COMPUTING QUOTIENTS OF FINITE STATE SYSTEMS[1]

Kenneth L. Calvert

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188
calvert@cs.utexas.edu

TR-91-23 July 1991

[1]Work supported (through Simon S. Lam) by the National Science Foundation under grant number NCR-9004464 and by a grant from the Texas Advanced Research Program.

Abstract

We present a theory of interacting finite state machines, which includes a binary composition operator ‘$\circ$’ and a satisfaction relation “*sat*.” The composition operation models the formation of a composite that results when two systems interact via synchronized actions (the so-called “rendezvous” model of interaction). The satisfaction relation captures the notion that one machine is an adequate replacement for another in any composite system. Composition of finite state machines is monotonic with respect to the satisfaction relation; this enables hierarchical modeling and component-wise refinement. The operational interpretation of the theory is similar to that of CSP.

Within this theory, we present a solution to problems of the following form: for given finite state machines A and B, find a machine X such that $B \circ X$ *sat* A, or show that no such machine exists. Such problems are called *quotient* problems because they involve “inversion” of the composition operator. Our solution improves on previous results by dealing fully with deadlock and divergence. The problem of finding a protocol converter for given protocols motivates the study of quotient problems; the method has been successfully applied to compute finite state machines representing protocol converters for realistic protocols.

1 Introduction

Consider a theory of state machines that features a composition operator "$\circ$" by which two state machines are combined to form a composite machine, and a relation *sat* on state machines, whose interpretation is that one machine is an adequate implementation of another. We are interested in the following problem: for given machines A and B, representing a system to be implemented and an existing component, respectively, we are required to specify a machine X, which can be combined with B to produce an implementation of A. That is, we want to solve for X in the following:

$$B \circ X \ \mathit{sat}\ A$$

We call this kind of problem a *quotient problem* (the name comes from viewing composition as a multiplicative operation that we are required to "invert"). Note that there may be any number of X's satisfying the above relation, including zero.

Problems of this type arise frequently in the design of distributed systems and have been the focus of some research [9, 14, 15, 11, 12]. In this paper we present a method of computing a solution when A and B are given as finite state machines. We define $\circ$ and *sat* for finite state machines to form a compositional theory, within which certain kinds of concurrent interacting systems (e.g., protocol implementations) can be modeled. The solution method presented here is motivated by the observation that the formal definition of a *protocol converter*—a device enabling implementations of different communication protocols to interact usefully without modification—has the form of the quotient equation above. Thus, our method constitutes a solution to the general problem of finding a protocol converter whenever the protocol implementations can be represented as finite state machines [3].

The rest of this paper is organized as follows. In the next section, we present a theory of finite state machines and prove some properties of FSMs. The "quotient algorithm" is is presented in two parts, in sections 3 and 4. Section 5 contains a discussion of related work and conclusions.

2 A Theory of Finite State Machines

We are interested in a general solution to the quotient problem because we would like to be able to solve such problems in the top-down development of concurrent or distributed systems. Therefore the formal framework within which we develop the solution should be powerful enough to serve as a vehicle for such development. That is, it should permit us to represent an abstract specification as a state machine (or a collection of logic formulas, or some other mathematical abstraction), and then establish formally that an implementation (represented as another state machine, a program, or another collection of formulas) satisfies that specification. Moreover, the theory should permit us to specify all—and only—the aspects of system behavior that are in some sense relevant.

The fundamental model of concurrency and interaction used here is the same as that introduced in formalisms such as CSP [2, 8] and CCS [13]. The relevant part of a system's behavior—that is, its interface with the environment—is characterized by a set of *actions*, through which it can synchronize (or "rendezvous") with its environment. These actions are postulated to be atomic and nonoverlapping. Occurrence of an action requires that it be *enabled* in both the system and the environment. System behavior is characterized by the sequences in which actions may and may not happen, and concurrency of two actions is represented by the possibility of their occurrence in either order. If, at some point, no action is enabled in both the system and its environment, then deadlock occurs, and no further interaction is possible. When multiple actions are enabled on both sides of the interface, one is "chosen" (by some unspecified means) to occur. This choice need not be fair; in particular, it may always prefer one action over another when both are enabled.

The following notational conventions will be observed. Function application is denoted by an infix dot (period); it has the highest syntactic binding power in expressions. An expression of the form $(\mathbf{op}\, x : R.x : T.x)$, where x is a list of variables bound in the expression, denotes the result of applying $\mathbf{op}$ to the bag of all $T.x$ such that $R.x = \mathbf{true}$. In this paper, "**op**" will be one of "$\forall$" or "$\exists$" or "**SET** ," and the expression yields either a boolean or a set. An omitted range is understood to be the constant **true**. These expressions obey the following laws:

- $(\forall\, x : R.x : T.x) \equiv (\forall\, x :: R.x \Rightarrow T.x)$
- $(\exists\, x : R.x : T.x) \equiv (\exists\, x :: R.x \wedge T.x)$
- $y \in (\mathbf{SET}\ \ x : R.x : T.x) \equiv (\exists\, x : R.x : y = T.x)$

The syntactic binding power of the boolean operators is, in order from most to least binding with those in parentheses being equal: $\neg$, $(\wedge, \vee)$, $(\Rightarrow, \Leftarrow)$, $\equiv$. Set operators have the relative precedence $(\cap, \cup, \div)$, $(\in, =)$, and bind more tightly than boolean operations.

2.1 Definitions

A finite state machine (FSM) is a 4-tuple (Σ, S, i, δ), where

- Σ is a finite set of named, atomic *actions*
- S is a finite set of *states*
- $i \in S$ is the distinguished *initial state*, and
- $\delta \subseteq S \times (\Sigma \cup \{\diamond\}) \times S$ is the *transition relation*.

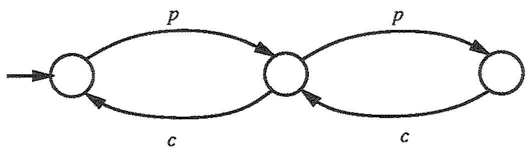

Figure 1: Producer-Consumer Synchronizer M

The transition relation δ defines how the state of the machine changes according to its current state and its interaction with the environment. For states m and m', and action e, if $(m, e, m') \in \delta$ we say that the action e *is enabled at* m; if e is also enabled in the environment, it can occur, and the state instantaneously becomes m'. The *null action*, represented by $\diamond$, is distinct from all actions in Σ; a transition $(m, \diamond, m') \in \delta$ is called an *internal* transition. Internal transitions may occur without participation by the environment, and represent interactions between the sub-components of a composite FSM. The presence of internal actions introduces *nondeterminism*, which means that the state of the machine after a given sequence of interface actions may not be uniquely determined. The choice of which transition occurs when several are enabled is completely nondeterministic; we assume only that if some transition is enabled, then *some* transition will occur.

The state sets of distinct FSMs are postulated to be distinct. We assume that actions are shared by at most two FSMs; this corresponds to an assumption that interfaces are two-sided, as opposed to many-sided.

An example of a simple finite state machine is given in Figure 1. (In figures the usual graphical representation of FSMs is used, with nodes representing states and labeled arcs representing transitions.) This machine, M, models a simple synchronizer. One user, the *Producer*, produces items that are used by the other user, the *Consumer*. When the Producer is ready to turn over an item, it enables the event p. When the Consumer is ready to accept an item, it enables the event c. The synchronizer M ensures that the Producer never gets more than two items ahead of the Consumer.

A finite sequence of actions is called a *trace*. In what follows, the variables t, u, and v range over traces. An individual action is treated as a trace of length one, and concatenation of traces is denoted by juxtaposition. Thus te is the (possibly empty) sequence t followed by the single action e. The null action $\diamond$ is a right and left zero of concatenation ($t\diamond = \diamond t = t$), and thus is identified with the empty sequence, or *null trace*. The notation t^k denotes a catenated sequence of k copies of t.

The transition relation δ is extended to the *trace relation* δ^* in the usual way by taking its reflexive and transitive closure. Formally, δ^* is defined to be the least relation (i.e. subset of $S \times \Sigma^* \times S$) satisfying

- $(m, \diamond, m) \in \delta^*$
- $(m_0, t, m_1) \in \delta^* \land (m_1, e, m_2) \in \delta \Rightarrow (m_0, te, m_2) \in \delta^*$

We abbreviate $(m, \diamond, m') \in \delta$ to $\delta(m, \diamond, m')$, and similarly $(m, t, m') \in \delta^*$ is rendered as $\delta^*(m, t, m')$.

In what follows, variables A, B, C, D, M, and N range over arbitrary FSMs. Variables ranging over arbitrary states of a particular finite state machine are represented by the lower-case variable denoting the FSM: m denotes a state of M, n is a state of N, etc. Primes and subscripts are used when naming multiple states of the same FSM, so that b', b_0, and b_1 all represent states of B. Note that the initial state is always denoted by i. Where ambiguity is possible regarding which FSM is referred to, we resort to subscripts.

A finite state machine can be viewed as a directed graph with labeled edges, and some graph terminology is useful in discussing the structure of FSMs. In particular, there is an obvious correspondence between the trace relation and the set of (labeled) *paths* in the graph. A *cycle* is a nonempty path that begins and ends at the same state.

An FSM defines a set of traces, which is just the language "accepted" by the FSM in the traditional automata-theoretic sense (with each state being an accepting state) [10]. For trace t and FSM M, we say "t is a trace of M" iff t is the sequence of actions along some path from the initial state of M. Formally this is represented by the expression $M(t)$, defined

$$M(t) \stackrel{\text{def}}{=} (\exists\, m :: \delta^*(i, t, m))$$

The traces of M represent *possible* behaviors of M. Clearly, the set of traces of an FSM is prefix-closed: $M(tt') \Rightarrow M(t)$. Also, $\diamond$ is a trace of every FSM.

Before defining composition of FSMs, we introduce some additional terms and predicates on states. For each state m, *out.m* is the subset of Σ containing the interface actions enabled at state m. Formally,

$$out.m \stackrel{\text{def}}{=} (\textbf{SET}\ m', e : e \in \Sigma \land \delta(m, e, m') : e)$$

For any state m, *ext.m* is a predicate that is true iff no internal transition originates at m. Formally,

$$ext.m \stackrel{\text{def}}{=} \neg(\exists\, m' : m \in S : \delta(m, \diamond, m'))$$

Where *ext.m* holds, the state of the machine can change only through interaction with the environment.

A state m is *divergent* if some cycle of internal transitions contains m. Operationally, because the environment cannot prevent internal transitions from occurring, a machine reaching a divergent state is not guaranteed ever to interact with its environment again, even if interface actions are continually enabled. Instead, it may "cycle" through internal transitions forever. The predicate *div* is true of divergent states:

$$div.m \overset{\text{def}}{=} (\exists\, m' : \delta(m, \diamond, m') : \delta^*(m', \diamond, m))$$

We say an FSM is divergent iff it has a reachable divergent state; i.e., there exist t and m such that $\delta^*(i, t, m)$ and $div.m$.

The following property is a consequence of the finiteness of state sets and the definitions of *ext* and *div*:

$$(0) \quad (\exists\, m :: \delta^*(i, t, m)) \Rightarrow (\exists\, m : \delta^*(i, t, m) : (ext.m \vee div.m))$$

2.2 Composition

The result of composing two FSMs is another FSM having a set of interface actions that is a subset of the union of those of the components. The state set of the composite machine is isomorphic to the cartesian product of the component state sets. We use the notation $\langle m\, n \rangle$ to denote the state of the composite machine corresponding to the pair (m, n) of states of M and N.

For FSMs M and N, we define $M \circ N$ to be (Σ, S, i, δ), where

- $\Sigma = \Sigma_M \div \Sigma_N$
- $S = S_M \times S_N$
- $i = \langle i_M\, i_N \rangle$
- δ is given by

$$\begin{aligned}\delta(\langle m\, n\rangle, e, \langle m'\, n'\rangle) \equiv\ & (e \in \Sigma \wedge \delta_M(m, e, m') \wedge n = n') \vee \\ & (e \in \Sigma \wedge \delta_N(n, e, n') \wedge m = m') \vee \\ & (e = \diamond \wedge \\ & \quad (\exists\, g : g \neq \diamond : \delta_M(m, g, m') \wedge \delta_N(n, g, n')))\end{aligned}$$

In general, a composite machine $M \circ N$ may have states that are not reachable from the initial state via any sequence of transitions, even though the corresponding states in M and N are reachable; these unreachable states play no part in our theory. Clearly composition is commutative up to isomorphism; because each action is shared by at most two FSMs, it is associative as well.

Observe that the synchronized interaction of the components—if they have any actions in common—become *internal* state transitions of the composite machine. This is illustrated in the figures below. Figure 2 shows a synchronizer

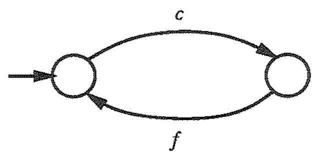

Figure 2: Synchronizer N

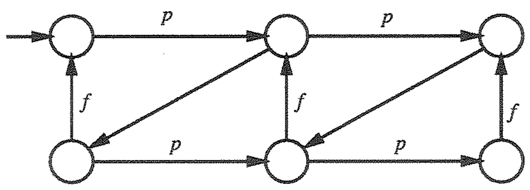

Figure 3: Composite FSM $M \circ N$

N, similar to M from previous example; however, N maintains the Producer and and Consumer tightly in lockstep, and its "Producer action" is c, while its "Consumer action" is f. Figure 3 depicts $M \circ N$, the machine formed by letting M and N interact, with N functioning as Consumer for M, and M acting as producer for N. The unlabeled arrows are the $\diamond$-transitions that result from synchronization and hiding of the common c actions. The "capacity" of this composite synchronizer is the sum of capacities of the components: at most three items can be produced without an item being consumed. The following properties, which are straightforward consequences of the foregoing definitions, express relationships between a composite $M \circ N$ and its components M and N.

(1) $\quad out.\langle m\, n\rangle = out.m \div out.n$

(2) $\quad ext.\langle m\, n\rangle \equiv ext.m \;\wedge\; ext.n \;\wedge\; (out.m \cap out.n = \emptyset)$

(3) $\quad div.\langle m\,n\rangle \equiv div.m \vee div.n \vee (\exists\, u : u \neq \diamond : \delta^*_M(m,u,m) \wedge \delta^*_N(n,u,n))$

To relate the traces of a composite to those of its components, we define the relation *zip*. Let $\Sigma = \Sigma_M \div \Sigma_N$. We define *zip* as the smallest relation (subset of $\Sigma^*_M \times \Sigma^*_N \times \Sigma^*$) such that, for all traces $u, u' \in \Sigma^*_M$, $v, v' \in \Sigma^*_N$, and $w, w' \in \Sigma^*$,

- $zip(\diamond, v, v)$
- $zip(u, \diamond, u)$
- $zip(u, u, \diamond)$
- $zip(u,v,w) \wedge zip(u',v',w') \Rightarrow zip(uu',vv',ww')$

As an example, for M and N in the figures above, we have $zip(pcp, cf, ppf)$ and $zip(pcp, cf, pfp)$, etc. The following properties follow from the definitions.

(4) $\quad (M \circ N)(w) \equiv (\exists\, u, v : zip(u,v,w) : M(u) \wedge N(v))$

(5) $\quad \delta^*(\langle m\,n\rangle, w, \langle m'\,n'\rangle) \equiv$
$\qquad (\exists\, u, v : zip(u,v,w) : \delta^*_M(m,u,m') \wedge \delta^*_N(n,v,n'))$

2.3 Satisfaction Relation

The intuitive meaning of M *sat* N is that (the system represented by) M is "as good as" N, and therefore can replace it in any composite system. The idea is that the environment should not be able to observe any difference between M and N. The underlying model of synchronous interaction more or less dictates that M *sat* N implies the following conditions on M and N:

- If M can engage in a finite sequence of actions (trace), then N can engage in the same sequence of actions. (Otherwise, replacing N with M in a composite might result in some bad state—e.g., a deadlock state—becoming reachable that was not reachable before.)
- If, after a certain trace, N is guaranteed to have some action in a given set enabled, then M, after the same trace, is guaranteed to have some action in the same set enabled. (This condition is also to ensure that if $N \circ D$ cannot reach a deadlock state, then $M \circ D$ cannot either.)
- If M can diverge after a certain trace, then N can diverge after the same trace.[1] (This condition, together with the first, ensures that a composite $M \circ D$ can diverge only if $N \circ D$ can.)

[1]The possibility of divergence disappears if we make a stronger assumption about the "fairness" of selection of state transitions for occurrence. In particular, if we assume that every infinitely-often-enabled transition occurs infinitely often, the possibility that all interface actions are forever pre-empted by internal transitions is ruled out. In this case, all states on a cycle of internal transitions can be aggregated into a single state, from which the system will eventually depart by any means possible. Such a strong fairness assumption places a burden

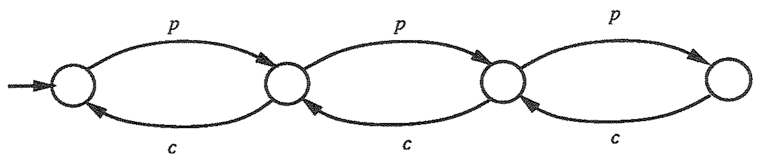

Figure 4: Producer-Consumer synchronizer N

These conditions are formalized as follows. For FSMs M and N, M *sat* N iff $\Sigma_M = \Sigma_N$ and both of the following conditions are satisfied for each trace t and state m of M:

$$\textbf{S0} \quad \delta^*_M(i,t,m) \wedge ext.m \Rightarrow (\exists\, n : \delta^*_N(i,t,n) : (ext.n \wedge out.n \subseteq out.m) \vee div.n)$$

$$\textbf{S1} \quad \delta^*_M(i,t,m) \wedge div.m \Rightarrow (\exists\, n : \delta^*_N(i,t,n) : div.n)$$

Figure 4 shows a finite state machine representing a producer-consumer synchronizer of capacity three; call this machine D. It can be shown that $M \circ N$ *sat* D, and in fact D *sat* $M \circ N$.[2] The operational interpretation of the relation *sat* is similar to that of the "more-deterministic" relation ($\sqsubseteq$) in the well-known failure semantics for CSP [8]. The main difference is the following. In CSP, a process that can initially diverge is understood to be capable of enabling (or not enabling) any of its interface actions, and so satisfies only the trivial specification "true." In the theory presented here, an initially-diverging process is not guaranteed to do anything, but may be relied on *not* to do some things; to the environment, divergence is indistinguishable from deadlock.

An immediate consequence of **S0** and **S1** is the following:

$$(6) \quad M \; sat \; N \Rightarrow (\forall t : M(t) : N(t))$$

The following property, proved in the appendix, states that composition is "monotonic" with respect to the satisfaction relation. This means that the individual components of a system can themselves be regarded as specifications, and can be implemented independently of each other.

on the implementor that seems to go beyond that embodied in the FSM itself: fairness is fundamentally an infinitary property, and does not fit nicely into a theory of finite state machines. A satisfaction relation based on a partial assumption of strong fairness is given in [4], but it does not have certain desirable properties.

[2] A complete development of this theory would include proof rules or sufficient conditions permitting one to establish M *sat* N for arbitrary M and N. We omit such a discussion here in favor of a thorough description of the solution method for quotient problems, which guarantees correctness. In general, it is computationally hard to decide whether M *sat* N.

(7) $M \; sat \; N \;\Rightarrow\; (M \circ D) \; sat \; (N \circ D)$

2.4 Deterministic and Semi-Deterministic FSMs

Different FSMs may be equivalent with respect to the *sat* relation; we have already noted D and $M \circ N$ in Figures 3 and 4 as examples. In this section we describe two classes of FSMs having structural characteristics that make reasoning about the satisfaction relation easier. The first is the class of *deterministic* FSMs. A FSM M is deterministic if at most one state is reachable via any trace; that is,

$$\delta^*(i, t, m) \wedge \delta^*(i, t, m') \Rightarrow m = m'$$

A necessary and sufficient condition for M to be deterministic is that it have no reachable internal transitions, and for any m and e, there is at most one m' such that $\delta(m, e, m')$. The environment of a deterministic FSM can have perfect knowledge of its state at all times, by observing the sequence of actions that occur. For any finite state machine M, a standard procedure [10] produces a deterministic FSM N such that $M(t) \equiv N(t)$ for any trace t, and in fact $N \; sat \; M$.

We next introduce the class of *semi-deterministic* FSMs, which properly contains the deterministic finite state machines. In a semi-deterministic FSM, all states reachable via a given trace are "localized" to the set of states reachable from a single state (via at most one internal transition). Thus, a semi-deterministic FSM may be considered deterministic with respect to a subset of its states. To define semi-determinism formally, we first define two relations derived from the transition relation. The first holds between two states if one is reachable from the other by at most a single internal transition:

$$\mu(m, m') \overset{\text{def}}{=} m = m' \vee \delta(m, \diamond, m')$$

The second is defined in terms of the first and the transition relation δ; it holds when there is a path from one state to another consisting of a single action, preceded by at most one internal transition. Formally, for any $e \in \Sigma$, m, and m':

$$\xi(m, e, m') \overset{\text{def}}{=} e \neq \diamond \wedge (\exists\, m_0 : \mu(m, m_0) : \delta(m_0, e, m'))$$

As usual we define ξ^*, the reflexive and transitive closure of ξ, to be the strongest relation satisfying:

- $\xi^*(m, \diamond, m)$
- $\xi^*(m_0, t, m_1) \wedge \xi(m_1, e, m_2) \Rightarrow \xi^*(m_0, te, m_2)$

A finite state machine is *semi-deterministic* iff it satisfies the following requirements:

- For each state m, $out.m = \emptyset \vee ext.m$. (No state has both internal transitions and actions enabled.)
- $\delta(m_0, \diamond, m_1) \wedge \delta(m_1, \diamond, m_2) \;\Rightarrow\; m_1 = m_2$. (A path containing two consecutive null transitions contains a divergent state.)
- $\xi(m, e, m_1) \wedge \xi(m, e, m_2) \;\Rightarrow\; m_1 = m_2$. (From any state, at most one state can be reached via the ξ-relation for a given action.)

The following properties follow from the above, for any semi-deterministic FSM M.

(8) $\xi^*(m_0, t, m_1) \wedge \xi^*(m_0, t, m_2) \;\Rightarrow\; m_1 = m_2$

(9) $\delta^*(i, t, m) \equiv (\exists\, m' : \xi^*(i, t, m') : \mu(m', m))$.

(10) $\xi^*(m_0, tt', m_2) \wedge \xi^*(m_0, t, m_1) \;\Rightarrow\; \xi^*(m_1, t', m_2)$

A state m is called a ***head state*** of a semi-deterministic FSM if $\xi^*(i, t, m)$ for some t. (Note that the initial state is a head state.) The important characteristic of semi-deterministic FSMs is that they can be viewed as deterministic with respect to their head states (see Property (8)).

Proposition 0 For any FSM M there exists a semi-deterministic FSM N such that M *sat* N and N *sat* M. □

Proof. By a standard construction described in [1] we can produce, for a given M, a deterministic FSM D such that $M(t) \equiv D(t)$, together with an injective mapping f from S_D to the nonempty sets of states of M, such that for every trace t and state d of D,

$$\delta^*_D(i, t, d) \;\equiv\; f.d = (\mathbf{SET}\ \ m : \delta^*_M(i, t, m) : m)$$

To construct a semi-deterministic equivalent for M, we modify this D to form D' by adding nondeterminism as follows. Each state d of D is split into two states d_h and d_x, and D' has a transition $\delta(d_h, \diamond, d_x)$ for each d. Each transition $\delta(d, e, d')$ of D becomes $\delta(d_x, e, d'_h)$ in D'. Now, for each d and $m \in f.d$ such that $ext.m$, we add a state d_m and an internal transition $\delta(d_h, \diamond, d_m)$ in D'. Also, we establish $out.d_m = out.m$ in D' by adding a transition $\delta(d_m, e, d'_h)$ for each $e \in out.m$ and the unique d'_h such that $\delta(d_x, e, d'_h)$ (by the determinizing construction, there exists such a d'_h for each e). Finally, if and only if there exists $m \in f.d$ such that $div.m$, we add a state d_v and two internal transitions $\delta(d_h, \diamond, d_v)$ and $\delta(d_v, \diamond, d_v)$ in D'. Clearly $D(t) \equiv D'(t)$ for any t. The proof that D' *sat* M and M *sat* D' is left as an exercise for the reader. □

3 Constructing a Solution

In the next two sections, we describe a general method for solving quotient problems in the the theory just described. For given FSMs A and B, we show how to construct a finite state machine C, such that $B \circ C$ *sat* A, or show that no such FSM exists. By Proposition 0, any given FSMs A and B can be transformed into equivalent semi-deterministic FSMs A' and B'. Thus any quotient problem can be made into an equivalent problem involving semi-deterministic FSMs. Therefore, for the algorithm described here, A and B are assumed to be semi-deterministic. We also assume that neither A nor B is divergent; extension to deal with the case of divergent A or B is straightforward.

Before describing the construction of the particular solution C, we develop several concepts properties that are required of any solution FSM. In the rest of this paper, the predicates and functions defined depend upon the given FSMs A and B. For brevity, our notation leaves this dependence implicit. Also, the variable D ranges over deterministic FSMs that are (or might be) solutions, or have certain properties required of all solutions; this includes the constructed solution C.

3.1 Preliminary Developments

The set of interface actions of any solution D (and of our constructed solution C) is $\Sigma = \Sigma_A \div \Sigma_B$, the set of all actions in Σ_A or Σ_B or not both. Thus C has two interfaces: one with B, and one with the environment. The variables u, v, and w and their subscripted variants range over traces in Σ^*, Σ_B^*, and Σ_A^*, respectively. Note that by definition a trace can be in at most two of these sets.

Consider the relationship among A, B, and a solution D. Each trace of D may match several different traces of B, and thus generates several different traces of the composite $B \circ D$; for correctness, each of these must be a trace of A. In other words, for each state d, there will in general be several distinct B-states b such that $\langle b\, d \rangle$ is a reachable state of $B \circ D$. Each of these may correspond to a different state of A. For correctness, for each reachable state $\langle b\, d \rangle$ of the composite, there must be a corresponding state a of A having certain characteristics.

Constructing C consists in defining its state set and transition relation. Our algorithm does this inductively, beginning with the initial state and analyzing the potential effect of each action in Σ at each state. The analysis ensures that for each state b of B that is reachable via occurrence of that action (and other actions not part of the interface with C), there is a corresponding state a of A, reachable via a similar sequence of actions.

We introduce some terminology relating paths and states of D, B, $B \circ D$, and A. The paths $\delta^*(b, v, b')$ in B and $\delta^*(d, v, d')$ in D are *concurrent* if there exists a path $\delta^*(\langle b\, d \rangle, w, \langle b'\, d' \rangle)$ in $B \circ D$ such that $zip(u, v, w)$. States b of B and d of D are *concurrent* if $\langle b\, d \rangle$ is a reachable state of $B \circ D$, i.e., if there exist concurrent paths $\delta^*(i, v, b)$ in B and $\delta^*(i, u, d)$ in D.

A head state a of A *corresponds to* a state $\langle b\, d\rangle$ of $B \circ D$ if there exists a trace w such that $\delta^*(i, w, \langle b\, d\rangle)$ and $\xi^*(i, w, a)$ in A. A path $\delta^*(\langle b\, d\rangle, w, \langle b'\, d'\rangle)$ in $B \circ D$ *corresponds to* a path $\xi^*(a, w, a')$ in A if $\langle b\, d\rangle$ corresponds to a.

A trace $u \in \Sigma^*$ is *safe* if for every concurrent trace v of B there is a corresponding trace of A. Formally, we define

$$\mathit{safe}.u \;\overset{\text{def}}{=}\; (\forall\, v, w : \mathit{zip}(u, v, w) \,\wedge\, B(v) : A(w))$$

We say FSM D is safe if all of D's traces are safe.

Proposition 1 D is safe if and only if every trace of $B \circ D$ is a trace of A. $\square$

Proof. We calculate:

$$\begin{array}{ll}
 & (\forall\, w : (B \circ D)(w) : A(w)) \\
= & \quad \{ \text{ property (4) } \} \\
 & (\forall\, w : (\exists\, u, v : \mathit{zip}(u, v, w) : D(u) \,\wedge\, B(v)) : A(w)) \\
= & \quad \{ \text{ predicate calculus } \} \\
 & (\forall\, u, v, w : \mathit{zip}(u, v, w) \,\wedge\, D(u) \,\wedge\, B(v) : A(w)) \\
= & \quad \{ \text{ predicate calculus } \} \\
 & (\forall\, u : D(u) : (\forall\, v, w : \mathit{zip}(u, v, w) \,\wedge\, B(v) : A(w))) \\
= & \quad \{ \text{ definition } \mathit{safe} \ \} \\
 & (\forall\, u : D(u) : \mathit{safe}.u) \\
= & \quad \{ \text{ definition } \} \\
 & D \text{ is safe}
\end{array}$$

$\square$

For head states a and b (of A and B respectively), and state d of D, the predicate $T.(a, b, d)$ means that b and d are concurrent, and a is the head state corresponding to $\langle b\, d\rangle$. Formally:

$$T.(a, b, d) \;\overset{\text{def}}{=}\; (\exists\, u, v, w : \mathit{zip}(u, v, w) : \xi^*_A(i, w, a) \,\wedge\, \delta^*_B(i, v, b) \,\wedge\, \delta^*_D(i, u, d))$$

The following result is an immediate consequence of (5) and the above definition:

$$(11) \quad T.(a, b, d) \;\equiv\; (\exists\, w : \xi^*(i, w, a) : \delta^*(i, w, \langle b\, d\rangle))$$

For any head state a of a semi-deterministic FSM A, and any set $G \subseteq \Sigma_A$, the predicate $\mathit{prog}.(a, G)$ means that ext holds at a state, internally-reachable from a, at which only events in G are enabled. Formally, we define

$$\mathit{prog}.(a, G) \;\overset{\text{def}}{=}\; (\exists\, a' : \mu(a, a') : \mathit{ext}.a' \,\wedge\, \mathit{out}.a' \subseteq G)$$

From this definition follows a "monotonicity" property:

(12) $G \subseteq G' \wedge prog.(a, G) \Rightarrow prog.(a, G')$.

For a state d of D, and a set G of actions of D, the predicate $next.(d, G)$ means that if the set of actions enabled at d contains G, then no deadlock can occur at d unless it can also occur at some corresponding A-state. We define

$$next.(d, G) \overset{\text{def}}{=} (\forall a, b : T.(a, b, d) : (out.b \cap G \neq \emptyset) \vee prog.(a, (out.b \div G)))$$

A property similar to (12) is immediate from this definition:

(13) $G \subseteq G' \wedge next.(d, G) \Rightarrow next.(d, G')$.

A state d of D is *progressive* iff $next.(d, out.d)$. A state is nonprogressive if it is not progressive. "D is progressive" means every state of D is progressive. (Note that a state that is not reachable is trivially progressive.)

We can now state necessary and sufficient conditions for D to be a solution in terms of the above-defined local properties.

Proposition 2 Let A and B be semi-deterministic and nondivergent, and let D be deterministic. Then $B \circ D$ *sat* A if and only if the following three conditions are satisfied: (i) D is safe; (ii) D is progressive; (iii) $B \circ D$ is nondivergent. □

Proof of this result is given in the Appendix.

We now turn to our solution method. The construction of the solution C takes place in two or three phases, corresponding to the above conditions for solutionhood. In the first step, we construct a safe FSM C_0. Then we refine it (if possible) so that it becomes progressive, yielding C_p. Finally, if the composite $B \circ C_p$ is divergent, we attempt break down C_p into parts and reassemble them to form C in a way that eliminates divergences from $B \circ C$. A failure in any of the three phases indicates that no solution exists, and provides an indication of why.

3.2 Constructing C_0

Our method of constructing the state set and transition relation of C_0 is based on the observation that, for any trace $u \in \Sigma^*$ of any solution D, the set of all pairs of (head) states of A and B reachable via traces that "*zip* with" u captures all available information about the composite system $B \circ D$ and how its behavior corresponds to that allowed by A. In particular, from such a set it can easily be determined whether a safe trace can safely be extended by a particular action. We shall construct C_0 so that each state c is identified with a distinct set of pairs of head states a and b such that $T.(a, b, c)$.

The next few definitions formalize the concepts needed to define the state space and transition relation of C_0 and prove its correctness.

First we define a function, h, that maps a trace to the set of all pairs of head states of A and B that are possible states of the system after that trace. For trace $u \in \Sigma^*$,

$$h.u \stackrel{\text{def}}{=} (\mathbf{SET}\ a, b, v, w : \xi_A^*(i, w, a) \land \xi_B^*(i, v, b) \land zip(u, v, w) : (a, b))$$

This function provides a way to abstract from the structure of a particular FSM and deal with traces, as indicated by the following result.

Proposition 3 The function h and the relation T satisfy, for any a, b, and d:

$$T.(a, b, d) \equiv (\exists\, u, b' : \delta_D^*(i, u, d) : (a, b') \in h.u \land \mu(b', b))$$

□

Proof. We observe for any a, b, and d,

$$\begin{array}{cl}
 & T.(a, b, d) \\
= & \quad \{ \text{ definition } T \ \} \\
 & (\exists\, u, v, w : zip(u, v, w) : \xi_A^*(i, w, a) \land \delta_B^*(i, v, b) \land \delta_D^*(i, u, d)) \\
= & \quad \{ \ B \text{ is semi-deterministic: (9) } \} \\
 & (\exists\, u, v, w : zip(u, v, w) : \xi_A^*(i, w, a) \land \\
 & \qquad (\exists\, b' : \xi_B^*(i, v, b') : \mu(b', b)\,) \land \delta_D^*(i, u, d)\,) \\
= & \quad \{ \text{ predicate calculus } \} \\
 & (\exists\, u, b' : \delta_D^*(i, u, d) : (\exists\, v, w : zip(u, v, w) : \xi_A^*(i, w, a) \land \xi_B^*(i, v, b')) \land \\
 & \qquad \mu(b', b)) \\
= & \quad \{ \text{ definition } h.u \ \} \\
 & (\exists\, u, b' : \delta_D^*(i, u, d) : (a, b') \in h.u \land \mu(b', b))
\end{array}$$

□

From the set of pairs $h.u$, it is possible to determine whether a trace u can safely be extended by a given action e. The predicate ss ("safe step") captures this notion. For a set J of pairs of head states (i.e., $h.u$ for some u) and action $e \in \Sigma$, $ss.(J, e)$ means that for any pair (a, b) in J, for each path from b in B concurrent with e there is a corresponding path from a. Formally,

$$\begin{array}{rcl}
ss.(J, e) & \stackrel{\text{def}}{=} & (\forall\, a, b, v, w\ : (a, b) \in J \land zip(e, v, w) \land (\exists\, b' :: \xi^*(b, v, b'))\ : \\
 & & (\exists\, a' :: \xi^*(a, w, a'))\,)
\end{array}$$

The use of predicate ss in the construction of C_0 is indicated by the following result, which is proved in the appendix.

Proposition 4 For any $u \in \Sigma^*$, $e \in \Sigma$:

$$safe.u \ \Rightarrow\ ss.(h.u, e) \equiv safe.ue$$

□

If the null trace is not safe, then there is no safe FSM (because the null trace is a trace of every FSM). In this case, by Proposition 2, it is clear that no solution exists. Therefore we proceed under the assumption

(14) $safe.\diamond$.

Note that this assumption is easily checked for given A and B by a simple inductive computation.

Observe that $h.\diamond$ is easily computed by an inductive closure procedure from the singleton set $\{(i_A, i_B)\}$. (Obviously $T.(i_A, i_B, i_C)$.) Given $h.u$, $h.ue$ is also straightforward to compute by inductive closure. We assume the existence of a function representing this computation: the function φ maps a set J of pairs and an action $e \in \Sigma$ to another set of pairs, and is defined by the equation $\varphi.(h.u, e) = h.ue$. (Note that the range of the function h is finite: the number of distinct sets of pairs of (head) states of A and B is finite.) Finally, for given $J = h.u$ and e, checking $ss.(J, e)$ is a simple local computation.

The point of these observations is that a safe machine C_0 can be defined in terms of $h.\diamond$, ss, and φ, and there is no need to deal explicitly with traces. We define $C_0 = (\Sigma, S_0, \delta_0, i)$, where each state $c \in S_0$ is identified with a distinct set $tag.c$, satisfying:

C0 $tag.i = h.\diamond$

C1 $\delta_0(c, e, c') \;\Rightarrow\; tag.c' = \varphi.(tag.c, e)$

C2 $ss.(tag.c, e) \equiv (\exists\, c' : c' \in S_0 : \delta_0(c, e, c'))$

That is, C_0 has a single state for each set $h.u$ such that u and all of its prefixes are safe. Note that C_0 has no internal transitions, and by definition has at most one transition for any action from any state; thus C_0 is deterministic. The actual construction of C_0 is described shortly; we first prove that the above conditions imply certain properties of C_0.

Proposition 5 If C_0 satisfies conditions **C0** and **C1**, then for any state c of C_0, and any trace u, $\delta_0^*(i, u, c) \;\Rightarrow\; tag.c = h.u$. □

Proof. The proof is by induction on the length of u. Because C_0 is deterministic, there is at most one c s.t. $\delta_0^*(i, u, c)$ for any u. The base case, $u = \diamond$, follows from **C0**. For the inductive step, assume $\delta_0^*(i, ue, c)$. By the definition of the trace relation, there exists a unique c' such that $\delta_0^*(i, u, c')$ and $\delta(c', e, c)$; by the inductive hypothesis, $tag.c' = h.u$. By **C1**, $tag.c = \varphi.(tag.c', e)$; thus $tag.c = \varphi.(h.u, e)$; the definition of φ then gives $tag.c = h.ue$. □

We say an FSM C is *maximal* if, for any D such that $B \circ D$ *sat* A,

$$(\forall\, u : u \in \Sigma^* : D(u) \;\Rightarrow\; C(u))$$

Proposition 6 If C_0 satisfies **C0-C2**, then

(15) C_0 is safe

(16) C_0 is maximal

$\square$

Proof of (15). We show $C_0(u) \Rightarrow safe.u$ by induction on the length of u. For the basis, $safe.\diamond$ holds by assumption (14). For the inductive step, assume $C_0(ue)$; that is, $\delta_0^*(i, ue, c)$ for some c. Then there exists a unique c' such that $\delta_0^*(i, u, c')$ and $\delta_0(c', e, c)$. By the Proposition 5, $tag.c' = h.u$, and by the inductive hypothesis we have $safe.u$. Using these facts, we calculate:

$$\begin{array}{ll}
 & safe.u \wedge tag.c' = h.u \wedge \delta_0(c', e, c) \\
\Rightarrow & \quad \{ \text{ predicate calculus and } \mathbf{C2} \} \\
 & safe.u \wedge tag.c' = h.u \wedge ss.(tag.c', e) \\
\Rightarrow & \quad \{ \text{ predicate calculus } \} \\
 & safe.u \wedge ss.(h.u, e) \\
\Rightarrow & \quad \{ \text{ Proposition 4 } \} \\
 & safe.ue
\end{array}$$

$\square$

Proof of (16). We have to show that for any solution D, $D(u) \Rightarrow C_0(u)$, for any trace u. Again we use induction on trace length. The base case holds trivially. For the inductive step, we observe for any trace u and action e,

$$\begin{array}{ll}
 & D(ue) \\
= & \quad \{ \text{ prefix-closure of trace sets } \} \\
 & D(u) \wedge D(ue) \\
\Rightarrow & \quad \{ D \text{ is a solution, hence is safe } \} \\
 & D(u) \wedge safe.u \wedge safe.ue \\
\Rightarrow & \quad \{ \text{ inductive hypothesis } \} \\
 & C_0(u) \wedge safe.u \wedge safe.ue \\
\Rightarrow & \quad \{ \text{ definition } C_0(u)\text{; Proposition 4 and pred. calc. } \} \\
 & (\exists\, c :: \delta_0^*(i, u, c)) \wedge ss.(h.u, e) \\
= & \quad \{ \text{ Proposition 5 } \} \\
 & (\exists\, c :: \delta_0^*(i, u, c) \wedge tag.c = h.u) \wedge ss.(h.u, e) \\
= & \quad \{ \text{ predicate calculus } \} \\
 & (\exists\, c :: \delta_0^*(i, u, c) \wedge tag.c = h.u \wedge ss.(h.u, e)) \\
\Rightarrow & \quad \{ \text{ rule of Leibniz } \} \\
 & (\exists\, c :: \delta_0^*(i, u, c) \wedge ss.(tag.c, e)) \\
\Rightarrow & \quad \{ \mathbf{C2} \} \\
 & (\exists\, c :: \delta_0^*(i, u, c) \wedge (\exists\, c' :: \delta(c, e, c'))) \\
\Rightarrow & \quad \{ \text{ definition of } \delta^* \}
\end{array}$$

```
$\langle\langle$ $safe.\diamond$ $\rangle\rangle$
$tag.i := h.\diamond$;
$new, S_0, \delta_0 := \{i\}, \emptyset, \emptyset$;
$\langle\langle$ (17) $\wedge$ (18) $\wedge$ (19) $\rangle\rangle$
while $new \neq \emptyset$ do:
    select $c \in new$ and remove it from $new$;
    for each $e \in \Sigma$ such that $ss.(tag.c, e)$ do:
        if $(\exists\, c' : c' \in new \cup S_0 : tag.c' = \varphi.(tag.c, e))$
            then $c'' := c'$
            else create $c''$;
                $tag.c'' := \varphi.(tag.c, e)$;
                $new := new \cup \{c''\}$
        end if $\langle\langle$ $tag.c'' = \varphi.(tag.c, e)$ $\rangle\rangle$
        $\delta_0 := \delta_0 \cup \{(c, e, c'')\}$
        $\langle\langle$ $(\exists\, c' : c' \in new \cup S_0 : \delta_0(c, e, c')) \equiv ss.(tag.c, e)$ $\rangle\rangle$
    end for
    $S_0 := S_0 \cup \{c\}$
$\langle\langle$ (17) $\wedge$ (18) $\wedge$ (19) $\rangle\rangle$
end while
```

Figure 5: Algorithm 0: Construction of C_0.

$$\begin{aligned} & (\exists\, c :: \delta_0^*(i, ue, c)) \\ = \quad & \quad \{ \text{ definition } \} \\ & C_0(ue) \end{aligned}$$

This completes the induction and the proof. □

Given implementations of ss and φ, conditions **C0**–**C2** suggest an inductive procedure like Algorithm 0 in Figure 5. (Algorithms are presented in a semi-formal Pascal-like language. Text $\langle\langle$ enclosed in brackets $\rangle\rangle$ is a comment.) It is not difficult to show that Algorithm 0 maintains the following invariants, which, together with the termination condition, imply **C0**–**C2**.

(17) $tag.i = h.\diamond$

(18) $(\forall\, c, c' : c, c' \in S \cup new : \delta(c, e, c') \Rightarrow tag.c' = \varphi.(tag.c, e))$

(19) $(\forall\, c : c \in S : ss.(tag.c, e) \equiv (\exists\, c' : c' \in S \cup new : \delta(c, e, c')))$

Algorithm 0 is guaranteed to produce a FSM when (14) is satisfied. That is, we can always produce a maximal safe FSM, provided the null trace is safe. This FSM will serve as an "upper bound" on the solution in the rest of the construction.

Note that C_0 potentially can have a state for each distinct set of pairs of head states of A and B. Thus, its size (i.e. number of states) may be an exponential function of the sizes of A and B. In our experience, however, this has not occurred; it seems to require a very complex interdependence between B and A. Note also that C_0 is not necessarily the "minimal" FSM for a given trace set, although it is deterministic. In particular, C_0 may have states that are distinct, even though they are functionally equivalent with respect to the set of paths originating in each state. It is possible, however, to minimize a deterministic FSM in time polynomial in the number of states [10].

3.2.1 Making C_0 Progressive

In the next step of the construction, we refine the output of the first step into a FSM that always has "enough" actions enabled. Crucial to this refinement process is the maximality of C_0, from which follow several useful properties.

Proposition 7 Let D be any safe, deterministic FSM, and let C be maximal and satisfy **C0** and **C1**, and assume $\delta_D^*(i, u, d)$ and $\delta_C^*(i, u, c)$. Then:

(20) $(\forall a, b : T.(a, b, c) : T.(a, b, d))$

(21) $out.d \subseteq out.c$

(22) $(\forall G : G \subseteq \Sigma : next.(d, G) \Rightarrow next.(c, G))$

(23) d is progressive $\Rightarrow$ c is progressive.

□

Proof. See Appendix. □

We "refine" C_0 by iterative removal of nonprogressive states (we "remove" a state from a FSM by removing the state and all its incoming and outgoing transitions). This is straightforward because all the information necessary to compute the predicate $next.(c, out.c)$ is contained in the state's *tag* set and in the transition relations of A and B. The refinement procedure is given as Algorithm 1 in Figure 6; it computes a sequence of state sets S_j and transition relations δ_j, beginning with S_0, δ_0, and satisfying

$$\begin{aligned} S_{j+1} &= (\textbf{SET } c : c \in S_j \wedge next.(c, out.c) : c) \cup \{i\} \\ \delta_{j+1} &= (\textbf{SET } c, e, c' : (c, e, c') \in \delta_j \wedge c, c' \in S_{j+1} : (c, e, c')) \end{aligned}$$

where i is always the initial state. Thus S_{j+1} is the progressive subset of S_j (plus i), and δ_{j+1} is δ_j restricted to S_{j+1}. Note that the initial state i is not removed. Iteration is required because removal of a state may shrink the *out* set of another state, making it non-progressive. Now define $C_j = (\Sigma, S_j, \delta_j, i)$.

```
$S, \delta := S_0, \delta_0$;
while $next.(i, out.i) \wedge (\exists c : c \in S : \neg next.(c, out.c))$ do:
    $S :=$ (SET $c : c \in S_j \wedge next.(c, out.c) : c) \cup \{i\}$
    $\delta_{j+1} :=$ (SET $c, e, c' : (c, e, c') \in \delta_j \wedge c, c' \in S_{j+1} : (c, e, c'))$
end while
let $C_p = (\Sigma, S, \delta, i)$;
if $next.(i, out.i)$
    then $\langle\langle$ $C_p$ is safe, progressive and maximal $\rangle\rangle$
    else $\langle\langle$ no safe and progressive FSM exists $\rangle\rangle$
end if
```

Figure 6: Algorithm 1: Making C_0 progressive

Proposition 8 For each FSM C_j in the sequence of machines defined above,

(24) if C_j is maximal and i is progressive in C_j, then C_{j+1} is maximal.

(25) If C_j is maximal and i is not progressive in C_j, then there is no progressive and safe FSM.

□

Proof of (24). Let C_j be maximal, and assume i is progressive in C_j. Let C_{j+1} be obtained from C_j by removal of some nonprogressive states and associated transitions. Let D be any solution. We show $D(u)$ implies $C_{j+1}(u)$ by induction on the length of u. The basis, $u = \diamond$, is trivial. For the inductive step, assume $\delta^*_D(i, ue, d)$. Because C_j is maximal and deterministic, there exist particular states c' and c of C_j such that $\delta^*_j(i, u, c')$ and $\delta_j(c', e, c)$. By the inductive hypothesis, u is a trace of C_{j+1}; thus $\delta^*_{j+1}(i, u, c')$ as well. Because D is a solution, d is progressive, hence—property (23)—c is progressive in C_j, hence $c \in S_{j+1}$ and thus $\delta_{j+1}(c', e, c)$. Therefore ue is a trace of C_{j+1}. □

Proof of (25). We show that the existence of a solution implies that the initial state of a C_j is progressive. Assume D is a solution. By Proposition 2, D is progressive, hence the initial state of D is progressive. We have $\delta^*_D(i, \diamond, i)$ and also $\delta^*_j(i, \diamond, i)$. Now, C_j is maximal, and satisfies conditions **C0** and **C1**, so the hypothesis of Proposition 7 is satisfied. By (23), then, the initial state of C_j is progressive. □

Removing states clearly preserves safety and maximality. Thus we conclude that if upon termination of Algorithm 1 the initial state of C_p is progressive, then C_p is safe, progressive, and maximal, and otherwise no solution exists.

4 Ensuring Nondivergence

We have shown how to construct a safe, progressive, and maximal FSM. Now, B is nondivergent by assumption, and C_p is nondivergent by construction. Property (3) then tells us that $B \circ C_p$ is nondivergent if and only if there are no *concurrent cycles* in B and C_p, i.e. there are no concurrent states b and c and non-null trace u such that $\delta^*_B(b, u, b)$ and $\delta^*_C(c, u, c)$. Note that this property can be checked by constructing $B \circ C_p$, which is straightforward.

If $B \circ C_p$ is nondivergent, then C_p is a solution. Otherwise, we attempt to derive C from C_p as described in this section. The hard part of the problem is detecting nonexistence. The method described here is expensive, but fortunately it is usually not necessary to deal with divergence. More commonly, the FSM C_p is a solution, or else it is possible to conclude after the second step that no solution exists.

The basic idea of our approach is to "unroll" the cycles of C_p to form a collection of sub-FSMs, each of which has the property that none of its paths from the initial state is concurrent with any cycle in B. Each path of one of these sub-FSMs is identical to some path of C_p. The sub-FSMs are then combined to form C in such a way that each cycle in C contains a path of one of the sub-FSMs. Thus, none of C's cycles is concurrent with any cycle of B, and hence the composite $B \circ C$ is nondivergent. If it is not possible to construct an adequate (in a sense to be defined later) collection of sub-FSMs, no solution exists.

4.1 Still More Definitions

A *tree-FSM* is a deterministic finite state machine with the structure of a directed tree; the initial state of the FSM is called the *root*. For tree-FSMs, we denote the initial state by r. The (unique) path from r to any state is called the *root path*. A state m is a *leaf state* of a tree-FSM iff it has no outgoing edges, i.e., it satisfies $out.m = \emptyset$. In what follows, "tree" means "tree-FSM." As in the foregoing sections, D refers here to an arbitrary FSM with $\Sigma_D = \Sigma$.

An *unrolling* of an FSM D consists of a tree-FSM M plus a labeling function L. The labeling function L maps each state of M to a tuple of the form $\langle d, \beta_1, \ldots, \beta_K \rangle$, where d is a state of D, and each β_j, $0 < j \leq K$, is either a head state of B, or the distinguished symbol $\otimes$. Each unrolling is associated with a particular state of D, namely the D-state in the label of the root of the tree-FSM; the unrolling is said to be *from that state*. By "a state of an unrolling" we mean a state of the tree-FSM of the unrolling.

By definition, the tree-FSM M and labeling function L of an unrolling of D from d together satisfy the following. For the initial state r, $L.r = \langle d, b_1, \ldots, b_K \rangle$, where $b_1, \ldots, b_K$ is some fixed ordering of all the head states of b such that, for some a, $T.(a, b, d)$—i.e., all the head states concurrent with d. For each state $m \neq r$, the label is determined inductively by the root path to m, according to:

$$\begin{aligned} L.m = \langle d, \beta_1, \ldots, \beta_K \rangle \wedge \delta(m, e, m') \;\Rightarrow \\ (\exists\, d' :: \delta(d, e, d') \wedge L.m' = \langle d', \beta'_1, \ldots \beta'_K \rangle) \end{aligned}$$

where, for each j, $0 < j \leq K$, β'_j in $L.m'$ is given by

$$\beta'_j = \begin{cases} b' & \text{if}\, \beta_j \neq \otimes \wedge \xi_B(\beta_j, e, b') \\ \otimes & \text{otherwise} \end{cases}$$

The individual elements of the tuple $L.m$ are represented by the following notation: $L_0.m$ is the D-state component and $L_j.m$, $0 < j \leq K$, is the jth B-state component. That is, $L.m = \langle L_0.m, L_1.m, \ldots, L_K.m \rangle$.

Note that for β'_j to be well-defined there can be at most one b' such that $\xi_B(\beta_j, e, b')$. This is the only place where we make essential use of the fact that B is semi-deterministic. (In particular, the construction of the safe and progressive C_p is essentially the same for arbitrary B.) Note also that

$$\delta_M(m, e, m') \;\Rightarrow\; \delta_D(L_0.m, e, L_0.m')$$

The idea behind an unrolling from d is that the β components of the labels "track" the paths in B that begin at a head state concurrent with d, indicating when each such path has ended by the presence of the symbol $\otimes$. For each state m of the unrolling, and each label component $L_j.m \neq \otimes$, $L_0.m$ and $L_j.m$ are concurrent, i.e. there exists a state a of A such that $T.(a, L_j.m, L_0.m)$. Moreover, for any states m and m', and any $u \in \Sigma^*$, $\delta^*_M(m, u, m')$ in the tree-FSM M implies $\delta^*_D(L_0.m, u, L_0.m')$, and if $L_j.m \neq \otimes$ and $L_j.m' \neq \otimes$ for $0 < j \leq K$, then $\xi^*_B(L_j.m, u, L_j.m')$. A straightforward induction shows that, for each state m of an unrolling from d, $L.m$ is uniquely determinied by d and the (unique) trace u such that $\delta^*_M(r, u, m)$.

A state m of an unrolling is *dead* iff $(\forall j : 0 < j \leq K : L_j.m = \otimes)$; a state is nondead iff it is not dead. A state m is *terminal* iff $out.(L_0.m) = \emptyset$. Note that every terminal state is a leaf of the tree-FSM.

An unrolling is *acceptable* iff every leaf is either dead or terminal. No path from root to leaf in an acceptable unrolling matches any concurrent cycle in B.

A state m in an unrolling is *progressive* iff $next.(L_0.m, out.m)$. A state is nonprogressive iff it is not progressive. An unrolling is progressive iff every state is either dead or progressive.

A transition $\delta(m, e, m')$ in an unrolling is a *back edge* iff m' is not dead and has the same label as one of its ancestors in the tree. That is, there exist traces u, u', and state m_0 such that $\delta^*(r, u, m_0)$ and $\delta^*(m_0, u', m)$, and $L.m_0 = L.m'$. The presence of a back edge in an unrolling indicates the presence of a cycle in B matching a concurrent cycle in D, as shown by the following proposition.

Proposition 9 For deterministic D, $B \circ D$ is divergent if there exists a reachable state d of D such that some unrolling from d contains a back edge. $\square$

Proof. Let (M, L) be an unrolling from d, and let $\delta(m, e, m')$ be a back edge. Then there exists an m_0, and traces u, u' such that

$$\delta^*(r, u, m_0) \wedge \delta^*(m_0, u', m) \wedge L.m_0 = L.m'$$

By the definition of back edge, m' is not dead, so there exists $j > 0$ such that $L_j.m' \neq \otimes$. Therefore let $L_j.m' = b$, for some state b of B. For that b, by the definition of unrolling, we have $\delta^*_B(b, u'e, b)$, and b. Letting $d' = L_0.m'$, d' and b are concurrent. Moreover, we also have $\delta^*_D(d', u'e, d')$, and thus by (3), $div.\langle b\, d' \rangle$. Hence $B \circ D$ is divergent. □

An unrolling is *trim* iff it has no back edges, and every dead state is a leaf. A trim unrolling does not contain any more edges than are necessary for acceptability; that is, it cannot be made "more acceptable" by adding edges.

Proposition 10 For any D and d, if there exists an acceptable and progressive unrolling of D from d, then there exists a trim, acceptable and progressive unrolling from d. □

Proof. (See Appendix.) □

Henceforth, we write "t.p.a." as an abbreviation for "trim, progressive and acceptable."

A trim unrolling is *full* iff none of its paths can be extended without violating either the definition of trimness or the definition of unrolling. That is, for each nondead state m, and each $e \in out.(L_0.m)$, either there exists m' such that $\delta(m, e, m')$ or such an edge would be a back edge. From this definition, it follows that all full trim unrollings from d have the same trace set, and therefore the same structure; hereafter we refer to "the" full trim unrolling from d.

Proposition 11 For any d, there exists a full trim unrolling of D from d. □

Proof. Algorithm 2, shown in Figure 7, constructs the tree-FSM and label function for the full trim unrolling from a given state d. The algorithm makes use of the transition relations of B and D, and the set of head states concurrent with d. The algorithm obviously produces a tree-FSM, and its labeling function satisfies the definition of an unrolling. The resulting unrolling is full and trim, because every nondead state m has a transition $\delta(m, e, m')$ for every $e \in out.(L_0.m)$, except those that would be back-edges. Also, no dead state has an outgoing transition. Termination is guaranteed because the number of distinct labels is finite (no root path of a trim FSM can be longer than the number of distinct labels). □

The foregoing definitions are given in terms of an arbitrary FSM D so that they can be applied in showing nonexistence of a solution; our aim, however, is

```
$new, S, \delta, L.r := \{r\}, \emptyset, \emptyset, \langle d, b_1, \ldots, b_K \rangle$
    $\langle\langle$ $b_j$'s = head states such that $T.(a, b_j, d)$ for some $a$ $\rangle\rangle$
**while** $new \neq \emptyset$ **do**:
    select $m \in new$;
    **if** $(\exists j : 0 < j \leq K : L_j.m \neq \otimes)$ **then**
        $\langle\langle$ $m$ is not dead $\rangle\rangle$
        **for each** $e \in \Sigma$ and $d'$ such that $\delta(L_0.m, e, d')$ **do**:
            $lab := \langle d', \beta'_1, \ldots, \beta'_K \rangle$
                $\langle\langle$ where $\beta'_j$s are per def'n of unrolling $\rangle\rangle$;
            **if** no $m'$ on the root path to $m$ has $L.m' = lab$ **then**
                create a new state $m'$;
                $L.m' := lab$;
                $new, \delta := new \cup \{m'\}, \delta \cup \{(m, e, m')\}$;
            **end if**
        **end for**
    **end if**
    $new, S := new - \{m\}, S \cup \{m\}$
**end while**
```

Figure 7: Algorithm 2: Constructing the full trim unrolling from d.

to deal with unrollings of the particular FSM C_p. The next few definitions deal with properties of unrollings of C_p.

A set W of unrollings from states of C_p is *reasonable* iff it satisfies the following conditions:

- every unrolling in W is t.p.a..
- W contains an unrolling from the initial state of C_p.
- For each unrolling (M, L) in W, and each leaf m of M, W contains an unrolling from $L_0.m$
- W contains at most one unrolling from any state of C_p.

The point of the concept of a reasonable set is that C can be constructed from a given reasonable set W of unrollings, by aggregating each leaf of each unrolling in the set with the root of the unrolling from the state corresponding to that leaf, as follows. For each state c of C_p such that W contains an unrolling (M, L) from c, S_C contains a state r_c for the root of the unrolling, plus a state m_c for each nonleaf state of the unrolling other than the root. For each transition $\delta(m, e, m')$ in M such that m' is a nonleaf state, C has a transition $\delta(m_c, e, m'_c)$. For each transition $\delta(m, e, m')$ in M such that m' is a leaf, C has a transition $\delta(m_c, e, r_{c'})$, where $c' = L_0.m'$. That is, the leaf m' is identified with the root

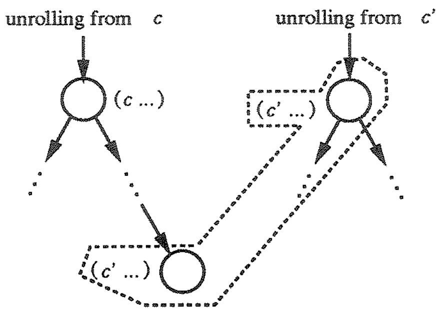

Figure 8: Connecting unrollings by identifying leaves and roots

of the unrolling from state $L_0.m'$ (existence of the latter unrolling is guaranteed by the definition of a reasonable set). The initial state of C is r_i, the state corresponding to the root of the unrolling from the initial state of C_p. Figure 8 illustrates the basic idea of the construction.

Proposition 12 The FSM C constructed from the reasonable set W as described above is a solution. □

The proof of Proposition 12 is left as an exercise for the reader.

Proposition 13 If there exists a solution, then there exists a reasonable set of unrollings of C_p. □

Proof. See appendix. □

4.2 Constructing a Reasonable Set

The remainder of this section is devoted to showing how to construct a reasonable set of unrollings if one exists. Proofs are mostly relegated to the appendix.

An unrolling (M, L) from c is a *sub-unrolling* of another unrolling (N, L') from c iff every trace of M is a trace of N. For any state c of C_p, an unrolling of C_p from c is defined to be *r-maximal* iff it is trim and any unrolling from c contained in a reasonable set of unrollings is a sub-unrolling of it.

Proposition 14 The full trim unrolling from any state c of C_p is r-maximal. □

Proof. See appendix. □

Proposition 15 If (M, L) is an r-maximal unrolling from c, and the initial state (root) of M is either a nondead, nonprogressive state or a nondead, nonterminal leaf, then no reasonable set contains an unrolling from c. □

```
⟨⟨ $(M, L)$ is an r-maximal unrolling from $c$ ⟩⟩
while $(M, L)$ has a state $m \neq r$ such that
        ($m$ is nondead and nonprogressive $\vee$
        $m$ is a nondead, nonterminal leaf) do:
    select any $m$ such that
            $m$ is a nondead and nonprogressive state $\vee$
            $m$ is a nondead and nonterminal leaf;
    remove $m$ from $(M, L)$
end while
if $r$ is nondead and nonprogressive $\vee$
  $r$ is a nondead and nonterminal leaf
    then ⟨⟨ no t.p.a. unrolling from $c$ exists ⟩⟩
    else ⟨⟨ $(M, L)$ is t.p.a. ⟩⟩
end if
```

Figure 9: Algorithm 3. Making (M, L) progressive and acceptable

Proof. See appendix. □

Proposition 16 Let (M, L) be an r-maximal unrolling from c, and let m be a nondead, nonterminal leaf or a nonprogressive, nondead state such that $m \neq r$. Let (M', L') be obtained from M by removing m. Then (M', L') is an r-maximal unrolling from c. □

Proof. See appendix. □

Now, an r-maximal unrolling from c is nonprogressive if and only if it contains a nondead, nonprogressive state; it is not acceptable if and only if it has a nondead, nonterminal leaf. The preceding result says that removing such states from an r-maximal unrolling preserves r-maximality; thus we can remove states as necessary until we obtain an r-maximal t.p.a. unrolling from c, or until it becomes evident that no such unrolling from c exists. This procedure is shown as Algorithm 3. Proposition 16 implies that Algorithm 3 maintains the invariant "(M, L) is r-maximal" if the unrolling is initially r-maximal; the condition of the root upon termination indicates whether a t.p.a. unrolling from c exists. Thus, we can use Algorithm 2 to construct an r-maximal unrolling for each state of C_p, and then use Algorithm 3 to obtain a t.p.a. unrolling from it. These observations lead us to consider construction of a set of t.p.a. unrollings from states of C_p that would be an "upper bound" on all reasonable sets, in the sense that any unrolling in any reasonable set would be a sub-unrolling of some unrolling in the set. As we shall see, such a set can be used to construct a reasonable set, or determine that none exists.

A set W of unrollings from states of C_p is called *optimal* iff both of the following conditions are satisfied:

- every unrolling in W is r-maximal
- if W does not contain an unrolling from c, then no reasonable set contains an unrolling from c.

By definition, if W is optimal and W' is reasonable, then every unrolling in W' is a sub-unrolling of some unrolling in W. The next results show how an optimal set can be refined into a reasonable set.

Proposition 17 If W is optimal, and does not contain an unrolling from the initial state of C_p, then no reasonable set of unrollings of C_p exists. □

Proof. Immediate from the definitions. □

Proposition 18 If W is optimal, $(M, L) \in W$, and (M', L') is the t.p.a. result of applying Algorithm 3 to (M, L), and W' is W with (M, L) replaced by (M', L'), then W' is optimal. □

Proof. See Appendix. □

Proposition 19 Let W be optimal, and $(M, L) \in W$ be an unrolling from c such that M has a leaf state m such that W does not contain an unrolling from $L_0.m$. If (M', L') is obtained from (M, L) by removing m, then (M', L') is r-maximal. □

Proof. See Appendix. □

The above results are the basis of the following method for constructing a reasonable set. First, we construct an r-maximal unrolling from each state c of C_p, using Algorithm 2. Then we apply Algorithm 3 to each such unrolling, and put each resulting t.p.a. unrolling in the set W_0 (for the others, there is no t.p.a. unrolling). Clearly W_0 is optimal. Next we iteratively refine W_0 while preserving optimality, by removing leaves as described above, and then applying Algorithm 3 to the resulting unrolling. If the result of Algorithm 3 is a t.p.a. unrolling, it remains in W_0; otherwise, its unrolling may be removed from W_0 without affecting optimality. Algorithm 4 summarizes this process. The second loop maintains the invariants "W_0 is optimal" and "every unrolling in W_0 is t.p.a.." The combination of these and the termination condition implies that upon termination, either W_0 is reasonable, or W_0 contains no unrolling from the initial state. In the latter case—by Proposition 15—no reasonable set exists, and hence—by Proposition 13—no solution exists. Termination of Algorithm 4 is guaranteed, because each iteration removes some leaf of some unrolling.

```
$W_0 := \emptyset$;
for each $c \in C_p.S$ do:
    construct the full trim unrolling from $c$ using Algorithm 2;
    apply Algorithm 3;
    if the result is a t.p.a. unrolling $(M, L)$ then
        $W_0 := W_0 \cup \{(M, L)\}$;
    end if
end for
$\langle\langle$ $W_0$ is optimal $\wedge$ every unrolling in $W_0$ is t.p.a. $\rangle\rangle$
while $W_0$ contains an unrolling from $i$
    $\wedge$ $W_0$ contains an unrolling $(M, L)$ with a leaf $m$,
    such that $W_0$ does not contain an unrolling from $L_0.m$
    do:
        $W_0 := W_0 - \{(M, L)\}$;
        remove $m$ from $(M, L)$ to form $(M', L')$;
        $\langle\langle$ $(M', L')$ is r-maximal $\rangle\rangle$
        apply Algorithm 3 to $(M', L')$ to form $(M'', L'')$;
        if $(M'', L'')$ is t.p.a. then
            $\langle\langle$ $(M'', L'')$ is r-maximal $\rangle\rangle$
            $W_0 := W_0 \cup \{(M'', L'')\}$
        end if
end while
$\langle\langle$ $W_0$ is reasonable or no reasonable set exists $\rangle\rangle$
```

Figure 10: Algorithm 4: Constructing a reasonable set

5 Summary and Conclusions

We have presented a general theory in which concurrent systems are modeled as synchronously-interacting finite state machines. We have presented a method that, given A and B, produces a finite state machine C such that $B \circ C$ *sat* A if one exists, or indicates why no solution exists. Algorithms 0 and 1 give a method of constructing a safe and progressive FSM. Algorithms 2-4 take that FSM and make it nondivergent (if necessary), or decide that it is not possible to do so.

A number of researchers have considered various forms of the quotient problem. Hoare and He [9] studied the "weakest prespecification" problem, in which *sequential* programs are modeled by relations on states, and the composition operator is relational composition. For Milner's Calculus of Communicating Systems [13], Parrow [14] has described a "semi-algorithmic" method of solving for X in the quotient equation, with the *bisimulation* relation as satisfaction. That method, which relies on human guidance in constructing the quotient, uses a tableau approach. Shields [15] also has given a solution in the context of CCS bisimulation. Lai and Sanders [11] studied the *weakest environment* operator for Hoare's Communicating Sequential Processes. The weakest environment for given A and B is defined to be the weakest process X satisfying $B||X \sqsubseteq A$ The CSP composition operator $||$ does not entail hiding the interactions between components, and Lai and Sanders' treatment does not deal with divergence.

Merlin and Bochmann [12] gave an efficient method of constructing a solution X from given A and B, where A, B and X are (deterministic) finite state machines. However, their theory did not deal with deadlock or divergence; hence their method produces a solution that may deadlock.

Some of the algorithms given here have exponential worst-case complexity. The size of the state space of C_0 can be of the order of $2^{|S_A \times S_B|}$, because each state of C_0 is identified with a distinct subset of $S_A \times S_B$. The size of the full trim unrolling from a state, used in the final phase of the algorithm, depends on the size of Σ_C and the number of B states concurrent with each state of C. If the maximum number of B states concurrent with any state of C is x, then an upper bound on the number of states of a full trim unrolling is $|\Sigma_C|^{|S_B|^x}$.

The algorithms given here are intended to be simple to present and prove; certain optimizations are possible. However, it is not likely that we can do significantly better on the quotient problem. A method of deciding whether a *safe* FSM exists can be used (via a polynomial transformation) to solve the NFA-INEQUIVALENCE problem, which is PSPACE-complete [7].

Our quotient solution method was developed for the purpose of computing specifications for different kinds of *protocol converters*. A protocol converter is device that enables implementations of different communication protocols to achieve some degree of useful interaction. The problem of designing a protocol converter for given formally-specified protocol implementations and desired service is an instance of the quotient problem [6]. The method described here has been successfully applied to some example protocol conversion problems [4, 5]. In those examples, the size of the output FSM was similar to that of the input machines, and the expensive third step of the algorithm was never necessary.

It should be noted that a solution produced by the method described here is not necessarily the "weakest" quotient FSM (in the sense that if D is a solution then D *sat* C). This is because C is deterministic and maximal, and therefore may have more events enabled at certain states than are necessary. However, in general there is no unique "weakest" solution to a quotient problem of this sort.

Appendix

Proof of (7). Given FSMs M, N and D, such that M *sat* N, we have to prove conditions **S0** and **S1** for $M \circ D$ and $N \circ D$. To prove **S0**, our obligation is (omitting the ranges for brevity):

$$\begin{aligned}&\delta^*(i,t,\langle m\,d\rangle) \wedge \mathit{ext}.\langle m\,d\rangle \;\Rightarrow\\ &\quad(\exists\,\langle n\,d\rangle :: \delta^*(i,t,\langle n\,d\rangle) \;\wedge\; ((\mathit{ext}.\langle n\,d\rangle \wedge \mathit{out}.\langle n\,d\rangle \subseteq \mathit{out}.\langle m\,d\rangle) \vee \mathit{div}.\langle n\,d\rangle))\end{aligned}$$

We observe

$$\begin{aligned}
&\quad \delta^*(i,t,\langle m\,d\rangle) \wedge \mathit{ext}.\langle m\,d\rangle\\
=&\quad\{\text{ composition properties (5) and (2) }\}\\
&\quad(\exists\,u,v :: \mathit{zip}(u,v,t) \wedge \delta^*_M(i,u,m) \wedge \delta^*_D(i,v,d)) \wedge\\
&\qquad\quad \mathit{ext}.m \wedge \mathit{ext}.d \wedge (\mathit{out}.m \cap \mathit{out}.d = \emptyset)\\
=&\quad\{\text{ predicate calculus }\}\\
&\quad(\exists\,u,v :: \delta^*_M(i,u,m) \wedge \mathit{ext}.m \wedge \mathit{zip}(u,v,t) \wedge\\
&\qquad\quad \delta^*_D(i,v,d) \wedge \mathit{ext}.d \wedge (\mathit{out}.m \cap \mathit{out}.d = \emptyset))\\
\Rightarrow&\quad\{\text{ hypothesis: } \mathbf{S0} \text{ for } M \text{ and } N \}\\
&\quad(\exists\,u,v :: (\exists\,n :: \delta^*_N(i,u,n) \wedge ((\mathit{ext}.n \wedge \mathit{out}.n \subseteq \mathit{out}.m) \vee \mathit{div}.n)) \wedge\\
&\qquad\quad \mathit{zip}(u,v,t) \wedge \delta^*_D(i,v,d) \wedge \mathit{ext}.d \wedge \mathit{ext}.m \wedge (\mathit{out}.m \cap \mathit{out}.d = \emptyset))\\
\Rightarrow&\quad\{\text{ predicate calculus }\}\\
&\quad(\exists\,n :: (\exists\,u,v :: \mathit{zip}(u,v,t) \wedge \delta^*_N(i,u,n) \wedge \delta^*_D(i,v,d)) \;\wedge\\
&\qquad\quad (\mathit{ext}.n \wedge \mathit{out}.n \subseteq \mathit{out}.m \wedge \mathit{ext}.d \wedge \mathit{out}.m \cap \mathit{out}.d = \emptyset)) \;\vee\; \mathit{div}.n)
\end{aligned}$$

At this point, we make use of the following "monotonicity" property of the symmetric set difference:

$$\mathit{out}.n \subseteq \mathit{out}.m \;\Rightarrow\; \mathit{out}.n \div \mathit{out}.d \subseteq \mathit{out}.m \div \mathit{out}.d$$

Now by Property (1), we have then

$$\mathit{out}.n \subseteq \mathit{out}.m \;\Rightarrow\; \mathit{out}.\langle n\,d\rangle \subseteq \mathit{out}.\langle m\,d\rangle$$

Also, $\mathit{out}.n \subseteq \mathit{out}.m$ and $\mathit{out}.m \cap \mathit{out}.d = \emptyset$ imply $\mathit{out}.n \cap \mathit{out}.d = \emptyset$. Thus the last line above implies

$$\begin{array}{ll}
& (\exists\, n : (\exists\, u,v : \mathit{zip}(u,v,t) : \delta^*_N(i,u,n) \wedge \delta^*_D(i,v,d)) \;: \\
& \qquad (\mathit{ext}.n \wedge \mathit{ext}.d \wedge \mathit{out}.n \cap \mathit{out}.d = \emptyset \wedge \\
& \qquad \mathit{out}.\langle n\,d\rangle \subseteq \mathit{out}.\langle m\,d\rangle) \vee \mathit{div}.n) \\
\Rightarrow & \quad \{ \text{ Properties (5) and (2) } \} \\
& (\exists\, n : \delta^*(i,t,\langle n\,d\rangle) \;: \\
& \qquad ((\mathit{ext}.\langle n\,d\rangle \wedge \mathit{out}.\langle n\,d\rangle \subseteq \mathit{out}.\langle m\,d\rangle) \vee \mathit{div}.\langle n\,d\rangle))
\end{array}$$

and thus we have established **S0**. For **S1**, our obligation is to show, for any trace t and state $\langle m\,d\rangle$,

$$\delta^*(i,t,\langle m\,d\rangle) \wedge \mathit{div}.\langle m\,d\rangle \;\Rightarrow\; (\exists\, \langle n\,d\rangle : \delta^*(i,t,\langle n\,d\rangle) : \mathit{div}.\langle n\,d\rangle)$$

To this end we observe, for any t and $\langle m\,d\rangle$,

$$\begin{array}{ll}
& \delta^*(i,t,\langle m\,d\rangle) \wedge \mathit{div}.\langle m\,d\rangle \\
= & \quad \{ \text{ properties (5) and (3) } \} \\
& (\exists\, u,v : \mathit{zip}(u,v,t) : \delta^*_M(i,u,m) \wedge \delta^*_D(i,v,d)) \;\wedge \\
& \qquad (\mathit{div}.m \vee \mathit{div}.d \;\vee \\
& \qquad (\exists\, w : w \neq \diamond : \delta^*_M(m,w,m) \wedge \delta^*_D(d,w,d))) \\
= & \quad \{ \text{ predicate calculus } \} \\
& (\exists\, u,v : \mathit{zip}(u,v,t) : \delta^*_M(i,u,m) \wedge \delta^*_D(i,v,d) \wedge \mathit{div}.m) \;\vee \\
& (\exists\, u,v : \mathit{zip}(u,v,t) : \delta^*_M(i,u,m) \wedge \delta^*_D(i,v,d) \wedge \mathit{div}.d) \;\vee \\
& (\exists\, u,v,w : \mathit{zip}(u,v,t) : \delta^*_M(i,u,m) \wedge \delta^*_D(i,v,d) \;\wedge \\
& \qquad w \neq \diamond \wedge \delta^*_M(m,w,m) \wedge \delta^*_D(d,w,d))
\end{array}$$

We consider each disjunct of the last formula as a separate case. For the first, we have:

$$\begin{array}{ll}
& (\exists\, u,v : \mathit{zip}(u,v,t) : \delta^*_M(i,u,m) \wedge \delta^*_D(i,v,d) \wedge \mathit{div}.m) \\
\Rightarrow & \quad \{ \text{ \textbf{S1} for } M \text{ and } N \} \\
& (\exists\, u,v : \mathit{zip}(u,v,t) : (\exists\, n : \delta^*_N(i,u,n) : \mathit{div}.n) \wedge \delta^*_D(i,v,d)) \\
= & \quad \{ \text{ predicate calculus } \} \\
& (\exists\, n :: (\exists\, u,v : \mathit{zip}(u,v,t) : \delta^*_N(i,u,n) \wedge \delta^*_D(i,v,d)) \wedge \mathit{div}.n) \\
= & \quad \{ \text{ property (5) } \} \\
& (\exists\, n :: \delta^*(i,t,\langle n\,d\rangle) \wedge \mathit{div}.n) \\
\Rightarrow & \quad \{ \text{ property (3), predicate calculus } \} \\
& (\exists\, \langle n\,d\rangle : \delta^*(i,t,\langle n\,d\rangle) : \mathit{div}.\langle n\,d\rangle)
\end{array}$$

For the second disjunct, we observe

$$\begin{array}{ll}
& (\exists\, u,v : \mathit{zip}(u,v,t) : \delta^*_M(i,u,m) \wedge \delta^*_D(i,v,d) \wedge \mathit{div}.d) \\
\Rightarrow & \quad \{ \text{ property (4); def'n trace predicate } \} \\
& (\exists\, u,v : \mathit{zip}(u,v,t) : (\exists\, n :: \delta^*_N(i,u,n)) \wedge \delta^*_D(i,v,d) \wedge \mathit{div}.d) \\
= & \quad \{ \text{ predicate calculus } \} \\
& (\exists\, n :: (\exists\, u,v : \mathit{zip}(u,v,t) : \delta^*_N(i,u,n) \wedge \delta^*_D(i,v,d)) \wedge \mathit{div}.d) \\
= & \quad \{ \text{ property (5) } \}
\end{array}$$

$$
\begin{array}{ll}
 & (\exists\, n : \delta^*(i,t,\langle n\, d\rangle) : div.d) \\
\Rightarrow & \quad \{ \text{ property (3), predicate calculus } \} \\
 & (\exists\, \langle n\, d\rangle : \delta^*(i,t,\langle n\, d\rangle) : div.\langle n\, d\rangle)
\end{array}
$$

In the case of the third disjunct we have a trace $w \neq \diamond$ such that

$$zip(u,v,t) \;\wedge\; \delta^*_M(i,u,m) \;\wedge\; \delta^*_D(i,v,d) \;\wedge\; \delta^*_M(m,w,m) \;\wedge\; \delta^*_D(d,w,d)$$

Now, by the definition of the trace relation, for every $k \geq 0$ we have $\delta^*_M(i,uw^k,m)$ and $\delta^*_D(i,vw^k,d)$. Because $M\ sat\ N$, we have also

$$(\forall\, k : 0 \leq k : (\exists\, n :: \delta^*_N(i,uw^k,n)))$$

However, the number of distinct states of N is finite, so there exist a particular state n_0, and natural numbers k_0 and j, such that

$$\delta^*_N(i,uw^{k_0},n_0) \;\wedge\; \delta^*_N(n_0,w^j,n_0)$$

But we also have for d, k_0 and j,

$$\delta^*_D(i,vw^{k_0},d) \;\wedge\; \delta^*_D(d,w^j,d)$$

Also, we have $zip(u,v,t)$ and trivially $zip(w^{k_0},w^{k_0},\diamond)$; from the definition of zip have then

$$zip(uw^{k_0},vw^{k_0},t)$$

Collecting the above, we have

$$
\begin{array}{ll}
 & zip(uw^{k_0},vw^{k_0},t) \;\wedge\; \delta^*_N(i,uw^{k_0},n_0) \;\wedge\; \delta^*_D(i,vw^{k_0},d) \;\wedge \\
 & \qquad \delta^*_D(d,w^j,d) \;\wedge\; \delta^*_N(n_0,w^j,n_0) \\
\Rightarrow & \quad \{ \text{ pred. calc., (5) and (3) } \} \\
 & \delta^*(i,t,\langle n_0\, d\rangle) \;\wedge\; div.\langle n_0\, d\rangle \\
\Rightarrow & \quad \{ \text{ predicate calculus } \} \\
 & (\exists\, \langle n\, d\rangle :: \delta^*(i,t,\langle n\, d\rangle) \;\wedge\; div.\langle n\, d\rangle)
\end{array}
$$

This completes the case analysis and the proof of Property (7). □

Proof of Proposition 2. Restating the proposition, we are required to show that, for given semideterministic and nondivergent A and B, the conjunction of

(i) D is safe

(ii) D is progressive

(iii) $B \circ D$ is nondivergent.

is equivalent to the conjunction of (restating conditions **S0** and **S1** for $B \circ D$ and A)

(iv) $(\forall \langle b\, d\rangle, w : \delta^*(i, w, \langle b\, d\rangle) \wedge \mathit{ext}.\langle b\, d\rangle :$

$$(\exists\, a : \delta^*(i, w, a) : ((\mathit{ext}.a \wedge \mathit{out}.a \subseteq \mathit{out}.\langle b\, d\rangle) \vee \mathit{div}.a)))$$

(v) $(\forall \langle b\, d\rangle, w : \delta^*(i, w, \langle b\, d\rangle) \wedge \mathit{div}.\langle b\, d\rangle : (\exists\, a :: \delta^*(i, w, a) \wedge \mathit{div}.a))$

We prove the following:

(a0) (iii) $\equiv$ (v)

(a1) (iii) $\wedge \neg$ (i) $\Rightarrow \neg$ (iv)

(a2) (i) $\wedge \neg$ (ii) $\Rightarrow \neg$ (iv)

(a3) (i) $\wedge$ (ii) $\Rightarrow$ (iv)

from which, by predicate caculus, we obtain

$$\text{(i)} \wedge \text{(ii)} \wedge \text{(iii)} \equiv \text{(iv)} \wedge \text{(v)}$$

Proof of (a0). Beginning with (v), we calculate:

$(\forall \langle b\, d\rangle, w : \delta^*(i, w, \langle b\, d\rangle) \wedge \mathit{div}.\langle b\, d\rangle : (\exists\, a :: \delta^*(i, w, a) \wedge \mathit{div}.a))$
$=$ { hypothesis: A is nondivergent }
$(\forall \langle b\, d\rangle, w : \delta^*(i, w, \langle b\, d\rangle) \wedge \mathit{div}.\langle b\, d\rangle : \mathbf{false})$
$=$ { predicate calculus }
$\neg(\exists \langle b\, d\rangle, w :: \delta^*(i, w, \langle b\, d\rangle) \wedge \mathit{div}.\langle b\, d\rangle)$
$=$ { definition }
$B \circ D$ is nondivergent

□

Proof of (a1). Assume $B \circ D$ is nondivergent and D is not safe, i.e. there exists a trace u such that $D(u)$ and $\neg \mathit{safe}.u$. We shall prove the negation of (iv). By the definitions of $D(u)$ and *safe*, there exist d such that $\delta^*_D(i, u, d)$, and traces w and v, and state b, such that $\mathit{zip}(u, v, w)$ and $\delta^*_B(i, v, b)$, while $(\forall\, a :: \neg\delta^*_A(i, w, a))$. From these observations, we calculate:

$\mathit{zip}(u, v, w) \wedge \delta^*_D(i, u, d) \wedge \delta^*_B(i, v, b)$
$\Rightarrow$ { (5) }
$\delta^*_{B\circ D}(i, w, \langle b\, d\rangle)$
$\Rightarrow$ { predicate calculus and (0) }
$(\exists \langle b\, d\rangle : \delta^*_{B\circ D}(i, w, \langle b\, d\rangle) : \mathit{ext}.\langle b\, d\rangle \vee \mathit{div}.\langle b\, d\rangle)$
$=$ { hypothesis: $B \circ D$ is nondivergent }
$(\exists \langle b\, d\rangle : \delta^*_{B\circ D}(i, w, \langle b\, d\rangle) : \mathit{ext}.\langle b\, d\rangle)$
$=$ { hypothesis; predicate calculus }
$(\exists \langle b\, d\rangle : \delta^*_{B\circ D}(i, w, \langle b\, d\rangle) \wedge \mathit{ext}.\langle b\, d\rangle : \forall\, a :: \neg\delta^*_A(i, w, a)))$
$\Rightarrow$ { predicate calculus }

$$
\begin{array}{ll}
 & (\exists\, \langle b\, d\rangle : \delta^*_{BoD}(i, w, \langle b\, d\rangle) \wedge ext.\langle b\, d\rangle : \\
 & \qquad (\forall\, a : \delta^*_A(i, w, a) : \neg((ext.a \wedge out.a \subseteq out.\langle b\, d\rangle) \vee div.a))\,) \\
= & \quad \{ \text{ predicate calculus } \} \\
 & \neg(\forall\, \langle b\, d\rangle : \delta^*_{BoD}(i, w, \langle b\, d\rangle) \wedge ext.\langle b\, d\rangle : \\
 & \qquad (\exists\, a : \delta^*_A(i, w, a) : ((ext.a \wedge out.a \subseteq out.\langle b\, d\rangle) \vee div.a))\,)
\end{array}
$$

which is the negation of (iv). □

Proof of (a2). Assuming D is not progressive, we prove the negation of (iv). By definitions of progressive, T, and *next*, there exists a trace w, and states a and $\langle b\, d\rangle$ such that $ext.\langle b\, d\rangle$, and $\delta^*(i, w, \langle b\, d\rangle)$ and $\xi^*_A(i, w, a)$, but there is no a' such that

$$\mu(a, a') \wedge ext.a' \wedge out.a' \subseteq out.\langle b\, d\rangle$$

Because A is semi-deterministic, it follows that there is no a such that $\delta^*_A(i, w, a)$ and $ext.a'$ and $out.a' \subseteq out.\langle b\, d\rangle$. That is, for this w and $\langle b\, d\rangle$, we have

$$\delta^*(i, w, \langle b\, d\rangle) \wedge ext.\langle b\, d\rangle \wedge \neg(\exists\, a : \delta^*_A(i, w, a) : ext.a \wedge out.a \subseteq out.\langle b\, d\rangle)$$

Because A is nondivergent, this implies the negation of (iv). □

Proof of (3). Assume D is safe and progressive. We prove (iv) by assuming $\delta^*(i, w, \langle b\, d\rangle)$ and $ext.\langle b\, d\rangle$ and showing the existence of a such that

$$\delta^*_A(i, w, a) \wedge ext.a \wedge out.a \subseteq out.\langle b\, d\rangle$$

Note that, because D is progressive, d is progressive. By the safety of D, there exists a such that $\xi^*_A(i, w, a)$ in A. We observe:

$$
\begin{array}{ll}
 & \delta^*(i, w, \langle b\, d\rangle) \wedge \xi^*_A(i, w, a) \wedge ext.\langle b\, d\rangle \\
\Rightarrow & \quad \{ \text{ Properties (11) and (2) } \} \\
 & T.(a, b, d) \wedge out.b \cap out.d = \emptyset \\
= & \quad \{ \; d \text{ is progressive: } next.(d, out.d) \; \} \\
 & T.(a, b, d) \wedge out.b \cap out.d = \emptyset \wedge next.(d, out.d) \\
\Rightarrow & \quad \{ \text{ definition of } next \; \} \\
 & (out.b \cap out.d \neq \emptyset \vee prog.(a, (out.b \div out.d)) \wedge out.b \cap out.d = \emptyset \\
= & \quad \{ \text{ pred. calc; (1) } \} \\
 & prog.(a, out.\langle b\, d\rangle) \\
\Rightarrow & \quad \{ \text{ def'n } prog \; \} \\
 & (\exists\, a' : \mu(a, a') : ext.a' \wedge out.a' \subseteq out.\langle b\, d\rangle) \\
\Rightarrow & \quad \{ \text{ hypothesis } \xi^*_A(i, w, a);\ \xi^*(i, w, a) \wedge \mu(a, a') \Rightarrow \delta^*(i, w, a') \; \} \\
 & (\exists\, a' : \delta^*_A(i, w, a') : ext.a' \wedge out.a' \subseteq out.\langle b\, d\rangle)
\end{array}
$$

Thus we have established (iv), and (3) is proved. □

This completes the proof of Proposition 2. □

Proof of Proposition 4. Under the assumption *safe.u*, we show $ss.(h.u, e) \Rightarrow$ *safe.ue*, and $\neg ss.(h.u).e \Rightarrow \neg$*safe.ue*. For the first, assume $ss.(h.u, e)$. We have to show *safe.ue*; that is, we must establish:

$$(\forall\, v, w : zip(ue, v, w) \wedge B(v) : A(w))$$

Let v, w be any traces (in Σ_B^* and Σ_A^* respectively) such that $zip(ue, v, w)$ and $B(v)$, i.e. there exists b such that $\delta_B^*(i, v, b)$). We shall show $A(w)$.

By the definition of *zip*, there exist v_0, v_1 and w_0, w_1 such that $v = v_0 v_1$, $w = w_0 w_1$ and $zip(u, v_0, w_0)$ and $zip(e, v_1, w_1)$. Because B is semi-deterministic, there exist b_0 and b_1 such that $\xi_B^*(i, v_0, b_0)$ and $\xi_B^*(b_0, v_1, b_1)$, and $\mu(b_1, b)$. We have:

$$\begin{array}{ll}
& zip(u, v_0, w_0) \wedge \xi^*(i, v_0, b_0) \\
\Rightarrow & \quad \{\ \xi^* \Rightarrow \delta^*;\ \text{definition of } B(t)\ \} \\
& zip(u, v_0, w_0) \wedge B(v_0) \\
\Rightarrow & \quad \{\ \text{hypothesis and def'n } safe.u\ \} \\
& A(w_0) \\
\Rightarrow & \quad \{\ \text{property (9)}\ \} \\
& (\exists\, a :: \xi_A^*(i, w_0, a))
\end{array}$$

Collecting selected terms, we have

$$\begin{array}{ll}
& zip(u, v_0, w_0) \wedge \xi_B^*(i, v_0, b_0) \wedge \xi_A^*(i, w_0, a_0) \\
& \qquad \wedge\ zip(e, v_1, w_1) \wedge \xi_B^*(b_0, v_1, b_1) \\
\Rightarrow & \quad \{\ \text{definition } h.u\ \} \\
& (a_0, b_0) \in h.u \wedge zip(e, v_1, w_1) \wedge \xi_B^*(b_0, v_1, b_1) \wedge \xi_A^*(i, w_0, a_0) \\
\Rightarrow & \quad \{\ \text{hypothesis and def'n } ss.(h.u, e)\ \} \\
& (\exists\, a_1 :: \xi_A^*(a_0, w_1, a_1)) \wedge \xi_A^*(i, w_0, a_0) \\
\Rightarrow & \quad \{\ \text{def'n } \xi^*;\ \xi^* \Rightarrow \delta^*\ \} \\
& (\exists\, a_1 :: \delta^*(i, w_0 w_1, a_1)) \\
= & \quad \{\ w = w_0 w_1;\ \text{definition of } A(t)\ \} \\
& A(w)
\end{array}$$

Thus we have established $ss.(h.u, e) \Rightarrow$ *safe.ue*

Now assume $\neg ss.(h.u, e)$; by the definition of *ss*, there exist a, b, v, b' and w such that

$$(a, b) \in h.u \wedge zip(e, v, w) \wedge \xi_B^*(b, v, b') \wedge \neg(\exists\, a' :: \xi_A^*(a, w, a'))$$

We shall establish $\neg$*safe.ue*. We observe

$$\begin{array}{ll}
& (a, b) \in h.u \wedge zip(e, v, w) \wedge \xi_B^*(b, v, b') \wedge \neg(\exists\, a' :: \xi_A^*(a, w, a')) \\
\Rightarrow & \quad \{\ \text{definition of } h.u\ \} \\
& (\exists\, v_0, w_0 :: zip(u, v_0, w_0) \wedge \xi_A^*(i, w_0, a) \wedge \xi_B^*(i, v_0, b)) \\
& \qquad \wedge\ zip(e, v, w) \wedge \xi_B^*(b, v, b') \wedge \neg(\exists\, a' :: \xi_A^*(a, w, a'))
\end{array}$$

$\Rightarrow$ { def'n of zip; property (10) }

$(\exists\, v_0, w_0 :: zip(ue, v_0 v, w_0 w) \wedge \xi^*_B(i, v_0 v, b') \wedge \neg(\exists\, a' :: \xi^*_A(i, w_0 w, a')))$

$\Rightarrow$ { predicate calculus }

$(\exists\, v', w' :: zip(ue, v', w') \wedge \xi^*_B(i, v', b') \wedge \neg(\exists\, a' :: \xi^*_A(i, w', a')))$

$\Rightarrow$ { properties of ξ^* }

$(\exists\, v', w' :: zip(ue, v', w') \wedge \delta^*_B(i, v', b') \wedge \neg(\exists\, a' :: \delta^*_A(i, w', a')))$

$\Rightarrow$ { def'n $B(v')$, $A(w')$ }

$(\exists\, v', w' : zip(ue, v', w') \wedge B(v') : \neg A(w'))$

$=$ { predicate calculus and definition of $safe$ }

$\neg safe.ue$

Thus we have shown $\neg ss.(h.u, e) \Rightarrow \neg safe.e$; this completes the proof. □

Proof of Proposition 7

Proof of (20). We show $T.(a, b, c) \Rightarrow T.(a, b, d)$ for any a, b and d satisfying the conditions of the lemma.

$T.(a, b, c)$

$=$ { Proposition 3 }

$(\exists\, u', b' : (ab') \in h.u' : \mu(b', b) \wedge \delta^*(i, u', c))$

$\Rightarrow$ { Proposition 5 }

$(\exists\, u', b' : (ab') \in tag.c : \mu(b', b) \wedge \delta^*(i, u', c))$

$\Rightarrow$ { hypothesis $\delta^*_C(i, u, c)$ and Proposition 5 }

$(\exists\, b' : (ab') \in h.u : \mu(b', b))$

$=$ { hypothesis $\delta^*_D(i, u, d)$ }

$(\exists\, b' : (ab') \in h.u : \mu(b', b) \wedge \delta^*(i, u, d))$

$\Rightarrow$ { predicate calculus }

$(\exists\, u, b' : (ab') \in h.u : \mu(b', b)) \wedge \delta^*(i, u, d))$

$=$ { Proposition 3 }

$T.(a, b, d)$

□

Proof of (21). We observe for any $e \in \Sigma$:

$e \in out.d$

$=$ { def'n out }

$(\exists\, d' :: \delta_D(d, e, d'))$

$\Rightarrow$ { hypothesis $\delta^*_D(i, u, d)$ and def'n δ^* }

$(\exists\, d' :: \delta^*_D(i, ue, d'))$

$\Rightarrow$ { C_0 is maximal; def'n maximal }

$(\exists\, c_0 :: \delta^*_0(i, ue, c_0))$

$\Rightarrow$ { def'n δ^* }

$(\exists\, c_0, c_1 :: \delta^*_0(i, u, c_1) \wedge \delta_0(c_1, e, c_0))$

$=$ { hypothesis $\delta^*_0(i, u, c)$ and C is deterministic }

$$
\begin{array}{ll}
 & (\exists\, c_0, c_1 : c = c_1 : \delta_0^*(i,u,c_1) \wedge \delta_0(c_1,e,c_0)) \\
\Rightarrow & \quad \{ \text{ predicate calculus } \} \\
 & (\exists\, c_0 :: \delta_0(c,e,c_0)) \\
= & \quad \{ \text{ definition } \mathit{out} \ \} \\
 & e \in \mathit{out}.c
\end{array}
$$

□

Proof of (22). We observe, for any $G \subseteq \Sigma$:

$$
\begin{array}{ll}
 & \mathit{next}.(d,G) \\
= & \quad \{ \text{ definition of } \mathit{next} \ \} \\
 & (\forall\, a,b : T.(a,b,d) : \mathit{out}.b \cap G \neq \emptyset \vee \mathit{prog}.(a,(\mathit{out}.b \div G))) \\
\Rightarrow & \quad \{ \text{ (20); predicate calculus } \} \\
 & (\forall\, a,b : T.(a,b,c) : \mathit{out}.b \cap G \neq \emptyset \vee \mathit{prog}.(a,(\mathit{out}.b \div G))) \\
= & \quad \{ \text{ definition } \mathit{next} \ \} \\
 & \mathit{next}.(c,G)
\end{array}
$$

□

Proof of (23). We observe:

$$
\begin{array}{ll}
 & d \text{ is progressive} \\
= & \quad \{ \text{ def'n progressive } \} \\
 & \mathit{next}.(d, \mathit{out}.d) \\
\Rightarrow & \quad \{ \text{ properties (21) and (13) } \} \\
 & \mathit{next}.(d, \mathit{out}.c) \\
\Rightarrow & \quad \{ \text{ property (22) with } G := \mathit{out}.c \ \} \\
 & \mathit{next}.(c, \mathit{out}.c) \\
= & \quad \{ \text{ def'n progressive } \} \\
 & c \text{ is progressive}
\end{array}
$$

□

Proof of Proposition 10. We show how to transform a given unrolling into a trim unrolling by removing certain states and transitions from its tree-FSM, while preserving progressiveness and acceptability. (In this proof, and in the the rest of this section, "removing a state from an unrolling" is understood to entail adjustment the state set and transition relation of the tree-FSM as well as the labeling function, so that the state's incoming edge (if any) and the subtree rooted at the state are removed.) By removing all states with a dead ancestor, every dead state becomes a leaf. Every new leaf is dead, so if the original tree was acceptable, the modified tree is as well.

Back edges can be removed as follows. Let m be the nearest ancestor of (i.e., distinct from, and on the root path to) m' such that $L.m = L.m'$. Thus the incoming edge to m' is a back edge. Consider the tree obtained by replacing the subtree rooted at m with the subtree rooted at m. We observe:

- Every state of the modified tree is a state of the original, and retains its same label.
- Every state in the modified tree has the same *out* set that it had in the original tree, so the modified tree is progressive if the original was.
- no new leaves are created, so acceptability is preserved.
- The number of back edges in the new tree is strictly less than in the original tree, because at least one has been removed.

This operation can be repeated until no back edges remain. After applying these two modifications, the resulting unrolling is trim. □

Proof of Proposition 13. Let D be any deterministic solution. We first prove two Lemmas.

Lemma. There exists a progressive and acceptable unrolling from each reachable state of D.

Proof. For every reachable state d of D, the full trim unrolling from d, as constructed by Algorithm 1, is acceptable and progressive. The construction includes all transitions except those that form back edges or originate in dead states. Because $B \circ D$ is nondivergent, by Proposition 9 no unrolling from any state of D contains a back edge; therefore the algorithm rejects only transitions originating in dead states. That is, a state is a leaf in the full trim unrolling from d only if it is dead or terminal, and hence the unrolling is acceptable. Each non-dead state m satisfies $out.m = out.(L_0.m)$, and so is progressive because $L_0.m$ is. □

Lemma. If $\delta^*_D(i,u,d)$ and $\delta^*_{C_p}(i,u,c)$, then there exists a t.p.a. unrolling (M',L') of C_p from c such that for each state m and trace u' of the unrolling there exists d' such that $\delta^*_D(d,u',d')$.

Proof. Let (M,L) be the progressive and acceptable unrolling from d whose existence is asserted by the preceding lemma. We first define a progressive and acceptable unrolling (M,L'') of C_p from c in terms of (M,L). Define $L''_0.r = c$, and define $L''_0.m$ for other states of M inductively: for each $\delta_M(m,e,m')$, let $L''_0.m'$ be the unique state c' such that $\delta_{C_p}(L''_0.m,e,c')$ (such a c' exists because $e \in out.(L_0.m)$, and by Proposition 7, $out.(L_0.m) \subseteq out.(L''_0.m)$.) Let the head states of B that are concurrent with d be $b_1,\ldots,b_K$. Again by Proposition 7, every state of B concurrent with c is also concurrent with d. Letting $b_1,\ldots,b_{K'}$ be the head states concurrent with c for some $K' \leq K$, the states concurrent with d can be arranged as $b_1,\ldots,b_{K'},\ldots,b_K$. We then put $L''_j.m = L_j.m$ for each j, $0 < j \leq K'$; because the structure of M is not changed, this definition satisfies the requirements of an unrolling. A state m of is dead in (M,L'')

if it is dead in (M, L), and (again by Proposition 7) progressive in (M, L'') if it is progressive in (M, L). Thus, (M, L'') is a progressive and acceptable unrolling from c. Now we can apply the transformation described in the proof of Proposition 10 to obtain a t.p.a. unrolling (M', L'). The modifications of that procedure preserve the property that $\delta_M(m, e, m')$ in M implies the existence of an edge $\delta_D(L''_0.m, e, L''_0.m')$ in D. A simple inductive argument shows that for any m and u', if $\delta^*_{M'}(r, u', m)$ then there exists d' such that $\delta^*_D(d, u', d')$. □

Now we are ready to prove the proposition. Given a solution D, we construct a reasonable set of unrollings of C_p using an inductive procedure. For the initial states, we have $\delta^*_{C_p}(i, \diamond, i)$, and and $\delta^*_D(i, \diamond, i)$. By the second Lemma above we can construct a t.p.a. unrolling (M, L) from the initial state of C_p. For each leaf m of this unrolling we have $\delta^*_M(r, u, m)$ and $\delta^*_{C_p}(i, u, L_0.m)$ for some u. By the second lemma there exists d' such that $\delta^*_D(i, u, d')$. Thus $L_0.m$ and d' satisfy the conditions of the hypothesis of the second Lemma, and we can again construct a t.p.a. unrolling from $L_0.m$ and add it to the set. We similarly construct an unrolling for the other leaves, iterating until for each leaf m of each unrolling in the set, the set contains an unrolling from the corresponding C_p-state. Closure will eventually be achieved because of the finiteness of the state set of C_p. The resulting set satisfies the definition of reasonableness by construction. □

Proof of Proposition 14. By the definition of r-maximality, it suffices to show that any t.p.a. unrolling from c is a sub-unrolling of the full trim unrolling from c. Let (M, L) be any t.p.a. unrolling from c. Let (N, L') be the full trim unrolling from c. We show by induction on trace length that every trace of M is a trace of N. The base case is trivial. For the inductive step, assume $\delta^*_M(r, u, m)$, and $\delta_M(m, e, m')$. By the inductive hypothesis there exists n such that $\delta^*_N(r, u, n)$. Because M's unrolling is trim, m is not dead, and the label of m' is unique on its root path. By the definition of unrolling, there is a corresponding transition $\delta(L_0.m, e, L_0.m')$ in C_p. Now, N is full, so (because m is not dead, and $\delta(m, e, m')$ is not a back edge) there is an n' and a corresponding transition $\delta(n, e, n')$ in N. (Recall that the labels along identical root paths of two unrollings from the same state are identical.) Therefore ue is a trace of N. Thus we have shown that every trace of M is a trace of N; it follows that (M, L) is a sub-unrolling of (N, L'). □

Proof of Proposition 15. Let (N, L') be a t.p.a. unrolling from c in some reasonable set; we shall derive a contradiction from this assumption. We observe that by the definition of unrolling, $L'.r_N = L.r_M$, i.e., the initial states of M and N have the same label, and furthermore because (M, L) is r-maximal we have $out.r_N \subseteq out.r_M$. It follows that if r_M is not progressive then r_N is not, and r_M is dead if and only if r_N is. Finally, if r_M is a leaf then r_N is.

Thus if r_M is nondead and nonprogressive, then r_N is nondead and nonprogressive, which contradicts the assumed progressiveness of (N, L'). If r_M

is a nondead, nonterminal leaf, then r_N is a nondead, nonterminal leaf, which contradicts the assumption that r_N is acceptable. Thus we have shown that no t.p.a. unrolling from c exists, and so no unrolling from c is a member of any reasonable set. □

Proof of Proposition 16. We have to show that (M', L') is trim, and any unrolling from c in a reasonable set is a sub-unrolling of (M', L'). Removing a state preserves trimness, so (M', L') is trim. Now, every trace of M that is not a trace of M' is of the form uu', where u is the unique trace such that $\delta^*_M(r, u, m)$. We shall show that no such trace is a trace of any t.p.a. unrolling from c; it then follows (thanks to r-maximality of (M, L)) that every trace of any t.p.a. unrolling from c is a trace of (M', L'), and hence (M', L') is r-maximal. By the prefix-closure of trace sets, it suffices to show that u is not a trace of any t.p.a. unrolling from c. Let (N, L'') be any t.p.a. unrolling from c. We assume $\delta^*_N(r, u, n)$, and derive a contradiction.

We have $L''.n = L.m$, and hence n is dead or terminal if and only if m is. If m is a leaf, then n is a leaf, by r-maximality of (M, L). Therefore if m is a nondead, nonterminal leaf, then n is a nondead, nonterminal leaf. This contradicts the assumption that N is acceptable.

On the other hand, if m is nondead and nonprogressive, then by the r-maximality of (M, L) we have $out.n \subseteq out.m$, hence (by the definition of progressive and (12)) n is nonprogressive in (N, L), again a contradiction. Thus the assumption $\delta^*_N(r, u, n)$ leads to a contradiction; we conclude that u is not a trace of any t.p.a. unrolling from c, and hence (M', L') is r-maximal. □

Proof of Proposition 18. Because W is optimal, (M, L) is r-maximal; by Propositions 15 and 16, (M', L') is also r-maximal; thus replacing (M, L) by (M', L') preserves optimality. □

Proof of Proposition 19. Observe that m is not the root of M (because (M, L) itself is an unrolling from $L_0.r$). Therefore there exist u and e such that $\delta^*_M(r, ue, m)$, and because m is a leaf, ue is the only trace of M that is not a trace of M'. To show r-maximality of (M', L'), it suffices to show that ue is not a trace of any unrolling from c in any reasonable set. Let (N, L'') be an unrolling from c in some reasonable set W'. We shall derive a contradiction from the assumption that ue is a trace of N.

Assume $\delta^*_N(r, ue, n)$ for some n. Because (M, L) is maximal, and m is a leaf, n is a leaf. By the definition of a reasonable set, W' contains an unrolling from $L''_0.n$. But W contains no unrolling from $L_0.m = L''_0.n$, hence by the definition of an optimal set, no reasonable set contains an unrolling from $L_0.m$; this contradicts the hypothesis that W' is reasonable. Therefore we conclude that ue is not a trace of N, and hence that (M', L') is r-maximal. □

References

[1] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the Association for Computing Machinery*, 31, 1984.

[3] Kenneth L. Calvert. *Protocol Conversion and Quotient Problems.* PhD thesis, University of Texas at Austin, May 1991. Department of Computer Sciences.

[4] Kenneth L. Calvert and Simon S. Lam. Deriving a protocol converter: a top-down method. In *Proceedings of ACM SIGCOMM '89 Symposium, Austin, TX*, September 1989.

[5] Kenneth L. Calvert and Simon S. Lam. Adaptors for protocol conversion. In *Proceedings IEEE INFOCOM '90*, June 1990.

[6] Kenneth L. Calvert and Simon S. Lam. Formal methods for protocol conversion. *IEEE Journal on Selected Areas of Communications*, 8(1), January 1990.

[7] M. Garey and D. Johnson. *Computers and Intractability.* Addison-Wesley, 1979.

[8] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1986.

[9] C. A. R. Hoare and He Jifeng. The weakest prespecification. *Information Processing Letters*, 24(2), January 1987.

[10] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[11] Luming Lai and J. W. Sanders. A weakest-environment calculus for communicating systems. Technical report, Programming Research Group, Oxford University Computing Laboratory, November 1988.

[12] Philip M. Merlin and Gregor von Bochmann. On the construction of submodule specifications and communications protocols. *ACM Transactions on Programming Languages and Systems*, 5(1), Jan 1983.

[13] Robin Milner. *A Calculus of Communicating Systems.* Springer-Verlag, 1980.

[14] Joachim Parrow. Submodule construction as equation solving in ccs. In *LNCS 287: Proceedings of Seventh Conference on Foundations of Software Technology and Theoretical Computer Science.* Springer-Verlag, 1987.

[15] M. W. Shields. Implicit system specification and the interface equation. *The Computer Journal*, 32(5):399–412, 1989.