

[Electronic Notes in Theoretical Computer Science 268 (2010) 61–75](http://dx.doi.org/10.1016/j.entcs.2010.12.006)

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Modular Verification of Interactive Systems with an Application to Biology

# Peter Dr´abik [1](#_bookmark0) Andrea Maggiolo-Schettini [2](#_bookmark0) Paolo Milazzo [3](#_bookmark0)

*Dipartimento di Informatica, Universita` 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

# Introduction

Many formalisms originally developed by computer scientists to model systems of interacting components have been applied to biology, also with extensions to al- low more precise descriptions of the biological behaviours [[3](#_bookmark13),[5](#_bookmark15),[7](#_bookmark17),[10](#_bookmark18),[17](#_bookmark27),[18](#_bookmark28)]. Model checking permits the verification of properties of a system (expressed as logical for- mulas) by exploring all the possible behaviours of the system. It has been success- fully 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](#_bookmark23),[11](#_bookmark19),[7](#_bookmark17)].

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

1 Email: [drabik@di.unipi.it](mailto:drabik@di.unipi.it)

2 Email: [maggiolo@di.unipi.it](mailto:maggiolo@di.unipi.it)

3 Email: [milazzo@di.unipi.it](mailto:milazzo@di.unipi.it)

1571-0661 © 2010 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

[doi:10.1016/j.entcs.2010.12.006](http://dx.doi.org/10.1016/j.entcs.2010.12.006)

identified in Grumberg and Long [[12](#_bookmark22)] as ACTL, the universal fragment of CTL temporal logic. A technique proposed by Attie [[1](#_bookmark11)] exploits the preservation of these properties in order to verify concurrent programs and synthesise systems from spec- ifications. Attie uses a formalism called synchronisation skeletons [[8](#_bookmark20)], 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 skele- tons 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 simul- taneously.

To be able to apply the proposed modular verification technique, the systems under consideration are subject to some restrictions. In particular, we assume infi- nite 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 prop- erties that can be verified in a modular way.

# Sync-programs

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

* 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 *Ci* and *Cj 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](#_bookmark2)), where the nodes are indices from *Ids* and there is an edge between nodes *i* and *j* if components *Ci* and *Cj* 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* ∈ |*I*| | *iIj*}. By *J* ⊆ *I* we denote a connected subgraph *J* of *I*.

With every component *Ci* a set *APi* 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* /= *j* then *APi* ∩ *APj* = ∅.

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

Definition 2.1 A *sync-automaton PI* , where *i* is a component index and *I* is an

*i*

Operon

Repressor

CRP-cAMP

Glucose

Allolactose

*β*-galactosidase

Lactose

Fig. 1. Interaction graph

interaction graph, is a tuple (*Si, S*0*, SCi, Ri*):

*i*

* *Si* ⊆ P(*APi*) is the set of *states*
* *S*0 ⊆ *Si* is the set of *initial states*

*i*

* *SCi* is the set of labels of the form ∧*j*∈*LAj*:*Bj* where *L* ⊆ *I*(*i*) and *Aj, Bj* are sets of atomic propositions drawn from *APj* or their negations. A label from *SCi* is called a *synchronisation condition*
* *Ri* ⊆ *Si* × *SCi* × *Si* are the *moves* between states.

Each state of a sync-automaton *PI* is a truth value assignment to atomic propo- sitions of component *Ci*. Each move is labelled by a synchronisation condition. We

*i*

denote a move from state *s* to state *t* with a label *c* by *s c ti*. The move from

*i i i* −→

state *si* to *ti* with label *c* intuitively means that automaton *PI* can move from *si* to

*i*

*ti* if the activities of automata in *I*(*i*) satisfy condition *c*.

The synchronisation condition is a label in form ∧*j*∈*LAj*:*Bj*, where *L* ⊆ *I*(*i*)

contains indices of the sync-automata with which *PI*

*i*

wants to synchronise. For

every *j* in *L*, sets of propositions *Aj* and *Bj* are to be satisfied in the starting and ending state, respectively, of the concurrently performed move of *PI* . In other

*j*

*j*

words, *Aj*:*Bj* in a label of a move of *PI*

*i*

says that every move in *PI*

that can be

performed in parallel with this move of *PI* is obliged to lead from a state satisfying

*i*

*Aj* to a state satisfying *Bj*.

Note that it is possible for *L* to be empty. Intuitively, this means that the

sync-automaton *PI*

*i*

does not have any requirements on other sync-automata. We

write a synchronisation condition of this form, i.e. ∧*j*∈∅*Aj*:*Bj*, as *NOSY NC*. Move

*s NOSY NC* *I*

*i* −−−−−−−→ *ti* represents an autonomous move of *Pi* i.e. the sync-automaton moves

without performing synchronisation.

In the special case that set *Aj* is empty, we shall write *true*:*Bj*. In this case

the sync-automaton is willing to synchronise with any move of *PI* satisfying *Bj*

*j*

in the ending state. Symmetrically, if *Bj* is empty, *Aj* needs to be “matched” in the starting state, and we write *Aj*:*true*. If both *Aj* and *Bj* are empty, the sync- automaton is ready to participate in synchronisation with any move of *PI* . To indicate which is the sync-automaton that is required to synchronise, we write this condition as *truej*:*truej*.

*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 *Aj* Ç for a condition of the form *Aj*:*Aj*.

On fig. [2](#_bookmark3) is the sync-automaton representing lactose from the example that we

will give in Section [4](#_bookmark4). The sync-automaton has two states. For each state we display only the atomic propositions true in that state. There is a *NOSY NC* move from between the two states and three looping moves each representing synchronisation with two other sync-automata.

*Beta high* Ç ∧*true*:*Glu high*



*Lac out*

*NOSY NC*

*Lac in*

*Beta high* Ç ∧

*Glu low* Ç

*Beta low* Ç ∧*true*:*Allo low*

Fig. 2. *PI*

*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*,... , n*}. The *sync- program* is a tuple *PI* = (*SI ,PI* || *...* ||*P I* ), where each *PI* is a sync-automaton. Set

0 1 *n* *i*

*SI* = *S*0 × *...* × *S*0 is the set of initial states of the sync-program.

0 1 *n*

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* ⊆ *I*. We denote this by the projection operator T*J* .

Definition 2.3 Let *J* ⊆ *I* be an interaction graph, where |*J* | = {*j*1*,... , jk*}. Let

*PI* = (*SI,PI* || *...* ||*P I* ) with *PI* = (*Si, S*0*, SCi, Ri*) for each *i* ∈ |*I*|. Then *PI* T*J* =

0 1 *n i* *i*

(*SJ ,PJ* || *...* ||*P J* ) with *PJ* = (*Sj, S*0*, SC*j *, R*j ) for each *j* ∈ |*J* | where

0 *j*1 *jk j*

*j j j*

* + - *SC*j = {∧*j*′∈*L*∩|*J* |*Aj*′ :*Bj*′ | ∧*j*′∈*LAj*′ :*Bj*′ ∈ *SCj*′ }

*j*

* + - *R*j = {*s*

*j*

Λ*j*′∈*L*∩|*J* |*Aj*′ :*Bj*′

*j* −−−−−−−−−−→

*tj* | *s*

Λ*j*′∈*LAj*′ :*Bj*′

*i* −−−−−−−−→ *tj*

∈ *Rj*}.

Initial states are *SJ* = *S*0 × *...* × *S*0 .

0 *j*1 *jn*

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

* 1. *Semantics*

Let *I* be an interaction graph, where |*I*| = {1*,... , n*}. An *I-state* is a tuple *s* = (*s*1*,... , sn*) where each *si* is a state of the sync-automaton *PI* . An *I*-state represents a configuration of a program. *I*-state *s* = {*s*1*,... , sn*} can be projected onto a single component index *i* ∈ |*I*| as follows: *s*[*i* = *si*. Similarly, *s* projected onto *J* ⊆ *I* with nodes {*j*1*,... , jk*}, is *s*[*J* = (*sj*1 *,... , sjk* ).

*i*

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*,... , n*}. Semantics of *PI* = (*S*0*,PI* || *...* ||*P I* ) is given by *I-structure* M*I* = (S*I ,* S0*,* R*I* ), where S*I* is

*I* 1 *n* *I*

a set of *I*-states, S0 ⊆ S*I* is the set of initial states and R*I* ⊆ S*I* × P(|*I*|) × S*I* is

*I*

a transition relation giving the transitions of *PI* . A *transition* (*s, l, t*) is in R*I* iff

*transition label l* is a minimal nonempty set such that

Λ*j*∈*LAj* :*Bj*

1. for all *i* ∈ *l* there is a move *s*[*i* −−−−−−−→

*t*[*i* in *PI* such that

* + for all *j* ∈ *L*, for all *p* ∈ *Aj*: *s*[*j*(*p*)= *tt* and for all *p* ∈ *Bj*: *t*[*j*(*p*)= *tt*

*i*

* + *L* ⊆ *l*

1. for all *i* ∈ |*I*|− *l*: *s*[*i* = *t*[*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 *PI*

*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 *PI*

*i*

performs an autonomous

*NOSY NC* move from *s*[*i* to *t*[*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 syn- chronisation requirements against other sync-automata are satisfied. In particular, for sync-automaton *PI* set *Aj* must be satisfied in the starting state and *Bj* 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.

*j*

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 synchro- nisation 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

*PI* contains *a A*:*B b*, *PI* contains a move *A a*:*b B* and sync-automaton *PI* has a

1 −−→ 2 −−→ 3

move *X* −¬−*a*→:*b*

*Y* . Then M*I* contains a transition ([*a, A, X*]*,* {1*,* 2}*,* [*b, B, X*]) repre-

senting that sync-automata with indices 1 and 2 synchronise and *PI* remains idle.

3

If in the previous example in *PI* we replace move *A* −*a*−→:*b B* by *A* −*N*−−*O*−*S*−*Y*−*N*−→*C B*,

2

then M*I* will contain two transitions. The first transition is ([*a, A, X*]*,* {2}*,* [*a, B, X*]) representing an autonomous move of sync-automaton *PI* . The second transition is ([*a, A, X*]*,* {1*,* 2}*,* [*b, B, X*]) and represents the synchronisation of sync-automata

2

with indices 1 and 2 on the moves *a* −*A*−:→*B*

*b* and *A* −*N*−−*O*−*S*−*Y*−*N*−→*C*

*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* ⊆ *I* such that

*l* ∩ |*J* | /= ∅ as follows: (*s, l, t*)[*J* = (*s*[*J, l* ∩ |*J* |*, t,* [*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 M*I* is a sequence of *I*-states and transition labels *π* = (*s*1*, l*1*, s*2*, l*2*,.. .*) such that for all *m*, (*sm, lm, sm*+1) ∈ R*I* . A *fullpath* is a maximal path.

A fullpath is infinite unless for some *sm*′ there is no *sm*′+1 and *lm*′ such that (*sm*′ *, lm*′ *, sm*′+1) ∈ R . Let *πm* denote the suffix of *π* starting in *m*-th *I*-state.

*I*

For a *J* ⊆ *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* ∈ *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* ∩ |*J* | /= ∅ (note that a *J* -block can consist of a single state). It also follows that *s*[*J* = *t*[*J* for any pair of states *s, t* in the same *J* -block. Thus, if *Bl* is a *J* -block, we define *Bl*[*J* to be *s*[*J* for some state *s* in *Bl*. We now give the formal definition of path projection. Let *Bln* denote the *n*-th *J* -block of *π*.

Let *π* be (*Bl*1*, l*1*, Bl*2*, l*2*,.. .*) where *Blm* is a *J* -block for all *m*. Then the path projection is given by: *π*[*J* = (*Bl*1[*J, l*1 ∩ |*J* |*, Bl*2[*J, l*2 ∩ |*J* |*,.. .*).

# Modular verification

In order to analyse the behaviour of a biological system we would like to verify properties of computation of sync-program *PI* representing the system. Say that a property *φJ* only regards part of the system, in particular the part involving only sync-automata from *J* ⊆ *I*. We would like to check satisfaction of *φJ* on semantics of *PI* . In order to avoid space explosion, we want to check it on smaller and more abstract semantics, in particular semantics of *PJ* = *PI* T*J* . The subprogram *PJ*

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 *PI* 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 *PI* is present as a computation of *PJ* .

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 *PI* projected onto *J* is a fullpath in the semantics of *PJ* . Firstly, we prove that every path in M*I* projected onto *J* is a path in M*J* .

Lemma 3.1 (Transition projection) *Let I be an interaction graph and* M*I* = (SI*,* S*I,* R*I* ) *the semantics of sync-program PI. For all I-states s, t in* S*I and all l* ∈ P(|*I*|)*, transition* (*s, l, t*) *is in* R*I iff for all J* ⊆ *I such that l* ∩ |*J* | /= ∅*,* (*s, l, t*)[*J is in* R*J , where* M*J* = (SJ *,* S*J ,* R*J* ) *is the semantics of sync-program PJ* = *PI* T*J.*

0

0

Proof. Direction right to left. Suppose that for any *J* ⊆ *I* such that *l* ∩ |*J* | /= ∅, (*s, l, t*)[*J* ∈ R*J* . By taking *J* = *I* we get (*s, l, t*) ∈ R*I*.

Direction left to right. Suppose that (*s, l, t*) ∈ R*I* , we will show (*s, l, t*)[*J* ∈ R*J*

for any *J* ⊆ *I* such that *l* ∩ |*J* | /= ∅. Let *i* be any element from *l* ∩ |*J* |.

* Since *i* ∈ *l*, according to the definition of semantics of *PI* move *s*[*i* Λ*j*∈*LAj* :*Bj*

−−−−−−−→

*t*[*i* ∈ *PI* . Then since *PJ* is subprogram of *PI* , *s*[*i* Λ*j*∈*L*∩|*J* |*Aj* :*Bj*

*t*[*i* ∈ *PJ* by

*i*

definition of subprogram.

−−−−−−−−−→ *i*

* For all *j* ∈ *L*, for all *p* ∈ *Aj*:*s*[*j*(*p*) = *tt* and for all *p* ∈ *Bj*:*t*[*j*(*p*) = *tt* and this implies satisfaction of the condition for all *j* ∈ *L* ∩ |*J* |.
* *L* ∩ |*J* | ⊆ *l* ∩ |*J* | because *L* ⊆ *l*.
* Also since for all *i* ∈ |*I*|−*l* : *s*[*i* = *t*[*i*, it holds for subset |*J* |∩(|*I*|−*l*)= |*J* |−*l*∩|*J* |.
* Minimality of *l* ∩ |*J* | comes from minimality of *l*, namely if *l* ∩ |*J* | is not minimal then neither *l* is minimal.

Thus, by definition of semantics of *PJ* , (*s*[*J, L* ∩ |*J* |*, t*[*J* )= (*s, l, t*)[*J* ∈ R*J* . 2

Lemma 3.2 (Path projection) *Let I be an interaction graph and* M*I semantics of sync-program PI. For every J* ⊆ *I if π is a path in* M*I then π*[*J is a path in* M*J , where* M*J is the semantics of sync-program PJ* = *PI* T*J.*

Proof. Let *π* = (*Bl*1*, l*1*, Bl*2*, l*2*,.. .*) be a path in (*M* )*I* and *Blm J* -blocks for all

*m*. By *sm* and *tm* denote first and last state of *Blm*, respectively. By definition of *I*-structure we have that transition (*tm, lm, sm*+1) is in M*I* for all *m*. By transition projection lemma transition (*tm, lm, sm*+1)[*J* = (*tm*[*J, lm* ∩ |*J* |*, sm*+1[*J* ) is in M*J* for all *m*. Now since *sm*[*J* = *tm*[*J* for all *m*, we get that (*sm*[*J, lm* ∩ |*J* |*, sm*+1[*J* ) in M*J* for all *m*. Hence sequence (*s*1[*J, l*1 ∩ |*J* |*, s*2[*J, l*2 ∩ |*J* |*,.. .*) satisfies the definition of a path in M*J* . 2

However, with computation defined as a fullpath, violation of the desired compu- tation preservation might occur. In particular, violation arises when an independent partition *P* of sync-program *PI* exists that can be executed forever, while not al- lowing execution of other enabled sync-automata outside *P* . Consider a fullpath *π* in 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 *π*[*J* is obtained. However, as in *t* some automata from *J* are enabled but not exe- cuted, in M*J* they can be executed and thus a path ending in *t*[*J* is not a fullpath. Hence, *π* is a computation of *PI* but *π*[*J* is not computation of *PJ* .

We need to refine the definition of computation, so that for all *J* a computation of *PI* projected onto *J* will be a computation in *PJ* . 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 M*I* .

Definition 3.3 A path *π* = (*s*1*, l*1*, s*2*, l*2*,.. .*) in M*I* is *fair* iff for all *i* ∈ |*I*| we have that {*m* | *i* ∈ *lm*} 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](#_bookmark4).

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 compu- tation under projection.

Lemma 3.4 (Fullpath projection) *Let J* ⊆ *I be an interaction graph. If π is a fair fullpath in* M*I, then π*[*J is a fair fullpath in* M*J .*

Proof. By path projection lemma *π*[*J* it is a path in M*J* . Since *π* is a fair path in M*I* by definition of path projection we get that *π*[*J* is a fair path in M*J* . From the definition of fairness follows that every fair path is infinite, i.e. it is a fullpath.2

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 ¬*p* are formulae for any atomic proposition *p*.
* If *f, g* are formulae, then so are *f* ∧ *g* and *f* ∨ *g*.
* If *f, g* are formulae, then so are *AXjf* , *A*[*fUg*] and *A*[*f Uwg*].

We define the logic ACTL*J* to be ACTL where the atomic propositions are drawn from *APJ* = {*APi* | *i* ∈ |*J* |}. Abbreviations in ACTL: *AF f* ≡ *A*[*true Uf* ] and *AGf* ≡ *A*[*f Uwfalse*].

Properties expressible by ACTL formulae represent a significant class of prop- erties investigated in systems biology literature as identified in [[15](#_bookmark25)], such as prop- erties 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](#_bookmark14)] is describable by an ACTL formula as well.

Occurrence, possible consequence, sequence and possible persistence are of in- herently 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 M*I,s* ▶ *f* (resp. M*I,π* ▶ *f* )

meaning that *f* is true in structure M*I* at state *s* (resp fair fullpath *π*). We define

▶ inductively:

* M*I, s*1 ▶ *true*. M*I, s*1 /▶ *false*. M*I, s*1 ▶ *p* iff *s*1(*p*)= *tt*.

M*I, s*1 ▶ ¬*p* iff *s*1(*p*)= *ff* .

* M*I, s*1 ▶ *f* ∧ *g* iff M*I, s*1 ▶ *f* and M*I, s*1 ▶ *g*.

M*I, s*1 ▶ *f* ∨ *g* iff M*I, s*1 ▶ *f* or M*I, s*1 ▶ *g*.

* M*I, s*1 ▶ *Af* iff for every fair fullpath *π* = (*s*1*, l*1*,.. .*) in M*I* : M*I,π* ▶ *f* .
* M*I,π* ▶ *f* iff M*I, s*1 ▶ *f* .
* M*I,π* ▶ *f* ∧ *g* iff M*I,π* ▶ *f* and M*I,π* ▶ *g*.

M*I,π* ▶ *f* ∨ *g* iff M*I,π* ▶ *f* or M*I,π* ▶ *g*.

* M*I,π* ▶ *Xlf* iff if (*s*1*, l*j*, s*2) ∈ R*I* and *l*j ⊇ *l* then M*I, π*2 ▶ *f* .
* M*I,π* ▶ *fUg* iff there exists *m* ∈ N such that M*I, πm* ▶ *g*

and for all *m*j *< m* : M *, πm*′ ▶ *f* .

*I*

* M *,π* ▶ *fU g* iff for all *m* ∈ N*,* if M *, πm*′ /▶ *g*

*I w* *I*

for all *m*j *< m* then M*I, πm* ▶ *f* .

Now we give the main theorem of the paper.

Theorem 3.7 (Property preservation) *Let J* ⊆ *I be an interaction graph, s an*

1. *state and f an ACTLJ property. If* M*J , s*[*J* ▶Φ *f then* M*I ,s* ▶Φ *f.*

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

*f* = *p*. By definition of state projection and the fact that *APi*s are pairwise disjoint, for all atomic propositions *p* from *APJ* we get that M*J , s*[*J* ▶Φ *p* iff M*I,s* ▶Φ *p*. Analogously for *f* = ¬*p*.

*f* = *g* ∧ *h*. From the assumption M*J , s*[*J* ▶Φ *g* ∧ *h* by CTL semantics, M*J , s*[*J* ▶Φ *g* and M*J , s*[*J* ▶Φ *h*. By induction hypothesis M*I,s* ▶Φ *g* and M*I,s* ▶Φ *h*. Hence, M*I,s* ▶Φ *g* ∧ *h*. Case *f* = *g* ∨ *h* is proved analogously.

*f* = *AXg*. Let *π* = (*s, l*1*, s*2*, l*2*,.. .*) be an arbitrary fair fullpath starting in *s*. If *l*1 ∩ |*J* | ≥ *l* then since |*J* | ⊇ *l* also *l*1 ≥ *l* and by CTL semantics M*I,π* ▶Φ *Xlg* vac- uously. If *l*1 ∩ |*J* | ⊇ *l* then by definition of path and by transition projection lemma we have that (*s*1[*J, l*1 ∩ |*J* |*, s*2[*J* ) ∈ R*J* . Now by CTL semantics M*J , π*2[*J* ▶Φ *g* and by induction hypothesis M*I, π*2 ▶Φ *g*. Hence M*I,π* ▶Φ *Xlg*. Since *π* was an arbitrary fair fullpath starting in *s*, we proved M*I,s* ▶Φ *AXlg*.

*f* = *A*[*gUwh*]. Let *π* be an arbitrary fair fullpath starting in *s*. We establish M*I,π* ▶Φ [*gUwh*]. By fullpath projection lemma *π*[*J* is a fair fullpath in M*J* , hence by the assumption M*J , π*[*J* ▶Φ [*gUwh*]. There are two cases:

* 1. M*J , π*[*J* ▶Φ *Gg*. Let *t* be any state along *π*. By CTL semantics M*J , t*[*J* ▶Φ *g*. by induction hypothesis we have M*I,t* ▶Φ *g*. Since *t* was an arbitrary state of *π*, we get M*I,π* ▶Φ *Gg* and thus M*I,π* ▶Φ *gUwh*.

′′

* 1. M*J , π*[*J* ▶Φ [*gUh*]. Let *sm* be the first state along *π*[*J* that satisfies *h*. Then

*J*

there is at least one state *sm*′′ along *π* such that *sm*′′ [*J* = *sm*′′ . Let *sm*′ be first

*J*

such state. By induction hypothesis M *, sm*′ ▶

*I*

Φ

*h*. From the definition of path

projection any *sm* with *m < m*j projects to *sm*[*J* that is before *sm*′ in *π*[*J* . By the assumption M*J , sm*[*J* ▶Φ *g*, hence by induction hypothesis M*I, sm* ▶Φ *g*. By CTL semantics we get M*I,π* ▶Φ *gUh*.

*J*

In both cases we showed M*I,π* ▶Φ *gUwh*. Since *π* was arbitrary fair fullpath starting in *s*, we conclude M*I,s* ▶Φ *A*[*gUwh*].

*f* = *A*[*gUh*]. Let *π* be an arbitrary fair fullpath starting in *s*. By fullpath projection lemma *π*[*J* is a fair fullpath in M*J* and by the assumption M*J , π*[*J* ▶Φ [*gUh*]. By the above case we get *s* ▶Φ *A*[*gUh*]. 2

# Application

We demonstrate our approach on modelling and verification of the *lac* operon reg- ulation taken from [[16](#_bookmark26)].

* 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](#_bookmark24)]. The *lac* operon contains three genes related to lactose metabolism. The *lac*

Z, Y and A genes encode *β*-galactosidase, galactoside permease and thiogalacto- side 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 lev- els, 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).

* 1. *The model*

We will fix the set of indices *Id* = {*lac, β, allo, op, rep, pos, glu*} representing lac- tose, *β*-galactosidase, allolactose, *lac* operon, repressor, CRP-cAMP regulation and glucose, respectively. The interaction graph *I* is on fig. [1](#_bookmark2). For the sake of sim- plicity 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 *PI* with *APlac* = {*Lac none, Laclow*}, *β*-galactosidase by

*lac*

*I* with *APglu* = {*Beta low, Beta high*} and allolactose by *PI*

*P*

*allo*

*β*

with *APallo* =

{*Allo none, Allo low*}. The *lac* operon is represented by *PI* with *APop* =

*op*

{*Act, Rep*}, *lac* repressor by *PI* with *APrep* = {*B*1*, B*2*, B*3*, Ballo*}, the positive

*rep*

regulation by *PI* with *APpos* = {*cAM P high, CRP* −*cAM P* } and finally glucose

*pos*

*I*

by *P*

*glu*

with *APglu* = {*Glu high, Glu low*}.

Sync-automaton *PI*

*lac*

(fig. [2](#_bookmark3) in Section [2](#_bookmark1)) has two states, mappings of the set of

atomic propositions *APlac* 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 *NOSY NC* 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 *PI* and *PI* . If this enzyme is absent, lactose is transformed

*glu*

*β*

to allolactose instead.

In sync-automaton *PI*

*β*

(fig. [3](#_bookmark5)) *β*-galactosidase has two states representing its

concentration level which are affected by activation and repression of *lac* operon *PI* . 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.

*op*

*Lac low* Ç ∧*true*:*Allo low Lac low* Ç ∧*Glu low* Ç



*true*:*Act* ∧ ¬*Rep*

*Beta low Beta high*

*Act* ∧ ¬*Rep*:*true*

*Lac low* Ç ∧*true*:*Glu high*

Fig. 3. *PI* – *β*-galactosidase

*β*

Allolactose *PI*

*allo*

(fig. [4](#_bookmark6)) 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 *PI*

*rep*

and its concentration will reduce.

*Lac low* Ç ∧*Beta low* Ç



*Lac low* Ç ∧*Beta low* Ç

*Allo none Allo low*

¬*BAllo*:*BAllo*

Fig. 4. *PI*

*allo*

– Allolactose

The *lac* operon *PI*

*op*

(fig. [5](#_bookmark7)) has four states, all possible truth value assignments

to *APop*. Atomic proposition *Act, Rep* represent that the *lac* operon activatated and repressed, respectively. Repression and unrepression (horizontal moves in fig. [5](#_bookmark7)) are

determined by negative regulation *PI*

*rep*

while activation and inactivation (vertical

moves in fig. [5](#_bookmark7)) by positive regulation *PI*

*pos*

. Note that full transcription of the

operon genes occurs only when both unrepressed and activated (state *Act,* ¬*Rep*). This state also determines the concentration of *β*-galactosidase.

¬*B*1:*true*

¬*Act,* ¬*Rep* ¬*Act, Rep*

*true*:¬*B*1

*Beta high*:*Beta low* ∧

*CRP* −*cAM P* :¬*CRP* −*cAM P*

*Beta low*:*Beta high* ∧

¬*CRP* −*cAM P* :*CRP* −*cAM P*

¬*CRP* −*cAM P* :*CRP* −*cAM P CRP* −*cAM P* :¬*CRP* −*cAM P*

*Act,* ¬*Rep*

¬*B*1:*true* ∧ *Beta high*:*Beta low*

*true*:¬*B*1 ∧ *Beta low*:*Beta high*

*Act, Rep*

Fig. 5. *PI*

*op*

– Lac operon

The *lac* repressor *PI*

*rep*

(fig. [6](#_bookmark8)) 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.

*Allo low*:*Allo none*

*B*1*,B*2 *B*1*,B*3

¬*Rep*:*Rep*



¬*Rep*:*Rep*

*B*1*,B*3*, Ballo*

*Allo low*:*Allo none*

*Rep*:¬*Rep* ∅

*B*1*,B*2*, Ballo*

*Rep*:¬*Rep*

Fig. 6. *PI*

*rep*

– Lac repressor protein (Negative regulation)

The positive regulation *PI*

*pos*

(fig. [7](#_bookmark9)) 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.

*I*

In *P*

*glu*

(fig. [8](#_bookmark10)) glucose concentration can be high or low. The decrease of its

concentration depends on factors that are not modelled. Increase of concentration

*Glu low* Ç



*Glu high* Ç

*Glu low* Ç

¬*Act*:*Act*

∅ *Glu high* Ç ∧*Act*:¬*Act*

*cAM P high*

*cAM P high, CRP* −*cAM P*

Fig. 7. *PI*

*pos*

– CRP-cAMP (Positive regulation)

*Lac low* Ç ∧*Beta high* Ç



*NOSY NC*

*Lac low* Ç ∧*Beta high* Ç

*Glu high Glu low*

*Lac low* Ç ∧*Beta high* Ç

*truepos*:*truepos truepos*:*truepos*

Fig. 8. *PI*

*glu*

– Glucose

can occur via reaction of lactose and *β*-galactosidase. It is nondeterministically decided when the concentration level of glucose is considered high enough to pass to state high. In addition, this component can be queried for the concentration level

*pos*.

by *P*

*I*

The sync-program describing the whole system is obtained by running all sync-

automata in parallel: *PI* = (*SI,PI* ||*P I* ||*P I*

||*P I* ||*P I*

||*P I*

||*P I*

). Set of initial

0 *lac β*

*allo*

*op rep*

*pos*

*glu*

states is a combination of sets of initial states of respective sync-automata.

* 1. *Experiments*

We check some known properties of *lac* operon regulation. The check can be per- formed by using any CTL model checking algorithm on the semantics of indicated subprograms. Satisfaction of a property on M*J* for *J* ⊆ *I* guarantees its satisfaction in M*I* .

The property *“The allolactose bound to the repressor implies that the operon is repressed”* represents the exclusion type as identified in [[15](#_bookmark25)]. The formula *AG*(*Ballo* → *Rep*) is verifiable on the semantics of *Prep,op*. A slightly more com- plicated formula is needed to express that *“The increase of allolactose concentra- tion can only be mediated by β-galactosidase in low concentration”*. The formula *AG*(*Allo none* ∧ *Beta high* → *A*(¬*Allo lowUBeta low*)) is true in the semantics of *Pallo,β* .

The oscillation property *“The operon will necessarily oscillate between repressed and unrepressed state”* expressed by *AG*((*rep* → *AF* ¬*rep*) ∧ (¬*rep* → *AF rep*)) is true in M*I* , but the verification in 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 *ACT Lglu,op* formula *AG*(*Glu low* → *AF* (*Act* ∧ ¬*Rep*)), it holds in M*I* but cannot be verified for any of the subprograms of *PI* , 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* −*cAM P complex is bound to the operon, the* lac *operon will eventually be fully expressed”*. Formula *AG*(*CRP cAM P* ∧ *Glu low* → *AF* (*Act* ∧ ¬*Rep*)) can be verified on the semantics of *Ppos,glu,op,β*.

# 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](#_bookmark21)]. For application in systems biology see [[16](#_bookmark26)].

A subprogram can be looked at as an abstraction of the program. In [[9](#_bookmark21)] property preservation is investigated in the framework of abstract interpretation. Other approaches consider reducing transition systems so that a particular property is preserved [[2](#_bookmark12)]. In [[6](#_bookmark16)] 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](#_bookmark22)]. We plan to improve our approach with a weaker notion of fairness, in the line of [[1](#_bookmark11)], 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).

1. 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.
2. 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.
3. 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.
4. Cardelli, L., *Brane calculi*, Computational Methods in Systems Biology (2005), pp. 257–278.
5. 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).
6. 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.
7. 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.
8. Dams, D., R. Gerth and O. Grumberg, *Abstract interpretation of reactive systems*, ACM Trans. Program. Lang. Syst. 19 (1997), pp. 253–291.
9. Danos, V. and C. Laneve, *Formal molecular biology*, Theor. Comput. Sci. 325 (2004), pp. 69–110.
10. 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.
11. Grumberg, O. and D. E. Long, *Model checking and modular verification*, ACM Trans. Program. Lang. Syst. 16 (1994), pp. 843–871.
12. 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.
13. Jacob, F. and J. Monod, *Genetic regulatory mechanisms in the synthesis of proteins.*, J Mol Biol 3

(1961), pp. 318–356.

1. 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.
2. 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.
3. 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.
4. 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.