Formal proof of theorems on genetic regulatory networks

Maxime Dénès
and Benjamin Lesage
Université de Rennes

Yves Bertot
INRIA Sophia Antipolis Méditerranée

Adrien Richard
CNRS I3S (UMR 6070), Sophia Antipolis

***Abstract*—We describe the formal verification of two theorems of theoretical biology. These theorems concern genetic regulatory networks: they gives, in a discrete modeling framework, relations between the topology and the dynamics of these biological networks. In the considered discrete modeling framework, the dynamics is described by a transition graph, where vertices are vectors indicating the expression level of each gene, and where edges represent the evolution of these expression levels. The topology is also described by a graph, called interaction graph, where vertices are genes and where edges correspond to influences between genes. The two results we formalize show that circuits of some kind must be present in the interaction graph if some behaviors are possible in the transition graph. This work was performed using the `ssreflect` extension of the Coq system.**

I. Introduction

Biologists often try to predict the dynamic behavior of complex biological systems composed of several genes by studying interactions between some of these genes. For instance, a gene may be known to activate or inhibit the expression of another one. An abstract interaction graph, with positive and negative edges, is then used as a tool to model activatory and inhibitory effects between genes.

Following this approach, Thomas studied about 20 years ago the relationship between the interaction graph and the dynamics of genetic regulatory systems. He stated that feedback circuits played a major role in the possible evolutions of a system [13]. More precisely, Thomas conjectured that (1) positive circuits are necessary for multistationarity and (2) negative circuits are necessary for sustained oscillations (a circuit is positive if it contains an even number of negative edges, and it is negative otherwise). Richard and Comet recently proved these conjectures in the discrete modeling framework developed by Thomas [10], [9]. This is of particular interest since multistationarity and sustained oscillations are involved in important biological phenomena: the differentiation and *homeostasis* processes respectively [13].

According to Thomas, genetic regulatory networks are amenable to a formal study using symbolic computation. Two kinds of objects need to be considered. The first kind consists in transition graphs. These are finite state automata, where the states describe the expression level of each gene, and transitions from one state to another are allowed only when a single expression level changes. The second kind of objects consists in interaction graphs. These are directed graphs, where the nodes correspond to genes, and where the edges are labeled with a positive or negative sign. A positive sign on an edge means that the source gene activates the target gene. A negative sign means that the source gene inhibits the target gene. Transition graphs and interaction graphs provide two points of view on the same biological system. There can be an arrow between two genes in the interaction graph only if some particular patterns occur in the transition graph.

The conjectures of Thomas proved by Richard and Comet express relations between local properties visible in the interaction graph and global dynamical properties visible in the state transition graph. They are results in discrete mathematics, where the biological concepts can be abstracted away. These results are well suited for a study in a theorem prover. We performed such a study using the Coq system [4], [1], using the `ssreflect` extension [6].

Our work aimed first at developing, within the theorem prover Coq, a formal definition of the notions of interaction and transition graphs, and then formalizing the proofs of Thomas' conjectures given by Richard and Comet [10], [9].

In section 2, we discuss the various choices that are made when modeling genetic regulatory networks. In section 3, we show how the various concepts are encoded in our formal development. In section 4, we concentrate on the structure of proofs for the two results that we have studied. Section 5 draws the main lessons from this experiment and discusses orientations for future work.

This paper describes a formal development, so that the specific syntax of the system we used (the Coq system) will surface at many places in the article. Notations of the form `forall x : T, P x -> Q x` should be read like the mathematical expression $\forall x \in T, P(x) \Rightarrow Q(x)$. Thus, we use full names for mathematical quantifiers, juxtaposition but not necessarily parentheses to denote function or predicate application (thus following the tradition of functional programming languages like ML or Haskell), and the arrow `->` will be used to represent both function types and implications, depending on the context.

II. Modeling genetic networks

At the genetic level, the state of a biological cell is often represented by the tuple describing the level of expression of each gene. For levels, we have the choice between three approaches: in the boolean approach, a gene can only express itself fully or not at all. In the discrete approach, levels are discrete and ordered, like integers. In the continuous approach, levels are given by real numbers and the tools to study their evolution are the tools of real analysis. However, in the continuous case, the complexity of the model is increased in a significant way. Theoretical studies of continuous models can be found in [7], [11], [12]. The boolean case can actually be viewed as a special case of the discrete case, where each gene only has two levels. Thomas showed that, contrary to the discrete case, the boolean case if often too caricatural to give realistic descriptions of the qualitative behavior of genetic regulatory networks [13].

In our work, we concentrate on the discrete model. Thus, the type used for states is a Cartesian product of values taken in finite ranges. Since the number of genes is an abstract variable of our study but is supposed to be the same for all states , it is sensible to use a vector type for the states. A vector type is akin to a type of lists with fixed length.

Once we have decided the type of states, we have to choose a representation for the dynamics of the system. One approach is to provide a function that maps every state x to a target state in the direction of which the system evolves when it is in state x. Another approach is to associate to every state x and every gene i a number between -1 and 1 that indicates if the level of i at state x is decreasing (-1) stable (0) or increasing (1). In other words, one associates to each state a vector of evolutions (whose components are -1, 0 or 1). The theorems we formalized actually use the two approaches.

There is a choice between allowing several genes to evolve at the same time or not. The first choice, known as the synchronous update, makes it possible to consider transitions where several genes evolve. The second choice, known as the asynchronous update, imposes that only one gene evolves in every transition. The first choice may seem more expressive, but Thomas [13] argued that this approach could be avoided for biological reasons: cells are naturally asynchronous.

Asynchronous transition graphs of can exhibit various properties that are of interest for biologists. For instance they can contain non-trivial strongly connected components, which are important basic blocks in biological oscillations. For another example, transition graphs may contain several subsets of states with no possible evolution from one subset to the other, and this appears in cell differentiation. The word used for this property is multistationarity.

Aside from the transition graph which we just described, we also consider an interaction graph, where the vertices are genes and such that there exists an edge from one gene to another if the evolution of the latter depends on the evolution of the former. The edge is then labelled with a positive (negative) sign if an increase of the source gene induces an increase (decrease) of the target gene. The interaction graph is a summary of the whole transition graph: it is much smaller since the number of nodes is only the number of genes.

The work we describe in this paper aims at finding necessary properties in the interaction network for the presence of interesting properties in the transition graph. The two results we studied are as follows:

1) If the transition graph describes a multi-stationarity, then the interaction graph contains a positive circuit [10].
2) If the transition graph describes sustained oscillations, then the interaction graph contains a negative circuit [9].

The first part of the work is to describe how an interaction graph is computed from a transition graph. Then the problem of each theorem is solved by reducing the size of the input interaction network, for instance by removing genes one by one.

III. Encoding the basic structures and properties

From the point of view of formal proof, one main difficulty is to handle the finite ranges for the levels of each gene and for vectors of gene levels. In theory, states should be members of a Cartesian product of n intervals, where the size of each interval is given by the number of levels that are available for the corresponding gene. Since the number n of genes is a parameter of our study, this would require defining the intervals as a family indexed by the integers of the interval $[0, n-1]$ (denoted `'I_(n)` in the `ssreflect` jargon) and then describing the data type as the cartesian product of this family, in mathematical notation

$$\Pi_{i\in[0,n-1]}[0,l_i].$$

For a specialist in type theory, the theory that underlies the Coq system, this looks like an ideal application case for dependent types, however the resulting data type is unwieldy because its members are functions and the type theory usually makes it difficult to compare functions for equality and to express that this type is actually finite.

We took an alternative solution, that relies slightly less on dependent types. We actually work with the type:

$$\Pi_{i\in[0,n-1]}[0,N_{max}]$$

where N_{max} is the maximum of $\{l_i | i \in [0,n-1]\}$, and then we express explicitely that we only look at the states that respect the strong obligation that the component at rank i is smaller than the value l_i. From the encoding point of view, we first assume that we are given the number n of genes and a function `max_levels` that associates to each gene the number of levels for this gene (l_i in the informal mathematical notation above). We define `N_max` to be the maximum of this function, plus one. Next, we define a predicate on any value `x` in the interval [1, `N_max`] to express when it is smaller than the max level for a given index `i`. This predicate is called `level_interval`. We then rely on `ssreflect` notions for functions with finite domain (notation `ffun`) to describe states as finite functions from the interval [1, n] to the interval [1, `N_max`] with the additional constraint that the value for each i is bound to be less than `max_levels` i. The notation `Record stateType ...` in the following lines indicates that an object of type `stateType` actually contains two parts: the first part, named `state_val` is function, the second part does not receive a name, but it is a proof that `state_val` satifies the predicate `family level_interval`. In a nutshell, `family p f` is satisfied if for every `i`, the value of the function `f` in `i` is smaller than `p i`. The function `State` can be used to construct an object of type `stateType` from a given function and the proof of the corresponding constraint.

```
Variable n : nat.
Variable max_levels : 'I_(n) -> nat.

Definition N_max :=
  (\max_(i : 'I_(n)) max_levels i).+1.

Definition level_interval (i: 'I_(n))
                   : pred 'I_(N_max) :=
        fun x => 1 <= x <= max_levels i.

Record stateType : Type :=
  State
   {state_val :>
           {ffun 'I_(n) -> 'I_(N_max)} ;
    _ : family level_interval state_val}.
```

We used a datatype representing the mathematical field $\mathbb{Z}/3\mathbb{Z}$ to represent evolutions of genes. A decrease is represented by the value -1=2, while an increase is represented by the value 1 and stability is represented by the value 0. given two states x and y and a gene number i, we compute the evolution of a gene number i needed to go from x to y using a function `evo` which is defined in the following manner, where `one_evo`, `mone_evo`, and `zero_evo` represent respectively the values `1`, `-1`, and `0`.

```
Definition evo :=
  if x i < y i then one_evo
  else if y i < x i then mone_evo
  else zero_evo.
```

Similarly, the sign of edges in the interaction graph is described by a number in the data type representing $\mathbb{Z}/2\mathbb{Z}$.

1) Paths in the interaction graph: To describe paths in the interaction graph, we use the pre-defined notion of paths from the `ssreflect` library, where a boolean predicate is used to characterize the non-empty lists where two successive elements have to satisfy the edge predicate, given as a plain binary relation. However our edges are signed and they don't fit directly this approach. To circumvent this difficulty, we consider paths where the elements are edges instead of nodes and we use an adjacency relation between edges instead of the edge predicate. Edges from the interaction graph are represented by triplets with two genes (represented by numbers in `'I_(n)`) and a sign; the set of all edges in a given interaction graph is represented by a boolean predicate.

```
Variable interact_edge :
     pred ('I_(n) * sign * 'I_(n)).
```

The adjacency relation between two edges is satisfied exactly when the edges are in the graph (i.e., they satisfy the boolean predicate) and the target of the first edge is the origin of the second one. Interaction paths are paths for this adjacency relation. The adjacency between two edges is represented by the boolean predicate `interact_rel`. In the function's code, the notation `[&& .. & ..]` stands for the conjunction of several formulas.

```
Definition interact_rel
   (u v :('I_(n)*sign*'I_(n))) :=
   [&& (interact_edge u) ,
       (interact_edge v) &
       let: (i1,s1,j1) := u in
       let: (i2,s2,j2) := v in
         (j1 == i2)].
```

The function `interact_path` describes the paths in the interaction graph. It is a predicate on non-empty sequences of edges, represented by a first edge `u` and a possibly empty sequence `s`.

```
Definition interact_path u s :=
  if s is [::] then
    interact_edge u
  else
    path interact_rel u s.
```

The type of `s` is `seq`, a specific type of lists, used when the elements are in a type where equality is decidable, and the notation `[::]` is for the empty sequence of type `seq`.

An other predicate `interact_circuit` is used for paths where the origin of the first edge and the target of the last edge coincide.

For the two theorems that we studied, the variable `interact_edge` is actually instantiated with two different relations, corresponding to different ways to compute interaction edges from the transition graph.

The sign of paths (defined to be the product of the signs of its edges) is easily computed recursively from the sequences of edges and we have a collection of lemmas describing the interplay of signs and other operations on paths.

2) Describing the dynamics: One of the approaches to describe the dynamics of a biological system is to provide a target function f that maps every state to another state denoting the direction in which the system evolves.

The set of all states reachable in one transition is computed by checking the genes that have different levels between the state x and the state $f(x)$, and allowing these genes to evolve, one at a time, in the direction given by f.

The auxiliary function `evo` compares two states x and y with respect to a coordinate i and returns -1, 0, or 1 depending on whether the i coordinate decreases, stays the same, or increases when moving from x to y.

The state `apply_evo` x y j is the neighbor of state x, in the same direction as y, with only the j coordinate changed. In our development, it takes a little reasoning to make sure that the state we obtain is licit, in the sense that the level for the coordinate j is still smaller than `max_levels` j. The reason is that if y was a licit state, and the level in y for the coordinate j is larger than the level in x, then the level $x(j)+1$ is larger than the level $x(j)$ and smaller than the level $y(j)$ and thus it is still in the correct interval. The proof goes similarly when $y(j)$ is smaller than $x(j)$. Because constructing the new state requires putting together the actual computation of the new state and the construction of the proof that this state is licit, the code for the function `apply_evo` is too long to fit in this article.

The expression `apply_dir` $f\ x\ i$ represents the state to which the cell will evolve when starting from the state x and following the evolution prescribed by f for the gene i. This function is actually described with the help of `apply_evo`.

```
Definition apply_dir f x i :=
  apply_evo x (f x) i.
```

With this function, we can define a binary relation on states, which is satisfied as soon as there can be a transition from one state to another. This relation, called `trans_edge` is written as follows:

```
Definition trans_edge f x y :=
  existsb i, (i \in unstable_d f x) &&
  (y == apply_dir f x i)
```

In this definition, `unstable_d` describes the set of all genes for which the target function prescribes a change of level.

3) Interactions between genes: To describe how a gene can influence another can be done locally at the level of each couple of states x and y. One observes the difference between the evolution (-1, 0 or 1) that the gene i follows on the current state x and on the state that one obtains from x when gene j evolves of a unit in the direction of y. If the evolution of gene i changes in the same (opposite) sense that the gene j evolved, then j can be viewed as an activator (inhibitor) of i. In this case, we should record a positive (negative) edge from j to i in the interaction graph.

4) Jacobian vs sign function: In the continuous case, the interactions between genes are obtained from the Jacobian of the function that describes the dynamics of the system. In the discrete case, one can use an analogous definition using a discrete Jacobian matrix.

The basic idea of discrete Jacobian matrices is to look at the variation of the target function f when one moves around from a given state. However, the mere variation of f can yield misleading information: f may decrease even though it is attracting a gene's evolution in the same increasing direction. To account for this phenomenon, Richard and Comet propose in [10] to use a non-usual Jacobian matrix, where the coefficients can only be -1, 0, or 1, and where the coefficients are null as soon as the direction of evolution is unchanged.

For instance, we can consider a state x where genes i and j both have level 0 and the target function maps this state to a state z where both genes have level 1; consider a state y where genes i and j respectively have levels 1 and 0. Assume now that the target function maps y to a state where gene j has level 0. In this situation, gene i increases between the initial state x and y whearas the evolution of j is 1 in x and 0 in y. Thus, an increase in the level of i leads to a decrease in the evolution of j. This is recorded in the non-usual Jacobian matrix for states x and y with a -1 for the (j,i)-coefficient.

The computation of the non-usual Jacobian matrix is described in our formal development by the functions `j_threshold`, `jacobian`, and `jacobian'`. These three functions have 5 arguments: the first argument is the target function f, the next two arguments are the initial state x and state y (so that we actually compute a jacobian for each couple of states in the state space), the next two arguments are indices i and j for a location in the Jacobian matrix. When the value of the threshold function is `true` and gene i varies between x and y, we know that the evolution of gene i changes between the state x and the neighbor in the direction of y along the axis of gene j. The non-usual Jacobian matrix between two states is given by the function `jacobian'`.

```
Definition j_threshold
  (f:state -> state) (x y : state) i j :=
 let vi := evo x y i in
 let nbor := (apply_evo x y j in
    ((f x i).*2 <= add_evo ((x i).*2) vi
                           <= (f nbor i).*2)
 || ((f nbor i).*2 <=
     add_evo ((x i).*2) vi <= (f x i).*2).

Definition jacobian
 (f: state -> state) (x y : state) i j :=
  let vj := evo x y j in
     vj *[e]
     (evo (f x)  (f (apply_evo x y j)) i).

Definition jacobian' f x y i j  :=
  if j_threshold f x y i j
  then jacobian f x y i j
  else zero_evo.
```

From the non-usual jacobian matrix for states x and y, we can define a local graph of interaction between genes, where there is an edge between genes i and j with sign s, if i and j are two genes that have different levels in states x and y (this is expressed by the set `unstable`) and if the value of the jacobian matrix at coordinate j, i is s.

```
Definition interact_edge_jacobian f x y :
  pred ('I_(n) * sign * 'I_(n)) :=
  fun u:('I_(n) * sign * 'I_(n)) =>
    let: (i, s, j) := u in
    [&& (i \in unstable x y),
        (j \in unstable x y)
      & (jacobian' f x y j i == s)].
```

Alternatively, the dynamics of the system can directly be described using a direction function which indicates in which direction each gene is supposed to evolve. this direction function can actually be defined from the target function.

```
Definition dir :=
    ffun_of (fun t j => (evo t (f t) j)).
```

While the graph computed through the jacobian was local because it was relative to a given couple of states x and y, we also studied more global interaction graphs concerning several couples of states x and y at a time, but using the restrictive condition that y is the target state of x. In the following definition, we use a set A of states. The notation existsb is used to represent an existential quantification over a finite type. This quantification can then be used as a boolean formula. Note that this definition of edges only captures a subset of all the edges, since it only accepts edges towards j only if the evolution of j in the neighbor chose by apply_dir is non-zero. The fact that we only consider a subset of the edges is not serious because we will only prove the presence of circuits, not the absence: if we prove the presence of a circuit in a subset of the edges, then this circuit still exists in any superset.

```
Definition interact_edge_d
   : pred ('I_(n) * sign * 'I_(n)) :=
 fun u:('I_(n)*sign*'I_(n)) =>
   let: (i,s,j) := u in
    existsb x:state,
      [&& x \in A,
       (dir f x i) *[e]
         (dir f (apply_dir f x i) j) == s
      & dir f x j !=
          (dir f (apply_dir f x i) j)].
```

In this definition, we compute the edges of the interaction network by only looking at states from a given subset A of the whole set of states, and we also only consider transitions where the target state is such that its evolution for the gene j is non-null. As a result, the interaction graph we obtain is much smaller than the one obtain from Jacobian matrice. However, this is not an issue since as mentioned above: if an interaction network G contains less edges than an interaction network G', then every circuit in G is also a circuit in G'.

We needed to consider the two approaches to derive interaction graphs from the target function because the two articles each rely on a different approach. This also shows that the result proved in [9] is actually stronger than announced, since the circuit is found in a sub-graph of the complete interaction graph.

5) Local vs global interaction network: The interaction graphs defined above are local: they depend on the location of the system in the phase space. The easiest way to define a global interaction graph is to build the union of all local networks.

However, the graph described in [9] for the proof of the second Thomas conjecture is more restrictive (see III-4 for a precise definition). The graph we will work on is a sub-graph of the union of local graph. But since we aim at proving the existence of a circuit in this graph, this only strenghten the result.

IV. Formal proof of Thomas' conjectures

A. Conjecture on cyclic attractors and negative circuits

1) Statement of the conjecture: Thomas' conjecture on negative circuits states that the existence of a cyclic attractor in a transition graph implies that a negative circuit can be found in the associated interaction graph.

We need to formalize what is a cyclic attractor. We first decribe the notion of a trap domain as a set of states such that all edges with their origin in this set also have their target in this set.

```
Definition trans_trap_domain
   f (A:{set stateType}) :=
  0 < #|A| /\
  forall (x y:stateType),
    (trans_edge f x y) -> (x \in A) ->
    (y \in A).
```

Then, an attractor is a minimal trap domain w.r.t. inclusion.

```
Definition trans_attractor
  f A => trans_trap_domain f A /\
  (forall B, B \subset A ->
        trans_trap_domain f B -> B = A).
```

A cyclic attractor is simply described as an attractor with at least two elements. With these concepts, we can state the theorem formally.

```
Theorem thomas_negative :
 forall (f : state -> state)
        (A:{set state}),
  trans_cyclic_attractor f A ->
  exists u, exists s,
    interact_circuit
      (interact_edge_d f A) u s /\
    interact_path_sign(u::s) == mone_sign.
```

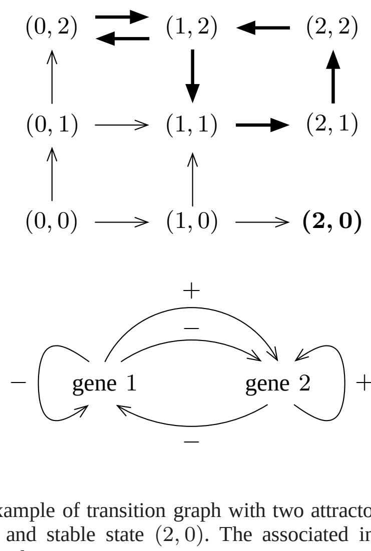

Fig. 1. An example of transition graph with two attractors: a cyclic attractor with 5 states, and stable state $(2,0)$. The associated interaction graph has both positive and negative circuits.

2) Overview of the proof: The body of the proof consists in three main lemmas and the final theorem.

- In Lemma 1, we show that if we consider a path in the transition graph and a gene j that is unstable in the final state of the path then there exists a gene i that is unstable in the original state of the path and such that there exists an interaction path from i to j whose sign is positive if the evolution of i in the original state equals the one of j in the final state, and negative otherwise.
- In Lemma 2, we show that for a cyclic attractor where at least one state has a unique successor there exists a negative circuit in the interaction graph. This lemma relies directly on Lemma 1. It is used as the base case in a proof by induction.
- In Lemma 3, we show that if there is a cyclic attractor where each state has at least two successors, then we can construct a sub-transition graph with a cyclic attractor. The nice property of this operation is that the interaction graph associated to the sub-transition graph is a subgraph of the interaction graph of the original transition graph. this lemma is used as the step case in the induction proof for the main theorem.

3) Example: Lemma 1 states that, given a path in the transition graph (with at least two states), if in the last state of this path a gene j has a different direction of evolution than in any preceding state of the path, there exists a gene i evolving at the first state of the path that interacts, directly or indirectly, with gene j. That is a path can be found from gene i to gene j in the interaction graph. Moreover, its sign is the product of the direction of evolution of gene i (at the first state) with the one of gene j (at the last state).

```
Lemma LemmaOne :
 forall (r:nat) (x0:state)
        (s:seq state) (j:'I_(n)),
  size s = (r.+1)%N ->
  trans_path f x0 s ->
  j \in unstable_d f (last x0 s) ->
  (forall t:state,
     t \in belast x0 s ->
     dir f t j != dir f (last x0 s) j) ->
  exists i:'I_(n),
   i \in (unstable_d f x0) /\
   exists s1:sign, exists s2:sign,
     interact_exists_path
       (interact_edge_d f
           [set t | t \in (x0::s)])
       i j (s1 *[s] s2) /\
     dir f x0 i == s1 /\
     dir f (last x0 s) j == s2.
```

The predicate `interact_exists_path` R i j s expresses the existence, in the interaction graph defined by the relation R, of a path of sign s from gene i to gene j. Here the interaction graph is computed over the states composing the path represented by the list `x0::s`. Note that `ssreflect` provides us with convenient notations to describe `x0::s` as a list, as a path in the interaction graph, or as a set of states (`[set t | t \in (x0::s)]`). The notation `\in` is also versatile: it can be applied on a set, a list, or a boolean predicate. Also s_1 `*[s]` s_2 denotes the product of signs s_1 and s_2.

The proof of all three lemmas takes approximately 500 lines of Coq script.

B. Conjecture on multistationarity and positive loops

Thomas conjecture on positive loops states that the existence of several stable states in the transition graph implies that a positive loop can be found in the associated interaction graph. Richard and Comet actually proved a stronger result considering two disjoint trap domains.

```
Corollary CorollaryOne :
  forall (A B : {set state}) x y,
   trans_trap_domain f A ->
   trans_trap_domain f B ->
   [disjoint A & B] ->
   x \in A -> y \in B ->
  (forall x' y',
     x' \in A -> y' \in B ->
     ((x'== x) && (y' == y)) ||
     ~~((pi_box x' y')
         \subset (pi_box x y))) ->
  exists z,
   z \in (pi_box x y) /\
   exists hd, exists C,
    interact_circuit
     (interact_edge_jacobian f z y)
     hd C /\
   one_sign == interact_path_sign (hd::C).
```

In this statement, the notation `pi_box` x y represents the set of all states whose Manhattan distance to x is smaller than the distance between x and y and similarly for the distance to y. In other words, it is the set of all states whose level for each gene is between the level in x and the level in y. This result is stronger, because it gives information on the location of the state z whose local interaction network must exhibit a positive circuit. This state must be "between" two states from the two trap domains that realize the minimum distance between them.

Note that in Figure 1, the transition graph has two disjointed trap domains, since it has two attractors, and that the corresponding (global) interaction graph has indeed a positive circuit.

In the following, we will say that a state x is stable with respect to a state y if all the genes evolve in the direction opposite to y. The central idea of the proof is to show that if x is stable with respect to y and no positive circuit can be found in any local interaction graph evaluated with a couple of states taken between x and y, then there is a path from y to x. The structure of the proof relies on the following lemmas:

- In Lemma 1, we show that if x is stable with respect to y, then the local interaction graph evaluated at state x in the direction y cannot contain a negative circuit. We then show that if there is no positive circuit and x is stable with respect to y then there is simply no circuit in the local interaction graph evaluated with the couple x and y, and we can find a gene j that has no successor in this graph. This step is just state in a sentence in the original proof. For the formal proof, we needed to exhibit a function that follows edges in the interaction graph until it finds one without a successor. Such a function is defined by well-founded induction.

- In Lemma 2, we show that there exists a state z between x and y, such that z is also stable with respect to y, and such that there exists an edge in the transition graph from z to x. This state z is actually constructed by making the gene j evolve in the direction of y from state x. The evolution of gene j in x, as well as the evolution of all the genes whose level at x and y is different, goes in the opposite direction of y. In the state z, the evolution of the these genes must also go in the opposite direction of y, otherwise there would be an interaction from j to one of these genes in the local interaction graph evaluated at x in the direction of y.
- In Lemma 3, we actually use Lemma 2 as the step case in an inductive proof showing that there is a path from y to x, assuming that all local interaction graphs evaluated with a couple of states taken between x and y satisfy the constraint of not containing any positive circuit.

1) *Example:* Here is the statement of Lemma 2:

```
Lemma LemmaTwo : forall (x y : state),
  x != y -> state_stable x y  ->
  [no + circuit x & y ] ->
  exists z,
    z \in pi_box x y /\
    state_stable z y /\
    trans_edge f z x.
```

Now, if we have two distinct trap domains in the transition graph, then we can consider two states, x and y, one in each domain, that realize the minimum distance between the trap domains. Then x is stable with respect to y (and y is stable with respect to x). So when there is no positive circuit in the local interaction graph evaluated with states between x and y, by using Lemma 3, we obtain that there is a path from y to x and so from one trap domain to the other, this contradicts the assumption.

V. CONCLUSION

The proofs of the two theorems were formalized using the Coq system and the `ssreflect` extension during two internships of approximately two months each, with no prior knowledge of formal development in the Coq system. The `ssreflect` library turned out to be quite usable, even in the training phase. The main advantage of this library comes from the fact that it already contains significant results about finite data structures and graphs, since these aspects already played a role in the formal verification of the proof of the four color theorem by Gonthier [5].

One lesson of this work is that formal developments about theoretical models of biological systems are confronted with a wide variety of choices in the models: whether gene expression is described as integer levels or as boolean levels, whether interaction edges are signed or not. Also, the way the dynamics of the system is described varies from one author to the other. Even the method used to relate edges in the interaction network to edges in the transition graph varies from one paper to the other. An interesting challenge is to develop a formal library that shares as much as possible between the models being used.

The next question that comes to mind is whether this kind of formal study is useful to biologists. A first modest contribution would be that the formal development could be useful to theoretical biologists. The wide variety of choices at the time of modelling a problem of theoretical biology makes that the field is rather heterogeneous: a large number of conjectures have been solved, but each proof is valid in a given model and it is questionable whether a proof obtain in one model (say with boolean levels) still makes sense in another (say with integer levels). Some papers actually relate proofs made at several levels. We plan to address one such proof in a future experiment.

REFERENCES

[1] Y. Bertot P. Castéran. Coq'Art: The Calculus of Inductive Constructions. Springer, 2004.
[2] Non cité dans l'article
[3] Non cité dans l'article
[4] G. Dowek, A. Felty, H. Hebelin, G. Huet, C. Murthy, C. Parent, C. Paulin-Mohring, and B. Werner. The Coq Proof Assistant User's Guide. INRIA, 1993.
[5] G. Gonthier. A computer-checked proof of four color theorem. http://research.microsoft.com/∼gonthier/4colproof.pdf.
[6] G. Gonthier and A. Mahboubi. A small scale reflexion extension for the coq system. https://hal.inria.fr/inria-00258384.
[7] M. Kaufman, C. Soulé, and R. Thomas. A new necessary condition on interaction graphs for multistationarity. Journal of Theoretical Biology Vol.248, 2007.
[8] Non cité dans l'article
[9] A. Richard. On the link between oscillations and negative circuits in discrete genetic regulatory networks. JOBIM, 2007.
[10] A. Richard and J.-P. Comet. Necessary conditions for multistationarity in discrete dynamical systems. Discrete Applied Mathematics, 155(18):2403-2413, 2007.
[11] C. Soulé. Graphic requirements for multistationarity. ComPlexUs, 2003.
[12] C. Soulé. Mathematical approaches to differentiation and gene regulation. C.R. Paris Biologies 329, 2006.
[13] R. Thomas and R. D'Ari. Biological feedback. CRC Press, Boca Raton, Fla, USA, 1990.