Theoretical Computer Science 594 (2015) 151–179

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Qualitative analysis of gene regulatory networks by temporal logic

Sohei Ito[a,*], Takuma Ichinose, Masaya Shimakawa[b], Naoko Izumi[c], Shigeki Hagihara[b], Naoki Yonezaki[d]

[a] *National Fisheries University, 2-7-1 Nagata-Honmachi, Shimonoseki, Yamaguchi 759-6595, Japan*
[b] *Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro-ku, Tokyo 152-8552, Japan*
[c] *Jumonji University, 2-1-28 Sugasawa, Niiza, Saitama 352-8510, Japan*
[d] *The Open University of Japan, 2-11 Wakaba, Mihama-ku, Chiba City, Chiba 261-8586, Japan*

A R T I C L E I N F O

Article history:
Received 14 November 2014
Received in revised form 20 May 2015
Accepted 5 June 2015
Available online 11 June 2015
Communicated by P. Aziz Abdulla

Keywords:
Systems biology
Gene regulatory networks
Temporal logic
Reactive system verification

A B S T R A C T

In this article we propose a novel formalism to model and analyse gene regulatory networks using a well-established formal verification technique. We model the possible behaviours of networks by logical formulae in linear temporal logic (LTL). By checking the satisfiability of LTL, it is possible to check whether some or all behaviours satisfy a given biological property, which is difficult in quantitative analyses such as the ordinary differential equation approach. Owing to the complexity of LTL satisfiability checking, analysis of large networks is generally intractable in this method. To mitigate this computational difficulty, we developed two methods. One is a modular checking method where we divide a network into subnetworks, check them individually, and then integrate them. The other is an approximate analysis method in which we specify behaviours in simpler formulae which compress or expand the possible behaviours of networks. In the approximate method, we focused on network motifs and presented approximate specifications for them. We confirmed by experiments that both methods improved the analysis of large networks.

1. Introduction

One of the difficulties in analysing biological systems is the incompleteness of quantitative biological information, which hinders the mathematical analysis based on traditional thermodynamics models. In this respect, qualitative methods based on formal verification techniques have gathered good attention and several methods have been proposed with several different formalisms [14,17,9,8,42].

In this article, we present a new constraint-based method for modelling and analysing gene regulatory networks (or simply gene networks). This method is based on the paradigm of verification of reactive system specifications [35,49,24]. Reactive system specification stipulates how the system should or can behave over time. There are several properties of reactive system specifications to be verified such as satisfiability, strong satisfiability and realisability [37,1,35].

* Corresponding author at: Department of Fisheries Distribution and Management, National Fisheries University, 2-7-1 Nagata-Honmachi, Shimonoseki, Yamaguchi 759-6595, Japan.
E-mail address: ito@fish-u.ac.jp (S. Ito).

http://dx.doi.org/10.1016/j.tcs.2015.06.017

We found that dynamic behaviours of gene networks can also be qualitatively characterised similarly to the case of reactive systems, under the suitable abstraction. In our method, behaviours are abstracted to transition systems. A state in a transition system represents a configuration (state of affair) of a gene network at a certain time point, such as the expression levels of each gene and whether each gene is expressed or not. A state can be seen as a set of such facts that are true at a certain time point. Thus a transition system represents how the state of a gene network changes over time. In other words, a transition system represents a dynamic behaviour of a gene network. A fact of a state can be mathematically represented as a proposition. Then the problem of modelling possible behaviours of a gene network is reduced to give a suitable specification using these propositions. The possible behaviours of a gene network are transition systems which satisfy the specification. We show that such specification can be systematically obtained from a gene network using linear temporal logic (LTL) [16]. This corresponds to give a constraint that the 'correct' behaviour of a gene network should satisfy. Expected biological properties such as reachability, stability, oscillation or any other temporal properties about the timing of gene expressions are also described in LTL. The problem of analysing a property of a gene network is reduced to check the *satisfiability* of these formulae. Our first contribution is this conceptual framework of modelling behaviours of gene regulatory networks as LTL formulae and analyse biological properties by LTL-satisfiability checking.

The complexity of LTL-satisfiability checking is PSPACE-complete [44], and known algorithms have exponential time complexity with respect to the length of an input formula. The length of a formula specifying possible behaviours of a network is proportional to the size of the network in our method. Therefore, some methods are strongly desirable which eases this computational cost. As such methods, we develop two methods – a modular method and an approximate method. These are the second and the third contributions in this article.

In modular analysis method, a network is divided into several subnetworks. Since each specification of subnetworks is much smaller than that of the entire network, computation of the possible behaviours of each subnetwork is much faster. Furthermore, we can abstract local propositions (only used in the subnetwork) from each subnetwork behaviours. Then we integrate the possible behaviours of the subnetworks to obtain the behaviours of the entire network.

In approximate analysis method, we specify the set of possible behaviours of (sub)networks without using some local propositions. Such specifications are simpler than the original specifications but are generally not equivalent, that is to say, they are approximate specifications of networks. There are two kinds of approximation. One is under-approximation in which possible behaviours are compressed, and the other is over-approximation in which possible behaviours are expanded. They can be used instead of the original specifications to check network properties, that is, if the approximate specifications are satisfiable/unsatisfiable then so are the original ones. As a result, the correctness of approximate analysis is theoretically guaranteed.

It is not trivial to find approximate specifications for any networks. Therefore we consider some 'templates' of gene networks and give approximate specifications for them. As templates of gene networks, we can use network motifs [3], since they are network patterns that occur in many gene regulatory networks. The motifs we study in this article are negative auto-regulation, coherent type 1 feed-forward loops, incoherent type 1 feed-forward loops, single-input modules and multi-output feed-forward loops.

This article is organised as follows. Section 2 introduces the logical structure which describes abstract behaviours of gene regulatory networks. In Section 3, we show how networks are qualitatively modelled and analysed by the LTL satisfiability checking. Then we demonstrate our method by analysing a network of circadian clock and a network for mucus production in *Pseudomonas aeruginosa*. In Section 4, we introduce the modular analysis method and prove the correctness of it. We demonstrate our modular method using example networks and discuss the results. In Section 5, we present the approximate analysis method and introduce approximate specifications for network motifs. We discuss experimental results of the approximate method. Furthermore, the experimental result of the combination of the modular method and the approximate method is reported. In Section 6, we compare our method to other qualitative analysis methods of biological systems. The final section offers some conclusions and discusses future directions.

This article is a revised and extended version of our conference papers: [29,28,27]. We revised the modelling method, presented the formal proofs of our methods and added new experiments.

2. Logical conceptualisation of network behaviours

A gene is a certain segment of DNA which encodes proteins. Proteins play essential roles in living organisms. Suitable concentration of each protein must be maintained for each cell to function properly. Proteins are known to exist a certain period of time and are eventually degraded. To maintain suitable concentration, proteins are produced through the process of *gene expression*. In the process of gene expression, genes are transcribed into messenger RNAs (mRNAs) by RNA polymerase. Then the mRNAs are transferred to ribosomes. Finally the ribosomes translate the mRNAs to proteins according to genetic code. Gene expression is regulated by proteins called *transcription factors*. Transcription factors bind to *promoter* regions of genes which are located on upstream regions of the genes and promote or block the recruitment of RNA polymerase (Fig. 1). Thus transcription factors regulate gene expression by changing the rate of transcription. There are two types of regulation – activation and inhibition. Activation means that the transcription factor increases the rate of gene expression. Inhibition is the opposite of activation – it decreases the rate of gene expression. Since transcription factors are proteins, they are also coded by some genes. Thus if a gene whose product is a transcription factor of another gene is expressed, that gene *regulates* another gene. Some transcription factors work with other transcription factors or proteins called co-regulators to

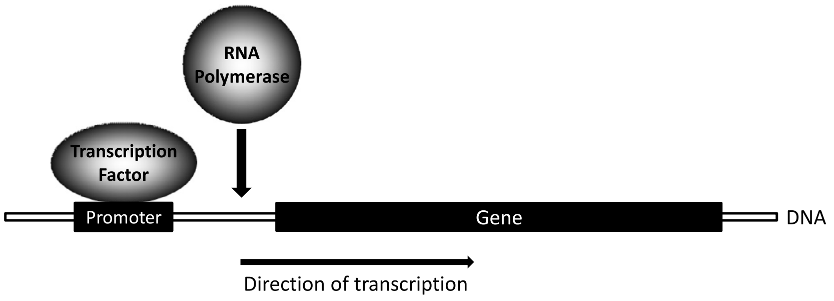

Fig. 1. Gene is a segment of DNA which encodes proteins. A promoter of a gene is an upstream region of the gene. Transcription factor binds to the promoter and regulates gene expression.

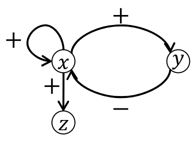

Fig. 2. An example of a gene network. Nodes x, y and z represent genes and edges represent regulation relation among them. Plus-edges represent activation relationship and minus-edges represent inhibition relationship.

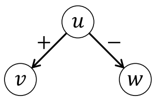

Fig. 3. Gene u activates v and inhibits w.

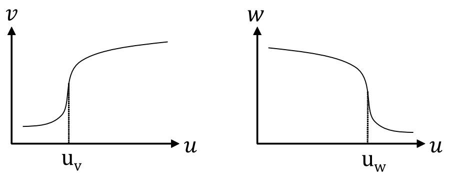

Fig. 4. Regulation effect.

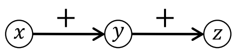

Fig. 5. An example network.

regulate gene expression. Such regulation relations among genes are graphically represented as *gene regulatory networks* (or gene networks). An example of a gene network is given in Fig. 2.

In gene regulation, a regulator is often inefficient below a threshold concentration, and its effect rapidly increases above this threshold [47]. The sigmoid nature of gene regulation is shown in Fig. 4, where gene u activates v and inhibits w (Fig. 3). Each axis represents the concentration of products for each gene.

Some important landmark concentration values for u are 1) the basal level,[1] 2) the level $\mathrm{u_v}$ at which u begins to affect v, and 3) the level $\mathrm{u_w}$ at which u begins to affect w. In this case, whether genes are active or not can be specified by the expression levels of their regulator genes. If the concentration of u exceeds $\mathrm{u_v}$ then v is active (ON), and if the concentration of u exceeds $\mathrm{u_w}$ then w is not active (OFF). We exploit this switching view of genes to capture behaviours in transition systems.

We now illustrate how we capture behaviours of gene regulatory networks as transition systems using a simple example network (Fig. 5) in which gene x activates gene y and gene y activates gene z.

Let the threshold of x for y be $\mathrm{x_y}$ and that of y for z be $\mathrm{y_z}$. We consider the behaviour depicted in Fig. 6 and try to express it as a transition system. In this behaviour, x begins to be expressed at time t_0; that is, the concentration of its products begins to increase. At time t_1, the concentration of the products of x exceeds $\mathrm{x_y}$, which is the threshold for the activation of y. Thus y begins to be expressed at t_1. At time t_2, x stops being expressed and the concentration of its products begins to decrease. At time t_3, the concentration of products of x falls below $\mathrm{x_y}$ and y stops being expressed; that

[1] A gene is not expressed or expressed at very low rate.

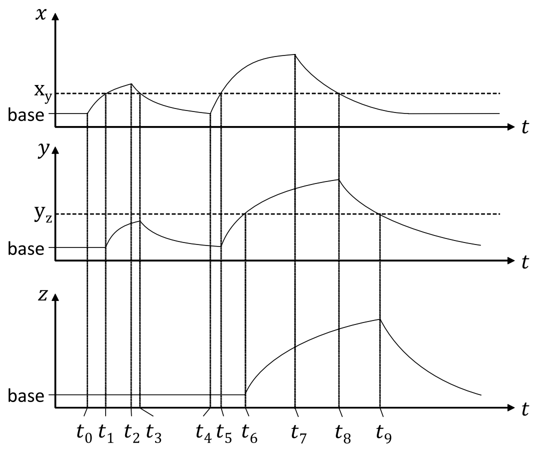

Fig. 6. Change of concentrations over time.

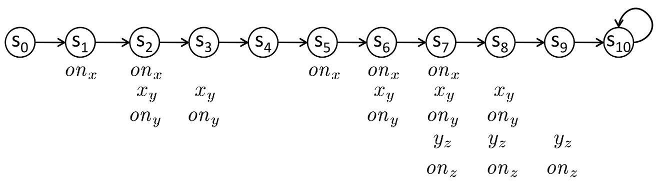

Fig. 7. State transition system corresponding to Fig. 6.

is, the concentration of y begins to decrease. After a while, x begins to be expressed again at time t_4 and y begins to be expressed at t_5. In this case, y crosses the activation threshold for z at time t_6 and z begins to be expressed. At t_7, x stops being expressed and begins to decrease. At t_8, x falls below $\mathrm{x_y}$ and y stops being expressed. At t_9, y falls below $\mathrm{y_z}$ and z stops being expressed, after which x, y and z stay at their basal levels.

We introduce some logical propositions to obtain a symbolic representation of behaviours of this network. Based on the above observation, we introduce propositions that represent whether genes are active or not (ON or OFF) and whether concentrations of products of genes exceed their threshold values. In this network, we introduce the propositions on_x, on_y, on_z, x_y and y_z. The meaning of each proposition is:

- on_x, on_y, on_z: whether gene x, y and z is active.
- x_y: whether the concentration of the products of x *exceeds* the threshold $\mathrm{x_y}$.
- y_z: whether the concentration of the products of y *exceeds* the threshold $\mathrm{y_z}$.

Notation. Threshold values appear in roman and propositions corresponding to the thresholds in italics.

Using these propositions, we discretise the above behaviour to the sequence of states (called *transition system*) shown in Fig. 7, where $s_0, \dots, s_{10}$ are states, edges represent state transitions that abstract the temporal evolution of the system, and the propositions below each state mean that they are true in that state.

State s_0 represents the interval $[0, t_0)$, state s_1 represents the interval $[t_0, t_1)$, ...and state s_{10} represents $[t_9, \infty)$.

A single state transition can represent any length of time, since the actual duration of the transition (in real time) is immaterial[2] in this abstraction. Therefore, the difference between $t_2 - t_0$ and $t_7 - t_4$, the durations of the input signal to x in Fig. 6, are not captured directly in the transition system of Fig. 7. Only the order of events is important.

In this abstraction, the real values of thresholds are also irrelevant. Propositions such as x_y merely represent the fact that the concentration of x is above the level at which x activates y.

In our abstraction, we think that behaviours are identical if they have the same transition system. Such logical abstraction preserves essential qualitative features of the dynamics such as oscillation, steady states, multistationarity, and reachability for such states [18,19,32]. However, we cannot reason about quantitative properties, such as rate of production/degradation of products, real values of concentrations, real-time durations, stability of oscillation and so on.

[2] This property is called *speed independence* [39].

3. Qualitative analysis of gene regulatory networks in LTL

In this section, we show how to model and analyse behaviours of gene regulatory networks using LTL, based on the conceptualisation of gene network and its behaviours introduced in the previous section.

3.1. Linear temporal logic

First we introduce Linear Temporal Logic (LTL) as our modelling language. LTL is a suitable language for describing temporal evolution of systems and is used in software/hardware system verifications [12]. In LTL we can specify various temporal properties on linear time structures like Fig. 7. In what follows, we present a formal definition of LTL.

First we introduce the time structure of LTL. If A is a finite set, A^ω denotes the set of all infinite sequences on A. The i-th element of $\sigma \in A^\omega$ is denoted by $\sigma[i]$.

Definition 1. Let AP be a set of atomic propositions. A *time structure* on AP is a sequence $\sigma \in \mathfrak{P}(AP)^\omega$ where $\mathfrak{P}(AP)$ is the powerset of AP.

We next define the syntax of LTL formulae.

Definition 2. Let AP be a set of atomic propositions. Proposition $p \in AP$ is a formula of LTL. If ϕ and ψ are formulae, then $\neg\phi, \phi \wedge \psi$ and $\phi U \psi$ are also formulae of LTL.

The formal semantics are given below.

Definition 3. Let σ be a time structure and ϕ be a formula of LTL. We write $\sigma \models \phi$ for 'ϕ is true in σ'. The satisfaction relation $\models$ is defined inductively as follows.

$$\begin{array}{lll}
\sigma \models p & \text{iff} & p \in \sigma[0] \text{ for } p \in AP \\
\sigma \models \neg\phi & \text{iff} & \sigma \not\models \phi \\
\sigma \models \phi \wedge \psi & \text{iff} & \sigma \models \phi \text{ and } \sigma \models \psi \\
\sigma \models \phi U \psi & \text{iff} & (\exists i \geq 0)(\sigma^i \models \psi \text{ and } \forall j (0 \leq j < i)\sigma^j \models \phi)
\end{array}$$

where $\sigma^i = \sigma[i]\sigma[i+1]\ldots$, the i-th suffix of σ.

We introduce the following abbreviations: $\bot \equiv p \wedge \neg p$ for some $p \in AP$, $\top \equiv \neg\bot$, $\phi \vee \psi \equiv \neg(\neg\phi \wedge \neg\psi)$, $\phi \rightarrow \psi \equiv \neg\phi \vee \psi$, $\phi \leftrightarrow \psi \equiv (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$, $F\phi \equiv \top U \phi$, $G\phi \equiv \neg F \neg\phi$, and $\phi W \psi \equiv (\phi U \psi) \vee G\phi$, where ϕ and ψ are LTL formulae.

Intuitively, $F\phi$ means 'ϕ holds at some future time', $G\phi$ means 'ϕ holds globally' and $\phi U \psi$ means 'ϕ continues to hold until ψ holds'. $\phi W \psi$ is the 'weak until' operator in that ψ may not hold, in that case ϕ must always hold.

Finally we introduce the notion of *satisfiability*.

Definition 4. An LTL formula ϕ is *satisfiable* if there exists a time structure σ such that $\sigma \models \phi$. We say that σ is a *model* of ϕ if $\sigma \models \phi$.

3.2. Analysis of gene regulatory networks by satisfiability checking in LTL

As we can see in Section 2, a behaviour of a gene regulatory network can be seen as a time structure on atomic propositions. Let AP be the set of atomic propositions for describing states of a given network. Formally, a behaviour of a given network is an element of $\mathfrak{P}(AP)^\omega$. However, not all of the sequences in $\mathfrak{P}(AP)^\omega$ are possible behaviours of a given network. For example, in the network of Fig. 5, y cannot be ON before x becomes ON if y totally depends on x. Thus, possible behaviours of a network corresponds to a subset of $\mathfrak{P}(AP)^\omega$.

We are interested in analysing whether a given network behaves as expected or not. To answer this, we need to characterise possible behaviours of a network. In quantitative analysis, behaviours are usually described as ordinary differential equations. We, however, characterise the possible behaviours of a network in LTL based on the logical conceptualisation of network behaviours. A formula which characterises possible behaviours of a network is considered as a constraint which the possible behaviours of a network should satisfy. Suppose formula ϕ is the behaviour description of a given network, then the set of models of ϕ, $\{\sigma \mid \sigma \models \phi\}$, are the possible behaviours of the network.

Now the problem of checking whether the behaviours of a given network satisfy a biological property is formulated in terms of LTL. We specify the given biological property as an LTL formula ψ. The first type of analysis is to check whether there is a behaviour of the network which satisfies a given biological property ψ. This problem is reduced to checking the satisfiability of the formula $\phi \wedge \psi$, since

$$\begin{aligned} &\exists\sigma.\sigma \models \phi \text{ and } \sigma \models \psi \\ \Leftrightarrow\ &\exists\sigma.\sigma \models \phi \wedge \psi. \end{aligned}$$

This means that there exists a sequence σ which is a possible behaviour of the network (i.e. satisfies ϕ) and satisfies the biological property ψ. The second type of analysis is to check whether all possible behaviours of the network satisfy a given biological property ψ. This problem is reduced to checking the unsatisfiability of $\phi \wedge \neg\psi$, since

$$\begin{aligned} &\forall\sigma.\sigma \models \phi \text{ implies } \sigma \models \psi \\ \Leftrightarrow\ &\forall\sigma.\sigma \models \phi \rightarrow \psi \\ \Leftrightarrow\ &\forall\sigma.\sigma \models \neg\phi \vee \psi \\ \Leftrightarrow\ &\forall\sigma.\sigma \not\models \phi \wedge \neg\psi. \end{aligned}$$

This means if a sequence σ is possible in the network, then it necessarily satisfies the given biological property ψ.

The scheme for qualitative analysis of behaviours of a given network is summed up as follows:

1. Specify a formula ϕ that characterises possible behaviours of the network.
2. Specify a formula ψ that represents a biological property of interest.
3. Check the satisfiability of $\phi \wedge \psi$ or unsatisfiability of $\phi \wedge \neg\psi$.

Then we have the questions: how do we specify such behaviour specifications ϕ and what are properties ψ? We answer these questions in the subsequent sections.

3.3. Specification of behaviours in LTL

We show how we specify a characterisation of the possible behaviours of a given network in LTL. As in Section 2, we use the following propositions to specify it:

- on_u for each gene u in a given network. We interpret on_u as 'gene u is *active*'.
- u_v for each regulation from u to v in a given network. We interpret u_v as 'gene u is expressed *beyond* the threshold to activate/inhibit v'.

Additionally, we may introduce other propositions for each landmark concentration value of gene u that is not a threshold for any other genes (say, u_{low}, u_{high} and so on), or may introduce several thresholds for the same regulation (e.g. the lower threshold and the higher one).

The idea of specifying possible behaviours of a network is based on the following qualitative principle:

- Genes are ON when their activators are expressed over some threshold.
- Genes are OFF when their inhibitors are expressed over some threshold.
- If genes are ON, the concentrations of their products increase.
- If genes are OFF, the concentrations of their products decrease.

Thus we specify the above principles in LTL using the propositions introduced earlier. The switching conditions for gene u can be specified by its regulators (say, $x, y, \ldots$) using propositions ($x_u, y_u, \ldots$) corresponding to their threshold values. The concentration increase or decrease for gene u can be specified by its threshold values that u has. For this, the total order of threshold values must be fixed.

Now we show how to specify the above principles in LTL.

Conditions for gene activation and inhibition First we consider the simple case in which a gene is regulated by a single gene. For example, let gene v be regulated only by u. If the effect of u on v is positive, then v is turned ON when the concentration of u exceeds the threshold $\mathrm{u_v}$. We have two choices to describe this phenomenon in LTL. One is

$$G(u_v \rightarrow on_v)$$

and the other is

$$G(u_v \leftrightarrow on_v).$$

The former allows on_v to be true when u_v is not, but the latter does not. The former specification takes hidden activators or external regulation for v into account. In this case we do not consider hidden negative regulation to v since if we consider it, the relationship 'gene u activates v' is lost. The choice of which one we use depends on the system, the situation or the assumption.

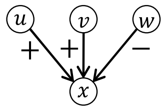

Fig. 8. Gene u, v activates x and gene w inhibits x.

On the other hand, if the effect of u on v is negative, this case is described as:

$$G(u_v \rightarrow \neg on_v)$$

Similarly we can choose $G(u_v \leftrightarrow \neg on_v)$ for the same reason.

Now we consider a gene that is regulated by multiple genes. In general, the multivariate regulation functions of organisms are unknown [3]. Thus we only describe the trivial facts. For example, we assume that genes u and v activate x, and gene w inhibits x (Fig. 8). Then we have the following facts:

- If u and v exceed $\mathrm{u_x}$ and $\mathrm{v_x}$ respectively, and w does not exceed $\mathrm{w_x}$, then x is ON. This is described as follows:

$$G((u_x \wedge v_x \wedge \neg w_x) \rightarrow on_x).$$

- If u and v do not exceed $\mathrm{u_x}$ and $\mathrm{v_x}$ respectively, and w exceeds $\mathrm{w_x}$, then x is OFF. This is described as follows:

$$G((\neg u_x \wedge \neg v_x \wedge w_x) \rightarrow \neg on_x).$$

Note that if we take contrapositive of the second formula, we have

$$G(on_x \rightarrow (u_x \vee v_x \vee \neg w_x)).$$

Thus when gene x is ON, it is possible that both gene u and v are not effective. This means we do not exclude the possibility of hidden positive regulation to gene x. If we are to exclude it, we add the clause

$$G(on_x \rightarrow (u_x \vee v_x)).$$

For the same reason if we exclude the possibility of hidden negative regulation to gene x, we add the clause

$$G(\neg on_x \rightarrow w_x).$$

If we know more information about the multivariate regulation function of gene x, we can reflect such facts in LTL specification. For example, if we know that the positive effect of u and v on x is disjunctive, we have

$$G(((u_x \vee v_x) \wedge \neg w_x) \rightarrow on_x)$$

for the condition of when gene x is ON. Or, if we know that the negative regulation effect of w is dominant and overpowers other positive effects, we have

$$G(w_x \rightarrow \neg on_x)$$

for the condition of when gene x is OFF.

We can introduce multiple expression levels for a single regulation relation. For example, in the relation 'gene w inhibits gene x' in Fig. 8, we may introduce two expression levels: $\mathrm{w_x^0}$ and $\mathrm{w_x^1}$ ($\mathrm{w_x^0} < \mathrm{w_x^1}$). These two levels of gene w represent the difference of power of inhibition to gene x. The level $\mathrm{w_x^0}$ is a low expression level above which gene x is OFF if both gene u and v are not effective. The level $\mathrm{w_x^1}$ is high expression level of gene w above which the negative effect of w overpowers both gene u and v. This can be described as follows:

$$G((\neg u_x \wedge \neg v_x \wedge w_x^0) \rightarrow \neg on_x),$$

$$G(w_x^1 \rightarrow \neg on_x).$$

Of course we can introduce more expression levels for gene w to finely capture the multiple regulation.

In gene regulation, some genes regulate not genes but the regulation effect itself. For example some gene's product intercepts another gene's product, which causes the inhibition of the latter gene's regulation effect. Let us consider a case where x inhibits y and z inhibits the regulation effect of x on y. In this case y is turned OFF when x affects y but z does not affect the regulation. To describe this, we introduce a threshold $\mathrm{z_x}$ above which z inhibits the effect of x (Fig. 9). We can describe this as follows:

$$G((x_y \wedge \neg z_x) \rightarrow \neg on_y).$$

In this case, $\mathrm{z_x}$ may not be a fixed value but should rather be considered as a function that takes the concentration of x and returns the threshold of z. The proposition z_x simply says that z influences the regulation effect of x and the real value of the concentration of z does not matter.

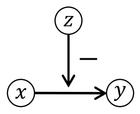

Fig. 9. Gene x inhibits y. Gene z inhibits the effect of x on y.

Total order of threshold values Before introducing the description of concentration changes of gene products, we need to specify the fixed total order of threshold values. Since the concentration changes are described as 'if a gene is ON, the concentration level will reach at the next level'. To specify this, we need to articulate what is the 'next' level in LTL. Assume that u regulates $x_1, x_2, \dots, x_m$ and the threshold values for them are in this ascending order. This order relation can be described in LTL as follows:

$$\bigwedge_{1\le i<m} G(u_{x_{i+1}} \to u_{x_i}).$$

For example, $G(u_{x_2} \to u_{x_1})$ means that if the current expression level is beyond the threshold $\mathrm{u}_{\mathrm{x}_2}$, it is also beyond $\mathrm{u}_{\mathrm{x}_1}$ since $\mathrm{u}_{\mathrm{x}_1}$ is lower than $\mathrm{u}_{\mathrm{x}_2}$. Note that the propositions u_{x_1} and u_{x_2} are interpreted as gene u is expressed *beyond* the threshold $\mathrm{u}_{\mathrm{x}_1}$ and $\mathrm{u}_{\mathrm{x}_2}$, respectively.

Concentration changes when genes are ON If gene u is ON the concentration of its product increases over time. To specify this principle in LTL, we have two kinds of specification – a strong one and a weak one – depending on the strictness of increase. In the strong specification, if a gene is indefinitely ON (i.e. it never becomes OFF), the expression level strictly increases (i.e. reaches to the next level). In the weak specification, although a gene is indefinitely ON, the expression level can increase or keep its current level.

In what follows, we assume that gene u regulates genes x_1, x_2, …, x_m and the threshold values for them are in this ascending order.

First we introduce the strong specification:

$$G(on_u \to F(\neg on_u \vee u_{x_1})), \tag{1}$$
$$G((on_u \wedge u_{x_1}) \to (u_{x_1} U(\neg on_u \vee u_{x_2}))), \tag{2}$$
$$G((on_u \wedge u_{x_2}) \to (u_{x_2} U(\neg on_u \vee u_{x_3}))), \tag{3}$$
$$\vdots$$
$$G((on_u \wedge u_{x_{m-1}}) \to (u_{x_{m-1}} U(\neg on_u \vee u_{x_m}))), \tag{4}$$
$$G((on_u \wedge u_{x_m}) \to (u_{x_m} W \neg on_u)). \tag{5}$$

To see what the above formula says, suppose that u is ON and its concentration is between $\mathrm{u}_{\mathrm{x}_2}$ and $\mathrm{u}_{\mathrm{x}_3}$. Recall that the proposition u_{x_i} means the concentration of u exceeds the threshold $\mathrm{u}_{\mathrm{x}_i}$. Thus the left-hand sides of (1)–(3) in the above formula hold. From the specification on the total order of thresholds, if u_{x_2} is true then u_{x_1} is also true. Accordingly, (1)–(3) may be summed up as that concentration of u is not less than $\mathrm{u}_{\mathrm{x}_2}$ until u is turned OFF, otherwise it eventually exceeds $\mathrm{u}_{\mathrm{x}_3}$. Behaviours that satisfy this constraint have a starting concentration of u between $\mathrm{u}_{\mathrm{x}_2}$ and $\mathrm{u}_{\mathrm{x}_3}$, and in some future the concentration of u exceeds $\mathrm{u}_{\mathrm{x}_3}$ but until that time it remains above $\mathrm{u}_{\mathrm{x}_2}$ (Fig. 10(a)). The exception is that u is turned OFF before reaching $\mathrm{u}_{\mathrm{x}_3}$, so that u does not exceed $\mathrm{u}_{\mathrm{x}_3}$ (Fig. 10(b)). Behaviours in which u falls below $\mathrm{u}_{\mathrm{x}_2}$ while being ON are excluded. Moreover, u is not allowed to remain between $\mathrm{u}_{\mathrm{x}_2}$ and $\mathrm{u}_{\mathrm{x}_3}$ indefinitely although it is ON. We consider such behaviours to be incorrect in the strong specification. If the concentration of u is basal, only (1) applies. If u is above $\mathrm{u}_{\mathrm{x}_m}$, which is the greatest threshold, then all clauses apply but are absorbed into (5). As a consequence, the above formula says that the expression level of u does not decrease as long as u is ON and must increase (unless the expression level of u is greater than $\mathrm{u}_{\mathrm{x}_m}$) if u is always ON.

Next we introduce the weak specification:

$$G(on_u \to F(\neg on_u \vee u_{x_1})),$$
$$G((on_u \wedge u_{x_1}) \to (u_{x_1} W \neg on_u)),$$
$$G((on_u \wedge u_{x_2}) \to (u_{x_2} W \neg on_u)),$$
$$\vdots$$
$$G((on_u \wedge u_{x_m}) \to (u_{x_m} W \neg on_u)).$$

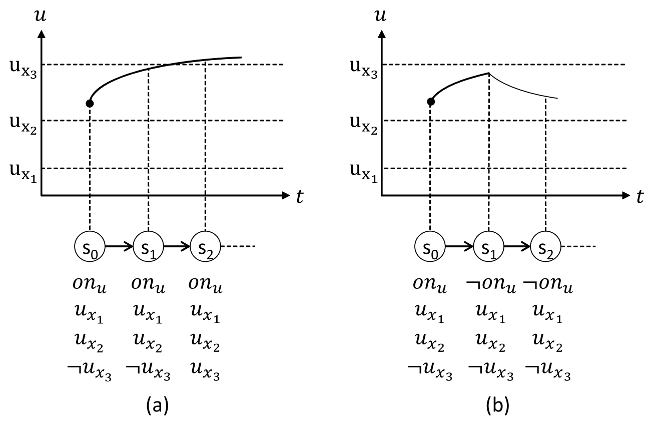
u
u_{x_3}
u_{x_2}
u_{x_1}
t
s_0
s_1
s_2
on_u on_u on_u
u_{x_1} u_{x_1} u_{x_1}
u_{x_2} u_{x_2} u_{x_2}
$\neg u_{x_3}$ $\neg u_{x_3}$ u_{x_3}
(a)
on_u $\neg on_u$ $\neg on_u$
u_{x_1} u_{x_1} u_{x_1}
u_{x_2} u_{x_2} u_{x_2}
$\neg u_{x_3}$ $\neg u_{x_3}$ $\neg u_{x_3}$
(b)

Fig. 10. Typical behaviours when gene u is ON. (a) The expression level eventually increases. (b) Gene u becomes OFF before its level reaches u_{x_3}.

The difference compared with the strong specification is that behaviours in which u keeps its concentration even if it is always ON are allowed; that is, the concentration does not have to increase strictly. This represents a situation where generation and degradation are in equilibrium.

Concentration changes when genes are OFF This is symmetric to the case when genes are ON. We again assume that gene u regulates $x_1, x_2, \dots, x_m$ and the threshold values are in this ascending order. We also have both a strong specification and a weak one.

The strong specification is as follows:

$$
\begin{aligned}
&G(\neg on_u \to F(on_u \vee \neg u_{x_m})),\\
&G((\neg on_u \wedge \neg u_{x_m}) \to (\neg u_{x_m} U (on_u \vee \neg u_{x_{m-1}}))),\\
&G((\neg on_u \wedge \neg u_{x_{m-1}}) \to (\neg u_{x_{m-1}} U (on_u \vee \neg u_{x_{m-2}}))),\\
&\vdots\\
&G((\neg on_u \wedge \neg u_{x_2}) \to (\neg u_{x_2} U (on_u \vee \neg u_{x_1}))),\\
&G((\neg on_u \wedge \neg u_{x_1}) \to (\neg u_{x_1} W on_u)).
\end{aligned}
$$

The weak specification is as follows:

$$
\begin{aligned}
&G(\neg on_u \to F(on_u \vee \neg u_{x_m})),\\
&G((\neg on_u \wedge \neg u_{x_m}) \to (\neg u_{x_m} W on_u)),\\
&G((\neg on_u \wedge \neg u_{x_{m-1}}) \to (\neg u_{x_{m-1}} W on_u)),\\
&\vdots\\
&G((\neg on_u \wedge \neg u_{x_1}) \to (\neg u_{x_1} W on_u)).
\end{aligned}
$$

In the strong specification, it is not possible that u keeps its concentration when it is always OFF but is possible in the weak specification.

We make a comment on our interpretation of 'increasing' and 'decreasing' of gene expression levels. Let us consider a behaviour such that gene x is expressed between a threshold x_1 and x_2 and gene x is ON, then finally gene x reaches x_2. Such behaviour is represented as a time structure $\dots\{on_x, x_1\}\{on_x, x_1, x_2\}\dots$. The natural interpretation of this behaviour is that gene x is expressed during the transition from the state $\{on_x, x_1\}$ to $\{on_x, x_1, x_2\}$. Another interpretation seems to be possible: a gene x once goes down (remaining over x_1) before reaching x_2 (Fig. 11(a)). We, however, choose the first natural interpretation. The latter behaviour is represented as a different behaviour as shown in Fig. 11(b). Note that the discrete behaviour in Fig. 11(b) does not violate the behaviour principles described so far. As a result, the sentence 'gene x is increasing (ON)' is interpreted as its first derivative is positive. Similarly, 'gene x is decreasing (OFF)' is interpreted as its first derivative is negative.

What about stuttering, i.e. the same state occurs successively many times? For example, in the discrete behaviour of Fig. 11(b), we may have the same state as s_1 10 times between s_0 and s_1. Such behaviour is possible and does not violate the behaviour principles since our LTL does not use next-time operator 'X' and is stutter-invariant [36].

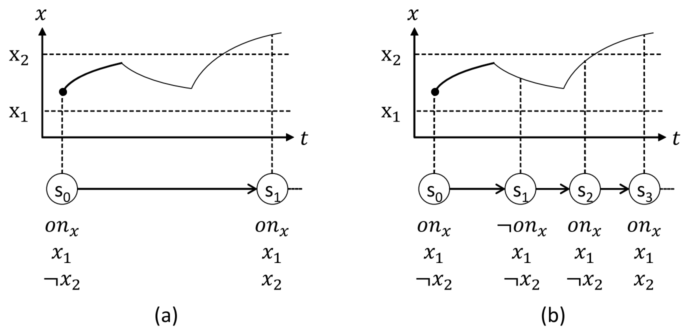

Fig. 11. (a) A wrong interpretation of a behaviour in which gene x is increasing but goes down a little while before reaches a threshold x_2. (b) A correct discretisation of the behaviour.

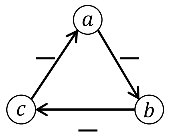

Fig. 12. Synthetic circadian clock. The transcription product of gene a inhibits gene b, gene b similarly inhibits gene c and finally gene c inhibits gene a. This network produces periodic expression pattern of each gene.

The behavioural specification for a given network is the conjunction of all clauses which are derived according to the behaviour principles introduced this section. Note that all clauses is enclosed in 'G' operator. Since $G\varphi \wedge G\psi \leftrightarrow G(\varphi \wedge \psi)$ is an axiom of LTL, we can bind up all clauses in one 'G' operator.

3.4. Biological properties in LTL

Many biologically interesting properties can be described in temporal logic [10,4,17]. For example, the property 'the system eventually reaches a state in which gene x is active but gene y is not active' is a type of *reachability* described as $F(on_x \wedge \neg on_y)$. The property 'the concentration of x is always above the threshold x_y' is a type of *stability* described as Gx_y. *Oscillation*, where 'some property ϕ is alternately true and false indefinitely', is described as $GF\phi \wedge GF\neg\phi$. Conditional properties can also be specified. For example, 'if gene x is always OFF then the property ϕ holds' is described as $(G\neg on_x) \rightarrow \phi$. Furthermore, we can use any combination of the above.

We do not need to confine ourselves to the above templates. We can use full LTL to specify properties of interest.

3.5. Example analysis 1

To see how an actual gene network is analysed in our framework, we demonstrate our method by the synthetic circadian clock [15] depicted in Fig. 12.

We introduce the set of propositions $\{on_a, on_b, on_c, a_b, b_c, c_a\}$. Using these propositions, the behaviour specification of this network is given as follows.

$$
\begin{aligned}
&G(a_b \leftrightarrow \neg on_b) \wedge \\
&G(b_c \leftrightarrow \neg on_c) \wedge \\
&G(c_a \leftrightarrow \neg on_a) \wedge \\
&G(on_a \rightarrow F(a_b \vee \neg on_a)) \wedge \\
&G(on_a \wedge a_b \rightarrow (a_b W \neg on_a)) \wedge \\
&G(\neg on_a \rightarrow F(\neg a_b \vee on_a)) \wedge \\
&G(\neg on_a \wedge \neg a_b \rightarrow (\neg a_b W on_a)) \wedge \\
&\ldots
\end{aligned}
$$

In this specification, we assume that each gene can be expressed autonomously if the inhibitor is not effective.

For this network let us check the property 'each gene oscillates', which is written in LTL as:

$$GFon_a \wedge GF\neg on_a \wedge$$

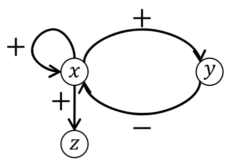

Fig. 13. The network of mucus production in *P. aeruginosa*, where x positively regulates mucus production, represented as z, and y inhibits x, which x positively regulates.

$$GFon_b \wedge GF\neg on_b \wedge$$

$$GFon_c \wedge GF\neg on_c.$$

The analysis is performed by checking the satisfiability of the conjunction of the above formula. There are several tools for checking satisfiability of LTL (see [40]). We used T^3-builder [5] to check LTL satisfiability and had the answer 'Yes'. This means that the network can produce the behaviour in which all genes oscillates.

Then we have a question. Can all genes be always OFF? Let us check the property:

$$G(\neg on_a \wedge \neg on_b \wedge \neg on_c).$$

Then the result was 'No'. This means it is impossible that all genes are indefinitely OFF (i.e. it never becomes ON).

3.6. Example analysis 2

Another example is the analysis of the mucus production system in the bacteria *Pseudomonas aeruginosa*. *P. aeruginosa* produces a heavy mucus (alginate) in the lungs of cystic fibrosis patients, causing respiration deficiency and being the major cause of mortality [21]. Bacteria isolated from the lungs of such patients can form stable mucus colonies, with a majority of these bacteria presenting a mutation. Hence it is natural to think that the mutation is the cause of the transition to the mucoid state. However, we show that wild-type bacteria have multistationarity where one stable state regularly produces mucus while the other does not; that is to say, the change from the non-mucoid state to the mucoid state can be epigenetic (a stable change of phenotype without mutation). This example is borrowed from [9,22].

The gene regulatory network that controls mucus production has been elucidated [43,23] and is depicted in Fig. 13. In this figure, z represents alginate synthesis (i.e. mucus production), x activates mucus production, and y is an inhibitor of x.

We introduce the set of propositions $\{on_x, on_y, on_z, x_x, x_y, x_z, y_x\}$, where z is not a gene, but on_z means that mucus is produced.

Among the thresholds for concentrations of x, it has been shown that $\mathrm{x_z}$ is the highest [23]. Thus there are two possibilities for the order, $\mathrm{x_x < x_y < x_z}$ or $\mathrm{x_y < x_x < x_z}$. We have two specifications depending on the order of the thresholds. The behaviour specification for the order $\mathrm{x_x < x_y < x_z}$ is given in Fig. 14, where we chose the strong specification.

The properties that should be checked are as follows:

- The bacteria regularly produces mucus: Gon_z.
- The bacteria never produces mucus: $G\neg on_z$.

We check whether each property, in conjunction with the behavioural specification, is satisfiable. The result of checking is that both properties are satisfiable in both threshold orderings. Therefore, it is *possible* that the wild-type bacteria have both mucoid and non-mucoid behaviour. This result motivates us to verify this hypothesis experimentally.

In the above analysis we do not constrain the multivariate regulation function for x which merges the inputs from x and y. That is, when both x and y are effective, x has a choice of active or inactive. Now we assume that the negative effect from y is superior to the positive effect from x. In this case the bacteria may not become mucoid state since $\mathrm{x_y < x_z}$. We check this hypothesis. We modify the behavioural specification by replacing the clause $G((\neg x_x \wedge y_x) \rightarrow \neg on_x)$ in Fig. 14 with $G(y_x \rightarrow \neg on_x)$. We check whether the modified behavioural specification with the property Gon_z is not satisfiable. This is actually the case for both orderings of $\mathrm{x_x}$ and $\mathrm{x_y}$. These results mean the hypothesis that wild-type *P. aeruginosa* may have a stable mucoid state is rebutted by the assumption that the negative effect of y overpowers the positive effect of x.

According to the current biological knowledge, all wild-type *P. aeruginosa* strains have the genetic capacity to synthesise alginate but normally produce only very small amounts of this polymer. Mucoid phenotype is only observed in mutants, and the conversion to this phenotype from wild-type is not observed outside the human host [20] in which the bacteria are mutated. Our latter analysis coincides with these biological facts. Our analysis assumes that in the regulation function of gene x the negative effect of gene y overpowers the positive effect of gene x. If it is confirmed experimentally, our model clarifies why wild-type *P. aeruginosa* do not have mucoid phenotype.

$$
\begin{aligned}
&G(x_z \to x_y) \wedge\\
&G(x_y \to x_x) \wedge\\
&G((x_x \wedge \neg y_x) \to on_x) \wedge\\
&G((\neg x_x \wedge y_x) \to \neg on_x) \wedge\\
&G(x_y \leftrightarrow on_y) \wedge\\
&G(x_z \leftrightarrow on_z) \wedge\\
&G(on_x \to F(\neg on_x \vee x_x)) \wedge\\
&G((on_x \wedge x_x) \to (x_x U(\neg on_x \vee x_y))) \wedge\\
&G((on_x \wedge x_y) \to (x_y U(\neg on_x \vee x_z))) \wedge\\
&G((on_x \wedge x_z) \to (x_z W \neg on_x)) \wedge\\
&G(\neg on_x \to F(on_x \vee \neg x_z)) \wedge\\
&G((\neg on_x \wedge \neg x_z) \to (\neg x_z U(on_x \vee \neg x_y))) \wedge\\
&G((\neg on_x \wedge \neg x_y) \to (\neg x_y U(on_x \vee \neg x_x))) \wedge\\
&G((\neg on_x \wedge \neg x_x) \to (\neg x_x W on_x)) \wedge\\
&G(on_y \to F(\neg on_y \vee y_x)) \wedge\\
&G((on_y \wedge y_x) \to (y_x W \neg on_y)) \wedge\\
&G(\neg on_y \to F(on_y \vee \neg y_x)) \wedge\\
&G((\neg on_y \wedge \neg y_x) \to (\neg y_x W on_y))
\end{aligned}
$$

Fig. 14. Specification for possible behaviours of the network for mucus production in *P. aeruginosa*.

3.7. About complexity

Analysis in our method is based on LTL satisfiability checking, which is a PSPACE-complete problem [44]. Therefore, the known algorithms are exponential in the size of an input formula. As we can see from Section 3.3, the length of a formula specifying possible behaviours of a network is proportional to the size of the network (the number of nodes and edges). Therefore, some techniques are strongly desirable which ease this computational cost. We develop two techniques for this purpose. The first is a modular analysis which is discussed in the next section, while the second is an approximate analysis which is discussed in Section 5.

4. Modular analysis of gene regulatory networks

In this section we present a method for modular analysis of gene regulatory networks. A network is divided into several subnetworks, and the possible behaviours of the subnetworks are integrated to obtain behaviours of the whole network. By ignoring local propositions in the subnetworks, we reduce the dimension of the state space. Thus this method reduces the cost of the analysis of a gene network.

4.1. Preliminary

We first present the mathematical preliminaries for LTL satisfiability checking.

Definition 5. Let $\sigma, \sigma' \in \mathfrak{P}(A)^\omega$. The expression $\sigma \oplus \sigma'$ denotes the sequence $(\sigma[0] \cup \sigma'[0])(\sigma[1] \cup \sigma'[1])(\sigma[2] \cup \sigma'[2])\ldots$.

Definition 6. Let ϕ be an LTL formula. *Prop*(ϕ) denotes the set of propositions occurring in ϕ.

The next definition is of the Büchi automaton, which is a kind of ω-automata accepting infinite words.

Definition 7. A *Büchi automaton* is a quintuple $\langle Q, \Sigma, \delta, q_I, F\rangle$, where Q is a finite set of states, Σ is a finite alphabet, $\delta : Q \times \Sigma \to \mathfrak{P}(Q)$ is the state transition function, $q_I \in Q$ is the initial state, and $F \subseteq Q$ is the set of accepting states. A *run* of a Büchi automaton on an infinite word $\alpha = \alpha[0]\alpha[1]\cdots \in \Sigma^\omega$ is an infinite sequence $\rho = \rho[0]\rho[1]\cdots \in Q^\omega$, such that $\rho[0] = q_I$ and $\rho[i+1] \in \delta(\rho[i], \alpha[i])$ for all $i \geq 0$. An infinite word α is *accepted* by the automaton if the run over α visits at least one state in F infinitely often. We denote the set of infinite words accepted by an automaton $\mathcal{A}$ by $L(\mathcal{A})$.

It is known that Büchi automata are closed under intersection [48].

Theorem 1. *Büchi automata are closed under intersection. Namely, for any Büchi automata $\mathcal{A}$ and $\mathcal{B}$, it is possible to construct a Büchi automaton that accepts $L(\mathcal{A}) \cap L(\mathcal{B})$, where the alphabet sets of $\mathcal{A}$ and $\mathcal{B}$ are the same.*

A time structure satisfying the LTL formula ϕ is an infinite word over the alphabet $\Sigma = \mathfrak{P}(Prop(\phi))$. The next theorem [50] states that we can construct a Büchi automaton that exactly accepts the models of ϕ. In the following theorem, $|\phi|$ denotes the length of ϕ.

Theorem 2 *(Vardi). Given an LTL formula ϕ, one can construct a Büchi automaton $\mathcal{A}_\phi = \langle Q, \Sigma, \delta, q_I, F\rangle$ such that $|Q|$ is in $2^{O(|\phi|)}$, $\Sigma = \mathfrak{P}(Prop(\phi))$ and $L(\mathcal{A}_\phi) = \{\sigma \in \mathfrak{P}(Prop(\phi))^\omega \mid \sigma \models \phi\}$.*

Corollary 1. *An LTL formula ϕ is satisfiable if and only if $L(\mathcal{A}_\phi) \neq \emptyset$.*

This corollary says that the problem of LTL satisfiability checking is reduced to emptiness testing of a Büchi automaton.

4.2. Modular satisfiability checking of LTL

Suppose that $\phi = \phi_1 \wedge \cdots \wedge \phi_n$. The idea of modular method is to construct the automaton A_ϕ by constructing the automata $A_{\phi_1}, \ldots, A_{\phi_n}$ individually and intersecting them. First we introduce some terminology.

Definition 8. Let A and B be finite sets such that $A \subseteq B$ and suppose that $\sigma \in \mathfrak{P}(B)^\omega$. The sequence $\sigma \downarrow A$ denotes $(\sigma[0] \cap A)(\sigma[1] \cap A) \cdots \in \mathfrak{P}(A)^\omega$. Let $L \subseteq \mathfrak{P}(B)^\omega$. The set $L \downarrow A$ denotes $\{\sigma \downarrow A \mid \sigma \in L\}$, the restriction of L to A. Let $M \subseteq \mathfrak{P}(A)^\omega$. The set $\overline{M}^B \subseteq \mathfrak{P}(B)^\omega$ denotes the maximum set such that $M = \overline{M}^B \downarrow A$.

Proposition 1. *$\overline{M}^B$ always exists.*

Proof. It is easily seen that $X = \bigcup_{\sigma \in M}\{\sigma \oplus \rho \mid \rho \in \mathfrak{P}(B - A)^\omega\}$ is the maximum set that satisfies $M = X \downarrow A$. □

Proposition 2. *Let $\mathcal{A} = \langle Q, \mathfrak{P}(A), \delta, q_I, F\rangle$ be a Büchi automaton. For any finite set $B \supseteq A$ there exists a Büchi automaton that accepts $\overline{L(\mathcal{A})}^B$.*

Proof. Replace the alphabet set in $\mathcal{A}$ by $\mathfrak{P}(B)$. □

Hereafter, we assume that $\phi = \phi_1 \wedge \cdots \wedge \phi_n$ is an LTL formula. Moreover, we simply write $\overline{L(\mathcal{A}_{\phi_i})}$ for $\overline{L(\mathcal{A}_{\phi_i})}^{Prop(\phi)}$.

Proposition 3. $L(\mathcal{A}_\phi) = \overline{L(\mathcal{A}_{\phi_1})} \cap \cdots \cap \overline{L(\mathcal{A}_{\phi_n})}$.

Proof. Let $\sigma \in L(\mathcal{A}_\phi)$. By Theorem 2, we have $\sigma \models \phi$. Therefore, $\sigma \models \phi_i$ for $i = 1, \ldots, n$. From this, we have $\sigma \downarrow \mathfrak{P}(Prop(\phi_i)) \models \phi_i$, that is, $\sigma \downarrow \mathfrak{P}(Prop(\phi_i)) \in L(A_{\phi_i})$ for $i = 1, \ldots, n$. By Definition 8, we have $\sigma \in \overline{L(\mathcal{A}_{\phi_i})}$ for $i = 1, \ldots, n$. The converse of this argument also holds. □

Definition 9. $LP(\phi_i) = Prop(\phi) - \bigcup_{1 \le j \le n, j \neq i} Prop(\phi_j)$ is the set of *local propositions in* ϕ_i.

Propositions that are not local to any ϕ_i are called *global propositions*.

Definition 10. Let $\mathcal{A}_{\phi_i} = \langle Q, \Sigma, \delta, q_I, F\rangle$. $\mathcal{A}^-_{\phi_i}$ denotes the Büchi automaton $\langle Q, \Sigma', \delta', q_I, F\rangle$ where $\Sigma' = \{s - LP(\phi_i) \mid s \in \Sigma\}$ and

$$\delta'(q, s') = \bigcup_{l \subseteq LP(\phi_i)} \delta(q, s' \cup l).$$

Intuitively, $\mathcal{A}^-_{\phi_i}$ is obtained by ignoring local propositions $LP(\phi_i)$ from the transition function. We need the following lemma for modular satisfiability checking.

Lemma 1. *$L(\mathcal{A}^-_{\phi_i}) = \{\sigma \downarrow (Prop(\phi) - LP(\phi_i)) \mid \sigma \in L(\mathcal{A}_{\phi_i})\}$ for $i = 1, \ldots, n$.*

Proof. Let $\mathcal{A}_{\phi_i} = \langle Q, \Sigma, \delta, q_I, F \rangle$ and $\mathcal{A}^-_{\phi_i} = \langle Q, \Sigma', \delta', q_I, F \rangle$. Suppose that $\rho \in L(\mathcal{A}^-_{\phi_i})$ and that $q[0]q[1]q[2]\dots$ is an accepting run in $\mathcal{A}^-_{\phi_i}$ over ρ. By the definition of a run, we have $q[j+1] \in \delta'(q[j], \rho[j])$ for all $j \geq 0$. Since $q[0]q[1]q[2]\dots$ is also an accepting run in $\mathcal{A}_{\phi_i}$, there exists an infinite word σ on which it is the run; that is to say, $q[j+1] \in \delta(q[j], \sigma[j])$ for all $j \geq 0$. By the definition of $\mathcal{A}^-_{\phi_i}$, there exists $\sigma'[j] \subseteq LP(\phi_i)$ such that $\sigma[j] = \rho[j] \cup \sigma'[j]$ for all $j \geq 0$. Thus we have $\sigma = \rho \oplus \sigma'$, i.e. $\sigma \downarrow (Prop(\phi) - LP(\phi_i)) = \rho$. The converse inclusion is trivial. $\square$

Corollary 2. $\overline{L(\mathcal{A}_{\phi_i})} = \overline{L(\mathcal{A}^-_{\phi_i})}$.

Proof. First we prove $\overline{L(\mathcal{A}_{\phi_i})} \subseteq \overline{L(\mathcal{A}^-_{\phi_i})}$. Let $\sigma \in \overline{L(\mathcal{A}_{\phi_i})}$. By the proof of Proposition 1, there exists $\sigma' \in L(\mathcal{A}_{\phi_i})$ and $\rho \in \mathfrak{P}(Prop(\phi) - Prop(\phi_i))^\omega$ such that $\sigma = \sigma' \oplus \rho$. By the proof of Lemma 1, there exists $\sigma'' \in L(\mathcal{A}^-_{\phi_i})$ and $\sigma''' \in \mathfrak{P}(LP(\phi_i))^\omega$ such that $\sigma' = \sigma'' \oplus \sigma'''$. Thus we have $\sigma = \sigma'' \oplus (\sigma''' \oplus \rho)$, where $\sigma''' \oplus \rho \in \mathfrak{P}((Prop(\phi) - Prop(\phi_i)) \cup LP(\phi_i))^\omega = \mathfrak{P}(Prop(\phi) - (Prop(\phi_i) - LP(\phi_i)))^\omega$. As a consequence, we have $\sigma'' \oplus (\sigma''' \oplus \rho) = \sigma \in \overline{L(\mathcal{A}^-_{\phi_i})}$. Now we prove the inverse inclusion. Let $\sigma \in \overline{L(\mathcal{A}^-_{\phi_i})}$. By the proof of Proposition 1, there exists $\sigma' \in L(\mathcal{A}^-_{\phi_i})$ and $\rho \in \mathfrak{P}(Prop(\phi) - (Prop(\phi_i) - LP(\phi_i)))^\omega$ such that $\sigma = \sigma' \oplus \rho$. Since $Prop(\phi) - (Prop(\phi_i) - LP(\phi_i)) = (Prop(\phi) - Prop(\phi_i)) \cup LP(\phi_i)$, we have $\rho = \rho' \oplus \rho''$ for some $\rho' \in \mathfrak{P}(Prop(\phi) - Prop(\phi_i))^\omega$ and $\rho'' \in \mathfrak{P}(LP(\phi_i))^\omega$. Thus $\sigma = \sigma' \oplus \rho = \sigma' \oplus (\rho' \oplus \rho'') = (\sigma' \oplus \rho'') \oplus \rho'$. Here since we know $\sigma' = (\sigma' \oplus \rho'') \downarrow (Prop(\phi_i) - LP(\phi_i)) \in L(\mathcal{A}^-_{\phi_i})$, we have $\sigma' \oplus \rho'' \in L(A_{\phi_i})$ by Lemma 1. Thus by applying Proposition 1, we have $(\sigma' \oplus \rho'') \oplus \rho' = \sigma \in \overline{L(A_{\phi_i})}$. $\square$

The following theorem is the main result of this section.

Theorem 3. $L(\mathcal{A}_\phi) \neq \emptyset \Leftrightarrow \overline{L(\mathcal{A}^-_{\phi_1})} \cap \dots \cap \overline{L(\mathcal{A}^-_{\phi_n})} \neq \emptyset$.

Proof. By Proposition 3, this claim is equivalent to

$$\overline{L(\mathcal{A}_{\phi_1})} \cap \dots \cap \overline{L(\mathcal{A}_{\phi_n})} \neq \emptyset \Leftrightarrow \overline{L(\mathcal{A}^-_{\phi_1})} \cap \dots \cap \overline{L(\mathcal{A}^-_{\phi_n})} \neq \emptyset.$$

This is the case since by Corollary 2, we have $\overline{L(\mathcal{A}_{\phi_i})} = \overline{L(\mathcal{A}^-_{\phi_i})}$ for $i = 1, \dots, n$. $\square$

From this theorem, we have the following modular satisfiability checking method for $\phi = \phi_1 \wedge \dots \wedge \phi_n$.

1. Construct $\mathcal{A}_{\phi_i}$ for each i.
2. Abstract $\mathcal{A}_{\phi_i}$ to $\mathcal{A}^-_{\phi_i}$ by deleting the local propositions $LP(\phi_i)$.
3. Intersect all $\mathcal{A}^-_{\phi_i}$. This is done by using Proposition 2 and Theorem 1.
4. Check non-emptiness of the intersected automaton.

The key of efficacy is to abstract local propositions from automata $\mathcal{A}_{\phi_i}$.

The more local propositions we have, the more we can abstract automata $\mathcal{A}^-_{\phi_i}$, and thus the intersected automaton will be semantically simple and the cost of checking non-emptiness will be reduced. Thus the efficacy of modular checking depends on whether there are many local propositions. Note that this method, however, needs extra costs of intersecting automata whose complexity is linear in the product of the sizes of intersected automata.

Our modular method can be compared to Aoshima et al.'s modular method [6,5]. They divide an LTL formula into several modules, compute constraints on global propositions of each module and replace them with those constraints. Their modular method is at LTL level. In our method, we compute Büchi automata for each module and simplify them, then compute a product of each automaton. So our modular method is at automaton level. Since computing constraints on global propositions of each module is not trivial, our modular method is more applicable.

4.3. Application to gene regulatory network analysis

We apply the modular satisfiability checking of LTL to our framework of network analysis.

Let $\phi = \phi_1 \wedge \dots \wedge \phi_n$ be a behavioural specification of a network and ψ be a biological property. We check whether $\phi \wedge \psi$ is satisfiable or $\phi \wedge \neg\psi$ is unsatisfiable. By Theorem 3, $\phi \wedge \psi$ is satisfiable if and only if $\overline{L(\mathcal{A}^-_{\phi_1})} \cap \dots \cap \overline{L(\mathcal{A}^-_{\phi_n})} \cap \overline{L(\mathcal{A}_\psi)} \neq \emptyset$. Similarly, $\phi \wedge \neg\psi$ is unsatisfiable if and only if $\overline{L(\mathcal{A}^-_{\phi_1})} \cap \dots \cap \overline{L(\mathcal{A}^-_{\phi_n})} \cap \overline{L(\mathcal{A}_{\neg\psi})} = \emptyset$.

The problem is how to subdivide ϕ into $\phi_1 \wedge \dots \wedge \phi_n$. For the analysis of gene regulatory networks, behaviour specification of a network can be decomposed into the specifications for its subnetworks. Therefore, we can take ϕ_i as a behavioural specification for each subnetwork. The local propositions for ϕ_i are propositions concerning nodes and edges which are 'confined' to subnetworks, that is to say, nodes that are only connected by edges in the subnetwork. Subnetworks that contain many such local propositions represent a good division. Note that propositions contained in ψ are global propositions.

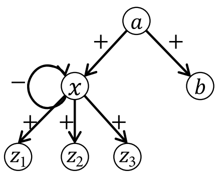

Fig. 15. Example.

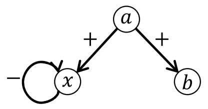

Fig. 16. Subnetwork 1.

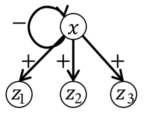

Fig. 17. Subnetwork 2.

$$
\begin{array}{lcl}
G(& (a_x \to a_b) & \wedge \\
 & (a_b \leftrightarrow on_b) & \wedge \\
 & (on_a \to F(a_b \vee \neg on_a)) & \wedge \\
 & ((on_a \wedge a_b) \to (a_b W \neg on_a)) & \wedge \\
 & ((on_a \wedge a_x) \to (a_x W \neg on_a)) & \wedge \\
 & (\neg on_a \to F(\neg a_x \vee on_a)) & \wedge \\
 & ((\neg on_a \wedge \neg a_x) \to (\neg a_x W on_a)) & \wedge \\
 & ((\neg on_a \wedge \neg a_b) \to (\neg a_b W on_a)) &)
\end{array}
$$

Fig. 18. Specification for subnetwork 1.

4.4. Result and discussion

In this section, we apply our modular method for three example networks and evaluate how the modular method improves the efficiency.

First example is the artificial network depicted in Fig. 15. We subdivide this network into two subnetworks depicted in Figs. 16 and 17.

For this network, we do not explicitly give a biological property ψ but assume that it consists of the propositions $on_a, on_b, on_{z_1}, on_{z_2}$ and on_{z_3}, and consider a as the system input and b, z_1, z_2 and z_3 as the system outputs. For each subnetwork we give behavioural specifications (Figs. 18, 19). We can see that a_b is the local proposition in the specification for subnetwork 1 and that x_x, on_x, x_{z_1}, x_{z_2} and x_{z_3} are the local propositions for subnetwork 2.

Note that since gene x is on the boundary between subnetworks 1 and 2, we can choose which subnetwork specification includes the clauses about the regulation of x (from gene a and itself). If we include them in to the specification for subnetwork 1, proposition a_x will be a local proposition but on_x and x_x will not be local propositions. Since we preferred to have more local propositions, we included the clauses into the specification for subnetwork 2.

In Table 1, we show the size and number of propositions for each automaton and analysis time[3] (the sum of automaton construction time and emptiness testing time). Translations from LTL to Büchi automata[4] and computing their intersections make use of our implementation based on Aoshima's algorithm [5].

This example is rather small and has so few local propositions that we do not benefit from modular analysis. We could not compensate the extra cost of intersecting automata.

Let us consider another example depicted in Fig. 20, involving *malT* gene expression in *Escherichia coli* taken from [2].

[3] The following computational environment was used: CPU Intel(R) Core(TM) i7-3820 3.60 GHz and 32 GB of RAM.

[4] For technical reasons, we used generalised Büchi automata from which we can construct equivalent Büchi automata.

$$
\begin{array}{lcl}
G(& (x_x \to x_{z_3}) & \wedge \\
 & (x_{z_3} \to x_{z_2}) & \wedge \\
 & (x_{z_2} \to x_{z_1}) & \wedge \\
 & (x_{z_1} \leftrightarrow on_{z_1}) & \wedge \\
 & (x_{z_2} \leftrightarrow on_{z_2}) & \wedge \\
 & (x_{z_3} \leftrightarrow on_{z_3}) & \wedge \\
 & ((a_x \wedge \neg x_x) \to on_x) & \wedge \\
 & (\neg a_x \to \neg on_x) & \wedge \\
 & (on_x \to F(x_{z_1} \vee \neg on_x)) & \wedge \\
 & ((on_x \wedge x_{z_1}) \to (x_{z_1} W \neg on_x)) & \wedge \\
 & ((on_x \wedge x_{z_2}) \to (x_{z_2} W \neg on_x)) & \wedge \\
 & ((on_x \wedge x_{z_3}) \to (x_{z_3} W \neg on_x)) & \wedge \\
 & ((on_x \wedge x_x) \to (x_x W \neg on_x)) & \wedge \\
 & (\neg on_x \to F(\neg x_x \vee on_x)) & \wedge \\
 & ((\neg on_x \wedge \neg x_x) \to (\neg x_x W on_x)) & \wedge \\
 & ((\neg on_x \wedge \neg x_{z_3}) \to (\neg x_{z_3} W on_x)) & \wedge \\
 & ((\neg on_x \wedge \neg x_{z_2}) \to (\neg x_{z_2} W on_x)) & \wedge \\
 & ((\neg on_x \wedge \neg x_{z_1}) \to (\neg x_{z_1} W on_x)) &)
\end{array}
$$

Fig. 19. Specification for subnetwork 2.

Table 1
The result of analyses for the example network. ‘S’, ‘E’ and ‘P’ represents the number of states, edges and propositions of the automaton, respectively. ‘T’ represents the entire time of the analysis. ‘T(ET)’ represents the time of empty-testing of the automaton. ‘Direct’ means the network is not divided in the analysis. ‘Modular’ means the network is divided into subnetwork 1 and subnetwork 2 in the analysis.

	S	E	P	T	T(ET)
Direct	33	682	12	0.020 s	<0.001 s
Modular	31	592	7	0.143 s	<0.001 s

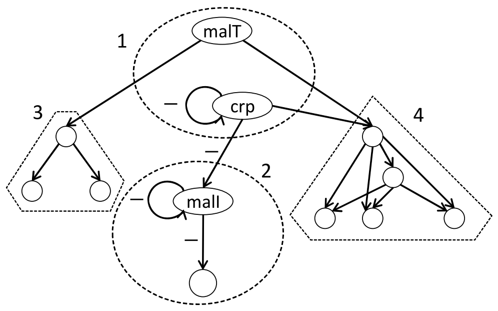

Fig. 20. The network from *E. coli* involving the *malT* gene. Positive signs on edges are omitted.

As depicted in Fig. 20, we divide this network into subnetwork 1 to 4. We perform three modular analyses (Modular (A) to (C) in the table) depending on how we divide the network. We show the results in Table 2.

Except Modular (C), modular analysis is better than direct analysis thanks to the improvement of emptiness testing time. This improvement is attributed to the reduction of the number of propositions. Note that the size of automaton is the same as the direct analysis, but the number of propositions are few. This shows how it becomes easier to check emptiness of automata if we have fewer propositions. Unfortunately Modular (C) is not so efficient. The reason is that the size of

Table 2
The result of analyses of the network in Fig. 20. 'S', 'E' and 'P' represents the number of states, edges and propositions of the automaton, respectively. 'T' represents the entire time of the analysis. 'T(ET)' represents the time of empty-testing of the automaton. 'Direct' means the network is not divided in the analysis. 'Modular (A)' means the network is divided into two networks, i.e. subnetworks 1–3 and subnetwork 4. 'Modular (B)' means the network is divided into three networks, i.e. subnetworks 1&2, subnetwork 3 and subnetwork 4. 'Modular (C)' means the network is divided into four networks.

	S	E	P	T	T(ET)
Direct	8281	14 698 200	28	28.054 s	26.474 s
Modular (A)	8281	14 698 200	8	8.317 s	3.108 s
Modular (B)	8281	14 698 200	9	6.579 s	3.164 s
Modular (C)	15 121	52 414 560	10	38.511 s	26.521 s

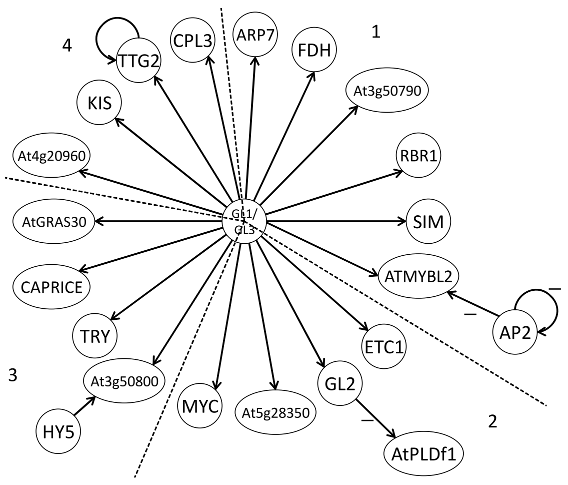

Fig. 21. A network in *Arabidopsis thaliana*.

Table 3
The result of analyses of the network in Fig. 21. 'S', 'E' and 'P' represents the number of states, edges and propositions of the automaton, respectively. 'T' represents the entire time of the analysis. 'T(ET)' represents the time of empty-testing of the automaton. 'Direct' means the network is not divided in the analysis. 'Modular (A)' means the network is divided into two subnetworks, i.e. subnetworks 1&2 and subnetworks 3&4. 'Modular (B)' means the network is divided into four subnetworks.

	S	E	P	T	T (ET)
Direct	3041	4 395 400	46	3.092 s	2.272 s
Modular (A)	2961	3 775 532	20	2.432 s	0.648 s
Modular (B)	2961	3 910 028	22	3.013 s	0.624 s

automaton with Modular (C) becomes larger than Direct thus the computation of the product automaton was not negligible. However, the empty testing time was about the same as the direct analysis although the size of the automaton is much larger. This indicates that the reduction of the number of proposition actually facilitates emptiness testing. This result shows that the number of division also affects the efficiency of modular analysis.

Let us see another example shown in Fig. 21. This network is obtained from ReIN,[5] which is a database of genes of *Arabidopsis thaliana*. We divide this network into four subnetworks, as numbered in Fig. 21. We perform two modular analyses depending on how we divide the network. The results are depicted in Table 3.

Since the direct analysis does not take much time, the benefit we have from modular analysis is a bit small. Even though, if we compare the time of empty testing, the cost is considerably reduced. Modular analysis almost compensates the cost of computing product automata and file I/O.

5 http://arabidopsis.med.ohio-state.edu/REIN/.

5. Approximate analysis

In modular analysis, we obtain the automaton $\mathcal{A}_{\phi_i}$ from each subnetwork specification ϕ_i, abstract (or simplify) $\mathcal{A}_{\phi_i}$ by ignoring local propositions, then intersect them to obtain the automaton of the entire network. We now try to simplify not the automata but the subnetwork specifications themselves by abstracting local propositions. Since we forget some propositions, we may not obtain formulae equivalent to the original ones, but may be able to find stronger or weaker ones. A stronger one is sufficient for checking satisfiability of the original one and a weaker one is sufficient for checking unsatisfiability. This means that the set of behaviours are compressed or expanded from the behaviours characterised by the original specifications. We call such formulae *approximate behavioural specifications* for subnetworks.

What about biological properties? We do not consider approximation of biological properties by two reasons. First reason is that it is difficult to find approximate formulae for biological properties. Moreover, it is difficult to ensure that approximate formula correctly reflects the intended biological property. Second reason is that formulae which describe biological properties are not so large in general. Therefore we do not benefit from approximation of biological properties.

5.1. General framework of approximate analysis

First we introduce some formal terminology for approximate analysis of LTL satisfiability.

Definition 11. Let ϕ and ψ be LTL formulae such that $Prop(\psi) \subseteq Prop(\phi)$. We define the relation $\preceq$ between LTL formulae as follows:

$$\psi \preceq \phi \Leftrightarrow L(\mathcal{A}_\psi) \subseteq L(\mathcal{A}_\phi) \downarrow Prop(\psi).$$

We call ψ an *under-approximation* of ϕ.

Definition 12. Let ϕ and ψ be LTL formulae such that $Prop(\psi) \subseteq Prop(\phi)$. We define the relation $\sqsupseteq$ between LTL formulae as follows:

$$\psi \sqsupseteq \phi \Leftrightarrow L(\mathcal{A}_\psi) \supseteq L(\mathcal{A}_\phi) \downarrow Prop(\psi).$$

We call ψ an *over-approximation* of ϕ.

Remark 1. The relation $\preceq$ is the inverse relation of $\sqsupseteq$ when $Prop(\psi) = Prop(\phi)$. We have $\bot \preceq \phi$ and $\top \sqsupseteq \phi$ for any ϕ.

The following theorems state that $\psi \preceq \phi$ means that ψ is simpler (i.e. having fewer propositions) and stronger than ϕ while $\psi \sqsupseteq \phi$ means that ψ is simpler and weaker than ϕ.

Theorem 4. *Let ϕ and ψ be LTL formulae such that $\psi \preceq \phi$. If ψ is satisfiable then ϕ is satisfiable.*

Proof. Proof by contrapositive. Suppose that ϕ is not satisfiable. By Corollary 1, $L(\mathcal{A}_\phi) = \emptyset$. Then $L(\mathcal{A}_\phi) \downarrow Prop(\psi) = \emptyset$. By Definition 11, we have $L(\mathcal{A}_\psi) = \emptyset$, that is, ψ is not satisfiable. □

Theorem 5. *Let ϕ and ψ be LTL formulae such that $\psi \sqsupseteq \phi$. If ψ is not satisfiable then ϕ is not satisfiable.*

Proof. Proof by contrapositive. Suppose that ϕ is satisfiable. By Corollary 1, $L(\mathcal{A}_\phi) \neq \emptyset$. Thus there exists $\sigma \in L(\mathcal{A}_\phi)$. By Definition 12, there exists $\rho \in L(\mathcal{A}_\psi)$ such that $\sigma \downarrow Prop(\phi) = \rho$. Thus $L(\mathcal{A}_\psi) \neq \emptyset$; that is, ψ is satisfiable. □

Lemma 2. *The weak specification is an over-approximation of the strong specification.*

Proof. By definitions of strong specification and weak specification (Section 3.3). □

We now present the approximate analysis method. Let $\phi = \phi_1 \wedge \cdots \wedge \phi_n$ be a behavioural specification and ψ be a biological property. When we check the satisfiability of $\phi \wedge \psi$, we replace ϕ_i by ψ_i, which does not contain local propositions, such that $\psi_i \preceq \phi_i$, and check the satisfiability of $\psi_1 \wedge \cdots \wedge \psi_n \wedge \psi$. If this is satisfiable, then $\phi \wedge \psi$ is also satisfiable by Theorem 4. When we check the unsatisfiability of $\phi \wedge \neg\psi$, we replace ϕ_i by ψ_i such that $\psi_i \sqsupseteq \phi_i$, and check the unsatisfiability of $\psi_1 \wedge \cdots \wedge \psi_n \wedge \neg\psi$. If this is unsatisfiable, then $\phi \wedge \neg\psi$ is also unsatisfiable by Theorem 5.

Remark 2. We can combine approximate analysis and modular analysis by checking the satisfiability of the approximate specification modularly.

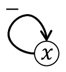

Fig. 22. Negative auto-regulation.

5.2. *Approximate specifications for network motifs*

In general, it is non-trivial to find a 'good' approximation ψ for arbitrary ϕ even if ϕ is a behavioural specification for a subnetwork of a gene regulatory network. However, it is known that gene regulatory networks contain a small set of recurring regulation patterns, called network motifs [2,3]. If we have approximate specifications for such motifs, we can reuse them for analyses of any gene networks. Therefore, we focus on approximate specifications for network motifs.

There are possibly many approximate specifications for network motifs. Since our problem is to analyse biological properties of gene networks, the approximation should be biologically meaningful. Otherwise the approximate method will produce a lot of false positives/negatives. To give biologically meaningful approximation, we focus on the biological functions of network motifs. Such functions are considered to be important or useful to many organisms and the network motifs will be used to realise their functions. We consider five motifs whose functions are well-studied [2,3]: negative auto-regulation, coherent type 1 feed-forward loops, incoherent type 1 feed-forward loops, single-input modules and multi-output feed-forward loops.

When we consider under-approximation, we concentrate on specific behaviours amongst all possible behaviours. The functions of network motifs are suitable for such specific behaviours since they are likely to function that way in the organisms, or the analysis problem may be related to such functions. That means we exclude behaviours which do not conform to motif functions.

As for over-approximation, we need to allow extra behaviours which is prohibited in the original specification. Thus extra behaviours may more or less violate the behaviour principles, e.g. 'a gene can be ON nevertheless its inhibitor is ON'. If we allow too much of such behaviours, almost all biological properties will be satisfied and the approximate analysis will not be useful. The easiest way to obtain over-approximation is just to eliminate some clauses from the original specification. Such approximate specifications may not be biologically interpreted but is sometimes useful for approximate analysis. Note, however, that such approximation does not necessarily abstract local propositions unless we carefully choose clauses we eliminate. In contrast, under-approximations cannot be obtained in such a syntactic way. We need to devise suitable approximate specifications by hand.

In the following, under-approximations are given for strong specifications and over-approximations are given for weak specifications.

Theorem 6. *Under-approximations for strong specifications are also under-approximations for weak specifications, and over-approximations for weak specifications are also over-approximations for strong specifications.*

Proof. By Lemma 2. □

Negative auto-regulation Negative auto-regulation is depicted in Fig. 22. This motif has the function of response acceleration [2,3]. In our abstraction, this function cannot be described since we cannot refer to an actual response time in LTL; that is, accelerated behaviours and non-accelerated behaviours cannot be distinguished. Therefore, we shall ignore negative auto-regulation in our analysis. For simplicity the approximate specification below is given under the assumption that there is one input and one output for x which are represented as the level in_x of input and $\mathrm{x}_{\mathrm{out}}$ of x, respectively. However, this can be easily generalised to multiple inputs and outputs.

First we show the strong specification for negative auto-regulation (weak one is obtained by replacing U-operator with W-operator):

$$G((in_x \wedge \neg x_x) \rightarrow on_x),$$
$$G(\neg in_x \rightarrow \neg on_x),$$
$$G(on_x \rightarrow F(\neg on_x \vee x_{out})),$$
$$G((on_x \wedge x_{out}) \rightarrow (x_{out} U(\neg on_x \vee x_x))),$$
$$G((on_x \wedge x_x) \rightarrow (x_x W \neg on_x)),$$
$$G(\neg on_x \rightarrow F(on_x \vee \neg x_x)),$$
$$G((\neg on_x \wedge \neg x_x) \rightarrow (\neg x_x U(on_x \vee \neg x_{out}))),$$
$$G((\neg on_x \wedge \neg x_{out}) \rightarrow (\neg x_{out} W on_x)),$$
$$G(x_x \rightarrow x_{out}).$$

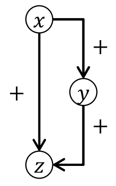

Fig. 23. Coherent type 1 feed-forward loop.

We now present the following under-approximation for negative auto-regulation in which negative auto-regulation of x is ignored:

$$G(in_x \leftrightarrow on_x),$$
$$G(on_x \rightarrow F(x_{out} \vee \neg on_x)),$$
$$G((on_x \wedge x_{out}) \rightarrow (x_{out} W \neg on_x)),$$
$$G(\neg on_x \rightarrow F(\neg x_{out} \vee on_x)),$$
$$G((\neg on_x \wedge \neg x_{out}) \rightarrow (\neg x_{out} W on_x)).$$

The abstracted local proposition is x_x.

An over-approximation for this network motif is given below:

$$G(\neg in_x \rightarrow \neg on_x),$$
$$G(on_x \rightarrow F(\neg on_x \vee x_{out})),$$
$$G((on_x \wedge x_{out}) \rightarrow (x_{out} W \neg on_x)),$$
$$G((\neg on_x \wedge \neg x_{out}) \rightarrow (\neg x_{out} W on_x))$$

The abstracted local proposition is x_x. This specification is obtained by eliminating clauses including proposition x_x from the original weak specification. If we biologically interpret this specification, it allows extra behaviours such that gene x is OFF even if the input to x is coming, or gene x is expressed continually beyond $\mathrm{x_{out}}$ even if it is turned OFF.

Coherent type 1 feed-forward loop The coherent type 1 feed-forward loop (C1-FFL) is the pattern depicted in Fig. 23. There are two types of input function (AND/OR) for z that merge the influence of x and y [2,3]. For the AND function, C1-FFL shows a delay after stimulation by x, but no delay when the stimulation stops. For the OR function, the FFL has the opposite effect to the AND case; that is, it shows no delay after stimulation by x but shows a delay when the stimulation stops.

We show the strong specification of this motif for AND version with $\mathrm{x_y} < \mathrm{x_z}$ (the weak one can be obtained by replacing U-operator by W-operator):

$$G(on_x \rightarrow F(\neg on_x \vee x_y)),$$
$$G((on_x \wedge x_y) \rightarrow (x_y U(\neg on_x \vee x_z))),$$
$$G((on_x \wedge x_z) \rightarrow (x_z W \neg on_x)),$$
$$G(\neg on_x \rightarrow F(on_x \vee \neg x_z)),$$
$$G((\neg on_x \wedge \neg x_z) \rightarrow (\neg x_z U(on_x \vee \neg x_y))),$$
$$G((\neg on_x \wedge \neg x_y) \rightarrow (\neg x_y W on_x)),$$
$$G(x_z \rightarrow x_y),$$
$$G(x_y \leftrightarrow on_y),$$
$$G((on_y \wedge \neg y_z) \rightarrow F(\neg on_y \vee y_z)),$$
$$G((on_y \wedge y_z) \rightarrow (y_z W \neg on_y)),$$
$$G((\neg on_y \wedge y_z) \rightarrow F(on_y \vee \neg y_z)),$$
$$G((\neg on_y \wedge \neg y_z) \rightarrow (\neg y_z W on_y)),$$
$$G((x_z \wedge y_z) \leftrightarrow on_z).$$

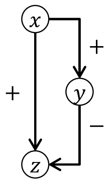

Fig. 24. Incoherent Type 1 Feed-forward loop.

For under-approximation, we ignore y and consider them as simple regulations. Thus the difference between AND and OR does not occur in approximate formula. Although the original specifications for this motif depend on the orderings of the thresholds $\mathrm{x_y}$ and $\mathrm{x_z}$, we can present a single under-approximation as follows:

$$G(on_x \to F(on_z \vee \neg on_x)),$$
$$G((on_x \wedge on_z) \to (on_z W \neg on_x)),$$
$$G(\neg on_x \to F(\neg on_z \vee on_x)),$$
$$G((\neg on_x \wedge \neg on_z) \to (\neg on_z W on_x)).$$

The abstracted local propositions are x_y, x_z, y_z and on_y.

An over-approximation for AND version with $\mathrm{x_y} < \mathrm{x_z}$ is given below:

$$G(on_x \to F(on_y \vee \neg on_x)),$$
$$G((on_x \wedge on_y) \to (on_y W \neg on_x)),$$
$$G((on_x \wedge x_z) \to (x_z W \neg on_x)),$$
$$G(\neg on_x \to F(\neg x_z \vee on_x)),$$
$$G((\neg on_x \wedge \neg x_z) \to (\neg x_z \vee on_x)),$$
$$G((\neg on_x \wedge \neg on_y) \to (\neg on_y W on_x)),$$
$$G(x_z \to on_y),$$
$$G((on_y \wedge \neg y_z) \to F(\neg on_y \vee y_z)),$$
$$G((on_y \wedge y_z) \to (y_z W \neg on_y)),$$
$$G((\neg on_y \wedge y_z) \to F(on_y \vee \neg y_z)),$$
$$G((\neg on_y \wedge \neg y_z) \to (\neg y_z W on_y)),$$
$$G((x_z \wedge y_z) \leftrightarrow on_z).$$

Note that we abstracted one proposition x_y. In fact, this over-approximation satisfies both inclusions in the right hand side of Definition 12, which means the over-approximation does not accommodate any extra behaviour. The reason is that this 'over'-approximation is obtained by substituting x_y by on_y in the original weak specification. Due to the clause $G(x_y \leftrightarrow on_y)$, this approximation does not change the meaning of the formula.

An over-approximation for the OR version can be obtained by just changing $\wedge$-operator in the switching condition on gene z by $\vee$-operator.

Incoherent type 1 feed-forward loop Incoherent type 1 feed-forward loop (I1-FFL) is a pattern depicted in Fig. 24. Assume that the threshold of x for z is lower than that of x for y. Since x activates z, if x becomes ON, z will be turned ON. Then, if gene x keeps being expressed, y eventually becomes ON. Since y inhibits z, z will become OFF afterwards. As a result, this motif generates pulse-like dynamics on z.

The strong specification for this motif is as follows (the weak one can be obtained similarly as for the previous motifs):

$$G(on_x \to F(\neg on_x \vee x_z)),$$
$$G((on_x \wedge x_z) \to (x_z U(\neg on_x \vee x_y))),$$
$$G((on_x \wedge x_y) \to (x_y W \neg on_x)),$$
$$G(\neg on_x \to F(on_x \vee \neg x_y)),$$
$$G((\neg on_x \wedge \neg x_y) \to (\neg x_y U(on_x \vee \neg x_z))),$$

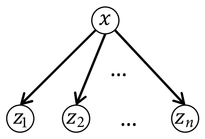

Fig. 25. Single-input module.

$$G((\neg on_x \wedge \neg x_z) \rightarrow (\neg x_z W on_x)),$$
$$G(x_y \rightarrow x_z),$$
$$G(x_y \leftrightarrow on_y),$$
$$G(on_y \rightarrow F(\neg on_y \vee y_z)),$$
$$G((on_y \wedge y_z) \rightarrow (y_z W \neg on_y)),$$
$$G(\neg on_y \rightarrow F(on_y \vee \neg y_z)),$$
$$G((\neg on_y \wedge \neg y_z) \rightarrow (\neg y_z W on_y)),$$
$$G((x_z \wedge \neg y_z) \leftrightarrow on_z).$$

Focusing on the pulse-like dynamics, we have the following under-approximation:

$$G((on_x \wedge \neg on_y) \rightarrow F(on_z \vee \neg on_x)),$$
$$G((on_x \wedge on_z) \rightarrow (on_z U on_y)),$$
$$G((on_x \wedge on_y) \rightarrow ((on_y \wedge \neg on_z) W \neg on_x)),$$
$$G(\neg on_x \rightarrow (\neg on_z \wedge \neg on_y)).$$

The abstracted local propositions are x_y, x_z and y_z.

An over-approximation for this motif is given below:

$$G(on_x \rightarrow F(\neg on_x \vee x_z)),$$
$$G((on_x \wedge x_z) \rightarrow (x_z W \neg on_x)),$$
$$G((on_x \wedge on_y) \rightarrow (on_y W \neg on_x)),$$
$$G(\neg on_x \rightarrow F(on_x \vee \neg on_y)),$$
$$G((\neg on_x \wedge \neg on_y) \rightarrow (\neg on_y W on_x)),$$
$$G((\neg on_x \wedge \neg x_z) \rightarrow (\neg x_z W on_x)),$$
$$G(on_y \rightarrow x_z),$$
$$G(on_y \rightarrow F(\neg on_y \vee y_z)),$$
$$G((on_y \wedge y_z) \rightarrow (y_z W \neg on_y)),$$
$$G(\neg on_y \rightarrow F(on_y \vee \neg y_z)),$$
$$G((\neg on_y \wedge \neg y_z) \rightarrow (\neg y_z W on_y)),$$
$$G((\neg x_z \wedge y_z) \leftrightarrow on_z).$$

We abstracted proposition x_y. Like the over-approximation for C1-FFL, this over-approximation in principle characterises the same behaviour set as the original weak specification. Thus this over-approximation can be seen as some kind of 'optimisation' of the weak specification for this motif.

Single-input module A single-input module is a pattern in which one regulator (called the master gene) regulates a group of target genes (Fig. 25). All regulations from the master gene are of the same type (positive or negative). We only consider the positive case but the negative case is similar.

The function of this motif is a last-in first-out (LIFO) temporal order on expressions of target genes. Assume that the thresholds for $\mathrm{z}_1, \mathrm{z}_2, \ldots, \mathrm{z}_\mathrm{n}$ occur in this ascending order. When the master regulator x is ON, the regulated genes $z_1, z_2, \ldots, z_n$ are turned ON in this order. When x is turned OFF, genes $z_n, z_{n-1}, \ldots z_1$ are turned OFF in this order.

For simplicity we set $n = 2$ in the following but generalisation is easy. The strong specification for this network motif is as follows (the weak one can be obtained similarly for other motifs):

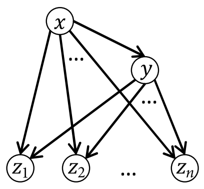

Fig. 26. Multi-output feed-forward loop.

$$G(on_x \to F(\neg on_x \vee x_{z_1})),$$
$$G((on_x \wedge x_{z_1}) \to (x_{z_1} U (\neg on_x \vee x_{z_2}))),$$
$$G((on_x \wedge x_{z_2}) \to (x_{z_2} W \neg on_x),$$
$$G(\neg on_x \to F(on_x \vee \neg x_{z_2})),$$
$$G((\neg on_x \wedge \neg x_{z_2}) \to (\neg x_{z_2} U (on_x \vee \neg x_{z_1}))),$$
$$G((\neg on_x \wedge \neg x_{z_1}) \to (\neg x_{z_1} W on_x)),$$
$$G(x_{z_2} \to x_{z_1}),$$
$$G(x_{z_1} \leftrightarrow on_{z_1}),$$
$$G(x_{z_2} \leftrightarrow on_{z_2}).$$

We present an under-approximation.

$$G(on_{z_2} \to on_{z_1}),$$
$$G(on_x \to (on_x U on_{z_2})),$$
$$G((on_x \wedge on_{z_1}) \to (on_{z_1} W \neg on_x)),$$
$$G((on_x \wedge on_{z_2}) \to (on_{z_2} W \neg on_x)),$$
$$G(\neg on_x \to (\neg on_x U \neg on_{z_1})),$$
$$G((\neg on_x \wedge \neg on_{z_2}) \to (\neg on_{z_2} W on_x)),$$
$$G((\neg on_x \wedge \neg on_{z_1}) \to (\neg on_{z_1} W on_x)).$$

The abstracted local propositions are x_{z_1} and x_{z_2}. Behaviours that satisfy this formula are such that once x is turned ON it remains ON until all target genes become active, and once x is turned OFF it remains OFF until all target genes become inactive. Therefore, behaviours such that x is turned OFF before all target genes become active or x is turned ON before all target genes become inactive are excluded from the possible behaviours obtained using the original specification.

We now present an over-approximation:

$$G(on_{z_2} \to on_{z_1}),$$
$$G((on_x \wedge on_{z_1}) \to (on_{z_1} W \neg on_x)),$$
$$G((on_x \wedge on_{z_2}) \to (on_{z_2} W \neg on_x)),$$
$$G((\neg on_x \wedge \neg on_{z_2}) \to (\neg on_{z_2} W on_x)),$$
$$G((\neg on_x \wedge \neg on_{z_1}) \to (\neg on_{z_1} W on_x)).$$

Propositions x_{z_1} and x_{z_2} are ignored. This over-approximation says that the temporal order of activation and inactivation of target genes is preserved but some genes may not be activated when x is turned ON or inactivated when x is turned OFF. Such behaviours are also allowed in the weak specification but we can specify the same constraint without local propositions. Note, however, that this over-approximation allows extra behaviours. The example is that first gene x, z_1 and z_2 are ON then gene x is turned OFF but gene z_1 and gene z_2 are still ON.

Multi-output feed-forward loop A multi-output feed-forward loop is a generalisation of a feed-forward loop with n target genes (Fig. 26). The function of this motif is interesting when each input function for z_i is OR and the threshold orders for x and y are inverted, that is, $\mathrm{x}_{z_1} < \mathrm{x}_{z_2} < \cdots < \mathrm{x}_{z_n}$ and $\mathrm{y}_{z_1} > \mathrm{y}_{z_2} > \cdots > \mathrm{y}_{z_n}$. Moreover, we assume that the threshold x_y is smaller than any of x_{z_i}s. In this case this motif can generate a first-in first-out (FIFO) temporal order on expression of target genes. The activation order is $z_1 z_2 \ldots z_n$ and the inactivation order is the opposite. This is because, when gene x becomes ON for sufficient time, gene y and z_1 to z_n will become ON in this order due to the order of thresholds x_{z_1} to x_{z_n}.

Since gene x is expressed sufficiently long, we assume that gene y is expressed at the highest expression level y_{z_1}. At this situation we turn gene x OFF. Then the expression level of gene x begins to decrease and finally will fall to a basal level. At this moment, genes z_1 to z_n are still ON since the level of gene y is highest for a while and activates z_1 to z_n. Since the activation effect of gene x to y disappears, the level of gene y begins to decrease and finally falls below the highest threshold y_{z_1} and gene z_1 is turned OFF (recall that the threshold order of gene y is the opposite of that of gene x). In this manner z_2 to z_n will be turned OFF in this order.

We show the strong specification of this network motif (we set $n = 2$ but generalisation is easy):

$$\begin{aligned}
&G(on_x \rightarrow F(\neg on_x \vee x_y)),\\
&G((on_x \wedge x_y) \rightarrow (x_y U(\neg on_x \vee x_{z_1}))),\\
&G((on_x \wedge x_{z_1}) \rightarrow (x_{z_1} U(\neg on_x \vee x_{z_2}))),\\
&G((on_x \wedge x_{z_2}) \rightarrow (x_{z_2} W \neg on_x)),\\
&G(\neg on_x \rightarrow F(on_x \vee \neg x_{z_2})),\\
&G((\neg on_x \wedge \neg x_{z_2}) \rightarrow (\neg x_{z_2} U(on_x \vee \neg x_{z_1}))),\\
&G((\neg on_x \wedge \neg x_{z_1}) \rightarrow (\neg x_{z_1} U(on_x \vee \neg x_y))),\\
&G((\neg on_x \wedge \neg x_y) \rightarrow (\neg x_y W on_x)),\\
&G(x_{z_2} \rightarrow x_{z_1}),\\
&G(x_{z_1} \rightarrow x_y),\\
&G(x_y \leftrightarrow on_y),\\
&G(on_y \rightarrow F(\neg on_y \vee y_{z_2})),\\
&G((on_y \wedge y_{z_2}) \rightarrow (y_{z_2} U(\neg on_y \vee y_{z_1}))),\\
&G((on_y \wedge y_{z_1}) \rightarrow (y_{z_1} W \neg on_y)),\\
&G(\neg on_y \rightarrow F(on_y \vee \neg y_{z_1})),\\
&G((\neg on_y \wedge \neg y_{z_1}) \rightarrow (\neg y_{z_1} U(on_y \vee \neg x_{z_2}))),\\
&G((\neg on_y \wedge \neg y_{z_2}) \rightarrow (\neg y_{z_2} W on_y),\\
&G(y_{z_1} \rightarrow y_{z_2}),\\
&G((x_{z_1} \vee y_{z_1}) \rightarrow on_{z_1}),\\
&G((\neg x_{z_1} \wedge \neg y_{z_1}) \rightarrow \neg on_{z_1}),\\
&G((x_{z_2} \vee y_{z_2}) \rightarrow on_{z_2}),\\
&G((\neg x_{z_2} \wedge \neg y_{z_2}) \rightarrow \neg on_{z_2}).
\end{aligned}$$

Focusing on the property mentioned above we have the following under-approximation:

$$\begin{aligned}
&G(on_x \rightarrow (on_{z_2} \rightarrow on_{z_1})),\\
&G(\neg on_x \rightarrow (\neg on_{z_2} \rightarrow \neg on_{z_1})),\\
&G((on_x \rightarrow (on_x U on_{z_1}))),\\
&G((on_x \wedge on_{z_1}) \rightarrow ((on_x \wedge on_{z_1}) U(on_x \wedge on_{z_2}))),\\
&G((on_x \wedge on_{z_2}) \rightarrow (on_{z_2} W \neg on_x)),\\
&G(\neg on_x \rightarrow (\neg on_x U \neg on_{z_1})),\\
&G((\neg on_x \wedge \neg on_{z_1}) \rightarrow ((\neg on_x \wedge \neg on_{z_1}) U(\neg on_x \wedge \neg on_{z_2}))),\\
&G((\neg on_x \wedge \neg on_{z_2}) \rightarrow (\neg on_{z_2} W on_x)).
\end{aligned}$$

The abstracted local propositions are x_y, x_{z_1}, x_{z_2}, on_y, y_{z_1} and y_{z_2}. This formula says that when x is turned ON, z_1 and z_2 are activated in this order, and that when x is turned OFF, z_1 and z_2 are inactivated in the opposite order.

An over-approximation is given as follows:

Table 4
The result of analyses for the network in Fig. 15.

	S	E	P	T	T(ET)
D	33	682	12	0.020 s	<0.001 s
M	31	592	7	0.143 s	<0.001 s
A(under)	25	396	8	0.018 s	<0.001 s
A(over)	33	692	8	0.020 s	<0.001 s
MA(under)	25	396	6	0.152 s	<0.001 s
MA(over)	33	664	6	0.160 s	<0.001 s

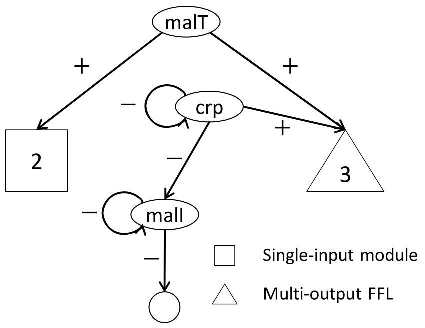

Fig. 27. A network in *E. coli*.

$$G(((on_x \vee on_y) \wedge on_{z_1}) \rightarrow (on_{z_1} W (\neg on_x \vee \neg on_y))),$$
$$G(((on_x \vee on_y) \wedge on_{z_2}) \rightarrow (on_{z_2} W (\neg on_x \vee \neg on_y))),$$
$$G((\neg on_x \wedge \neg on_y \wedge \neg on_{z_1}) \rightarrow (\neg on_{z_1} W (on_x \vee on_y))),$$
$$G((\neg on_x \wedge \neg on_y \wedge \neg on_{z_2}) \rightarrow (\neg on_{z_2} W (on_x \vee on_y))).$$

This specification does not ensure that gene z_1 and z_2 will be ON even if gene x or y are ON. However, if they become ON, they keep being ON as long as both gene x and y are ON. Similarly the fact that both gene x and y are OFF does not force gene z_1 and z_2 to be OFF. However, once they are turned OFF, they keep being OFF as long as both gene x and y are OFF.

5.3. Result and discussion

We demonstrate the approximate analysis method developed thus far and compare it with the direct analysis method and modular analysis method developed in Section 4. The combination of the modular and approximate method is also experimented.

We use the same examples as in the experiments of modular analysis (Section 4.4), including the results of direct and modular analysis in the tables for comparison.

The first example is the network depicted in Fig. 15. There are two motifs in the network, one a negative auto-regulation and the other a single-input module. We show the results in Table 4. In the table, 'D' represents direct analysis, 'M' modular, 'A' approximate and 'MA' the combination of modular and approximate. Approximate analysis is performed for both under- and over-approximation. 'S' represents the number of states of the automaton, 'E' the number of edges of the automaton, 'P' the number of propositions and 'T' the total analysis time (second). 'T(ET)' represents the time of empty-testing of the automaton.

The second example is the network in *Escherichia coli* involving the *malT* gene. We redraw this network in Fig. 27, emphasising network motifs. The numbers in the box and the triangle are the numbers of target genes in each motif. In this network we approximate two negative auto-regulations, one single-input module and one multi-output feed-forward loop. The results are shown in Table 5. The row 'M' is 'Modular (B)' in Table 2, which was the best among the divisions.

The third example is the network from *Arabidopsis thaliana* depicted in Fig. 21. In this network, we can find a single-input module which has 18 target genes in the motif. There are some genes which have regulators other than the master gene. Thus we cannot use the approximate specification introduced above directly. Fortunately the modification is easy. We do not omit propositions x_y if y has a regulator other than the master gene x, and specify the conditions for activation and inhibition of y the same as the original ones. The results are shown in Table 6. The row 'M' is 'Modular (A)' in Table 3, which was the best among the divisions.

In all cases, the cost of analysis is improved using approximate analysis, particularly for the larger networks. Compared to modular analysis, approximate analysis is much efficient since we do not have to intersect automata. The modular analysis, however, has a merit that we can apply it for any network which does not contain network motifs.

Table 5
The results of analyses of the network in Fig. 27.

	S	E	P	T	T(ET)
D	8281	14 698 200	28	28.054 s	26.474 s
M	8281	14 698 200	9	6.579 s	3.164 s
A(under)	865	233 118	16	0.111 s	0.068 s
A(over)	2305	3 085 344	17	0.606 s	0.516 s
MA(under)	865	201 294	9	0.410 s	0.044 s
MA(over)	2113	659 832	10	0.567 s	0.148 s

Table 6
The results of analyses of the network in Fig. 21.

	S	E	P	T	T(ET)
D	3041	4 395 400	46	3.092 s	2.272 s
M	2961	3 775 532	20	2.432 s	0.648 s
A(under)	1281	549 492	29	0.325 s	0.180 s
A(over)	1601	899 182	30	0.448 s	0.296 s
MA(under)	1281	498 828	19	1.120 s	0.096 s
MA(over)	1921	1 038 784	19	0.996 s	0.192 s

Using both modular and approximate analysis is not more efficient than the approximate analysis in all the examples except for over-approximation in the second example. The reason is that the approximate specifications are not so large in these examples. Therefore the cost of intersecting automata is relatively high in these experiments. Note, however, that the time of emptiness testing is reduced by modular&approximate analysis compared to mere approximate analysis. If the network is large, we may benefit from the combination of modular analysis and approximate analysis.

6. Related work

In this section, we describe some other qualitative methods for biological systems.

BIOCHAM [17] is a language and programming environment for modelling and simulating biochemical systems, and checking their temporal properties. Reactions are written as rules like `A+B=>C`, and simulations are performed by replacing objects on the left-hand side with those on the right-hand side. Since there are many possible rules that can be executed in each state, there are many possible successor states for each state depending on the rule applied. After simulation, we have a non-deterministic transition graph whose nodes are possible states and edges are state transitions. The set of possible behaviours of the simulation over-approximates the set of all behaviours of the system which varies depending on the kinetic parameters. A biological property is written in computation tree logic (CTL), a type of branching time logic, and checked in the resulting transition graph. In BIOCHAM, presence or absence of objects is the only matter considered. On the contrary, objects have a level of concentration in our model. In addition, the description of behaviour is more expressive and flexible than the rule description in BIOCHAM, where we can only specify the next state. For example, the rule `A + B => C` indicates that if the objects `A` and `B` appear, then the object `C` appears in the *next state*. In our method, however, we can specify a more flexible rule such as ‘if the objects `A` and `B` appear, then the object `C` appears in *some future state*’. Moreover, we can specify the temporal order of events, such as ‘if the objects `A` and `B` appear, the object `C` appears first and then the object `D` appears’.

SMBioNet [9] is a tool for formally analysing temporal properties of gene regulatory networks. In SMBioNet, genes have concentration thresholds for activation or inhibition of each of their regulating genes. A configuration of systems is represented as a vector of expression values, which are segmented by thresholds. For example, if a gene has two thresholds, then it has three levels – 0, 1, and 2. Behaviours of a network are captured as a transition system on the vectors of values for genes in the network. The temporal evolution of systems is described by transition functions on the vectors. Temporal properties are described in CTL, and verification of them is performed by model-checking [12] on the resulting transition systems. The description of behaviours in our system is more expressive than the functional description in SMBioNet since SMBioNet considers only expression levels of genes. In contrast our framework considers, in addition, the modes of genes (ON/OFF). Moreover, the state transitions of SMBioNet are restricted to those of Hamming distance of 1, that is to say, only transitions from state (00) to (10) or (01) are allowed.

GNA [14] is a computational tool for the modelling and simulation of gene regulatory networks. GNA achieves simulation using piecewise linear differential equation models and generates state transition systems that represent possible behaviours of networks. The qualitative dynamics of a system are completely determined by inequality constraints defining the ordering between thresholds and stable equilibria of the system. Network properties of interest are checked automatically using model checking [8]. This method assumes that the functions of multivariate regulation are known since the piecewise linear differential equations are approximations of ordinary differential equations. However, the majority of such functions in any organism are unknown [3]. On the contrary, our method does not need information about such functions, and is therefore more applicable for the current databases of gene regulation such as Reactome[6] [13], GeneCards[7] [41], Metacyc[8][31], Ingenuity® Knowledge Base,[9] and KEGG[10] [30].

Although the above tools are useful for checking whether a biological property can be true in network behaviours, it is unknown how to introduce techniques to improve efficiency.

As we can see from the above studies, temporal logic is useful in analysing biological systems. Other studies using temporal logic to analyse biological systems includes [10,4,34]. Systems are usually modelled by (possibly an approximate form of) ordinary differential equations. Behaviours of biological systems are abstracted to state transition systems or automata after simulating the differential equations. Temporal logic is only used to describe the biological properties to be checked through the model checking approach [12]. That is, these methods are based on a quantitative description of behaviours.

RoVerGeNe [7] aims to analyse behaviours of gene networks under parameter uncertainty or search for valid parameter sets to satisfy a given property. Behaviours are modelled as piecewise-multiaffine differential equations in which they assume regulation functions and threshold values are known. Their formalism is intermediate of quantitative and qualitative approach. They need some quantitative information but can perform fine-grained analysis compared to purely qualitative approach.

Qualitative networks [42] are an extension of Boolean networks [33,46] in which biological components (e.g. genes and proteins) have multiple level. The level of each component is updated by the associated target function which represents the cumulative influence of the activation and inhibition from other components. At each step, the level of each component only changes by a single level. Biological properties are verified by analysing the steady state of these networks. Thus properties considered in Qualitative networks are not temporal properties but just the relationships on expression levels of molecules. For efficient analysis of large networks, modular computation of possible behaviours is proposed and implemented in Qualitative networks. In Qualitative networks, careful and complicated modelling is needed to capture the asynchronous non-determinism of a system, which is a common problem in analysing a system which contains different time scale reactions. In our approach, by focusing on the causal relationships, we do not care for time scale of biological reactions. Moreover, the biological properties checked in our approach is not limited to those in steady state but any temporal property. Another difference is that the network behaviour is deterministic in Qualitative networks, i.e. the model of the network only describes a single behaviour. In our framework we model all possible behaviours of a network.

Our approach differs from process algebraic approaches such as stochastic π-calculus [38] or Bio-PEPA [11]. These models are based on biochemical reactions, and each molecule is modelled by a process. Modelled systems are simulated stochastically or translated into ordinary differential equations, continuous time Markov chains or PRISM [26,25] models. We need precise molecular mechanisms and quantitative information on parameter variables to give a model, and thus these approaches are also quantitative. In contrast, our approach only assumes information about activation and inhibition relationships between genes and therefore is more applicable. If we have more information such as the ordering of thresholds or biases in multivariate regulation, or timing or causal relationship of activation or inhibition (e.g. 'gene a and b are known to be activated simultaneously' or 'gene a must precede gene b'), we can incorporate such information into our temporal specification of a system.

7. Conclusion

In this article, we have presented a method for analysing the dynamics of gene regulatory networks using LTL satisfiability checking. To ease analysis of large networks, we developed the modular analysis method and the approximate analysis method. Experimental results show that both methods are efficient in analysing large networks.

There are several future directions in our qualitative framework.

An interesting future direction is deriving an additional constraint ϕ' to force all behaviours of a gene regulatory network to satisfy some observed property ψ. This may facilitate finding meaningful biological facts such as the order of thresholds, regulation biases of multiple regulation or possibility of hidden nodes in networks. To achieve this, we need a method that extracts useful information from automata that are generally huge as shown in Section 5.3, since they represent possible behaviours of networks.

As to the modular analysis, developing a light simplification algorithm of Büchi automata will be an important work. In modular analysis, we just abstracted local propositions from automata for subnetwork specifications. If we do not take costs into consideration, we can simplify the automata using polynomial time simulation-based algorithm [45]. This powerful simplification considerably reduces the size of automata. Since small automata are desirable for intersection, we can apply this simplification algorithm to the automata for subnetwork specifications. From the experiments which are not reported in this article, however, the total analysis time is not necessarily improved due to the extra polynomial cost of simplification. However, if we have a 'light' simplification algorithm, we may reduce the cost of the modular analysis through automata

[6] http://www.reactome.org/ReactomeGWT/entrypoint.html.
[7] http://www.genecards.org/.
[8] http://metacyc.org/.
[9] http://www.ingenuity.com/science/knowledge_base.html.
[10] http://www.genome.jp/kegg/.

simplification. In this regard, we should keep in mind that if we are only interested in non-emptiness of automata, we can apply a more aggressive simplification in which the accepting language may be changed while non-emptiness is preserved.

As to the approximate analysis, we need to study more network motifs and find approximate specifications for them.

We are now at a stage to consider applications of our method to real problems in biology, biomedicine, pathology, pharmacology, and so on. Through such application, we will find the necessity of further development and extension of our logical framework.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Number 26730153. We would like to express our gratitude to anonymous reviewers for giving us many suggestions to improve the manuscript.

References

[1] M. Abadi, L. Lamport, P. Wolper, Realizable and unrealizable specifications of reactive systems, in: ICALP '89: Proceedings of the 16th International Colloquium on Automata, Languages and Programming, in: LNCS, vol. 372, Springer-Verlag, London, UK, 1989, pp. 1–17.
[2] U. Alon, An Introduction To Systems Biology: Design Principles Of Biological Circuits, Chapman and Hall/CRC, 2006.
[3] U. Alon, Network motifs: theory and experimental approaches, Nat. Rev. Genet. 8 (6) (June 2007) 450–461.
[4] M. Antoniotti, A. Policriti, N. Ugel, B. Mishra, Model building and model checking for biochemical processes, Cell Biochem. Biophys. 38 (3) (2003) 271–286.
[5] T. Aoshima, K. Sakuma, N. Yonezaki, An efficient verification procedure supporting evolution of reactive system specifications, in: Proceedings of the 4th International Workshop on Principles of Software Evolution, IWPSE '01, ACM, New York, NY, USA, 2001, pp. 182–185.
[6] T. Aoshima, N. Yonezaki, An efficient tableau-based verification method with partial evaluation for reactive system specifications, in: Proceedings of the 10th European–Japanese Conference on Information Modelling and Knowledge Bases, 2000, pp. 363–374.
[7] G. Batt, C. Belta, R. Weiss, Temporal logic analysis of gene networks under parameter uncertainty, IEEE Trans. Automat. Control 53 (2008) 215–229.
[8] G. Batt, D. Ropers, H. de Jong, J. Geiselmann, R. Mateescu, M. Page, D. Schneider, Validation of qualitative models of genetic regulatory networks by model checking: analysis of the nutritional stress response in *Escherichia coli*, Bioinformatics 21 (Suppl. 1) (2005) i19–i28.
[9] G. Bernot, J.P. Comet, A. Richard, J. Guespin, Application of formal methods to biological regulatory networks: extending Thomas' asynchronous logical approach with temporal logic, J. Theoret. Biol. 229 (3) (2004) 339–347.
[10] N. Chabrier, F. Fages, Symbolic model checking of biochemical networks, in: Computational Methods in Systems Biology, CMSB' 03, in: LNCS, vol. 2602, Springer-Verlag, 2003, pp. 149–162.
[11] F. Ciocchetta, J. Hillston, Bio-PEPA: a framework for the modelling and analysis of biological systems, Theoret. Comput. Sci. 410 (August 2009) 3065–3084.
[12] E. Clarke, O. Grumberg, D. Peled, Model Checking, MIT Press, 1999.
[13] D. Croft, G. O'Kelly, G. Wu, R. Haw, M. Gillespie, L. Matthews, M. Caudy, P. Garapati, G. Gopinath, B. Jassal, S. Jupe, I. Kalatskaya, S. Mahajan, B. May, N. Ndegwa, E. Schmidt, V. Shamovsky, C. Yung, E. Birney, H. Hermjakob, P. D'Eustachio, L. Stein, Reactome: a database of reactions, pathways and biological processes, Nucleic Acids Res. 39 (Database-Issue) (2011) 691–697.
[14] H. de Jong, J. Geiselmann, G. Hernandez, M. Page, Genetic network analyzer: qualitative simulation of genetic regulatory networks, Bioinformatics 19 (3) (2003) 336–344.
[15] M.B. Elowitz, S. Leibler, A synthetic oscillatory network of transcriptional regulators, Nature 403 (2000) 335–338.
[16] E. Allen Emerson, Temporal and modal logic, in: Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B), MIT Press, 1990, pp. 995–1072.
[17] F. Fages, S. Soliman, N. Chabrier-Rivier, Modelling and querying interaction networks in the biochemical abstract machine BIOCHAM, J. Biol. Phys. Chem. 4 (2004) 64–73.
[18] L. Glass, S.A. Kauffman, Co-operative components, spatial localization and oscillatory cellular dynamics, J. Theoret. Biol. 34 (2) (February 1972) 219–237.
[19] L. Glass, S.A. Kauffman, The logical analysis of continuous, non-linear biochemical control networks, J. Theoret. Biol. 39 (1) (April 1973) 103–129.
[20] J.R.W. Govan, V. Deretic, Microbial pathogenesis in cystic fibrosis: mucoid *Pseudomonas aeruginosa* and *Burkholderia cepacia*, Microbiol. Rev. 60 (1996) 539–574.
[21] J.R.W. Govan, G.S. Harris, *Pseudomonas aeruginosa* and cystic fibrosis: unusual bacterial adaptation and pathogenesis, Microbiol. Sci. 3 (10) (1986) 302–308.
[22] J. Guespin, G. Bernot, J.P. Comet, A. Mérieau, A. Richard, C. Hulen, B. Polack, Epigenesis and dynamic similarity in two regulatory networks in *Pseudomonas aeruginosa*, Acta Biotheor. 52 (4) (2004) 379–390.
[23] J. Guespin, M. Kaufman, Positive feedback circuits and adaptive regulations in bacteria, Acta Biotheor. 49 (4) (2001) 207–218.
[24] S. Hagihara, N. Yonezaki, Completeness of verification methods for approaching to realizable reactive specifications, in: Proceedings of 1st Asian Working Conference on Verified Software, AWCVS'06, vol. 348, 2006, pp. 242–257, UNU-IIST technical report.
[25] J. Heath, M. Kwiatkowska, G. Norman, D. Parker, O. Tymchyshyn, Probabilistic model checking of complex biological pathways, Theoret. Comput. Sci. 391 (February 2008) 239–257.
[26] A. Hinton, M. Kwiatkowska, G. Norman, D. Parker, PRISM: a tool for automatic verification of probabilistic systems, in: H. Hermanns, J. Palsberg (Eds.), Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'06, in: LNCS, vol. 3920, Springer, 2006, pp. 441–444.
[27] S. Ito, T. Ichinose, M. Shimakawa, N. Izumi, S. Hagihara, N. Yonezaki, Modular analysis of gene networks by linear temporal logic, J. Integr. Bioinform. 10 (2) (2013).
[28] S. Ito, T. Ichinose, M. Shimakawa, N. Izumi, S. Hagihara, N. Yonezaki, Qualitative analysis of gene regulatory networks using network motifs, in: Proceedings of the 4th International Conference on Bioinformatics Models, Methods and Algorithms, BIOINFORMATICS2013, 2013, pp. 15–24.
[29] S. Ito, N. Izumi, S. Hagihara, N. Yonezaki, Qualitative analysis of gene regulatory networks by satisfiability checking of linear temporal logic, in: Proceedings of the 10th IEEE International Conference on Bioinformatics & Bioengineering, BIBE2010, 2010, pp. 232–237.
[30] M. Kanehisa, S. Goto, Y. Sato, M. Furumichi, M. Tanabe, KEGG for integration and interpretation of large-scale molecular data sets, Nucleic Acids Res. 40 (2012) D109–D114.
[31] P.D. Karp, M. Riley, S.M. Paley, A. Pellegrini-Toole, The MetaCyc database, Nucleic Acids Res. 30 (1) (2002) 59–61.
[32] M. Kaufman, J. Urbain, R. Thomas, Towards a logical analysis of the immune response, J. Theoret. Biol. 114 (4) (1985) 527–561.
[33] S.A. Kaufman, Metabolic stability and epigenesis in randomly constructed genetic nets, J. Theoret. Biol. 22 (3) (1969) 437–467.

[34] M. Kwiatkowska, G. Norman, D. Parker, Using probabilistic model checking in systems biology, SIGMETRICS Perf. Eval. Rev. 35 (4) (2008) 14–21.
[35] R. Mori, N. Yonezaki, Several realizability concepts in reactive objects, in: Information Modeling and Knowledge Bases IV, 1993, pp. 407–424.
[36] D. Peled, T. Wilke, Stutter-invariant temporal properties are expressible without the next-time operator, Inform. Process. Lett. 63 (5) (1997) 243–246.
[37] A. Pnueli, R. Rosner, On the synthesis of a reactive module, in: POPL '89: Proceedings of the 16th ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages, ACM, New York, NY, USA, 1989, pp. 179–190.
[38] C. Priami, A. Regev, E. Shapiro, W. Silverman, Application of a stochastic name-passing calculus to representation and simulation of molecular processes, Inform. Process. Lett. 80 (October 2001) 25–31.
[39] A. Rabinovich, On translations of temporal logic of actions into monadic second-order logic, Theoret. Comput. Sci. 193 (February 1998) 197–214.
[40] K.Y. Rozier, M.Y. Vardi, LTL satisfiability checking, in: Proceedings of the 14th International SPIN Conference on Model Checking Software, in: LNCS, vol. 4595, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 149–167.
[41] M. Safran, I. Dalah, J. Alexander, N. Rosen, T. Iny Stein, M. Shmoish, N. Nativ, I. Bahir, T. Doniger, H. Krug, A. Sirota-Madi, T. Olender, Y. Golan, G. Stelzer, A. Harel, D. Lancet, GeneCards Version 3: the human gene integrator, Database, 2010(0):baq020+, August 2010.
[42] M.A. Schaub, T.A. Henzinger, J. Fisher, Qualitative networks: a symbolic approach to analyze biological signaling networks, BMC Syst. Biol. 1 (4) (2007).
[43] M.J. Schurr, D.W. Martin, M.H. Mudd, V. Deretic, Gene cluster controlling conversion to alginate-overproducing phenotype in *Pseudomonas aeruginosa*: functional analysis in a heterologous host and role in the instability of mucoidy, J. Bacteriol. 176 (1994) 3375–3382.
[44] A.P. Sistla, E.M. Clarke, The complexity of propositional linear temporal logics, J. ACM 32 (July 1985) 733–749.
[45] F. Somenzi, R. Bloem, Efficient Büchi automata from LTL formulae, in: Proceedings of the 12th International Conference on Computer Aided Verification, in: LNCS, vol. 1855, Springer-Verlag, London, UK, 2000, pp. 248–263.
[46] R. Thomas, Boolean formalization of genetic control circuits, J. Theoret. Biol. 42 (3) (1973) 563–585.
[47] R. Thomas, M. Kaufman, Multistationarity, the basis of cell differentiation and memory. II. Logical analysis of regulatory networks in terms of feedback circuits, Chaos 11 (1) (2001) 180–195.
[48] W. Thomas, Automata on infinite objects, in: Jan van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B), MIT Press, 1990, pp. 133–164.
[49] V. Vanitha, K. Yamashita, K. Fukuzawa, N. Yonezaki, A method for structuralisation of evolutional specifications of reactive systems, in: ICSE 2000, The Third International Workshop on Intelligent Software Engineering, WISE3, 2000, pp. 30–38.
[50] M.Y. Vardi, P. Wolper, Reasoning about infinite computations, Inform. and Comput. 115 (November 1994) 1–37.