stuttering.

Proof: Since g is a μ-strategy and β_μ is in μ, it follows that f' is a μ-strategy. Since $\theta \simeq_{\neg\mu} \rho$ iff $\natural_{\neg\mu}\theta = \natural_{\neg\mu}\rho$, the definitions of Ξ (in step 4) and f' imply that f' is invariant under $\neg\mu$-stuttering. By definition, f' is a total function.

6. $\Xi(\tau) \in \mathcal{O}_\mu(g)$ for any behavior $\tau \in \mathcal{O}_\mu(f')$.

   Proof: We assume $\tau \in \mathcal{O}_\mu(f')$ and prove $\Xi(\tau) \in \mathcal{O}_\mu(g)$. It is simpler to prove $\Xi(\tau) \in \mathcal{O}_\mu(g)$ if τ has no $\neg\mu$-stuttering steps. We will therefore prove $\natural_{\neg\mu}\tau \in \mathcal{O}_\mu(g)$ (step 6.6) and then observe (in the proof of 6.7) that $\Xi(\tau)$ equals $\Xi(\natural_{\neg\mu}\tau)$. The proof of $\Xi(\natural_{\neg\mu}\tau) \in \mathcal{O}_\mu(g)$ is an intricate exercise in verifying that our definitions of Λ and f' work properly.

   6.1. For any behavior prefix ψ, $|\Lambda(\psi)| = |\psi|$, and if $\psi = \natural_{\neg\mu}\psi$ then $\Xi(\psi) = \Lambda(\psi)$.

   Proof: First, assume that ψ is finite. A simple induction on $|\psi|$ shows that Λ is length-preserving. If $\psi = \natural_{\neg\mu}\psi$, then 4 (the definition of Ξ) implies $\Xi(\psi) = \Lambda(\psi)$. The case of ψ infinite follows from the finite case by taking limits.

   6.2. Let ρ be a finite behavior prefix with $\rho = \natural_{\neg\mu}\rho$ and let m equal $|\rho|$. If $\Xi(\rho) = \eta \cdot (\alpha, s)$ and $\alpha \in \mu$, then

   (a) $\alpha = \mathbf{a}_m(\rho)$ and $s = \mathbf{s}_m(\rho)$,

   (b) $\eta = \Lambda(\rho|_{m-1})$, and

   (c) if $\mathbf{s}_m(\rho) = \mathbf{s}_{m-1}(\rho)$, then neither (i) $\eta = \phi|_{m-1}$, $\mathbf{s}_m(\phi) = \mathbf{s}_{m-1}(\phi)$, and $\mathbf{a}_m(\phi) \in \neg\mu$, nor (ii) $g(\eta) = \perp$ holds.

   Proof: We assume $\rho = \natural_{\neg\mu}\rho$, $\Xi(\rho) = \eta \cdot (\alpha, s)$, and $\alpha \in \mu$, and we prove (a)–(c).

   6.2.1. For any finite θ, (i) $\Lambda(\theta)_{\mathbf{s}} = \theta_{\mathbf{s}}$, and (ii) if $\Lambda(\theta)_{\mathbf{a}} \in \mu$ then $\Lambda(\theta)_{\mathbf{a}} = \theta_{\mathbf{a}}$.

   Proof: By 3.

   $\boxed{6.2.2.}$ $\alpha = \mathbf{a}_m(\rho)$ and $s = \mathbf{s}_m(\rho)$.

   Proof: By 6.2.1, since 6.1 implies $\Xi(\rho) = \Lambda(\rho)$.

   $\boxed{6.2.3.}$ $\eta = \Lambda(\rho|_{m-1})$

   Proof: By 6.1, $\eta \cdot (\alpha, s) = \Lambda(\rho)$, and 3 (the definition of Λ) implies that $\Lambda(\rho) = \Lambda(\rho|_{m-1}) \cdot (\gamma, t)$ for some γ and t.

   $\boxed{6.2.4.}$ If $\mathbf{s}_m(\rho) = \mathbf{s}_{m-1}(\rho)$, then it is not the case that: (i) $\eta = \phi|_{m-1}$, (ii) $\mathbf{s}_m(\phi) = \mathbf{s}_{m-1}(\phi)$, and (iii) $\mathbf{a}_m(\phi) \in \neg\mu$.

   Proof: We assume (i)–(iii) and (iv) $\mathbf{s}_m(\rho) = \mathbf{s}_{m-1}(\rho)$, and we obtain a contradiction. Let θ equal $\rho|_{m-1}$. Then η equals $\Lambda(\theta)$ by 6.2.3, and $s = \theta_{\mathbf{s}} = \mathbf{s}_{m-1}(\phi)$ by (i), (iv), and 6.2.1. Applying 3 with $m-1$ substituted for k, using 6.1 (to infer $|\Xi(\rho|_{m-1})| = m - 1$) and the assumption that $\alpha \in \mu$, yields $\Lambda(\rho) = \phi|_m$. Hence, $\alpha = \mathbf{a}_m(\phi)$, which by (iii) contradicts the assumption $\alpha \in \mu$.

   $\boxed{6.2.5.}$ If $\mathbf{s}_m(\rho) = \mathbf{s}_{m-1}(\rho)$, then $g(\eta) \neq \perp$.

   Proof: We assume (i) $\mathbf{s}_m(\rho) = \mathbf{s}_{m-1}(\rho)$ and (ii) $g(\eta) = \perp$, and we obtain a contradiction. Let $\theta = \rho|_{m-1}$. Then η equals $\Lambda(\theta)$ by 6.2.3, so $s = \theta_{\mathbf{s}}$ by 6.2.1 and (i). By 6.2.4 and 3 (the definition of Λ), we see that (ii), $s = \theta_{\mathbf{s}}$, and the hypothesis $\alpha \in \mu$ imply $\Lambda(\rho)_{\mathbf{a}} = \beta_{\neg\mu}$. This contradicts the hypothesis that α, which equals $\Lambda(\rho)_{\mathbf{a}}$, is in μ.

6.3. For any finite behavior prefix ρ, if $\rho = \natural_{\neg\mu}\rho$ and ρ ends according to μ, f', then $\Xi(\rho)$ ends according to μ, g.

Proof: We assume that $\rho = \natural_{\neg\mu}\rho$ and ρ ends according to μ, f', and we prove that $\Xi(\rho)$ ends according to μ, g. Since this is trivial if $|\Xi(\rho)| = 0$ or $\Xi(\rho)_{\mathbf{a}} \in \neg\mu$, it suffices to assume that $\Xi(\rho) = \eta \cdot (\alpha, s)$ with $\alpha \in \mu$ and prove that $(\alpha, s) = g(\eta)$. Let m equal $|\rho|$.

6.3.1. $(\alpha, s) = f'(\rho|_{m-1})$

Proof: By 6.2(a) and the hypothesis that ρ ends according to μ, f'.

6.3.2. $\eta = \Xi(\rho|_{m-1})$

Proof: By 6.1 and 6.2(b).

6.3.3. $f'(\rho|_{m-1}) = g(\Xi(\rho|_{m-1}))$

Proof: We assume $f'(\rho|_{m-1}) \neq g(\Xi(\rho|_{m-1}))$ and obtain a contradiction. Substituting $\rho|_{m-1}$ for ρ in 5 shows that if $f'(\rho|_{m-1}) \neq g(\Xi(\rho|_{m-1}))$ then $f'(\rho|_{m-1}) = (\beta_\mu, \mathbf{s}_{m-1}(\rho))$. Hence, 6.3.1 and 6.2(a) imply $\mathbf{s}_m(\rho) = \mathbf{s}_{m-1}(\rho)$. Substituting $\rho|_{m-1}$ for ρ in 5 again, and using 6.1 to infer $|\Xi(\rho|_{m-1})| = m - 1$, then shows that 6.3.2 and 6.2(c) imply $f'(\rho|_{m-1}) = g(\Xi(\rho|_{m-1}))$, which is the required contradiction.

$\boxed{6.3.4.}$ $(\alpha, s) = g(\eta)$.

Proof: By 6.3.1–6.3.3.

6.4. (a) $\mathbf{a}_m(\tau) \in \mu$ for infinitely many values of m, and (b) $\natural_{\neg\mu}\tau \in \mathcal{O}_\mu(f')$.

Proof: Part (a) follows from the hypothesis that $\tau \in \mathcal{O}_\mu(f')$, since f' is a total function. This implies that $\natural_{\neg\mu}\tau$ is a behavior. Since f' is invariant under $\neg\mu$-stuttering, τ is an outcome of f' iff $\natural_{\neg\mu}\tau$ is. The behavior $\natural_{\neg\mu}\tau$ is a fair outcome because (a) implies that it contains infinitely many μ actions.

6.5. $\Xi(\natural_{\neg\mu}\tau)$ is a μ-outcome of g.

Proof: The definition of $\natural_{\neg\mu}$ implies that $\rho = \natural_{\neg\mu}\rho$ for every prefix ρ of $\natural_{\neg\mu}\tau$. By monotonicity of Λ and 6.1, every prefix of $\Xi(\natural_{\neg\mu}\tau)$ equals $\Xi(\natural_{\neg\mu}\rho)$ for some finite prefix ρ of τ. The result then follows from 6.4(b) and 6.3.

6.6. $\Xi(\natural_{\neg\mu}\tau) \in \mathcal{O}_\mu(g)$

Proof: Let ψ denote $\natural_{\neg\mu}\tau$. By 6.5, it suffices to prove that $\mathbf{a}_m(\Xi(\psi)) \in \mu$ or $g(\Xi(\psi)|_m) = \bot$, for infinitely many values of m. Since $\phi \in \mathcal{O}_\mu(g)$, we may assume that $\Xi(\psi) \neq \phi$.

6.6.1. $\Lambda(\psi)|_k = \Lambda(\psi|_k)$ for any natural number k.

Proof: By 3, which asserts that $\Lambda(\psi) = \lim_{m\to\infty} \Lambda(\psi|_m)$, and 6.1.

6.6.2. Choose k such that $\Xi(\psi)|_k$ is not a prefix of ϕ.

Proof: The existence of k follows from the hypothesis that $\Xi(\psi) \neq \phi$.

6.6.3. For all $m > k$, if $\mathbf{a}_m(\psi) \in \mu$ then $\mathbf{a}_m(\Xi(\psi)) \in \mu$ or $g(\Xi(\psi)|_{m-1}) = \bot$.

Proof: Substituting $\psi|_m$ for ρ in 3 yields

$$
\begin{array}{l}
\textbf{if } \Lambda(\psi|_{m-1}) = \phi|_k \wedge \ldots \\
\quad \textbf{then } \ldots \\
\quad \textbf{else if } g(\Lambda(\psi|_{m-1})) = \bot \wedge \ldots \\
\qquad\quad \textbf{then } \ldots \\
\qquad\quad \textbf{else } \Lambda(\psi|_m) = \ldots \cdot (\mathbf{a}_m(\psi), \ldots)
\end{array}
$$

By 6.6.1 and 6.6.2, $m > k$ implies $\Lambda(\psi|_{m-1})$ is not a prefix of ϕ. Hence, $\Lambda(\psi|_m)_{\mathbf{a}} = \mathbf{a}_m(\psi)$ or $g(\Lambda(\psi|_{m-1})) = \perp$. But 6.6.1 implies $\Lambda(\psi|_m)_{\mathbf{a}} = \mathbf{a}_m(\Lambda(\psi))$ and $g(\Lambda(\psi|_{m-1})) = g(\Lambda(\psi)|_{m-1})$. Hence, $\mathbf{a}_m(\psi) \in \mu$ implies $\mathbf{a}_m(\Lambda(\psi)) \in \mu$ or $g(\Lambda(\psi)|_{m-1}) = \perp$. The result now follows from 6.1, which implies that $\Lambda(\psi) = \Xi(\psi)$.

6.6.4. $\mathbf{a}_m(\psi) \in \mu$ for infinitely many values of m.

Proof: By 6.4(a), since $\psi = \natural_{\neg\mu}\tau$ and $\natural_{\neg\mu}$ preserves actions in μ.

$\boxed{6.6.5.}$ $\mathbf{a}_m(\Xi(\psi)) \in \mu$ or $g(\Xi(\psi)|_m) = \perp$, for infinitely many values of m.

Proof: By 6.6.3 and 6.6.4, since $A_m \vee B_m$ holds for infinitely many values of m iff A_m holds for infinitely many values of m or B_m holds for infinitely many values of m.

$\boxed{6.7.}$ $\Xi(\tau) \in \mathcal{O}_\mu(g)$

Proof: By 6.6, since $\Xi(\natural_{\neg\mu}\tau) = \Xi(\tau)$ by 4 (the definition of Ξ) and the idempotence of $\natural_{\neg\mu}$.

$\boxed{7.}$ For any behavior $\tau \in \mathcal{O}_\mu(f')$, there exists a behavior $\upsilon \in \mathcal{O}_\mu(f)$ such that $\upsilon \simeq \tau$.

Proof: By 6, $\tau \in \mathcal{O}_\mu(f')$ implies $\Xi(\tau) \in \mathcal{O}_\mu(g)$. By 1, there exists a behavior υ in $\mathcal{O}_\mu(g)$ such that $\Xi(\tau) \simeq \upsilon$. By 4, $\tau \simeq \Xi(\tau)$. The transitivity of $\simeq$ then yields $\tau \simeq \upsilon$.

$\boxed{8.}$ There exists a behavior $\sigma' \in \mathcal{O}_\mu(f')$ such that $\sigma' \simeq \sigma$.

Proof: We will construct σ' from ϕ by replacing $\neg\mu$-stuttering steps with μ-stuttering steps. The proof that σ' is a μ-outcome of f' is a matter of checking the definitions of Λ and f'. Fairness will follow from having chosen ϕ with infinitely many $\neg\mu$-stuttering steps.

8.1. Define σ' as follows. For all $k \geq 0$,

$\mathbf{s}_k(\sigma') = \mathbf{s}_k(\phi)$
if $\mathbf{a}_{k+1}(\phi) \in \neg\mu \;\wedge\; \mathbf{s}_{k+1}(\phi) = \mathbf{s}_k(\phi)$
then $\mathbf{a}_{k+1}(\sigma') = \beta_\mu$
else $\mathbf{a}_{k+1}(\sigma') = \mathbf{a}_{k+1}(\phi)$

8.2. $\natural_{\neg\mu}(\sigma'|_k) = \sigma'|_k$ for all $k \geq 0$.

Proof: By the definition 8.1, σ' has no $\neg\mu$-stuttering steps.

8.3. $\Lambda(\sigma'|_k) = \phi|_k$ for all $k \geq 0$.

Proof: The proof is by induction on k. The result is obvious for $k = 0$. We assume it true for k and prove it for $k+1$. We consider two cases:

<u>Case 8.3A</u>. $\mathbf{a}_{k+1}(\sigma') \neq \mathbf{a}_{k+1}(\phi)$

8.3A.1. $\mathbf{a}_{k+1}(\sigma') \in \mu$, $\mathbf{a}_{k+1}(\phi) \in \neg\mu$, and $\mathbf{s}_{k+1}(\phi) = \mathbf{s}_k(\phi)$.

Proof: By the definition of σ' and the assumption that $\mathbf{a}_{k+1}(\sigma') \neq \mathbf{a}_{k+1}(\phi)$.

8.3A.2. $\mathbf{s}_{k+1}(\sigma') = \mathbf{s}_k(\sigma')$.

Proof: By 8.1 (the definition of σ') and 8.3A.1, since $\mathbf{s}_m(\sigma') = \mathbf{s}_m(\phi)$ for all m.

$\boxed{8.3A.3.}$ $\Lambda(\sigma'|_{k+1}) = \phi|_{k+1}$

Proof: By the induction hypothesis, 8.1, 8.3A.1, 8.3A.2, and 3 (the definition of Λ).

Case 8.3B. $\mathbf{a}_{k+1}(\sigma') = \mathbf{a}_{k+1}(\phi)$

We assume that $\Lambda(\sigma'|_{k+1}) \neq \phi|_{k+1}$ and obtain a contradiction.

8.3B.1. $\mathbf{a}_{k+1}(\sigma') \in \mu$ and $g(\Lambda(\sigma'|_k)) = \perp$.

Proof: By the induction hypothesis, 8.1 (which implies $\mathbf{s}_{k+1}(\phi) = \mathbf{s}_{k+1}(\sigma')$), the assumption that $\Lambda(\sigma'|_{k+1}) \neq \phi|_{k+1}$, and 3 (the definition of Λ).

8.3B.2. $\mathbf{a}_{k+1}(\phi) \in \mu$ and $g(\phi|_k) = \perp$.

Proof: By 8.3B.1, the hypothesis that $\mathbf{a}_{k+1}(\sigma') = \mathbf{a}_{k+1}(\phi)$, and the induction hypothesis that $\Lambda(\sigma'|_k) = \phi|_k$.

8.3B.3. Contradiction.

Proof: 8.3B.2 and the hypothesis that ϕ is a μ-outcome of g.

8.4. $\Xi(\sigma'|_k) = \phi|_k$ for all $k \geq 0$.

Proof: By 8.2, 8.3, and 4 (the definition of Ξ).

$\boxed{8.5.}$ $\sigma' \simeq \sigma$

Proof: 8.2, 8.3, and the definition of Ξ imply $\phi = \Xi(\sigma')$. Step 4 asserts $\sigma' \simeq \Xi(\sigma')$, so $\sigma' \simeq \phi$. By 1, $\phi \simeq \sigma$, so $\sigma' \simeq \sigma$ follows from the transitivity of $\simeq$.

$\boxed{8.6.}$ $\sigma' \in \mathcal{O}_\mu(f')$

Proof: Since ϕ contains an infinite number of $\neg\mu$-stuttering steps (by 1), 8.1 (the definition of σ') implies that $\mathbf{a}_m(\sigma') \in \mu$ for infinitely many values of m. Therefore, to show that $\sigma' \in \mathcal{O}_\mu(f')$, it suffices to assume that $\mathbf{a}_{m+1}(\sigma') \in \mu$ and prove that $(\mathbf{a}_{m+1}(\sigma'), \mathbf{s}_{m+1}(\sigma')) = f'(\sigma'|_m)$. We consider two cases.

Case 8.6A. $\mathbf{a}_{m+1}(\sigma') \neq \mathbf{a}_{m+1}(\phi)$

8.6A.1. $\mathbf{a}_{m+1}(\phi) \in \neg\mu$, $\mathbf{s}_{m+1}(\phi) = \mathbf{s}_m(\phi)$, and $\mathbf{a}_{m+1}(\sigma') = \beta_\mu$.

Proof: By 8.1 (the definition of σ').

$\boxed{8.6A.2.}$ $f'(\sigma'|_m) = (\mathbf{a}_{m+1}(\sigma'), \mathbf{s}_{m+1}(\sigma'))$.

Proof: By 8.6A.1, 8.4, 8.1 (which implies $\mathbf{s}_m(\sigma') = \mathbf{s}_m(\phi)$ and $\mathbf{s}_{m+1}(\sigma') = \mathbf{s}_{m+1}(\phi)$), and the definition of f'.

Case 8.6B. $\mathbf{a}_{m+1}(\sigma') = \mathbf{a}_{m+1}(\phi)$

8.6B.1. $g(\phi|_m) = (\mathbf{a}_{m+1}(\phi), \mathbf{s}_{m+1}(\phi))$

Proof: Since $\mathbf{a}_{m+1}(\sigma')$ is assumed to be in μ, and ϕ is a μ-outcome of g.

8.6B.2. $f'(\sigma'|_m) = g(\Xi(\sigma'|_m))$

Proof: By definition of f', since $\mathbf{a}_{m+1}(\phi) \in \mu$ by hypothesis, $g(\phi|_m) \neq \perp$ by 8.6B.1, and $\sigma'|_m = \phi|_m$ by 8.4.

$\boxed{8.6B.3.}$ $f'(\sigma'|_m) = (\mathbf{a}_{m+1}(\sigma'), \mathbf{s}_{m+1}(\sigma'))$

Proof: By 8.6B.1, 8.6B.2, 8.1, 8.4, and the assumption $\mathbf{a}_{m+1}(\sigma') = \mathbf{a}_{m+1}(\phi)$.

End Proof of Lemma 3

Proposition 2 *For any agent set μ and any property P, let $\mathcal{S}_\mu(P)$ be the subset of $\mathcal{R}_\mu(P)$ consisting of the union of all sets $\mathcal{O}_\mu(f)$ contained in P such that f is a total μ-strategy that is invariant under $\neg\mu$-stuttering. Then every behavior in $\mathcal{R}_\mu(P)$ is stuttering-equivalent to a behavior in $\mathcal{S}_\mu(P)$.*

Proof of Proposition 2
This is an immediate consequence of Lemma 3.
End Proof of Proposition 2

Proposition 3 *For any properties P and Q and any agent set μ, if $P \subseteq Q$ then $\mathcal{R}_\mu(P) \subseteq \mathcal{R}_\mu(Q)$.*

Proof of Proposition 3
We assume $P \subseteq Q$ and $\sigma \in \mathcal{R}_\mu(P)$, and we prove $\sigma \in \mathcal{R}_\mu(Q)$.
1. Choose a μ-strategy f such that $\sigma \in \mathcal{O}_\mu(f) \subseteq P$.
   Proof: The existence of f follows from the definition of $\mathcal{R}_\mu(P)$.
2. $\sigma \in \mathcal{O}_\mu(f) \subseteq Q$
   Proof: 1 and the hypothesis $P \subseteq Q$.

$\boxed{3.}$ $\sigma \in \mathcal{R}_\mu(Q)$
   Proof: By 2 and the definition of $\mathcal{R}_\mu(Q)$.

End Proof of Proposition 3

Proposition 4 *For any property P and agent set μ, $\mathcal{R}_\mu(\mathcal{R}_\mu(P)) = \mathcal{R}_\mu(P)$.*

Proof of Proposition 4
We assume that P is a property and μ is an agent set, and we prove $\mathcal{R}_\mu(\mathcal{R}_\mu(P)) = \mathcal{R}_\mu(P)$. The set $\mathcal{R}_\mu(P)$ consists of all outcomes of winning strategies when the system is trying to produce an outcome in P. Any such strategy is also a winning strategy when the system is trying to produce an outcome in $\mathcal{R}_\mu(P)$, so $\mathcal{R}_\mu(\mathcal{R}_\mu(P))$ must equal $\mathcal{R}_\mu(P)$. The formal proof is as follows.

$\boxed{1.}$ $\mathcal{R}_\mu(\mathcal{R}_\mu(P)) \subseteq \mathcal{R}_\mu(P)$
   Proof: By Proposition 3, since $\mathcal{R}_\mu(P) \subseteq P$.

$\boxed{2.}$ $\mathcal{R}_\mu(P) \subseteq \mathcal{R}_\mu(\mathcal{R}_\mu(P))$
   Proof: We assume that $\sigma \in \mathcal{R}_\mu(P)$ and prove that $\sigma \in \mathcal{R}_\mu(\mathcal{R}_\mu(P))$.
   2.1. Choose a μ-strategy f such that $\sigma \in \mathcal{O}_\mu(f) \subseteq P$.
      Proof: f exists by definition of $\mathcal{R}_\mu(P)$.
   2.2. $\mathcal{O}_\mu(f) \subseteq \mathcal{R}_\mu(P)$
      Proof: By definition of $\mathcal{R}_\mu(P)$.
   2.3. $\sigma \in \mathcal{R}_\mu(\mathcal{R}_\mu(P))$
      Proof: By definition of $\mathcal{R}_\mu(\mathcal{R}_\mu(P))$, since $\mathcal{O}_\mu(f) \subseteq \mathcal{R}_\mu(P)$ by 2.2 and $\sigma \in \mathcal{O}_\mu(f)$ by 2.1.

End Proof of Proposition 4

Proposition 5 *For any property P and agent set μ, $\mathcal{R}_\mu(P) = \overline{\mathcal{R}_\mu(P)} \cap P$.*

Proof of Proposition 5

$\boxed{1.}$ $\mathcal{R}_\mu(P) \subseteq \overline{\mathcal{R}_\mu(P)} \cap P$
   Proof: $\mathcal{R}_\mu(P)$ is included both in $\overline{\mathcal{R}_\mu(P)}$ (by the definition of closure) and in P (by the definition of $\mathcal{R}_\mu$).

$\boxed{2.}$ $\overline{\mathcal{R}_\mu(P)} \cap P \subseteq \mathcal{R}_\mu(P)$

Proof: We assume $\sigma \in \overline{\mathcal{R}_\mu(P)} \cap P$ and prove $\sigma \in \mathcal{R}_\mu(P)$. By definition of $\mathcal{R}_\mu(P)$, it suffices to prove that there exists a μ-strategy f such that $\sigma \in \mathcal{O}_\mu(f)$ and $\mathcal{O}_\mu(f) \subseteq P$. To construct f, we will choose a sequence ϕ_i of behaviors in $\mathcal{R}_\mu(P)$ having σ as their limit, and strategies g_i that produce the ϕ_i. We will define f so it tries to produce σ. As it does so, it is acting like g_i for all sufficiently large i. When f can no longer produce σ, it continues to act like one of the g_i.

2.1. For all $i \geq 0$, choose a behavior ϕ_i in $\mathcal{R}_\mu(P)$ such that $\sigma|_i$ is a prefix of ϕ_i.

Proof: The ϕ_i exist by definition of closure, since $\sigma \in \overline{\mathcal{R}_\mu(P)}$

2.2. For all $i \geq 0$, choose a μ-strategy g_i such that $\sigma|_i$ is a prefix of a fair outcome of g_i and $\mathcal{O}_\mu(g_i) \subseteq P$.

Proof: By 2.1, there exist μ-strategies g_i with $\phi_i \in \mathcal{O}_\mu(g_i) \subseteq P$ in 2.1.

2.3. Define the μ-strategy f as follows.

$$\begin{array}{l} \textbf{if } \rho = \sigma|_j \text{ for some } j \\ \quad \textbf{then if } \mathbf{a}_{j+1}(\sigma) \in \mu \textbf{ then } f(\rho) = (\mathbf{a}_{j+1}(\sigma), \mathbf{s}_{j+1}(\sigma)) \\ \qquad\qquad\qquad\qquad\quad \textbf{else } f(\rho) = \bot \\ \quad \textbf{else } f(\rho) = g_i(\rho), \text{ where } i \text{ is the smallest integer} \\ \qquad\qquad\qquad\qquad\quad \text{such that } \rho|_i \neq \sigma|_i \end{array}$$

Proof: f is a μ-strategy since each g_i is (by 2.2).

$\boxed{2.4.}$ $\sigma \in \mathcal{O}_\mu(f)$

Proof: By 2.3, if $\mathbf{a}_m(\sigma) \in \mu$ then $f(\sigma|_{m-1}) = (\mathbf{a}_m(\sigma), \mathbf{s}_m(\sigma))$, for all $m > 0$. Thus σ is a μ-outcome of f. Furthermore, 2.3 implies that $f(\sigma|_m)$ is undefined if $\mathbf{a}_{m+1}(\sigma) \in \neg\mu$, so σ is fair.

2.5. For all $\tau \in \mathcal{O}_\mu(f)$, if $\tau \neq \sigma$ then $\tau \in \mathcal{O}_\mu(g_i)$ for some i.

Proof: Assume $\tau \in \mathcal{O}_\mu(f)$ and $\tau \neq \sigma$. Let i be the smallest integer such that $\tau|_i \neq \sigma|_i$. We show that $\tau \in \mathcal{O}_\mu(g_i)$.

2.5.1. For all $j \geq i$, $f(\tau|_j) = g_i(\tau|_j)$.

Proof: By definition of i, if $j \geq i$ then $\sigma|_j \neq \tau|_j$. The result then follows from 2.3 (the definition of f).

2.5.2. τ is a μ-outcome of g_i.

Proof: We must show that for all $j \geq 0$, if $\mathbf{a}_{j+1}(\tau) \in \mu$ then $g_i(\tau|_j) = (\mathbf{a}_{j+1}(\tau), \mathbf{s}_{j+1}(\tau))$. We split the proof into two cases.

<u>Case 2.5.2A</u>. $j < i$

2.5.2A.1. $\sigma|_j = \tau|_j$

Proof: By the hypothesis that $j < i$ and the definition of i.

2.5.2A.2. $(\mathbf{a}_{j+1}(\tau), \mathbf{s}_{j+1}(\tau)) = (\mathbf{a}_{j+1}(\sigma), \mathbf{s}_{j+1}(\sigma))$

Proof: By 2.5.2A.1, 2.3, and the hypotheses that τ is a μ-outcome of f and $\mathbf{a}_{j+1}(\tau) \in \mu$.

2.5.2A.3. $\sigma|_{j+1}$ is a prefix of $\sigma|_i$.

Proof: By hypothesis that $j < i$.

2.5.2A.4. $g_i(\sigma|_j) = (\mathbf{a}_{j+1}(\sigma), \mathbf{s}_{j+1}(\sigma))$

Proof: 2.5.2A.2 and the assumption $\mathbf{a}_{j+1}(\tau) \in \mu$ imply $\mathbf{a}_{j+1}(\sigma) \in \mu$. The result then follows from 2.5.2A.3 and 2.2, which asserts that $\sigma|_i$ is a partial μ-outcome of g_i.

$\boxed{\text{2.5.2A.5.}}$ $g_i(\tau|_j) = (\mathbf{a}_{j+1}(\tau), \mathbf{s}_{j+1}(\tau))$

Proof: By 2.5.2A.1, 2.5.2A.2, and 2.5.2A.4.

<u>Case 2.5.2B</u>. $j \geq i$

2.5.1 and the two hypotheses $\tau \in \mathcal{O}_\mu(f)$ and $\mathbf{a}_{j+1}(\tau) \in \mu$ imply $g_i(\tau|_j) = (\mathbf{a}_{j+1}(\tau), \mathbf{s}_{j+1}(\tau))$.

$\boxed{\text{2.5.3.}}$ τ is a fair μ-outcome of g_i.

Proof: 2.5.2 asserts that τ is a μ-outcome of g_i, and fairness follows from 2.5.1 and the assumption that τ is a fair outcome of f.

$\boxed{\text{2.6.}}$ $\mathcal{O}_\mu(f) \subseteq P$

Proof: We assume $\tau \in \mathcal{O}_\mu(f)$ and prove that $\tau \in P$. This is immediate if $\tau = \sigma$, because σ is in P by hypothesis. If $\tau \neq \sigma$, it follows from 2.5 and 2.2.

End Proof of Proposition 5

Proposition 6 *For any nonempty safety property P and any agent set μ, property P constrains at most μ iff $P = \mathcal{R}_\mu(P)$.*

Proof of Proposition 6

We assume that P is a nonempty safety property and μ is an agent set. Since $\mathcal{R}_\mu(P) \subseteq P$ by definition of $\mathcal{R}_\mu(P)$, it suffices to prove that $P \subseteq \mathcal{R}_\mu(P)$ iff P constrains at most μ.

$\boxed{\text{1.}}$ If P constrains at most μ, then $P \subseteq \mathcal{R}_\mu(P)$.

Proof: We assume that σ is any behavior in P and construct a μ-strategy f such that $\sigma \in \mathcal{O}_\mu(f)$ and $\mathcal{O}_\mu(f) \subseteq P$. We will define f so it tries to produce the outcome σ and does nothing if this is no longer possible. Since P is a safety property, doing nothing cannot violate P.

1.1. For any finite behavior prefix ρ, define $f(\rho)$ by

if $\rho = \sigma|_m$ and $\mathbf{a}_{m+1}(\sigma) \in \mu$, for some m
then $f(\rho) = (\mathbf{a}_{m+1}(\sigma), \mathbf{s}_{m+1}(\sigma))$
else $f(\rho) = \perp$

Then f is a μ-strategy.

Proof: f is obviously a μ-strategy.

$\boxed{\text{1.2.}}$ $\sigma \in \mathcal{O}_\mu(f)$

Proof: It is immediate from 1.1 (the definition of f) that σ is a μ-outcome of f. It is a fair μ-outcome because $\mathbf{a}_{m+1}(\sigma) \in \neg\mu$ implies that $f(\sigma|_m) = \perp$.

$\boxed{\text{1.3.}}$ $\mathcal{O}_\mu(f) \subseteq P$

Proof: We assume that τ is an arbitrary behavior in $\mathcal{O}_\mu(f)$ but not in P and derive a contradiction.

1.3.1. Let m be the smallest integer such that $\widehat{\tau|_m} \notin P$.

Proof: m exists because P is a safety property and, by hypothesis, $\tau \notin P$.

1.3.2. $m > 0$ and $\mathbf{a}_m(\tau) \in \mu$.

Proof: By 1.3.1 and the hypothesis that P constrains at most μ.

1.3.3. $f(\tau|_{m-1}) = (\mathbf{a}_m(\tau), \mathbf{s}_m(\tau))$

Proof: By 1.3.2 and the hypothesis that τ is a μ-outcome of f.

1.3.4. $(\mathbf{a}_m(\tau), \mathbf{s}_m(\tau)) = (\mathbf{a}_m(\sigma), \mathbf{s}_m(\sigma))$

Proof: By 1.3.3 and 1.1 (the definition of f).

1.3.5. $\tau|_{m-1} = \sigma|_{m-1}$

Proof: By 1.3.3, since 1.1 (the definition of f) implies ρ is in the domain of f iff ρ is a prefix of σ.

1.3.6. $\widehat{\tau|_m} = \widehat{\sigma|_m}$

Proof: By 1.3.4 and 1.3.5.

1.3.7. $\widehat{\sigma|_m} \in P$

Proof: By the hypotheses that P is a safety property and $\sigma \in P$.

$\boxed{1.3.8.}$ Contradiction.

Proof: 1.3.1, 1.3.6, and 1.3.7.

$\boxed{2.}$ If $P \subseteq \mathcal{R}_\mu(P)$, then P constrains at most μ.

Proof: We assume that P does not constrain at most μ and prove that there exists a behavior in P that is not in $\mathcal{R}_\mu(P)$. The behavior will be one in which the environment could have taken a step that would have violated P, but chose not to. Thus, the behavior cannot be produced by a winning strategy for μ.

2.1. Choose $\sigma \notin P$ and $m \geq 0$ such that $\widehat{\sigma|_m} \notin P$ and either (i) $m = 0$ or (ii) $\widehat{\sigma|_{m-1}} \in P$ and $\mathbf{a}_m(\sigma) \notin \mu$.

Proof: Such a σ exists by the assumption that P does not constrain at most μ.

2.2. If $m = 0$ then $\mathcal{R}_\mu(P) = \emptyset$.

Proof: We assume $m = 0$ and prove that $\mathcal{R}_\mu(P) = \emptyset$. The proof involves showing that if there is some initial state in which the system loses, then it has no winning strategy.

2.2.1. For any behavior τ, if $\mathbf{s}_0(\tau) = \mathbf{s}_0(\sigma)$ then $\tau \notin P$.

Proof: 2.1 and the hypothesis $m = 0$ imply $\widehat{\sigma|_0} \notin P$. Hence, $\mathbf{s}_0(\tau) = \mathbf{s}_0(\sigma)$ implies $\widehat{\tau|_0} \notin P$, which implies $\tau \notin P$ because P is a safety property.

2.2.2. For any state s and any μ-strategy f, there exists a behavior $\tau \in \mathcal{O}_\mu(f)$ such that $\mathbf{s}_0(\tau) = s$.

Proof: Let s be any state and let α be any agent not in μ. Define τ inductively by letting $\mathbf{s}_0(\tau) = s$ and for $i > 0$, if $\tau|_{i-1}$ is in the domain of f, then $(\mathbf{a}_i(\tau), \mathbf{s}_i(\tau)) = f(\tau|_{i-1})$, otherwise $\mathbf{a}_i(\tau) = \alpha$ and $\mathbf{s}_i(\tau) = \mathbf{s}_{i-1}(\tau)$.

$\boxed{2.2.3.}$ $\mathcal{R}_\mu(P) = \emptyset$

Proof: 2.2.1 and 2.2.2 imply that there exists no μ-strategy f with $\mathcal{O}_\mu(f) \subseteq P$.

2.3. If $m > 0$, then $\widehat{\sigma|_{m-1}}$ is in P but not in $\mathcal{R}_\mu(P)$.

Proof: Assume $m > 0$, so $\widehat{\sigma|_{m-1}} \in P$ by 2.1. Let f be any μ-strategy with $\widehat{\sigma|_{m-1}} \in \mathcal{O}_\mu(f)$. We prove that there exists a behavior τ in $\mathcal{O}_\mu(f)$ that is not in P. We will take τ to be an outcome in which $\sigma|_{m-1}$ is produced and then the environment adds $(\mathbf{a}_m(\sigma), \mathbf{s}_m(\sigma))$.

2.3.1. Let α be any agent in $\neg\mu$, and let τ be the behavior such that $\tau|_m = \sigma|_m$ and, for all $i > m$, if $\tau|_{i-1}$ is in the domain of f, then $(\mathbf{a}_i(\tau), \mathbf{s}_i(\tau)) = f(\tau|_{i-1})$, otherwise $\mathbf{a}_i(\tau) = \alpha$ and $\mathbf{s}_i(\tau) = \mathbf{s}_{i-1}(\tau)$.

$\boxed{2.3.2.}$ $\tau \in \mathcal{O}_\mu(f)$

Proof: To prove that τ is a μ-outcome of f, we must show that for all $i > 0$, if $\mathbf{a}_i(\tau) \in \mu$ then $(\mathbf{a}_i(\tau), \mathbf{s}_i(\tau)) = f(\tau|_{i-1})$. For $i < m$, this follows because $\tau|_i = \sigma|_i$ (by 2.3.1) and $\widehat{\sigma|_{m-1}}$ is a μ-outcome of f (by hypothesis). For $i = m-1$, it follows because $\mathbf{a}_m(\sigma) \notin \mu$ (by 2.1 and the assumption that $m > 0$). For $i > m$, it follows immediately from the definition of τ. Fairness follows from the definition of τ.

$\boxed{2.3.3.}$ $\tau \notin P$

Proof: $\widehat{\sigma|_m} \notin P$ by 2.1 (the choice of σ), and $\tau|_m = \sigma|_m$ by 2.3.1 (the definition of τ), so $\widehat{\tau|_m} \notin P$. Since P is a safety property, this implies $\tau \notin P$.

$\boxed{2.4.}$ There exists a behavior in P that is not in $\mathcal{R}_\mu(P)$.

Proof: If $m > 0$, this follows from 2.3. If $m = 0$, it follows from 2.2 and the hypothesis that P is nonempty.

End Proof of Proposition 6

Proposition 7 *For any agent set μ, if P is a μ-realizable property then $\overline{\mathcal{R}_\mu(P)}$ constrains at most μ.*

Proof of Proposition 7

1. For any property Q, if Q is a safety property then $\mathcal{R}_\mu(Q)$ is a safety property.
   Proof: By Proposition 5, $\mathcal{R}_\mu(Q) = \overline{\mathcal{R}_\mu(Q)} \cap Q$. The conjunction of safety properties is a safety property (since safety properties are closed sets), so $\mathcal{R}_\mu(Q)$ is a safety property.
2. $\mathcal{R}_\mu(\overline{\mathcal{R}_\mu(P)}) = \overline{\mathcal{R}_\mu(P)}$

$\boxed{2.1.}$ $\mathcal{R}_\mu(\overline{\mathcal{R}_\mu(P)}) \subseteq \overline{\mathcal{R}_\mu(P)}$

Proof: By definition of $\mathcal{R}_\mu$.

$\boxed{2.2.}$ $\overline{\mathcal{R}_\mu(P)} \subseteq \mathcal{R}_\mu(\overline{\mathcal{R}_\mu(P)})$

2.2.1. $\mathcal{R}_\mu(\mathcal{R}_\mu(P)) \subseteq \mathcal{R}_\mu(\overline{\mathcal{R}_\mu(P)})$

Proof: By monotonicity of $\mathcal{R}$, since $Q \subseteq \overline{Q}$ for any property Q.

2.2.2. $\mathcal{R}_\mu(P) \subseteq \mathcal{R}_\mu(\overline{\mathcal{R}_\mu(P)})$

Proof: By 2.2.1 and Proposition 4.

2.2.3. $\mathcal{R}_\mu(\overline{\mathcal{R}_\mu(P)})$ is a closed set.

Proof: By 1.

$\boxed{2.2.4.}$ $\overline{\mathcal{R}_\mu(P)} \subseteq \mathcal{R}_\mu(\overline{\mathcal{R}_\mu(P)})$

Proof: By 2.2.2 and 2.2.3, since $\overline{\mathcal{R}_\mu(P)}$ is by definition the smallest closed set containing $\mathcal{R}_\mu(P)$.

$\boxed{3.}$ $\overline{\mathcal{R}_\mu(P)}$ constrains at most μ.

Proof: By 2, Proposition 6, and the hypothesis that P is μ-realizable, so $\mathcal{R}_\mu(P)$ is nonempty.

End Proof of Proposition 7

Lemma 4 *Let μ be an agent set, and let E and M be properties such that:*

1. *$E = I \cap P$, where*
   (a) *I is a state predicate.*
   (b) *P is a safety property that constrains at most $\neg\mu$.*
2. *$\overline{M}$ constrains at most μ.*

For any behavior σ and $i \geq 0$, if $\sigma \in \mathcal{R}_\mu(E \Rightarrow M)$ and $\widehat{\sigma|_i} \notin \overline{M}$, then $i > 0$ and $\widehat{\sigma|_{i-1}} \notin E$.

Proof of Lemma 4

Proof: We assume $\sigma \in \mathcal{R}_\mu(E \Rightarrow M)$ and $\widehat{\sigma|_i} \notin \overline{M}$, and we prove that $i > 0$ and $\widehat{\sigma|_{i-1}} \notin E$.

$\boxed{1.}$ $i > 0$

Proof: By the assumption $\widehat{\sigma|_i} \notin \overline{M}$, the hypothesis that $\overline{M}$ constrains at most μ, and the definition of *constrains at most*.

2. For all $j \geq 0$, if $\widehat{\sigma|_j} \notin \overline{M}$ then $\widehat{\sigma|_j} \notin E$.

Proof: We assume $\widehat{\sigma|_j} \notin \overline{M}$ and $\widehat{\sigma|_j} \in E$, and obtain a contradiction. We will first construct a behavior τ that equals σ for its first j steps, after which it follows a strategy that puts it in $E \Rightarrow M$, taking $\neg\mu$-stuttering steps whenever the strategy is undefined. We will then construct ϕ by changing those $\neg\mu$-stuttering steps to μ-stuttering steps. This ϕ will not be in M because σ violates M by its j^{th} step, and it will be in E because it will not have any $\neg\mu$ steps that can violate E, so $\phi \notin (E \Rightarrow M)$. This will lead to a contradiction because $\tau \in (E \Rightarrow M)$ and $\phi \simeq \tau$.

2.1. Choose a behavior τ such that:
   (a) $\tau|_j = \sigma|_j$
   (b) $\tau \in (E \Rightarrow M)$
   (c) $\mathbf{a}_{k+1}(\tau) \in \neg\mu$ implies $\mathbf{s}_{k+1}(\tau) = \mathbf{s}_k(\tau)$, for all $k \geq j$.

2.1.1. Choose a μ-strategy f such that $\sigma \in \mathcal{O}_\mu(f) \subseteq (E \Rightarrow M)$.

Proof: f exists by the assumption $\sigma \in \mathcal{R}_\mu(E \Rightarrow M)$ and the definition of $\mathcal{R}_\mu$.

2.1.2. Choose $\beta_{\neg\mu} \in \neg\mu$ and define τ by $\tau|_j = \sigma|_j$ and, for all $k \geq j$:

$$\begin{array}{l} \textbf{if } f(\tau|_k) \neq \bot \textbf{ then } (\mathbf{a}_{k+1}(\tau), \mathbf{s}_{k+1}(\tau)) = f(\tau|_k) \\ \qquad\qquad\qquad\quad \textbf{else } (\mathbf{a}_{k+1}(\tau), \mathbf{s}_{k+1}(\tau)) = (\beta_{\neg\mu}, \mathbf{s}_k(\tau)) \end{array}$$

Proof: $\beta_{\neg\mu}$ exists by the assumption that μ is an agent set, which implies that $\neg\mu$ is nonempty.

$\boxed{2.1.3.}$ $\tau|_j = \sigma|_j$

Proof: By 2.1.2 (the definition of τ).

2.1.4. $\tau \in \mathcal{O}_\mu(f)$

Proof: To show that τ is a μ-outcome of f, we must show that $\tau|_k$ ends according to f for all $k \geq 0$. For $k < j$, this follows from 2.1.1 and 2.1.3. For $k \geq j$, it follows from 2.1.2. The outcome τ is fair because 2.1.2 implies that, for all $k \geq j$, the strategy f is undefined on $\tau|_k$ iff $\mathbf{a}_{k+1}(\tau) \in \neg\mu$.

$\boxed{2.1.5.}$ $\tau \in (E \Rightarrow M)$

Proof: From 2.1.1 (the choice of f) and 2.1.4.

$\boxed{2.1.6.}$ $\mathbf{a}_{k+1}(\tau) \in \neg\mu$ implies $\mathbf{s}_{k+1}(\tau) = \mathbf{s}_k(\tau)$, for all $k \geq j$.

Proof: By 2.1.2 (the definition of τ) and 2.1.1, which asserts that f is a μ-strategy.

2.2. Choose a behavior ϕ such that:

(a) $\phi|_j = \sigma|_j$

(b) $\phi \simeq \tau$

(c) $\mathbf{a}_{k+1}(\phi) \in \mu$, for all $k \geq j$.

2.2.1. Choose $\beta_\mu \in \mu$ and define ϕ by $\phi|_j = \tau|_j$ and, for all $k \geq j$:

$\mathbf{s}_{k+1}(\phi) = \mathbf{s}_k(\tau)$

if $\mathbf{s}_{k+1}(\tau) \neq \mathbf{s}_k(\tau)$ **then** $\mathbf{a}_{k+1}(\phi) = \mathbf{a}_{k+1}(\tau)$

else $\mathbf{a}_{k+1}(\phi) = \beta_\mu$

Proof: β_μ exists by the assumption that μ is an agent set and therefore nonempty.

$\boxed{2.2.2.}$ $\phi|_j = \tau|_j$

Proof: By 2.2.1 and 2.1(a).

$\boxed{2.2.3.}$ $\phi \simeq \tau$

Proof: By 2.2.1, since ϕ is obtained from τ by changing only agents on stuttering steps.

$\boxed{2.2.4.}$ $\mathbf{a}_{k+1}(\phi) \in \mu$, for all $k \geq j$.

Proof: By 2.2.1 (the definition of ϕ) and 2.1(c).

2.3. $\phi \in E$

2.3.1. $\widehat{\phi|_j} \in E$

Proof: By 2.2(a) and the assumption $\widehat{\sigma|_j} \in E$.

2.3.2. $\phi \in I$

Proof: By 2.3.1 and the hypotheses that I is a state predicate and $E = I \cap P$.

2.3.3. $\phi \in P$

Proof: By 2.3.1, 2.2(c), and the hypotheses that P constrains at most $\neg\mu$ and $E = I \cap P$.

$\boxed{2.3.4.}$ $\phi \in E$

Proof: By 2.3.2 and 2.3.3, since $E = I \cap P$ by hypothesis.

2.4. $\phi \notin M$

2.4.1. $\widehat{\phi|_j} \notin \overline{M}$

Proof: 2.2(a) and the assumption $\widehat{\sigma|_j} \notin \overline{M}$.

2.4.2. $\phi \notin \overline{M}$

Proof: By 2.4.1, since $\overline{M}$ is a safety property.

$\boxed{2.4.3.}$ $\phi \notin M$

Proof: By 2.4.2, since $M \subseteq \overline{M}$.

2.5. $\phi \notin (E \Rightarrow M)$

Proof: By 2.3 and 2.4.

2.6. $\tau \notin (E \Rightarrow M)$

Proof: By 2.2(b), 2.5, and the hypothesis that E and M are properties.

$\boxed{2.7.}$ Contradiction.

Proof: By 2.6 and 2.1(b).

$\boxed{3.}$ $\widehat{\sigma|_{i-1}} \notin E$

Proof: Let j be the smallest natural number such that $\widehat{\sigma|_j} \notin \overline{M}$. The hypothesis $\widehat{\sigma|_i} \notin \overline{M}$ implies $j \leq i$. We now consider two cases.

Case 3A. $j < i$

3A.1. $\widehat{\sigma|_k} \notin \overline{M}$, for all $k \geq j$.

Proof: $\widehat{\sigma|_k} \notin \overline{M}$ by definition of j, and $\overline{M}$ is a safety property.

3A.2. $\widehat{\sigma|_{i-1}} \notin \overline{M}$

Proof: By 3A.1 and the assumption $j < i$.

$\boxed{3A.3.}$ $\widehat{\sigma|_{i-1}} \notin E$

Proof: By 3A.2 and 2.

Case 3B. $j = i$

3B.1. $\widehat{\sigma|_{i-1}} \in \overline{M}$

Proof: By definition of j and the assumption $j = i$, since $i > 0$ by 1.

3B.2. $\mathbf{a}_i(\sigma) \in \mu$

Proof: By 3B.1, the assumption $\widehat{\sigma|_i} \notin \overline{M}$, and the hypothesis that $\overline{M}$ constrains at most μ.

3B.3. $\widehat{\sigma|_i} \notin E$

Proof: By 2 and the assumption $\widehat{\sigma|_i} \notin \overline{M}$.

$\boxed{3B.4.}$ $\widehat{\sigma|_{i-1}} \notin E$

Proof: By 3B.2, 3B.3, and the hypothesis that E constrains at most $\neg\mu$.

End Proof of Lemma 4

Proposition 8 *Let μ be an agent set, I a state predicate, P a safety property that constrains at most $\neg\mu$, and Q a safety property that constrains at most μ. Then $\mathcal{R}_\mu(I \cap P \Rightarrow Q)$ equals $I \cap P \mathrel{-\!\!\triangleright} Q$.*

Proof of Proposition 8

$\boxed{1.}$ $\mathcal{R}_\mu(I \cap P \Rightarrow Q) \subseteq (I \cap P \mathrel{-\!\!\triangleright} Q)$

Proof: We assume that there exists a behavior σ such that $\sigma \in \mathcal{R}_\mu(I \cap P \Rightarrow Q)$ and $\sigma \notin (I \cap P \mathrel{-\!\!\triangleright} Q)$, and we obtain a contradiction.

1.1. Let i be the smallest integer such that $\widehat{\sigma|_i} \notin (I \cap P \mathrel{-\!\!\triangleright} Q)$.
Proof: i exists by the hypothesis $\sigma \notin (I \cap P \mathrel{-\!\!\triangleright} Q)$ and the definition of $I \cap P \mathrel{-\!\!\triangleright} Q$.

1.2. $\widehat{\sigma|_i} \notin Q$
Proof: By 1.1.

1.3. $\widehat{\sigma|_{i-1}} \in P$
Proof: By 1.1, which implies $\widehat{\sigma|_i} \in P$, since P is a safety property and $(\sigma|_i)|_{i-1}$ equals $\sigma|_{i-1}$.

$\boxed{1.4.}$ Contradiction.
Proof: By 1.2, 1.3, the hypothesis $\sigma \in \mathcal{R}_\mu(I \cap P \Rightarrow Q)$, and Lemma 4 (substituting Q for M).

$\boxed{2.}$ $(I \cap P \mathrel{-\!\!\triangleright} Q) \subseteq \mathcal{R}_\mu(I \cap P \Rightarrow Q)$

2.1. $(I \cap P \mathrel{-\!\!\triangleright} Q) \subseteq (I \cap P \Rightarrow Q)$
Proof: By definition of $(I \cap P \mathrel{-\!\!\triangleright} Q)$

2.2. $\mathcal{R}_\mu(I \cap P \mathrel{-\!\!\triangleright} Q) \subseteq \mathcal{R}_\mu(I \cap P \Rightarrow Q)$
Proof: By 2.1 and Proposition 3.

2.3. $I \cap P \mathrel{-\!\!\triangleright} Q$ constrains at most μ.
Proof: $I \cap P \mathrel{-\!\!\triangleright} Q$ is a safety property and $Q \subseteq (I \cap P \mathrel{-\!\!\triangleright} Q)$ by definition of $I \cap P \mathrel{-\!\!\triangleright} Q$. Since Q constrains at most μ by hypothesis, any safety property containing Q also constrains at most μ.

2.4. $I \cap P \mathrel{-\!\!\triangleright} Q$ is nonempty.
Proof: Q is nonempty by the hypothesis that it constrains at most μ, and Q is a subset of $I \cap P \mathrel{-\!\!\triangleright} Q$.

2.5. $\mathcal{R}_\mu(I \cap P \mathrel{-\!\!\triangleright} Q) = I \cap P \mathrel{-\!\!\triangleright} Q$
Proof: By 2.3, 2.4, and Proposition 6.

$\boxed{2.6.}$ $(I \cap P \mathrel{-\!\!\triangleright} Q) \subseteq \mathcal{R}_\mu(I \cap P \Rightarrow Q)$
Proof: By 2.2 and 2.5.

End Proof of Proposition 8

Proposition 9 *For any agent set μ, safety property M, and arbitrary property L, the following three conditions are equivalent:*

(a) For every finite behavior ρ such that $\widehat{\rho} \in M$, there exist a μ-strategy f with $\mathcal{O}_\mu(f) \subseteq M \cap L$ and a behavior $\sigma \in \mathcal{O}_\mu(f)$ with ρ a prefix of σ.

(b) For every finite behavior ρ such that $\widehat{\rho} \in M$, there exist a μ-strategy f with $\mathcal{O}_\mu(f) \subseteq M \cap L$ and a behavior $\sigma \in \mathcal{O}_\mu(f)$ with ρ stuttering-equivalent to a prefix of σ.

(c) The pair $(M,\ M \cap L)$ is machine-closed, and $M \cap L$ is μ-receptive.

Proof of Proposition 9

It is obvious that (a) implies (b). We prove that (b) implies (c), and that (c) implies (a). We first assume that (b) holds and prove (c).

1. $M \subseteq \overline{\mathcal{R}_\mu(M \cap L)}$

   Proof: We assume $\sigma \in M$ and prove that $\sigma \in \overline{\mathcal{R}_\mu(M \cap L)}$. By definition of the topology, it suffices to assume that $i > 0$ and prove that there exists a behavior $\tau \in \mathcal{R}_\mu(M \cap L)$ such that $\sigma|_i \simeq \tau|_j$, for some j.

   1.1. $\widehat{\sigma|_i} \in M$

   Proof: Since M is closed by hypothesis, and the definition of the topology.

   1.2. Choose a μ-strategy f with $\mathcal{O}_\mu(f) \subseteq M \cap L$ and a behavior $\tau \in \mathcal{O}_\mu(f)$ such that $\sigma|_i \simeq \tau|_j$ for some j.

   Proof: By (b) and 1.1, substituting $\sigma|_i$ for ρ and τ for σ in (b).

   $\boxed{1.3.}$ $\tau \in \mathcal{R}_\mu(M \cap L)$ and $\sigma|_i \simeq \tau|_j$, for some j.

   Proof: By 1.2, since $\mathcal{O}_\mu(f) \subseteq M \cap L$ implies $\mathcal{O}_\mu(f) \subseteq \mathcal{R}_\mu(M \cap L)$ by definition of $\mathcal{R}_\mu$.

$\boxed{2.}$ The pair $(M, M \cap L)$ is machine-closed.

   Proof: We must prove that $M = \overline{M \cap L}$.

   $\boxed{2.1.}$ $\overline{M \cap L} \subseteq M$

   2.1.1. $\overline{M \cap L} \subseteq \overline{M}$

   Proof: By monotonicity of closure.

   2.1.2. $\overline{M} = M$

   Proof: By hypothesis, M is a safety property.

   $\boxed{2.1.3.}$ $\overline{M \cap L} \subseteq M$

   Proof: By 2.1.1 and 2.1.2.

   2.2. $\overline{\mathcal{R}_\mu(M \cap L)} \subseteq \overline{M \cap L}$

   Proof: Since $\mathcal{R}_\mu(U) \subseteq U$ for any property U, and closure is monotone.

   $\boxed{2.3.}$ $M \subseteq \overline{M \cap L}$

   Proof: By 1 and 2.2.

$\boxed{3.}$ $M \cap L$ is μ-receptive.

   Proof: By definition, we must prove that $M \cap L = \mathcal{R}_\mu(M \cap L)$.

   3.1. $M \cap L \subseteq \overline{\mathcal{R}_\mu(M \cap L)}$

   Proof: By 1.

   3.2. $\overline{\mathcal{R}_\mu(M \cap L)} \cap M \cap L = M \cap L$

   Proof: By 3.1.

   3.3. $\mathcal{R}_\mu(M \cap L) = \overline{\mathcal{R}_\mu(M \cap L)} \cap M \cap L$

   Proof: By Proposition 5.

   $\boxed{3.4.}$ $M \cap L = \mathcal{R}_\mu(M \cap L)$

   Proof: By 3.2 and 3.3.

We now assume that (c) holds and prove (a). Let ρ be a finite behavior with $\widehat{\rho} \in M$. We must find a μ-strategy f with $\mathcal{O}_\mu(f) \subseteq M \cap L$ and a behavior $\sigma \in \mathcal{O}_\mu(f)$ with ρ a prefix of σ.

$\boxed{1.}$ Choose $\sigma \in M \cap L$ such that ρ is a prefix of σ.

Proof: σ exists by the machine-closure hypothesis ($M = \overline{M \cap L}$).

2. $\sigma \in \mathcal{R}_\mu(M \cap L)$

   Proof: By 1 and the hypothesis that $M \cap L$ is receptive.

$\boxed{3.}$ There exists a μ-strategy f such that $\sigma \in \mathcal{O}_\mu(f)$ and $\mathcal{O}_\mu(f) \subseteq M \cap L$.

   Proof: By 2 and the definition of $\mathcal{R}_\mu(M \cap L)$.

End Proof of Proposition 9

Proposition 10 *Let μ be an agent set, let* $\mathbf{x}$ *be the projection function from* $\mathbf{S} \times \mathbf{X}$ *to* $\mathbf{X}$*, and let $I_{\mathbf{X}}$ be an* $\mathbf{S} \times \mathbf{X}$*-predicate, $\mathcal{N}$ a next-state relation on* $\mathbf{S} \times \mathbf{X}$*, and L an* $\mathbf{S} \times \mathbf{X}$*-property. Let M equal (3) and let P equal (2). Assume that:*

(a) For all $s \in \mathbf{S}$ there exists $x \in \mathbf{X}$ such that $(s, x) \in I_{\mathbf{X}}$.

(b) The pair $(\mathcal{TA}_\mu(\mathcal{N}),\ \mathcal{TA}_\mu(\mathcal{N}) \cap (I_{\mathbf{X}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{X}}) \Rightarrow L))$ is μ-machine-realizable.

(c) M is a safety property.

Then (M, P) is μ-machine-realizable.

Proof of Proposition 10

By part (b) of Proposition 9, it suffices to assume that ρ is a finite behavior prefix such that $\widehat{\rho} \in \exists \mathbf{x} : I_{\mathbf{X}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{X}}) \cap \mathcal{TA}_\mu(\mathcal{N})$ and to construct a μ-strategy f such that $\mathcal{O}_\mu(f)$ is a subset of $\exists \mathbf{x} : I_{\mathbf{X}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{X}}) \cap \mathcal{TA}_\mu(\mathcal{N}) \cap L$, and a behavior $\sigma \in \mathcal{O}_\mu(f)$ such that $\rho \simeq \sigma|_j$ for some j. To construct f and σ, we will first choose a strategy g whose outcomes all lie in $\mathcal{TA}_\mu(\mathcal{N}) \cap (I_{\mathbf{X}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{X}}) \Rightarrow L)$ and a behavior ϕ produced by g whose projection (by $\Pi_{\mathbf{S}}$) has ρ as a prefix. We will then define an "inverse projection" Ψ from $\mathbf{S}$-behaviors to $\mathbf{S} \times \mathbf{X}$-behaviors whose image contains ϕ, and will define f to be g composed with Ψ and σ to be the projection of ϕ.

1. Choose $\phi \in I_{\mathbf{X}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{X}}) \cap \mathcal{TA}_\mu(\mathcal{N})$ such that $\Pi_S(\phi) \simeq \widehat{\rho}$.

   Proof: The existence of ϕ follows from the hypothesis that $\widehat{\rho} \in \exists \mathbf{x} : I_{\mathbf{X}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{X}}) \cap \mathcal{TA}_\mu(\mathcal{N})$ and the definition of existential quantification.

2. Choose j such that $\Pi_S(\phi|_j) \simeq \rho$.

   Proof: j exists by 1, which asserts that $\Pi_S(\phi) \simeq \widehat{\rho}$.

3. Choose a μ-strategy g such that $\mathcal{O}_\mu(g) \subseteq \mathcal{TA}_\mu(\mathcal{N}) \cap (I_{\mathbf{X}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{X}}) \Rightarrow L)$ and $\phi|_j$ is a partial μ-outcome of g.

   Proof: By hypothesis (b), the definition of machine-realizability, and part (a) of Proposition 9 (with $\phi|_j$ substituted for ρ).

4. Choose a behavior $\tau \in \mathcal{O}_\mu(g)$ such that $\tau|_j = \phi|_j$ and $\tau \in I_{\mathbf{X}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{X}}) \cap \mathcal{TA}_\mu(\mathcal{N}) \cap L$.

   $\boxed{4.1.}$ Define τ inductively as follows, where $\beta_{\neg\mu}$ is any element of $\neg\mu$.

   if $i \leq j$ **then** $\mathbf{a}_i(\tau) = \mathbf{a}_i(\phi)$ and $\mathbf{s}_i(\tau) = \mathbf{s}_i(\phi)$

   if $i > j$ **then if** $g(\tau|_{i-1}) = \perp$
   then $(\mathbf{a}_i(\tau), \mathbf{s}_i(\tau)) = (\beta_{\neg\mu}, \mathbf{s}_{i-1}(\tau))$
   else $(\mathbf{a}_i(\tau), \mathbf{s}_i(\tau)) = g(\tau|_{i-1})$

Then $\tau|_j = \phi|_j$ and $\tau \in \mathcal{O}_\mu(g)$.

Proof: By construction, $\tau|_j = \phi|_j$. Since $\phi|_j$ is a partial μ-outcome of g (by 3), the definition of τ implies that τ is a μ-outcome. It is a fair μ-outcome because, for all $i > j$, $\mathbf{a}_i(\tau) \in \neg\mu$ iff $g(\tau|_{i-1}) = \bot$.

$\boxed{4.2.}$ $\tau \in I_{\mathbf{X}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{X}}) \cap \mathcal{TA}_\mu(\mathcal{N}) \cap L$

Proof: By 4.1 and 3, $\tau \in \mathcal{O}_\mu(g) \subseteq \mathcal{TA}_\mu(\mathcal{N}) \cap (I_{\mathbf{X}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{X}}) \Rightarrow L)$. It therefore suffices to prove that $\tau \in I_{\mathbf{X}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{X}})$, which means proving that (i) $\mathbf{s}_0(\tau) \in I_{\mathbf{X}}$ and (ii) $\mathbf{a}_i(\tau) \in \neg\mu$ implies $\Pi_{\mathbf{X}}(\mathbf{s}_{i-1}(\tau)) = \Pi_{\mathbf{X}}(\mathbf{s}_i(\tau))$. But (i) holds because $\mathbf{s}_0(\tau) = \mathbf{s}_0(\phi)$ and $\mathbf{s}_0(\phi) \in I_{\mathbf{X}}$ by 1. Condition (ii) follows for $i \leq j$ by 1, since $\tau|_j = \phi|_j$ by 4.1. For $i > j$, (ii) follows immediately from the definition of τ.

5. Choose a monotone mapping Ψ from behavior prefixes with state space $\mathbf{S}$ to behavior prefixes with state space $\mathbf{S} \times \mathbf{X}$ such that
   (a) For any finite behavior prefix η
      (i) $\Pi_{\mathbf{S}}(\Psi(\eta)) = \eta$
      (ii) $\mathbf{s}_0(\Psi(\eta)) \in I_{\mathbf{X}}$
      (iii) For all $i > 0$, if $\mathbf{a}_i(\Psi(\eta)) \in \neg\mu$ then $(\mathbf{s}_{i-1}(\Psi(\eta)), \mathbf{s}_i(\Psi(\eta))) \in \mathcal{U}_{\mathbf{X}}$.
      (iv) For any agent α and state s in $\mathbf{S}$, if $\Pi_{\mathbf{S}}(g(\Psi(\eta))) = (\alpha, s)$, then $\Psi(\eta \cdot (\alpha, s)) = \Psi(\eta) \cdot g(\Psi(\eta))$.
   (b) $\Psi(\xi) = \lim_{m\to\infty} \Psi(\xi|_m)$ for any behavior ξ.
   (c) $\Psi(\Pi_{\mathbf{S}}(\tau|_i)) = \tau|_i$, for all $i \geq 0$.

Proof: We define $\Psi(\eta)$ for any finite behavior prefix η by induction on $|\eta|$ as follows.

if $|\eta| = 0$
  then if $\mathbf{s}_0(\eta) = \Pi_{\mathbf{S}}(\mathbf{s}_0(\tau))$
    then $\mathbf{s}_0(\Psi(\eta)) = \mathbf{s}_0(\tau)$
    else $\mathbf{s}_0(\Psi(\eta)) = (\mathbf{s}_0(\eta), x)$ for any x with $(\mathbf{s}_0(\eta), x) \in I_{\mathbf{X}}$.
if $\eta = \theta \cdot (\alpha, s)$
  then if $\alpha \in \neg\mu$
    then $\Psi(\eta) = \Psi(\theta) \cdot (\alpha, (s, \Pi_{\mathbf{X}}(\Psi(\theta)_{\mathbf{s}})))$
    else **if** $(\alpha, s) = \Pi_{\mathbf{S}}(g(\Psi(\theta)))$
      then $\Psi(\eta) = \Psi(\theta) \cdot g(\Psi(\theta))$
      else $\Psi(\eta) = \Psi(\theta) \cdot (\alpha, (s, x))$ for any x in $\mathbf{X}$.

Note that in the case $|\eta| = 0$, the x chosen in the **else** clause exists by hypothesis (a) of the Proposition. We take (b) as the definition of Ψ for behaviors. The monotonicity of Ψ is immediate from the definition, so the limit in (b) exists. Property (a)(ii) follows from the case $|\eta| = 0$ in the definition of $\Psi(\eta)$. Properties (a)(i), (a)(iii), and (a)(iv) follow from the definition of Ψ by induction on $|\eta|$. The proof of (c) is by induction on i. For $i = 0$, it follows immediately from the definition of Ψ. For the induction step, we assume $\tau|_i = \tau|_{i-1} \cdot (\alpha, (s, x))$ and $\Psi(\Pi_{\mathbf{S}}(\tau|_{i-1})) = \tau|_{i-1}$ and prove $\Psi(\Pi_{\mathbf{S}}(\tau|_i)) = \tau|_i$. If $\alpha \in \mu$, then this follows from the definition of Ψ because $\tau \in \mathcal{O}_\mu(g)$ (by 4). If $\alpha \notin \mu$, then it follows from the definition of Ψ because $\tau \in \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{X}})$ (also by 4).

6. For any finite behavior prefix η with states in $\mathbf{S}$, define $f(\eta)$ to equal $\Pi_{\mathbf{S}}(g(\Psi(\eta)))$, where $\Pi_{\mathbf{S}}$ is extended to a mapping from $\mathbf{A} \times (\mathbf{S} \times \mathbf{X})$ to $\mathbf{A} \times \mathbf{S}$ in the obvious way. Then f is a μ-strategy, and $\xi \in \mathcal{O}_\mu(f)$ implies $\Psi(\xi) \in \mathcal{O}_\mu(g)$.

   6.1. $\eta_{\mathbf{a}} = \Psi(\eta)_{\mathbf{a}}$, for any finite behavior prefix η.
   Proof: By 5(a)(i) and the definition of $\Pi_{\mathbf{S}}$.

   $\boxed{6.2.}$ f is a μ-strategy.
   Proof: By 6.1, since g is a μ-strategy (by 3).

   6.3. For any finite behavior prefix η, if η ends according to μ, f then $\Psi(\eta)$ ends according to μ, g.
   Proof: If $\eta_{\mathbf{a}} \in \neg\mu$, then this follows from 6.1. If $\eta_{\mathbf{a}} \in \mu$, then it follows from 5(a)(iv).

   $\boxed{6.4.}$ If $\xi \in \mathcal{O}_\mu(f)$ then $\Psi(\xi) \in \mathcal{O}_\mu(g)$.
   Proof: Assume that $\xi \in \mathcal{O}_\mu(f)$. Since ξ is a μ-outcome of f, 6.3 implies that $\Psi(\xi)$ is a μ-outcome of g. Since $f(\eta) = \perp$ iff $g(\Psi(\eta)) = \perp$, 6.1 and the fairness of ξ implies that $\Psi(\xi)$ is also fair.

7. If $\xi \in \mathcal{O}_\mu(f)$ then $\Psi(\xi) \in I_{\mathbf{X}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{X}}) \cap \mathcal{TA}_\mu(\mathcal{N}) \cap L$.
   Proof: By 3 and 6, $\Psi(\xi) \in \mathcal{TA}_\mu(\mathcal{N}) \cap (I_{\mathbf{X}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{X}}) \Rightarrow L)$. By 5(a)(ii) and 5(a)(iii), $\Psi(\xi) \in I_{\mathbf{X}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{X}})$.

$\boxed{8.}$ $\mathcal{O}_\mu(f) \subseteq \exists\mathbf{x} : I_{\mathbf{X}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{X}}) \cap \mathcal{TA}_\mu(\mathcal{N}) \cap L$
   Proof: By definition of existential quantification, it suffices to prove that for every behavior $\xi \in \mathcal{O}_\mu(f)$ there exists a behavior $\xi' \in I_{\mathbf{X}} \cap \mathcal{TA}_{\neg\mu}(\mathcal{U}_{\mathbf{X}}) \cap \mathcal{TA}_\mu(\mathcal{N}) \cap L$ such that $\Pi_{\mathbf{S}}(\xi') = \xi$. By 5(a)(i) and 7, we can let ξ' equal $\Psi(\xi)$.

$\boxed{9.}$ Let $\sigma = \Pi_{\mathbf{S}}(\tau)$. Then $\sigma \in \mathcal{O}_\mu(f)$ and $\rho \simeq \sigma|_j$.

   9.1. $f(\sigma|_i) = \Pi_{\mathbf{S}}(g(\tau|_i))$, for all $i \geq 0$.
   Proof: By 5(c) and 6 (the definition of f).

   9.2. For all $i \geq 0$, if $\mathbf{a}_{i+1}(\sigma) \in \mu$ then $\sigma|_{i+1} = \sigma|_i \cdot f(\sigma|_i)$.
   Proof: By 4, $\tau \in \mathcal{O}_\mu(g)$. Therefore, 5(a)(i) implies that if $\mathbf{a}_{i+1}(\sigma) \in \mu$ then $\tau|_{i+1} = \tau|_i \cdot g(\tau|_i)$. Hence, $\Pi_{\mathbf{S}}(\tau|_{i+1})$, which by definition equals $\sigma|_{i+1}$, is equal to $\Pi_{\mathbf{S}}(\tau|_i \cdot g(\tau|_i))$. The desired result now follows from 9.1.

   $\boxed{9.3.}$ $\sigma \in \mathcal{O}_\mu(f)$
   Proof: By 9.2, σ is a μ-outcome of f. Since 4 asserts that τ is a fair μ-outcome of g, fairness of σ follows from 5(a)(i), 5(c), and 6, which imply that $\mathbf{a}_i(\sigma) \in \mu$ iff $\mathbf{a}_i(\tau) \in \mu$ and that $f(\sigma|_i)$ is defined iff $g(\tau|_i)$ is.

   $\boxed{9.4.}$ $\rho \simeq \sigma|_j$
   Proof: By 2, 4 (which asserts $\tau|_j = \phi|_j$), and the definition of σ.

End Proof of Proposition 10

Theorem 1 *If I is a state predicate, $(E_S, E_S \cap E_L)$ is $\neg\mu$-machine-realizable, M_S is a safety property, and M_L is any property, then*

$$I \cap E_S \cap E_L \;\Rightarrow\; M_S \cap M_L$$

and

$$I \cap E_S \;\Rightarrow\; M_S \cap (E_L \Rightarrow M_L)$$

are μ-equirealizable.

Proof of Theorem 1

We assume that I is a state predicate, $(E_S, E_S \cap E_L)$ is $\neg\mu$-machine-realizable, M_S is a safety property, and M_L is any property, and we prove

$$\mathcal{R}_\mu(I \cap E_S \cap E_L \Rightarrow M_S \cap M_L) = \mathcal{R}_\mu(I \cap E_S \Rightarrow M_S \cap (E_L \Rightarrow M_L))$$

$\boxed{1.}$ $\mathcal{R}_\mu(I \cap E_S \Rightarrow M_S \cap (E_L \Rightarrow M_L)) \subseteq \mathcal{R}_\mu(I \cap E_S \cap E_L \Rightarrow M_S \cap M_L)$
Proof: By Proposition 3, since $(I \cap E_S \Rightarrow M_S \cap (E_L \Rightarrow M_L)) \subseteq (I \cap E_S \cap E_L \Rightarrow M_S \cap M_L)$ by propositional reasoning.

$\boxed{2.}$ $\mathcal{R}_\mu(I \cap E_S \cap E_L \Rightarrow M_S \cap M_L) \subseteq \mathcal{R}_\mu(I \cap E_S \Rightarrow M_S \cap (E_L \Rightarrow M_L))$
Proof: Let f be a μ-strategy such that $\mathcal{O}_\mu(f) \subseteq (I \cap E_S \cap E_L \Rightarrow M_S \cap M_L)$; we must prove that $\mathcal{O}_\mu(f) \subseteq (I \cap E_S \Rightarrow M_S \cap (E_L \Rightarrow M_L))$. We assume $\sigma \in \mathcal{O}_\mu(f)$ and prove that $\sigma \in (I \cap E_S \Rightarrow M_S \cap (E_L \Rightarrow M_L))$. We do this by assuming $\sigma \notin (I \cap E_S \Rightarrow M_S \cap (E_L \Rightarrow M_L))$ and obtaining a contradiction.

The proof rests on the observation that the hypotheses imply $\sigma \in I \cap E_S$, $\sigma \notin M_S$, and $\sigma \notin E_L$. Since M_S is a safety property, σ must violate it at some finite point, while it is still possible for the environment to satisfy E_L. The contradiction is obtained by playing the strategy f, from the point at which σ violates M_S, against an environment strategy h (constructed in step 2.4) that achieves $E_S \cap E_L$ (producing the behavior ϕ of step 2.5). For technical reasons, we replace σ and f in this argument with a behavior $\tau \simeq \sigma$ and a total strategy g, obtained from Lemma 3.

2.1. Choose a total μ-strategy g and a behavior τ such that $\tau \simeq \sigma$, $\tau \in \mathcal{O}_\mu(g)$, and every behavior in $\mathcal{O}_\mu(g)$ is stuttering-equivalent to a behavior in $\mathcal{O}_\mu(f)$.
Proof: g and τ exist by Lemma 3.

2.2. $\tau \in I \cap E_S$, $\tau \notin M_S$, and $\tau \notin E_L$.

2.2.1. $\tau \notin (I \cap E_S \Rightarrow M_S \cap (E_L \Rightarrow M_L))$
Proof: $\tau \simeq \sigma$ by 2.1 (the definition of τ), $\sigma \notin (I \cap E_S \Rightarrow M_S \cap (E_L \Rightarrow M_L))$ by hypothesis, and properties are by definition closed under stuttering-equivalence.

2.2.2. $\tau \in (I \cap E_S \cap E_L \Rightarrow M_S \cap M_L)$
Proof: $\tau \simeq \sigma$ by 2.1, $\sigma \in (I \cap E_S \cap E_L \Rightarrow M_S \cap M_L)$ since $\sigma \in \mathcal{O}_\mu(f) \subseteq (I \cap E_S \cap E_L \Rightarrow M_S \cap M_L)$ by hypothesis, and properties are closed under stuttering-equivalence.

$\boxed{2.2.3.}$ $\tau \in I \cap E_S$, $\tau \notin M_S$, and $\tau \notin E_L$.
Proof: From 2.2.1 and 2.2.2, by propositional reasoning.

2.3. Choose $i \geq 0$ such that $\widehat{\tau|_i} \notin M_S$.
Proof: Such an i exists because M_S is a safety property by hypothesis and $\tau \notin M_S$ by 2.2.

2.4. Choose a $\neg\mu$-strategy h and a behavior $\psi \in \mathcal{O}_{\neg\mu}(h)$ such that $\psi|_i = \tau|_i$ and $\mathcal{O}_{\neg\mu}(h) \subseteq E_S \cap E_L$.

Proof: The existence of h and ψ follows from 2.2, the hypothesis that $(E_S, E_S \cap E_L)$ is $\neg\mu$-machine-realizable, and Proposition 9.

2.5. Choose a behavior ϕ such that

(a) $\phi|_i = \tau|_i$

(b) $\phi \in \mathcal{O}_\mu(g)$

(c) $\phi \in \mathcal{O}_{\neg\mu}(h)$

Proof: Define ϕ by $\phi|_i = \psi|_i$ and, for all $j \geq i$,

if j is odd or $h(\phi|_j)$ is undefined
then $(\mathbf{a}_{j+1}(\phi), \mathbf{s}_{j+1}(\phi)) = g(\phi|_j)$
else $(\mathbf{a}_{j+1}(\phi), \mathbf{s}_{j+1}(\phi)) = h(\phi|_j)$

$\boxed{2.5.1.}$ ϕ is a behavior

Proof: g is total by 2.1.

$\boxed{2.5.2.}$ $\phi|_i = \tau|_i$

Proof: By 2.4 and the definition of ϕ.

2.5.3. ϕ is a μ-outcome of g.

Proof: We must prove that if $\mathbf{a}_{j+1}(\phi) \in \mu$ then $g(\phi|_j) = (\mathbf{a}_{j+1}(\phi), \mathbf{s}_{j+1}(\phi))$. For $j < i$, this holds because $\phi|_i = \tau|_i$ by 2.5.2, and $\tau \in \mathcal{O}_\mu(g)$ by 2.1. For $j \geq i$, it holds by the definition of ϕ and 2.4, which asserts that h is a $\neg\mu$-strategy.

$\boxed{2.5.4.}$ $\phi \in \mathcal{O}_\mu(g)$

Proof: ϕ is an outcome by 2.5.3. It is fair because, by definition, ϕ has infinitely many steps of the form $g(\phi|_j)$, which are μ steps because g is a μ-strategy (by 2.1).

2.5.5. ϕ is a $\neg\mu$-outcome of h.

Proof: We must prove that if $\mathbf{a}_{j+1}(\phi) \in \neg\mu$ then $h(\phi|_j) = (\mathbf{a}_{j+1}(\phi), \mathbf{s}_{j+1}(\phi))$. For $j < i$, this holds because $\phi|_i = \psi|_i$ by definition of ϕ, and $\psi \in \mathcal{O}_{\neg\mu}(h)$ by 2.4. For $j \geq i$, it holds by definition of ϕ and 2.1, which asserts that g is a μ-strategy.

$\boxed{2.5.6.}$ $\phi \in \mathcal{O}_{\neg\mu}(h)$

Proof: ϕ is an outcome by 2.5.5. It is fair because, by definition of ϕ, either $h(\phi|_j)$ is undefined infinitely often or else an infinite number of steps of ϕ are of the form $h(\phi|_j)$, which are $\neg\mu$ steps because 2.4 asserts that h is a $\neg\mu$-strategy.

2.6. $\phi \notin M_S$

Proof: $\widehat{\phi|_i} = \widehat{\tau|_i}$ by 2.5(a), $\widehat{\tau|_i} \notin M_S$ by 2.3, and M_S is a safety property by hypothesis.

2.7. $\phi \in (I \cap E_S \cap E_L \Rightarrow M_S \cap M_L)$

Proof: 2.5(b) asserts that $\phi \in \mathcal{O}_\mu(g)$, so 2.1 implies that ϕ is stuttering-equivalent to an element of $\mathcal{O}_\mu(f)$. Hence $\phi \in (I \cap E_S \cap E_L \Rightarrow M_S \cap M_L)$ because $\mathcal{O}_\mu(f) \subseteq (I \cap E_S \cap E_L \Rightarrow M_S \cap M_L)$ by hypothesis, and properties are closed under stuttering-equivalence.

2.8. $\phi \in I \cap E_S \cap E_L$

Proof: ϕ is in I because $\tau \in I$ by 2.2, $\mathbf{s}_0(\phi) = \mathbf{s}_0(\tau)$ by 2.5(a), and I is a state predicate by hypothesis. It is in $E_S \cap E_L$ because $\phi \in \mathcal{O}_{\neg\mu}(h)$ by 2.5(c), and $\mathcal{O}_{\neg\mu}(h) \subseteq E_S \cap E_L$ by 2.4.

$\boxed{2.9.}$ Contradiction.

Proof: 2.7 and 2.8 imply $\phi \in M_S \cap M_L$, which contradicts 2.6.

End Proof of Theorem 1

Corollary *Let μ be any agent set, let $\mathbf{x}$ be the projection function from $\mathbf{S} \times \mathbf{X}$ to $\mathbf{X}$, let I be an $\mathbf{S}$-predicate, let $(E_S, E_S \cap E_L)$ be a $\neg\mu$-machine-realizable pair of $\mathbf{S}$-properties, and let M_S and M_L be $\mathbf{S} \times \mathbf{X}$-properties such that $\exists \mathbf{x} : M_S$ is a safety property. Then*

$$I \cap E_S \cap E_L \;\Rightarrow\; \exists \mathbf{x} : M_S \cap M_L$$

and

$$I \cap E_S \;\Rightarrow\; \exists \mathbf{x} : M_S \cap (E_L \Rightarrow M_L)$$

are μ-equirealizable.

Proof of Corollary

Substituting $\exists \mathbf{x} : M_S$ for M_S and $\exists \mathbf{x} : M_S \cap M_L$ for M_L in Theorem 1 shows that

$$I \cap E_S \cap E_L \;\Rightarrow\; \exists \mathbf{x} : M_S \cap M_L$$

and

$$I \cap E_S \;\Rightarrow\; ((\exists \mathbf{x} : M_S) \cap \exists \mathbf{x} : (E_L \Rightarrow M_S \cap M_L))$$

are μ-equirealizable. Since E_L does not depend on the $\mathbf{x}$ component, it follows from the definition of existential quantification and simple logical deduction that $(\exists \mathbf{x} : M_S) \cap \exists \mathbf{x} : (E_L \Rightarrow M_S \cap M_L)$ equals $\exists \mathbf{x} : M_S \cap (E_L \Rightarrow M_L)$.

End Proof of Corollary

Proposition 11 *For any disjoint pair of agent sets μ_1 and μ_2, and any properties P_1 and P_2, the property $\mathcal{R}_{\mu_1}(P_1) \cap \mathcal{R}_{\mu_2}(P_2)$ is $\mu_1 \cup \mu_2$-receptive.*

Proof of Proposition 11

By definition of receptiveness, it suffices to assume $\sigma \in \mathcal{O}_{\mu_i}(f_i) \subseteq \mathcal{R}_{\mu_i}(P_i)$ for $i = 1, 2$, where the f_i are μ_i-strategies, and to construct a $\mu_1 \cup \mu_2$-strategy g such that $\sigma \in \mathcal{O}_{\mu_1 \cup \mu_2}(g) \subseteq \mathcal{O}_{\mu_1}(f_1) \cap \mathcal{O}_{\mu_2}(f_2)$. We will define g to be the strategy that begins by trying to generate σ, and when that is no longer possible, performs either an f_1 or an f_2 step, alternating between the two when it can do either.

1. For any finite behavior prefix ρ, define $g(\rho)$ as follows (where $\max \emptyset$ equals $-\infty$).

   if $\rho = \sigma|_j$, for some $j \geq 0$

     then if $\mathbf{a}_{j+1}(\sigma) \in \mu_1 \cup \mu_2$

       then $g(\rho) = (\mathbf{a}_{j+1}(\sigma), \mathbf{s}_{j+1}(\sigma))$

       else $g(\rho) = \perp$

else **if** $(f_2(\rho) = \bot) \vee ((f_1(\rho) \neq \bot) \wedge (0 < n) \wedge (\mathbf{a}_n(\rho) \in \mu_2))$,
where $n = \max\{k \leq |\rho| : \mathbf{a}_k(\rho) \in \mu_1 \cup \mu_2\}$
then $g(\rho) = f_1(\rho)$
else $g(\rho) = f_2(\rho)$

Then g is a $\mu_1 \cup \mu_2$-strategy.

Proof: g is a $\mu_1 \cup \mu_2$-strategy because each f_i is a μ_i-strategy.

$\boxed{2.}$ $\sigma \in \mathcal{O}_{\mu_1 \cup \mu_2}(g)$

Proof: It is immediate from 1 (the definition of g) that σ is a $\mu_1 \cup \mu_2$-outcome of g.

$\boxed{3.}$ $\mathcal{O}_{\mu_1 \cup \mu_2}(g) \subseteq \mathcal{O}_{\mu_1}(f_1) \cap \mathcal{O}_{\mu_2}(f_2)$

Proof: We assume $\tau \in \mathcal{O}_{\mu_1 \cup \mu_2}(g)$ and $i \in \{1, 2\}$, and we prove that $\tau \in \mathcal{O}_{\mu_i}(f_i)$. Since $\sigma \in \mathcal{O}_{\mu_1}(f_1) \cap \mathcal{O}_{\mu_2}(f_2)$ by hypothesis, we may assume that $\tau \neq \sigma$.

3.1. τ is a μ_i-outcome of f_i.

Proof: It suffices to assume that $\mathbf{a}_{j+1}(\tau) \in \mu_i$ and prove that $f_i(\tau|_j) = (\mathbf{a}_{j+1}(\tau), \mathbf{s}_{j+1}(\tau))$. There are two cases.

<u>Case 3.1A</u>. $\tau|_j = \sigma|_j$

In this case, the desired result follows from the hypothesis that $\sigma \in \mathcal{O}_{\mu_i}(f_i)$.

<u>Case 3.1B</u>. $\tau|_j \neq \sigma|_j$

Since τ is a $\mu_1 \cup \mu_2$-outcome of g by hypothesis, and $\mathbf{a}_{j+1}(\tau) \in \mu_i$ implies $\mathbf{a}_{j+1}(\tau) \in \mu_1 \cup \mu_2$, it suffices to prove that if $g(\tau|_j) = (\alpha, s)$ with $\alpha \in \mu_i$, then $f_i(\tau|_j) = g(\tau|_j)$. The desired equality follows from 1, the assumption that $\tau|_j \neq \sigma|_j$, and the hypothesis that μ_1 and μ_2 are disjoint.

$\boxed{3.2.}$ $\tau \in \mathcal{O}_{\mu_i}(f_i)$

Proof: 3.1 asserts that τ is a μ_i-outcome of f_i, so we need only prove that it is a fair outcome. We assume that τ has only finitely many μ_i steps and prove that $f_i(\tau|_j)$ is undefined for infinitely many values of $j \geq 0$.

<u>Case 3.2A</u>. $\mathbf{a}_j(\tau) \in \mu_1 \cup \mu_2$ for only finitely many $j \geq 0$.

In this case, the hypothesis $\tau \in \mathcal{O}_{\mu_1 \cup \mu_2}(g)$ implies that $g(\tau|_j)$ is undefined for infinitely many j. By 1, if ρ is not a prefix of σ, then $g(\rho)$ is undefined iff both $f_1(\rho)$ and $f_2(\rho)$ are undefined. Hence, $g(\tau|_j)$ undefined for infinitely many j and the assumption $\tau \neq \sigma$ imply that $f_i(\tau|_j)$ must be undefined for infinitely many values of j.

<u>Case 3.2B</u>. $\mathbf{a}_j(\tau) \in \mu_1 \cup \mu_2$ for infinitely many $j \geq 0$.

3.2B.1. Choose $l \geq 0$ such that for all $j \geq l$,

(a) If $\mathbf{a}_j(\tau) \in \mu_1 \cup \mu_2$ then (i) $\mathbf{a}_j(\tau) \notin \mu_i$, and (ii) $n > 0$ implies $\mathbf{a}_n(\tau) \notin \mu_i$, where $n = \max\{k \leq j-1 : \mathbf{a}_k(\tau) \in \mu_1 \cup \mu_2\}$.

(b) $\tau|_{j-1} \neq \sigma|_{j-1}$

Proof: We can choose l satisfying (a) by the assumptions that τ has only finitely many μ_i steps and that $\mathbf{a}_j(\tau) \in \mu_1 \cup \mu_2$ for infinitely many $j \geq 0$. Since $\tau \neq \sigma$, we can choose l large enough so that (b) also holds.

3.2B.2. For all $j \geq l$, if $\mathbf{a}_j(\tau) \in \mu_1 \cup \mu_2$, then $f_i(\tau|_{j-1}) = \bot$.

Proof: Assume $j \geq l$ and $\mathbf{a}_j(\tau) \in \mu_1 \cup \mu_2$. Since $\tau \in \mathcal{O}_{\mu_1 \cup \mu_2}(g)$, we have $g(\tau|_{j-1}) = (\mathbf{a}_j(\tau), \mathbf{s}_j(\tau))$. Since $\mathbf{a}_j(\tau) \notin \mu_i$ by 3.2B.1(a), and f_i is a μ_i-strategy, we infer $g(\tau|_{j-1}) \neq f_i(\tau|_{j-1})$. In the definition of $g(\rho)$ in 1, if ρ is not a prefix of σ and $n \leq 0$ implies $\mathbf{a}_n(\rho) \notin \mu_i$, then $g(\rho) \neq f_i(\rho)$ implies $f_i(\rho) = \perp$. Hence, 3.2B.1 implies $f_i(\tau|_{j-1}) = \perp$.

$\boxed{3.2B.3.}$ $f_i(\tau|_j)$ is undefined for infinitely many $j \geq l$.
Proof: By 3.2B.2, $f_i(\tau|_j)$ is undefined for all $j \geq l$ with $\mathbf{a}_j(\tau) \in \mu_1 \cup \mu_2$, and by hypothesis, there are infinitely many such j.

End Proof of Proposition 11

Proposition 12 *If μ_1, μ_2, and $\mu_1 \cup \mu_2$ are agent sets and E, E_1, and M_2 are properties such that:*

1. *$E = I \cap P$ where*
   (a) *I is a state predicate.*
   (b) *P is a safety property that constrains at most $\neg(\mu_1 \cup \mu_2)$.*
2. *E_1 is a safety property.*
3. *$\mu_1 \cap \mu_2 = \emptyset$*
4. *M_2 is a μ_2-abstract property.*

Then the rule of inference

$$\frac{E \cap M_2 \subseteq E_1}{E \cap \overline{M_2} \subseteq E_1}$$

is sound.

Proof of Proposition 12
Proof: We assume $E \cap M_2 \subseteq E_1$ and prove $E \cap \overline{M_2} \subseteq E_1$. We do this by assuming the existence of a behavior σ in $E \cap \overline{M_2}$ but not in E_1, and obtaining a contradiction. We will obtain the contradiction by constructing a behavior in $E \cap M_2$ that is not in E_1. We will first construct a behavior ϕ in M_2 by continuing σ from the point at which it violates the safety property E_1. We will then modify ϕ by replacing agents in $\neg(\mu_1 \cup \mu_2)$ with agents in μ_1 to obtain a behavior τ that will still be in M_2 (because M_2 is μ_2-abstract), in E (because only $\neg(\mu_1 \cup \mu_2)$ steps can violate E), but not in E_1.

1. Choose $i \geq 0$ such that $\widehat{\sigma|_i} \in E \cap \overline{M_2}$ and $\widehat{\sigma|_i} \notin E_1$.
   Proof: Since E_1 is a safety property and $\sigma \notin E_1$, there exists an i such that $\widehat{\sigma|_i} \notin E_1$. Since E and $\overline{M_2}$ are safety properties, $E \cap \overline{M_2}$ is also a safety property. Hence, the assumption $\sigma \in E \cap \overline{M_2}$ implies $\widehat{\sigma|_i} \in E \cap \overline{M_2}$.
2. Choose a behavior ϕ in M_2 such that $\sigma|_i = \phi|_i$.
   Proof: ϕ exists by 1, which asserts $\sigma \in \overline{M_2}$, and the definition of closure.
3. Choose $\beta \in \mu_1$ and let τ be the behavior such that, for all $k \geq 0$:

$\mathbf{s}_k(\tau) = \mathbf{s}_k(\phi)$
if $(k+1 \leq i) \vee (\mathbf{a}_{k+1}(\phi) \in (\mu_1 \cup \mu_2))$ **then** $\mathbf{a}_{k+1}(\tau) = \mathbf{a}_{k+1}(\phi)$
else $\mathbf{a}_{k+1}(\tau) = \beta$

Then τ is μ_2-equivalent to ϕ.

Proof: τ is the same as ϕ except that some agents not in $\mu_1 \cup \mu_2$ have been replaced by β, an agent in μ_1. Since μ_1 and μ_2 are disjoint by hypothesis 3, τ is μ_2-equivalent to ϕ.

4. $\tau|_i = \sigma|_i$
Proof: By 2 and 3, which implies $\phi|_i = \tau|_i$.

5. $\tau \in E \cap M_2$
5.1. $\tau \in I$
Proof: 1 and 4 imply $\widehat{\tau|_i} \in I$, since $E \subseteq I$; and I is a state predicate by hypothesis 1(a).
5.2. $\tau \in P$
5.2.1. $\mathbf{a}_k(\tau) \in (\mu_1 \cup \mu_2)$, for all $k \geq i$.
Proof: By 3.
5.2.2. $\widehat{\tau|_i} \in P$
Proof: By 1 and 4, since $E \subseteq P$.
$\boxed{5.2.3.}$ $\tau \in P$
Proof: By 5.2.1, 5.2.2, and hypothesis 1(b), which asserts that P constrains at most $\neg(\mu_1 \cup \mu_2)$.
5.3. $\tau \in M_2$
Proof: $\phi \in M_2$ by 2, τ is μ_2-equivalent to ϕ by 3, and M_2 is μ_2-abstract by hypothesis 4.
$\boxed{5.4.}$ $\tau \in E \cap M_2$
Proof: By 5.1, 5.2, and 5.3, since E equals $I \cap P$.

6. $\tau \notin E_1$
Proof: 1 and 4 imply $\widehat{\tau|_i} \notin E_1$, and E_1 is a safety property by hypothesis 2.

$\boxed{7.}$ Contradiction.
Proof: 5, 6, and the hypothesis $E \cap M_2 \subseteq E_1$.

End Proof of Proposition 12

Theorem 2 *If μ_1, μ_2, and $\mu_1 \cup \mu_2$ are agent sets and E, E_1, E_2, M_1, and M_2 are properties such that:*

1. *$E = I \cap P$, $E_1 = I_1 \cap P_1$, and $E_2 = I_2 \cap P_2$, where*
   (a) *I, I_1, and I_2 are state predicates.*
   (b) *P, P_1, and P_2 are safety properties that constrain at most $\neg(\mu_1 \cup \mu_2)$, $\neg\mu_1$, and $\neg\mu_2$, respectively.*
2. *$\overline{M_1}$ and $\overline{M_2}$ constrain at most μ_1 and μ_2, respectively.*
3. *$\mu_1 \cap \mu_2 = \emptyset$*

Then the rule of inference

$$\frac{E \cap \overline{M_1} \cap \overline{M_2} \;\subseteq\; E_1 \cap E_2}{\mathcal{R}_{\mu_1}(E_1 \Rightarrow M_1) \cap \mathcal{R}_{\mu_2}(E_2 \Rightarrow M_2) \;\subseteq\; \mathcal{R}_{\mu_1 \cup \mu_2}(E \Rightarrow M_1 \cap M_2)}$$

is sound.

Proof of Theorem 2

Proof: We assume the hypotheses of the theorem, and we prove the soundness of the inference rule by assuming its hypothesis and deducing its conclusion.

1. $\mathcal{R}_{\mu_1}(E_1 \Rightarrow M_1) \cap \mathcal{R}_{\mu_2}(E_2 \Rightarrow M_2) \;\subseteq\; (E \Rightarrow M_1 \cap M_2)$.
   Proof: We assume $\sigma \in \mathcal{R}_{\mu_1}(E_1 \Rightarrow M_1) \cap \mathcal{R}_{\mu_2}(E_2 \Rightarrow M_2)$ and $\sigma \notin (E \Rightarrow M_1 \cap M_2)$, and we obtain a contradiction.
   1.1. $\sigma \notin E_1 \cap E_2$
      1.1.1. Choose $j \in \{1, 2\}$ such that $\sigma \notin M_j$.
         Proof: The assumption $\sigma \notin (E \Rightarrow M_1 \cap M_2)$ implies $\sigma \notin M_1$ or $\sigma \notin M_2$.
      1.1.2. $\sigma \in (E_j \Rightarrow M_j)$
         Proof: By the assumption $\sigma \in \mathcal{R}_{\mu_1}(E_1 \Rightarrow M_1) \cap \mathcal{R}_{\mu_2}(E_2 \Rightarrow M_2)$ and the definition of $\mathcal{R}_{\mu_j}$.
      1.1.3. $\sigma \notin E_j$
         Proof: By 1.1.1 and 1.1.2.
      $\boxed{1.1.4.}$ $\sigma \notin E_1 \cap E_2$
         Proof: By 1.1.3.
   1.2. Let i be the smallest natural number such that $\widehat{\sigma|_i} \notin E_1 \cap E_2$.
      Proof: Such an i exists by 1.1, because hypothesis 1 implies that $E_1 \cap E_2$ is a safety property.
   1.3. $\widehat{\sigma|_i} \notin E \cap \overline{M_1} \cap \overline{M_2}$
      Proof: By 1.2 and the assumption that the hypothesis of the inference rule holds.
   1.4. $\widehat{\sigma|_i} \in E$
      Proof: The assumption $\sigma \notin (E \Rightarrow M_1 \cap M_2)$ implies $\sigma \in E$, and E is a safety property.
   1.5. $\widehat{\sigma|_i} \notin \overline{M_1} \cap \overline{M_2}$
      Proof: By 1.3 and 1.4.
   1.6. $i > 0$ and $\widehat{\sigma|_{i-1}} \notin E_1 \cap E_2$.
      Proof: By 1.5, there exists $j \in \{1, 2\}$ such that $\widehat{\sigma|_i} \notin \overline{M_j}$. Hypotheses 1 and 2 of the theorem, and the assumption $\sigma \in \mathcal{R}_{\mu_j}(E_j \Rightarrow M_j)$ then allow us to apply Lemma 4, substituting μ_j, I_j, P_j, and M_j for μ, I, P, and M, to conclude $i > 0$ and $\widehat{\sigma|_{i-1}} \notin E_j$.
   1.7. Contradiction.
      Proof: By 1.6 and the choice of i in 1.2.

$\boxed{2.}$ $\mathcal{R}_{\mu_1}(E_1 \Rightarrow M_1) \cap \mathcal{R}_{\mu_2}(E_2 \Rightarrow M_2) \;\subseteq\; \mathcal{R}_{\mu_1 \cup \mu_2}(E \Rightarrow M_1 \cap M_2)$

Proof: By 1, Proposition 3, and Proposition 11, which we can apply by hypothesis 3.

End Proof of Theorem 2

Acknowledgements

Greg Nelson and Cynthia Hibbard provided useful comments on the original version of this article. Eugene Stark sent us his thesis proposal and pointed out the correspondence between our definitions and the ones in his thesis.

Glossary

$\mathbf{a}_i(\sigma)$: The i^{th} agent of behavior σ.

f, g, h: Strategies, except in Section 5.2.2, where f is a refinement mapping.

inp, mid, out: State components from the example in Figure 3.

$\mathbf{s}_i(\sigma)$: The i^{th} state of behavior σ.

s, t: States.

$\mathbf{x}$, $\mathbf{y}$: Internal state components.

$\mathbf{A}$: The set of all agents.

E: An environment assumption (a property).

E_S, E_L: Safety and liveness parts of E (in Section 4.3).

I: A state predicate.

$I_{\mathbf{x}}$: An initial condition for an internal state component $\mathbf{x}$.

$\mathcal{I}$: The identity next-state relation.

L: A progress property.

M: A system guarantee (a property).

M_S, M_L: Safety and liveness parts of M (in Section 4.3).

$\mathcal{N}$: A next-state relation.

$\mathcal{N}_E$, $\mathcal{N}_M$: Next-state relations of an environment and a system.

$\mathcal{O}_\mu(f)$: The set of behaviors generated by μ-strategy f.

P, Q: Sets of behaviors—usually properties.

$\mathcal{R}_\mu(P)$: The μ-realizable part of P.

S: A specification.

$\mathbf{S}$: The set of all states.

$\mathcal{TA}(\mathcal{N})$**:** The property defined by the next-state relation $\mathcal{N}$ of a complete program.

$\mathcal{TA}_\mu(\mathcal{N})$**:** The property asserting that every μ-step satisfies the next-state relation $\mathcal{N}$.

$\mathcal{U}_{\mathbf{x}}$**:** The next-state relation asserting that state component $\mathbf{x}$ is unchanged.

$V(P, \sigma)$**:** The step number of the first step at which behavior σ violates property P.

$\mathbf{X}$**,** $\mathbf{Y}$**:** Sets of internal states.

α**,** β**:** Agents.

β_μ**:** An agent in μ.

μ**:** A set of agents, usually an agent set.

η**,** θ**,** ρ**:** Behavior prefixes, usually finite.

ξ**,** σ**,** τ**,** ϕ**:** Behavior prefixes, usually infinite.

ψ**:** A behavior prefix (finite or infinite).

Λ**,** Ξ**:** Mappings on behavior prefixes.

$\boldsymbol{\Pi}$**:** A system (not a formally defined concept).

$\Pi_{\mathbf{S}}$**:** The projection mapping onto the external states.

$\Pi_{\mathbf{X}}$ The projection mapping onto the internal states.

$\exists\mathbf{x}$**:** Existential quantification over a state component $\mathbf{x}$.

$s \xrightarrow{\alpha} t$**:** A step performed by agent α.

$\overline{P}$**:** The closure of P (the smallest safety property containing P).

$P \Rightarrow Q$**:** The property consisting of all behaviors that are in Q or not in P.

$P \rightarrow\!\!\triangleright\, Q$**:** The property asserting that Q holds as long as P does.

$S|^x_y$**:** The result of substituting x for y in the formula for S.

$\rho \cdot (\alpha, s)$**:** The finite behavior prefix obtained by concatenating $\xrightarrow{\alpha} s$ to the end of ρ.

$\rho_{\mathbf{a}}$**:** The last agent of ρ.

$\rho_{\mathbf{s}}$**:** The last state of ρ.

$|\rho|$**:** The length of ρ.

$\widehat{\rho}$**:** The behavior obtained by extending the finite behavior prefix ρ with stuttering steps.

$\sigma|_n$**:** The finite behavior prefix consisting of the first n steps of σ.

$\natural_\mu \sigma$**:** The behavior prefix obtained by removing μ-stuttering steps from σ.

$\simeq$**:** Stuttering-equivalence.

$\simeq_\mu$**:** μ-stuttering-equivalence.

$\{P\}\boldsymbol{\Pi}\{Q\}$**:** A Hoare triple.

$f(\rho) = \perp$**:** Asserts that ρ is not in the domain of f.

References

[AFK88] Krzysztof R. Apt, Nissim Francez, and Shmuel Katz. Appraising fairness in languages for distributed programming. *Distributed Computing*, 2:226–241, 1988.

[AL91] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.

[ALW89] Martín Abadi, Leslie Lamport, and Pierre Wolper. Realizable and unrealizable specifications of reactive systems. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 1–17. Springer-Verlag, July 1989.

[AS85] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.

[BDDW91] Manfred Broy, Frank Dederichs, Claus Dendorfer, and Rainer Weber. Characterizing the behaviour of reactive systems by trace sets. In Eike Best and Grzegorz Rozenberg, editors, *3rd Workshop on Concurrency and Compositionality*, volume 191 of *GMD-Studien*, pages 47–56, Saint Augustin, Germany, 1991. GMD. Extended abstract.

[BKP86] Howard Barringer, Ruurd Kuiper, and Amir Pnueli. A really abstract concurrent model and its temporal logic. In *Thirteenth Annual ACM Symposium on Principles of Programming Languages*, pages 173–183. ACM, January 1986.

[Dav64] Morton Davis. Infinite games of perfect information. In M. Dresher, L. S. Shapley, and A. W. Tucker, editors, *Advances in game theory*, volume 52 of *Annals of Mathematics Studies*, pages 85–101. Princeton University Press, Princeton, New Jersey, 1964.

[dBdRR90] J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors. *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, volume 430 of *Lecture Notes in Computer Science*, Berlin, 1990. Springer-Verlag.

[Dil88] David L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* PhD thesis, Carnegie Mellon University, February 1988.

[Hoa72] C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Series in Computer Science. Prentice-Hall International, London, 1985.

[HP85] David Harel and Amir Pnueli. On the development of reactive systems. In K. R. Apt, editor, *Logics and models of concurrent systems*, volume F13 of *NATO ASI Series*, pages 477–498. Springer-Verlag, 1985.

[Lam83a] Leslie Lamport. Specifying concurrent program modules. *ACM Transactions on Programming Languages and Systems*, 5(2):190–222, April 1983.

[Lam83b] Leslie Lamport. What good is temporal logic? In R. E. A. Mason, editor, *Information Processing 83: Proceedings of the IFIP 9th World Congress*, pages 657–668, Paris, September 1983. IFIP, North Holland.

[Lam84] Leslie Lamport. Solved problems, unsolved problems and non-problems in concurrency. In Jayadev Misra, editor, *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, pages 1–11, New York, August 1984. ACM. Invited address presented at 1983 Symposium.

[Lam89] Leslie Lamport. A simple approach to specifying concurrent systems. *Communications of the ACM*, 32(1):32–45, January 1989.

[Lam90] Leslie Lamport. A temporal logic of actions. research report 57, Digital Equipment Corporation, Systems Research Center, April 1990. A revised version to appear.

[LS84a] Simon S. Lam and A. Udaya Shankar. Protocol verification via projections. *IEEE Transactions on Software Engineering*, SE-10(4):325–342, July 1984.

[LS84b] Leslie Lamport and Fred B. Schneider. The "Hoare logic" of CSP, and all that. *ACM Transactions on Programming Languages and Systems*, 6(2):281–296, April 1984.

[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.

[MC81] Jayadev Misra and K. Mani Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, SE-7(4):417–426, July 1981.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, New York, 1980.

[MP87] Zohar Manna and Amir Pnueli. A hierarchy of temporal properties. Technical Report STAN-CS-87-1186, Department of Computer Science, Stanford University, October 1987.

[OG76] Susan Owicki and David Gries. Verifying properties of parallel programs: An axiomatic approach. *Communications of the ACM*, 19(5):279–284, May 1976.

[OL82] Susan Owicki and Leslie Lamport. Proving liveness properties of concurrent programs. *ACM Transactions on Programming Languages and Systems*, 4(3):455–495, July 1982.

[Pnu84] Amir Pnueli. In transition from global to modular temporal reasoning about programs. In Krzysztof R. Apt, editor, *Logics and Models of Concurrent Systems*, NATO ASI Series, pages 123–144. Springer-Verlag, October 1984.

[Sta84] Eugene W. Stark. *Foundations of a theory of Specification for Distributed Systems*. PhD thesis, M. I. T., August 1984.

[Sta85] Eugene W. Stark. A proof technique for rely/guarantee properties. In S. N. Maheshwari, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 206 of *Lecture Notes in Computer Science*, pages 369–391, Berlin, 1985. Springer-Verlag.

Index

$\overline{P}$, 13
$S|^{x}_{y}$, 32, 33
$\simeq$, 12
$\simeq_{\mu}$, 12
$\Pi_{\mathbf{S}}$, 26
$\Pi_{\mathbf{X}}$, 26
$\perp$, 47
$\cdot$ (concatenation), 48
$\Rightarrow$, 8, 12
$\rightarrowtriangle$, 13
$|\rho|$, 48
$\natural_{\mu}\sigma$, 12
$\sigma|_{m}$, 11
$\rho_{\mathbf{a}}$, 48
$\rho_{\mathbf{s}}$, 48
$\widehat{\sigma}$, 12
μ-abstract, 13
μ-equirealizable, 16
μ-equivalent, 13
μ-machine-realizable, 25
μ-outcome, 16
  partial, 48
μ-realizable, 16
μ-realizable part, 16
μ-receptive, 16
μ-strategy, 16
μ-stuttering step, 12
μ-stuttering-equivalence, 12

$\mathbf{A}$, 10
$\mathbf{a}_{i}(\sigma)$, 11
abstraction functions, 38
actions
  atomic, 11
  joint, 7
  system versus environment, 7, 14
  temporal logic of, 47
  versus agents, 11
  versus states, 6
Ada, 6
agent set, 10
  disjointness of, 42
  of a partial program, 24
agents, 8, 10, 33
  system versus environment, 8
Alpern, Bowen, 13
Apt, Krzysztof R., 22
assumption, environment, 1
auxiliary variables, 38

Barringer, Howard, 11
behavior, 8, 11
behavior prefix, 11
Broy, Manfred, 18

case, 48
CCS, 6
Chandy, K. Mani, 5
circular reasoning, 2, 9, 39, 91
closed set, 13
closure, 13
Composition Principle
  for safety properties, 5
  for sequential programs, 2
  informal statement, 2
  Pnueli's, 5
  Stark's, 5
  theorem, 41
concatenation, 48
conjunction, composition as, 9, 33, 47

constrains at most, 15
CSP, 5, 7, 23, 47

Davis, Morton, 15
dense set, 13
Dill, David L., 16

end according to, 48
equirealizable, 15, 16
existential quantification, 26
    and explicitness, 46
explicitness, 46

failure-set semantics, 47
fair realizability, 17
fairness, 22
    strong, 25
feasibility, 22
Francez, Nissim, 22

game, realization, 15
guarantee, system, 1

hiding, 26
history variables, 38
Hoare logic, 2
Hoare triple, 2

$\mathcal{I}$, 37
I/O automata, 1
identity relation, 37
implements, 34
    transitivity of, 34
initial predicate
    of a complete program, 20
    of a partial program, 24
initial state, 13
    and refinement mapping, 36
    chosen by environment, 15
inp, 31

interleaving, 11, 47
invariant, 37
invariant under stuttering, 17

Katz, Shmuel, 22
Kuiper, Ruurd, 11

Lam, Simon S., 1
length of a behavior prefix, 48
liveness property, 5, 13
Lynch, Nancy, 1

machine-closed, 22
machine-realizable, 25
Manna, Zohar, 22
mid, 31
Milner, Robin, 6
Misra, Jayadev, 5
monotone, 48
monotonicity of $\mathcal{R}_\mu$, 18
moves of realization game, 15

$\mathcal{N}_E$, 36
$\mathcal{N}_M$, 36
next-state relation
    and refinement mapping, 37
    of a complete program, 20
    of a partial program, 24
nondeterminism, external and internal, 7
normal form, 27, 30

$\mathcal{O}_\mu(f)$, 16
out, 31
outcome, 15, 16
    fair, 16
Owicki-Gries method, 37

parallel composition, 32
partial correctness, 2

Pascal, 1, 23
Pnueli, Amir, 5, 11, 22, 35
postconditions, 3
preamble, 48
preconditions, 3
program
  abstract, 1, 27, 47
  complete, 20
  partial, 23
  semantics of, 47
  sequential, 2
progress property, 20, 21, 24
projection
  functions, 26
  method of, 1
proof style, explanation of, 48
property, 8, 12
  invariance, 37
  liveness, 5, 13
  safety, 5, 13
prophecy variables, 38
protocol, 3, 43

$\mathcal{R}_\mu$, 16
reactive systems, 3
realizable, 15, 16
realizable part, 15, 16
realization game, 15
reasoning, circular, *see* circular reasoning
receptive, 16
refinement mapping, 36
renaming, 32, 42

$\mathbf{S}$, 10
$\mathbf{s}_i(\sigma)$, 11
$\mathbf{S}$-predicate, 10
safety property, 5, 13
Schneider, Fred B., 13
sequential composition, 2, 32
sequential program, 2
Shankar, A. Udaya, 1
Stark, Eugene W., 5, 16, 18
state component
  and refinement mapping, 36
  externally observable, 26
  internal, 12, 26, 29
state predicate, 10
  as property, 12
  is safety property, 13
states, 6, 10, *see* initial state, 32
  internal, 12, 18, 26, 28, 36
  of a complete program, 20
  of a partial program, 23
  reachable, 37
  universal set of, 23, 33
  versus actions, 6
step, 11
  stuttering, *see* stuttering step
strategy, 16
stuttering step, 12
  as internal action, 17
stuttering-equivalence, 12
  and logic, 46
  and strategies, 17

$\mathcal{TA}(\mathcal{N})$, 20
$\mathcal{TA}_\mu(\mathcal{N})$, 24
topology, 13
transition-axiom method, 1, 10, 27
Tuttle, Mark, 1

$\mathcal{U}_{\mathbf{X}}$, 26

$V(P, \sigma)$, 15

$\mathbf{X}$, 26
$\mathbf{x}$, 26

$\mathbf{Y}$, 36