Compositional Model Checking

E. M. Clarke D. E. Long K. L. McMillan

April 19, 1989
CMU-CS-89-145

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

To appear in the Proceedings of the Fourth IEEE Symposium on Logic in Computer Science, June 4-8, 1989, Asilomar, CA.

Abstract

We describe a method for reducing the complexity of temporal logic model checking in systems composed of many parallel processes. The goal is to check properties of the components of a system and then deduce global properties from these local properties. The main difficulty with this type of approach is that local properties are often not preserved at the global level. We present a general framework for using additional *interface processes* to model the environment for a component. These interface processes are typically much simpler than the full environment of the component. By composing a component with its interface processes and then checking properties of this composition, we can guarantee that these properties will be preserved at the global level. We give two example compositional systems based on the logic CTL*.

This research was sponsored in part by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976, Amendment 20, under Contract Number F33615-87-C-1499, monitored by the: Avionics Laboratory, Air Force Wright Aeronautical Laboratories, Aeronautical Systems Division (AFCS), and Wright-Patterson AFB, Ohio 45433-6543.

This research was also partially supported by NSF grant CCR-87-226-33. The second author holds an NSF graduate fellowship.

The views and conclusions contained in this document are those of the authors and and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, the National Science Foundation or the U.S. Government.

1. Introduction

Temporal logic model checking procedures ([6, 7, 12, 19, 23, 24]) have been successful in finding subtle errors in relatively small finite state systems ([3, 4, 10]), but they all suffer from one apparently unavoidable problem: the state explosion problem. This problem arises in systems composed of many parallel processes. In general, the size of a parallel composition may grow as the product of the sizes of the components. Because of this phenomenon, a program with a relatively small number of processes may have far too many states for a model checking procedure to be directly useful. An obvious technique for avoiding the state explosion problem is to exploit the natural decomposition of a complex parallel program into processes. The goal of this approach is to verify the processes individually and then piece the results together to conclude that the original program is correct. The main difficulty is knowing when some property of a component process remains true in a parallel composition involving that process. It is easy to come up with examples where a critical property of some process is not preserved when the process is composed with another process. A similar technique may be used when the program is constructed in a modular or hierarchical fashion. In this case, lower level components may be simplified by hiding details that do not need to be visible externally and merging those states that become indistinguishable. If the original program is reconstructed from the simplified components, then the number of states will, in general, be much smaller and the program can be checked for correctness more easily. The problem this time is ensuring that the simplified program satisfies the same logical properties as the original program.

There have been a number of other papers on compositional techniques for reasoning about systems of processes. Milner's CCS calculus [20] is certainly compositional in nature, but it is only suitable for showing equivalence between processes and does not handle liveness properties. Barringer [1] has written several papers that show how to give a compositional temporal semantics for a parallel programming language like CSP, but it is not clear how such a semantics can be used in developing a compositional model checking algorithm. Pnueli [22] has developed a compositional proof system for temporal logic that is based on the assume-guarantee paradigm. The primitive formulas in his logic are triples of the form $\langle\varphi\rangle P\langle\psi\rangle$, where φ and ψ are temporal formulas and P is a process. A formula is true if P is guaranteed to behave according to ψ in any execution in which the environment behaves according to φ. One problem with this approach is the potential difficulty in expressing the necessary assumptions, since this can amount to giving the complete specification for an automaton in temporal logic. Josko [18] has developed a model checking algorithm in which the environment is modeled by a temporal logic formula, but the complexity of his procedure is exponential in worst case. Mishra and Clarke [21] have investigated a model checking algorithm for asynchronous circuits that can exploit the hierarchical structure of the circuit, but their approach is restricted to a particular fragment of the logic CTL and is not as general as the one presented here.

In the present paper we use a different, although equally natural approach; we model the environment of a process by another process called an interface process. A rule of inference called the interface rule provides the basis for our compositional model checking technique. The interface rule allows us to deduce properties of a composition by checking properties of the individual processes. The complexity of showing that φ holds for $P_1 \,||\, P_2$ by using this technique is roughly the the same as the complexity of computing the parallel composition of P_1 and the interface process for P_2. We present an algorithm for constructing the interface processes from P_1 and P_2. We also give a general framework for the interface rule that is independent of any particular process model or logic. Within this framework we state four simple conditions that must be true of a process model and an associated logic in order for the inference rule to hold. These conditions can be easily checked to show that a new logic satisfies the inference rule.

We give two examples of compositional systems for which the inference rule is valid. Both systems use a branching time logic based on the temporal logic CTL* [11]. The first example uses an asynchronous process model and a notion of composition that is similar to the one used in theoretical CSP [16]. We define an equivalence relation on processes that allows for finite stuttering along computation paths. This definition is appropriate for reasoning about asynchronous processes since there is no notion of "next system state" in such cases. To illustrate the ideas, we prove the correctness of a tree arbiter. The proof is interesting since it shows how a simple inductive inductive argument can be combined with model checking to show that tree arbiters of arbitrary size are correct. The second system is designed for synchronous digital systems. There is an efficient algorithm ($O(n \log n)$) for deciding equivalence of processes in this model. As an illustration of this model, we consider a simple CPU with decoupled access and execution units.

2. The interface rule

In this section we give the basic rule of inference that is used in the remainder of the paper for obtaining compositional proofs for systems of finite state processes. We show that the rule is sound whenever the set of processes and the logic for reasoning about them satisfy four general and easily checked conditions. The importance of this section is not in the complexity of the soundness proof (which is quite simple) but in the generality of the conditions that we give. Let $\mathcal{P}$ be a set of finite state processes, and assume that we know what it means for two processes P_1 and P_2 to be equivalent ($P_1 \equiv P_2$). Each process will have associated with it a certain set of atomic propositions that is used in distinguishing states and transitions. Σ_P will denote the set associated with process P. The set of propositions associated with the parallel composition of two processes will be the union of the sets associated with the individual processes: $\Sigma_{P_1 \| P_2} = \Sigma_{P_1} \cup \Sigma_{P_2}$. $P \downarrow \Sigma_1$ will be the restriction of P to Σ_1. This process is obtained by hiding all of the symbols in Σ_P that are not in Σ_1; consequently, $\Sigma_{P \downarrow \Sigma_1} = \Sigma_P \cap \Sigma_1$.

Suppose, in addition, that we have a logic $\mathcal{L}$ for reasoning about the processes in $\mathcal{P}$ and that we know what it means for a formula φ to be true of a process P ($P \models \varphi$). Each formula will be constructed from some set of atomic propositions, and we will write $\varphi \in \mathcal{L}(\Sigma)$ to indicate that the propositions used in φ are a subset of Σ. The *interface rule* deals with the parallel composition of two processes P_1 and P_2. The reader might think of the processes as being connected by a set of wires as shown in figure 1, where the wires correspond to symbols in $\Sigma_{P_1} \cap \Sigma_{P_2}$. A_1 and A_2 are interface processes for P_2 and P_1, respectively. Intuitively, A_1 is all that P_2 can observe of P_1 through the wires that connect them. An analogous relationship holds between processes A_2 and P_1. There are two basic inference rules, which we state below:

$$\frac{\begin{array}{c} P_1 \downarrow \Sigma_{P_2} \equiv A_1, \\ \varphi \in \mathcal{L}(\Sigma_{P_2}), \\ A_1 \| P_2 \models \varphi \end{array}}{P_1 \| P_2 \models \varphi} \qquad\qquad \frac{\begin{array}{c} P_2 \downarrow \Sigma_{P_1} \equiv A_2, \\ \psi \in \mathcal{L}(\Sigma_{P_1}), \\ P_1 \| A_2 \models \psi \end{array}}{P_1 \| P_2 \models \psi}$$

If a state minimization procedure is known for $\mathcal{P}$, then A_1 can be obtained by running this algorithm on $P_1 \downarrow \Sigma_{P_2}$. If such a procedure is not available, the rule will still be useful as long as the size of A_1 is significantly less than the size of P_1. A similar comment applies to A_2. The soundness of the interface rule depends on the properties that we enumerate below:

i. Suppose $\Sigma_{P_1} = \Sigma_{P_2}$, then $P_1 \equiv P_2$ implies $\forall \varphi \in \mathcal{L}(\Sigma_{P_1})[P_1 \models \varphi \leftrightarrow P_2 \models \varphi]$.

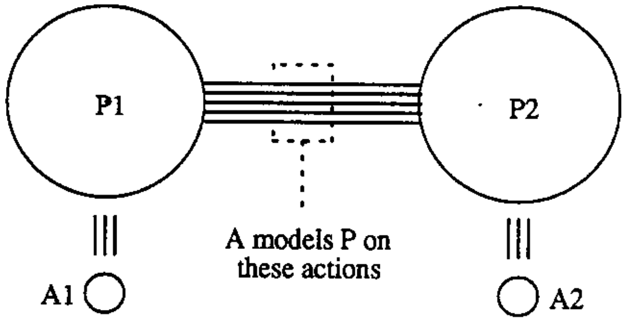

Figure 1: The interface rule

ii. If $P_1 \equiv P_2$ and Q is another process, then $P_1 \,||\, Q \equiv P_2 \,||\, Q$ and $Q \,||\, P_1 \equiv Q \,||\, P_2$.

iii. $(P_1 \,||\, P_2) \downarrow \Sigma_{P_1} \equiv P_1 \,||(P_2 \downarrow \Sigma_{P_1})$ and $(P_1 \,||\, P_2) \downarrow \Sigma_{P_2} \equiv (P_1 \downarrow \Sigma_{P_2}) \,||\, P_2$.

iv. If $\varphi \in \mathcal{L}(\Sigma)$ and $\Sigma \subseteq \Sigma_P$, then $P \models \varphi$ iff $P \downarrow \Sigma \models \varphi$.

It is easy to show that $P_1 \,||\, P_2 \models \varphi$ follows from the three hypotheses of the theorem and the above four properties.

Proof We note that $P_1 \downarrow \Sigma_{P_2} \equiv A_1$, so $A_1 \,||\, P_2 \equiv (P_1 \downarrow \Sigma_{P_2}) \,||\, P_2$. Also $(P_1 \downarrow \Sigma_{P_2}) \,||\, P_2 \equiv (P_1 \,||\, P_2) \downarrow \Sigma_{P_2}$. Hence $A_1 \,||\, P_2 \equiv (P_1 \,||\, P_2) \downarrow \Sigma_{P_2}$ by transitivity of $\equiv$. Now $A_1 \,||\, P_2 \models \varphi$, so $(P_1 \,||\, P_2) \downarrow \Sigma_{P_2} \models \varphi$. It follows that $P_1 \,||\, P_2 \models \varphi$ since $\varphi \in \mathcal{L}(P_2)$. □

A similar technique can be used to show the soundness of simple rules like:

$$\frac{\begin{array}{cc} P_1 \downarrow \Sigma_{P_2} \equiv A_1, & P_2 \downarrow \Sigma_{P_1} \equiv A_2, \\ \varphi \in \mathcal{L}(\Sigma_{P_2}), & \psi \in \mathcal{L}(\Sigma_{P_1}), \\ A_1 \,||\, P_2 \models \varphi, & P_1 \,||\, A_2 \models \psi \end{array}}{P_1 \,||\, P_2 \models \varphi \wedge \psi}$$

We can handle arbitrary boolean combinations of formulas in an analogous manner.

3. An asynchronous process model

We begin by defining a simple model of a communicating system. Processes in this model are asynchronous and communicate using shared synchronization actions. They are represented as a form of transition system [20].

Definition 1 *A finite transition system, or fts, is a quadruple $L = \langle K, q_0, \Sigma, \Delta \rangle$, where:*

i. K is a finite set of states.

ii. $q_0 \in K$ *is an initial state.*

iii. Σ *is a finite set of actions not containing* τ.

iv. $\Delta \subseteq K \times (\Sigma \cup \{\tau\}) \times K$ *is a transition relation.*

Here, τ represents an internal action of the fts. We can then define the operations of composition, hiding, and renaming on fts. The notation $\Delta(p, \sigma)$ is used to indicate $\{q | (p, \sigma, q) \in \Delta\}$ if $\sigma \in \Sigma$. By convention, $\Delta(p, \sigma) = \{p\}$ if $\sigma \notin \Sigma$.

Definition 2 $L'' = L \| L'$ *(the composition of L and L') is given by:*

i. $K'' = K \times K'$.

ii. $q''_0 = (q_0, q'_0)$.

iii. $\Sigma'' = \Sigma \cup \Sigma'$.

iv. $\Delta''((p,p'), \sigma) = \Delta(p, \sigma) \times \Delta'(p', \sigma)$ *for* $\sigma \in \Sigma''$.
$\Delta''((p,p'), \tau) = \Delta(p, \tau) \times \{p'\} \cup \{p\} \times \Delta'(p', \tau)$.

The notion of composition used here is in the style of CSP [16]; an action σ in the first fts synchronizes with an identical action in the second fts.

Definition 3 $L' = L \setminus \sigma$ *(L with* σ *hidden) is defined when* $\sigma \in \Sigma$ *and is given by:*

i. $K' = K$.

ii. $q'_0 = q_0$.

iii. $\Sigma' = \Sigma \setminus \{\sigma\}$.

iv. $\Delta'(p, \nu) = \Delta(p, \nu)$ *if* $\nu \in \Sigma'$.
$\Delta'(p, \tau) = \Delta(p, \tau) \cup \Delta(p, \sigma)$.

Definition 4 $L' = L[\sigma_0 / \sigma_1]$ *(L with* σ_0 *renamed to* σ_1*) is defined when* $\sigma_0 \in \Sigma$ *and is given by:*

i. $K' = K$.

ii. $q'_0 = q_0$.

iii. $\Sigma' = (\Sigma \setminus \{\sigma_0\}) \cup \{\sigma_1\}$.

iv. $\Delta'(p, \nu) = \Delta(p, \nu)$ *if* $\nu \in (\Sigma \setminus \{\sigma_0\}) \cup \{\tau\}$.
$\Delta'(p, \sigma_1) = \Delta(p, \sigma_0)$.

We note that the operations as defined have certain obvious algebraic properties. For example, composition is commutative and associative up to isomorphism.

Associated with an fts is a set of runs through the structure. We use the following definitions and terminology. For a finite set S, the notation S^* denotes the set of finite sequences of elements of S, S^ω denotes the infinite sequences, and $S^\infty = S^* \cup S^\omega$.

Definition 5 *A run from $q \in K$ is a pair (q, π), where $\pi = (p_0, \sigma_0, p_1)(p_1, \sigma_1, p_2)\ldots \in \Delta^\infty$ and $p_0 = q$. A run from the initial state q_0 is often simply called a run.*

We write P^*_q and P^ω_q for the finite and infinite runs from q, and define $P^\infty_q = P^*_q \cup P^\omega_q$.

For the required equivalence between structures, we use the notion of a bisimulation [20]. Formally, for an fts L, we have the following.

Definition 6 *$F : 2^{K\times K} \rightarrow 2^{K\times K}$ is monotonic if $R_0 \subseteq R_1 \subseteq K \times K$ implies $F(R_0) \subseteq F(R_1)$. F is equivalence-preserving if $R \subseteq K \times K$ being an equivalence relation implies $F(R)$ is an equivalence relation.*

At this point we restrict our attention to F which are monotonic.

Definition 7 *R is an F-bisimulation if $R \subseteq F(R)$. We define $p \approx_F q$ iff there is an F-bisimulation R such that $p\ R\ q$.*

We also define a series of approximations to $\approx_F$ as follows.

Definition 8 *Define $\approx_{n,F}$ inductively by:*

i. $\approx_{0,F} = K \times K$.

ii. $\approx_{n+1,F} = F(\approx_{n,F})$.

Define $\approx_{\omega,F} = \bigcap_n \approx_{n,F}$.

We then have the following simple results.

Proposition 1 *$\approx_F$ possesses the following properties:*

i. $\approx_F$ is the largest F-bisimulation under $\subseteq$.

ii. $\approx_F$ is the maximum fixpoint of F.

iii. $\approx_F = \approx_{\omega,F}$.

iv. If F is equivalence-preserving, then $\approx_F$ is an equivalence relation.

The proof is straightforward and is omitted.

4. An asynchronous process logic

We illustrate the development of a compositional system within the framework of branching time temporal logic. The logic we use is essentially the computation tree logic (CTL) of Emerson and Clarke [6]. The process model is that defined in the previous section. Because CTL is state-oriented rather than transition-oriented, we must be able to define the labeling of a state.

We introduce the following notation. $\mathbf{ls}(\rho)$ is the last state of a finite run. $\mathbf{st}(\rho)$ is the sequence of states the run ρ passes through. $\mathbf{ac}(\rho)$ is the sequence of actions (excluding τ actions) performed. $\mathbf{inf}(\rho)$ is the set of states which occur infinitely often. We consider the last state of a finite run as repeating.

Definition 9 *Let* $\rho = (q_0, (q_0, \sigma_0, q_1)(q_1, \sigma_1, q_2)\ldots) \in P^{\infty}_{q_0}$. *Define:*

i. $\mathbf{ls}(\rho) = q_n$ *if* $|\rho| = n$.

ii. $\mathbf{st}(\rho) = q_0 q_1 \cdots$

iii. $\mathbf{ac}(\rho) = (\sigma_0 \sigma_1 \sigma_2 \ldots) \downarrow \Sigma$.

iv. $\mathbf{inf}(\rho) = \{\mathbf{ls}(\rho)\}$ *if ρ is finite.*
$\mathbf{inf}(\rho) = \{q | \mathbf{st}(\rho) \downarrow \{q\} = q^{\omega}\}$ *if ρ is infinite.*

We denote concatenation of runs by juxtaposition. The notation $\rho \prec \theta$ means that the run θ is a (proper) suffix of the run ρ. ρ/θ will be defined when $\rho \preceq \theta$ to be the η such that $\rho = \eta\theta$.

The labeling $\mathcal{L}(\rho)$ of a run will be the set of actions which are performed an odd number of times. We think of each action as toggling a state variable which is associated with that action. The labeling $\mathcal{L}(q)$ of a state is the set of labelings of runs which end at that state. We will restrict our attention to fts which have unique labelings for each of their states (i.e., for which $|\mathcal{L}(q)| = 1$). We will typically write $\mathcal{L}(q)$ for this unique labeling (as opposed to the set containing this unique labelling). Formally, we define the labeling as follows.

Definition 10 *Let* $\rho \in P^{*}_{q_0}$. *Define* $\mathcal{L}(\rho) = \{\sigma | \mathbf{ac}(\rho) \downarrow \{\sigma\} = \sigma^{n}, n \text{ odd}\}$. *Define* $\mathcal{L}(q) = \{\mathcal{L}(\rho) | \rho \in P^{*}_{q_0}$ *and* $\mathbf{ls}(\rho) = q\}$.

Definition 11 *The set of formulas $\mathit{CTL}(\Sigma)$ is defined as the smallest set of state formulas such that:*

i. If $\sigma \in \Sigma$ then σ is a state formula.

ii. If φ is a state formula, so is $\neg\varphi$.

iii. If φ and ψ are state formulas, so is $\varphi \wedge \psi$.

iv. If φ is a path formula, then $\exists\varphi$ is a state formula.

v. If φ and ψ are state formulas then $(\varphi \mathcal{U} \psi)$ is a path formula.

vi. *If φ is a path formula, so is $\neg\varphi$.*

We use the following abbreviations:

$$\begin{array}{rcl@{\qquad\qquad}rcl}
\varphi \vee \psi &\equiv& \neg(\neg\varphi \wedge \neg\psi) & \varphi \rightarrow \psi &\equiv& \neg\varphi \vee \psi \\
\varphi \leftrightarrow \psi &\equiv& (\varphi \wedge \psi) \vee (\neg\varphi \wedge \neg\psi) & \forall\varphi &\equiv& \neg\exists\neg\varphi \\
F\varphi &\equiv& (\mathbf{true}\,\mathcal{U}\varphi) & G\varphi &\equiv& \neg F\neg\varphi
\end{array}$$

In general $\mathcal{CTL}(\Sigma)$ formulas may express the existence of a potentially infinite computation. When dealing with infinite paths, it is often desirable to impose certain fairness constraints on the possible paths. For example, we would typically wish to insure that each element of a composition must make progress if possible. One method for doing this would be to extend our notion of an fts to contain some type of fairness constraint. Here we take a simpler but less flexible approach. A run is fair if every transition which is enabled infinitely often occurs infinitely often. This condition is sufficient to guarantee progress by all components in a composition, and we show show that it is suitable for use with $\mathcal{CTL}(\Sigma)$. It is *not* suitable for use with a linear temporal logic.

Definition 12 *$\rho \in P_p^\infty$ is a fair path from p if $\forall q \in \mathbf{inf}(\rho)\forall(q,\sigma,r) \in \Delta\ (\rho\downarrow\{(q,\sigma,r)\} = (q,\sigma,r)^\infty)$.*

Definition 13 *A set of states $S \subseteq K$ is said to be closed if $\Delta(S, \Sigma \cup \{\tau\}) \subseteq S$, i.e., if every state reachable from S is a member of S.*

Given a notion of a fair run, we define the semantics of $\mathcal{CTL}(\Sigma)$ as follows. In what follows, we identify a run $\rho \in P_q^\infty$ with its starting state q.

Definition 14 *Satisfaction of $\varphi \in \mathcal{CTL}(\Sigma)$ by L at state q (denoted $L, q \models \varphi$), is defined by:*

i. *$L, q \models \sigma$ iff $\sigma \in \mathcal{L}(q)$.*

ii. *For φ a state formula, $L, q \models \neg\varphi$ iff not $L, q \models \varphi$.*

iii. *$L, q \models \varphi \wedge \psi$ iff $L, q \models \varphi$ and $L, q \models \psi$.*

iv. *$L, q \models \exists\varphi$ iff there exists a fair run $\rho \in P_q^\infty$ such that $L, \rho \models \varphi$.*

v. *$L, \rho \models (\varphi\mathcal{U}\psi)$ iff there is a θ with $\rho \preceq \theta$ such that:*

   (a) *$L, \theta \models \psi$.*

   (b) *$L, \eta \models \varphi$ for all η satisfying $\rho \preceq \eta \prec \theta$.*

vi. *For φ a path formula, $L, \rho \models \neg\varphi$ iff not $L, \rho \models \varphi$.*

We now define a notion of equivalence which is designed to preserve $\mathcal{CTL}(\Sigma)$ formulas. This definition is designed to equate fts which differ only by finite stuttering. The notion here is similar to the notion of stuttering equivalence defined in [5], but it only makes reference to finite paths.

Definition 15 *$p \overset{\sigma}{\Rightarrow}_R q$, where $R \subseteq K \times K$, if there exists $(p,(p,\tau,p_1)\dots(p_{n-1},\tau,p_n)(p_n,\sigma,q)) \in P_p^*$ such that $p\ R\ p_1\ R \dots R\ p_n$. In this definition, we allow $\sigma = \tau$.*

Definition 16 $F_p(R) = \{(p,q) | \forall\sigma \in \Sigma \cup \{\tau\}$*:*

i. $\forall p' [p \overset{\sigma}{\Rightarrow}_R p'$ *implies* $\exists q' (q \overset{\sigma}{\Rightarrow}_R q' \wedge p'\ R\ q')]$

ii. $\forall q' [q \overset{\sigma}{\Rightarrow}_R q'$ *implies* $\exists p' (p \overset{\sigma}{\Rightarrow}_R p' \wedge p'\ R\ q')]\}$.

Proposition 2 *F_p is monotonic and equivalence-preserving.*

Proof The proof of monotonicity is straightforward. To see that F_p is equivalence preserving, assume R is an equivalence relation. It is clear that $F_p(R)$ will be reflexive and symmetric. Suppose $(p,q),(q,r) \in F_p(R)$, and let $p \overset{\sigma}{\Rightarrow}_R p'$. Since $(p,q) \in F_p(R)$, there exists q' such that $q \overset{\sigma}{\Rightarrow}_R q'$ and $p'\ R\ q'$. Since $(q,r) \in F_p(R)$, there exists r' such that $r \overset{\sigma}{\Rightarrow}_R r'$ and $q'\ R\ r'$. By transitivity of R, we have $p'\ R\ r'$. Similarly, if $r \overset{\sigma}{\Rightarrow}_R r'$, there exists p' such that $p \overset{\sigma}{\Rightarrow}_R p'$ and $p'\ R\ r'$. Hence $(p,r) \in F_p(R)$. □

We denote the equivalence induced by F_p by $\approx_p$. $L \approx_p L'$ will indicate that $q_0 \approx_p q_0'$. Let $\overset{\sigma}{\Rightarrow}_p$ denote the relation $\overset{\sigma}{\Rightarrow}_R$ for $R = \approx_p$.

Lemma 1 *Assume L and L' are fts, $\Sigma = \Sigma'$. Let $p \in K$, $p' \in K'$ be states, $\mathcal{L}(p) = \mathcal{L}(p') \wedge p \approx_p p'$, and let $\rho \in P_p^*$. Then there exist $\rho' \in P_{p'}^*$ and partitions $B_0 B_1 \dots B_n$ and $B_0' B_1' \dots B_n'$ of* $\mathrm{st}(\rho)$ *and* $\mathrm{st}(\rho')$ *such that for all $q \in B_i$ and $q' \in B_i'$, $\mathcal{L}(q) = \mathcal{L}(q') \wedge q \approx_p q'$.*

Proof By induction on $m = |\rho|$.

Case $m = 0$:
Choose $\rho' = (p', \epsilon)$, $B_0 = p$, $B_0' = p'$.

Case $m + 1$, assuming the result for m:
Write $\rho = \theta(p_m, \sigma, p_{m+1})$. By induction hypothesis, there exist $\theta' \in P_{p'}^*$ and partitions $B_0 B_1 \dots B_n$ and $B_0' B_1' \dots B_n'$ of $\mathrm{st}(\theta)$ and $\mathrm{st}(\theta')$ such that for all $q \in B_i$ and $q' \in B_i'$, $\mathcal{L}(q) = \mathcal{L}(q') \wedge q \approx_p q'$. Let $r_0' = \mathrm{ls}(\theta')$. Note that $p_m \approx_p r_0'$ and $p_m \overset{\sigma}{\Rightarrow}_p p_{m+1}$. Hence there is $\eta' = (r_0', \tau, r_1') \dots (r_{k-1}', \sigma, r_k')$ which shows that $r_0' \overset{\sigma}{\Rightarrow}_p r_k'$ with $p_{m+1} \approx_p r_k'$. By the definition of labeling, we have $\mathcal{L}(p_m) = \mathcal{L}(r_0') = \mathcal{L}(r_1') = \dots = \mathcal{L}(r_{k-1}')$ and $\mathcal{L}(p_{m+1}) = \mathcal{L}(r_k')$. Further, $p_m \approx_p r_0' \approx_p r_1' \approx_p \dots \approx_p r_{k-1}'$. Let $\rho' = \theta' \eta'$, and define partitions of $\mathrm{st}(\rho)$ and $\mathrm{st}(\rho')$ by:

i. $C_i = B_i$, $C_i' = B_i'$ for $0 \leq i < n$.

ii. $C_n = B_n$, $C_n' = B_n' r_1' \dots r_{k-1}'$.

iii. $C_{n+1} = p_{m+1}$, $C_{n+1}' = r_k'$.

By the above, these satisfy the required conditions. □

Lemma 2 *Let $\rho \in P_p^*$ with* $\mathrm{ls}(\rho) = q$. *There exists $\theta \in P_q^\infty$ such that $\rho\theta$ is fair.*

Proof Write $r_1 \sqsubseteq r_0$ iff r_1 is reachable from r_0. The relation $r_0 \sim r_1$ iff $r_0 \sqsubseteq r_1 \wedge r_1 \sqsubseteq r_0$ is trivially an equivalence relation on the set of states, and $\sqsubseteq$ is a partial order modulo $\sim$. Consider the maximal chains headed by the equivalence class of q. Since the set of states is finite, each such chain has a least element. Let the equivalence class of r be such a least element. Note that this equivalence class represents a closed and strongly connected set of states. From the definition of fairness, a run $\zeta \in P_r^\infty$ which

i. visits each state in turn, and

ii. at each state, takes each transition in turn

is fair. Let $\eta \in P_q^*$ be a run with $\mathbf{ls}(\eta) = r$. Then $\theta = \eta\zeta$ satisfies the required conditions. □

Lemma 3 *If $\rho \in P_p^\infty$ is fair, then* $\mathbf{inf}(\rho)$ *is closed.*

Proof Let $q \in \mathbf{inf}(\rho)$ and $(q, \sigma, r) \in \Delta$. Since q is visited infinitely often, by the definition of fairness, r is visited infinitely often. Hence $r \in \mathbf{inf}(\rho)$, and so $\mathbf{inf}(\rho)$ is closed. □

Theorem 1 *Assume L and L' are fts with $\Sigma = \Sigma'$. Let p and p' be states of L and L' with $\mathcal{L}(p) = \mathcal{L}(p') \wedge p \approx_p p'$. Then for all $\varphi \in \mathcal{CTL}(\Sigma)$:*

$$L, p \models \varphi \textit{ iff } L', p' \models \varphi.$$

Proof By induction on the structure of φ.

$\varphi = \sigma \in \Sigma$:
$L, p \models \sigma$ iff $\sigma \in \mathcal{L}(p)$ iff $\sigma \in \mathcal{L}(p')$ iff $L', p' \models \sigma$.

$\varphi = \neg\psi$, a state formula:
$L, p \models \neg\psi$ iff not $L, p \models \psi$ iff not $L', p' \models \psi$ iff $L', p' \models \neg\psi$.

$\varphi = \psi_0 \wedge \psi_1$:
$L, p \models \psi_0 \wedge \psi_1$ iff $L, p \models \psi_0$ and $L, p \models \psi_1$ iff $L', p' \models \psi_0$ and $L', p' \models \psi_1$ iff $L', p' \models \psi_0 \wedge \psi_1$.

$\varphi = \exists\psi$:
We consider two cases here.

Case 1, ψ is logically equivalent to $(\psi_0 \mathcal{U} \psi_1)$:
Assume $L, p \models \varphi$. Then there is a fair run $\rho \in P_p^\infty$ such that $L, \rho \models \psi$. By definition, there is θ with $\rho \preceq \theta$ such that:

i. $L, \theta \models \psi_1$.

ii. $L, \eta \models \psi_0$ for all η satisfying $\rho \preceq \eta \prec \theta$.

Choose some $\zeta' \in P_{p'}^*$ corresponding to ρ/θ as in lemma 1, and pick θ' as in lemma 2 so that $\zeta'\theta'$ is fair. Define $\rho' = \zeta'\theta'$. By the choice of ζ' and the induction hypothesis, we have:

i. $L', \theta' \models \psi_1$.

ii. $L', \eta' \models \psi_0$ for all η' satisfying $\rho' \preceq \eta' \prec \theta'$.

Since $\rho' \in P_{p'}^\infty$, $L', p' \models \varphi$. The converse is similar.

Case 2, ψ is logically equivalent to $\neg(\psi_0 \mathcal{U} \psi_1)$:

Suppose $L, p \models \varphi$. There are two cases. Assume first that there is a fair run $\rho \in P_p^\infty$ and a θ with $\rho \preceq \theta$ such that:

i. $L, \theta \models \neg\psi_0, \neg\psi_1$.

ii. $L, \eta \models \neg\psi_1$ for all $\rho \preceq \eta \prec \theta$.

In this situation, we proceed as in case 1 above to find a fair run $\rho' \in P_{p'}^\infty$ which demonstrates that $L', p' \models \varphi$. In the second case, there is a fair run $\rho \in P_p^\infty$ such that for every θ satisfying $\rho \preceq \theta$, we have $L, \theta \models \psi_0, \neg\psi_1$. By lemma 3, $\mathbf{inf}(\rho)$ is closed. Define $I' = \{q' \in K' | \mathcal{L}(q) = \mathcal{L}(q') \wedge q \approx_p q'$ for some $q \in \mathbf{inf}(\rho)\}$. We want to show that I' is closed. To see this, suppose $q' \in I'$ and $(q', \sigma, r') \in \Delta'$. Now $q' \overset{\sigma}{\Rightarrow}_p r'$, and by definition of I', there is $q \in \mathbf{inf}(\rho)$ such that $\mathcal{L}(q) = \mathcal{L}(q') \wedge q \approx_p q'$. Hence there is r such that $q \overset{\sigma}{\Rightarrow}_p r$ and $r \approx_p r'$. Further, $\mathcal{L}(r) = \mathcal{L}(r')$, and $r \in \mathbf{inf}(\rho)$ since $q \overset{\sigma}{\Rightarrow}_p r$. Thus $r' \in I'$ and I' is closed.

Now for every $q \in \mathbf{inf}(\rho)$, we have $L, q \models \psi_0, \neg\psi_1$, and so by induction hypothesis $L', q' \models \psi_0, \neg\psi_1$ for every $q' \in I'$. Choose θ such that $\rho \preceq \theta$ and $\mathbf{ls}(\rho/\theta) \in \mathbf{inf}(\rho)$. As in case 1 above, there is $\zeta' \in P_{q'}^{*}$ corresponding to ρ/θ, and there is θ' so that $\zeta'\theta'$ is fair. Define $\rho' = \zeta'\theta'$. By induction hypothesis, for all $\rho' \preceq \eta' \prec \theta'$ we have $L', \eta' \models \psi_0, \neg\psi_1$. Also note that $\mathbf{ls}(\zeta') \in I'$. Hence for all $\theta' \preceq \eta'$, we have $L', \eta' \models \psi_0, \neg\psi_1$. Thus $L', \rho' \models \neg(\psi_0 \mathcal{U} \psi_1)$, so $L', p' \models \varphi$. The converse is similar. □

Corollary 1 *If $L \approx_p L'$, then for all $\varphi \in CTL(\Sigma)$ we have $L, q_0 \models \varphi$ iff $L', q'_0 \models \varphi$.*

We also show that the equivalence is a congruence with respect to the operations. For hiding and renaming, the result is trivial. For composition we have the following.

Theorem 2 *Let L_0, L'_0, L_1, and L'_1 be fts with $\Sigma_0 = \Sigma'_0$, $\Sigma_1 = \Sigma'_1$, $L_0 \approx_p L'_0$, and $L_1 \approx_p L'_1$. Then $L_0 \,||\, L_1 \approx_p L'_0 \,||\, L'_1$.*

Proof Define R by $(p_0, p_1)\ R\ (p'_0, p'_1)$ iff $p_0 \approx_p p'_0$ and $p_1 \approx_p p'_1$. It is trivial to check that R is an F_p-bisimulation, and hence that $R \subseteq \approx_p$. But now from the hypotheses, $q_0 \approx_p q'_0$, $q_1 \approx_p q'_1$, so $(q_0, q_1)\ R\ (q'_0, q'_1)$, and hence $L_0 \,||\, L_1 \approx_p L'_0 \,||\, L'_1$. □

Using corollary 1 and theorem 2 it is easy to show that the first two conditions required for the interface rule are satisfied. The last two conditions have straightforward proofs.

As an example, we consider a tree arbiter used to control access to a shared resource. An arbiter cell has three communication channels which we denote by *C0*, *C1*, and *Cp*. Each channel consists of two signals, r and a, representing a request and an acknowledgement. A user request on one of the channels *C0* or *C1* initiates a request to a server on channel *Cp*. After an acknowledge is received on *Cp*, an acknowledge is passed on to the user. At this point the user is assumed to have access to the shared resource. The user initiates another request/acknowledge cycle when finished. By combining arbiter cells into a binary tree, we can form an arbiter for any number of users. An arbiter with three cells is shown in figure 2. The specification here is based on an example presented in [9]. We will represent an arbiter cell by the composition of the fts shown in figure 3. The fts for the users and the server are shown in figure 4.

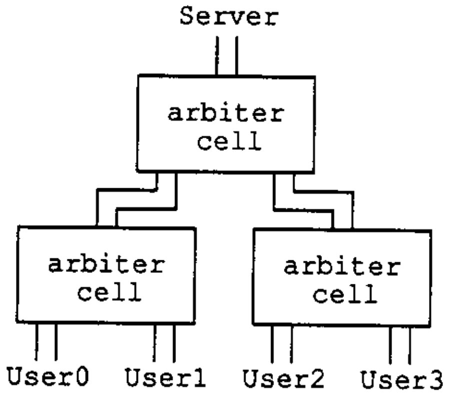

Figure 2: Three cell tree arbiter

We write *Arbiter* for the arbiter cell. To verify the class of tree arbiters, we begin by checking the following relations:

$$\begin{aligned}
&((\mathit{Arbiter} \,||\, \mathit{User0} \,||\, \mathit{User1})[rp\,/\,r0'][ap\,/\,a0'])\\
&\qquad \setminus r0 \setminus a0 \setminus r1 \setminus a1 \setminus t0 \setminus t1 \approx_p \mathit{User0'}\\
&((\mathit{Arbiter} \,||\, \mathit{User0} \,||\, \mathit{User1})[rp\,/\,r1'][ap\,/\,a1'])\\
&\qquad \setminus r0 \setminus a0 \setminus r1 \setminus a1 \setminus t0 \setminus t1 \approx_p \mathit{User1'}\\
&((\mathit{Arbiter} \,||\, \mathit{User1} \,||\, \mathit{Server})[r0\,/\,rp'][a0\,/\,ap'])\\
&\qquad \setminus r1 \setminus a1 \setminus rp \setminus ap \setminus t0 \setminus t1 \approx_p \mathit{Server'}\\
&((\mathit{Arbiter} \,||\, \mathit{User0} \,||\, \mathit{Server})[r1\,/\,rp'][a1\,/\,ap'])\\
&\qquad \setminus r0 \setminus a0 \setminus rp \setminus ap \setminus t0 \setminus t1 \approx_p \mathit{Server'}
\end{aligned}$$

The first relation here indicates that when we compose two users with an arbiter cell and restrict to the actions for the server port, the result is equivalent to another user. Thus, a user process can be used as an interface process for the two users and the arbiter cell. The other relations have similar interpretations. From these, we can perform an induction on the structure of a tree arbiter to deduce that each cell in an arbiter with users at the leaves and a single server at the root is equivalent to a cell in an environment of two users and a server, i.e., to:

$$\mathit{Arbiter} \,||\, \mathit{User0} \,||\, \mathit{User1} \,||\, \mathit{Server}$$

Properties of the entire arbiter are deduced from properties of these components. For example, liveness for the first user can be checked immediately by verifying:

$$\forall G(r0 \rightarrow \forall F(r0 \wedge a0)).$$

We can ensure mutual exclusion by checking:

$$\forall G(\neg(r0 \wedge a0) \vee \neg(r1 \wedge a1))$$

and:

$$\forall G((r0 \wedge a0) \rightarrow (rp \wedge ap)) \wedge \forall G((r1 \wedge a1) \rightarrow (rp \wedge ap)).$$

This example illustrates how it is sometimes possible to reason inductively about a system using the interface theorem.

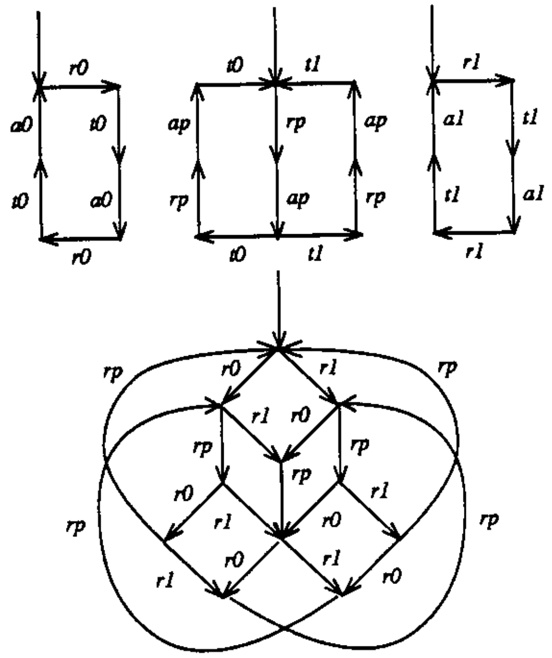

Figure 3: Tree arbiter cell

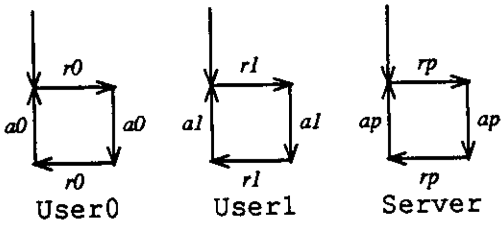

Figure 4: Users and server

5. A synchronous model and logic

In this section, we introduce a simple formal model of finite state machine composition which satisfies the conditions for the interface theorem and is applicable to synchronous hardware controllers. We also show by an example how the interface theorem may be applied to automatic verification of digital hardware.

We use a model of communicating Moore machines to represent modular hardware control units.

Definition 17 *A Moore machine is given by a structure* $M = (K, q_0, \Sigma_i, \Sigma_o, \Gamma, \Theta)$ *where*

i. *K is the set if states*

ii. $q_0 \in K$ *is the initial state*

iii. Σ_i *and* Σ_o *are disjoint sets of propositional variables (representing the input and outputs respectively)*

iv. *Γ is a mapping $K \rightarrow \Sigma_o \rightarrow \{\top, \bot\}$ (for each state, a truth assignment to each output)*

v. *Θ is a mapping $K \times K \rightarrow \mathcal{F}(\Sigma_i)$, where $\mathcal{F}(\Sigma_i)$ is the set of propositional logic formulas on Σ_i ($\Theta(s_1, s_2)$ represents the transition condition from state s_1 to state s_2).*

Definition 18 *A run of a Moore machine M beginning at a state s_0 is a sequence $(s_0, J_0)(s_1, J_1)\ldots$, where $s_0, s_1, \ldots \in K$, $J_0, J_1, \ldots$ are mappings $\Sigma_i \rightarrow \{\top, \bot\}$ and the following condition holds:*

$$\forall i \geq 0 : J_i \models \Theta(s_i, s_{i+1})$$

Definition 19 *Given a propositional formula f and a mapping $A : \Sigma \rightarrow \{\top, \bot\}$, let $f[A]$ be the formula f with each variable $x \in \Sigma$ replaced by $A(x)$.*

Definition 20 *The parallel composition $M \,||\, M'$, where Σ_o and Σ'_o are disjoint, is a Moore machine M'' such that*

i. $K'' = K \times K'$

ii. $q''_0 = (q_0, q'_0)$

iii. $\Sigma''_i = (\Sigma_i - \Sigma'_o) \cup (\Sigma'_i - \Sigma_o)$

iv. $\Sigma''_o = \Sigma_o \cup \Sigma'_o$

v. $\Gamma''(s, s') = \Gamma s \cup \Gamma' s'$

vi. $\Theta''((s_1, s'_1), (s_2, s'_2)) =$
$\Theta(s_1, s_2)[\Gamma' s'_1 x] \wedge \Theta'(s'_1, s'_2)[\Gamma s_1]$.

Definition 21 *The restriction of a Moore machine M to a set of variables Σ (where $\Sigma_i \subseteq \Sigma$), denoted $M' = M \downarrow \Sigma$, is identical to M except that $\Sigma'_o = \Sigma_o \cap \Sigma$ and for all $s \in K$, $\Gamma' s = \Gamma s \downarrow \Sigma$.*

Using this process model, we now have to define a logic and an appropriate equivalence. The logic we use will be the temporal logic CTL* [11]. For the notion of equivalence for Moore machines, we first define equivalence between states of a Moore machine.

Definition 22 *The state equivalence relation $\sim_M$ (written $\sim$ where the context is unambiguous) of a Moore machine M is the unique greatest fixpoint of the functional $F_M : K \times K \rightarrow K \times K$ such that $(s_1, s_2) \in F_M(R)$ iff*

i. $(s_1, s_2) \in R$

ii. $\Gamma s_1 = \Gamma s_2$

iii. $\forall A : \Sigma_i \rightarrow \{\top, \bot\}, \forall s' \in K,$

(a) $A \models \Theta(s_1, s')$ *implies* $\exists s'' \in K$, $(s', s'') \in R$ *and* $A \models \Theta(s_2, s'')$

(b) $A \models \Theta(s_2, s')$ *implies* $\exists s'' \in K$, $(s'', s') \in R$ *and* $A \models \Theta(s_1, s'')$

Here, we have omitted the proof that F_M is monotonic and equivalence preserving.

Definition 23 *Given two Moore machines M and M′,* $\Sigma_o = \Sigma'_o$, $\Sigma_i = \Sigma'_i$ *and* $K \cap K' = \emptyset$, *their disjoint sum* $M'' = M + M'$ *is the Moore machine* $(K \cup K', q''_0, \Sigma_i, \Sigma_o, \Gamma \cup \Gamma', \Theta \cup \Theta')$, *where* q''_0 *is undefined.*

Definition 24 *Given two Moore machines M and M′, let* $\sim$ *be the state equivalence relation of* $M + M'$. *We say M and M′ are equivalent, denoted by* $M \sim M'$ *iff* $q_0 \sim q'_0$.

We define the CTL* semantics for a Moore machine M by deriving a Kripke structure K_M from M and then using the standard Kripke structure semantics for CTL*.

Definition 25 *Given a Moore machine M, let* $K_M = (S, S_0, R, L)$, *where*

i. *S is a set of pairs* (s, A), *where* $s \in K$ *and A is a mapping* $\Sigma_i \to \{\top, \bot\}$.

ii. $S_0 = \{(s, A) \in S | s = q_0\}$

iii. *L is a map* $S \to 2^{\Sigma_i \cup \Sigma_o}$ *where* $L(s, A) = \{x \in \Sigma_o | (\Gamma s)x = \top\} \cup \{x \in \Sigma_i | Ax = \top\}$

iv. $R \subseteq S \times S$ *is defined such that* $(s_1, A_1) R (s_2, A_2)$ *iff* $A_1 \models \Theta(s_1, s_2)$.

Note that the set of runs of K_M is identical to the set of runs of L.

Definition 26 *If f is a CTL* formula, we define* $M \models f$ *iff* $K_M \models f$

As an example, we consider a model of the controller of a CPU, which is illustrated as a block diagram in figure 5. A detailed description of the CPU and a partial formal specification can be found in [8]. The CPU is broken into two units, called the access unit and the execution unit, in order to allow concurrency between memory operations and instruction execution. The access unit's function is to fetch instructions and store them in an instruction queue, and to maintain a cache of the top location of the CPU stack in a special register. The execution unit's function is to interpret instructions of the CPU's machine code (which it stack based). The two controllers of the access and execution units (which we will refer to as AU and EU) are designed as communicating Moore machines. A major part of the temporal logic specification for the CPU's controller defines correct behavior for the AU and consists of formulas on the set of signals which are inputs or outputs of the AU (which we will call Σ_{AU}). A simple example of such a formula is the following

$$\forall G \forall F\, fetch$$

This formula is a liveness property which states that instructions are fetched from the access unit to the execution unit infinitely often (*fetch* is actually a propositional formula defined in terms of request and acknowledge signals between the EU and AU).

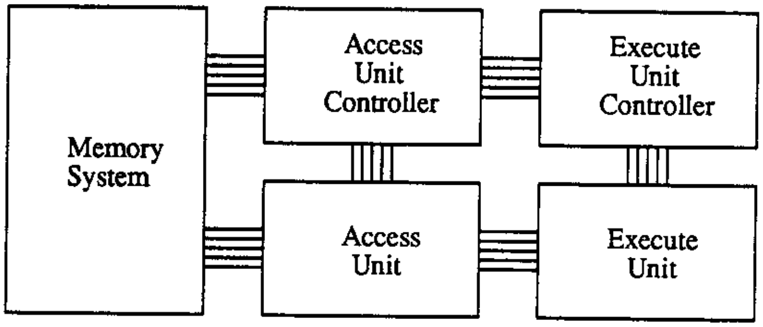

Figure 5: CPU block diagram

The parallel composition of the AU and EU controllers in our design has approximately 1100 reachable states. However, by restricting the outputs of the EU to those in Σ_{AU}, and then minimizing it, we obtain an interface process EU' such that $AU \| EU'$ has only 196 reachable states. The reason for this reduction is that, while the execution unit interprets many different instructions, the memory accesses of these instructions fall into a few basic patterns. By checking the AU specifications on this reduced composition, we are making use of the interface rule. In order to justify this step, however, we must prove the four conditions of section 2.

To demonstrate the first condition of the interface rule, we would like to show that $M \sim M'$ implies $K_M \equiv K_{M'}$, where $\equiv$ denotes traditional strong bisimulation equivalence. A straightforward induction can then be used to show that M and M' satisfy the same set of CTL* formulas. We first define a relation $\approx$ on the states of K_M as follows.

Definition 27 *Given a Kripke structure K_M, and two states $(s_1,A_1),(s_2,A_2) \in S$, $(s_1,A_1) \approx (s_2,A_2)$ iff $s_1 \sim s_2$ and $A_1 = A_2$.*

Lemma 4 *The relation $\approx$ is a strong bisimulation on K_M.*

Proof Assume that $(s_1,A)R(s_1',B)$ and $(s_1,A) \approx (s_2,A)$. Then by definitions 25 and 27,

$$\begin{aligned} L(s_1,A) &= \{x \in \Sigma_o | (\Gamma s_1)x = \top\} \cup \{x \in \Sigma_i | Ax = \top\} \\ &= \{x \in \Sigma_o | (\Gamma s_2)x = \top\} \cup \{x \in \Sigma_i | Ax = \top\} \\ &= L(s_2,A). \end{aligned}$$

By definition 25, $A \models \Theta(s_1,s_1')$, and by definition 27, $s_1 \sim s_2$. Thus, since $\sim$ is a fixpoint of F_M, $\exists s_2' \in K, A \models \Theta(s_2,s_2')$ and $s_1' \sim s_2'$. Hence $(s_1',B) \approx (s_2',B)$ and $(s_2,A)R(s_2',B)$. Proof of the other direction is similar. □

Corollary 2 *If $M \sim M'$ then $K_M \equiv K_{M'}$.*

Proof $M \sim M'$ implies a state equivalence $\sim$ on $M + M'$ such that $q_0 \sim q_0'$. By lemma 4, this implies a strong bisimulation $\equiv$ on $K_{M+M'} = K_M + K_{M'}$ such that, for all $A : \Sigma_i \rightarrow \{\top,\bot\}$, $(q_0,A) \equiv (q_0',A)$. □

The equivalence $\sim$ on states of a Moore machine is essentially the same notion used to define equivalence in the coarsest partitioning algorithm of [17]. Thus there is an algorithm for determining if $M \sim M'$

(i.e., for determining if the initial states of M and M' fall into the same equivalence class) in time $O((|M| + |M'|)\log(|M| + |M'|))$, assuming the transition conditions are coded in disjunctive normal form. Further, M can be minimized in time $O(|M| \log |M|)$.

In order to satisfy the second condition of the interface rule, we must show that $M_1 \sim M_1'$ and $M_2 \sim M_2'$ imply $M_1 \| M_2 \sim M_1' \| M_2'$. To do this, we define a relation $\approx$ (different from $\approx$ above) on the states of $(M_1 \| M_2) + (M_1' \| M_2')$ as follows:

Definition 28 *$(s_1, s_2) \approx (s_1', s_2')$ iff $s_1 \sim s_1'$ and $s_2 \sim s_2'$.*

We note that the relation $\approx$ is a $F_{(M_1 \| M_2)+(M_1' \| M_2')}$ – *bisimulation.* Hence we have

Proposition 3 *$M_1 \sim M_1'$ and $M_2 \sim M_2'$ imply $M_1 \| M_2 \sim M_1' \| M_2'$*

Proof Since the state equivalence relation $\sim$ on $(M_1 \| M_2) + (M_1' \| M_2')$ is the largest bisimulation of $F_{(M_1 \| M_2)+(M_1' \| M_2')}$, we have that $(s_1, s_2) \approx (s_1', s_2')$ implies $(s_1, s_2) \sim (s_1', s_2')$. By definition 28, $(q_{01}, q_{02}) \approx (q_{01}', q_{02}')$, therefore $(q_{01}, q_{02}) \sim (q_{01}', q_{02}')$, satisfying the definition of equivalence for $M_1 \| M_2$ and $M_1' \| M_2'$. □

We next prove that our definitions satisfy condition three. Although this condition requires equivalence, we show a stronger condition of equality.

Lemma 5 *$(M \| M') \downarrow \Sigma_M = M \| (M' \downarrow \Sigma_M)$*

Proof We consider the Θ component. Let $P = (M \| M') \downarrow \Sigma_M$ and $Q = M \| (M' \downarrow \Sigma_M)$. Then we have

$$\begin{aligned}
& \Theta_P((s_1, s_1'), (s_2, s_2')) \\
= \; & \Theta(s_1, s_2)[\Gamma' s_1'] \wedge \Theta'(s_1', s_2')[\Gamma s_1] \\
= \; & \Theta(s_1, s_2)[(\Gamma' \downarrow \Sigma_M) s_1'] \wedge \Theta'(s_1', s_2')[\Gamma s_1] \\
= \; & \Theta(s_1, s_2)[\Gamma_{M' \downarrow \Sigma_M} s_1'] \wedge \Theta_{M' \downarrow \Sigma_M}(s_1', s_2')[\Gamma s_1] \\
= \; & \Theta_Q((s_1, s_1'), (s_2, s_2'))
\end{aligned}$$

The other components are trivially equal. □

The last condition is trivially satisfied, since restricting a Moore machine M to a set of variables Σ is equivalent to restricting the labels of the Kripke structure K_M to Σ.

6. Directions for Future Research

The most important open question is, of course, whether the compositional techniques described in this paper will permit verification of much more complicated finite state concurrent systems than has previously been the case. This can only be determined by further experimentation. It is clear, however, that our technique is most suitable for loosely coupled systems. When this relationship does not hold, the interface processes may be large, and we do not get a significant state reduction by using the interface theorem. Fortunately, the parallel composition of two tightly coupled processes does not seem to generate as many states as the parallel composition of two loosely coupled processes of comparable size—there are simply not as many possible interleavings. Consequently, compositional reasoning may not be as important in this case as in the case of loosely coupled processes.

It will be interesting to see how well the results of this paper apply to other process models. For systems like finite transition systems with propositional dynamic logic [14] or CCS [20] with Hennessy-Milner logic, the results should be straightforward and will be given in the full version of the paper. Our techniques should work quite well for the Caesar system of Sifakis [23]. It should also be possible to apply our ideas to Berry's Esterel [2] and Harel's Statecharts [15] if we use a logic like CTL.

Finally, the techniques that we describe in this paper also have some limitations. For example, the interface rule allows us to handle formulas that are boolean combinations of temporal properties of the individual processes in a parallel composition. We are currently unable to handle more general properties involving temporal assertions about several processes. Developing more general rules seems like an important direction for research but also a very hard one. O. Grumberg [13] has obtained some negative results which indicate that it may be impossible to develop a fully general system of inference rules that will handle arbitrary temporal properties. Furthermore, in some cases it seems likely that it will be necessary to combine the use of the interface rule and model checker with proofs of validity for certain CTL formulas. In order to use the interface rule it may be necessary to prove an implication of the form $(\phi \wedge \psi) \rightarrow \delta$ where δ is another CTL formula that expresses a global property. We believe that in many cases it will be possible to use informal reasoning to establish such implications.

References

[1] H. Barringer. Using temporal logic in compositional specification of concurrent systems. In *Conference on Temporal Logic and Its Applications*, Leeds University, January 1986.

[2] G. Berry and L. Cosserat. *The ESTEREL Synchronous Programming Language and its Mathematical Semantics*. Technical Report, Ecole Nationale Superieune des Mines de Paris, 1984.

[3] M. C. Browne, E. M. Clarke, and D. L. Dill. Automatic circuit verification using temporal logic: two new examples. In *Formal Aspects of VLSI Design*, Elsevier Science Publishers, 1986.

[4] M. C. Browne, E. M. Clarke, D. L. Dill, and B. Mishra. Automatic verification of sequential circuits using temporal logic. *IEEE Transactions on Computers*, C-35(12), December 1986.

[5] M. C. Browne, E. M. Clarke, and O. Grumberg. *Characterizing Kripke Structures in Temporal Logic*. Technical Report, Carnegie Mellon University, Pittsburgh, PA 15213, January 1987.

[6] E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Proceedings of the Workshop on Logic of Programs*, pages 52–71, Springer-Verlag, 1981.

[7] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[8] E. M. Clarke, D. E. Long, and K. L. McMillan. A language for compositional specification and verification of finite state hardware controllers. In *Proceedings of the Conference on Hardware Description Languages*, 1989. To appear.

[9] D. L. Dill. *Trace Theory for Automatic Heirarchical Verification of Speed-Independent Circuits.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA 15213, 1987.

[10] D. L. Dill and E. M. Clarke. Automatic verification of asynchronous circuits using temporal logic. *IEE Proceedings*, 133, part E(5), September 1986.

[11] E. A. Emerson and J. Y. Halpern. "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *Journal of the Association of Computing Machinery*, 33(1):151–178, January 1986.

[12] E.A. Emerson and Chin Laung Lei. Modalities for model checking: branching time strikes back. In *Conference Record of the Twelth Annual ACM Symposium on Principles of Programming Languages*, New Orleans, La., January 1985.

[13] O. Grumberg. Personal communication.

[14] D. Harel. Dynamic logic. In D. Gabby and F. Guenthner, editors, *Handbook of Philosophical Logic II*, pages 498–544, Reidel, 1984.

[15] D. Harel. *Statecharts: A Visual Approach to Complex Systems.* Technical Report CS84-05, The Weizmann Institute of Science, February 1984.

[16] C. A. R. Hoare. Communicating sequential processes. *Communications of the Association of Computing Machinery*, 21(8):666–677, August 1978.

[17] J. E. Hopcroft. An $n \log n$ algorithm for minimizing the states in a finite automaton. In *The Theory of Machines and Computation*, pages 189–196, Academic Press, New York, N.Y., 1971.

[18] B. Josko. MCTL - an extension of CTL for modular verification of concurrent systems. In H. Barringer, editor, *Workshop on Temporal Logic*, University of Manchester, April 1987. To appear in LNCS.

[19] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Conference Record of the Twelth Annual ACM Symposium on Principles of Programming Languages*, New Orleans, La., January 1985.

[20] R. Milner. *A Calculus of Communicating Systems.* Volume 92 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1980.

[21] B. Mishra and E.M. Clarke. Hierarchical verification of asynchronous circuits using temporal logic. *Theoretical Computer Science*, 38:269–291, 1985.

[22] A. Pnueli. In transition for global to modular temporal reasoning about programs. In K. R. Apt, editor, *Logics and Models of Concurrent Systems*, pages 123–144, Springer-Verlag, 1984.

[23] J. P. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the Fifth International Symposium in Programming*, 1981.

[24] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the Conference on Logic in Computer Science*, Boston, Mass., June 1986.