Compositional Model Checking for Linear-Time Temporal Logic

Roope Kaivola
University of Helsinki, Department of Computer Science
Teollisuuskatu 23, SF-00510 Helsinki, Finland
tel. +358-0-708 4163, fax. +358-0-708 4441
email rkaivola@cc.helsinki.fi

Abstract. *Temporal logic model checking is a useful method for verifying properties of finite-state concurrent systems. However, due to the state explosion problem modular methods are essential in making the verification task manageable. One such method is to verify that certain properties ϕ_i are true of the submodules M_i of the system in all environments, and that the required property ϕ is a logical implication of these. This paper presents an algorithm deciding whether a nexttime-less linear temporal logic formula ϕ is true of a distributed variable module M in all environments. There are two versions of the algorithm: one allowing no fairness requirements and one for strongly fair concurrency. Both versions run in time $O(|M| \cdot 2^{c \cdot |\phi|})$. In addition to presenting the algorithms it is shown that given some reasonable assumptions the method is complete in the sense that all formulas ϕ true of $M_1 \| M_2$ can be verified by it.*

1 Introduction

One of the most important approaches to practical verification of propositional temporal logic properties of finite-state programs is automated model-checking: the execution of a program is modelled by a finite graph which can be directly interpreted as a temporal logic model, and a model-checking algorithm for the appropriate temporal logic can be run on the model. For many propositional temporal logics the model checking algorithms are of relatively low time-complexity, e.g. linear in size of the model and singly exponential in the size of the formula for the standard linear temporal logic [LP85].

However, despite the low time-complexity of model checking, the size of the execution graph is still often a prohibitive factor. An essential reason for this state-explosion problem is the modelling of concurrency by arbitrary interleavings of the atomic actions of the concurrent processes. In the general case the size of the complete model is exponential in the number of concurrent processes.

One way to avoid this problem is verifying the system directly on the basis of the individual processes without constructing a global execution graph. Here we follow the approach advocated by [MP91]. In this method a system M consists of several concurrent modules $M_1 \| \ldots \| M_n$. Verifying that ϕ is true of M is done by finding lemmas $\phi_1, \ldots, \phi_n$ such that each ϕ_i holds of M_i in every possible environment and $\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow \phi$ is a theorem.

In this paper we describe an algorithm deciding whether a nexttime-less temporal logic formula ϕ is true of a module M in every possible environment. There are two versions of the algorithm: the basic case in which the concurrency is modelled by potentially unfair interleavings, and the enhanced case with fair concurrency. Both versions run in a time $O(|M|\cdot 2^{c\cdot|\phi|})$. In other words, using this algorithm it costs no more to decide whether ϕ_i is true of M_i in all (fair) environments than it costs in the standard linear temporal logic model checking [LP85] to decide whether ϕ_i is true of M_i without any environment. The method of communication between processes is by distributed variables. These are shared variables with the restriction that only the process owning a variable may change its value.

We do not discuss the issue of how the lemmas ϕ_i are to be found, and leave it to the responsibility of the human verifier of a system. However, in Section 5 we prove that given certain reasonable assumptions the lemmas always exist. This means that the method is complete in the sense that every formula ϕ true of $M_1 \| M_2$ is, at least in theory, verifiable by it.

The modularity issues discussed here have been addressed by several researchers. [BKP84], [Pnu85] and [Bar86] describe compositional linear temporal logic semantics for distributed-variable systems similar to ours. However, their point of interest lies more in the axiomatic semantics and compositional proof rules than in presenting an explicit algorithm for modular verification. A method for compositional model-checking using *MCTL*, an extension of *CTL* allowing restricted linear temporal logic formulas and fairness requirements as assumptions, is discussed in [Jos89]. A modular model checking algorithm for a logic tailored for Petri-nets is presented in [BE91]. It is unclear whether either of these methods can be applied to the model and the linear temporal logic used here. An approach to CTL^* model checking in which the environment of a module is modelled by an additional interface module is presented in [CLM89]. This differs from our approach in the use of interface processes and in the underlying communication method. The problem of modular verification of communicating Moore-machines is addressed by [GL91]. However, it does not seem possible to transfer their results directly into an asynchronous distributed-variable model.

The paper proceeds as follows: we first recall some standard definitions and properties of nexttime-less linear temporal logic and present the concepts of a module and parallel composition of modules. Then we define the core concept of a ***satisfiability graph*** and show that in a sense it encodes all the necessary information about the behaviour of a module in any environment. To illustrate the approach in practice, a mutual exclusion protocol is verified as an example. In Section 5 the results are extended to deal with fair concurrency.

2 Linear temporal logic

In this section we recall the definitions of standard nexttime-less linear temporal logic.

Definition 2.1 The *alphabet of propositional linear temporal logic LTL'* consists of the set AP of atomic propositions and of the symbols $(,), \neg, \vee, \mathcal{U}$. The *well-formed formulas* (wffs) of LTL' are as follows:

- if $\phi \in AP$, then ϕ is a wff,
- if ϕ_1 and ϕ_2 are wffs, then $(\neg\phi_1)$, $(\phi_1 \vee \phi_2)$ and $(\phi_1 \mathcal{U} \phi_2)$ are wffs, and
- there are no other wffs.

We use the abbreviations $\top \equiv_{df} (p \vee (\neg p))$ for some fixed $p \in AP$, $(\phi_1 \wedge \phi_2) \equiv_{df} (\neg((\neg\phi_1) \vee (\neg\phi_2)))$, $(\phi_1 \Rightarrow \phi_2) \equiv_{df} ((\neg\phi_1) \vee \phi_2)$, $(\Diamond\phi) \equiv_{df} (\top\mathcal{U}\phi)$, $(\Box\phi) \equiv_{df}$

$(\neg(\Diamond(\neg\phi)))$, $(\phi_1 \mathcal{U}_w \phi_2) \equiv_{df} ((\phi_1 \mathcal{U} \phi_2) \vee (\Box\phi_1))$, and the ordinary precedence rules to reduce the number of parentheses. □

Definition 2.2 A *truth set* v is a set of atomic propositions, $v \subseteq AP$. A *truth set sequence* σ is a finite or infinite sequence of truth sets, $\sigma = (v_1, v_2, \ldots)$. If σ is a truth set sequence, σ_n is the n:th element of σ, $\sigma^{(n)}$ is the truth set sequence obtained by leaving the first n elements out of σ, and $\sigma \cap A$ is the truth set sequence $(v_1 \cap A, v_2 \cap A, \ldots)$. □

Definition 2.3 A *Kripke-model* K is an ordered 4-tuple $K = (S, I, R, V)$, where S is the set of *states*, $I \subseteq S$ is the set of *initial states*, $R \subseteq S \times S$ is the *transition relation*, and $V : S \rightarrow P(AP)$ is a *valuation* expressing the atomic formulas true in a state. ($P(X)$ denotes the powerset of X.)

A *path* p in a Kripke-model K is a finite or infinite sequence $p = (s_1, s_2, \ldots)$ such that $s_1 \in I$, and each $(s_i, s_{i+1}) \in R$. A path p is a *fullpath* iff either p is infinite or there is no further state reachable from the last state of p, i.e. if s_n is the last state of p then for all s, $(s_n, s) \notin R$. The truth set sequence corresponding to a fullpath $p = (s_1, s_2, \ldots)$, denoted $V(p)$, is $(V(s_1), V(s_2), \ldots)$. □

Next we augment Kripke-models with fairness constraints expressed in terms of transition sets [LP85]. The notion of fairness used is the so-called strong fairness, meaning that if an event is possible infinitely often it has to be realised infinitely often, too.

Definition 2.4 A *fair Kripke-model* K_f is an ordered 5-tuple $K_f = (S, I, R, V, F)$, where (S, I, R, V) is a Kripke-model and F is a finite set of fairness sets $F = \{f_1, \ldots f_n\}$ such that $\bigcup_{f_i \in F} f_i = R$.

The paths and fullpaths of K_f are those of the Kripke-model (S, I, R, V). A fairness set $f_i \in F$ is *enabled* in a state $s \in S$ iff there is an s' such that $(s, s') \in f_i$. A fairness set f_i is *active* in a transition (s, s') iff $(s, s') \in f_i$. A fullpath p is *fair* iff every fairness set f_i that is enabled in infinitely many states of p is active in infinitely many transitions of p, too. □

Definition 2.5 An LTL'-formula ϕ is true in a truth set sequence $\sigma = (v_1, v_2, \ldots)$ i.e. $\sigma \models \phi$, according to the following rules:

- If $\phi \in AP$, then $\sigma \models \phi$ iff $\phi \in v_1$.
- $\sigma \models \neg\phi$ iff not $\sigma \models \phi$.
- $\sigma \models \phi_1 \vee \phi_2$ iff $\sigma \models \phi_1$ or $\sigma \models \phi_2$.
- $\sigma \models \phi_1 \mathcal{U} \phi_2$ iff $\exists : 0 \leq i < |\sigma|$, such that $\sigma^{(i)} \models \phi_2$ and for all $0 \leq j < i$, $\sigma^{(j)} \models \phi_1$.

If ϕ is true in every truth set sequence σ we say that ϕ is a theorem and write $\models \phi$. An LTL'-formula ϕ is true in a (fair) Kripke-model K, denoted $K \models \phi$, iff $V(p) \models \phi$ for every (fair) fullpath p of K. □

The operators have their conventional meanings. The reflexivity of $\mathcal{U}$ and the lack of a nexttime-operator allow us to overlook truth sets in a truth set sequence if they do not differ from their predecessor.

Definition 2.6 Let $\sigma = (v_1, v_2, \ldots)$ be a truth set sequence. The *reduced form of σ*, denoted by $red(\sigma)$ is constructed by removing from σ all v_i such that $v_i = v_{i-1}$. If $A \subseteq AP$ and $red(\sigma \cap A) = red(\sigma' \cap A)$ we say that σ and σ' are *stuttering equivalent modulo A* and write $\sigma \approx \sigma'$ (mod A). □

Proposition 2.7 Let $A \subseteq AP$ and σ, σ' be truth set sequences such that $\sigma \ddot{\approx} \sigma'$ (mod A) and ϕ a formula containing only atomic propositions in A. Then $\sigma \models \phi$ iff $\sigma' \models \phi$.
Proof: Induction on the structure of ϕ [Lam83]. □

Corollary 2.8 Let $A \subseteq AP$ and ϕ be a formula containing only atomic propositions in A. Let K and K' be (fair) Kripke-models such that for every (fair) fullpath p in K there is a (fair) fullpath p' in K' such that $V(p) \ddot{\approx} V(p')$ (mod A) and vice versa for all (fair) fullpaths p' in K'. Then $K \models \phi$ iff $K' \models \phi$. □

3 Modules

In this section we give the formal definitions of modules and the parallel composition operator. Each module M has a set of its own variables that cannot be modified by the environment of M. In addition to its own variables a module may refer to a set of external variables and base decisions about its behaviour on their values. As in [MP81] a module is modelled by a directed graph, the states of which correspond to the execution states and the transitions to atomic actions. Here this model is simplified by allowing only boolean variables, by labelling the states of the model with the values of module's own variables, by dropping the assignment labels and by allowing only transition conditions consisting of a single proposition or its negation.

Definition 3.1 Let $A \subseteq AP$. By $\neg A$ we denote the set $\{\neg a \mid a \in A\}$, and by $L(A)$ the set $L(A) = A \cup \neg A \cup \{\top\}$, where $\top$ is an element not in A or $\neg A$. If $a \in L(A)$ and $A' \subseteq A$, we write $A' \models a$ iff either $a = \top$ or $a \in A'$ or $a = \neg a'$ where $a' \in A$ and $a' \notin A'$. □

Definition 3.2 A *module* M is an ordered 6-tuple (A_o, A_e, S, I, R, V), where

- $A_o \subseteq AP$ is the set of atomic propositions owned by module M,
- $A_e \subseteq AP \setminus A_o$ is the set of external atomic propositions visible to M,
- S is the set of states,
- $I \subseteq S$ is the set of initial states,
- $V : S \rightarrow P(A_o)$ is the truth valuation of module's own propositions, and
- $R \subseteq S \times L(A_e) \times S$ is the set of transitions. Each transition is labelled by a transition label which can be either trivially true $\top$, an external atomic proposition or a negation of such. If $(s_1, l, s_2) \in R$, we write $s_1 \xrightarrow{l} s_2 \in R$. □

Definition 3.3 A *fair module* M_f is an ordered 7-tuple $M_f = (A_o, A_e, S, I, R, V, F)$, where (A_o, A_e, S, I, R, V) is a module and F is a finite set of fairness sets $F = \{f_1, \ldots f_n\}$ such that $\bigcup_{f_i \in F} f_i = R$. A *basic fair module* M_f is a fair module such that $F = \{R\}$. □

Modules may be combined using binary parallel composition $\|$ which can be generalised to deal with n modules in the standard fashion. Concurrency is represented by interleaving the atomic actions of the submodules. If a transition in one of the submodules is labelled by an atomic proposition owned by the other submodule, a corresponding transition in the combined module can be taken only if the proposition evaluates to true in the initial state of the transition.

Definition 3.4 Let $M_1 = (A_{o1}, A_{e1}, S_1, I_1, R_1, V_1)$ and $M_2 = (A_{o2}, \ldots, V_2)$ be modules such that $A_{o1} \cap A_{o2} = \emptyset$. The *parallel composition* of them, $M_1 \parallel M_2$, is the module (A_o, A_e, S, I, R, V) defined as follows:

- $A_o = A_{o1} \cup A_{o2}$
- $A_e = (A_{e1} \cup A_{e2}) \setminus A_o$
- $S = S_1 \times S_2$
- $I = I_1 \times I_2$
- $R = \{(s_1,t) \xrightarrow[\top]{l} (s_2,t) \mid s_1 \xrightarrow{l} s_2 \in R_1 \text{ and } l \in L(A_e)\} \cup$
  $\{(s_1,t) \xrightarrow{\top} (s_2,t) \mid \exists l : s_1 \xrightarrow{l} s_2 \in R_1 \text{ and } l \in L(A_{o2}) \text{ and } V_2(t) \models l\} \cup$
  $\{(s,t_1) \xrightarrow[\top]{l} (s,t_2) \mid t_1 \xrightarrow{l} t_2 \in R_2 \text{ and } l \in L(A_e)\} \cup$
  $\{(s,t_1) \xrightarrow{\top} (s,t_2) \mid \exists l : t_1 \xrightarrow{l} t_2 \in R_2 \text{ and } l \in L(A_{o1}) \text{ and } V_1(s) \models l\}$
- $V(s_1, s_2) = V_1(s_1) \cup V_2(s_2)$ □

In the parallel composition of fair modules the fairness sets of the submodules are propagated to the resulting module. If a system is modelled by a parallel composition of basic fair modules, the definition guarantees that the fair fullpaths of the complete system are exactly those that are strongly fair with respect to each module.

Definition 3.5 Let $M_{f1} = (A_{o1}, A_{e1}, S_1, I_1, R_1, V_1, \{f_{11}, \ldots, f_{1n_1}\})$ and $M_{f2} = (A_{o2}, \ldots, V_2, \{f_{21}, \ldots, f_{2n_2}\})$ be fair modules. Then $M_{f1} \parallel M_{f2}$, is the fair module $(A_o, \ldots, V, \{f_1, \ldots, f_{n_1+n_2}\})$ where $(A_o, \ldots, V) = (A_{o1}, \ldots, V_1) \parallel (A_{o2}, \ldots, V_2)$ and for $i = 1, \ldots, n_1$

$f_i = \{(s_1,t) \xrightarrow[\top]{l} (s_2,t) \in R \mid s_1 \xrightarrow{l} s_2 \in f_{1i} \text{ and } l \in L(A_e)\} \cup$
$\{(s_1,t) \xrightarrow{\top} (s_2,t) \in R \mid \exists l : s_1 \xrightarrow{l} s_2 \in f_{1i} \text{ and } l \in L(A_{o2}) \text{ and } V_2(t) \models l\}$

and for $i = 1, \ldots, n_2$

$f_{i+n_1} = \{(s,t_1) \xrightarrow[\top]{l} (s,t_2) \in R \mid t_1 \xrightarrow{l} t_2 \in f_{2i} \text{ and } l \in L(A_e)\} \cup$
$\{(s,t_1) \xrightarrow{\top} (s,t_2) \in R \mid \exists l : t_1 \xrightarrow{l} t_2 \in f_{2i} \text{ and } l \in L(A_{o1}) \text{ and } V_1(s) \models l\}$ □

The availability of a transition in a module that does not have any nontrivial transition conditions cannot be influenced by its environment. Therefore, we treat such a module as corresponding to a complete system, and interpret formulas in the naturally induced Kripke-model.

Definition 3.6 Let $M = (A_o, A_e, S, I, R, V)$ be a module. M is *closed* iff $A_e = \emptyset$ and the *Kripke-model* $K(M)$ *corresponding to the closed module* M is defined as $K(M) = (S, I, R_K, V)$, where $(s_1, s_2) \in R_K$ iff $s_1 \xrightarrow{\top} s_2 \in R$.

Let $M_f = (A_o, \ldots, V, \{f_1, \ldots, f_n\})$ be a fair module. M_f is closed iff $A_e = \emptyset$ and the fair Kripke-model $K(M_f)$ corresponding to M_f is defined as $K(M_f) = (S, I, R_K, V, \{f_{K1}, \ldots, f_{Kn}\})$, where R_K is as above and for all $1 \leq i \leq n$: $(s_1, s_2) \in f_{Ki}$ iff $s_1 \xrightarrow{\top} s_2 \in f_i$.

A formula ϕ is true of a closed (fair) module M, $M \models \phi$, iff $K(M) \models \phi$. □

4 Compositional model checking

The basic method of compositional verification applied in this paper is that of [MP91]: when verifying that ϕ holds of $M_1 \parallel M_2$, find ϕ_1 and ϕ_2 such that ϕ_i holds of M_i in any environment and $\models \phi_1 \wedge \phi_2 \Rightarrow \phi$.

Definition 4.1 A formula ϕ is *modularly true* of M iff $M \parallel M' \models \phi$ for all modules M' such that $M \parallel M'$ is closed. If this is the case we write $M \overset{m}{\models} \phi$. □

The correctness of the verification method is asserted by the following result, which can be generalised to n processes.

Proposition 4.2 If $M_1 \overset{m}{\models} \phi_1$ and $M_2 \overset{m}{\models} \phi_2$, then $M_1 \| M_2 \overset{m}{\models} \phi_1 \wedge \phi_2$. □

The aim now is to find a method for verifying that $M \overset{m}{\models} \phi$. This can be done by constructing a Kripke-model K on the basis of M so that $M \overset{m}{\models} \phi$ iff $K \models \phi$ and by applying the standard linear temporal logic model checking algorithm [LP85] to check whether $K \models \phi$ holds. A straighforward approach is to construct a single environment of M so that it exhibits all the possible behaviours that any environment of M can produce and to combine this environment with M.

Definition 4.3 Let $M = (A_o, \dots, V)$ be a module. The ***chaotic environment*** of M, $ce(M)$ is the module $(A_e, \emptyset, P(A_e), P(A_e), P(A_e) \times P(A_e), V')$, where $V'(s) = s$. □

Proposition 4.4 Let $M = (A_o, \dots, V)$ be a module and ϕ a formula such that every atomic proposition occurring in ϕ is in $A_o \cup A_e$. Then $M \overset{m}{\models} \phi$ iff $M \| ce(M) \models \phi$.
Proof: If $M \overset{m}{\models} \phi$, then $M \| ce(M) \models \phi$ by the definition of $\overset{m}{\models}$. If $M \overset{m}{\not\models} \phi$, then there is an M' and a path p_1 in $K_1 = K(M \| M')$ such that $V_1(p_1) \not\models \phi$ holds. If $p_1 = ((s_1, t_1), (s_2, t_2), \dots)$, then $p_2 = ((s_1, V_1(t_1) \cap A_e), (s_2, V_1(t_2) \cap A_e), \dots)$ is a path in $K_2 = K(M \| ce(M))$. What is more, $V_1(p_1) \overset{st}{\approx} V_2(p_2) \pmod{A_o \cup A_e}$. By 2.7, $V_2(p_2) \not\models \phi$ which implies $M \| ce(M) \not\models \phi$. □

Please note that the set A_e can always be extended so that every atomic proposition occurring in ϕ is in $A_o \cup A_e$. Therefore, without loss of generality we suppose that this is the case in the following.

Checking whether $M \overset{m}{\models} \phi$ can be done by checking that $M \| ce(M) \models \phi$. This is exactly the method proposed in [MP91], where a transition system equivalent to $K(M \| ce(M))$ is denoted by S_M. As $|M \| ce(M)| = |M| \cdot 2^{|A_e|}$, the model checking will take $O(|M| \cdot 2^{|A_e|} \cdot 2^{c \cdot |\phi|})$ time. This is sensible only if $|A_e| < log(|M'|)$, where M' is the actual environment in which M is to work.

In this naive approach we are, in fact, doing a lot of unnecessary work. What we are actually interested in is the validity of ϕ in all the fullpaths of all the systems consisting of M and an environment M'. The only influence of the external atomic propositions not occurring in ϕ is the availability of some transitions in some states and, consequently, the existence or nonexistence of some fullpaths. However, if a certain transition is labelled by an external atomic proposition, there is always both an environment in which the transition is disabled and an environment in which the transition is enabled. The first possibility is always present, anyway, since an infinite execution of the environment unfair to M might prevent M from taking a transition even if it were enabled. Therefore, we do not need to pay any attention to whether transition conditions not occurring in ϕ are true or false, and we may simply drop them.

Definition 4.5 Let $M = (A_o, \dots, V)$ be a module and $A \subseteq A_o \cup A_e$. The ***satisfiability graph*** corresponding to M and A, $sg(M, A)$, is the Kripke-model (S_g, I_g, R_g, V_g) where

- $S_g = S \times P(A \setminus A_o)$
- $I_g = I \times P(A \setminus A_o)$
- $R_g = \{((s, A_1), (s, A_2)) \in S_g \times S_g\} \cup$
  $\{((s_1, A_1), (s_2, A_1)) \in S_g \times S_g \mid \exists l : s_1 \xrightarrow{l} s_2 \in R \text{ and } l \notin L(A) \text{ or } A_1 \models l\}$

- $V_g(s, A_1) = V(s) \cup A_1$ □

Proposition 4.6 Let $M = (A_o, \ldots, V)$ be a module, $A \subseteq A_o \cup A_e$ and ϕ a formula containing only atomic propositions from A. Then $M \models^{m} \phi$ iff $sg(M, A) \models \phi$.
Proof: By 4.4, $M \models^{m} \phi$ iff $M \| ce(M) \models \phi$. Here it is shown that $M \| ce(M) \models \phi$ iff $sg(M, A) \models \phi$.

Let us denote $M \| ce(M)$ by $M_c = (A_{co}, A_{ce}, S_c, I_c, R_c, V_c)$ and $sg(M, A)$ by $K_g = (S_g, I_g, R_g, V_g)$. The result is obtained by showing that given any fullpath p in $K(M_c)$ there is a fullpath p' in K_g such that $V_c(p) \approx V_g(p')$ (mod A) and vice versa, and by applying 2.8.

Assume that $K(M_c)$ has a fullpath p. From the definitions of $K()$ and M_c it is known that p is of the form $p = ((s_1, A_1), (s_1, A_2), \ldots)$, where each $s_i \in S$, $A_i \in P(A_e)$, and either $s_i = s_{i+1}$ or $A_i = A_{i+1}$ and there is an l such that $s_i \xrightarrow{l} s_{i+1} \in R$ and $A_i \models l$. If $s_i = s_{i+1}$, it is clear that $((s_i, A_i \cap A), (s_{i+1}, A_{i+1} \cap A)) \in R_g$. If $s_i \neq s_{i+1}$, then $A_i = A_{i+1}$ and either $l \notin L(A)$ or $l \in L(A)$ and $A_i \cap A \models l$. In both cases $((s_i, A_i \cap A), (s_{i+1}, A_{i+1} \cap A)) \in R_g$ follows. As $(s_1, A_1 \cap A) \in I_g$, $p' = ((s_1, A_1 \cap A), (s_2, A_2 \cap A), \ldots)$ is a fullpath in K_g. What is more, $V_c(p) \approx V_g(p')$ (mod A) since $V_c((s_i, A_i)) \cap A = (V(s_i) \cup A_i) \cap A = V_g((s_i, A_i \cap A)) \cap A$.

Assume now that K_g has a fullpath p. From the definition of K_g it is known that $p = ((s_1, A_1), (s_2, A_2), \ldots)$ where for each $((s_i, A_i), (s_{i+1}, A_{i+1}))$ either $s_i = s_{i+1}$ or $A_i = A_{i+1}$ and there is an l such that $s_i \xrightarrow{l} s_{i+1} \in R$ and either $A_i \models l$ or $l \notin L(A)$. If $s_i = s_{i+1}$, it is clear that $(s_i, A_i) \xrightarrow{T} (s_{i+1}, A_{i+1}) \in R_c$. If $A_i = A_{i+1}$ and $A_i \models l$, $(s_i, A_i) \xrightarrow{T} (s_{i+1}, A_{i+1}) \in R_c$, again. If $A_i = A_{i+1}$ and $l \notin L(A)$ and l is of the form $\neg l_1$, $l_1 \notin A_i$ which implies that $A_i \models l$. Consequently, $(s_i, A_i) \xrightarrow{T} (s_{i+1}, A_{i+1}) \in R_c$. Finally, if $A_i = A_{i+1}$ and $l \notin L(A)$ and l is not of the form $\neg l_1$, R_c has the transitions $(s_i, A_i) \xrightarrow{T} (s_i, A_i \cup \{l\}) \xrightarrow{T} (s_{i+1}, A_{i+1} \cup \{l\}) \xrightarrow{T} (s_{i+1}, A_{i+1})$. Therefore, for each transition of p in K_g we can construct a sequence of transitions in R_c so that they contain the same initial and final states. The required path p' in $K(M_c)$ is constructed by joining these sequences together. By the structure of p', $V_g(p) \approx V_c(p')$ (mod A) as well. □

By proposition 4.6 we can check whether $M \models^{m} \phi$ by taking A as the set containing only the atomic propositions in ϕ and by checking whether $sg(M, A) \models \phi$. Noticing that $|sg(M, A)| = |M| \cdot 2^{|A \setminus A_o|}$ and that $|A \setminus A_o|$ is limited by $|\phi|$, we acquire an upper limit $|M| \cdot 2^{|\phi|}$ to the size of the satisfiability graph. Since the satisfiability graph can be constructed in a time linear to its size, the model checking takes $O(|M| \cdot 2^{|\phi|} \cdot 2^{c \cdot |\phi|})$, i.e. $O(|M| \cdot 2^{c \cdot |\phi|})$ time. The total time requirement of the method consists of three parts: verifying that $M_1 \models^{m} \phi_1$, which takes $O(|M_1| \cdot 2^{c \cdot |\phi_1|})$ time, verifying that $M_2 \models^{m} \phi_2$, which takes $O(|M_2| \cdot 2^{c \cdot |\phi_2|})$ time, and verifying that $\models \phi_1 \wedge \phi_2 \Rightarrow \phi$, which takes $O(2^{c \cdot (|\phi_1| + |\phi_2| + |\phi|)})$ time [LPZ85]. Assuming $|\phi_1| \approx |\phi_2|$, the total time needed is thus $O((|M_1| + |M_2| + 2^{c \cdot |\phi|}) \cdot 2^{c \cdot |\phi_1|})$. Furthermore, if $|\phi| \approx |\phi_1|$, this is $O((|M_1| + |M_2|) \cdot 2^{c \cdot |\phi|})$.

This should be contrasted with the time requirement $O(|M_1| \cdot |M_2| \cdot 2^{c \cdot |\phi|})$ of the simple approach of just constructing the whole state space of the system and then running the model checker on it. If M_1 and M_2 are large and the required lemmas ϕ_1 and ϕ_2 relatively short, the compositional method presented here can therefore yield substantial savings. As an additional note, the restriction that transitions are labelled by single propositions or their negations can be dropped in favour of arbitrary formulas in the disjunctive normal form without affecting the time requirement.

```
Process $P_i$:
loop forever
    /* idle */
    $r_i := \top$
R:  for $j := 1$ to $i - 1$
        if $r_j$ then
            $r_i := \bot$
            wait until $\neg r_j$
            goto R
        end if
    end for
    for $j := i + 1$ to $n$
        wait until $\neg r_j$
    end for
C:  /* critical */
    $r_i := \bot$
end loop
```

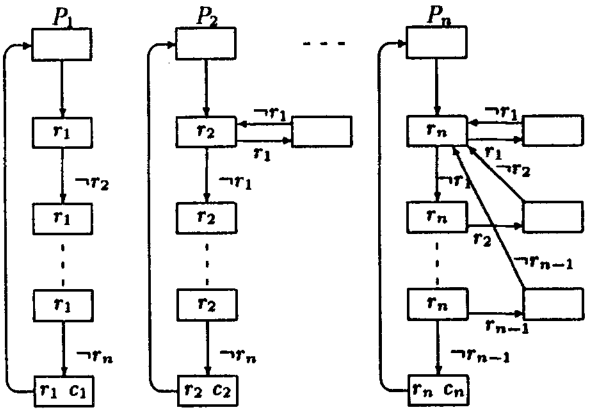

Figure 1: Mutual exclusion protocol

Although no systematic way of finding suitable lemmas is developed in this paper, the issue is naturally very important for real-world applications. Ideally, when creating a concurrent system the designer sets some requirements to each module so that the requirements to the complete system are met if the requirements to the individual modules are met. These requirements naturally form a good basis for the required lemmas. The applications of the method are best illustrated by an example.

Example 4.7 Figure 1 contains a textual description of a mutual exclusion protocol and the corresponding module graphs. Each state in the picture is labelled by the atomic propositions true in that state. The only propositions shown are r_i, which has the meaning *process i requests access to the critical section*, and c_i meaning *process i is in the critical section.* For each module P_i, $A_{oi} = \{r_i, c_i\}$ and $A_{ei} = \{r_j \mid 1 \leq j \leq n, j \neq i\}$. An obvious requirement to the system is that two processes are never together in the critical section, i.e. that $\bigwedge_{i \neq j} \Box(\neg c_j \vee \neg c_i)$.

If we analyse a system consisting of n such processes, the number of global states is $O(n^n)$. This means that e.g. for $n = 32$ the number of global states would be in the range 2^{160}, which rules out straightforward model checking.

Let us denote by ϕ the formula $\Box(\neg c_j \vee \neg c_i)$ for some fixed i and j. If we can show that ϕ holds for all $i \neq j$, the required result follows. As lemma ϕ_1 we may take the formula $\neg r_i \wedge \Box(c_i \Rightarrow r_i) \wedge \Box(\neg r_i \Rightarrow (\neg c_i)\mathcal{U}_w(r_i \wedge \neg r_j))$ and as lemma ϕ_2 the same formula with i:s and j:s reversed. It is easy to check that $\models \phi_1 \wedge \phi_2 \Rightarrow \phi$.

If we tried to verify that $P_i \overset{m}{\models} \phi_1$ using the naive method of 4.4, the size of the resulting Kripke-model would have a minimum of 2^{36} states. Even this would very likely be infeasible.

When verifying that $P_i \overset{m}{\models} \phi_1$ using the satisfiability graph construction, the only external proposition that we need to keep track of is r_j, and consequently the satisfiability graph has a maximum of 128 states. The same applies to verifying that $P_j \overset{m}{\models} \phi_2$. The advantages of the method are clear in this case. $\Box$

5 Fair concurrency

Despite the advantages of the verification method presented in the previous section, it still suffers from an important drawback: when two modules are combined in parallel, the resulting system in not required to be fair with respect to both modules. One counter-intuitive result of this is that even if M is a closed module, $M \models \phi$ does not necessarily imply $M \overset{\mathrm{m}}{\models} \phi$. In this section we present a remedy to these problems in the form of a variant of the satisfiability graph construction which can takes fairness requirements into consideration.

Definition 5.1 A formula ϕ is *modularly true* of a fair module M_f, i.e. $M_f \overset{\mathrm{fm}}{\models} \phi$, iff $M_f \,||\, M_f' \models \phi$ for all fair modules M_f' such that $M_f \,||\, M_f'$ is closed. □

The following result states that for a closed fair module it does not matter whether M is considered in or without an environment, i.e. that the fairness requirements remove the problem stated above.

Proposition 5.2 If M_f is a closed fair module, then $M_f \overset{\mathrm{fm}}{\models} \phi$ iff $M_f \models \phi$. □

Definition 5.3 Let $M_f = (A_o, \ldots, F)$ be a fair module. The *fair chaotic environment* of M_f, $ce(M_f)$ is the fair module $(A_o', \ldots, V', \{P(A_e) \times P(A_e)\})$, where $A_o', \ldots, V'$ are as in 4.3. □

Proposition 5.4 Let $M_f = (A_o, \ldots, F)$ be a fair module and ϕ a formula such that every atomic proposition occurring in ϕ is in $A_o \cup A_e$. Then $M_f \overset{\mathrm{fm}}{\models} \phi$ iff $M_f \,||\, ce(M_f) \models \phi$.
Proof: If $M_f \overset{\mathrm{fm}}{\models} \phi$, then $M_f \,||\, ce(M_f) \models \phi$ by the definition of $\overset{\mathrm{fm}}{\models}$. If $M_f \overset{\mathrm{fm}}{\not\models} \phi$, then there is an M_f' and a fair path p_1 in $K_1 = K(M_f \,||\, M_f')$ such that $V_1(p_1) \not\models \phi$ holds. As in 4.4 there is a path p_2 in $K_2 = K(M_f \,||\, ce(M_f))$ such that $V_1(p_1) \overset{\mathrm{st}}{\approx} V_2(p_2)$ (mod $A_o \cup A_e$). As p_1 is fair with respect to every fairness set $f \in F$, p_2 is, by its structure, also fair with respect to all the corresponding fairness sets of K_2. However, p_2 is not necessarily fair with respect to the one fairness set of K_2 corresponding to the single fairness set of $ce(M_f)$. This can be rectified by interleaving an infinite number of non-state-changing transitions of $ce(M_f)$ into p_2 without affecting $V_1(p_1) \overset{\mathrm{st}}{\approx} V_2(p_2)$ (mod $A_o \cup A_e$). □

In the construction of the satisfiability graph we have now the additional concern of discerning fair paths from unfair ones. What essentially happens when we move from the naive approach with the chaotic environment to the satisfiability graph is grouping together states (s, A') where the A':s differ only with respect to propositions not in A. This grouping, however, may introduce new dependencies between the enabled fairness sets, and may therefore have unwanted effects on which paths are regarded fair. The way to do away with these dependencies in order to reflect the original structure faithfully is to make as many copies of a state (s, A') as there are possible combinations of enabled fairness sets in the module, i.e. 2 copies in the case of a basic fair module, and $2^{|F|}$ copies in the general case. Each copy (s, A', F'), where $F' \subseteq F$, represents the possibility that in the chaotic environment it is possible to reach a state (s, A'') such that $A'' \cap A = A'$ and all the enabled fairness sets are in F'.

Definition 5.5 Let $M_f = (A_o, \ldots, F)$ be a fair module, $A \subseteq A_o \cup A_e$, $A_1 \subseteq A_e \cap A$, $A_1' \subseteq A_e$, $s \in S$ and $F_1 \subseteq F$. We say that (s, A_1') is an ***F_1-enabled A-extension of*** (s, A_1) iff $A_1' \cap A = A_1$ and for every $f \in F$ the following holds: if there are l and s' such that $s \xrightarrow{l} s' \in f$ and $A_1' \models l$, then $f \in F_1$. □

Definition 5.6 Let $M_f = (A_o, \ldots, F)$ be a fair module, where $F = \{f_1, \ldots, f_n\}$, and $A \subseteq A_o \cup A_e$. The *fair satisfiability graph* corresponding to M_f and A, $sg(M_f, A)$ is the fair Kripke-model $(S_g, I_g, R_g, V_g, F_g)$ defined as follows:

- $S_g = \{(s, A_1, F_1) \in S \times P(A \setminus A_o) \times P(F) \mid$
  $\exists A_1' : (s, A_1')$ is an F_1-enabled A-extension of $(s, A_1)\}$
- $I_g = (I \times P(A \setminus A_o) \times P(F)) \cap S_g$
- $R_g = \bigcup_{i=1}^{n+1} f_{gi}$, where for all $1 \leq i \leq n$
  $f_{gi} = \{((s_1, A_1, F_1), (s_2, A_2, F_2)) \in S_g \times S_g \mid f_i \in F_1$ and $A_1 = A_2$ and
  $\exists A_1' : \exists l : A_1' \models l$ and $s_1 \xrightarrow{l} s_2 \in f_i$ and
  (s_1, A_1') is an F_1-enabled A-extension of (s_1, A_1) and
  (s_2, A_1') is an F_2-enabled A-extension of $(s_2, A_2)\}$
  $f_{g\,n+1} = \{((s, A_1, F_1), (s, A_2, F_2)) \in S_g \times S_g\}$
- $V_g(s, A_1, F_1) = V(s) \cup A_1$
- $F_g = \{f_{g1}, \ldots, f_{gn}, f_{g\,n+1}\}$ □

Proposition 5.7 Let $M_f = (A_o, \ldots, F)$ be a fair module, where $F = \{f_1, \ldots, f_n\}$, and $A \subseteq A_o \cup A_e$. If ϕ is a formula containing only atomic propositions from A, then $M_f \overset{fm}{\models} \phi$ iff $sg(M_f, A) \models \phi$.

Proof: In the proof we use following notation: $M_f \| ce(M_f)$ is denoted by $M_c = (A_{co}, A_{ce}, S_c, I_c, R_c, V_c, F_c)$, where $F_c = \{f_{c1}, \ldots, f_{c\,n+1}\}$, and $sg(M_f, A)$ by $K_g = (S_g, I_g, R_g, V_g, F_g)$, where $F_g = \{f_{g1}, \ldots, f_{g\,n+1}\}$. The indexing of the fairness sets F_c and F_g is supposed to correspond to the indexing of the original fairness set F. If $(s_i, A_i) \in S_c$, then $en(s_i, A_i) = \{f_j \in F \mid f_{cj}$ is enabled in $(s_i, A_i)\}$.

The following facts are direct consequences of the definitions:

1 If $((s_i, A_i), (s_{i+1}, A_{i+1})) \in f_{cj}$, then
$((s_i, A_i \cap A, en(s_i, A_i)), (s_{i+1}, A_{i+1} \cap A, en(s_{i+1}, A_{i+1}))) \in f_{gj}$.

2 If $1 \leq j \leq n$, then f_{gj} is enabled in $(s_i, A_i, F_i) \in S_g$ only if $f_j \in F_i$. Consequently, if f_{gj} is enabled in $(s_i, A_i \cap A, en(s_i, A_i))$ then f_{cj} is enabled in (s_i, A_i).

3 If $((s_i, A_i, F_i), (s_{i+1}, A_{i+1}, F_{i+1})) \in f_{gj}$, then there exist A_i'' and A_{i+1}' such that $((s_i, A_i''), (s_{i+1}, A_{i+1}')) \in f_{cj}$, $A_i'' \cap A = A_i$, $A_{i+1}' \cap A = A_{i+1}$, and for all $1 \leq k \leq n+1$: if $f_k \in en(s_i, A_i'')$ then f_{ck} is enabled in (s_i, A_i, F_i), and if $f_k \in en(s_{i+1}, A_{i+1}')$ then f_{ck} is enabled in $(s_{i+1}, A_{i+1}, F_{i+1})$.

As in 4.6, the proof proceeds by showing that if $K(M_c)$ has a fair fullpath p then K_g has a fair fullpath p' such that $V_c(p) \overset{st}{\approx} V_g(p')$ (mod A) and vice versa.

Assume that $p = ((s_1, A_1), (s_2, A_2), \ldots)$ is a fair fullpath of $K(M_c)$. The required p' can be obtained as $p' = ((s_1, A_1 \cap A, en(s_1, A_1)), (s_2, A_2 \cap A, en(s_2, A_2)), \ldots)$. By 1 above and by the definition of R_g, p' is a fullpath. As p is fair, 1 and 2 imply that p' is fair as well. $V_c(p) \overset{st}{\approx} V_g(p')$ (mod A) is established as in 4.6.

Assume that $p = ((s_1, A_1, F_1), (s_2, A_2, F_2), \ldots)$ is a fair fullpath of K_g. The required p' can be obtained as $p' = ((s_1, A_1''), (s_2, A_2'), (s_2, A_2''), (s_3, A_3'), \ldots)$, where A_i' and A_i'' are as in 3 above. Note that for every transition in p there are two transitions in p'. By 3 above, p' is a fullpath. As p is fair, 3 implies that p' is fair as well. Finally, the definitions of A_i' and A_i'' used in constructing p' guarantee that $V_g(p) \overset{st}{\approx} V_c(p')$ (mod A). □

The size of the satisfiability graph is $|M| \cdot 2^{|F|} \cdot 2^{|A \setminus A_o|}$. In constructing the satisfiability graph the essential step is deciding whether $s_1 \xrightarrow{l} s_2 \in f_i$ implies $((s_1, A_1, F_1), (S_2, A_2, F_2)) \in f_{gi}$. This can be done by checking whether formula

$$l \wedge \bigwedge_{f \in F \setminus F_1} \bigwedge_{(s_1, a, s_1') \in f} \neg a \wedge \bigwedge_{f \in F \setminus F_2} \bigwedge_{(s_2, a, s_2') \in f} \neg a$$

is satisfiable, which can be done in linear time. Therefore, the satisfiability graph can be constructed in a time linear to its size. As checking the truth of ϕ over $sg(M_f, A)$ takes $O(|F| \cdot |sg(M_f, A)| \cdot 2^{c \cdot |\phi|})$ time, checking modular truth of ϕ in a fair module M_f requires $O(|F| \cdot |M_f| \cdot 2^{|F|} \cdot 2^{|\phi|} \cdot 2^{c \cdot |\phi|})$, i.e. $O(|M_f| \cdot 2^{c \cdot |F|} \cdot 2^{c \cdot |\phi|})$ time. This is less than that of the naive approach as long as $|F| < |AP_e \setminus A|$. Although the time requirement is exponential in $|F|$, this is unlikely to cause problems in practice as $|F|$ is typically very low.

Fairness between concurrent processes in a system is a special case of the more general fairness notion treated above. In order to model it only basic fair modules are needed. These have the property that all the transitions of a module form a single fairness set, i.e. that $|F| = 1$. In this case the time requirement reduces to $O(|M_f| \cdot 2^{c \cdot |\phi|})$, which means that we can answer the question about the truth of ϕ over module M in all strongly fair environments in the same time as over M in all environments.

As the last issue we discuss briefly the existence of the lemmas ϕ_1 and ϕ_2, and draw attention to a particular case in which we can be sure of their existence. Intuitively the following result shows that if every state of a module can be uniquely characterised by the atomic propositions true in that state, the lemmas always exist. This should come as no suprise, as the characterisability means that the structure of the modules can be axiomatised.

Proposition 5.8 Let M_1, M_2 be (fair) modules such that both V_1 and V_2 are injective, i.e. that for all $s, s' \in S_1$: $V_1(s) = V_1(s')$ implies $s = s'$ and alike for V_2.

If $M_1 \parallel M_2 \models^{fm} \phi$, there exist formulas ϕ_1 and ϕ_2 such that $M_1 \models^{fm} \phi_1$, $M_2 \models^{fm} \phi_2$ and $\models \phi_1 \wedge \phi_2 \Rightarrow \phi$.

Proof: Thanks to the characterisability of M_i we can construct formulas $ch(M_i)$ so that they completely describe the behaviour of M_i.

$$ch(s) = \bigwedge_{a \in V(s)} a \wedge \bigwedge_{a \in A_o \setminus V(s)} \neg a \qquad tr(s) = ch(s) \mathcal{U}_w (\bigvee_{(s,l,s') \in R} (ch(s') \wedge l))$$

$$ch(R) = \Box \bigwedge_{s \in S} (ch(s) \Rightarrow tr(s))$$

$$ch(f_i) = (\Box \Diamond \bigvee_{(s,l,s') \in f_i} (ch(s) \wedge l)) \Rightarrow (\Box \Diamond \bigvee_{(s,l,s') \in f_i} (ch(s) \mathcal{U} ch(s'))) \qquad ch(F) = \bigwedge_{f_i \in F} ch(f_i)$$

$$ch(M) = \Box \bigvee_{s \in S} ch(s) \wedge \bigvee_{s \in I} ch(s) \wedge ch(R) \wedge ch(F)$$

$$int(M, M') = \bigwedge_{a \in L(A_o)} \bigwedge_{a' \in L(A_o')} a \wedge a' \Rightarrow ((a \wedge a') \mathcal{U}_w ((\neg a \wedge a') \vee (a \wedge \neg a')))$$

(The formula $int(M, M')$ decrees that two propositions owned by different modules cannot change their truth-value at the same time.) Now $M_1 \models^{fm} ch(M_1) \wedge int(M_1, M_2)$ and alike for M_2. It remains to show that if $M_1 \parallel M_2 \models^{fm} \phi$ then $\models ch(M_1) \wedge int(M_1, M_2) \wedge ch(M_2) \Rightarrow \phi$. This can be done by supposing that, on the contrary, there were a Kripke-model K and a path p falsifying the implication, and by constructing a path p' in $(M_1 \parallel M_2) \parallel ce(M_1 \parallel M_2)$ falsifying ϕ as well, and thus creating a contradiction. □

It is clear that the lemmas constructed in the proof are of purely theoretical value due to their length. However, knowing that the method is complete in the sense that any property can be verified by it naturally adds confidence to it.

6 Discussion

In this paper we have presented algorithms deciding the truth of a nexttime-less linear temporal logic formula ϕ over a distributed-variable module M in all environments and in all fair environments. An interesting result obtained is that the time-complexity of checking whether a property holds of a module in all environments is essentially no higher than that of checking whether it holds without any environment. The method of modular verification suggested here seems to be very promising. However, what is still needed for a fully-fledged system is a methodology for creating the required lemmas.

References

[Bar86] Barringer, H.: Using Temporal Logic in the Compositional Specification of Concurrent Systems, in Galton, A. (ed.): *Temporal Logics and Their Applications*, Academic Press, 1987, pp. 59-90

[BKP84] Barringer, H. & Kuiper, R. & Pnueli, A.: Now You May Compose Temporal Logic Specification, in *Conference Record of the Sixteenth Annual ACM Symposium on Theory of Computing*, 1984, pp. 51-63

[BE91] Best, E. & Esparza, J.: *Model Checking of Persistent Petri Nets*, Hildesheimer Informatikberichte 11/91, Universität Hildesheim, Institut für Informatik, 1991, also presented in Computer Science Logic '91

[CLM89] Clarke, E. M. & Long, D. E. & McMillan, K. L.: Compositional Model Checking, in *Proceedings of the Fourth IEEE Symposium on Logic in Computer Science*, 1989, pp. 353-362

[GL91] Grümberg, O. & Long, D. E.: Model Checking and Modular Verification, in Baeten, J. C. M. & Groote, J. F. (eds.): *Proceedings of CONCUR'91, the 2nd International Conference on Concurrency Theory*, LNCS, vol. 527, Springer-Verlag, 1991, pp. 250-265

[Jos89] Josko, B.: Verifying the Correctness of AADL-modules Using Model Checking, in de Bakker, J.W. & de Roever, W.-P. & Rozenberg, G. (eds.): *Proceedings of the REX Workshop on Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness*, LNCS, vol. 430, Springer-Verlag, 1989, pp. 386-400

[Lam83] Lamport, L.: What Good is Temporal Logic?, in *Proceedings of the IFIP 9th World Computer Congress*, 1983, pp. 657-668

[LP85] Lichtenstein. O, & Pnueli, A.: Checking That Finite State Concurrent Programs Satisfy Their Linear Specification, in *Conference Record of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, 1985, pp. 97-107

[LPZ85] Lichtenstein. O, & Pnueli, A. & Zuck, L.: The Glory of The Past, in Parikh, R. (ed.): *Logics of Programs, Proceedings*, LNCS, vol. 193, Springer-Verlag, 1985, pp. 196-218

[MP81] Manna, Z. & Pnueli. A.: Verification of Concurrent Programs: The Temporal Framework, in Boyer, R. S. & Moore, J. S. (eds.): *The Correctness Problem in Computer Science*, Academic Press, 1981, pp. 215-273

[MP91] Manna, Z. & Pnueli. A.: *The Temporal Logic of Reactive and Concurrent Systems, vol. 1, Specification*, Springer-Verlag, 1991

[Pnu85] Pnueli, A.: In Transition from Global to Modular Temporal Reasoning About Programs, in Apt, K. R. (ed.): *Logics and Models of Concurrent Systems*, NATO ASI Series, vol. F13, Springer-Verlag, 1985, pp. 123-146