Modular Verification of Interactive Systems with an Application to Biology

Peter Drábik [1] Andrea Maggiolo-Schettini [2] Paolo Milazzo [3]

Dipartimento di Informatica, Università di Pisa
Largo B. Pontecorvo 3, 56127, Pisa, Italy

Abstract

We propose an automata-based formalism for the description of biological systems that allows properties expressed in the universal fragment of CTL to be verified in a modular way. As an example we show the modelling of *lac* operon regulation and the modular verification of some properties.

Keywords: systems biology, concurrent interactive systems, modular verification, model checking

1 Introduction

Many formalisms originally developed by computer scientists to model systems of interacting components have been applied to biology, also with extensions to allow more precise descriptions of the biological behaviours [3,5,7,10,17,18]. Model checking permits the verification of properties of a system (expressed as logical formulas) by exploring all the possible behaviours of the system. It has been successfully applied to analysis of biological systems. Examples of well-established formal frameworks that can be used to model, simulate and model check descriptions of biological systems are [13,11,7].

However, model checking techniques have traditionally suffered from the state explosion problem. A method for trying to avoid such a problem exploits the natural decomposition of the system. The goal is to verify properties of

[1] Email: drabik@di.unipi.it
[2] Email: maggiolo@di.unipi.it
[3] Email: milazzo@di.unipi.it

individual components and infer that these hold in the complete system. A class of properties whose satisfaction is preserved from the components to the complete system was identified in Grumberg and Long [12] as ACTL, the universal fragment of CTL temporal logic. A technique proposed by Attie [1] exploits the preservation of these properties in order to verify concurrent programs and synthesise systems from specifications. Attie uses a formalism called synchronisation skeletons [8], abstraction of sequential processes, suitable for describing distributed systems.

In this paper we propose an extension of Attie's approach and application of the modular verification technique to systems biology. However, synchronisation skeletons are not suitable for modelling biological systems because of their interleaving nature. In fact, in synchronisation skeletons a process may perform an autonomous transition by looking at other processes states.

We define sync-programs, an automata-based formalism of interactive systems which extends Attie's approach by allowing processes to perform transitions simultaneously.

To be able to apply the proposed modular verification technique, the systems under consideration are subject to some restrictions. In particular, we assume infinite behaviours with a finite number of states and fairness of systems. The fairness condition consists in requiring that each component of the system contributes to the overall behaviour with infinitely many transitions.

We apply our formalism to the *lac* operon regulation and we show some properties that can be verified in a modular way.

2 Sync-programs

In this section we define the syntax and the semantics of the sync-progams, namely Attie's synchronisation skeletons extended with synchronisation.

2.1 Syntax

To model biological systems, we use a component-based approach. Each component represents a biological entity, e.g. a protein or an enzyme. We assume a finite *set of indices* Ids, where every component has a unique *index* i. We say that distinct components C_i and C_j *interact directly*, when one can perform activity conditioned on the activity of the other. Direct interaction is distinguished from indirect interaction, which is mediated by a third party. We define undirected *interaction graph* I (see e.g. fig. 1), where the nodes are indices from Ids and there is an edge between nodes i and j if components C_i and C_j interact directly. We use notation iIj if there is an edge between i and j in graph I. By $|I|$ we denote the set of nodes in I and by $I(i)$ the set $\{j \in |I| \mid iIj\}$. By $J \subseteq I$ we denote a connected subgraph J of I.

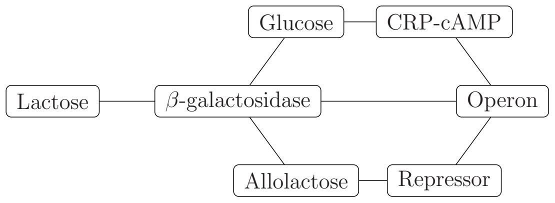


Fig. 1. Interaction graph

With every component C_i a set AP_i of *atomic propositions* is associated, which encode the state of the component. The sets of atomic propositions are pairwise disjoint for all the components, i.e. if $i \neq j$ then $AP_i \cap AP_j = \emptyset$.

A component is modelled by using a finite state machine called sync-automaton.

Definition 2.1 A *sync-automaton* P_i^I, where i is a component index and I is an interaction graph, is a tuple (S_i, S_i^0, SC_i, R_i):

- $S_i \subseteq 2^{AP_i}$ is the set of *states*
- $S_i^0 \subseteq S_i$ is the set of *initial states*
- SC_i is the set of labels of the form $\wedge_{j \in L} A_j{:}B_j$ where $L \subseteq I(i)$ and A_j, B_j are sets of atomic propositions drawn from AP_j or their negations. A label from SC_i is called a *synchronisation condition*
- $R_i \subseteq S_i \times SC_i \times S_i$ are the *moves* between states.

Each state of a sync-automaton P_i^I is a truth value assignment to atomic propositions of component C_i. Each move is labelled by a synchronisation condition. We denote a move from state s_i to state t_i with a label c by $s_i \xrightarrow{c} t_i$. The move from state s_i to t_i with label c intuitively means that automaton P_i^I can move from s_i to t_i if the activities of automata in $I(i)$ satisfy condition c.

The synchronisation condition is a label in form $\wedge_{j \in L} A_j{:}B_j$, where $L \subseteq I(i)$ contains indices of the sync-automata with which P_i^I wants to synchronise. For every j in L, 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.

Note that it is possible for L to be empty. Intuitively, this means that the sync-automaton P_i^I does not have any requirements on other sync-automata. We write a synchronisation condition of this form, i.e. $\wedge_{j \in \emptyset} A_j{:}B_j$, as $NOSYNC$. Move $s_i \xrightarrow{NOSYNC} t_i$ represents an autonomous move of P_i^I i.e. the sync-automaton moves without performing synchronisation.

In the special case that set A_j is empty, we shall write $true{:}B_j$. In this case the sync-automaton is willing to synchronise with any move of P_j^I satisfying B_j in the ending state. Symmetrically, if B_j is empty, A_j needs to be "matched" in the starting state, and we write $A_j{:}true$. If both A_j and B_j are empty, the sync-automaton is ready to participate in synchronisation with any move of P^I_j. To indicate which is the sync-automaton that is required to synchronise, we write this condition as $true_j{:}true_j$.

Moreover, note that multiple moves between the same pair of states are possible. Loops are covered by the definition as well, and we use an abbreviation $A_j \circlearrowleft$ for a condition of the form $A_j{:}A_j$.

On fig. 2 is the sync-automaton representing lactose from the example that we will give in Section 4. The sync-automaton has two states. For each state we display only the atomic propositions true in that state. There is a $NOSYNC$ move from between the two states and three looping moves each representing synchronisation with two other sync-automata.

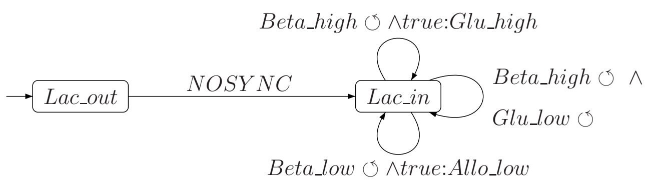


Fig. 2. P^I_{lac} – Lactose

By running in parallel sync-automata related by an interaction graph, we obtain a sync-program.

Definition 2.2 Let I be an interaction graph, where $|I| = \{1, \ldots, n\}$. The *sync-program* is a tuple $P^I = (S^I_0, P^I_1 || \ldots || P^I_n)$, where each P^I_i is a sync-automaton. Set $S^I_0 = S^0_1 \times \ldots \times S^0_n$ is the set of initial states of the sync-program.

A *sync-subprogram* represents behaviour of its sync-automata in isolation. We obtain a sync-subprogram by projecting a sync-program onto an interaction graph $J \subseteq I$. We denote this by the projection operator $\lceil J$.

Definition 2.3 Let $J \subseteq I$ be an interaction graph, where $|J| = \{j_1, \ldots, j_k\}$. Let $P^I = (S^I_0, P^I_1 || \ldots || P^I_n)$ with $P^I_i = (S_i, S^0_i, SC_i, R_i)$ for each $i \in |I|$. Then $P^I \lceil J = (S^J_0, P^J_{j_1} || \ldots || P^J_{j_k})$ with $P^J_j = (S_j, S^0_j, SC'_j, R'_j)$ for each $j \in |J|$ where

- $SC'_j = \{\wedge_{j' \in L \cap |J|} A_{j'}{:}B_{j'} \mid \wedge_{j' \in L} A_{j'}{:}B_{j'} \in SC_{j'}\}$
- $R'_j = \{s_j \xrightarrow{\wedge_{j' \in L \cap |J|} A_{j'}:B_{j'}} t_j \mid s_i \xrightarrow{\wedge_{j' \in L} A_{j'}:B_{j'}} t_j \in R_j\}$.

Initial states are $S^J_0 = S^0_{j_1} \times \ldots \times S^0_{j_n}$.

The projection contains sync-automata from J, each sync-automaton has the same states as its counterpart in P^I but synchronisation conditions on their moves concern only sync-automata from J. We remark that a sync-subprogram $P^I \lceil J$ is still a sync-program with interaction graph J, hence it can be also denoted by P^J.

2.2 Semantics

Let I be an interaction graph, where $|I| = \{1, \ldots, n\}$. An *I-state* is a tuple $s = (s_1, \ldots, s_n)$ where each s_i is a state of the sync-automaton P_i^I. An I-state represents a configuration of a program. I-state $s = \{s_1, \ldots, s_n\}$ can be projected onto a single component index $i \in |I|$ as follows: $s\lceil i = s_i$. Similarly, s projected onto $J \subseteq I$ with nodes $\{j_1, \ldots, j_k\}$, is $s\lceil J = (s_{j_1}, \ldots, s_{j_k})$.

Now we can proceed to defining the semantics of sync-programs as a labelled transition state system on I-states, called I-structure.

Definition 2.4 Let I be an interaction graph, where $|I| = \{1, \ldots, n\}$. Semantics of $P^I = (S_I^0, P_1^I|| \ldots ||P_n^I)$ is given by *I-structure* $\mathcal{M}_I = (\mathcal{S}_I, \mathcal{S}_I^0, \mathcal{R}_I)$, 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 $\mathcal{R}_I \subseteq \mathcal{S}_I \times 2^{|I|} \times \mathcal{S}_I$ is a transition relation giving the transitions of P^I. A *transition* (s, l, t) is in $\mathcal{R}_I$ iff *transition label* l is a minimal nonempty set such that

(i) for all $i \in l$ there is a move $s\lceil i \xrightarrow{\wedge_{j \in L} A_j : B_j} t\lceil i$ in P_i^I such that
- for all $j \in L$, for all $p \in A_j$: $s\lceil j(p) = tt$ and for all $p \in B_j$: $t\lceil j(p) = tt$
- $L \subseteq l$

(ii) for all $i \in |I| - l$: $s\lceil i = t\lceil i$.

A transition of the form (s, l, t) corresponds to the situation where sync-automata with indices in l perform moves and the rest stays idle.

Transition label l may contain only one index, let us assume it is i. In this case there is a move in the sync-automaton P_i^I that does not require synchronisation with other sync-automata, i.e. set L is empty. Note that conditions of (i) are satisfied vacuously. In this situation sync-automaton P_i^I performs an autonomous $NOSYNC$ move from $s\lceil i$ to $t\lceil i$.

Case in which l contains more indices corresponds to the synchronisation of the sync-automata. Sync-automata with index in l can perform a move if all their synchronisation requirements against other sync-automata are satisfied. In particular, for sync-automaton P_j^I set A_j must be satisfied in the starting state and B_j in the ending state of the transition, respectively. Moreover, inclusion of L in l guarantees that all the required sync-automata will really participate in the synchronisation.

Note that indirect synchronisation can be performed. It is possible that two sync-automata participate in the synchronisation without requiring synchronisation from each other, but both being connected through a third party. For instance, if a sync-automaton requires synchronisation with two other sync-automata, these are forced to perform a move synchronously even though they might not require synchronisation with each other directly.

The minimality requirement of the set l of indices guarantees that the synchronisation is indivisible. In other words, it is not possible that l is composed of more disjoint sets, each of which could be a label of a transition alone.

As an example of a transition, suppose that $|I| = \{1, 2, 3\}$ and sync-automaton P^I_1 contains $a \xrightarrow{A:B} b$, P^I_2 contains a move $A \xrightarrow{a:b} B$ and sync-automaton P^I_3 has a move $X \xrightarrow{\neg a:b} Y$. Then $\mathcal{M}_I$ contains a transition $([a, A, X], \{1, 2\}, [b, B, X])$ representing that sync-automata with indices 1 and 2 synchronise and P^I_3 remains idle.

If in the previous example in P^I_2 we replace move $A \xrightarrow{a:b} B$ by $A \xrightarrow{NOSYNC} B$, then $\mathcal{M}_I$ will contain two transitions. The first transition is $([a, A, X], \{2\}, [a, B, X])$ representing an autonomous move of sync-automaton P^I_2. The second transition is $([a, A, X], \{1, 2\}, [b, B, X])$ and represents the synchronisation of sync-automata with indices 1 and 2 on the moves $a \xrightarrow{A:B} b$ and $A \xrightarrow{NOSYNC} B$. It can be performed, even though the second automaton does not request synchronisation. Although this kind of synchronisation specification is likely not to be used frequently in practice, it is legitimate and covered by the semantics.

A transition (s, l, t) in an I-structure can be projected onto $J \subseteq I$ such that $l \cap |J| \neq \emptyset$ as follows: $(s, l, t)\lceil J = (s\lceil J, l \cap |J|, t, \lceil J)$.

A concept that will allow us to reason about properties of programs, is that of path.

Definition 2.5 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.

A fullpath is infinite unless for some $s^{m'}$ there is no $s^{m'+1}$ and $l^{m'}$ such that $(s^{m'}, l^{m'}, s^{m'+1}) \in \mathcal{R}_I$. Let π^m denote the suffix of π starting in m-th I-state.

For a $J \subseteq I$ let us define a J-block of π to be a maximal subsequence of π that starts and ends in a state and does not contain a transition label containing any i such that $i \in J$. Thus we can consider π to be a sequence of J-blocks with two successive J-blocks linked by a transition label l such that $l \cap |J| \neq \emptyset$ (note that a J-block can consist of a single state). It also follows that $s\lceil J = t\lceil J$ for any pair of states s, t in the same J-block. Thus, if Bl is a J-block, we define $Bl\lceil J$ to be $s\lceil J$ for some state s in Bl. We now give the formal definition of path projection. Let Bl^n denote the n-th J-block of π.

Let π be $(Bl^1, l^1, Bl^2, l^2, \ldots)$ where Bl^m is a J-block for all m. Then the path projection is given by: $\pi\lceil J = (Bl^1\lceil J, l^1 \cap |J|, Bl^2\lceil J, l^2 \cap |J|, \ldots)$.

3 Modular verification

In order to analyse the behaviour of a biological system we would like to verify properties of computation of sync-program P^I representing the system. Say that a property ϕ_J only regards part of the system, in particular the part involving only sync-automata from $J \subseteq I$. We would like to check satisfaction of ϕ_J on semantics of P^I. In order to avoid space explosion, we want to check it on smaller and more abstract semantics, in particular semantics of $P^J = P^I \lceil J$. The subprogram P^J abstracts from the behaviour of sync-automata non-present in J and poses less restrictions in terms of synchronisation requirements. Thus its semantics represents an overapproximation of the behaviour of part of P^I concerning J.

To be able to perform the verification on the smaller semantics we need to prove that every computation concerning sync-automata from J of the program P^I is present as a computation of P^J.

It is reasonable to define a computation as a fullpath in the semantics of the sync-program. We need to show that every fullpath in the semantics of P^I projected onto J is a fullpath in the semantics of P^J. Firstly, we prove that every path in $\mathcal{M}_I$ projected onto J is a path in $\mathcal{M}_J$.

Lemma 3.1 (Transition projection) *Let I be an interaction graph and $\mathcal{M}_I = (\mathcal{S}_\mathcal{I}, \mathcal{S}_0^I, \mathcal{R}_I)$ the semantics of sync-program P^I. For all I-states s, t in $\mathcal{S}_I$ and all $l \in 2^{|I|}$, 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 $P^J = P^I \lceil J$.*

Proof. Direction right to left. Suppose that for any $J \subseteq I$ such that $l \cap |J| \neq \emptyset$, $(s, l, t)\lceil J \in \mathcal{R}_J$. By taking $J = I$ we get $(s, l, t) \in \mathcal{R}_I$.

Direction left to right. Suppose that $(s, l, t) \in \mathcal{R}_I$, we will show $(s, l, t)\lceil J \in \mathcal{R}_J$ for any $J \subseteq I$ such that $l \cap |J| \neq \emptyset$.

Let i be any element from $l \cap |J|$.

- Since $i \in l$, according to the definition of semantics of P^I move $s\lceil i \xrightarrow{\wedge_{j \in L} A_j : B_j} t\lceil i \in P_i^I$. Then since P^J is subprogram of P^I, $s\lceil i \xrightarrow{\wedge_{j \in L \cap |J|} A_j : B_j} t\lceil i \in P_i^J$ by definition of subprogram.
- For all $j \in L$, for all $p \in A_j$:$s\lceil j(p) = tt$ and for all $p \in B_j$:$t\lceil j(p) = tt$ and this implies satisfaction of the condition for all $j \in L \cap |J|$.
- $L \cap |J| \subseteq l \cap |J|$ because $L \subseteq l$.
- Also since for all $i \in |I| - l : s\lceil i = t\lceil i$, it holds for subset $|J| \cap (|I| - l) = |J| - l \cap |J|$.
- Minimality of $l \cap |J|$ comes from minimality of l, namely if $l \cap |J|$ is not minimal then neither l is minimal.

Thus, by definition of semantics of P^J, $(s\lceil J, L \cap |J|, t\lceil J) = (s, l, t)\lceil J \in \mathcal{R}_J$. □

Lemma 3.2 (Path projection) *Let I be an interaction graph and $\mathcal{M}_I$ semantics of sync-program P^I. For every $J \subseteq I$ if π is a path in $\mathcal{M}_I$ then $\pi\lceil J$ is a path in $\mathcal{M}_J$, where $\mathcal{M}_J$ is the semantics of sync-program $P^J = P^I \lceil J$.*

Proof. Let $\pi = (Bl^1, l^1, Bl^2, l^2, \ldots)$ be a path in $(M)_I$ and Bl^m J-blocks for all m. By s^m and t^m denote first and last state of Bl^m, respectively. By definition of I-structure we have that transition (t^m, l^m, s^{m+1}) is in $\mathcal{M}_I$ for all m. By transition projection lemma transition $(t^m, l^m, s^{m+1})\lceil J = (t^m\lceil J, l^m \cap |J|, s^{m+1}\lceil J)$ is in $\mathcal{M}_J$ for all m. Now since $s^m\lceil J = t^m\lceil J$ for all m, we get that $(s^m\lceil J, l^m \cap |J|, s^{m+1}\lceil J)$ in $\mathcal{M}_J$ for all m. Hence sequence $(s^1\lceil J, l^1 \cap |J|, s^2\lceil J, l^2 \cap |J|, \ldots)$ satisfies the definition of a path in $\mathcal{M}_J$. □

However, with computation defined as a fullpath, violation of the desired computation preservation might occur. In particular, violation arises when an independent partition P of sync-program P^I exists that can be executed forever, while not allowing execution of other enabled sync-automata outside P. Consider a fullpath π in $\mathcal{M}_I$ and a state t from which on only sync-automata in P are executed. When projecting π onto $J = (I - P)$ composed only of idle sync-automata, a finite path $\pi\lceil J$ is obtained. However, as in t some automata from J are enabled but not executed, in $\mathcal{M}_J$ they can be executed and thus a path ending in $t\lceil J$ is not a fullpath. Hence, π is a computation of P^I but $\pi\lceil J$ is not computation of P^J.

We need to refine the definition of computation, so that for all J a computation of P^I projected onto J will be a computation in P^J. We restrict ourselves to a special class of "fair" computations, in particular those in which every sync-automaton is executed infinitely many times. We define fairness as a property of paths in $\mathcal{M}_I$.

Definition 3.3 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.

Note that every fair path is infinite and each component involved in a fair path must have an infinite behaviour. Finite behaviours of components can be simulated by adding looping moves to the final states.

For the systems we aim to describe, the fairness assumption is reasonable since we regard a behaviour of biological system correct when all components are able to perform their function. Moreover, there is a class of systems where all fullpaths are fair, we provide a non-trivial example of such a system in Section 4.

We remark, that transient behaviour of the system can be studied by considering only a portion its infinite behaviour.

Now we prove, that this definition of fairness guarantees preservation of computation under projection.

Lemma 3.4 (Fullpath projection) *Let $J \subseteq I$ be an interaction graph. If π is a fair fullpath in $\mathcal{M}_I$, then $\pi\lceil J$ is a fair fullpath in $\mathcal{M}_J$.*

Proof. By path projection lemma $\pi\lceil J$ it is a path in $\mathcal{M}_J$. Since π is a fair path in $\mathcal{M}_I$ by definition of path projection we get that $\pi\lceil J$ is a fair path in $\mathcal{M}_J$. From the definition of fairness follows that every fair path is infinite, i.e. it is a fullpath. $\square$

Following Attie, we assume the logic ACTL for specification of properties and we show that all ACTL properties that hold in a semantics of sync-subprogram also hold in the original sync-program. ACTL is the "universal fragment" of CTL which results from CTL by restricting negation to propositions and eliminating the existential path quantifier.

Definition 3.5 *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 $AX_j f$, $A[fUg]$ 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: $AFf \equiv A[true\ Uf]$ and $AGf \equiv A[f\ U_w false]$.

Properties expressible by ACTL formulae represent a significant class of properties investigated in systems biology literature as identified in [15], such as properties concerning exclusion (*It is not possible for a state S to occur*), necessary consequence (*If a state S_1 occurs, then it is necessarily followed by a state S_2*), and necessary persistence (*A state S must persist indefinitely*). Oscillatory behaviour [4] is describable by an ACTL formula as well.

Occurrence, possible consequence, sequence and possible persistence are of inherently existential nature, and are not expressible in ACTL.

Definition of the semantics of ACTL formulae on the I-structure follows. Note that only fair fullpaths are considered.

Definition 3.6 *Semantics of ACTL.* We define $\mathcal{M}_I, s \models f$ (resp. $\mathcal{M}_I, \pi \models f$) meaning that f is true in structure $\mathcal{M}_I$ at state s (resp fair fullpath π). We define $\models$ inductively:

- $\mathcal{M}_I, s^1 \models true$. $\mathcal{M}_I, s^1 \not\models false$. $\mathcal{M}_I, s^1 \models p$ iff $s^1(p) = tt$.
  $\mathcal{M}_I, s^1 \models \neg p$ iff $s^1(p) = ff$.
- $\mathcal{M}_I, s^1 \models f \wedge g$ iff $\mathcal{M}_I, s^1 \models f$ and $\mathcal{M}_I, s^1 \models g$.
  $\mathcal{M}_I, s^1 \models f \vee g$ iff $\mathcal{M}_I, s^1 \models f$ or $\mathcal{M}_I, s^1 \models g$.
- $\mathcal{M}_I, s^1 \models Af$ iff for every fair fullpath $\pi = (s^1, l^1, \ldots)$ in $\mathcal{M}_I : \mathcal{M}_I, \pi \models f$.
- $\mathcal{M}_I, \pi \models f$ iff $\mathcal{M}_I, s^1 \models f$.
- $\mathcal{M}_I, \pi \models f \wedge g$ iff $\mathcal{M}_I, \pi \models f$ and $\mathcal{M}_I, \pi \models g$.
  $\mathcal{M}_I, \pi \models f \vee g$ iff $\mathcal{M}_I, \pi \models f$ or $\mathcal{M}_I, \pi \models g$.
- $\mathcal{M}_I, \pi \models X_l f$ iff if $(s^1, l', s^2) \in \mathcal{R}_I$ and $l' \supseteq l$ then $\mathcal{M}_I, \pi^2 \models f$.

- $\mathcal{M}_I, \pi \vDash fUg$ iff there exists $m \in \mathbb{N}$ such that $\mathcal{M}_I, \pi^m \vDash g$ and for all $m' < m : \mathcal{M}_I, \pi^{m'} \vDash f$.
- $\mathcal{M}_I, \pi \vDash fU_wg$ iff for all $m \in \mathbb{N}$, if $\mathcal{M}_I, \pi^{m'} \nvDash g$ for all $m' < m$ then $\mathcal{M}_I, \pi^m \vDash f$.

Now we give the main theorem of the paper.

Theorem 3.7 (Property preservation) *Let $J \subseteq I$ be an interaction graph, s an I-state and f an $ACTL_J$ property. If $\mathcal{M}_J, s\lceil J \vDash_\Phi f$ then $\mathcal{M}_I, s \vDash_\Phi f$.*

Proof. By induction on the structure of f (for all s).

$f = p$. By definition of state projection and the fact that AP_is are pairwise disjoint, for all atomic propositions p from AP_J we get that $\mathcal{M}_J, s\lceil J \vDash_\Phi p$ iff $\mathcal{M}_I, s \vDash_\Phi p$. Analogously for $f = \neg p$.

$f = g \wedge h$. From the assumption $\mathcal{M}_J, s\lceil J \vDash_\Phi g \wedge h$ by CTL semantics, $\mathcal{M}_J, s\lceil J \vDash_\Phi g$ and $\mathcal{M}_J, s\lceil J \vDash_\Phi h$. By induction hypothesis $\mathcal{M}_I, s \vDash_\Phi g$ and $\mathcal{M}_I, s \vDash_\Phi h$. Hence, by CTL semantics $\mathcal{M}_I, s \vDash_\Phi g \wedge h$. Case $f = g \vee h$ is proved analogously.

$f = AXg$. Let $\pi = (s, l^1, s^2, l^2, \ldots)$ be an arbitrary fair fullpath starting in s. If $l^1 \cap |J| \supsetneq l$ then since $|J| \supseteq l$ also $l^1 \supsetneq l$ and by CTL semantics $\mathcal{M}_I, \pi \vDash_\Phi X_lg$ vacuously. If $l^1 \cap |J| \supseteq l$ then by definition of path and by transition projection lemma we have that $(s^1\lceil J, l^1 \cap |J|, s^2\lceil J) \in \mathcal{R}_J$. Now by CTL semantics $\mathcal{M}_J, \pi^2\lceil J \vDash_\Phi g$ and by induction hypothesis $\mathcal{M}_I, \pi^2 \vDash_\Phi g$. Hence $\mathcal{M}_I, \pi \vDash_\Phi X_lg$. Since π was an arbitrary fair fullpath starting in s, we proved $\mathcal{M}_I, s \vDash_\Phi AX_lg$.

$f = A[gU_wh]$. Let π be an arbitrary fair fullpath starting in s. We establish $\mathcal{M}_I, \pi \vDash_\Phi [gU_wh]$. By fullpath projection lemma $\pi\lceil J$ is a fair fullpath in $\mathcal{M}_J$, hence by the assumption $\mathcal{M}_J, \pi\lceil J \vDash_\Phi [gU_wh]$. There are two cases:

(i) $\mathcal{M}_J, \pi\lceil J \vDash_\Phi Gg$. Let t be any state along π. By CTL semantics $\mathcal{M}_J, t\lceil J \vDash_\Phi g$. by induction hypothesis we have $\mathcal{M}_I, t \vDash_\Phi g$. Since t was an arbitrary state of π, we get $\mathcal{M}_I, \pi \vDash_\Phi Gg$ and thus $\mathcal{M}_I, \pi \vDash_\Phi gU_wh$.

(ii) $\mathcal{M}_J, \pi\lceil J \vDash_\Phi [gUh]$. Let $s_J^{m''}$ be the first state along $\pi\lceil J$ that satisfies h. Then there is at least one state $s^{m''}$ along π such that $s^{m''}\lceil J = s_J^{m''}$. Let $s^{m'}$ be first such state. By induction hypothesis $\mathcal{M}_I, s^{m'} \vDash_\Phi h$. From the definition of path projection any s^m with $m < m'$ projects to $s^m\lceil J$ that is before $s_J^{m'}$ in $\pi\lceil J$. By the assumption $\mathcal{M}_J, s^m\lceil J \vDash_\Phi g$, hence by induction hypothesis $\mathcal{M}_I, s^m \vDash_\Phi g$. By CTL semantics we get $\mathcal{M}_I, \pi \vDash_\Phi gUh$.

In both cases we showed $\mathcal{M}_I, \pi \vDash_\Phi gU_wh$. Since π was arbitrary fair fullpath starting in s, we conclude $\mathcal{M}_I, s \vDash_\Phi A[gU_wh]$.

$f = A[gUh]$. Let π be an arbitrary fair fullpath starting in s. By fullpath projection lemma $\pi\lceil J$ is a fair fullpath in $\mathcal{M}_J$ and by the assumption $\mathcal{M}_J, \pi\lceil J \vDash_\Phi [gUh]$. By the above case we get $s \vDash_\Phi A[gUh]$. □

4 Application

We demonstrate our approach on modelling and verification of the *lac* operon regulation taken from [16].

4.1 Lac *operon regulation*

Bacteria have a simple general mechanism for coordinating the regulation of genes encoding products that participate in a set of related processes: these genes are clustered on the chromosome and are transcribed together. The gene cluster plus additional sequences that function together in regulation are called an operon [14].

The *lac* operon contains three genes related to lactose metabolism. The *lac* Z, Y and A genes encode β-galactosidase, galactoside permease and thiogalactoside transacetylase, respectively. β-galactosidase converts lactose to galactose and glucose or to allolactose. Galactoside permease transports lactose into the cell and thiogalactoside transacetylase appears to modify toxic galactosides to facilitate their removal from the cell.

In the absence of lactose, the *lac* operon genes are repressed–they are transcribed at a basal level. This negative regulation is done by a molecule called Lac repressor, which binds to the operon, blocking the activity of RNA polymerase. The binding sites are called operators, main operator is named O_1. The *lac* operon has two secondary binding sites for the Lac repressor: O_2 and O_3. To repress the operon, the Lac repressor binds to both the main operator and one of the two secondary sites.

When cells are provided with lactose, the *lac* operon is induced. An inducer molecule binds to a specific site on the Lac repressor, causing dissociation of the repressor from the operators. The inducer in the *lac* operon system is allolactose, an isomer of lactose. When unrepressed, transcription of *lac* genes is increased, but not at its highest level.

In addition, availability of glucose, the preferred energy source of bacteria, affects the expression of the *lac* genes. Expressing the genes for proteins that metabolise sugars such as lactose is wasteful when glucose is abundant. The *lac* operon deals with it through a positive regulation. The effect of glucose is mediated by cAMP, as a coactivator, and an activator protein known as cAMP receptor protein (CRP). When glucose is absent, CRP-cAMP binds to a site near the *lac* promoter and stimulates RNA transcription. In the presence of glucose, the synthesis of cAMP is inhibited and cAMP declines. Binding to DNA declines, thereby decreasing the expression of the *lac* operon.

CRP-cAMP is therefore a positive regulatory element responsive to glucose levels, whereas the Lac repressor is a negative regulatory element responsive to lactose. Consequently, strong induction of the *lac* operon requires both lactose (to inactivate the Lac repressor) and a lowered concentration of glucose (to trigger an increase in cAMP and increase binding of cAMP to CRP).

4.2 The model

We will fix the set of indices $Id = \{lac, \beta, allo, op, rep, pos, glu\}$ representing lactose, β-galactosidase, allolactose, *lac* operon, repressor, CRP-cAMP regulation and glucose, respectively. The interaction graph I is on fig. 1. For the sake of simplicity we do not model the activities of galactoside permease and thiogalactoside transacetylase.

We provide a sync-automaton for each biological component. In particular lactose is modelled by P^I_{lac} with $AP_{lac} = \{Lac\_none, Lac_{low}\}$, β-galactosidase by P^I_β with $AP_{glu} = \{Beta\_low, Beta\_high\}$ and allolactose by P^I_{allo} with $AP_{allo} = \{Allo\_none, Allo\_low\}$. The *lac* operon is represented by P^I_{op} with $AP_{op} = \{Act, Rep\}$, *lac* repressor by P^I_{rep} with $AP_{rep} = \{B1, B2, B3, Ballo\}$, the positive regulation by P^I_{pos} with $AP_{pos} = \{cAMP\_high, CRP{-}cAMP\}$ and finally glucose by P^I_{glu} with $AP_{glu} = \{Glu\_high, Glu\_low\}$.

Sync-automaton P^I_{lac} (fig. 2 in Section 2) has two states, mappings of the set of atomic propositions AP_{lac} to $\{tt, ff\}$. For each state we display only the atomic propositions true in that state. Initially, there is no lactose is in the cell. External lactose entering the cell is modelled as a $NOSYNC$ move because it is caused by mechanisms that are not considered in our model. When lactose is present, it can be transformed to glucose in presence of β-galactosidase enzyme. This is modelled as a synchronisation with P^I_{glu} and P^I_β. If this enzyme is absent, lactose is transformed to allolactose instead.

In sync-automaton P^I_β (fig. 3) β-galactosidase has two states representing its concentration level which are affected by activation and repression of *lac* operon P^I_{op}. When reacting with lactose, this enzyme, at low level, can produce allolactose or, at high level, can produce glucose and galactose. Since galactose does not participate in regulation we do not include it in our model.

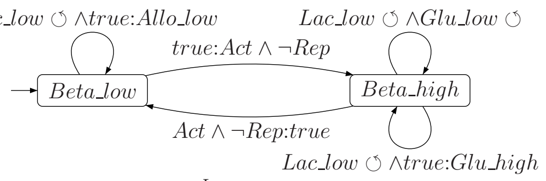


Fig. 3. P^I_β – β-galactosidase

Allolactose P^I_{allo} (fig. 4) can be present at low concentration in the cell or abstent. Its level is increased as a result of reaction of lactose with β-galactosidase enzyme. When present, it can bind to *lac* repressor P^I_{rep} and its concentration will reduce.

The *lac* operon P^I_{op} (fig. 5) has four states, all possible truth value assignments to AP_{op}. Atomic proposition *Act*, *Rep* represent that the *lac* operon ac-

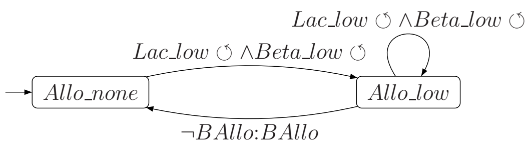


Fig. 4. P^I_{allo} – Allolactose

tivatated and repressed, respectively. Repression and unrepression (horizontal moves in fig. 5) are determined by negative regulation P^I_{rep} while activation and inactivation (vertical moves in fig. 5) by positive regulation P^I_{pos}. Note that full transcription of the operon genes occurs only when both unrepressed and activated (state $Act, \neg Rep$). This state also determines the concentration of β-galactosidase.

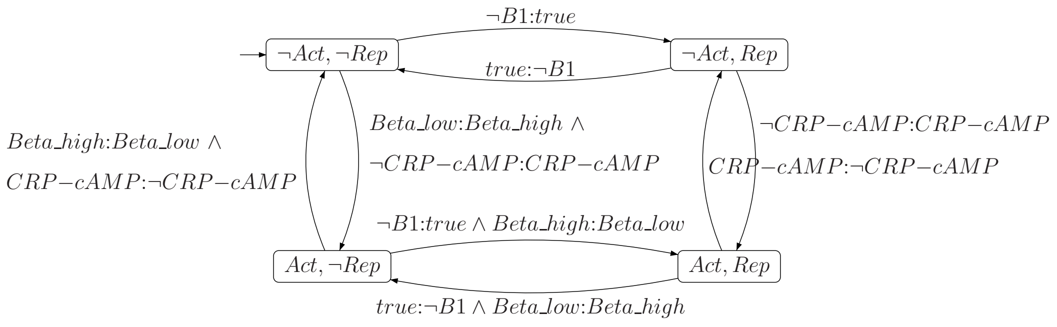


Fig. 5. P^I_{op} – Lac operon

The *lac* repressor P^I_{rep} (fig. 6) has five states. After binding of *lac* repressor protein to O_1 site, it might bind either to O_2 or O_3 site. These bindings repress the operon. When the inducer allolactose binds to the repressor, it releases operator sites and unrepresses the operon.

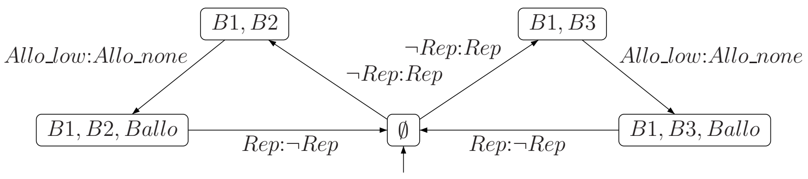


Fig. 6. P^I_{rep} – Lac repressor protein (Negative regulation)

The positive regulation P^I_{pos} (fig. 7) works as follows. When glucose level is low, cAMP concentration will be increased. Coactivator CRP creates a complex CRP-cAMP that binds to *lac* operon, stimulating the transcription. When the glucose concentration is increased, cAMP level will decrease and CRP-cAMP releases the operon site, deactivating the transcription.

In P^I_{glu} (fig. 8) glucose concentration can be high or low. The decrease of its concentration depends on factors that are not modelled. Increase of concentration can occur via reaction of lactose and β-galactosidase. It is non-deterministically decided when the concentration level of glucose is considered

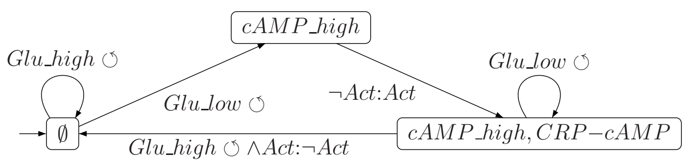
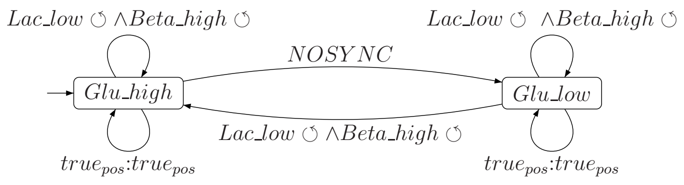


Fig. 7. P^I_{pos} – CRP-cAMP (Positive regulation)



Lac_low ↻ $\wedge Beta\_high$ ↻
Lac_low ↻ $\wedge Beta\_high$ ↻
$NOSYNC$
Glu_high
Glu_low
Lac_low ↻ $\wedge Beta\_high$ ↻
$true_{pos}{:}true_{pos}$
$true_{pos}{:}true_{pos}$

Fig. 8. P^I_{glu} – Glucose

high enough to pass to state high. In addition, this component can be queried for the concentration level by P^I_{pos}.

The sync-program describing the whole system is obtained by running all sync-automata in parallel: $P^I = (S^I_0, P^I_{lac}||P^I_\beta||P^I_{allo}||P^I_{op}||P^I_{rep}||P^I_{pos}||P^I_{glu})$. Set of initial states is a combination of sets of initial states of respective sync-automata.

4.3 Experiments

We check some known properties of *lac* operon regulation. The check can be performed by using any CTL model checking algorithm on the semantics of indicated subprograms. Satisfaction of a property on $\mathcal{M}_J$ for $J \subseteq I$ guarantees its satisfaction in $\mathcal{M}_I$.

The property *"The allolactose bound to the repressor implies that the operon is repressed"* represents the exclusion type as identified in [15]. The formula $AG(Ballo \rightarrow Rep)$ is verifiable on the semantics of $P^{rep,op}$. A slightly more complicated formula is needed to express that *"The increase of allolactose concentration can only be mediated by β-galactosidase in low concentration"*. The formula $AG(Allo\_none \wedge Beta\_high \rightarrow A(\neg Allo\_low U Beta\_low))$ is true in the semantics of $P^{allo,\beta}$.

The oscillation property *"The operon will necessarily oscillate between repressed and unrepressed state"* expressed by $AG((rep \rightarrow AF\neg rep) \wedge (\neg rep \rightarrow AF rep))$ is true in $\mathcal{M}_I$, but the verification in $\mathcal{M}_{op}$ fails. Thus the property preservation theorem cannot be invoked. However, by inspecting the model we can understand that by taking into consideration another component that is not mentioned in the formula we could succeed in verification. Indeed, the formula is true in the semantics of $P^{op,rep}$. Note the important role of fairness for satisfaction of this property that requires that sync-automaton is executed infinitely many times.

The property that demonstrates correctness of the model of *lac* operon regulation can be stated as follows: *"When the glucose concentration drops, the* lac *operon will eventually be fully expressed"*. Encoded as an $ACTL_{glu,op}$ formula $AG(Glu\_low \rightarrow AF(Act \wedge \neg Rep))$, it holds in $\mathcal{M}_I$ but cannot be verified for any of the subprograms of P^I, as it depends on activities of every component in the model.

We can prove in a modular manner a partial property that states that the negative regulation mechanism works as expected: *"When glucose concentration is low and $CRP{-}cAMP$ complex is bound to the operon, the* lac *operon will eventually be fully expressed"*. Formula $AG(CRP\_cAMP \wedge Glu\_low \rightarrow AF(Act \wedge \neg Rep))$ can be verified on the semantics of $P^{pos,glu,op,\beta}$.

5 Discussion

We have presented an approach for modelling and modular verification of properties of biological systems.

We have shown that truth of ACTL formulae is preserved from sync-subprograms to the program. Failure to verify a property in sync-subprograms does not help establishing its satisfaction in the whole program. However, it is worth noting, that in some cases model inspection aids finding a larger sync-subprogram that allows for successful verification of the property.

Preservation of falsehood of ACTL formulas amounts to full CTL preservation and can be obtained only under bisimilarity [9]. For application in systems biology see [16].

A subprogram can be looked at as an abstraction of the program. In [9] property preservation is investigated in the framework of abstract interpretation. Other approaches consider reducing transition systems so that a particular property is preserved [2]. In [6] the authors consider a modular approach to quantitative model checking in a biological context.

We believe that our property preservation theorem can be extended to preserve all ACTL* properties in the way used in [12]. We plan to improve our approach with a weaker notion of fairness, in the line of [1], and with the possibility of describing dynamic systems with run-time creation of sync-automata. A quantitative extension of the method would be desirable in order to describe the systems more precisely.

References

[1] Attie, P. C., *Synthesis of large dynamic concurrent programs from dynamic specifications*, CoRR **abs/0801.1687** (2008).

[2] Barbuti, R., N. D. Francesco, A. Santone and G. Vaglini, *Loreto: A tool for reducing state explosion in verification of lotos programs*, Softw., Pract. Exper. **29** (1999), pp. 1123–1147.

[3] Barbuti, R., A. Maggiolo-Schettini, P. Milazzo and A. Troina, *A calculus of looping sequences for modelling microbiological systems*, Fundam. Inf. **72** (2006), pp. 21–35.

[4] Calzone, L., N. Chabrier-Rivier, F. Fages and S. Soliman, *Machine learning biochemical networks from temporal logic properties*, Transactions on Computational Systems Biology VI (2006), pp. 68–94.

[5] Cardelli, L., *Brane calculi*, Computational Methods in Systems Biology (2005), pp. 257–278.

[6] Ciocchetta, F., M. L. Guerriero and J. Hillston, *Investigating modularity in the analysis of process algebra models of biochemical systems*, CoRR **abs/1002.4063** (2010).

[7] Ciocchetta, F. and J. Hillston, *Bio-pepa: A framework for the modelling and analysis of biological systems*, Theor. Comput. Sci. **410** (2009), pp. 3065–3084.

[8] Clarke, E. M. and E. A. Emerson, *Design and synthesis of synchronization skeletons using branching-time temporal logic*, in: *Logic of Programs, Workshop* (1982), pp. 52–71.

[9] Dams, D., R. Gerth and O. Grumberg, *Abstract interpretation of reactive systems*, ACM Trans. Program. Lang. Syst. **19** (1997), pp. 253–291.

[10] Danos, V. and C. Laneve, *Formal molecular biology*, Theor. Comput. Sci. **325** (2004), pp. 69–110.

[11] Fages, F., S. Soliman and N. Chabrier-Rivier, *Modelling and querying interaction networks in the biochemical abstract machine biocham*, Journal of Biological Physics and Chemistry **4** (2004), pp. 64–73.

[12] Grumberg, O. and D. E. Long, *Model checking and modular verification*, ACM Trans. Program. Lang. Syst. **16** (1994), pp. 843–871.

[13] Heath, J., M. Kwiatkowska, G. Norman, D. Parker and O. Tymchyshyn, *Probabilistic model checking of complex biological pathways*, Theor. Comput. Sci. **391** (2008), pp. 239–257.

[14] Jacob, F. and J. Monod, *Genetic regulatory mechanisms in the synthesis of proteins.*, J Mol Biol **3** (1961), pp. 318–356.

[15] Monteiro, P. T., D. Ropers, R. Mateescu, A. T. Freitas and H. de Jong, *Temporal logic patterns for querying dynamic models of cellular interaction networks*, Bioinformatics **24** (2008), pp. i227–233.

[16] Pinto, M. C., L. Foss, J. M. Mombach and L. Ribeiro, *Modelling, property verification and behavioural equivalence of lactose operon regulation*, Computers in Biology and Medicine **37** (2007), pp. 134–148.

[17] Priami, C., A. Regev, E. Shapiro and W. Silverman, *Application of a stochastic name-passing calculus to representation and simulation of molecular processes*, Inf. Process. Lett. **80** (2001), pp. 25–31.

[18] Regev, A., E. M. Panina, W. Silverman, L. Cardelli and E. Shapiro, *Bioambients: an abstraction for biological compartments*, Theor. Comput. Sci. **325** (2004), pp. 141–167.