An Efficient Method for Dynamic Analysis of Gene Regulatory Networks and *in silico* Gene Perturbation Experiments

Abhishek Garg, Ioannis Xenarios[1], Luis Mendoza[2], and Giovanni DeMicheli

Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland
[1] Merck Serono, Geneva, Switzerland
[2] Instituto de Investigaciones Biomédicas, UNAM, Mexico
`{abhishek.garg,giovanni.demicheli}@epfl.ch`,
`ioannis.xenarios@merckserono.net`, `lmendoza@biomedicas.unam.mx`

Abstract. With the increasing availability of experimental data on gene-gene and protein-protein interactions, modeling of gene regulatory networks has gained a special attention lately. To have a better understanding of these networks it is necessary to capture their dynamical properties, by computing its steady states. Various methods have been proposed to compute steady states but almost all of them suffer from the state space explosion problem with the increasing size of the networks. Hence it becomes difficult to model even moderate sized networks using these techniques. In this paper, we present a new representation of gene regulatory networks, which facilitates the steady state computation of networks as large as 1200 nodes and 5000 edges. We benchmarked and validated our algorithm on the T helper model from [8] and performed *in silico* knock out experiments: showing both a reduction in computation time and correct steady state identification.

1 Introduction

The face of biological research has evolved at an alarming rate over the last two decades. From a one-gene/one-protein analysis it has borne witness to a multitude of technologies that allows us to capture and integrate a vast amount of information generated by high throughput methods such as DNA microarrays, siRNA knock-down and protein-protein interactions. While a wealth of information is present on the interaction of the genes and proteins, the exact stoichiometry and precise kinetics still evades our technologies and understanding. In such situation, one could either wait to gather the crucial information on the precise biochemical processes or choose to model the flow of information in genetic regulatory networks. We chose the latter as we think that the information is sufficient already to identify qualitative behavior of the studied biological system. We also claim that enabling such kind of approaches should further the understanding of the design and identifications of keys elements that dictate cell fate.

The methodology presented here is an improvement of the methods described by [8] to model dicrete regulatory networks and proposes to use a data structure called binary decision diagram (BDD) to represent and manipulate Boolean networks. This data transformation enables the compact representation of the state space of the network and their efficient dynamic analysis. BDDs have primarily been used in several other applications like logic synthesis and testing in the field of Electronic Design Automation [21,22] and model checking [23,24]. In this paper, we show their application on biological regulatory networks. Some work on modelling the gene regulatory networks using the formal methods have been introduced in [27,28,29].

We use the already published *T helper cell* regulatory network [8,7] as a framework to validate our approach and show that our software, **GenYsis** finds all steady states and correctly identifies the outcome of gene perturbation experiments. We show that GenYsis scales well with the size of the network and can compute cell states in the network with size over 1000 nodes in reasonable time using modest computing resources.

2 Binary Decision Diagrams

2.1 Introduction

A Binary Decision Diagram(BDD) is a directed acyclic graph consisting of a root node, intermediate decision nodes and two terminal nodes, namely **0**-terminal and **1**-terminal. BDDs can be used for representing Boolean functions. Each variable of the function is represented as a decison node of the graph. Each decision node has two outgoing edges to represent evaluation of variable to **1** and **0**. All paths from root node to **1**-terminal gives the variable evaluations for which the function is true. There might be some variables missing in some of the paths. These variables have a “Don’t Care” evaluation, i.e. they can take either 0 or 1 value.

A simple BDD that represents the Boolean function $f = (a$ **AND** $b)$ **OR** c is shown in Figure 1.

It has three paths from root node (node f) to **1**-terminal node. For the path, $a \xrightarrow{0} c \xrightarrow{1} 1$, two possible assignments $(a = 0, b = 1, c = 1)$ and $(a = 0, b = 0, c = 1)$ leads to 1-terminal from root node f. Similarly, the path $a \xrightarrow{1} b \xrightarrow{1} 1$,

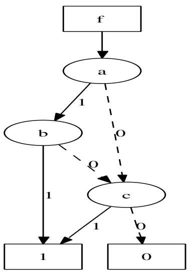


Fig. 1. BDD for the function $f = (a \wedge b) \vee c$

represents the assignments $(a = 1, b = 1, c = 1)$ and $(a = 1, b = 1, c = 0)$. Finally the last path $a \xrightarrow{1} b \xrightarrow{0} c \xrightarrow{1} 1$ representing $(a = 1, b = 0, c = 1)$ complete the five possible TRUE evaluations for the Boolean formula f.

Here, we use Reduced Ordered BDDs (ROBDDs), which are the compact reduced form of BDDs. For the sake of brevity whenever we say BDD in this paper, we are referring to ROBDDs.

The representation of Boolean functions as BDDs is memory efficient as isomorphic subgraphs can be shared by multiple nodes. The size of the BDD scales well in most cases with the size of the Boolean function and all the logic operations like AND, OR, Existential quantification, Universal quantification, etc. can be performed in polynomial (with the size of the BDD) time on this data structure [1]. This implicit representation does not require the explicit construction of a truth table and can be directly constructed from the Boolean function. Further details on BDD construction and logic operations on them is outside the scope of this paper, interested readers can find details in [1,2,3].

There are many existing packages that can be used for working with BDDs like CUDD, CMUBDD, etc. In this paper we use the CUDD package [20], which is the most efficient package for BDD representation and evaluation.

2.2 Representation of Gene Regulatory Networks

In this section we show how gene regulatory networks can be mapped to BDDs. We start with the representation of regulatory networks as Boolean functions and then we use these functions to construct corresponding BDD representation.

Given a gene regulatory network, the state of a node (or gene) i at time t is represented with the Boolean variable $x_i(t)$. To evaluate the evolution in time of each node, it is necessary to describe the state of each node at time $t+1$ as a function of state of those nodes acting as input at time t [8]:

$$x_i(t+1) = \left(\bigvee_{j=1}^{m} x_j^a(t)\right) \wedge \neg \left(\bigvee_{j=1}^{n} x_j^{in}(t)\right) \tag{1}$$

$$x_i(t+1) = \left(\bigvee_{j=1}^{m} x_j^a(t)\right) \tag{2}$$

$$x_i(t+1) = \neg \left(\bigvee_{j=1}^{n} x_j^{in}(t)\right) \tag{3}$$

$$x_j \in \{0,1\}$$
$$x_m^a \textit{ and } x_n^{in} \textit{ are the set of activators and inhibitors of } x_i$$
$$\wedge \textit{ and } \vee \textit{ represent logical AND and OR}$$

Equation 1 is used if the gene i has both activators and inhibitors. Equation 2 is used if the gene i has only activators and equation 3, if there are only inhibitors.

In equation 1, inhibitors are strong enough to change the state of a gene from 1 to 0, while activators can change the state from 0 to 1 if and only if there are no inhibitors acting on that gene.

A snapshot of the activity level of all the genes in the network at a time t is called the state of the network. The state of the network is represented by a Boolean vector of size N (number of genes in the network). Each bit of this vector represents whether the gene is active or inactive. Another Boolean vector of size N is used to represent the status of the genes at next step. We call the previous vector as *present state*(V_t) and latter one as *next state* (V_{t+1}).

The transition between states of the network can be either *synchronous* and *asynchronous*. If the transitions are synchronous, all the genes change their state at the same time point. If the transition is asynchronous, atmost one gene can change its state between two consecutive states. Biologically, it is more realistic to assume that genes have different response times, and hence an asynchronous network might seem more realistic. In this paper, we use the asynchronous model to represent state transition of the regulatory network and assume that time points are close enough, so that only one gene can make a transition at each time point.

Now we shall see how the Boolean functions in equations 1-3 can be used to construct a BDD representation. Let $T_i(V_t, V_{t+1})$ be the BDD representing transition of gene i from V_t to V_{t+1} and $T(V_t, V_{t+1})$ be the BDD representing the transition from state of the network at time t to state at time $t+1$. The relation between $T_i(V_t, V_{t+1})$ and $T(V_t, V_{t+1})$ is given by equation 4. Equation 4 says that all genes make asynchronous transitions and state of the network at time t can have multiple successor states.

$$T(V_t, V_{t+1}) = T_0(V_t, V_{t+1}) \vee T_1(V_t, V_{t+1}) \vee ... \vee\ T_N(V_t, V_{t+1}) \tag{4}$$

To impose the constraint that two consecutive states differ in atmost one gene evaluation, we define $T_i(V_t, V_{t+1})$ in equation 5, which states that for gene i, its evaluation at the next time step $v'_i(\in V_{t+1})$ and function $f_i(V_t)(= x_i(t+1))$ have the same value, and all the other genes remain at their activation level from the previous time step.

$$T_i(V_t, V_{t+1}) = (v'_i \leftrightarrow f_i(V_t)) \wedge \bigwedge_{j \neq i} (v'_j \leftrightarrow v_j) \tag{5}$$

Let us go through a small example to have a better understanding of the process.

Example 1. In figure 2, gene 'A' has an auto-activation and is inhibited by the presence of gene 'C'. Gene 'B' is activated by the presence of gene 'A' and presence of gene 'B' inhibits gene 'C'. The present state and next state vectors are given by $V_t = \{a, b, c\}$ and $V_{t+1} = \{a', b', c'\}$ respectively. Boolean functions describing this small network are given by:

$$a' = a \wedge \neg c \tag{6}$$

$$b' = a \tag{7}$$

$$c' = \neg b \tag{8}$$

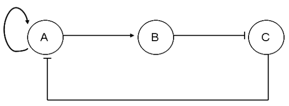


Fig. 2. An example of Gene Regulatory Network

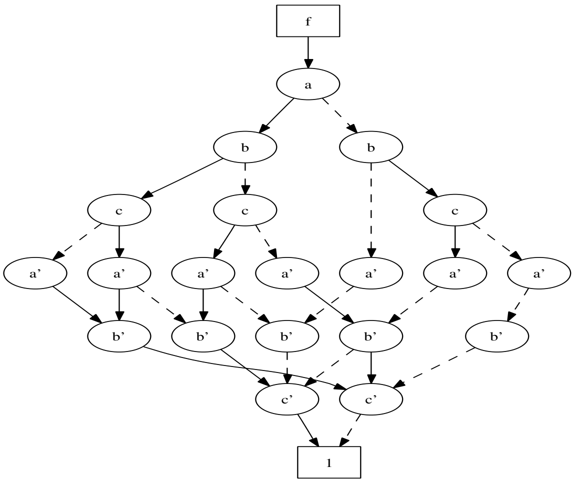


Fig. 3. BDD representing the state space of example in figure 2. The dashed edges represent **0** evaluation of the variables and the solid edges represent the **1** evaluation. For clarity, edges going to **0**-terminal are not shown in this figure.

Corresponding transition relations for each gene is then given by:

$$T_0(V_t, V_{t+1}) = a' \leftrightarrow (a \wedge \neg c) \wedge b' \leftrightarrow (b) \wedge c' \leftrightarrow (c) \tag{9}$$

$$T_1(V_t, V_{t+1}) = b' \leftrightarrow (a) \wedge a' \leftrightarrow (a) \wedge c' \leftrightarrow (c) \tag{10}$$

$$T_2(V_t, V_{t+1}) = c' \leftrightarrow (\neg b) \wedge a' \leftrightarrow (a) \wedge b' \leftrightarrow (b) \tag{11}$$

The BDD representation for $T(V_t, V_{t+1})$ by using equations 9-11 in equation 4 is shown in figure 3. For clarity in figure 3, edges pointing to **0**-terminal are removed. This BDD represents all the possible state transitions of the network. To find the immediate successor states of a given state of the network(say for example $a = 1, b = 0, c = 1$), the following steps can be performed on BDD $T(V_t, V_{t+1})$:

1. Assign initial activity levels to the genes a, b,and c (i.e. $v_i \in V_t$).
2. Remove all outgoing edges which do not satisfy the evaluations in step 1.
3. Find all the paths from root node to **1**-terminal and for each path only print variables in set V_{t+1}.
4. Swap variable names from the set V_{t+1} with the corresponding variable names in the set V_t, on all the printed paths. □

The steps given in above example can be implemented by using efficient logic functions such as "AND" and "Existential Quantify" along with the expressive power of BDDs [1,2], as follows:

1. Construct a BDD 'X' which represents the initial state.
2. Take logical 'AND' of BDD 'X' with the BDD $T(V_t, V_{t+1})$.
3. Existentially quantify out variables in V_t from the resulting BDD.
4. Swap variables $v_i' \in V_{t+1}$ with $v_i \in V_t$ in the BDD got from the last step.

The BDD formed after executing step 4 represents all the immediate successor states from a given initial state.

3 Computing the Steady States

In this section, we will see how to efficiently compute steady states on the BDD representation of the gene regulatory networks. But first we shall give some definitions that we are going to use in the rest of the section. Let, f be the state transition function.

Definition 1. *Forward image, $I_f(S(V_t))$ is the set of immediate successors of the states in $S(V_t)$ on the state transition graph.*

Definition 2. *Backward image, $I_b(S(V_t))$ is the set of immediate predecessors of the states in $S(V_t)$ on the state transition graph.*

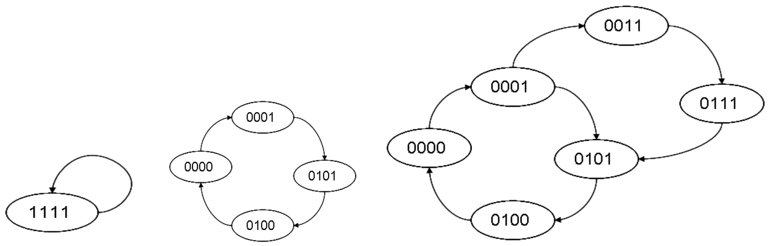
Definition 3. *Forward reachable states $FR(S_0)$ from the states S_0 are all the states that can be reached from S_0 by iteratively computing forward image in the transition relation $T(V_t, V_{t+1})$ until no new states are reachable.*

Definition 4. *Backward reachable states, $BR(S_0)$, are all the states in $T(V_t, V_{t+1})$ whose forward reachable states contain S_0.*

Definition 5. *Steady State is the set of states $SS(V_t)$ having the following two properties:*

1. *Forward image $I_f(SS(V_t))$ is same as $SS(V_t)$.*
2. *For all the states in $SS(V_t)$, if that state is reached once, then the probability of revisiting that state is one. [19]*

The first property of a steady state implies that there are only three possible variants of steady states as shown in figure 4. The second property of steady state ensures that there are only simple cycles(figure 4(a) and 4(b)) in the set $SS(V_{ps})$ and invalidates the third kind of steady state(figure 4(c)) with complex loops as some of the states in this loop might not be revisited. The first property is contained in the second property of steady states. An efficient algorithm can be designed for finding steady states that satisfy the first property, though any efficient algorithm satisfying property 2 is not known yet. Here we use the modified algorithm by [18] for computing the set of steady states satisfying property 1 and then remove the false steady states of type III (in figure 4) from that set.



(a) Self Loop (b) Simple Loop (c) Complex Loop(false steady state)

Fig. 4. Different types of steady states

Algorithm [18] for computing steady states is based on two main theorems. For the sake of completeness of the paper, we present these theorems and algorithms here again. Proof of these theorems can be found in [18].

Theorem 1. *A state $i \in S$ is a steady state if and only if $FR(i) \subseteq BR(i)$. State i is transient otherwise.*

Theorem 2. *If state $i \in S$ is transient, then states in $BR(i)$ are all transient. If state i is steady, then all the states in $FR(i)$ are steady states. In the latter case set $\{BR(i) - FR(i)\}$ are all transient.*

Based on these two theorems, the algorithm for steady state computation is given in Algorithm 2. Algorithm 2 uses the functions *forward_set()* and *backward_set()* for computing forward reachable ($FR(S)$) and backward reachable ($BR(S)$) states respectively. These functions are given in Algorithm 1. In Algorithm 1, FS^k and RS^k, are the frontier set and reachable set respectively in the k^{th} iteration of the while loop. Frontier set (Backward set) in iteration $k+1$, is computed by taking the forward (backward) image of the frontier (backward) set in the k^{th} iteration and removing from this image set the states that have already been explored in previous iterations (which are stored in Reached Set). Reached Set is updated by adding the new states from frontier(backward) set. This process is iterated until no new states can be added to Reached Set. The final Reached Set represents the forward (backward) reachable set from the set of initial states S_0.

The Algorithm 2 uses Theorems 1 and 2. In line 5 of Algorithm 2, a prospective steady state is selected from the state space T' and forward and backward reachable sets from this seed state are computed in lines 6 and 7. Then Theorem 1 as implemented in line 8, checks if the seed state is a steady state. If the seed state is indeed a steady state then using Theorem 2 (as implemented in lines 9-12), all the states in forward reachable set are declared steady states in line 9 and rest of the states in backward reachable set are declared transient states in line 10. Otherwise, the seed state and all the other states in backward reachable set are declared transient in line 12. In line 13, state space is reduced by removing the states that have already been tested for reachability and the process is repeated to find another steady state on the reduced state space. This process is

Algorithm 1. Computing Forward and Backward reachable sets

```
1  $forward\_set(S_0, T)$
2  /* $backward\_set(S_0, T)$ */
3  begin
4      $RS^{(0)} \longleftarrow \emptyset, FS^{(0)} \longleftarrow \{S_0\}$
5      $k \longleftarrow 0$
6      while $FS^{(k)} \neq \emptyset$ do
7          $FS^{(k+1)} = I_f(FS^{(k)})(V_{t+1} \leftarrow V_t) \wedge \overline{RS^{(k)}}$
8          /* $FS^{(k+1)} = I_b(FS^{(k)})(V_t \leftarrow V_{t+1}) \wedge \overline{RS^{(k)}}$ */
9          $RS^{(k+1)} = RS^{(k)} \vee FS^{(k+1)}$
10         $k \longleftarrow k + 1$
11     return $(FR(S_0) \longleftarrow RS^{(k)})$
12     /* return $(BR(S_0) \longleftarrow RS^{(k)}(V_{t+1} \leftarrow V_t))$ */
13 end
```

iterated until the whole state space is explored (i.e. until $T \neq \emptyset$). Since in each iteration, states in backward reachable set are removed from the state space, the size of the state space reduces in each iteration. The number of iterations also depends upon how the seed state is selected.

Function initial_state() in Algorithm 2 selects a prospective steady state from the given state space T'. In this function (implemented in lines 17-25), a BDD representing a random path from the root node to 1-terminal, is selected in line 17. The variables $v_i \in V_{t+1}$ on this path P are removed (line 18) and the resulting BDD is called the intial state,s. Forward reachable set from this random initial state is then computed in lines 19-24. During the forward set computation, when the frontier set evaluates to $\emptyset$ in iteration k, a random state is taken from the frontier set in iteration $k-1$ and returned as the seed state. The motivation behind this function is that a state in the last frontier set is more likely to be a steady state then a random state in the state space T. This function differs from the one given in [18], in which the authors propose to do forward reachability until a user-defined depth (i.e. k is taken as input). But in our experience the number of iterations of while loop in line 4 of Algorithm 2 can be reduced by a large factor if we do complete forward reachability and select a state from the frontier set in the last iteration as compared to selecting from one of the intermediate iterations. This is because any state from frontier set of k^{th} iteration should have a larger backward reachable set then any other state in previous $k-1$ iterations. And larger the size of backward reachable set, smaller the number of iterations required to exhaust the state space.

The Algorithm 2 gives the set of steady states satisfying property 1 as mentioned in the definition of steady states. Pseudocode for doing the complete dynamic analysis is given in Algorithm 3, which uses the function *isFalseLoop()* to check for false steady states. In the function *isFalseLoop()*, given a steady state S, a random state s_0 is selected from this set S and the image of s_0 is computed in line 14. In lines 15-19, we test if the immediate successor state of

Algorithm 2. Steady State Algorithm

```
1  $all\_Steady\_States(T)$
2  begin
3      $T' \longleftarrow T$
4      while $T' \neq \emptyset$ do
5          $s \longleftarrow initial\_state(T')$
6          $FR(s) \longleftarrow forward\_set(s, T')$
7          $BR(s) \longleftarrow backward\_set(s, T')$
8          if $FR(s) \wedge \overline{BR(s)} = \emptyset$ then
9              report $FR(s)$ as a steady state
10             report $BR(s) \wedge \overline{FR(s)}$ as all transient states
11         else
12             report $s \vee BR(s)$ as all transient states
13         $T' \longleftarrow T' \wedge \overline{s \vee BR(s)}$
14 end
15 $initial\_state(T)$
16 begin
17     $P = random\_path\_to\_1\_node(T)$
18     $s(V_t) = \exists_{v \in V_t} P$
19     $RS^{(0)} \longleftarrow \emptyset, FS^{(0)} \longleftarrow \{s\}$
20     $k \longleftarrow 0$
21     while $FS^{(k)} \neq \emptyset$ do
22         $FS^{(k+1)} = I_f(FS^{(k)})(V_{t+1} \leftarrow V_t) \wedge \overline{RS^{(k)}}$
23         $RS^{(k+1)} = RS^{(k)} \vee FS^{(k+1)}$
24         $k \longleftarrow k + 1$
25     $s \longleftarrow random\_path\_to\_1\_node(FS^{(k-1)})$
26     return $s$
27 end
```

this initial state is a single state or a set of state. To do this, a random path(or the state) from the image set is computed(line 15) and removed from this set in line 16. If the resulting set is not empty (line 17), then the given steady state is declared false. Otherwise the lines 13-21 are iterated with the frontier set and the reached set being updated as in line 20 and 21. Function *isFalseLoop()* removes all the type III steady states, because these steady states will always contain one or more states with two possible immediate successors. All the other steady states are simple loops and are reported genuine by this function.

3.1 Results

We have implemented our software, **GenYsis** in C++ using the CUDD software package for BDD manipulation. To analyse the computational efficiency of our methodology, we have tested GenYsis on a range of biological networks of varying complexity. In Table 1 we report the time taken by GenYsis on these

Algorithm 3. Computing all genuine Steady States

```
Input  : Transition function $T$
Output: Steady state array $SS[]$
comp_steady_states()
begin
    $SS[] = all\_Steady\_States(T)$
    for $i = 0$ to $SS[].size()$ do
        if $isFalseLoop(SS[i], T) == false$ then
            report $SS[i]$ as a genuine steady state
end
isFalseLoop(S, T)
begin
    $s_0 = random\_path\_to\_1\_node(S)$
    $RS^{(0)} \longleftarrow \emptyset, FS^{(0)} \longleftarrow \{s_0\}$
    $k \longleftarrow 0$
    while $FS^{(k)} \neq \emptyset$ do
        $FS^{(k+1)} = I_f(FS^{(k)})(V_{t+1} \leftarrow V_t)$
        $s' = random\_path\_to\_1\_node(FS^{(k+1)})$
        $FS_{temp} = FS^{(k+1)} \wedge \overline{s'}$
        if $FS_{temp} \neq \emptyset$ then
            /* false steady state */
            return true
        $FS^{(k+1)} = FS^{(k+1)} \wedge \overline{RS^{(k)}}$
        $RS^{(k+1)} = RS^{(k)} \vee FS^{(k+1)}$
        $k \longleftarrow k + 1$
    /* genuine steady state */
    return false
end
```

Table 1. Computational results on some gene regulatory networks

Network	Nodes	Edges	Steady States	Number of Iterations	Memory Usage	Time taken (in sec)		
						BDD const.	Steady State	Total
Th network	23	34	3	3	< 15 MB	0.001	0.04	0.041
network2	114	129	1	1	< 15 MB	0.03	0.001	0.031
network3	669	2710	4	10	< 17 MB	0.07	3.15	3.22
network4	1263	5031	1	1	< 57 MB	0.95	314.55	315.55

sample networks. The run time is divided into two parts: time taken to construct BDD and time taken to compute steady states. We also measure the memory requirements for each sample network when analysed by GenYsis. All the results are reported on a 1.6 GHz machine running on linux operating system.

In Table 1, we see that the networks with a size as big as 1263 nodes and 5235 edges can be analysed in less then 6 minutes by using GenYsis. Finding all possible steady states for large network was not feasible with the previous methodologies based on finding the characteristic state of all the feedback loops in the network. Also, GenYsis follows a very intutive way to explore the state space of the network rather then an indirect and difficult to comprehend way of computing the characteristic state.

4 T Helper Cell Differentiation

The vertebrate immune system is constituted by diverse cell populations. Here, we will focus in the CD4+ T lymphocytes known as T helper (Th) cells. These cells conform a particularly suitable differentiation model, because there is a type of precursors cells (Th0), which upon receiving an appropriate antigenic stimulus in vitro, can be further differentiated into cytokine-secreting effector cells, either Th1 or Th2 cells. At the molecular level, Th1 and Th2 cells can be distinguished by their pattern of cytokine secretion, which are responsible for their central role in cell mediated immunity (Th1 cells) and humoral responses (Th2 cells). Understanding the molecular mechanisms that regulate the differentiation process from Th0 towards either Th1 or Th2 is very important, since an immune response biased towards the Th1 phenotype result in the appearance of autoimmune diseases, and an enhanced Th2 response can originate allergic reactions [4,10].

There are several factors at the cellular and molecular levels that determine the differentiation of T helper cells. Importantly, the cytokines present in the cellular milieu play a key role in directing Th cell polarization. On the one hand, IFN-γ, IL-12, IL-18 and IL-27 are the major cytokines that promote Th1 development [11].And on the other hand, IL-4 is the major cytokine responsible for driving Th2 responses. Besides this positive roles of cytokines in the differentiation process, there exist also a mutual inhibitory mechanism. Specifically, IFN-γ play a role in inhibiting the development of Th2 cells, whereas IL-4 inhibits the appearance of Th1 cells. This interplay of positive and negative signals, at both the cellular and molecular levels, creates a complexity that is very suitable for analysis by the modeling approach.

Due to its physiological relevance, there are various mathematical models that have been proposed for describing the differentiation, activation and proliferation of T helper lymphocytes. Most of these models, however, focus on interactions established among the diverse cell populations that somehow modify the differentiation of Th cells [5,17]. Also, other modeling efforts have been aimed at understanding the mechanism of the generation of antibody and T-cell receptor diversity, as well as the molecular networks of cytokine or immunoglobulin interactions [6,16].

Recently we published the first analyses on the gene regulatory network that controls the differentiation process from Th0 to either Th1 or Th2 cells [8,7]. The network (Fig 5) is made of 23 nodes, 26 positive and 8 negative interactions. Importantly, the model does not need to be seen as metabolic pathway, or a reaction network, but rather as an information processing network.

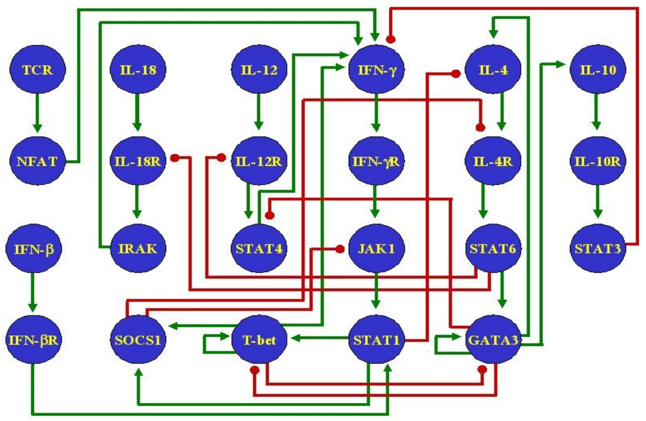


Fig. 5. Th network. The regulatory network that controls the differentiation process of t helper cells. Positive regulatory interactions are with pointed arrow head and negetive interactions with round arrow head.

We already studied the dynamical behavior of the Th network using both discrete and continuous approaches. Such studies permitted the identification of all the stable states of the system. Specifically, the dynamical system obtained from the network has three stable fixed points, which correspond to the patterns of activation observed in normal Th0, Th1 and Th2 cells. Moreover, we were able to modify the model so as to describe the patterns of expression of null mutants, as well as constitutive-expression variants.

Central to our previous analyses is the use of the generalized logical analysis [14,15] for the qualitative analysis of the dynamical properties of the system by focusing on the feedback loops present in the network. Besides helping to understand the Th network, the generalized logical analysis has been applied to other regulatory networks, including those involved in organ differentiation control in the flowers of *Arabidopsis thaliana* [9], and in the initiation of segmentation during *Drosophila melanogaster* embryogenesis [12,13]. Despite its usefulness, the generalized logical analysis has two main drawbacks. First, the computational time needed to analyze all possible feedback loops in a network grows very fast, so that it is not feasible to study large networks. And second, to study the behavior of mutants, it is necessary to create alternative models where the parameters reflect the intended mutation, so that the number of models multiplies by the number of intended mutants. Hence, an alternative, faster and more easily scalable methodology is required for the study of the dynamical properties of biological networks. Our new approach of BDD representation for gene regulatory networks, can provide an alternate way for efficiently analyzing feedback loops in the network and perform *in silico* gene perturbation experiments. When we apply GenYsis on the T helper cell network of Figure 5, we get the three wild type steady states as listed in Table 2.

Table 2. Steady state of the wild type and virtual knock-out of IFN-γ and IFN-γR

Knocked Genes	Active genes in steady states
wild type	IFN-γ Tbet SOCS-1 IFN-γR
	All the genes are inactive
	GATA-3 IL-10 IL-10R IL-4 IL-4R STAT3 STAT6
IFN-γ^-	Tbet SOCS-1
	All the genes are inactive
	GATA-3 IL-10 IL-10R IL-4 IL-4R STAT3 STAT6
IFN-γR$^-$	IFN-γ Tbet SOCS-1
	All the genes are inactive
	GATA-3 IL-10 IL-10R IL-4 IL-4R STAT3 STAT6

These steady states correspond to the molecular profiles observed in Th0, Th1 and Th2 cells respectively. The first steady state reflects the pattern of Th0 cells, which are precursor cells that do not produce any of the cytokines included in the model (IFN-β, IFN-γ, IL-10, IL-12, IL-18 and IL-4). The second steady state represents Th1 cells with high activation of IFN-γ, IFN-γR, T-bet and SOCS1. Finally the third steady state corresponds to the activation observed in th2 cells, with high level of activation of GATA-3, IL-10, IL-10R, IL-4, IL-4R, STAT3 and STAT6. These results also match those published in [8]. GenYsis took only 0.04 seconds to compute these steady states.

In the literature, modeling of Th cell differentiation at the molecular level has been shown to be very useful to bring insight into the origin of the unexpected phenotypes. Previously [7], we made an explanation for the unexpected phenotypic similarity between IFN-γ and IFN-γR loss-of-function mutants (figure 5) [26] [25]. Similarly, using our new BDD based methods we performed virtual knock-out on both IFN-γ and IFN-γR (see Table 2) and compared it to the unperturbed system (wild type). In the case of the IFN-γ knock-out, both the IFN-γ and its receptor are removed from the identified steady state. However when IFN-γR is knocked out, the steady state observed still contains the production of IFN-γ. This is similar to what was obtained with the standard GLA approach from [7], but GenYsis is 100x faster then latter.

5 Conclusion

This paper gives an efficient way for modeling the gene regulatory networks and perform dynamic analysis on them. This new approach can model very efficiently even the biggest regulatory networks available to the modeling community and provide means to perform *in silico* experiments on them. The proposed method has been applied on a T helper cell regulatory network. From the whole range of experiments that were tested with GenYsis, we have reported in this paper, two very interesting knock-outs which have been studied extensively by the modelling community for a long time. In future, we will be extending GenYsis to perform whole suite of *in silico* gene perturbation experiments including gene over-expression and multiple perturbations.

Acknowledgements

This work was supported in part by ENFIN, a Network of Excellence funded by the European Commission within its FP6 Programme, under the thematic area "Life Sciences, genomics and biotechnology for health", contract number LSHG-CT-2005-518254.

Abhishek Garg would like to thank Luca Benini for his help and discussion on the technical details of this paper.

References

1. Bryant, R.E., 'Graph-Based Algorithms for Boolean Function Manipulation'. *IEEE Trans. on Computers*,Vol. 35 (1986), 677–691.
2. Burch, J.R. and Clarke, E.M. and Long, D.E. and MacMillan, K.L. and Dill, D.L., 'Symbolic Model Checking for Sequential Circuit Verification'. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13 (1994), 401–424.
3. Touati, H.J, Savoj, H., Lin, B., Brayton, R.K., Sangiovanni-Vincentelli, A., 'Implicit state enumeration of finite-state machines using BDDs'. *Proc. of ICCAD*, 1990.
4. Agnello, D., Lankford, C.S.R., Bream, J., Morinobu, A., Gadina, M., OShea, J. and Frucht, D.M., 'Cytokines and transcription factors that regulate T helper cell differentiation: new players and new insights'. *J. Clin. Immun.*, Vol. 23 (2003), 147–162
5. Bergmann, C. and van Hemmen, J.L., 'Th1 or Th2: how an appropriate T helper response can be made'. *Bull. Math. Bio.*, Vol. 63 (2001), 405-430.
6. Krueger, G.R., Marshall, G.R., Junker, U., Schroeder, H., Buja, L.M. and Wang, G., 'Growth factors, cytokines, chemokines and neuropeptides in the modeling of T-cells'. *In Vivo*, Vol. 16(2002), 365-586.
7. Mendoza, L., A network model for the control of the differentiation process in Th cells. BioSystems, Vol 84 (2006), 101-114.
8. Mendoza, L. and Xenarios, I., 'A method for the generation of standardized qualitative dynamical systems of regulatory networks'. *Theoretical Biology and Medical Modelling*, Vol. 3 (2006).
9. Mendoza, L., Thieffry, D., Alvarez-Buylla, E.R., 'Genetic control of flower morphogenesis in Arabidopsis thaliana: a logical analysis.' *BioInfo.*, Vol. 15 (1999), 593-606.
10. Murphy, K.M. and Reiner, S.L., 'The lineage decisions on helper T cells'. *Nat. Rev. Immun.*, 2002, 933-944.
11. Szabo,S.J., Sullivan, B.M., Peng, S.L. and Glimcher, L.H., 'Molecular mechanisms regulating Th1 immune responses'. *Ann. Rev. Immun.*, Vol. 21 (2003), 713-758.
12. Thieffry, D. and Sánchez, L., 'Alternative epigenetic states understood in terms of specific regulatory structures'. *Ann. N.Y. Acad. Sci.*, Vol. 981 (2002), 135-153.
13. Sánchez, L. and Thieffry, D., 'Segmenting the fly embryo: a logical analysis of the pair-rule cross-regulatory module'. *Jour. Theo. Bio.*, Vol. 224 (2003), 517-537.
14. Thomas, R., 'Regulatory networks seen as asynchronous automata: a logical description'. *Jour. Theo. Bio.*, Vol. 153 (1991), 1-23.
15. Thomas, R., Thieffry, D. and Kaufman, M., 'Dynamical behaviour of biological regulatory networks-I. Biological role of feedback loops and practical use of the concept of the loop-characteristic state'. *Bull. Math. Biology*, Vol. 57 (1995), 247-276.

16. Weisbuch, G., DeBoer, R.J. and Perelson, A.S., ‘Localized memories in idiotypic networks’. *Jour. Theo. Bio.*, Vol. 146 (1990), 483-499.
17. Yates, A., Bergmann, C., van Hemmen, J.L., Stark, J. and Callard, R., ‘Cytokine-modulated regulation of helper T cell populations’. *Jour. Theo. Bio.*, Vol. 206 (2000), 539-560.
18. Xie, A. and Beerel, P.A., ‘Efficient State Classification of Finite State Markov Chains’. *Proc. of DAC*, 1998.
19. Hachtel, G. Macii, E., Pardo, A. and Somenzi, F., ‘Markovian analysis of large finite state machines’. *IEEE Trans. on CAD*, Vol. 15 (1996), 1479-1493.
20. Somenzi, F, ‘CUDD: CU Decision Diagram Package Release 2.4.1.’. *University of Colorado at Boulder*. 2005.
21. Brayton, R.K., Sangiovanni-Vincentelli, A.L., McMullen, C.T., and Hachtel, G.D., *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.
22. DeMicheli, G., *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1994.
23. Burch, J.R., Clarke, E.M., MacMillan, K.L., Dill, D.L. and Hwang, L.H., ‘Symbolic Model Checking:: 10^{20} States and Beyond’. *In Proc. of the IEEE Symp. on Logic in Computer Science*, 1990.
24. Alur, R., Henzinger, T.A., Mang, F.Y.C., Qadeer, S., Rajamani, S.K. and Tasiran, S., ‘MOCHA: Modularity in Model Checking’. *CAV*, 1998.
25. Diehl, S., Anguita, J., Hoffmeyer, A., Zapton, T., Ihle, J.N., Fikrig, E. and Rincón, M., ‘Inhibition of Th1 differentiation by IL-6 is mediated by SOCS1’. *Immunity*, Vol. 13 (2000), 805-815.
26. Tang, H., Sharp, G.C., Peterson, K.P. and Braley-Mullen, H., ‘IFN-g-deficient mice develop severe granulomatous experimental autoimmune thyroiditis with eosinophil infiltration in thyroids’. *Jour. Immun.*, Vol. 160 (1998), 5105-5112.
27. Bernot, G., Comet, J.P., Richard, A. and Guespin, J., ‘Application of formal methods to biological regulatory networks: extending Thomas’ asynchronous logical approach with temporal logic’. *Jour. Theo. Bio.*, Vol. 229 (2004), 339-347.
28. Devloo,V., Hansen, P. and Labb, M., ‘Identification Of All Steady States In Large Biological Systems By Logical Analysis’. *Bull. Math. Bio.*, Vol. 65 (2003), 1025-1051.
29. Chabrier, N., Fages, F. and Soliman, S., ‘BIOCHAM’. *Proc. of CMSB*, May 2004.