On Conditions for Modular Verification in Systems of Synchronising Components

Peter Drábik[1,2], Andrea Maggiolo-Schettini[1], and Paolo Milazzo[1]

[1] Dipartimento di Informatica, Università di Pisa, Italy,
[2] Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche, Pisa, Italy

Abstract. Property preservation is investigated as an approach to modular verification, leading to reduction of the property verification time for formal models. For modelling purposes, formalisms with multi-way synchronisations are considered. For the modular verification technique to work, a specific type of synchronisation is required for which a necessary condition is identified. It is a requirement on the semantics of the formalism, which is restricted to permit simultaneous execution only of component moves that make reference to each other.

Keywords: modular verification, synchronisation, automata

1 Introduction

Formal verification techniques, such as model checking, permit the verification of properties of a system by exploring all of its possible behaviours. In many cases, the complexity of the behaviour of a system of interest makes the analysis by means of model checking infeasible for realistic models. The problem is that this analysis technique typically relies on a state space representation whose size rapidly becomes intractable. This problem is known as *state space explosion*.

A method for trying to avoid the state space explosion problem is to consider a decomposition of the system and apply a modular verification method. In particular, properties to be verified often concern only a small portion of the modelled system rather than the system as a whole. Hence, for each property it would be useful to isolate a minimal fragment of the model that is necessary for the verification of such a property. This could allow global properties to be inferred from properties of the system components.

In [7, 6] we proposed a modular verification approach in which a system of interest is described by means of an automata-based formalism that supports modular construction. The approach, called *property preservation*, allows qualitative aspects of systems to be analysed with the guarantee that properties proved to hold in a suitable model fragment also hold in the whole model. The model fragment is obtained by applying a projection operation on the system description that removes unnecessary components. The logic considered for property specification is $ACTL^-$[10], the universal fragment of temporal logic CTL.

Our modular verification framework is based on results of Grumberg et al. [10] and on their application to the theory of concurrent systems proposed by Attie

CONCURRENCY, SPECIFICATION AND PROGRAMMING
M. Szczuka et al. (eds.): Proceedings of the international workshop CS&P 2011
September 28-30, Pułtusk, Poland, pp. 111-122

[1]. In particular, our approach extends Attie's one with the ability of modelling synchronisations among system components.

In our previous work the need of adding *synchronisation* to a modular verification framework was motivated by the interest in the verification of properties of biological systems. However, synchronisation is an aspect of many other kinds of systems (concurrent processes, communication protocols, etc.) and it can take several different forms. In this paper we investigate the conditions that a form of synchronisation must satisfy in order to enable modular verification. We show that in order to make modular verification possible, the semantics of the formalism must permit simultaneous execution only of component moves that reference each other in the synchronisation requirements. We call such a synchronisation complete. This restriction is necessary to avoid synchronisations of two components to rely on a third party that could be removed by the projection operation.

The paper is structured as follows. In Section 2 we define a notion of simulation between Labelled Transition Systems (LTSs), which are used as semantic models, that ensures ACTL$^-$ property preservation. In Section 3 we describe a modular verification approach for concurrent systems without synchronisation, which is essentially a reformulation of Attie's approach. In Section 4 we discuss how to include synchronisation in the modular verification framework and give the main result on the conditions that have to be satisfied in order to make modular verification possible. Finally, in Section 5 we discuss the applicability of our results to other formalisms for synchronising processes and draw our conclusions.

2 Property preservation on LTSs

We define a simulation relation between Labelled Transition Systems (LTSs) and show that any ACTL$^-$ property is preserved between LTSs that are in the relation. Our presentation follows the work of Grumberg and Long [10].

We consider LTSs whose states are built over *atomic propositions* from the set AP_I, where I is a finite *index set*. The set AP_I is a union of pairwise disjoint sets of atomic propositions AP_i where $i \in I$. States of an LTS built over AP_I are called I-states and are defined as follows. Let I be an index set, where $I = \{1, \ldots, n\}$. An *I-state* is defined as a tuple $s = (s_1, \ldots, s_n)$, where $s_i \subseteq AP_i$ is a subset of atomic propositions from AP_i true in that state and $i \in I$. A part of the I state $s = (s_1, \ldots, s_n)$ related to index i is obtained by the *projection operator* denoted $s\lceil i$ is s_i. Over I-states we construct LTSs called I-structures.

Definition 1. *Let $I = \{1, \ldots, n\}$ be an index set. An* I-structure *is a labelled transition system $\mathcal{M}_I = (\mathcal{S}_I, \mathcal{S}_I^0, \mathcal{R}_I)$, where $\mathcal{S}_I$ is a set of I-states, $\mathcal{S}_I^0 \subseteq \mathcal{S}_I$ is the set of initial states and $\mathcal{R}_I \subseteq \mathcal{S}_I \times \mathcal{P}(I) \times \mathcal{S}_I$ is a transition relation. Each transition is a triple (s, l, t), where label l is such that if $s\lceil i \neq t\lceil i$ then $i \in l$.*

Transitions in an I-structure include labels that indicate the parts of the I-states that are changed by the transition.

A *path* in an I-structure $\mathcal{M}_I$ is a sequence of I-states and transition labels $\pi = (s^1, l^1, s^2, l^2, \ldots)$ such that for all m, $(s^m, l^m, s^{m+1}) \in \mathcal{R}_I$. A *fullpath* is a

maximal path. For the purposes of verification, we assume a notion of unconditional fairness. We say that a path $\pi = (s^1, l^1, s^2, l^2, \ldots)$ in $\mathcal{M}_I$ is *fair* iff for all $i \in I$ we have that $\{m \mid i \in l^m\}$ is infinite.

To be able to define the relation of simulation, we introduce an operation of *path projection*. First, we say that an I-state $s = \{s_1, \ldots, s_n\}$ can be projected onto $J \subseteq I$ with indices $\{j_1, \ldots, j_k\}$ as follows: $s\lceil J = (s_{j_1}, \ldots, s_{j_k})$. A transition (s, l, t) in an I-structure can be projected onto $J \subseteq I$ such that $l \cap J \neq \emptyset$ by projecting the I-states and the label: $(s, l, t)\lceil J = (s\lceil J, l \cap J, t\lceil J)$. Intuitively, a path in I-structure is projected onto $J \subseteq I$ by projecting transitions with label l such that $l \cap J \neq \emptyset$ onto J . Moreover, transitions with $l \cap J = \emptyset$ are removed because the part of the I-state related to J is not changed by the transition.

The simulation relation on LTSs captures the concept of the whole system and an overapproximation of its part.

Definition 2. *Let I be an index set and $J \subseteq I$. Consider an I-structure $\mathcal{M}_1$ and a J-structure $\mathcal{M}_2$. We say that $\mathcal{M}_2$* simulates *$\mathcal{M}_1$ iff for each fair path π in $\mathcal{M}_1$, $\mathcal{M}_2$ contains $\pi\lceil J$.*

Now we describe the logic, ACTL$^-$, that is used for specification of the properties of LTSs. ACTL is the "universal fragment" of Computational Tree Logic (CTL) which results from CTL by restricting negation to propositions and eliminating the existential path quantifier. The logic we will use, ACTL$^-$, is ACTL without the AX modality.

Definition 3. *The* syntax of ACTL$^-$ *is defined inductively as follows:*

- *The constants true and false are formulae. p and $\neg p$ are formulae for any atomic proposition p.*
- *If f, g are formulae, then so are $f \wedge g$ and $f \vee g$.*
- *If f, g are formulae, then so are $A[f\ U\ g]$ and $A[f\ U_w\ g]$.*

We define the logic ACTL^-_J to be ACTL$^-$ where the atomic propositions are drawn from $AP_J = \{AP_i \mid i \in J\}$. Abbreviations in ACTL$^-$: $AFf \equiv A[true\ U\ f]$ and $AGf \equiv A[f\ U_w\ false]$.

The intuitive semantics of ACTL$^-$ formulae evaluated on an I-structure is: *true* and *false* are true in all and in no I-state, respectively; the truth of atomic propositions in a state is given by the set of atomic propositions included in the state; the logical connectives "and" and "or" have the usual semantics, $A[f\ U\ g]$ and $A[f\ U_w\ g]$ express that f is satisfied in each state on each fair fullpath leading from the current state until g is satisfied. This is permitted to never occur in the latter, in which case f holds forever. Note that only fair fullpaths are considered.

Finally, the simulation relation of two structures ensures that if an ACTL$^-$ formula holds in the first, then it holds in the other.

Theorem 1 (Property preservation). *Let I be an index set and $J \subseteq I$. Let $\mathcal{M}_I$ and $\mathcal{M}_J$ be I- and J-structures, respectively and s an I-state. If $\mathcal{M}_J$ simulates $\mathcal{M}_I$, then if $\mathcal{M}_J, s\lceil J \vDash f$ then $\mathcal{M}_I, s \vDash f$ for all $ACTL^-_J$ properties f.*

3 Modular Verification in the Absence of Synchronisation

The property preservation theorem can be used for modular verification of ACTL$^-$ properties. Consider a program P_1, whose semantics is an I-structure $\mathcal{M}_1$ and an ACTL^-_J property f to be verified on $\mathcal{M}_1$. Instead of constructing $\mathcal{M}_1$ and verifying whether it satisfies f, we construct a J-structure $\mathcal{M}_2$ s.t.

- $\mathcal{M}_2$ simulates $\mathcal{M}_1$, which implies property preservation of ACTL^-_J formulae,
- to avoid the state explosion $\mathcal{M}_2$ is constructed directly, without passing through $\mathcal{M}_1$.

In order to construct $\mathcal{M}_2$ directly, it is necessary to have for the formalism of program P_1 a syntactic projection operation onto $J \subseteq I$, that guarantees that a syntactic projection on J of program P_1 has a J-structure as its semantics.

Consider a component-based formalism, where components are represented by finite state automata. A program is composed of a parallel composition of the individual automata, such that an automaton performs a move based on the states of the other automata. Note that no moves are performed synchronously.

We describe a synchronisation-free formalism called synchronisation skeletons [4] used also by Attie [1]. Note that for the purpose of the present paper we consider a simplified version of synchronisation skeletons without shared variables. A synchronisation skeleton is a directed graph where each node is a local state and each arc has a label that specifies an *enabling condition*. This condition describes the states of other synchronisation skeletons that make the move enabled.

Definition 4. *A* synchronisation skeleton SP_i^I*, where i is an index and I is an index set, is a tuple (S_i, S_i^0, R_i):*

- $S_i \subseteq \mathcal{P}(AP_i)$ *is the set of* states*;*
- $S_i^0 \subseteq S_i$ *is the set of* initial states*;*
- $R_i \subseteq S_i \times EC_i \times S_i$*, where $EC_i \subseteq \bigcup_{L \subseteq I-\{i\}} \Pi_{j \in L} \mathcal{P}(AP_j)$, are the* moves *between states.*

A label of a move is called an enabling condition.

Definition 5. *An* enabling condition *of a synchronisation skeleton SP_i^I is a label of the form $\{A_j \mid i \in L\}$, where $L \subseteq I - \{i\}$ and A_j is a set of atomic propositions drawn from AP_j or their negations.*

We display an enabling condition $\{A_j \mid i \in L\}$, unless said otherwise, as a formula $\wedge_{j \in L} A_j$ where A_j is a conjunction of atomic propositions from AP_j.

A *synchronisation skeleton program* consists of a parallel composition of synchronisation skeletons related by an index set I.

Definition 6. *Let $I = \{1, \ldots, n\}$ be an index set. The* synchronisation skeleton program *is a tuple $SP^I = (S_0^I, SP_1^I || \ldots || SP_n^I)$, where each SP_i^I is a synchronisation skeleton. The set $S_0^I = S_1^0 \times \ldots \times S_n^0$ is the set of initial states of the synchronisation skeleton program.*

The parallelism is modelled in the usual way by the non-deterministic interleaving of moves of the individual synchronisation skeletons of the processes P_i. Hence, at each step of the computation, some process with an enabled arc is non-deterministically selected to be executed next. Now we define the semantics of a synchronisation skeleton formally.

Let I be an index set where $I = \{1, \ldots, n\}$. As in the previous section, an *I-state* is a n-tuple $s = \{s_1, \ldots, s_n\}$, where $s_i \in S_i$ for all $i \in I$. An I-state represents a global state composed of local states of all synchronisation skeletons in I. Note that $s\lceil i$ denotes the i-th element of an I-state s.

Definition 7. *Let $I = \{1, \ldots, n\}$ be an index set. The* semantics *of $SP^I = (S_0^I, SP_1^I || \ldots || SP_n^I)$ is given by the* I-structure *$\mathcal{M}_I = (\mathcal{S}_I, \mathcal{S}_I^0, \mathcal{R}_I)$, where*

- *$\mathcal{S}_I = \Pi_{i \in I} S_i$ is a set of I-states*
- *$\mathcal{S}_I^0 \subseteq \mathcal{S}_I$ is the set of initial states*
- *$\mathcal{R}_I \subseteq \mathcal{S}_I \times \mathcal{P}(I) \times \mathcal{S}_I$ is a transition relation giving the transitions of SP^I. A transition (s, l, t) is in $\mathcal{R}_I$ iff there is an index i, such that*
  - *there is a move $s_i \xrightarrow{cc_i} t_i$ such that*
    - *$s_i = s\lceil i$ and $t_i = t\lceil i$*
    - *$cc_i = \{A_j \mid i \in L\}$ and $L \in I - \{i\}$ and for all $j \in L$, A_j is true in s_j*
  - *for all j such that $j \neq i$, $s\lceil j = t\lceil j$.*

The *syntactic projection* is used in order to obtain a sub-program composed of the components with indices from $J \subseteq I$. Intuitively, the projection leaves out components that are not in J and removes references in the conditions to these components. Note that after an application of this simple and intuitive projection operator on a synchronisation skeleton program, again a synchronisation skeleton program is obtained. The formal definition follows.

Definition 8. *Let $J \subseteq I$ be an index set and $J = \{j_1, \ldots, j_k\}$. Let $P^I = (S_0^I, P_1^I || \ldots || P_n^I)$ with $P_i^I = (S_i, S_i^0, R_i)$ for each $i \in I$.*
Then $P^I \restriction J = (S_0^J, P_{j_1}^J || \ldots || P_{j_k}^J)$ with $P_j^J = (S_j, S_j^0, R'_j)$ for each $j \in J$ where

- *S_j and S_j^0 are as in P_j^I;*
- *$R'_j = \{s_j \xrightarrow{\wedge_{j' \in L \cap J} A_{j'}} t_j \mid s_i \xrightarrow{\wedge_{j' \in L} A_{j'}} t_j \in R_j\}$.*

Initial states are $S_0^J = S_{j_1}^0 \times \ldots \times S_{j_n}^0$.

By using the syntactic projection defined above, we obtain a synchronisation skeleton program whose semantics simulates the semantics of the original synchronisation skeleton program [1].

Lemma 1 (Simulation). *Let I be an index set and $J \subseteq I$. Consider synchronisation skeleton program P_1 with semantics $\mathcal{M}_1$ and $P_2 = P_1 \restriction J$ with semantics $\mathcal{M}_2$. Then $\mathcal{M}_2$ simulates $\mathcal{M}_1$.*

Thus the modular verification of an ACTL_J^- formula can be performed on the syntactic projection of the program as guaranteed by Theorem 1.

4 Modular Verification in the Presence of Synchronisation

Synchronisation is a characteristic feature of many kinds of systems, ranging from biological systems to concurrent processes, communication protocols and others. For the purpose of modelling such systems, a formalism with synchronisation as a primitive is necessary. Under a synchronisation primitive we understand a mechanism that allows an automaton to express for each move conditions on moves of other automata, without which it cannot be executed. These conditions are taken into account formally by the semantics of the formalism when executing moves simultaneously. The semantics can deal with this in several different ways.

In this section we investigate the impact of resolving the synchronisation by the semantics on the modular verification. First we introduce an extension of the synchronisation skeleton formalism by synchronisation. Then, for a simple syntactic projection in the style of the previous section, we obtain a sub-program for which we try to prove the simulation result. As this is not true in general, we study the conditions on the semantics that allow to obtain the result.

4.1 A Formalism with Synchronisation

In previous work we extended the synchronisation skeleton programs formalism by synchronisation. The formalism called sync-programs is defined as follows.

A *sync-automaton* is an analogue of a synchronisation skeleton, with its set of states S_i, set of initial states S_i^0. Different from the synchronisation skeleton, the set R_i of labelled moves between states is $R_i \subseteq S_i \times SC_i \times S_i$, where $SC_i \subseteq \mathcal{P}(\bigcup_{j\in I}(\mathcal{P}(AP_j) \times \mathcal{P}(AP_j)))$.

The move label is called a *synchronisation condition* and consists of a set of requirements against other sync-automata, each formed by a precondition and a postcondition on the move that is expected to be performed concurrently.

Definition 9. *A* synchronisation condition *of sync-automaton P_i^I is a label of the form $\{A_{j_1}{:}B_{j_1}, \ldots, A_{j_n}{:}B_{j_n}\}$, where $\{j_1, \ldots, j_n\} \subseteq I - \{i\}$ and A_j, B_j are sets of atomic propositions drawn from AP_j or their negations.*

We denote a synchronisation condition $\{Aj_1{:}Bj_1, \ldots, Aj_n{:}Bj_n\}$ as a formula $\wedge_{j\in L} A_j{:}B_j$, where $L = \{j_1, \ldots, j_n\}$ and A_j, B_j are conjunctions of atomic propositions from AP_j. The set $L \subseteq I{-}\{i\}$ contains indices of the sync-automata with which P_i^I wants to synchronise. For every j in L, the sets of propositions A_j and B_j are to be satisfied in the starting and ending state, respectively, of the concurrently performed move of P_j^I. In other words, $A_j{:}B_j$ in a label of a move of P_i^I says that every move in P_j^I that can be performed in parallel with this move of P_i^I is obliged to lead from a state satisfying A_j to a state satisfying B_j.

As before, a *sync-program* consists of a parallel composition of sync-automata related by an index set I. The semantics of a sync-program P^I is an I-structure where states are I-states and transitions represent simultaneous executions of several related moves.

Definition 10. *Let $I = \{1, \ldots, n\}$ be an index set. The* semantics *of $P^I = (S_I^0, P_1^I || ... || P_n^I)$ is given by the* I-structure *$\mathcal{M}_I = (\mathcal{S}_I, \mathcal{S}_I^0, \mathcal{R}_I)$ where $\mathcal{S}_I, \mathcal{S}_I^0$ and*

$\mathcal{R}_I$ are the sets of states, initial states and transitions, respectively. A transition (s, l, t) is in $\mathcal{R}_I$ iff there is a nonempty set MOV of moves of P^I such that $l = types(MOV)$ and MOV has support (s, t).

If MOV is a set of moves, $types(MOV)$ gives a set of indices of automata whose moves are present in MOV. The label l thus contains the set of components which participate in the transition.

A pair of I-states such that the second state can be reached from the first one by carrying out the synchronisation in question is called a support of MOV.

Definition 11. *Let I be an index set. Consider a set of moves MOV consisting of moves $s_i \xrightarrow{sc_i} t_i$ for all $i \in types(MOV)$. We call a couple of states (s, t) the* support of MOV *iff s and t are I-states and*

1. *$s\lceil i = s_i$ for all $i \in types(MOV)$*
2. *$t\lceil i = t_i$ for all $i \in types(MOV)$*
3. *for all $i \in I - types(MOV)$: $s\lceil i = t\lceil i$.*

That is, the only part of the I-state that is changed by a transition is the one influenced by MOV. In other words MOV contains moves which perform a move simultaneously and the rest of automata stay idle.

We call the set of moves performed simultaneously a *synchronisation*. Before giving a definition of a synchronisation, we give the intuition and then provide the most general definition that is compatible with the intuition.

A synchronisation should contain moves from distinct automata, because it is impossible that more moves of the same automaton be performed concurrently. The next stipulation is that for each move its conditions on other moves are satisfied, making the synchronisation sound. The last requirement is that only related moves should be considered. This is to guarantee the "atomicity" of the synchronisation, that is just the moves that wish to participate on the synchronisation do so. The above specification is formalised in the following definition.

Definition 12. *Let I be an index set. We call a set of moves MOV a* synchronisation *iff*

1. *all moves in MOV are from distinct sync-automata in I*
2. *if $m \in MOV$ is of type i and has the form $s_i \xrightarrow{\wedge_{j \in L} A_j : B_j} t_i$, then for all $j \in L$ there is a move $m' \in MOV$ of type j and has the form $s_j \xrightarrow{sc_j} t_j$ and for all $p \in A_j$: $s_j(p) = tt$ and for all $p \in B_j$: $t_j(p) = tt$*
3. *Let G_{MOV} be a graph, where vertices are the moves from MOV and there is a directed edge from m_1 to m_2 iff move m_1 contains a reference to m_2 in its synchronisation condition. Then G_{MOV} is weakly connected.*

A directed graph is weakly connected iff when replacing each directed edge by an undirected one, we obtain a connected (undirected) graph. In other words, any pair of vertices is connected through an undirected path. Note that weak connectivity corresponds to our intuition of considering only related moves, in particular MOV cannot be partitioned in two sets such that both are correct synchronisations.

4.2 Conditions for Modular Verification

As in Section 3 we consider a *syntactic projection* that leaves out components that are not in J and remove references in the conditions to these components. The only change is in the part where synchronisation conditions are projected, but only because synchronisation conditions contain pre- and postconditions as opposed to a single enabling condition. We would like to prove that the semantics of the projected sync-program simulates the semantics of the original sync program. However, in general this is *not* the case. A counterexample follows.

Example 1. Let $m_1 = A \xrightarrow{X:\neg X \wedge Z:\neg Z} \neg A$, $m_2 = X \xrightarrow{A:\neg A} \neg X$, $m_3 = Z \xrightarrow{A:\neg A} \neg Z$ where the moves m_1, m_2 and m_3 belong to sync-automaton a_1, a_2 and a_3, respectively. Consider the transition $(s, l, t) = ([A, X, Z], \{1, 2, 3\}, [\neg A, \neg X, \neg Z])$ that is present in the semantics of a sync-program composed of sync-automata a_1, a_2 and a_3. The tuple $MOV_1 = \{m_1, m_2, m_3\}$ is a synchronisation according the Definition 12. But $(s, l, t)\lceil\{2, 3\}$ is not a transition in the sync-program projected onto $\{2, 3\}$ since the moves $m_2\restriction\{2, 3\} = X \xrightarrow{NOSYNC} \neg X$ and $m_3\restriction\{2, 3\} = Z \xrightarrow{NOSYNC} \neg Z$ do not form a synchronisation. In particular the third point from the definition of a synchronisation (Def. 12) is not satisfied, as the graph $G_{MOV_1\lceil\{2,3\}} = (\{a_2, a_3\}, \emptyset)$ is not weakly connected. Therefore the semantics of sync-program $P^{2,3} = \{a_2||a_3\}$ does not simulate the semantics of $P^{1,2,3} = \{a_1||a_2||a_3\}$.

As we can see from the example, with the current definition of semantics the simulation cannot be proved for sync-programs. Since the syntactical projection is fixed for sync-programs, we can study additional conditions on the semantics of sync-programs to guarantee simulation.

To prove that the semantics of the projection and the original semantics are in a simulation relation, by the definition of the simulation one must show preservation of any fair path from the semantics of the projected program to the semantics of the original program.

The proof is twofold. First it needs to be shown that each path becomes a path after the projection. Second, one must prove that a fair path still remains fair. While the latter follows trivially from the definition of fairness, the former is implied by the Transition projection lemma.

Lemma 2 (Transition projection). *Let I be an index set, $\mathcal{M}_I = (\mathcal{S}_\mathcal{I}, \mathcal{S}_0^I, \mathcal{R}_I)$ the semantics of sync-program P^I. For all I-states s, t in $\mathcal{S}_I$ and all $l \in \mathcal{P}(I)$, transition (s, l, t) is in $\mathcal{R}_I$ iff for all $J \subseteq I$ such that $l \cap J \neq \emptyset$, $(s, l, t)\lceil J$ is in $\mathcal{R}_J$, where $\mathcal{M}_J = (\mathcal{S}_\mathcal{J}, \mathcal{S}_0^J, \mathcal{R}_J)$ is the semantics of sync-program $P^J = P^I\restriction J$.*

The path preservation follows from application of the Transition projection lemma to each transition of the considered path. Hence, the Transition projection lemma constitutes a necessary condition for simulation. However, the Transition projection lemma does not hold for the example 1. We investigate for which definitions of synchronisation we are able to prove the Transition projection lemma. The example suggests that the synchronisation preservation fails.

Lemma 3. *If MOV is a synchronisation of moves from P^I, then $MOV\restriction J$ is a synchronisation of moves from sync-subprogram $P^J = P^I\restriction J$.*

It is easy to see that the violation of this desired result comes from the third point of Definition 12.

Weak directed connectivity of the graph from the definition of synchronisation is not preserved by the syntactic projection. Hence, it is necessary to impose a stronger condition as a property of the graph in the definition of synchronisation. Let *Prop* denote the desired graph property. Then these are formal requirements on the property *Prop*.

1. *Prop* implies the weak directed connectivity (if $Prop(G_{MOV})$ then G_{MOV} is weakly directed)
2. if $Prop(G_{MOV})$ then $Prop(G_{MOV\restriction J})$ where $J \subseteq I$.

The effect of projection of a synchronisation (Def. 8) is that it removes nodes and their induced edges, thus producing an induced subgraph.

The second requirement can be rephrased by using graph theoretical concepts. The property *Prop* is induced hereditary: *Prop* is *induced hereditary* iff if *Prop* holds for G, then it holds for all induced subgraphs H of G [3].

Some induced hereditary properties [8] are completeness, planarity, outerplanarity, bipartity, acyclicity, having max degree, interval graphs, chordal graphs. Not induced hereditary: weak connectivity (as above), connectivity (if a graph contains a directed path from u to v or a directed path from v to u for every pair of vertices u, v), strong connectivity (if it contains a directed path from u to v and a directed path from v to u for every pair of vertices u, v).

We prove that the only graph property that satisfies the two characteristics above is completeness.

Lemma 4. *If $Prop(G)$ implies $Prop(H)$ for all induced subgraphs H of G and $Prop(G)$ implies $WeakConnected(G)$ then $Prop(G) \leftrightarrow Complete(G)$.*

Proof. Let us suppose by contradiction, that *Prop* is different from *Complete*.

Let G be a graph such that $G \in Prop$. By the assumption G is not complete. That means that there are vertices v and v' such that the edge (v, v') is not in G. Then consider the subgraph G' induced by the vertices v and v'. Since *Prop* is induced hereditary, G' is in *Prop*. Then by the assumption G' is weakly connected. However, this contradicts to the fact that there is no edge between v and v'. Thus the assumption that *Prop* is different from *Complete* is wrong and this completes the proof.

This means that in order to satisfy the Transition projection lemma, a synchronisation has to consider only moves of automata that reference each other in the synchronisation condition. We call such a synchronisation complete.

Definition 13. *Let I be an index set. We call a set of moves MOV a* complete synchronisation *iff*

1. *as in Definition 12*
2. *as in Definition 12*
3. *Let G_{MOV} be a graph, where vertices are the moves from MOV and there is a directed edge from m_1 to m_2 iff move m_1 contains a reference to m_2 in its synchronisation condition. Then G_{MOV} is* complete.

We proved that the complete synchronisation is preserved by projection. This implies that a semantics that uses the complete synchronisation satisfies the Transition projection lemma. Hence we have proved the Simulation lemma for sync-programs with complete synchronisation.

Lemma 5 (Simulation). *Let I be an index set and $J \subseteq I$. We consider sync-programs with complete synchronisation. Consider sync-program P_1 with semantics $\mathcal{M}_1$ and $P_2 = P_1 \restriction J$ with semantics $\mathcal{M}_2$. Then $\mathcal{M}_2$ simulates $\mathcal{M}_1$.*

Thus the modular verification of an ACTL_J^- formula can be performed on the syntactic projection of the sync-program as guaranteed by Theorem 1.

5 Discussion

We have investigated property preservation as an approach to modular verification, leading to reduction of the property verification time for formal models.

For the purpose of modelling we consider formalims with multi-way synchronisation. In particular, in an automata-based language each move of an automaton indicates all the moves of other automata without which it is not willing to be performed. Driven by the intuition, we defined a "minimalist" type of synchronisation. However, we have discovered that in order for the modular verification technique to work, a specific type of synchronisation is required for which we have identified a necessary condition. This condition is a requirement on the semantics of the formalism, which is restricted to permit simultaneous execution only of automata moves, that reference each other. This is necessary to avoid synchronisations of two components to rely on a third party that could be removed by the projection operation when doing the modular verification.

This result has a generalisation to other formalisms. If one wants the modular verification for a formalism with synchronisation, the semantics of the formalism has to respect the *complete synchronisation schema*.

A semantics of a formalism with the synchronisation of automata/processes (in the following we use the name automata) has the complete synchronisation schema when its concept of synchronisation, that is determining which moves/actions/events/activities (in the following we use the name move) that are performed simultaneously, follows the same principles as the synchronisation in (Def. 13), namely: (i) moves are from distinct automata; (ii) for each move its conditions on other moves are satisfied; (iii) each move is willing to synchronise with all the participating moves.

There are some well known formalisms that enable synchronisation of multiple processes through shared actions, for example CSP [12] and PEPA [11]. In the synchronisation, action names are used without a reference to the process names.

In order to satisfy the third condition, it is necessary that all processes currently prepared to perform an action really do so. In CSP, even a stronger condition is satisfied: if an action is in the alphabet of a process then its participation on the action execution is necessary [9]. In PEPA, let us consider the fragment where all rates r of all activities$(\alpha, r).S$ are infinite (we denote it PEPA$_\infty$). In a way this is equivalent to a qualitative version of the calculus where all activities are instantaneous. The synchronisation of two processes $C_1 \bowtie_L C_2$ is specified as a synchronisation on all activities from L, other activities are independent. Again, the satisfaction of the third condition is guaranteed. In CCS [13], processes may interact on two complementary actions a and $\bar{a}$. Only two agents may participate in each interaction. Since processes cannot choose to perform a or $\bar{a}$ without the partner, our syntactical projection does not apply here.

As a result, since implicitly all three requirements for the synchronisation in the semantics of all CSP and PEPA$_\infty$ are satisfied, their semantics have complete synchronisation schema. Thus it is possible to prove the preservation of all ACTL$^-$ formulae for CSP and PEPA$_\infty$.

As regards related work, in the definition of our modular verification technique we have followed the "property preservation" approach, namely that truth of ACTL$^-$ formulae is preserved from sync-subprograms to the program. This has been originally considered by Grumberg et al. [10] and Dams et al. [5] in different contexts. Other approaches to modular verification infer properties of a system from some properties of its components, e.g. [2].

We have investigated the way the definition of synchronisation influences the characteristics that a subsystem obtained via projection always contains sound behaviour with respect to the whole system. A somehow related study was done in [14] for Team automata. Team automata is a formalism based on finite-state automata synchronising on shared actions. The distinctive feature of this formalism is that an automaton does not necessarily participate in every synchronisation of an action it shares (different from most other models of concurrent systems). Therefore the transition relation of a Team automaton is not uniquely determined by its constituting component automata. There is freedom to choose for an action a synchronisation strategy, namely how automata synchronise on it. Some of the possibilities are these three:

- free – actions are never executed simultaneously by more than one automaton
- action-indispensable – actions are executed as synchronisations in which all automata having them in the alphabet must participate
- state-indispensable – actions are executed as synchronisation in which only automata which are ready participate

In [14] the computations and behaviour of Team automata in relation to those of their constituting component automata has been studied in detail. Several types of Team automata that satisfy compositionality could be identified, their behaviour can be described in terms of its constituting component automata. In particular compositionality is satisfied in an important case of Team automata, namely where all actions are action-indispensable, i.e. guarantee the participation of all components that share the action in sync.

In our approach we assume a notion of fairness. The fairness condition consists of requiring that each component of the system contributes to the overall behaviour with infinitely many transitions. This work has been motivated by the study of a formalism for descriptions of biological systems, where such an assumption is reasonable. For the more general setting of systems of synchronising components, one might think of more general notions of fairness, but then the notion of simulation needs to be adjusted.

The reduced program is obtained by a simple and intuitive projection operator, which leaves out some component and removes references to them. This posed constraints on the semantic function (synchronisation). If one wants to extend the approach beyond the restrictions on the synchronisation, i.e. to formalisms with different types of synchronisation, one must consider more sophisticated forms of syntactic projections.

References

1. Paul C. Attie. Synthesis of large dynamic concurrent programs from dynamic specifications. *CoRR*, abs/0801.1687, 2008.
2. Saddek Bensalem, Marius Bozga, Joseph Sifakis, and Thanh-Hung Nguyen. Compositional verification for component-based systems and application. In *ATVA '08: Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis*, pages 64–79, Berlin, Heidelberg, 2008. Springer-Verlag.
3. M Borowiecki, Izak Broere, M Frick, and P Mihok. A survey of hereditary properties of graphs. *Discuss. Math. Graph*, 17:5–50, 1997.
4. Edmund M Clarke and E Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. *Logics of Programs*, pages 52–71, 1982.
5. Dennis Dams, Rob Gerth, and Orna Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19(2):253–291, 1997.
6. Peter Drábik, Andrea Maggiolo-Schettini, and Paolo Milazzo. Dynamic sync-programs for modular verification of biological systems. In *Workshop on Non-Classical Models of Automata and applications (NCMA'10)*, Jena, Germany, 2010.
7. Peter Drábik, Andrea Maggiolo-Schettini, and Paolo Milazzo. Modular Verification of Interactive Systems with an Application to Biology. *Annals of Computer Science*, in press, 2011.
8. Uriel Feige and S. Kogan. The hardness of approximating hereditary properties. *Available on: http://research.microsoft.com/research/theory /feige/homepagefiles/hereditary.pdf*, pages 1–12, 2005.
9. Colin Fidge. A comparative introduction to CSP, CCS and LOTOS. *Software Verification Research Centre, Univ. of Queensland, Tech. Rep*, pages 93–24, 1994.
10. Orna Grumberg and David E. Long. Model checking and modular verification. *ACM Trans. Program. Lang. Syst.*, 16(3):843–871, 1994.
11. J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
12. C A R Hoare. *Communicating Sequential Processes*, volume 9. Prentice-Hall International, London, August 1985.
13. R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.
14. Maurice ter Beek and Jetty Kleijn. Team automata satisfying compositionality. *FME 2003: Formal Methods*, pages 381–400, 2003.