Model Checking Gene Regulatory Networks ⋆

Mirco Giacobbe*, Călin C. Guet*, Ashutosh Gupta*†,
Thomas A. Henzinger*, Tiago Paixão*, and Tatjana Petrov*

IST Austria, Austria* TIFR, India†

Abstract. The behaviour of gene regulatory networks (GRNs) is typically analysed using simulation-based statistical testing-like methods. In this paper, we demonstrate that we can replace this approach by a formal verification-like method that gives higher assurance and scalability. We focus on Wagner's weighted GRN model with varying weights, which is used in evolutionary biology. In the model, weight parameters represent the gene interaction strength that may change due to genetic mutations. For a property of interest, we synthesise the constraints over the parameter space that represent the set of GRNs satisfying the property. We experimentally show that our parameter synthesis procedure computes the mutational robustness of GRNs –an important problem of interest in evolutionary biology– more efficiently than the classical simulation method. We specify the property in linear temporal logics. We employ symbolic bounded model checking and SMT solving to compute the space of GRNs that satisfy the property, which amounts to synthesizing a set of linear constraints on the weights.

1 Introduction

Gene regulatory networks (GRNs) are one of the most prevalent and fundamental type of biological networks whose main actors are genes regulating other genes. A topology of a GRN is represented by a graph of interactions among a finite set of genes, where nodes represent genes, and edges denote the type of regulation (activation or repression) between the genes, if any. In [21], Wagner introduced a simple but useful model for GRNs that captures important features of GRNs. In the model, a system state specifies the activity of each gene as a Boolean value. The system is executed in discrete time steps, and all gene values are synchronously and deterministically updated: a gene active at time n affects the value of its neighbouring genes at time $n+1$. This effect is modelled through two kinds of parameters: *threshold* parameters assigned to each gene, which specify

⋆ This research was supported by the European Research Council (ERC) under grant 267989 (QUAREM), the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE) and Z211-N23 (Wittgenstein Award), the European Union's SAGE grant agreement no. 618091, ERC Advanced Grant ERC-2009-AdG-250152, the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement no. 291734, and the SNSF Early Postdoc.Mobility Fellowship, the grant number P2EZP2_148797.

the strength necessary to sustain the gene's activity, and *weight* parameters assigned to pairs of genes, which denote the strength of their directed effect.

Some properties of GRNs can be expressed in linear temporal logic (LTL) (such as reaching a steady-state), where atomic propositions are modelled by gene values. A single GRN may or may not satisfy a property of interest. Biologists are often interested in the behavior of *populations of GRNs*, and in presence of environmental perturbations. For example, the parameters of GRNs from a population may change from one generation to another due to mutations, and the distribution over the different GRNs in a population changes accordingly. We refer to the set of GRNs obtained by varying parameters on a fixed topology as *GRN space*. For a given population of GRNs instantiated from a GRN space, typical quantities of interest refer to the long-run average behavior. For example, *robustness* refers to the averaged satisfiability of the property within a population of GRNs, after an extended number of generations. In this context, Wagner's model of GRN has been used to show that mutational robustness can gradually evolve in GRNs [10], that sexual reproduction can enhance robustness to recombination [1], or to predict the phenotypic effect of mutations [17]. The computational analysis used in these studies relies on explicitly executing GRNs, with the purpose of checking if they satisfy the property. Then, in order to compute the robustness of a population of GRNs, the satisfaction check must be performed repeatedly for many different GRNs. In other words, robustness is estimated by statistically sampling GRNs from the GRN space and executing each of them until the property is (dis)proven. In this work, we pursue formal analysis of Wagner's GRNs which allows to avoid repeated executions of GRNs, and to compute mutational robustness with higher assurance and scalability.

In this paper, we present a novel method for synthesizing the space of parameters which characterize GRNs that satisfy a given property. These constraints eliminate the need of explicitly executing the GRN to check the satisfaction of the property. Importantly, the synthesized parameter constraints allow to efficiently answer questions that are very difficult or impossible to answer by simulation, e.g. emptiness check or parameter sensitivity analysis. In this work, we chose to demonstrate how the synthesized constraints can be used to compute the robustness of a population of GRNs with respect to genetic mutations. Since constraint evaluation is usually faster than executing a GRN, the constraints pre-computation enables faster computation of robustness. This further allows to compute the robustness with higher precision, within the same computational time. Moreover, it sometimes becomes possible to replace the statistical sampling with the exact computation of robustness.

In our method, for a given GRN space and LTL property, we used SMT solving and bounded model checking to generate a set of constraints such that a GRN satisfies the LTL property if and only if its weight parameters satisfy the constraints. The key insight in this method is that the obtained constraints are complex Boolean combinations of linear inequalities. Solving linear constraints has been the focus of both industry and academia for some time. However, the technology for solving linear constraints with Boolean structure, namely SMT

solving, has matured only in the last decade [3]. This technology has enabled us to successfully apply an SMT solver to generate the desired constraints.

We have built a tool which computes the constraints for a given GRN space and a property expressed in a fragment of LTL. In order to demonstrate the effectiveness of our method, we computed the robustness of five GRNs listed in [8], and for three GRNs known to exhibit oscillatory behavior. We first synthesized the constraints and then we used them to estimate robustness based on statistical sampling of GRNs from the GRN space. Then, in order to compare the performance with the simulation-based methods, we implemented the approximate computation of robustness, where the satisfiability of the property is verified by executing the GRNs explicitly. The results show that in six out of eight tested networks, the pre-computation of constraints provides greater efficiency, performing up to three times faster than the simulation method.

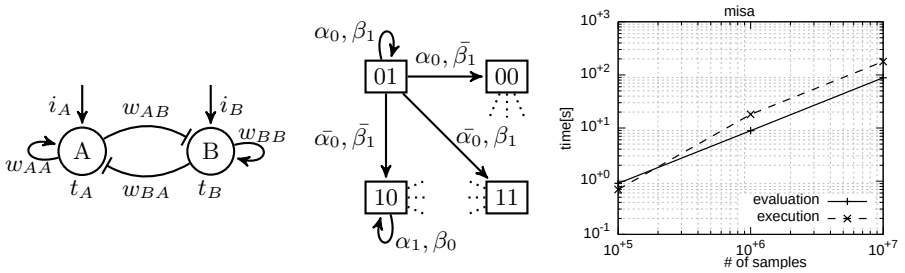
Related Work Formal verification techniques are already used for aiding various aspects of biological research [12, 16, 15, 22]. In particular, the robustness of models of biochemical systems with respect to temporal properties has been studied [18, 6, 19]. Our work is, to the best of our knowledge, the first application of formal verification to studying the evolution of mutational robustness of gene regulatory networks and, as such, it opens up a novel application area for the formal verification community. As previously discussed, with respect to related studies in evolutionary biology, our method can offer a higher degree of assurance, more accuracy, and better scalability than the traditional, simulation-based approaches. In addition, while the mutational robustness has been studied only for invariant properties, our method allows to compute the mutational robustness for non-trivial temporal properties that are expressible in LTL, such as bistability or oscillations between gene states.

1.1 Motivating example

In the following, we will illustrate the main idea of the paper on an example of a GRN space T generated from the GRN network shown in Fig. 1(a). Two genes A and B inhibit each other, and both genes have a self-activating loop. The parameters (i_A, i_B) represent constant inputs, which we assume to be regulated by some other genes that are not presented in the figure. Each of the genes is assigned a threshold value (t_A, t_B), and each edge is assigned a weight (w_{AA},w_{AB}, w_{BA}, w_{BB}). The dynamics of a GRN-individual chosen from T depends on these parameters. Genes are in either active or inactive state, which we represent with Boolean variables. For a given initial state of all genes, and for fixed values of weights and thresholds, the values of all genes evolve deterministically in discrete time-steps that are synchronized over all genes. Let a (resp. b) be the Boolean variable representing the activity of gene A (resp. B). We denote a GRN state by a pair (a, b). Let τ be the function that governs the dynamics of $\mathcal{G}$ (see Def. 3):

$$\tau(a, b) = (i_A + aw_{AA} - bw_{BA} > t_A,\ i_B + bw_{BB} - aw_{AB} > t_B)$$

The next state of a gene is the result of arithmetically adding the influence from other active genes.



(a) (b) (c)

Fig. 1: Motivating example. a) The topology of a GRN with mutual inhibition of genes A and B. b) The labelled transition system where labels denote the linear constraints which enable the transition. α_0, α_1, β_0, and β_1 denote linear atomic formulas $i_A - w_{BA} \leq t_A$, $i_B + w_{BB} > t_B$, $i_B - w_{AB} \leq t_B$, and $i_A + w_{AA} > t_A$ respectively. c) Run-times comparison between execution method (dashed lines) and our evaluation method (solid lines).

The topology of mutually inhibiting pair of genes is known to be bistable: whenever one gene is highly expressed and the other is barely expressed, the system remains stable [13, 19]. The bistability property can be written as the following LTL formula (see Def. 4):

$$(A \wedge \neg B \implies \Box(A \wedge \neg B)) \wedge (\neg A \wedge B \implies \Box(\neg A \wedge B)).$$

Let us fix values for parameters $t_A = t_B = 0.6$, $w_{AB} = w_{BA} = w_{BB} = 0.3$, and $i_A = i_B = \frac{2}{3}$. Then, we can check that a GRN is bistable by executing the GRN. Indeed, for the given parameter values, the states $(0, 1)$ and $(1, 0)$ are fixed points of τ. In other words, the GRN with those parameter values have two stable states: if they start in state $(0, 1)$ (resp. $(1, 0)$), they remain there. Now let us choose $i_A = \frac{2}{3}$, $i_B = \frac{1}{3}$. Again, by executing the GRN, we can conclude that it does not satisfy the property: at state $(0, 1)$, B does not have sufficiently strong activation to surpass its threshold and the system jumps to $(0, 0)$. Intuitively, since the external activation of B is too small, the phenotype has changed to a single stable state. In general, it is not hard to inspect that the bistability property will be met by any choice of parameters satisfying the following constraints:

$$\{i_A - w_{BA} \leq t_A,\ i_A + w_{AA} > t_A,\ i_B - w_{AB} \leq t_B,\ i_B + w_{BB} > t_B\}. \qquad (1)$$

Let's now suppose that we want to compute the robustness of T in presence of variations on edges due to mutations. Then, for each possible value of parameters, one needs to verify if the respective GRN satisfies the property. Using the constraints (1), one may verify GRNs without executing them.

Our method automatizes this idea to any given GRN topology and any property specified in LTL. We first encode T as a parametrized labelled transition system, partly shown in Fig. 1(b). Our implementation does not explicitly construct this transition system, nor executes the GRNs (the implementation is

described in Section 5). Then, we apply symbolic model checking to compute the constraints which represent the space of GRN's from T that satisfy the bi-stability property.

To illustrate the scalability of our method in comparison with the standard methods, in Fig. 1(c), we compare the performance of computing the mutational robustness with and without precomputing the constraints (referred to as *evaluation* and *execution* method respectively). We choose a mutation model such that each parameter takes 13 possible values distributed according to the binomial distribution (see Appendix in [14] for more details on the mutation model). We estimate the robustness value by statistical sampling of the possible parameter values. For a small number of samples, our method is slower because we spend extra time in computing the constraints. However, more samples may be necessary for achieving the desired precision. As the number of samples increases, our method becomes faster, because each evaluation of the constraints is two times faster than checking bistability by executing GRN-individuals. For 1.2×10^5 many simulations, execution and evaluation methods take same total time, and the robustness value estimated from these many samples lies in the interval $(0.8871, 0.8907)$ with 95% confidence. Hence, for this GRN, if one needs better precision for the robustness value, our method is preferred.

One may think that for this example, we may compute exact robustness because the number of parameter values is only 13^6 (four weights and two inputs). For simplicity of illustration, we chose this example, and we later present examples with a much larger space of parameters, for which exact computation of robustness is infeasible.

2 Preliminaries

In this section, we start by defining a *GRN space*, which will serve to specify common features for GRNs from the same population. These common features are types of gene interactions (topology), constant parameters (thresholds), and ranges of parameter values that are subject to some environmental perturbation (weights). Then, we formally introduce a model of an individual GRN from the GRN space and temporal logic to express its properties.

2.1 GRN space

The key characteristics of the behaviour of a GRN are typically summarised by a directed graph where nodes represent genes and edges denote the type of regulation between the genes. A regulation edge is either *activation* (one gene's activity increases the activity of the other gene) or *repression* (one gene's activity decreases the activity of the other gene) [20]. In Wagner's model of a GRN, in addition to the activation types between genes, each gene is assigned a *threshold* and each edge (pair of genes) is assigned a *weight*. The threshold of a gene models the amount of activation level necessary to sustain activity of the gene.

The weight on an edge quantifies the influence of the source gene on destination gene of the edge by means of a non-negative rational number.

We extend the Wagner's model by allowing a range of values for weight parameters. We call our model GRN space, denoting that all GRNs instantiated from that space share the same topology, and their parameters fall into given ranges. We assume that each gene always has some minimum level of expression without any external influence. In the model, this constant input is incorporated by a special gene which is always active, and activates all other genes from the network. The weight on the edge between the special gene and some other gene represents the minimum level of activation. The minimal activation is also subject to perturbation.

Definition 1 (GRN space) A gene regulatory network space is given by a tuple $T = (G, g_{in}, \rightarrow, \dashv, t, w^{max}, W)$, where

- $G = \{g_1, \ldots, g_d\}$ is a finite ordered set of genes,
- $g_{in} \in G$ is the special gene used to model the constant input for all genes,
- $\rightarrow \subseteq G \times G$ is the activation relation such that $\forall g \in G \setminus \{g_{in}\}\ (g_{in}, g) \in \rightarrow$ and $\forall g\ (g, g_{in}) \notin \rightarrow$,
- $\dashv \subseteq G \times G$ is the repression relation such that $\dashv \cap \rightarrow = \emptyset \wedge \forall g\ (g, g_{in}) \notin \dashv$,
- $t\colon G \rightarrow \mathbb{Q}$ is the threshold function such that $\forall g \in G \setminus \{g_{in}\}\ t(g) \geq 0$ and $t(g_{in}) < 0$,
- $w^{max}\colon (\rightarrow \cup \dashv) \rightarrow \mathbb{Q}_{\geq 0}$ is the maximum value of an activation/repression,
- $W = \mathcal{P}((\rightarrow \cup \dashv) \rightarrow \mathbb{Q}_{\geq 0})$ assigns a set of possible weight functions to each activation/inhibition relation, so that $w \in W \Rightarrow \forall (g, g') \in \rightarrow \cup \dashv\, w(g, g') \leq w^{max}(g, g')$.

2.2 GRN-individual

Definition 2 (GRN-individual) A GRN-individual $\mathcal{G}$ is a pair (T, w), where $w \in W$ is a weight function from the GRN space.

A state $\sigma\colon G \rightarrow \mathbb{B}$ of a GRN-individual $\mathcal{G} = (T, w)$ denotes the activation state of each gene in terms of a Boolean value. Let $\Sigma(\mathcal{G})$ (resp. $\Sigma(T)$) denote the set of all states of $\mathcal{G}$ (resp. T), such that $\sigma(g_{in}) = true$. The GRN model executes in discrete time steps by updating all the activation states synchronously and deterministically according to the following rule: a gene is active at next time if and only if the total influence on that gene, from genes active at current time, surpasses its threshold.

Definition 3 (Semantics of a GRN-individual) A *run* of a GRN-individual $\mathcal{G} = (T, w)$ is an infinite sequence of states $\sigma_0, \sigma_1, \ldots$ such that $\sigma_n \in \Sigma(\mathcal{G})$ and $\tau(\sigma_n) = \sigma_{n+1}$ for all $n \geq 0$, where $\tau\colon \Sigma(\mathcal{G}) \rightarrow \Sigma(\mathcal{G})$ is a deterministic transition function defined by

$$\tau(\sigma) := \lambda g'. \left[\sum_{\{g \mid \sigma(g) \wedge (g,g') \in \rightarrow\}} w(g) \quad - \sum_{\{g \mid \sigma(g) \wedge (g,g') \in \dashv\}} w(g) \quad > t(g') \right]. \tag{2}$$

The language of $\mathcal{G}$, denoted by $[\![\mathcal{G}]\!]$, is a set of all possible runs of $\mathcal{G}$. Note that a GRN-individual does not specify the initial state. Therefore, $[\![\mathcal{G}]\!]$ may contain more than one run.

2.3 Temporal properties

A GRN exists in a living organism to exhibit certain behaviors. Here we present a linear temporal logic (LTL) to express the expected behaviors of GRNs.

Definition 4 (Syntax of Linear Temporal Logic) The language of linear temporal logic fomulae is given by the grammar $\varphi ::= g \mid (\neg\varphi) \mid (\varphi \vee \varphi) \mid (\varphi \mathcal{U} \varphi)$, where $g \in G$ is a gene.

Linear temporal properties are evaluated over all (in)finite runs of states from $\Sigma(G)$. Let us consider a run $r = \sigma_1, \sigma_2, \sigma_3, \cdots \in \Sigma(G)^* \cup \Sigma(G)^\infty$. Let r^i be the suffix of r after i states and r_i is the ith state of r. The satisfaction relation $\models$ between a run and an LTL formula is defined as follows:

$r \models g$ if $r_1(g)$, $\quad r \models \neg\varphi$ if $r \not\models \varphi$, $\quad r \models \varphi_1 \vee \varphi_2$ if $r \models \varphi_1$ or $r \models \varphi_2$,
$r \models (\varphi_1 \mathcal{U} \varphi_2)$ if $\exists i. r^i \models \varphi_2$ and $\forall j \leq i. r^j \models \varphi_1$.

Note that if $|r| < i$ then r^i has no meaning. In such a situation, the above semantics returns *undefined*, i.e., r is too short to decide the LTL formula. We say a language $\mathcal{L} \models \varphi$ if for each run $r \in \mathcal{L}$, $r \models \varphi$, and a GRN $\mathcal{G} \models \varphi$ if $\mathcal{L}[\![\mathcal{G}]\!] \models \varphi$. Let $\Diamond\varphi$ be shorthand of $true \mathcal{U} \varphi$ and $\Box\varphi$ be shorthand of $\neg\Diamond\neg\varphi$.

Note that we did not include next operator in the definition of LTL. This is because a GRN does not expect something is to be done in strictly next cycle.

3 Algorithm for parameter synthesis

In this section we present an algorithm for synthesising the weights' space corresponding to a given property in linear temporal logic. The method combines LTL model checking [2] and satisfiability modulo theory (SMT) solving [4].

The method operates in two steps. First, we represent any GRN-individual from the GRN space with a parametrized transition system: a transition exists between every two states, and it is labelled by linear constraints, that are necessary and sufficient constraints to enable that transition in a concrete GRN-individual (for example, see Fig. 1b). We say that a run of the parametrized transition system is *feasible* if the conjunction of all the constraints labelled along the run is satisfiable. Second, we search for all the feasible runs that satisfy the desired LTL property and we record the constraints collected along them. The disjunction of such run constraints fully characterises the regions of weights which ensure that LTL property holds in the respective GRN-individual.

Let $V = \{v_1, \ldots, v_n\}$ be a finite set of variable names. For some rationals $k_1, \ldots, k_n \in \mathbb{Q}$, let $k_1 v_1 + \cdots + k_n v_n + t > 0$ and $k_1 v_1 + \cdots + k_n v_n + t \geq 0$ be strict and non-strict linear inequalities over V, respectively. Let $polyhedra(V)$ be the set of all finite conjunctions of linear inequalities over V.

Definition 5 (Parametrized transition system) For a given GRN space T and a rational parameters map $v : G \to V$, the *parametrized transition system* (T, v) is a labelled transition system $(\Sigma(T), \Phi)$, where the labeling of the edges $\Phi : \Sigma(T) \times \Sigma(T) \to polyhedra(V)$ is defined as follows:

$$\Phi := \lambda\sigma\sigma'. \bigwedge_{g' \in G} \left[\sum_{\{g|\sigma(g) \wedge (g,g') \in \longrightarrow\}} v(g) \quad - \sum_{\{g|\sigma(g) \wedge (g,g') \in \dashv\}} v(g) \;>\; t(g') \iff \sigma'(g') \right].$$

$\Phi(\sigma, \sigma')$ says that a gene g' is active in σ' iff the weighted sum of activation and suppression activity of the regulators of g' is above its threshold.

A *run* of (T, v) is a sequence of states $\sigma_0, \sigma_1, \ldots$ such that $\sigma_n \in \Sigma(T)$ for all $n \geq 0$, and $\Phi(\sigma_0, \sigma_1) \wedge \Phi(\sigma_1, \sigma_2) \wedge \ldots$ is said to be the *run constraint* of the run. A run is feasible if its run constraint is satisfiable. We denote by $[\![(T, v)]\!]$ the set of feasible traces for (T, v). For a weight function w, let $\Phi(\sigma, \sigma')[w/v]$ denote the formula obtained by substituting v by w and let $(T, v)[w/v] = (\Sigma(T), \Phi')$, where $\Phi'(\sigma, \sigma') = \Phi(\sigma, \sigma')[w/v]$ for each $\sigma, \sigma' \in \Sigma(T)$.

In the following text, we refer to the parametrized transition system (T, v) and an LTL property φ. Moreover, we denote the run constraint of run $r = \sigma_0, \sigma_1, \ldots \in [\![(T, v)]\!]$ by $cons(r)$.

Lemma 1. For a weight function w, the set of feasible runs of $(T, v)[w/v]$ is equal to $[\![(T, w)]\!]$.

The proof of the above lemma follows from the definition of the semantics for GRN-individual. Note that the run constraints are conjunctions of linear inequalities. Therefore, we apply efficient SMT solvers to analyze (T, v).

3.1 Constraint generation via model checking

Now our goal is to synthesize the constraints over v which characterise exactly the set of weight functions w, for which (T, w) satisfies φ. Each feasible run violating φ reports a set of constraints which weight parameters should avoid. Once all runs violating φ are accounted for, the desired region of weights is completely characterized. More explicitly, the desired space of weights is obtained by conjuncting negations of run constraints of all feasible runs that satisfy $\neg\varphi$.

In Fig. 2, we present our algorithm GenCons for the constraint generation. GenCons unfolds (T, v) in depth-first-order manner to search for runs which satisfy $\neg\varphi$. At line 3, GenCons calls recursive function GenConsRec to do the unfolding for each state in $\Sigma(T)$. GenConsRec takes six input parameters. The parameter $run.\sigma$ and $runCons$ are the states of the currently traced run and its respective run constraint. The third parameter are the constraints, collected due to the discovery of counterexamples, *i.e.*, runs which violate φ. The forth, fifth and sixth parameter are the description of the input transition system and the LTL property φ. Since GRN-individuals have deterministic transitions, we only need to look for the lassos upto length $|\Sigma(T)|$ for LTL model checking. Therefore, we execute the loop at line 7 only if the $run.\sigma$ has length smaller than $|\Sigma(T)|$. The

```
function GenCons((T, v) = (Σ(T), Φ), φ)
begin
1  goodCons := true
2  for each σ ∈ Σ(T) do
3   goodCons := GenConsRec(σ, true, goodCons, Σ(T), Φ, φ)
4  done
5  return goodCons
end

function GenConsRec(run.σ, runCons, goodCons, Σ(T), Φ, φ)
begin
6  if |run.σ| < |Σ(T)| then
7     for each σ' ∈ Σ(T) do
8        runCons' := runCons ∧ Φ(σ, σ')
9        if goodCons ∧ runCons' is sat then
10          if run.σσ' ⊨ ¬φ then                    (* check may return undef *)
11             goodCons := goodCons ∧ ¬runCons'
12          else
13             goodCons := GenConsRec(run.σσ', runCons', goodCons, Σ(T), Φ, φ)
14    done
15 return goodCons
end
```

Fig. 2: Counterexample guided computation of the mutation space feasible wrt. φ. Let "." be an operator that appends two sequences. $run.\sigma\sigma' \models \neg\varphi$ can be implemented by converting $\neg\varphi$ into a Büchi automaton and searching for an accepting run over $run.\sigma\sigma'$. However, a finite path may be too short to decide whether φ holds or not. In that case, the condition at line 10 fails. Since (T, v) is finite, the finite runs are bound to form lassos within $|\Sigma(T)|$ steps. If a finite run forms a lasso, then the truth value of $run.\sigma\sigma' \models \neg\varphi$ will be determined.

loop iterates over each state in $\Sigma(T)$. The condition at line 9 checks if $run.\sigma\sigma'$ is feasible and, if it is not, the loop goes to another iteration. Otherwise, the condition at line 10 checks if $run.\sigma\sigma' \models \neg\varphi$. Note that $run.\sigma\sigma' \models \neg\varphi$ may also return undefined because the run may be too short to decide the LTL property. If the condition returns true, we add negation of the run constraint in *goodCons*. Otherwise, we make a recursive call to extend the run at line 13. *goodCons* tells us the set of values of v for which we have discovered no counterexample. GenCons returns *goodCons* at the end.

Since run constraints are always a conjunction of linear inequalities, *goodCons* is a conjunction of clauses over linear inequalities. Therefore, we can apply efficient SMT technology to evaluate the condition at line 9. The following theorem states that the algorithm GenCons computes the parameter region which satisfies property φ.

Theorem 1. For every weight function $w \in W$, the desired set of weight functions for which a GRN-individual satisfies φ equals the weight functions which

satisfy the constraints returned by GenCons:

$$(T, w) \models \varphi \text{ iff } w \models \textsc{GenCons}((T, v), \varphi).$$

Proof. The statement amounts to showing that the sets $A = \{w \mid (T, w) \models \varphi\}$ and $B = \bigcap_{r \in [\![(T,v)]\!] \wedge r \models \neg\varphi} \{w \mid w \models \neg cons(r)\}$ are equivalent. Notice that $W \setminus A = \bigcup_{r \in [\![(T,v)]\!] \wedge r \models \neg\varphi} \{w \mid w \models cons(r)\} = W \setminus B$.

We use the above presentation of the algorithm for easy readability. However, our implementation differs significantly from the presentation. We follow the encoding of [7] to encode the path exploration as a bounded-model checking problem. Further details about implementation are available in Section 5. The algorithm has exponential complexity in the size of T. However, one may view the above procedure as the clause learning in SMT solvers, where clauses are learnt when the LTL formula is violated [23]. Similar to SMT solvers, in practice, this algorithm may not suffer from the worst-case complexity.

Example 1. The GRN `osc3` (shown in Fig. 3) was the model of a pioneering work in synthetic biology [11], and it is known to provide oscillatory behaviour: each gene should alternate its expression between 'on' and 'off' state:

$$\varphi_3 = \textstyle\bigwedge_{v \in \{A,B,C\}} (v \Rightarrow \Diamond\neg v) \wedge (\neg v \Rightarrow \Diamond v).$$

The solutions are the constraints: $(T, w) \models \varphi_3$ iff $(i_A > t_A) \wedge (i_B > t_B) \wedge (i_C > t_C) \wedge (i_B - w_{AB} \leq t_B) \wedge (i_C - w_{BC} \leq t_C) \wedge (i_A - w_{CA} \leq t_A)$.

4 Computing Robustness

In this section, we present an application of our parameter synthesis algorithm, namely computing robustness of GRNs in presence of mutations. To this end, we formalize GRN-population and its robustness. Then, we present a method to compute the robustness using our synthesized parameters.

A GRN-population models a large number of GRN-individuals with varying weights. All the GRN-individuals are defined over the same GRN space, hence they differ only in their weight functions. The GRN-population is characterised by the GRN space T and a probability distribution over the weight functions. In the experimental section, we will use the range of weights W and the distribution π based on the mutation model outlined in the Appendix in [14].

Definition 6 (GRN-population) A GRN-population is a pair $\mathcal{Z} = (T, \pi)$, where $\pi : W \to [0, 1]$ is a probability distribution over all weight functions from the GRN space T, i.e., $\sum_{w \in W} \pi(w) = 1$.

We write $\varphi(\mathcal{Z}) \in [0, 1]$ to denote an expectation that a GRN instantiated from a GRN-population $\mathcal{Z} = (T, \pi)$ satisfies φ. The value $\varphi(\mathcal{Z})$ is in the interval $[0, 1]$ and we call it robustness.

Definition 7 (Robustness) Let $\mathcal{Z} = (T, \pi)$ be a GRN-population, and φ be an LTL formula which expresses the desired LTL property. Then, the robustness of $\mathcal{Z}$ with respect to the property φ is given by

$$\varphi(\mathcal{Z}) := \sum_{\{w | [\![(T,w)]\!] \models \varphi\}} \pi(w)$$

The above definition extends that of [10], because it allows for expressing any LTL property as a phenotype, and hence it can capture more complex properties such as oscillatory behaviour. In the following, we will present an algorithm for computing the robustness, which employs algorithm GenCons.

4.1 Evaluating robustness

Let us suppose we get a GRN-population $\mathcal{Z} = (T, \pi)$ and LTL property φ as input to compute robustness. For small size of GRN space T, robustness can be computed by explicitly enumerating all the GRN-individuals from T, and verifying each GRN-individual against φ. The probabilities of all satisfying GRN-individuals are added up. However, the exhaustive enumeration of the GRN space is often intractable due to a large range of weight functions W in T. In those cases, the robustness is estimated statistically: a number of GRN-individuals are sampled from T according to the distribution π, and the fraction of satisfying GRN-individuals is stored. The sampling experiment is repeated a number of times, and the mean (respectively variance) of the stored values are reported as robustness (respectively precision).

Depending on how a sampled GRN-individual is verified against the LTL property, we have two methods:

- In the first method, which we will call *execution* method, each sampled GRN-individual is verified by executing the GRN-individual from all initial states and checking if each run satisfies φ;
- In the second method, which we will call *evaluation* method, the constraints are first precomputed with GenCons, and each sampled GRN-individual is verified by evaluating the constraints.

Clearly, the time of computing the constraints initially renders the evaluation method less efficient. This cost is amortized when verifying a GRN-individual by constraint evaluation is faster than by execution. In the experimental section, we compare the overall performance of the two approaches on a number of GRNs from literature.

5 Experimental results

We implemented a tool which synthesizes the parameter constraints for a given LTL property (see Section 3), and the methods for computing the mutational robustness (see Section 4.1). We ran our tool on a set of GRNs from the literature.

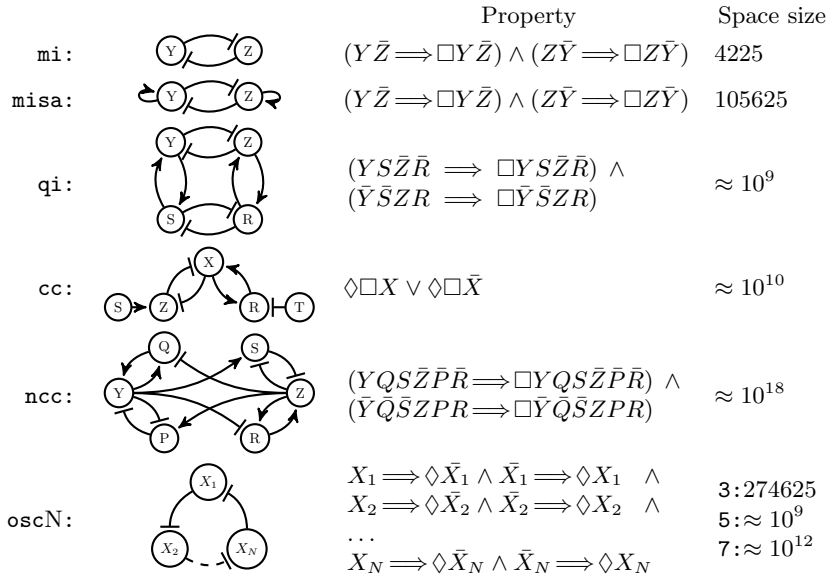


Fig. 3: GRN benchmarks. `mi`, `misa` (mutual inhibition), `qi` (quad inhibition), and `ncc` (cell cycle switch) satisfy different forms of bistability. For the networks `ci` (cell cycle switch), the value of gene eventually stabilizes [9]. In `osc3`, also known as the *repressilator* [11], the gene values alternate. `osc5` and `osc7` (not shown) are generalizations of `osc3`, and also exhibit oscilating behavior.

5.1 Implementation

Our implementation does not explicitly construct the parametrised transition system described in Section 3 (Dfn. 5 and Alg. 2). Instead, we encode the bounded model-checking (Alg. 2) as a satisfiability problem, and we use an SMT solver to efficiently find *goodCons*. More concretely, we initially build a formula which encodes the parametrized transition system and it is satisfied if and only if some run of (T, v) satisfies $\neg\varphi$. If the encoding formula is satisfiable, the constraints $cons(r)$ along the run are collected, and $\neg cons(r)$ is added to *goodCons*. Then, we expand the encoding formula by adding $\neg cons(r)$, so as to rule out finding the same run again. We continue the search until no satisfying assignment of the encoding formula can be found. The algorithm always terminates because the validity of the property is always decided on finite runs (as explained in Section 3).

The implementation involves 8k lines of C++ and we use Z3 SMT solver as the solving engine. We use CUDD to reduce the size of the Boolean structure of the resulting formula. We ran the experiments on a GridEngine managed cluster system. The tool is available online[1].

[1] `http://pub.ist.ac.at/~mgiacobbe/grnmc.tar.gz`

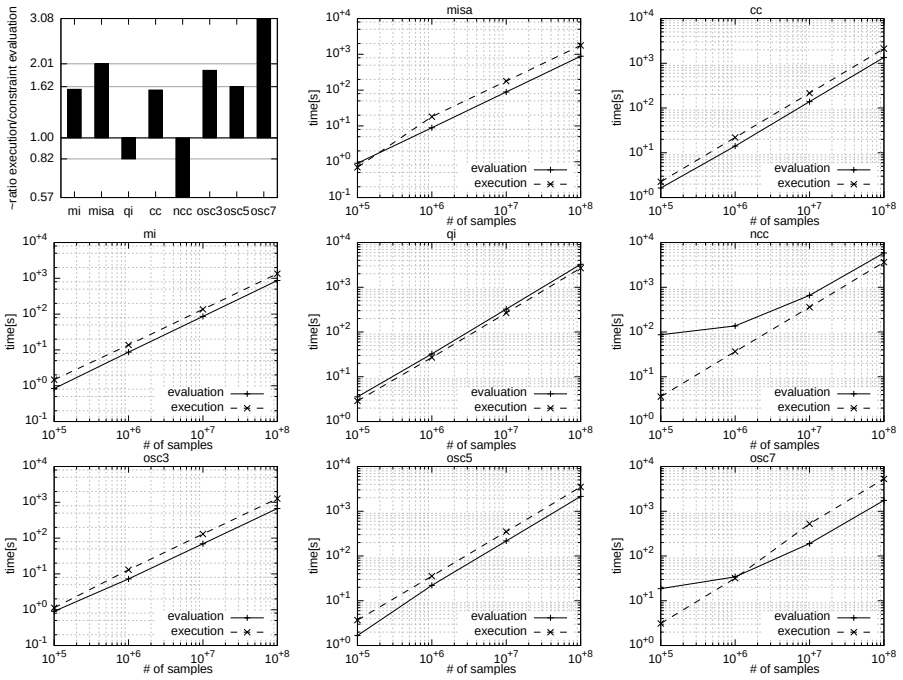


Fig. 4: The comparison in performance when mutational robustness is statistically estimated, and a property check is performed either by evaluation, or by execution (see Section 4.1 for the description of the two methods). The bar (top-left) shows the ratio of average times needed to verify the property of one sampled GRN-individual. For example, for `osc7`, the performance of evaluation method is more than three times faster. The other graphs show how the robustness computation time depends on the total number of sampled GRNs (in order to obtain robustness and to estimate the precision, we computed the mean of 100 experiments, each containing a number of samples ranging from 10^3 to 10^6). The graph is shown in log-log scale. The non-linear slope of the evaluation method is most notable in examples `mcc` and `osc7`, and it is due to the longer time used for compute the constraints.

5.2 Performance evaluation

We chose eight GRN topologies as benchmarks for our tool. The benchmarks are presented in Fig. 3. The first five of the GRN topologies are collected from [8]. On these benchmarks we check for the steady-state properties. On the final three GRN topologies, we check for the oscillatory behavior. The results are presented in Fig. 4.

We ran the robustness computation by the evaluation and execution methods (the methods are described in Section 4.1). In order to obtain robustness and to estimate the precision, we computed the mean of 100 experiments, each contain-

ing a number of samples ranging from 10^3 to 10^6. The total computation time in the execution methods linearly depends on the number of samples used. The total computation time in the evaluation method depends linearly on the number of samples, but initially needs time to compute the constraints. Technically, the time needed to compute robustness by execution method is $t_{ex} = k_{ex}p$, and the time needed to compute robustness by evaluation approach $t_{ev} = k_{ev}p + t_c$, where p represents the total number of samples used, t_c is the time to compute the constraints, and k_{ex} (resp. k_{ev}) is the time needed to verify the property by evaluation (resp. execution). We used linear regression to estimate the parameters k_{ex} and k_{ev}, and we present the ratio $\frac{k_{ex}}{k_{ev}}$ in top-left position of Fig. 4. The results indicate that on six out of eight tested networks, evaluation is more efficient than execution. For some networks, such as `osc7`, the time for computing the constraints is large, and the gain in performance becomes visible only once the number of samples is larger than 10^6.

6 Conclusion and discussion

We pursued formal analysis of Wagner's GRN model, which allows symbolic reasoning about the behavior of GRNs under parameter perturbations. More precisely, for a given space of GRNs and a property specified in LTL, we have synthesized the space of parameters for which the concrete, individual GRN from a given space satisfies the property. The resulting space of parameters is represented by complex linear inequalities. In our analysis, we encoded a bounded model-checking search into a satisfiability problem, and we used efficient SMT solvers to find the desired constraints. We demonstrated that these constraints can be used to efficiently compute the mutational robustness of populations of GRNs. Our results have shown the cases in which the computation can be three times faster than the standard (simulation) techniques employed in computational biology.

While computing mutational robustness is one of the applications of our synthesized constraints, the constraints allow to efficiently answer many other questions that are very difficult or impossible to answer by executing the sampled GRNs. In our future work, we aim to work on further applications of our method, such as parameter sensitivity analysis for Wagner's model. Moreover, we plan to work on the method for exact computation of robustness by applying the point counting algorithm [5].

Wagner's GRN model is maybe the simplest dynamical model of a GRN – there are many ways to add expressiveness to it: for example, by incorporating multi-state expression level of genes, non-determinism, asynchronous updates, stochasticity. We are planning to study these variations and chart the territory of applicability of our method.

References

1. R. B. R. Azevedo, R. Lohaus, S. Srinivasan, K. K. Dang, and C. L. Burch. Sexual reproduction selects for robustness and negative epistasis in artificial gene net-

works. *Nature*, 440(7080):87–90, Mar. 2006.
2. C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.
3. C. Barrett, M. Deters, L. de Moura, A. Oliveras, and A. Stump. 6 years of SMT-COMP. *Journal of Automated Reasoning*, 50(3):243–277, 2013.
4. C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. *Handbook of satisfiability*, 185:825–885, 2009.
5. A. Barvinok and J. E. Pommersheim. An algorithmic theory of lattice points in polyhedra. *New perspectives in algebraic combinatorics*, 38:91–147, 1999.
6. G. Batt, C. Belta, and R. Weiss. Model checking genetic regulatory networks with parameter uncertainty. In *Hybrid systems: computation and control*, pages 61–75. Springer, 2007.
7. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in computers*, 58:117–148, 2003.
8. L. Cardelli. Morphisms of reaction networks that couple structure to function. *BMC Systems Biology*, 8(1):84+, 2014.
9. L. Cardelli and A. Csikász-Nagy. The cell cycle switch computes approximate majority. *Scientific reports*, 2, 2012.
10. S. Ciliberti, O. C. Martin, and A. Wagner. Robustness can evolve gradually in complex regulatory gene networks with varying topology. *PLoS Computational Biology*, 3(2), 2007.
11. M. B. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, 2000.
12. J. Fisher and T. A. Henzinger. Executable cell biology. *Nature biotechnology*, 25(11):1239–1249, 2007.
13. T. S. Gardner, C. R. Cantor, and J. J. Collins. Construction of a genetic toggle switch in escherichia coli. *Nature*, 403(6767):339–342, 2000.
14. M. Giacobbe, C. C. Guet, A. Gupta, T. A. Henzinger, T. Paixao, and T. Petrov. Model checking gene regulatory networks. *arXiv preprint arXiv:1410.7704*, 2014.
15. S. K. Jha, E. M. Clarke, C. J. Langmead, A. Legay, A. Platzer, and P. Zuliani. A bayesian approach to model checking biological systems. In *Computational Methods in Systems Biology*, pages 218–234. Springer, 2009.
16. M. Kwiatkowska, G. Norman, and D. Parker. Using probabilistic model checking in systems biology. *ACM SIGMETRICS Performance Evaluation Review*, 35(4):14–21, 2008.
17. T. MacCarthy, R. Seymour, and A. Pomiankowski. The evolutionary potential of the drosophila sex determination gene network. *Journal of Theoretical Biology*, 225(4):461468, 2003.
18. R. Mateescu, P. T. Monteiro, E. Dumas, and H. De Jong. Ctrl: Extension of ctl with regular expressions and fairness operators to verify genetic regulatory networks. *Theoretical Computer Science*, 412(26):2854–2883, 2011.
19. A. Rizk, G. Batt, F. Fages, and S. Soliman. A general computational method for robustness analysis with applications to synthetic gene networks. *Bioinformatics*, 25(12):i169–i178, 2009.
20. T. Schlitt and A. Brazma. Current approaches to gene regulatory network modelling. *BMC bioinformatics*, 8(Suppl 6):S9, 2007.
21. A. Wagner. Does evolutionary plasticity evolve? *Evolution*, 50(3):1008–1023, 1996.
22. B. Yordanov, C. M. Wintersteiger, Y. Hamadi, and H. Kugler. SMT-based analysis of biological computation. In *NASA Formal Methods*, pages 78–92. Springer, 2013.
23. L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Computer Aided Verification*, pages 279–285. IEEE Press, 2001.