BIOINFORMATICS ORIGINAL PAPER

Vol. 24 no. 17 2008, pages 1917–1925
doi:10.1093/bioinformatics/btn336

Systems biology

Synchronous versus asynchronous modeling of gene regulatory networks

Abhishek Garg[1,*], Alessandro Di Cara[2], Ioannis Xenarios[3], Luis Mendoza[4] and Giovanni De Micheli [1]

[1]Ecole Polytechnique Federale de Lausanne, Station 14, 1015 Lausanne, [2]Merck Serono, Geneva, [3]Swiss Institute of Bioinformatics, Vital-IT Group, 1015 Lausanne, Switzerland and [4]Instituto de Investigaciones Biomédicas, Universidad Nacional Autónoma de México, México

Received on February 01, 2008; revised on June 10, 2008; accepted on July 01, 2008

Advance Access publication July 9, 2008

Associate Editor: Olga Troyanskaya

ABSTRACT

Motivation: *In silico* modeling of gene regulatory networks has gained some momentum recently due to increased interest in analyzing the dynamics of biological systems. This has been further facilitated by the increasing availability of experimental data on gene–gene, protein–protein and gene–protein interactions. The two dynamical properties that are often experimentally testable are perturbations and stable steady states. Although a lot of work has been done on the identification of steady states, not much work has been reported on *in silico* modeling of cellular differentiation processes.

Results: In this manuscript, we provide algorithms based on reduced ordered binary decision diagrams (ROBDDs) for Boolean modeling of gene regulatory networks. Algorithms for synchronous and asynchronous transition models have been proposed and their corresponding computational properties have been analyzed. These algorithms allow users to compute cyclic attractors of large networks that are currently not feasible using existing software.

Hereby we provide a framework to analyze the effect of multiple gene perturbation protocols, and their effect on cell differentiation processes. These algorithms were validated on the T-helper model showing the correct steady state identification and Th1–Th2 cellular differentiation process.

Availability: The software binaries for Windows and Linux platforms can be downloaded from http://si2.epfl.ch/∼garg/genysis.html.

Contact: abhishek.garg@epfl.ch

1 INTRODUCTION

Qualitative modeling of gene regulatory networks has been addressed by various authors (Albert and Othmer, 2003; Bernot *et al*., 2004; Chabrier *et al*., 2004; Devloo *et al*., 2003; Fauré *et al*., 2006; Kauffman, 1969; Klamt *et al*., 2006; Mendoza and Xenarios, 2006; Naldi *et al*., 2007; Remy *et al*., 2006; Thomas, 1991; Thomas and Kaufman, 1995). To model these networks, models with different updating schemes have been proposed, namely synchronous (Fauré *et al*., 2006; Naldi *et al*., 2007; Remy *et al*., 2006), asynchronous (Devloo *et al*., 2003; Fauré *et al*., 2006; Garg *et al*., 2007; Thomas, 1991) and semi-asynchronous models (Fauré *et al*., 2006). In a synchronous dynamic model, all genes change their expression levels simultaneously in consecutive time points. This model is computationally tractable for very large networks but at the expense of accuracy as, in reality, different genes take different time to make the transition from one expression level to another. Asynchronous dynamic models, in which all genes take different time for making a transition, are closer to biological phenomena but they are more complex to model and to analyze. The synchronous and asynchronous dynamics of Boolean models of gene regulatory networks have been qualitatively addressed in the past (Fauré *et al*., 2006). Studies have shown that different updating rules may lead to different attractors. Usually asynchronous rules result in a larger number of states conforming an attractor, as well as lengthier transitory states leading to an attractor. It is clear that it is important to use adequate updating rules to obtain a biologically correct dynamical model. However, till date no efficient algorithm has been reported to compute all the attractors of large regulatory networks using both synchronous and asynchronous updating rules.

In this manuscript, we extend a previous methodology (Garg *et al*., 2007) so as to identify the different kinds of attractors that may exist in gene regulatory networks. We compare computational efficiency of synchronous and asynchronous models of gene regulatory networks and propose an efficient method that performs combined synchronous–asynchronous simulations for faster and accurate identification of attractors. We also extend our previous methodology to incorporate gene perturbation experiments on regulatory networks.

2 DYNAMIC MODELS OF GENE REGULATORY NETWORKS

We previously described (Garg *et al*., 2007) how gene regulatory networks can be modeled efficiently using reduced ordered binary decision diagrams (ROBDDs or in short BDDs) (Bryant, 1986). BDDs are directed acyclic graphs that can represent large Boolean functions in a space efficient manner, and are computationally suitable for complex Boolean operations (e.g. logical AND, OR, etc.) and set operations (e.g. Union, Intersection, etc.). To map gene regulatory networks on BDDs, the first step is to transform networks into Boolean functions which represent the dynamics of

*To whom correspondence should be addressed.

a model. All the operations that can be performed on Boolean functions can also be performed on their corresponding BDD representations. In this manuscript, we use Boolean functions and Boolean variables to represent gene regulatory networks and to describe our algorithms. However, all Boolean operations in our software are implemented using BDDs.

In Equation (1), activator (or inhibitor) functions represent the set of genes that have a collective activating (or inhibiting) impact on the level of activation of a given gene. Equations (1) and (2) can be understood better with the help of Figure 1.

Given a gene regulatory network, the state of a node (or gene) i at time t is represented by a Boolean variable $x_i(t)$. To evaluate the evolution in time of each gene, it is necessary to describe the state of each gene at time $t+1$ as a function of the state of those genes acting as its input at time t. 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 at time t is represented by a Boolean vector, $\mathbf{x}_t$, 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, $\mathbf{x}_{t+1}$, of size N is used to represent the status of the network in the next step. We call the previous vector *present state* and the latter one *next state*.

Following the standardized qualitative dynamical systems methodology (Mendoza and Xenarios, 2006), Equation (1) states that inhibitor functions are strong enough to change the state of the output gene from 1 to 0, while activator functions can change the state from 0 to 1 if and only if no inhibitor functions are acting on that gene. Although this might give an impression of a repressor dominated system, in practice, a situation where an activator dominates the repressor can be represented by Equations (1) and (2) as shown in Figure 2.

$$x_i(t+1)=\left(\bigvee_{l=1}^{m} f^{a}_{x_{i,l}}(t)\right)\wedge\neg\left(\bigvee_{l=1}^{n} f^{in}_{x_{i,l}}(t)\right) \tag{1}$$

$$f^{a,in}_{x_{i,l}}(t)=\left(\bigwedge_{j=1}^{p} x^{a}_{j}(t)\right)\wedge\left(\bigwedge_{j=1}^{q}\neg x^{in}_{j}(t)\right) \tag{2}$$

$$x_j \in \{0,1\}$$

$f^{a}_{x_m}$ are the set of activator functions of x_i

$f^{in}_{x_n}$ are the set of inhibitor functions of x_i

x^{a}_{p} are the set of activators of functions f_{x_i}

x^{in}_{q} are the set of inhibitors of functions f_{x_i}

$\wedge$, $\vee$ *and* $\neg$ represent logical AND, OR and negation

Equations (1) and (2) represent the dynamics of individual genes independent of the dynamics of the other genes in the network. To model the dynamics of the complete network, one has to couple the dynamics of these genes. This can be done by defining the transition function, $T(\mathbf{x}_t,\mathbf{x}_{t+1})$, of the state of the network. $T(\mathbf{x}_t,\mathbf{x}_{t+1})$ represents the transition from the present state $\mathbf{x}_t$ to the next state $\mathbf{x}_{t+1}$.

Transition functions can be either *synchronous* or *asynchronous*. If the transition function is synchronous, all the genes are updated according to their rules at the same time. If the transition function

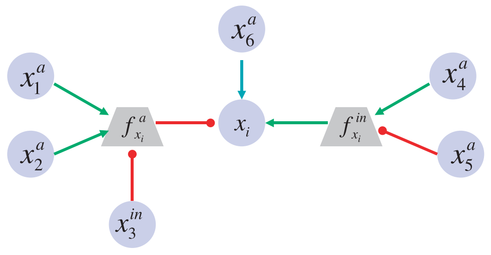

Fig. 1. An example of a gene regulatory network. Circle-headed arrows represent inhibition and arrows represent activation. Nodes with label f are only used for explanation and are not part of the gene regulatory network. $f^{in}_{x_{i,1}}(t)=(x_1\wedge x_2\wedge\neg x_3)$, $f^{a}_{x_{i,1}}(t)=(x_4\wedge\neg x_5)$, $f^{a}_{x_{i,2}}(t)=x_6$. $x_i(t+1)=\left(f^{a}_{x_{i,1}}(t)\vee f^{a}_{x_{i,2}}(t)\right)\wedge\neg f^{in}_{x_{i,1}}(t)$.

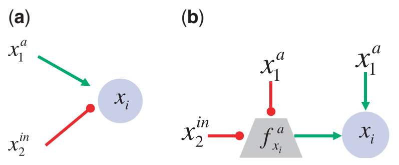

Fig. 2. A small representation of how activator dominated circuits can be represented using Equations (1) and (2). (**a**) A small gene regulatory network where gene x^a_1 dominates over x^{in}_2. (**b**) Expanded functional representation using Equations (1) and (2), where $f^a_{x_i}(t)=(\neg x^a_1\wedge\neg x^{in}_2)$, $x_i(t+1)\ =\ (x^a_1\vee f^a_{x_i}(t))$.

is asynchronous, at most one gene is updated between two consecutive states. In the following sections we define synchronous and asynchronous models in terms of these functions and variables.

2.1 Synchronous model

A synchronous model can be described by the following set of equations:

$$T_i(\mathbf{x}_t,\mathbf{x}_{t+1})\ =\ \left(x_i^{t+1}\leftrightarrow x_i(t+1)\right) \tag{3}$$

$$T(\mathbf{x}_t,\mathbf{x}_{t+1})\ =\ T_0(\mathbf{x}_t,\mathbf{x}_{t+1})\wedge\ldots\wedge\ T_N(\mathbf{x}_t,\mathbf{x}_{t+1}) \tag{4}$$

Equation (3) gives the transition function for a single gene i. Symbol $\leftrightarrow$ stands for logical equivalence and in Equation (3), represents that the value of a gene in the next time step, x_i^{t+1} is equal to the value of the function $x_i(t+1)$ [as defined in Equation (1)]. Equation (4) states that all the genes in the network make a simultaneous transition from the present state $\mathbf{x}_t$ to the next state $\mathbf{x}_{t+1}$.

If a state transition graph is constructed using Equations (3) and (4), then each state has only one transition going out of it. Hence, assuming that all the genes can take '0' or '1' levels of expression, the number of states and the number of transitions in the state transition graph are both equal to 2^N, where N is the number of genes in the network. If this state transition graph is explicitly represented and traversed, then an exponential number of states restricts the computation to small sized networks. To avoid explicit enumeration, we proposed in Garg *et al.* (2007) an implicit method based on ROBDD (Burch *et al.*, 1994; Xie and Beerel, 1998) to compute attractors using Boolean equations having a similar representation as Equations (1–4). For completeness, we present them here again

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          $FS^{(k+1)} = I_f(FS^{(k)})(x_{t+1} \leftarrow x_t) \wedge \overline{RS^{(k)}}$
8          /* $FS^{(k+1)} = I_b(FS^{(k)})(x_t \leftarrow x_{t+1}) \wedge \overline{RS^{(k)}}$ */
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 ($BR(S_0) \longleftarrow RS^{(k)}(x_{t+1} \leftarrow x_t)$) */
13 end
```

in Algorithms 1 and 2, which are based on the following definitions and theorems:

DEFINITION 1. *Successor (Predecessor). Given a state of the network* $\mathbf{x}_t$ *(*$\mathbf{x}'_t$*), all the states* $\mathbf{x}'_t$ *(*$\mathbf{x}_t$*) such that* $T(\mathbf{x}_t,\mathbf{x}'_t)=1$ *are the successor (predecessor) states of the state* $\mathbf{x}_t$ *(*$\mathbf{x}'_t$*).*

DEFINITION 2. *Forward (Backward) image,* $I^f_T(S(x_t))$ *(*$I^b_T(S(x_t))$ *) is the set of immediate successors (predecessors) of the states in the set* $S(x_t)$ *on the state transition graph defined under the transition function* T*.*

DEFINITION 3. *Forward reachable states* $FR(S_0)$ *from the states set* S_0 *are all the states that can be reached from the states in the set* S_0 *by iteratively computing forward image on the transition function* $T(x_t,x_{t+1})$ *until no new states are reachable.*

DEFINITION 4. *Backward reachable states,* $BR(S_0)$*, are all the states* x_t *whose forward reachable set contain at least one state in* S_0*.*

DEFINITION 5. *An Attractor is the set of states* $SS(x_t)$ *such that for all the states* $s\in SS(x_t)$*, the forward reachable set* $FR(s)$ *is the same as* $SS(x_t)$*.*

DEFINITION 6. *A Steady State is an attractor that consists of a single state.*

THEOREM 1. *A state* $i\in S$ *is a part of an attractor 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 a part of an attractor, then all the states in* $FR(i)$ *are also part of the same attractor. In the latter case set* $\{BR(i)-FR(i)\}$ *has all the transient states.*

Proof of these theorems can be found in Xie and Beerel (1998). Based on these two theorems, the procedure for attractor computation is given in Algorithm 2. This algorithm uses Theorems 1 and 2. In Line 5 of Algorithm 2, a seed state is selected from the state space T' and forward and backward reachable states from this seed state are computed in Lines 6 and 7. Then Theorem 1, as implemented in Line 8, checks if the seed state (from Line 5) is part of an attractor. If the seed state is indeed part of an attractor, then using Theorem 2 (as implemented in Lines 9–12), all the states in the forward reachable set are declared to from an attractor in

Algorithm 2 Algorithm for computing attractors

```
1  all_attractors(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 an attractor
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     $s(V_t)$ = random_state(T)
18     $RS^{(0)} \longleftarrow \emptyset, FS^{(0)} \longleftarrow \{s\}$
19     $k \longleftarrow 0$
20     while $FS^{(k)} \neq \emptyset$ do
21         $FS^{(k+1)} = I_f(FS^{(k)})(x_{t+1} \leftarrow x_t) \wedge \overline{RS^{(k)}}$
22         $RS^{(k+1)} = RS^{(k)} \vee FS^{(k+1)}$
23         $k \longleftarrow k+1$
24     $s \longleftarrow$ random_state($FS^{(k-1)}$)
25     return $s$
26 end
```

Line 9 and the rest of the states in the backward reachable set are declared transient in Line 10. Otherwise, the seed state and all the other states in the backward reachable set are declared transient in Line 12. In Line 13, the state space is reduced by removing those states that have already been tested for reachability and the process is repeated to find another attractor on the reduced state space. This process is iterated until the whole state space is explored (i.e. until $T\neq\emptyset$). The states in the backward reachable set are removed from the state space in each iteration, resulting in the continuous size reduction of the latter. One should note that the number of iterations of Lines 4–13 depends upon how the seed state is selected in Line 5.

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, the while loop in Lines 6–10 computes the reachable states iteratively starting from the intial set of states S_0, where k-th iteration represents the states reachable in k time steps from S_0. FS^k and RS^k are the frontier set and the reachable set respectively in the k-th iteration of the while loop. The Frontier set in the k-th iteration, contains the states which have been reached for the first time in the $(k-1)$th iteration of the while loop. The Reachable set in the k-th iteration contains all reachable states from the initial set S_0 up to k iterations. The Frontier set in iteration $k+1$ is computed by taking the forward image (backward image for backward reachable set computation) of the frontier 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). The Reached Set is updated by adding the new states from the frontier 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.

Function *initial_state*() in Algorithm 2 selects a seed state from the given state space T'. In this function (implemented in Lines 17–25), a random initial state is selected from the transition state space T in Line 17. The 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 part of an attractor than a random state in the state space T. For synchronous models, it can be proved that the seed state selected in this way is guaranteed to be a part of an attractor.

Results of running the Algorithm 2 on some of the benchmark networks are given in Table 1. From the results we see that the synchronous algorithm scales well with the size of the network and can compute all the attractors in reasonable time and memory. The benchmarking was performed on a 1.8 GHz Dual Core Pentium machine with 1 GB of RAM running on Linux Fedora Core 5.

2.2 Asynchronous model

In the synchronous model, we made the assumption that all genes make a transition at the same time and take an equivalent amount of time in changing their expression levels. This assumption is biologically unrealistic, but there is seldom enough kinetic information to be able to discern the precise order and duration of state transitions. To address this problem we make use of the asynchronous model, where we make three assumptions. The first assumption is that only one gene can make a transition (or be updated) in a single step. The second assumption is that a state can have a self transition if and only if none of the genes can change their expression levels (i.e. $x_i(t+1)=\mathbf{x}_t^i\ \forall\ i$). Finally, the third assumption is that every gene is equally likely to make a transition. That means every state can have potentially N successor states, where each successor state differs from the present state in only one gene expression.

As in Equation (3) for synchronous models, we first give the transition function for each gene i for an asynchronous model. To model the first assumption of asynchronous models, which states only one gene can be updated between two consecutive time steps, we use the following relation:

$$TP_i(\mathbf{x}_t,\mathbf{x}_{t+1}) = \left(x_{t+1}^i \leftrightarrow x_i(t+1)\right) \wedge \bigwedge_{j\neq i}\left(x_{t+1}^j \leftrightarrow x_t^j\right) \qquad (5)$$

Equation (5) states that gene i in the next time step, takes the value as defined by the function $x_i(t+1)$ in Equation (1) and all the other genes stay at their current expression levels. For second assumption, we would like to check if all the genes stay at the same level in the next time step. For that we compute a *flag function*, F:

$$F(x_t) = \bigwedge_{i=1}^{N}\left(x_i(t+1)\ \bar{\oplus}\ x_t^i\right) \qquad (6)$$

The Flag function $F(x_t)=0$ iff $x_i(t+1)\neq x_t^i$ for at least one gene i. The transition relation for gene i is then given by:

$$T_i(\mathbf{x}_t,\mathbf{x}_{t+1}) = \{F(x_t)\vee\left(x_i(t+1)\ \oplus\ x_t^i\right)\}\wedge TP_i(\mathbf{x}_t,\mathbf{x}_{t+1}) \qquad (7)$$

In Equation (7), the transition function for gene i is given by $TP_i(\mathbf{x}_t,\mathbf{x}_{t+1})$ if either all the genes stay at the same level in the next time step (i.e. the flag function, $F(x_t)=1$) or if the expression of gene i in the next step can be different from its expression in the present time step (i.e. $\left(x_i(t+1)\ \oplus\ x_t^i\right)=1$).

To compute the corresponding asynchronous transition function, we take the disjunction of Equation (7) for all the genes in the network:

$$T(\mathbf{x}_t,\mathbf{x}_{t+1}) = T_0(\mathbf{x}_t,\mathbf{x}_{t+1})\vee....\vee\ T_N(\mathbf{x}_t,\mathbf{x}_{t+1}) \qquad (8)$$

Equation (8) incorporates the third assumption, which states that from a given state, the network can have multiple next states and each next state can differ from the present state in at most one gene expression. Note that Equation (8) is the counterpart of Equation (4) for synchronous models, with the difference that the conjunctions ($\wedge$) have been replaced by disjunctions ($\vee$).

The asynchronous model has 2^N states, and each of them can have upto N outgoing transitions, making a total of $N\cdot 2^N$ transitions in the worst case. This means that the number of transitions in an asynchronous model can be more than those in the corresponding synchronous model by upto a factor of N. This increased number of transitions has a considerable impact on the BDD representation of the transition function, whose complexity increases with the number of transitions that it needs to model.

In asynchronous models, the attractors are computed as in the synchronous model but by using the transition functions as described in Equations (5–8). Column 9 of Table 1 presents the benchmarking of the asynchronous model. The increased run time to compute the attractors in asynchronous models is due to both the size of the state transition diagram and the heuristics used to select seed states in Algorithm 2. Contrary to synchronous models, for the asynchronous model, the *initial_state*() function in Algorithm 2 does not guarantee to return a state that forms a part of an attractor. This creates a potential problem, since, for computing the set of attractors common to the synchronous and the asynchronous model of a gene regulatory network, Algorithm 2 for the asynchronous model may require a large number of iterations. For this reason, in Table 1, mammalian cell, T-helper(Th) and T-cell receptor have a greater differences in computational time for synchronous and asynchronous model as compared to the run time difference for the dendritic cell network. The large time difference between the two models for the Network 1 is due to the extremely large size and complex configuration of this network.

2.3 Modeling asynchronously using a synchronous model

As is evident from the results from Table 1, BDD representation and manipulation can be far more efficient on a synchronous representation as compared to the asynchronous representation. A similar conclusion has been made in an entirely different context in electronics design automation commmunity, where verification of asynchronous circuits using BDDs is known to be more computationally challenging than synchronous circuits (Baldamus and Schneider, 2001; Burch *et al.*, 1991). To tackle this issue, in this section we propose a combined synchronous–asynchronous traversal technique. We will see that, by using this combined traversal technique, we can find the attractors of an asynchronous model of networks with as many as 1200 nodes and

Table 1. Benchmarking of the synchronous model (Column 8) using Algorithm 2, asynchronous model (Column 9) using Algorithm 2 and combined synchronous asynchronous model (Column 10) using Algorithm 3

Network	Nodes	Edges	Number of Attractors				Time taken (in sec)		
			Self Loops	Simple	Simple 2	Complex	sync	async	combined
Mammalian Cell	10	39	1	0	1	1	0.1	0.26	0.22
T-helper	23	34	3	0	0	0	0.12	0.35	0.4
Dendritic Cell	114	129	0	1	0	0	0.32	0.37	0.49
T-cell receptor	40	58	1	0	9	7	3.0	960	460
Network 1	1263	5031	1	0	0	0	200	*	730

A cut-off time of 1 h was used and the algorithms which could not finish computation within this time were terminated (represented by '*'). Mammalian cell network is taken from (Fauré *et al*., 2006), Th from (Mendoza and Xenarios, 2006) and T-cell receptor from (Klamt *et al*., 2006). The Dendritic cell network was generated by semi-automatic mining of literature evidence. Network 1 is a full literature mined Insulin growth factor regulatory network. It has been developed through automatic literature mining tools that build a tentative regulatory network based on the set of keywords such as activation/inhibition.

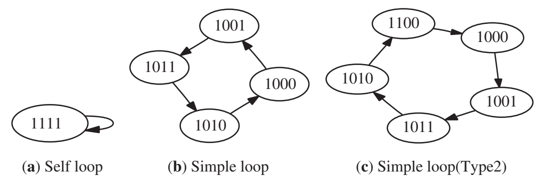

Fig. 3. Possible types of attractors in the synchronous model.

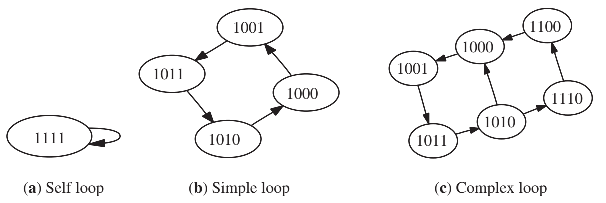

Fig. 4. Possible types of attractors in the asynchronous model.

over 5000 edges (Table 1), which is not otherwise possible using the standalone asynchronous model of Section 2.2.

First, we would like to differentiate between the attractors of synchronous and asynchronous models of gene regulatory networks. An *attractor* is a set of states such that all the states in the set are reachable from each other and all the transitions emerging from the states in this set have the destination state belonging to this set. Based on this definition, an attractor could be formed by a single state or a set of states. A single state attractor is called a *self loop*, otherwise it can be either a *simple loop* or a *complex loop*. A *simple loop* is a cycle of states such that each state can have exactly one successor state. A *complex loop* is formed by two or more overlapping simple loops. A self loop attractor is called a steady state.

Since synchronous networks can have only one outgoing transition from any state, an attractor in synchronous networks can only be of two types: (a) self loops, and (b) simple loops. Simple loops for synchronous models can again be divided into two subclasses: (1) where any two consecutive states of the loop differ in exactly a single gene expression and (2) where at least two consecutive states of the loop differ in more than one gene expression ($1010 \rightarrow 1100$ in Figure 3c). These possible attractors are shown in Figure 3.

Similarly, an asynchronous model can have three classes of attractors, namely, (a) self loops, (b) simple loops, and (c) complex loops. These are shown in Figure 4. Since the definition of asynchronous models allows only a single gene expression change between any consecutive states, they can have only one kind of simple loops (unlike synchronous models).

Self loops and simple loops found in the asynchronous model of a gene regulatory network are also present in the synchronous model of the same network. The two models can only differ in the complex loop and simple loops of the second type (i.e. Fig. 3c). Though simple loops (of Fig. 3c) may lead to a complex loop (of Fig. 4c) in an asynchronous model for some gene regulatory networks, the presence of the former is not necessary for the existence of complex loops. This necessitates the computation of complex loops on asynchronous models, while self loops and simple loops of asynchronous models can be determined by simulating the synchronous model.

We propose to use the algorithms given in Section 2.1 to compute the common attractors on the synchronous model and then compute the complex loop attractors on the asynchronous model. This can improve the efficiency of the algorithms proposed in Section 2.1 for two reasons:

(1) The common set of attractors can be computed in fewer iterations of Algorithm 2 for the synchronous models than that required for the asynchronous model.
(2) ROBDDs for asynchronous models are more complex than those for synchronous models. This makes all the logic operations like AND, OR, Quantify, etc. computationally demanding. Computing some of the attractors on the synchronous model should improve the computational efficiency.

Algorithm 3 details the combined synchronous–asynchronous traversal technique. In this algorithm, we first compute the synchronous attractors in Line 3. Then in Lines 4–7, the synchronous attractors that do not exist in the asynchronous model are deleted. In Line 8, the backward reachable states from the remaining attractors are computed on the asynchronous state transition diagram. These backward reachable states are removed from the state space in Line 9 (using Theorem 2) and the remaining attractors of the asynchronous

Algorithm 3 Computing asynchronous attractors

```
1  $comp\_async\_attractors(T_{sync}, T_{async})$
2  begin
3      $SS_{sync}[\,] = all\_attractors(T_{sync})$
4      for $i = 0$ to $SS_{sync}.size()$ do
5          if $isFalseLoop(SS[i], T_{sync}) == false$ then
6              $SS_{async} = SS_{async} \cup SS_{sync}[i]$
7              report $SS[i]$ as an attractor
8      $BR(SS_{async}) \longleftarrow backward\_set(SS_{async}, T_{async})$
9      $T'_{async} \longleftarrow T_{async} \wedge \overline{SS_{async} \vee BR(SS_{async})}$
10     $SS_{async}[\,] = all\_attractors(T'_{async})$
11 end
12 $isFalseLoop(S, T)$
13 begin
14     $s_0 = random\_state(S)$
15     $RS^{(0)} \longleftarrow \emptyset, FS^{(0)} \longleftarrow \{s_0\}$
16     $k \longleftarrow 0$
17     while $FS^{(k)} \neq \emptyset$ do
18         $FS^{(k+1)} = I_f(FS^{(k)})(x_{t+1} \leftarrow x_t)$
19         $nVarDiff = \sum_{i=1}^{N} FS_i^{(k)} \oplus FS_i^{(k+1)}$
20         if $nVarDiff \geq 2$ then
21             /* false asynchronous attractor */
22             return true
23         $FS^{(k+1)} = FS^{(k+1)} \wedge \overline{RS^{(k)}}$
24         $RS^{(k+1)} = RS^{(k)} \vee FS^{(k+1)}$
25         $k \longleftarrow k + 1$
26     /* genuine asynchronous attractor */
27     return false
28 end
```

model are computed in Line 10 on the reduced state space using Algorithm 2.

The function *isFalseLoop*() in lines 12–28 checks for the false synchronous attractors. In Line 14 of this function, a state s_0 is randomly selected from the states set S. Then two sets, namely the reached set RS^0 and frontier set FS^0, are defined and initialized to null and s_0, respectively. The superscript of *FS* and *RS* stands for the iteration number. Then the state reachable in one step from the current frontier set is computed in Line 18. Since we are making this computation on synchronous models, there would be only one state in the new frontier set. Then in Line 19, we compute the number of bits by which the current and the new frontier set differ. If the number of bits by which these two states differ is more than 1, then the attractor is declared false (Lines 20–22). Otherwise, the new frontier set is modified in Line 23. If the new frontier state has already been explored then this modification makes it an empty set. The new frontier set is added to the reached states set (Line 24) and the process is iterated until the frontier set is empty (Line 17). If for all the consecutive states of an attractor, the number of bits by which they differ is exactly 1, then the attractor is declared genuine (Line 27).

The results of using this combined model are listed in the last column of Table 1. The improvement of the combined model over the asynchronous model is more evident from the results on T-cell receptor and Network 1 gene regulatory networks. While processing for Network 1, the asynchronous model could not finish the computation in 1 h whereas the combined method

Algorithm 4 Algorithm for *in-silico* gene perturbation experiments

Input : Genetic Network and perturbation experiments.
Output: DAG representing steady state analysis.

```
1  begin
2      compute $T_{sync}$ and $T_{async}$
3      for $i = 0$ to $L$ do
4          compute $f_k^i$, $f_e^i$ and $f_{GP}^i$
5          $T'_{sync} = T_{sync} \wedge f_{GP}^i$
6          $T'_{async} = T_{async} \wedge f_{GP}^i$
7          $SS^i[\,] = all\_attractors(T'_{sync}, T'_{async})$
8          if $i \neq 0$ then
9              for $k = 0$ to $SS^{i-1}.size()$ do
10                 $FR \longleftarrow forward\_set(SS^{i-1}[k], T_{async})$
11                 for $j = 0$ to $SS^i.size()$ do
12                     if $SS^i[j] \subseteq FR$ then
13                         Draw an edge between nodes
                           $SS^{i-1}[k]$ and $SS^i[j]$
14 end
```

computed the attractors in 12 min and for the T-cell receptor network, the performance almost doubled. For T-helper and dendritic cell networks, the combined model takes marginally more time than the asynchronous model but this might be attributed to the fact that there is a fixed overhead involved in computing the backward reachable set in Line 8 of Algorithm 2 that is not compensated by the small run time difference between the synchronous and the asynchronous model.

3 MODELING GENE PERTURBATIONS

Computing attractors on gene regulatory networks gives an insight into the cell differentiation process. If the computed steady states (i.e. self-loop attractors) have a clear correspondance to biological cell states and other types of attractors have a biological explanation, then it is possible to be confident of the general validity of the model. In that case, it would be interesting for biologists to study the results of gene perturbation experiments on the given network. In this section, we extend our mathematical model of gene regulatory network to perform perturbation experiments. We investigate two kinds of gene perturbations:

(1) Over-expression: this represents the constant expression of a gene at a high expression level. In Boolean logic, this means that the gene is ‘ON’ or ‘1’ all the time.

(2) Knock-out: this represents the case when a gene is silenced and it does not participate in the network dynamics. That means gene is ‘OFF’ or at level ‘0’ all the time.

We modify Boolean Equations (1–3) and 5 to encode knowledge about all possible gene–perturbations in the model and then, during the analysis phase, we select the genes to be perturbed dynamically. Encoding this information in the model itself helps in sharing information between different perturbation experiments. Also, such a modeling approach permits answering questions such as what perturbations may give the desired steady states without explicitly performing all the possible gene perturbations. We use two N bits Boolean vectors **k** and **e**, called knocked out genes and over expressed genes, respectively. If a bit i of **k** (or **e**) evaluates to 1 (i.e. $k^i = 1$ or $e^i = 1$), then it means that gene i is knocked out (or over expressed). Only one of k^i and e^i can evaluate to 1 for any given i. To encode this information, we use the modified Boolean variables $\tilde{x}_i$'s as in Equation (9).

$$\tilde{x}_i = \left(x_i \vee e^i\right) \wedge \neg k^i \tag{9}$$

Equation (9) states that if gene i is over expressed (i.e. if $e^i = 1$), then $\tilde{x}_i = 1$. If gene i has been knocked out (i.e. if $k^i = 1$), then $\tilde{x}_i = 0$. If the gene is normal (i.e. $e^i = 0$ and $k^i = 0$), $\tilde{x}_i = x_i$. Now, we modify Equations (1–2) to include the perturbation information. The modified equations are given in Equations (10–11).

$$x_i(t+1) = \neg k^i \wedge \left\{ e^i \vee \left\{ \left(\bigvee_{l=1}^{m} f^a_{x_{i,l}}(t) \right) \wedge \neg \left(\bigvee_{l=1}^{n} f^{in}_{x_{i,l}}(t) \right) \right\} \right\} \tag{10}$$

$$f^{a,in}_{x_{i,l}}(t) = \left(\bigwedge_{j=1}^{p} \tilde{x}^a_j(t) \right) \wedge \left(\bigwedge_{j=1}^{q} \neg \tilde{x}^{in}_j(t) \right) \tag{11}$$

k^j *is* true if gene j is knocked out

e^j *is* true if gene j is over expressed

Equation (10) states that if gene i is over expressed (i.e. if $e^i = 1$), then $x_i(t+1) = 1$. If gene i has been knocked out (i.e. if $k^i = 1$), then $x_i(t+1) = 0$. If the gene is normal (i.e. $e^i = 0$ and $k^i = 0$), Equation (10) is same as Equation (1). Equation (11) is exactly the same as Equation (2), with modified variables $\tilde{x}_i$.

The transition function for synchronous models given in Equation (3) is modified to Equation (12) and asynchronous models in Equation (5) is modified to Equation (13).

$$T_i(\mathbf{x}_t, \mathbf{x}_{t+1}) = x^i_{t+1} \leftrightarrow \left\{ \neg k^i \wedge \left(e^i \vee x_i(t+1) \right) \right\} \tag{12}$$

$$TP_i(\mathbf{x}_t, \mathbf{x}_{t+1}) = \left(x^i_{t+1} \leftrightarrow \left\{ \neg k^i \wedge \left(e^i \vee x_i(t+1) \right) \right\} \right) \wedge \bigwedge_{j \neq i} \left\{ x^j_{t+1} \leftrightarrow \left\{ \neg k^j \wedge \left(e^j \vee x^j_t \right) \right\} \right\} \tag{13}$$

The rest of the equations for synchronous and asynchronous models remain the same as in Equation (4) and Equations (6–8), respectively.

Equations (9–13) along with Equations (4, 6–8) represent the state space with all possible gene perturbations. Given a perturbation experiment, we restrict the state space to only those perturbations which are part of the experiment and compute attractors on that restricted state space. For this, we define three Boolean functions f_k, f_e and f_{GP} to represent information of the knocked out, over expressed and perturbed genes, respectively. f_{GP} is further expressed as a function of f_k and f_e. These functions are given in Equations (14–16).

$$f_{GP} = f_k \wedge f_e \tag{14}$$

$$f_k = \left(\bigwedge_{i:k^i=1} k^i \right) \wedge \left(\bigwedge_{i:k^i=0} \neg k^i \right) \tag{15}$$

$$f_e = \left(\bigwedge_{i:e^i=1} e^i \right) \wedge \left(\bigwedge_{i:e^i=0} \neg e^i \right) \tag{16}$$

A perturbation experiment can have multiple levels (or orders) of perturbations. Each level i has a number of genes which are perturbed synchronously. These perturbed genes are encoded into f^i_k, f^i_e and f^i_{GP} using Equations (14–16), where i is the level of the perturbation experiment. Perturbations in consecutive levels, starting from level 1 and going down to L levels, are done sequentially. Level 0 always represents the original network without any perturbation.

The idea is formally described in Algorithm 4. In this algorithm, the main loop in Lines 3–13 is iterated over all levels L. For every level i, corresponding functions f_e, f_k and f_{GP} are constructed using Equations (14–16) in Line 4. Then the transition function is restricted to the state space defined by this perturbation experiment (Lines 5, 6). The attractors are computed on the perturbed network in Line 7 using the combined synchronous–asynchronous method. Once the attractors are found, we compare the forward reachability of attractors of the previous level with the attractors of the current level of a perturbation experiment. This is done in Lines 8–13. For every attractor computed in the previous level, i.e. level $i-1$, we compute the forward reachable states on the new transition function (Line 10). Then we check all the attractors in the current level i that are contained in this forward reachable set (Line 12). Lines 3–13 can be repeated for different experiments on the same network without constructing a new model of the regulatory network.

4 MODELING THE T-HELPER NETWORK

We implemented our methodology in the software package `genYsis` (http://si2.epfl.ch/∼garg/genysis.html) and tested it in the study of the signaling network (Fig. 5) that controls the differentiation of T-helper (Th) cells (Mendoza and Xenarios, 2006). Th cells have three main phenotypes: the precursor Th0, and the effector Th1 and Th2, which secrete IFN-γ and IL-4, respectively. The differentiation from Th0 to either Th1 or Th2 phenotypes depends on the integration of diverse molecular and cellular clues. It has been shown that the Th network is sufficient to describe qualitatively the differentiation of Th cells, studied by modeling it as a discrete dynamical system (Mendoza, 2006), a continuous dynamical system (Mendoza and Xenarios, 2006), a Petri net (Remy *et al*., 2006), and a binary decision diagram (Garg *et al*., 2007).

There is a wealth of experimental data on the effect of cytokines on the differentiation of Th cells (Agnello *et al*., 2003; Bergmann and van Hemmen, 2001), and we tested our algorithm by simulating some experimental treatments. First, it was necessary to demonstrate that our algorithm is able to correctly identify the attractors of the wild-type (unperturbed) Th network. Table 2 shows that our methodology correctly identifies the three stable steady states on

Table 2. Steady states of the Th Cell

Perturbed genes	Active genes in steady states								Cell type
	All the genes are inactive								Th0
wild type	IFN-γ	Tbet	SOCS-1	IFN-γR					Th1
	IL-10	IL-10R	GATA-3	STAT3	STAT6	IL-4	IL-4R		Th2
IL-12 over expressed	IFN-γ	Tbet	SOCS-1	IFN-γR	IL-12	IL-12R	STAT4		Th1
	IL-10	IL-10R	GATA-3	STAT3	IL-12	STAT6	IL-4	IL-4R	Th2
IL-4 over expressed	IFN-γ	Tbet	SOCS-1	IFN-γR	IL-4				Th1
	IL-10	IL-10R	GATA-3	STAT3	STAT6	IL-4	IL-4R		Th2

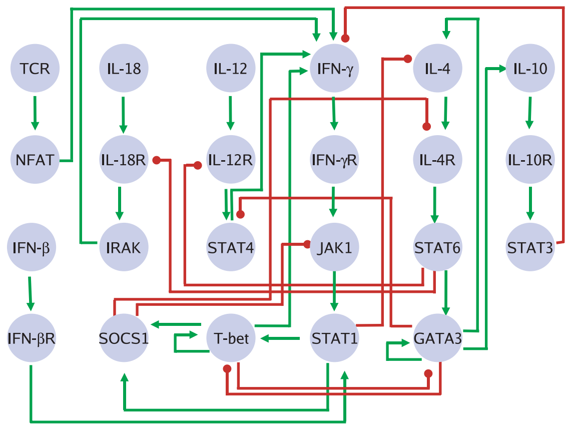

Fig. 5. Th network. The regulatory network that controls the differentiation process of Th cells. Positive regulatory interactions are shown with a pointed arrow head and negative interactions with a round arrow head.

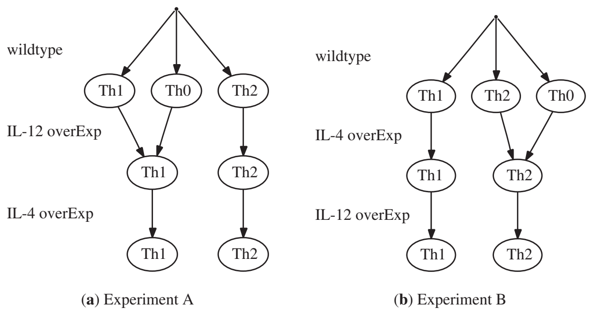

Fig. 6. Results of gene perturbation experiments.

the Th network, which represent the activation patterns observed in Th0, Th1 and Th2 cells.

Next, we simulated the effect of subjecting Th cells to two consecutive stimuli, first a constant saturating concentration of IL-12, and then changing it to a saturating concentration of IL-4. As shown in Figure 6a, this combination of signals has the result of eliminating the Th0 steady state (or self loop attractor). If the system is in the Th0 state, the constant activation of IL-12 moves it to the Th1 state, where it stays even after the inactivation of IL-12 and the constant presence of IL-4. In contrast, if the system starts in the Th1 or Th2 states, the two consecutive signals are incapable of moving the system to another attractor. The steady state profile on IL-12 and IL-4 over expression are shown in Table 2.

Finally, we did the simulation of the same perturbations as described above, but in reverse order. That is, activating IL-4 to its highest level, and then inactivating it and activating IL-12 instead. Results are the same as before, shown in Figure 6b. The only difference was that in this simulation, the network in the Th0 states receives the IL-4 and moves to Th2, where it stays after the elimination of IL-4 and the activation of IL-12.

These simulations show that Th0 state is unstable under the perturbation of IL-12 or IL-4, which act as differentiation signals to take the system to the Th1 or Th2 states, respectively. In contrast, the Th1 and Th2 states are stable under the perturbation of the IL-4 and IL-12 nodes. These results are in total agreement with experimental data (Murphy and Reiner, 2002) and reported simulations of the Th network using a different mathematical framework (Mendoza and Xenarios, 2006).

5 CONCLUSION

In this article, we have addressed the computational issues of different dynamic models for gene regulatory networks. We have shown, using our software `genYsis` on networks of varying sizes, that asynchronous modeling of even moderately sized gene regulatory networks can be computationally impossible to perform in a decent run time whereas synchronous behaviour of the same networks can be modeled in a few minutes. We introduced a combined synchronous–asynchronous modeling approach that can represent the asynchronous behaviour of regulatory networks in run time proportional to the synchronous models. Using this efficient modeling approach, we then introduced the framework for performing perturbation experiments on gene regulatory networks. We have shown the application of our software on the Th regulatory network. In contrast to existing softwares (e.g. Gene Network Analyzer, Cell Net Analyzer, GinSim, etc.) for similar applications, `genYsis` can identify all kinds of attractors that may exist in the gene regulatory networks. The combined synchronous–asynchronous algorithm has been integrated in the latest version of the simulation package SQUAD (Di Cara *et al.*, 2007). The methodology for gene perturbation analysis will be incorporated into SQUAD in its next release. In the meantime, `genYsis` is freely available as a standalone package on http://si2.epfl.ch/∼garg/genysis.html.

ACKNOWLEDGEMENTS

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

Conflict of Interest: none declared.

REFERENCES

Albert,R. and Othmer,H.G. (2003) The topology of the regulatory interactions predicts the expression pattern of the Drosophila segment polarity genes. *J. Theor. Biol.*, **223**, 1–18.

Agnello,D. *et al*. (2003) Cytokines and transcription factors that regulate T helper cell differentiation: new players and new insights. *J. Clin. Immunol.*, **23**, 147–162.

Baldamus,M. and Schneider,K. (2001) The BDD Space Complexity of Different Forms of Concurrency. In *Proceedings of ICACSD '01*. Newcastle upon Tyne, UK, pp. 231–242.

Bergmann,C. and van Hemmen,J.L. (2001) Th1 or Th2: how an appropriate T helper response can be made. *Bull. Math. Biol.*, **63**, 405–430.

Bernot,G. *et al*. (2004) Application of formal methods to biological regulatory networks: extending Thomas' asynchronous logical approach with temporal logic. *J. Theor. Biol.*, **229**, 339–347.

Bryant,R.E. (1986) Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput.*, **35**, 677–691.

Burch,J.R. *et al*. (1991) Symbolic model checking with partitioned transition relations. In *Proceedings of International Conference on VLSI'91*. Edinburgh, Scotland, pp. 49–58.

Burch,J.R. *et al*. (1994) Symbolic model checking for sequential circuit verification. *IEEE Comput-Aid D*, **13**.

Chabrier,N. *et al*. (2004) BIOCHAM. In *Proceeding of Computational Method of Systems Biology*. Paris, France, pp. 172–191.

Devloo,V. *et al*. (2003) Identification of all steady states in large biological systems by logical analysis. *Bull. Math. Biol.*, **65**, 1025–1051.

Di Cara, *et al*. (2007) Dynamic simulation of regulatory networks using SQUAD. *BMC Bioinformatics*, **8**, 462.

Fauré,A. *et al*. (2006) Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle. *Bioinformatics*, **22**, e124–e131.

Garg,A. *et al*. (2007) An efficient method for dynamic analysis of gene regulatory networks. *Springer LNBI*, **4453** , 62–76.

Kauffman,S.A. (1969) Metabolic stability and epigenesis in randomly constructed genetic nets. *J. Theor. Biol.*, **22**, 437–467.

Klamt,S. *et al*. (2006) A methodology for the structural and functional analysis of signaling and regulatory networks. *BMC Bioinformatics*, **7**, 56.

Murphy,K.M. and Reiner,S.L. (2002) The lineage decisions on helper T cells. *Nat. Rev. Immunol.*, **2**, 933–944.

Mendoza,L. and Xenarios,I. (2006) A method for the generation of standardized qualitative dynamical systems of regulatory networks. *Theor. Biol. Med. Model.*, **3**, 13.

Mendoza,L. (2006) A network model for the control of the differentiation process in Th cells. BioSystems, **84**, 101–114.

Naldi,A. *et al*. (2007) Decision diagrams for the representation and analysis of logical models of genetic networks. *LNCS/LNBI*, **4695**, 233–247.

Remy,E. *et al*. (2006) From logical regulatory graphs to standard petri nets: dynamical roles and functionality of feedback circuits. *Springer LNCS*, **4230**, 56–72.

Thomas,R. (1991) Regulatory networks seen as asynchronous automata: a logical description. *J. Theor. Biol.*, **153**, 1–23.

Thomas,R. and Kaufman,M. (1995) 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. Biol.*, **57**, 247–276.

Xie,A. and Beerel,P.A. (1998) Efficient state classification of Finite State Markov Chains. In *Proceedings of DAC*. San Francisco, CA, USA, pp. 605–610.