THE MODULAR FRAMEWORK OF COMPUTER-AIDED VERIFICATION

Gil Shurek and Orna Grumberg
Computer Science Department, The Technion
Haifa 32000, Israel
E-address: orna@techsel (BITNET), orna@sel.technion.ac.il (CSNET)

1. INTRODUCTION

Temporal logic model checking procedures have been proven to be a feasible approach to automatic verification of relatively small finite-state systems. As the expressive power of temporal logic makes it adequate for specifying properties of reactive systems and as interesting examples are typically those of concurrent and distributed systems, it seems natural to consider the applicability area of automatic model checking as that of reactive finite-state concurrent systems. This context, however, exposes the major disadvantage of model checkers, which is their global nature. By this we mean that only a complete system can be verified to meet its temporal specifications. This globality, on the one hand, induces the "state explosion problem" and, on the other hand, prevents the realization of the "pre-verified component methodology", presented below.

The *state explosion problem* arises in systems composed of many loosely coupled components. The size of such a system may grow as the product of the sizes of its components. Thus, space requirement might become too large for a model checking procedure to be applicable. Obviously, state explosion affects the time-complexity as well.

The *pre-verified component methodology* refers to the verification of systems containing an already verified component, that avoids redoing the component's proof. When verifying a specific system, it should be possible to exploit the pre-verification of the component. As a result, the average verification time-complexity over the set of systems containing this pre-verified component should be reduced.

It seems natural to consider the use of *compositional* reasoning as a solution to the state explosion problem. Exploiting the modular structure of a system, a compositional verification method applies model checking procedures to individual components, to check properties from which the specification of the global system is deduced. The construction of the global system is avoided, and thus space requirements are reduced. Hopefully time-complexity is reduced as well. The desirability of using pre-verified components raises the need for our method to be *modular* too. By this we mean that it should be possible to apply a verification procedure to an individual component when no specific environment is available. The difference between compositionality and modularity lies in the amount of information that the verification procedure has about one component when verifying the other. In a compositional approach complete knowledge about the environment may be used, while in the modular approach no information about the actual environment is available. Note that our terminology adapts the terms "compositionality" and "modularity", yielding notions that differ slightly from the traditional ones.

This paper presents a modular, semi-automatic method for the verification of finite-state programs with temporal logic specifications. The *purpose* of this work, however, goes beyond the search for a technical solution to modular verification. The additional goal is to gain a wider understanding of the modular framework. In a related work [SG90] we discuss motivation, solutions, solutions' properties, and evaluation criteria in that context, e.g.:

- An obvious quality of a global model checker that should be preserved in a modular procedure is the ability to verify arbitrary properties of a distributed system, not being restricted to properties of single components within the compound system.
- Modular procedures have no knowledge about the environment. Hence, applying the checking procedure to one component, an *assumption* on the behavior of the environment is often necessary in order to deduce a global property. Assumptions and their effect on the verification process are thoroughly discussed in [SG90].

Following the previous discussion, this work is a search for a partially automated modular verification procedure supported by a model checking procedure, for reactive, finite-state distributed systems. The systems are modeled by finite structures described as state-transition graphs, where asynchronous actions have an interleaving semantics. We assume that specifications are given by propositional temporal logic formulas that are interpreted over the computation tree of the system's transition graph.

Several works on compositional techniques for reasoning about reactive concurrent systems established the background for our research: [P85],[J87], and [CLM89]. As we regard the "*Assume – Guarantee*" paradigm [P85] to be the preferable modular scheme, we begin with an exploration of its underlying mechanism. This is done by presenting a deductive proof system scheme (*AG*) which is an abstraction of that paradigm. Considering a further abstraction (the *AGS*) we claim that, such paradigms essentially manipulate infinite sets of models. The main difficulty in implementing an *AG*-like scheme as a semi-automatic procedure, is identified to be the need to represent these sets so that they can be handled automatically.

We present an *AG*-like proof system scheme, denoted *AGM*, in which a set of models is represented by a single model. In this scheme assumptions are expressed by finite state-transition structures, while the specifications are given in a temporal language. The scheme defines complete proof systems, in which soundness depends on conditions that should be satisfied in an actual framework (i.e., an actual model of computation and a temporal language). Implementations of this scheme as semi-automatic procedures are based on an automatic model checker that is not necessarily modular.

We show how a proof system can be derived from the *AGM* scheme. As the model of computation we choose a simplified version of *CCS* [M80] which describes asynchronous communicating systems. The specification language is a sublanguage of CTL^* (CTL^*_{-E}) which is more expressive than *LTL*. In CTL^*_{-E} negation is restricted to the atomic propositions and the E ("there exists") path quantifier is eliminated. A generalized version of *AGM* (AGM^*) can be realized in this framework. The AGM^* enables simultaneous use of partial information on both components of the distributed system. In AGM^* , verification can be viewed as state-reduction of the global model without construction of the model. The reduction is relative to the property to be verified (as opposed to minimization which guarantees semantic equivalence of models). Proof systems derived from the *AGM* and the AGM^* schemes can be implemented as semi-automatic procedures for modular verification. Properly used, these procedures enable the use of pre-verified components, and guarantee reduced memory requirement and even improved time complexity, compared to a global model checker.

In Section 2 we describe the *AG* and *AGS* proof system schemes. In Section 3 the *AGM* and its generalized version *AGM** are presented. Section 4 presents the actual proof system derived from these schemes. In [SG90] this section includes an application of that proof system to a program.

2. THE *AG* AND *AGS* SCHEMES

2.1 The *AG* Proof System Scheme

The *AG* deductive proof system scheme presented in this section should be viewed as an abstraction of Pnueli's assume-guarantee paradigm [P85].

Let **L** be a logic used as a reasoning and specification language for distributed systems. Let M be a set of models, containing component models and composed models, for which ' $\|$ ' denotes the composition operation. Each composed model m is accompanied with a fixed decomposition into two component models m_A, m_B, such that $m = m_A \| m_B$. For a component model $m \in M$, $C_A[m]$ denotes the set of all models in M of the form $m_A \| m_B$, for which m_A is m (i.e., $C_A[m] = \{m_A \| m_B \mid m_A = m\}$). $C_B[m]$ is defined similarly.
For $\varphi_1, \varphi_2 \in$ **L**, the notation $\varphi_1 \models_{C_i[m]} \varphi_2$, $i \in \{A,B\}$ means that every model in $C_i[m]$ which satisfies φ_1, satisfies φ_2 as well. Thus, φ_2 is the consequence of φ_1 in $C_i[m]$. $\vdash_{C_i[m]}$ is a proof system that validates **L** formulas in $C_i[m]$. The soundness of this system is defined as: $\forall \varphi_1, \varphi_2 \in \mathbf{L}\ [\varphi_1 \vdash_{C_i[m]} \varphi_2 \Rightarrow \varphi_1 \models_{C_i[m]} \varphi_2]$. When $\varphi_1 \equiv$ true, the definition results in $\vdash_{C_i[m]} \varphi_2 \Rightarrow \models_{C_i[m]} \varphi_2$ which means that if φ_2 is provable in $\vdash_{C_i[m]}$ then φ_2 is true in the composition of m with *any* environment.

Given the proof systems $\vdash_{C_A[m_1]}$ and $\vdash_{C_B[m_2]}$ we present below a set of *AG* proof systems (one for each model $m_1 \| m_2$). The proof system $\vdash_{AG-m_1\|m_2}$ for the verification of **L**-formulas in the composed model $m_1 \| m_2$ is defined as follows:

Axioms:

$$\{\varphi \mid \vdash_{C_A[m_1]} \varphi\} \cup \{\varphi \mid \vdash_{C_B[m_2]} \varphi\}$$

Inference rules:

$$\frac{\vdash_{AG-m_1\|m_2} \varphi \qquad \varphi \vdash_{C_A[m_1]} \psi}{\vdash_{AG-m_1\|m_2} \psi} \qquad\qquad \frac{\vdash_{AG-m_1\|m_2} \varphi \qquad \varphi \vdash_{C_B[m_2]} \psi}{\vdash_{AG-m_1\|m_2} \psi}$$

The soundness of $AG-m_1 \| m_2$ is defined as: $\forall \psi \in \mathbf{L}\ [\ \vdash_{AG-m_1\|m_2} \psi \Rightarrow m_1 \| m_2 \models \psi\ ]$.

Theorem 1: Given sound proof systems $\underset{C_A[m_1]}{\vdash}$ and $\underset{C_B[m_2]}{\vdash}$, $\underset{AG-m_1 \| m_2}{\vdash}$ is sound.

The $AG-m_1 \| m_2$ is a scheme for a proof system, capable of verifying properties of the distributed system $m_1 \| m_2$, without being restricted to the behavior of a single component in the context of the global system. Moreover, it can be considered modular according to our terminology. Examining a typical AG proof, its modularity can be observed:

$$\underset{C_A[m_1]}{\vdash} \varphi_n\,,\ \varphi_n \underset{C_B[m_2]}{\vdash} \varphi_{n-1} \ \cdots\ \varphi_1 \underset{C_A[m_1]}{\vdash} \varphi$$

We are interested in a semi-automatic implementation of the AG scheme, where the user is required to supply ("guess") the sequence of assumptions φ_n to φ_1 and the validation in $C_A[m_1]$ and in $C_B[m_2]$ is done automatically. The m_1 proof-block is the set of proof stages in which $\underset{C_A[m_1]}{\vdash}$ is applied. When assumptions are given, the m_1 proof block can be carried out without having to consider any specific "m_2". Given a component model m_i, if the proof-block consisting of all $\underset{C_B[\ ]}{\vdash}$ stages can be carried out successfully with respect to $\underset{C_B[m_i]}{\vdash}$, then m_1 proof-block can be completed to a full AG proof, deducing $m_1 \| m_i \models \varphi$.

2.2 The AG-sets (AGS) Proof System Scheme

In this section we present another proof system scheme, AG-sets (AGS), which is an abstraction of AG, meant to explore the underlying mechanisms of AG. In AGS, sets of models are used directly as the reasoning language (similar to the *Semantic Model* of [AL89]). However, we still assume temporal specification. Therefore, an inference rule that derives **L** formulas (from a previously "derived" model set) is necessary. Defined below is an AGS proof system for $m_1 \| m_2$.

$gs[m_1 \| m_2]$ is a set of model sets, defined inductively by:
Atoms ("axioms"):

$$\{C \mid C \supseteq C_A[m_1]\} \cup \{C \mid C \supseteq C_B[m_2]\}$$

Closure operations ("inference rules"):

$$\frac{C \in gs[m_1 \| m_2] \qquad C' \supseteq C \cap C_A[m_1]}{C' \in gs[m_1 \| m_2]} \qquad\qquad \frac{C \in gs[m_1 \| m_2] \qquad C' \supseteq C \cap C_B[m_2]}{C' \in gs[m_1 \| m_2]}$$

The inference rule:

$$\frac{\varphi \in \mathbf{L} \qquad C \in gs[m_1 \| m_2] \qquad \underset{C}{\models} \varphi}{\underset{AGS-m_1 \| m_2}{\vdash} \varphi}$$

Theorem 2: $AGS-m_1 \| m_2$ is sound and complete.

Noticing that both AG and AGS basically manipulate potentially infinite sets of models, leads us to identify the major obstacle to the implementation of AG. In AG, sets of models are represented either by themselves ($C_A[m_1]$ and $C_B[m_2]$) or by L-formulas (i.e., φ defines and represents the set of all models satisfying φ). The lack of uniform representation for these sets forces AG implementations (e.g., [J87]) to provide conversion procedures (which are, in general, cumbersome and time-consuming) to enable constructive manipulation of model sets. In AGS we abstract away from the *representation problem.* The AGS proof system is theoretic and has no explicit representation for sets. This system is not intended to be implemented directly, but to inspire the definition of other proof system schemes.
In the next section we present an AGS-like proof system, in which sets are represented in a way they can be handled constructively.

3. THE *AG*-MODELS (*AGM*) SCHEMES

3.1 The *AGM* Proof System Scheme

The AGM is an AGS-like proof system scheme, in which the basic elements handled by the deductive procedure are models (i.e., finite state-transition structures). These basic elements are used to represent the potentially infinite sets of models, discussed in the previous section. The AGM proof system is a modular model checking system for finite state distributed system, based on their decomposition into two components.

Let M be a model domain consisting of component models m_i and composed models m, which may differ syntactically. Let Σ be an alphabet and let Σ_m denote the alphabet associated with the model m. Σ_m consists of state labels and/or transition labels. Let $\|$ denote a composition operator that applied to two component models m_i, m_j results in a composed model m, i.e., $m = m_i \| m_j$. Also, let $|_\Sigma$ denote a restriction transformer, that applied to a composed model results in a component model with alphabet set restricted to Σ.

Let **L** be a temporal language interpreted over the domain of models M. Let **L**(Σ) denote the set of formulas in **L** defined over the set of atomic propositions in Σ. We assume (for the semantics of **L**) that if $\varphi \notin L(\Sigma_m)$ then $m \not\models \varphi$.

Let $\sqsupseteq$ denote a *Preorder* (reflexive and transitive relation) defined over M, which has the following semantic property: $m_A \sqsupseteq m_B$ implies that all **L** formulas that are satisfied in m_A, are satisfied in m_B (i.e., if $m_A \sqsupseteq m_B$ than $\forall \varphi \in \mathbf{L}$, $m_A \models \varphi \Rightarrow m_B \models \varphi$).

Let $\vdash_{\overline{MC}}$ denote a proof system for checking the satisfaction of **L** formulas in M models (i.e., a *model checker).*
The AGM definition inherits the structure of the AGS definition, where inference rules which do not deduce temporal formulas, are referred to as closure operations. Defined below is an AGM proof system for $m_1 \| m_2$.

gm[m$_1$ || m$_2$] is a set of models, defined inductively by:
Atoms : $\{m' \| m_2 \mid m' \sqsupseteq m_1\} \cup \{m_1 \| m'' \mid m'' \sqsupseteq m_2\}$
closure operations :

$$\frac{\begin{array}{c} m_1 \parallel m'' \in gm[m_1 \parallel m_2] \\ m' \sqsupseteq (m_1 \parallel m'')|_{\Sigma_{m_1}} \end{array}}{m' \parallel m_2 \in gm[m_1 \parallel m_2]} \qquad \frac{\begin{array}{c} m' \parallel m_2 \in gm[m_1 \parallel m_2] \\ m'' \sqsupseteq (m' \parallel m_2)|_{\Sigma_{m_2}} \end{array}}{m_1 \parallel m'' \in gm[m_1 \parallel m_2]}$$

The inference rule :

$$\frac{\begin{array}{l} m \in gm[m_1 \parallel m_2] \\ \varphi \in \mathbf{L}(\Sigma_m) \\ \underset{MC}{\vdash\!\!\!\longrightarrow} m \models \varphi \end{array}}{\underset{AGM-m_1 \parallel m_2}{\vdash\!\!\!\longrightarrow} \varphi}$$

The soundness of the *AGM* scheme depends on the following conditions (regarding the language, the models domain, the composition operator, the restriction transformer and the semantic preorder) :

Soundness Conditions :

1. For all m', m_A, m_B component models in M, if $m_A \sqsupseteq m_B$ then $m_A \parallel m' \sqsupseteq m_B \parallel m'$.
2. For all m', m_A, m_B component models in M, if $m_A \parallel m' \sqsupseteq m_B \parallel m'$ then $(m_A \parallel m')|_{\Sigma_{m'}} \sqsupseteq (m_B \parallel m')|_{\Sigma_{m'}}$.
3. For all m_A, m_B component models in M , $((m_A \parallel m_B)|_{\Sigma_{m_A}}) \parallel m_B \sqsupseteq m_A \parallel m_B$.

When the composition operator is *not commutative* the dual conditions are required as well.

Theorem 3: Given that the soundness conditions are fulfilled then:

3.1 Using a sound proof system $\underset{MC}{\vdash\!\!\!\longrightarrow}$, $\underset{AGM-m_1 \parallel m_2}{\vdash\!\!\!\longrightarrow}$ is sound.

3.2 Using a complete proof system $\underset{MC}{\vdash\!\!\!\longrightarrow}$, $\underset{AGM-m_1 \parallel m_2}{\vdash\!\!\!\longrightarrow}$ is complete.

Comment: Systems which are composed of more than two components, can be handled by *AGM*-like schemes.

3.1.1 Evaluation

The *AGM* scheme is capable of verifying any property of the global distributed system, which is expressible in **L**. Examining a typical $AGM-m_1 \parallel m_2$ proof, verifying that $m_1 \parallel m_2 \models \varphi$, its modularity can be observed. Regarding the state explosion problem, this proof system is feasible as long as the size of the structures representing the assumptions is significantly less than the size of the components' models. Applying the pre-verified components methodology, feasibility depends on the assumptions strength. As "greater" ($\sqsupseteq$) structures make weaker assumptions, the *AGM* enables the user to weaken the supplied assumptions provided that the knowledge needed for the verification of the temporal specification is preserved.

3.1.2 Resemblance to the *AGS* Scheme

In the *AGM* scheme, a single model of a composed system represents an *AGS* basic element: a set of models. The model represents the set of *composed* models in *M* which are *smaller* with respect to the semantic relation $\sqsupseteq$ (i.e., *m* represents a set of models in which every formula satisfied by *m* is valid). The "greater" model can be viewed as holding partial information about the "small" one. Thus, the preorder between two basic elements of the *AGM* is analogous to the subset relation between *AGS* sets. The task of combining partial knowledge about the checked system with full information about one of the components, which is carried out by the intersection operator in the *AGS* scheme, is achieved here by applying the composition operator itself (aided by the restriction transformer).

3.1.3 Implementation

A computer-aided implementation of the *AGM* proof system is expected to carry out automatically the following actions: **a.** Models composition, **b.** Applying the restriction transformer, **c.** Verifying the satisfaction of **L** formulas in *M* models, **d.** Verifying the preorder between models.
It is not difficult to imagine implementable definitions of a composition operator and a restriction transformer which make tasks **a.** and **b.** algorithmic. To automate task **c.** , a global model checker can be applied. The only task which is not clearly decidable even for the finite-state case is the verification of the semantic preorder (task **d.**). Any useful implementation of the *AGM* system should suggest a semantic preorder which is syntactically identified by an efficient algorithm.

In section **4.** we suggest an implementation of *AGM* system in which all four tasks are algorithmically handled. As can be seen there, the need for a decidable preorder affects the choice of the temporal specification language.

3.2 The AGM^* Proof System Scheme

Setting stricter soundness conditions, the AGM^* which is a generalized version of *AGM*, can be defined. The AGM^* enables simultaneous use of partial information on both components of the distributed system. Using this scheme it is possible to conduct a proof, along which only the knowledge that is essential for the verification of the final specification is preserved.

$\mathbf{gm^*[m_1 \,||\, m_2]}$:

Atoms : $\{ m_A \,||\, m_B \;\mid\; m_A \sqsupseteq m_1 \,,\, m_B \sqsupseteq m_2 \}$

Closure operation :

$$\frac{\begin{array}{l} \bar{m}, \bar{\bar{m}} \in gm^*[m_1 \,||\, m_2] \\ m' \sqsupseteq \bar{m}|_{\Sigma_{m_1}} \\ m'' \sqsupseteq \bar{\bar{m}}|_{\Sigma_{m_2}} \end{array}}{m' \,||\, m'' \in gm^*[m_1 \,||\, m_2]}$$

The inference rule :

$$\frac{m \in gm^*[m_1 \| m_2] \qquad \varphi \in \mathrm{L}(\Sigma_m) \qquad \vdash_{MC} m \models \varphi}{\vdash_{AGM^*-m_1 \| m_2} \varphi}$$

In the case where "smaller" ($\sqsupseteq$) models can often be represented by smaller structures, the AGM^* enables further reduction in the size of the structures handled by the proof system (in comparison with AGM). Thus, further reduction in time and space complexity is achieved.

4. ASYNCHRONOUS COMMUNICATING SYSTEMS

4.1 The Model of Computation

We use here a simple computation model for distributed systems. The components of these systems are asynchronous processes, that communicate through synchronized actions. This model is similar to the CCS model [M80]. Our version is restricted to one application of the composition operation (i.e., every model of a compound system is composed of two process models). In this version, compound models are syntactically guaranteed to be deadlock free, i.e., when a composed system is verified using this type of models, all the deadlocked sub-structures of the computation tree of the system are ignored.

4.2 The Specification Language

Defined below is a propositional temporal logic which is a sublanguage of CTL^* [CE81]. Let Σ be the disjoint union of AP_Σ and AC_Σ. The language $CTL^*_{-E}(\Sigma)$ is defined as the smallest set of state formulas such that:

(1) If $A \in AP_\Sigma$, then A and $\neg A$ are state formulas.
(2) If φ and ψ are state formulas, then $\varphi \vee \psi$ and $\varphi \wedge \psi$ are state formulas.
(3) If φ is a path formula, then $\mathrm{A}\varphi$ is a state formula.
(4) If φ is a state formula, then φ is a path formula.
(5) If φ and ψ are path formulas, then $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \,\mathrm{U}\, \psi$, $\mathrm{X}\varphi$ and $\mathrm{G}\varphi$ are path formulas.

Note that in CTL^*_{-E} negations are applied only to atomic formulas. Thus, the path quantifier **E** ("for some computation path") is eliminated *semantically* from CTL^*_{-E}. The semantics of $CTL^*_{-E}(\Sigma_m)$ is defined in the usual way, with respect to the Kripke structure K_m [HC77], generated from the (process or system) model m by ignoring the model's transition labels.
The satisfaction of a state formula $\varphi \in CTL^*_{-E}(\Sigma)$ in a model m (with s_0 the initial state), denoted $m \models \varphi$, is defined by :

(1) If $\varphi \in CTL^*_{-E}(\Sigma_m)$ then $m \models \varphi$ iff $m, s_0 \models \varphi$.

(2) If $\varphi \notin CTL^*_{-E}(\Sigma_m)$ then $m \not\models \varphi$.

We say that one language is *more expressive* then the other if there exists a formula in the one that does not have an equivalent formula in the other. Let $\succ$ denote "more expressive" and let CTL_{-E} be $CTL \cap CTL^*_{-E}$. Comparing CTL^*_{-E} with CTL^*, CTL and LTL

[CG87] we get the following lemma :

Lemma 3: (a) $CTL^* \succ CTL^*_{-E}$; (b) $CTL^*_{-E} \succ CTL_{-E}$; (c) $CTL^*_{-E} \succ LTL$.

We can now choose a model checking procedure $\vdash_{\overline{MC}}$, needed in our modular proof systems (*AGM* and *AGM**). As CTL^*_{-E} is a sub language of CTL^* there is a model checking algorithm which is exponential in the size of the specification and linear in the size of the Kripke structure (i.e., $O(2^{|\varphi|}(|S|+|R|))$) [EL85]. One may prefer to restrict the specifications to CTL_{-E}, gaining a linear model checking algorithm (i.e., $O(|\varphi|(|S|+|R|))$) [CES86].

4.3 The Preorder

The choice of CTL^*_{-E} as a specification language is motivated by a unique semantic quality: The validity of CTL^*_{-E} formulas is preserved over "*pruned*" computation trees. A description of this property is presented below. In this context, we view the semantics of CTL^*_{-E} as defined over computation trees. The originating transition graphs (if exist) of these trees are omitted. Let T be a computation tree, and $\varphi \in CTL^*_{-E}$ be a temporal formula. $T,v \models \varphi$ denotes that φ is satisfied at node v in T. A *branching node* in T is a node that has at list two immediate descendants. Eliminating a subtree of T rooted at v, such that v is an immediate descendant of a branching node, is defined as *pruning T at v*.

Theorem 5: If T' is generated by pruning T at nodes which are descendants of v, then

$$\forall \varphi \in CTL^*_{-E} \;\; [\;\; T,v \models \varphi \;\Rightarrow\; T',v \models \varphi \;\;].$$

Long [L89] defined a syntactic preorder over Kripke structures, which partially identifies the *pruning relation* of their computation trees. Basically, we adopt his relation as our choice of the required semantic preorder $\sqsupseteq$. Assimilated in our system, the definition is enhanced to encompass more cases for which the semantic property holds, including more cases for which the pruning relation holds. In addition, it is modified to consider transition labels and to verify the containment of the atomic proposition sets.

Let $m = \langle AC,AP,S,R,s_0,L\rangle$ and $m' = \langle AC',AP',S',R',s'_0,L'\rangle$ be either both process models, or both system models, such that $AP \subseteq AP'$. A sequence of relations $\boldsymbol{F}_i$, such that $\forall i \geq 0 \;\; \boldsymbol{F}_i \subseteq S \times S'$, is defined as follows:

For all $s \in S$, $s' \in S'$

1. $s\,\boldsymbol{F}_0\, s'$ iff $L(s) = L'(s') \cap AP$.
2. $s\,\boldsymbol{F}_{n+1}\, s'$ iff -
   2.1 $s\,\boldsymbol{F}_n\, s'$.
   2.2 $\forall s'_1 \in S'$, for all action α $[\, s' \xrightarrow{\alpha} s'_1 \;\Rightarrow\; \exists s_1 \in S\, [\, s \xrightarrow{\alpha} s_1 \;\wedge\; s_1\, \boldsymbol{F}_n\, s'_1 \,]]$.

$\boldsymbol{F} \subseteq S \times S'$ is defined by: For all $s \in S$, $s' \in S'$, $s\,\boldsymbol{F}\, s'$ iff $\forall i \geq 0 \;\; s\,\boldsymbol{F}_i\, s'$.

Let $m = \langle AC,AP,S,R,s_0,L\rangle$ and $m' = \langle AC',AP',S',R',s'_0,L'\rangle$. The relation $\geq$ over the model domain M is defined as follows:

$m \geq m'$ iff -

i. m and m' are either both process models, or both system models.

ii. $AP \subseteq AP'$.

iii. $s_0\, \boldsymbol{F}\, s'_0$.

Theorem 6: If $m_1 \geq m_2$ then for every Σ, and for every formula $\varphi \in CTL^*_{-E}(\Sigma)$, $m_1 \models \varphi \Rightarrow m_2 \models \varphi$.

There is an algorithm to verify that two models are preordered, which is polynomial-time in their size. Using $\geq$ as the required syntactically-identified semantic preorder, note that "greater" ($\geq$) models are, in general, expressed by smaller structures. The implications of this property were previously discussed.

Theorem 7: The *AGM* (and AGM^*) soundness conditions are satisfied by the model of computation, the language CTL^*_{-E} , and the preorder $\geq$.

Establishing that, the presentation of a semi automatic model checking system derived from these schemes, is completed. In [SG90] we apply these proof rules to a program.

REFERENCES

[AL89] M. Abadi and L. Lamport, "Composing Specifications", REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness, Mook, May 1989.

[CE81] E.M. Clarke, and E.A. Emerson, "Synthesis of Synchronization Skeletons for Branching Time Temporal Logic", Proc. of Workshop on Logic of Programs, Yorktown-Heights, 1981.

[CES86] 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, pp. 244-263, 1986.

[CG87] E.M. Clarke, O. Grumberg, "Research on automatic verification of finite-state concurrent systems", Annual Reviews of Computer Science, Vol. 2, 269-290, 1987 (J.F. Traub, editor).

[CLM89] E.M. Clarke, D.E. Long, and K.L. McMillan, "Compositional Model Checker". Proc. of the 4th IEEE Symp. on Logic in Computer Science, Asilomar, June 1989.

[EL85] E.A. Emerson, and C. Lei, "Modalities for Model Checking: Branching Time Strikes Back", 12th Symposium on Principles of Programming Languages, New Orleans, La., January 1985.

[HC77] G.E. Hughes, and M.J. Creswell, "An introduction to Modal Logic", London: Methuen, 1977.

[J87] B. Josko, "MCTL - An Extension of CTL for Modular Verification of Concurrent Systems". Workshop on Temporal Logic, (H. Barringer, ed.) University of Manchester, April 1987, LNCS 398, pp. 165-187.

[L89] D.E. Long, Private communication.

[M80] R. Milner, A Calculus of Communicating Systems, Springer Lecture Notes on Computer Science, Vol. 92, 1980.

[P85] A. Pnueli, "In Transition from Global to Modular Temporal Reasoning about Programs". Logics and Models of Concurrent Systems, (K. Apt, ed.), Vol. 13 of NATO ASI Series F: Computer and System Sciences, Springer-Verlag, 123-144, 1985.

[SG90] G. Shurek and O. Grumberg, "THE MODULAR FRAMEWORK OF COMPUTER-AIDED VERIFICATION Motivation, Solutions, and Evaluation Criteria", Workshop on Computer-Aided Verification, Rutgers, NJ., June 1990, to appear in ACM/AMS DIMACS series.