# Proving stabilization of biological systems

Byron Cook[1,2], Jasmin Fisher[1], Elzbieta Krepska[1,3], and Nir Piterman[4]

[1] Microsoft Research
[2] Queen Mary, University of London
[3] VU University Amsterdam
[4] University of Leicester

**Abstract.** We describe an efficient procedure for proving stabilization of biological systems modeled as qualitative networks or genetic regulatory networks. For scalability, our procedure uses modular proof techniques, where state-space exploration is applied only locally to small pieces of the system rather than the entire system as a whole. Our procedure exploits the observation that, in practice, the form of modular proofs can be restricted to a very limited set. For completeness, our technique falls back on a non-compositional counterexample search. Using our new procedure, we have solved a number of challenging published examples, including: a 3-D model of the mammalian epidermis; a model of metabolic networks operating in type-2 diabetes; a model of fate determination of vulval precursor cells in the *C. elegans* worm; and a model of *pair-rule* regulation during segmentation in the *Drosophila* embryo. Our results show many orders of magnitude speedup in cases where previous stabilization proving techniques were known to succeed, and new results in cases where tools had previously failed.

## 1 Introduction

Biologists are increasingly turning to computer-science techniques in the quest to understand and predict the behavior of complex biological systems [1–3]. In particular, application of formal verification tools to models of biological processes is gaining impetus among biologists. In some cases known formal verification techniques work well (*e.g.* [4–7]). Unfortunately in other cases—such as proving stabilization [8]—we find existing abstractions and heuristics to be ineffective.

In this paper we address the open challenge to find scalable algorithms for proving stabilization of biological systems. A proof of stabilization elucidates system robustness with respect to time, while stabilization counterexamples give insight into system homeostasis – in both cases the result is useful to biologists. In computer science terms, stabilization means the existence of a *unique* fixpoint state that is always eventually reached. We are trying to prove this property of large parallel systems, where the size of these systems forces us to use some form of modular reasoning. Since stabilization is formally a liveness property, we must be careful when using the powerful cyclic modular proof rules (*e.g.* [9, 10]), as they are only sound in the context of safety [11]. Furthermore, we find that the complex temporal interactions between the modules are crucial to

the stabilization of the system as a whole; meaning that we cannot use scalable techniques that simply abstract away the interactions altogether.

In this paper we present a procedure for proving stabilization of biological systems modeled as communicating components in the qualitative networks formalism [12] with synchronous updates of variables, or a genetic regulatory networks [13], where updates are asynchronous. We compose these stabilization proofs out of small lemmas that can be solved using quick local proof techniques on the components. The key to our tool's performance is the observation that it suffices to take the lemmas only of a very limited form: $[FG(p_1) \wedge \ldots \wedge FG(p_k)] \Rightarrow FG(q)$, where $p_1 \ldots p_k$ are atomic formulae over inputs of a small component that we want to reason about, $q$ is a atomic formula about this component's output, $F$ denotes "eventually" in LTL [16], and $G$ denotes "always". We compute the set of all provable lemmas of this form by iterative strengthening. After this procedure, if for each component $v$ its lemma implies $FG(v = k_v)$ for some constant $k_v$, that means that we have proved stabilization. If some component is left unfixed, then we use the lemmas to restrict the counterexample search space.

Our stabilization proving procedure is sound and complete. We experimentally confirm that it is scalable. We find that our lemma generation procedure accelerates both the proving as well as the disproving of stabilization. Section 4 demonstrates with experimental evidence how our lemma generation procedure leads to many orders of magnitude speedup in cases where known previous techniques work, and new results in cases where known techniques fail. These include challenging published examples such as: a 3-D model of the mammalian epidermis based on [12]; a model of metabolic networks operating in type-2 diabetes [17]; a model of fate determination of vulval precursor cells in *C. elegans* [18]; and a model of *pair-rule* cross-regulation during segmentation in *Drosophila* fly embryo [19]. Applying our procedure to the multidimensional model of epidermis revealed a bug in the model from [12], as we proved the system non-stabilizing. Consulting the biological papers corroborated that the model was, in fact, in disagreement with the biological evidence. After fixing the bug we could then prove the system stabilizing (see Section 2).

*Related work.* With the exception of [12], no tools have been previously reported that are directly tailored to the problem of proving stabilization or other liveness properties of large biological systems modeled as discrete systems (*e.g.* qualitative networks). Classic theory of stability of differential equations is applied to continuous systems, *e.g.* in [20]. Recent work is known on the stability of hybrid systems, *e.g.* [21–23]. In the context of stabilization for discrete systems, [12] uses the compositional structure of a system modeled as qualitative network to accelerate the computation of a fixpoint-based computation of the reachable states. However, the final check is not modular, and thus is less scalable than our approach. Genetic regulatory networks [13] have been extensively studied, *e.g.* in [13, 19], but the analysis relies on state space enumeration, which is not scalable, or stable states computation that does not account for reachability [14].

The current state-of-the-art amongst biologists interested in stabilization is to use either techniques from [12] or other off-the-shelf model checking tools

for finite-state systems. Recently developed tools for proving liveness of infinite-state systems (*e.g.* [24]) could also be used. As we show in Section 4, our procedure is many orders of magnitude faster than previously known approaches. The challenge is that biological models are very large, causing timeouts and out-of-memory failures for most tools not based on modular proof strategies. Note also that stabilization is not directly expressible in popular temporal logics, *e.g.* CTL or LTL, unless support for quantifiers is added, making the encoding of stabilization tricky in most formal verification tools. Qualitative networks could be implemented in Lustre [15], which however supports checking only safety properties.

We are not the first to attempt to address the difficulty of modular reasoning for liveness. For example, several previous papers have reported on heuristics tailored to the problem of proving liveness of non-blocking concurrent programs [24, 25]. Their motivation is the same as here, but the approaches used differ as they are tailored to different problems. Another technique, as found in [26], is to use induction over time to facilitate the modular proving of liveness properties of finite-state models. In [26] the modular decomposition is given manually, whereas in our work we use the structure of the biological system to our advantage when automating the search for the modular decomposition. To show that our proofs are non-circular we use an argument similar to that of [26].

Our algorithm depends on a domain $\mathcal{L}$ over which lemmas range. When handling qualitative networks and genetic regulatory networks all variables range over domains of the form $\{0, \ldots, n\}$. Furthermore, the updates of variables are always in increments or decrements of 1. As our aim is to prove stabilization, lemmas that restrict variables to *one* subrange of their domain turn out to be sufficient. This insight is the basis for an optimization of the lemma generation algorithm, which works extremely fast in practice. When considering this optimization, our technique can be thought of as analyzing the system using abstract interpretation over the interval domain [27]. A similar usage of abstract interpretation to produce *tail invariants* used for termination proofs appears in [28].

*Limitations and advantages.* Our technique is geared towards the efficient proving of stabilization where the proof can be teased out by examining the system's compositional structure. This lemma-generation strategy comes with an overhead that can potentially hinder rather than help performance in some cases. In Section 4 we demonstrate an example of this.

An advantage of our procedure is compositionality: the local stabilization lemmas give a specification that, when established for new components, implies the whole system's stabilization without re-running the entire procedure. This can lead to experimenting with alternative components (*e.g.* testing modified components during a search for new drugs). This observation also leads to a practical advantage, as we check lemmas in parallel during the proof search.

## 2 Example: Skin cells

Figure 1 contains a pictorial view of a simplified model of mammalian epidermis (outermost skin layer) that consists of five stacked cells [12]. Each cell represents

a single skin layer and communicates with neighboring cells. The bottommost cells proliferate, migrate upwards and eventually decide to die and thus contribute to the cornified skin surface. It is this balance between proliferation and cell death that makes the system interesting to biologists: too much death is detriment to the skin, too little is cancerous. The original model is expressed as a qualitative network [12]. Formal definitions of qualitative and regulatory networks are given later, here we describe the epidermis model only informally.

The example model includes a few executing components, each updating a single variable. See, e.g., $\texttt{wnt}_3$ or $\texttt{NotchIC}_3$ in Fig. 1. Each variable holds a value, which is a number in $\{0, 1 \dots N\}$, where $N + 1$ is a predefined, globally-fixed *granularity*. A *target function*, $T_v$, associated with each variable, $v$, determines how the variable is updated: if $v < T_v$ then $v' = v + 1$, if $v > T_v$ then $v' = v - 1$, else $v' = v$. In a qualitative network all variables are updated synchronously in parallel and in a genetic regulatory network they are updated asynchronously.

Intuitively, the update function of each variable is designed such that the value of the variable follows its target, which depends on other variables. In the biological setting, the typical target of a variable, $v$, combines positive influence of variables $w_1, w_2, \dots w_s$ with negative influence of variables $w_{s+1}, w_{s+2}, \dots w_{s+r}$ and ignores all other variables in the network:

$$T_v(w_1, w_2, \dots w_{s+r}) = \max\left(0, \left\lfloor \frac{1}{s}\sum_{k=1}^{s} w_k - \frac{1}{r}\sum_{k=1}^{r} w_{s+k} \right\rfloor\right)$$

Graphically, this is often represented as an *influence graph* with $\rightarrow$ edges between each of $w_1, w_2, \dots, w_s$ and $v$ and $\dashv$ edges between each of $w_{s+1}, w_{s+2}, \dots, w_{s+r}$ and $v$. In this section we discuss only several target functions used in the skin example. Refer to papers [12, 17–19] for target functions used to model a large spectrum of aspects of signaling pathways, metabolic and genetic regulatory networks. In the skin example, the target of $\texttt{wnt}_3$ is $T_{\texttt{wnt}_3} = N - \texttt{NotchIC}_3$, which means that $\texttt{NotchIC}_3$ inhibits $\texttt{wnt}_3$ (in Fig. 1 this fact is indicated by a 'blocking' arrow from $\texttt{NotchIC}_3$ to $\texttt{wnt}_3$). The target of $\texttt{NotchIC}_3$ is $T_{\texttt{NotchIC}_3} = \min(3, \texttt{deltaext}_3)$ and is indicated by an underline. The targets of the ext-variables round averaged cell inputs, which effectively requires at least one of the components to be present for some event to take place:

$$T_{\texttt{deltaext}_1} = \left\lceil \frac{\texttt{delta}_0 + \texttt{delta}_2}{2} \right\rceil, \qquad T_{\texttt{wntext}_1} = \left\lfloor \frac{\texttt{wnt}_0 + \texttt{wnt}_2}{2} \right\rfloor.$$

Figure 1 shows behavior of four selected variables, based on their target function.

*Stabilization.* If all executions end in *the same* cycle, and that cycle has length 1, then we say the network *stabilizes*. Note that both qualitative and regulatory networks are finite-state systems with only infinite executions. Thus, every execution must eventually end in *some type* of cycle. Stabilization guarantees both that the system has only a single fixpoint and that the fixpoint is always eventually reached—a violation of this property is the existence of two fixpoints or a cycle of length greater than 1. Biologists are often interested to see what this fixpoint is when it exists, and to see a counterexample when it does not.

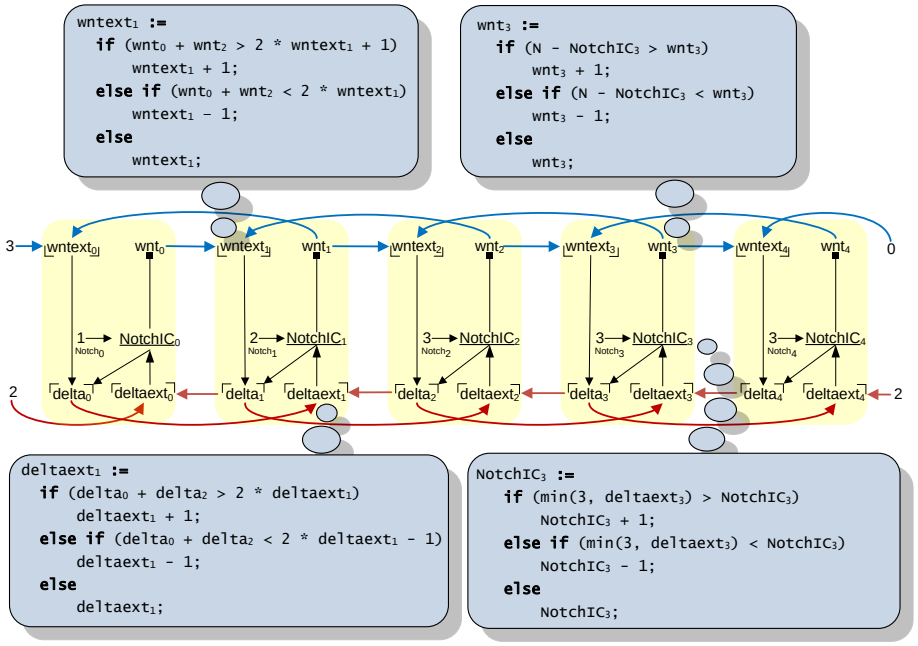


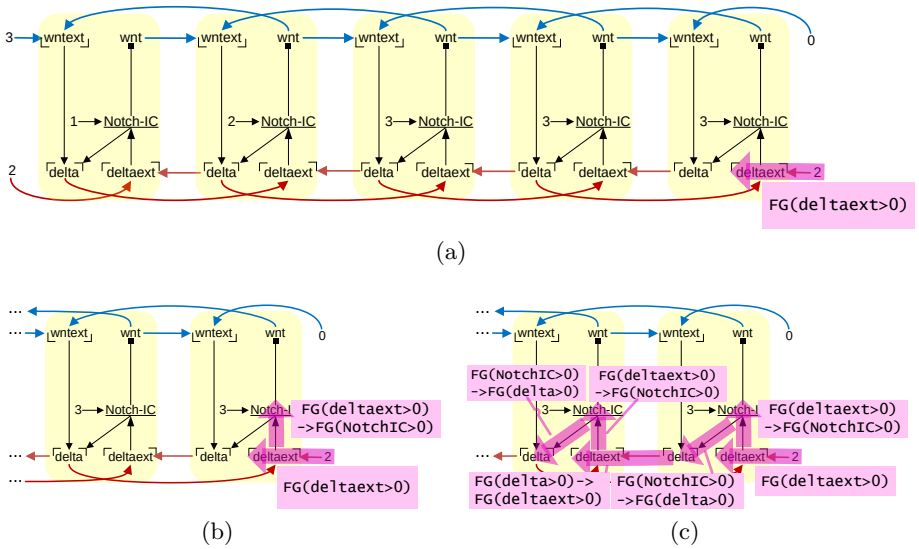

**Fig. 1.** Pictorial view of the skin model (rightmost cell is at skin surface). The bubbles show the underlying update functions for several of the variables in the model.

When applied to the skin example, our tool incrementally finds a modular proof of stabilization, as depicted in Fig. 2. The tool starts by guessing simple facts with the form $FG(p)$ about variables that can be proved locally, *i.e.* using the update function of only one variable with the definitions of the other variables abstracted away, see Fig. 2(a). In this case, we can infer locally the lemma $FG(\texttt{deltaext}_4 > 0)$ in the top cell. This property is provable using only local reasoning because the $\texttt{deltaext}_4$ variable follows a target $\lceil (2 + \texttt{delta}_3)/2 \rceil$, which is always a positive number, independently of the value of $\texttt{delta}_3$.

In the next step, we iteratively use proved facts to guide the search for additional facts to conclude. We search for locally provable facts of the form $FG(p) \Rightarrow FG(q)$, where we only try proving $FG(p) \Rightarrow FG(q)$ if $FG(p)$ is a consequent in a previous iteration. In our example, we can locally infer that $FG(\texttt{deltaext}_4 > 0) \Rightarrow FG(\texttt{NotchIC}_4 > 0)$, see Fig. 2(b). This implication holds because $\texttt{NotchIC}_4$ in the top cell follows a target, which effectively equals $\texttt{deltaext}_4$. Since $\texttt{deltaext}_4$ is eventually always positive, so is $\texttt{NotchIC}_4$.

In the next round, we can prove $FG(\texttt{NotchIC}_4 > 0) \Rightarrow FG(\texttt{wnt}_4 < N)$ in the top cell, see Fig. 2(c). This property holds locally, because the target of $\texttt{wnt}_4$ is $N - \texttt{NotchIC}_4$. Figure 2(c) contains also several subsequent stages of the proof. We continue such reasoning until no new implications can be deduced. At that point, if we conclude $\ldots FG(v = k_v)$ for some $k_v \in \{0, 1 \ldots N\}$ for each variable $v$, then we have found a global stable state and proved that the model stabilizes.

*A bug in the skin model.* Applying our tool to the 1-D skin model described above proved the model stabilizing. Contrastingly, applying the tool to the 2-D skin




(a) (b) (c)

**Fig. 2.** Proof steps of skin model stabilization. Each arrow denotes a lemma.

model built out of several interconnected such 1-D models, revealed that the 2-D model is *not* stabilizing. This result was biologically surprising, so we suspected a bug in the original [12] model. After consulting biological literature [29], we discovered that the bug was real, *i.e.* the original model was in disagreement with biological evidence. The fix proposed was to change the value of the Notch protein (constant input of $\texttt{NotchIC}_0$ in the bottommost skin layer) from 0 to 1. By doing so we effectively introduced a low level of Notch protein into the basal layer of epidermis. With the bug fixed, we proved the multidimensional model stabilizing. While this finding offered no new biological insight, it helped to repair the existing model and confirmed usefulness of the method to biologists.

## 3 Stabilization algorithm

We now describe our algorithm, which attempts to efficiently prove stabilization of systems using the modular strategy exemplified in the previous section.

*Preliminaries.* Following [12], a *qualitative network* (QN), $Q(V, T, N, n)$, of granularity $N+1$ consists of $n$ variables: $V = (v_1, v_2 \dots v_n)$. The state of the system is a finite map $s : V \to \{0, 1, \dots N\}$. The initial state is random. Each variable $v_i \in V$ has a *target function* $T_i \in T$ associated with it: $T_i{:}\{0, 1, \dots N\}^n \to \{0, 1, \dots N\}$. Qualitative networks update the variables using synchronous parallelism.

Target functions in qualitative networks direct the execution of the network: from state $v = (v_1, v_2 \dots v_n)$, the *next state* $v' = (v'_1, v'_2 \dots v'_n)$ is computed by:

$$v'_i = \begin{cases} v_i + 1 & v_i < T_i(v), \\ v_i & v_i = T_i(v), \\ v_i - 1 & v_i > T_i(v). \end{cases} \tag{1}$$

A *genetic regulatory network* (GRN) [13], $G(V, M, T, n)$, consists of $n$ discrete variables: $V = (v_1, v_2 \dots v_n)$ bounded individually by $M : V \to \{1, \dots, N\}$. Nodes have target functions from $T$ associated with them that govern updates of variables as in (1). The updates are asynchronous, which is the major difference between GRNs and QNs. We additionally assume that the updates are fair, *i.e.* each variable that is not equal to its target value is eventually updated.

Qualitative networks and genetic regulatory networks, as such, prove to be a suitable formalism to model some biological systems [12, 13, 17–19]. A target function of a variable $v$ is typically a simple algebraic function, such as sum, over several other variables $w_1, w_2 \dots w_m$. We often say that $v$ *depends* on $w_1, w_2 \dots w_m$ or that $w_1, w_2 \dots w_m$ are *inputs* of $v$. $Q \,|_v$ denotes the restriction of $Q$ to the variable $v$ and its inputs, where the inputs behave arbitrarily. In the following, we use the term *network* to refer to both QNs and GRNs.

We say that a network is *stabilizing* if there exists a unique state $s$, that is eventually reached in all executions, such that $T(s) = s$. Intuitively, this means that starting from an arbitrary state, we always end up in a fixpoint and always the same one. Formally, we are attempting to prove the existence of a unique state $(k_1, k_2, \dots k_n)$ such that $FG(\forall v_i \in V.\ v_i = k_i)$. Note that the stabilization property is not expressible in LTL unless we add support for both existential and universal quantification over states.

We define $\mathcal{L}$ to be a finite set of predicates that range over the simple inequalities of the form $m \le v \le M$, where $m$ and $M$ are constants in $\{0, 1, \dots, N\}$.

We use the term *local lemma* over a variable $v$ to represent proved assertions of the form $FG(p_1) \wedge FG(p_2) \wedge \dots \wedge FG(p_m) \Rightarrow FG(q)$ where $q \in \mathcal{L}$ restricts $v$ and $p_1$ through $p_m$ are predicates about variables in the network proved previously.

*Algorithm.* Since networks considered are finite and all executions are infinite, each execution of the system must end in a cycle. We consider all possible executions of a network and note the trichotomy: (a) all executions end in the same fixpoint (the network stabilizes); or (b) there exists an execution that ends in a cycle of length greater than 1 (the network cycles); or (c) all executions end in a fixpoint and there exist at least two different fixpoints (the network bifurcates). As described later in this section, our algorithm covers all of these cases, and is therefore complete. We note that completeness depends on the finiteness of networks considered and on the fact that the algorithm falls back on the non-compositional Search routine.

Our procedure is displayed in Alg. 1. It first applies a local lemma generation procedure GenLemmas (Alg. 2) that we explain next in this section. In all practical cases we find that the lemmas found during this phase directly imply stabilization in cases where the model does stabilize. If no proof has been found, the strategy is reversed: our procedure searches for one of two types of counterexamples: *multiple fixpoints* and *non-trivial cycles*. Both counterexample finding procedures are complete; therefore, in the instance that GenLemmas does not prove stabilization and yet no counterexample is found, we have still proved stabilization. The procedure Search$(V^{\min}, V^{\max}, Q)$ is used by Alg. 1 to look for existence of a counterexample in a network $Q$. Search uses the proved

variable constraints $V^{\min}$ and $V^{\max}$ to reduce the state space in which it needs to explore. If SEARCH is unable to find a counterexample, no counterexample exists. Thus, in this case, we know that we need only find a single trivial cycle. This is easily done using a decision procedure as is done in SEARCH.

*Lemma generation (Alg. 2).* The key idea behind our approach is to first find *local lemmas* about the update functions for specific variables in the network. That is, if a variable $v$ locally depends on variables $w_1, w_2, \ldots w_m$, we compute lemmas about interactions between $v$ and $w_i$'s of the following form:

$$FG(p_1) \wedge FG(p_2) \wedge \cdots \wedge FG(p_m) \Rightarrow FG(q)$$

where $p_i$'s are predicates in $\mathcal{L}$ about variables $w_j$'s and $q$ is a predicate about $v$. We compute the local lemmas until no new ones can be deduced. If for each variable $v \in V$ we can use the lemmas to prove that $FG(v = k)$ for some constant $k$, then we can report that the system is stabilizing.

The procedure GENLEMMAS, Alg. 2, iteratively computes a set of lemmas, $\mathcal{P}$. During the iterative search it maintains a set of frontier variables, $\mathcal{F}$, for which new facts have been proved, but not used yet. Initially, $\mathcal{F}$ contains all unfixed variables in the network. The procedure repeatedly picks $w \in \mathcal{F}$ and generates new local lemmas about variables that depend on $w$. The lemmas are used to update $V^{\min}$ and $V^{\max}$, which overapproximate the least and upper bounds of values of variables; namely, for each $v \in V$ we have $FG\left(V^{\min}(v) \leq v \leq V^{\max}(v)\right)$. Alg. 2 terminates because the variable's bounds can be updated at most $N$ times, so each variable can be enqueued at most $N$ times. From this also follows that GENLEMMAS performs no exponential explorations. Generation of the local lemmas NEWLEMMAS is shown in Alg. 3. It proceeds via a search over the language of base inequalities $\mathcal{L}$: the predicates over $v$ that improve current approximation and that are proven to hold, are returned. Termination of Alg. 3 follows from finiteness of $|\mathcal{L}|{=}\mathcal{O}(nN)$. The worst-case complexity of Alg. 2 is thus $\mathcal{O}(n^2 N|\mathcal{L}|)$ assuming constant cost of `PROVE` (see the following discussion).

Recall that with $Q|_v$ we denote the restriction of $Q$ to a variable $v$ and its inputs. The call $\texttt{PROVE}(\phi, Q|_v)$ is the application of model checking techniques to prove that $Q|_v$ respects the property $\phi$. The key to the performance of our implementation is that checking $\phi$ *locally* is extremely fast. Since we are able to prove stabilization of the entire system while only ever applying `PROVE` to small parts of the system, our procedure is very efficient. That, coupled with the fact that `PROVE` calls can be executed in batches and thus in parallel on as many processors as are available, makes the method scalable.

Theorem 1 [36] establishes the soundness and completeness of our method.

*Domain specific optimization.* Until now we have presented a general procedure that works with most models of concurrent update, and all possible update relations (not just those defined per variable to follow the target functions). However, due to specific target functions used in biological networks, we can reimplement the lemma generation routine in a way that PROVE is never needed, leading to significant performance improvements. Our alternative procedure F-NEWLEMMAS is shown in Alg. 4. We consider a variable $v$ and its inputs

**Alg 1.** Stabilization proving procedure

**input** $Q$ : QualitativeNetwork($N$)
**output** fixpoint or counterexample
  $(V^{\min}, V^{\max}) \coloneqq$ GenLemmas($Q$)
  **if** $\big(\forall v \in V \;.\; V^{\min}(v) = V^{\max}(v)\big)$ **then**
    **return** stabilizing at fixpoint $V^{\min}$
  **else if** Search($V^{\min}, V^{\max}, Q$) finds a counter-example $\pi$ **then**
    **return** non-stabilizing with counterexample $\pi$
  **else**
    find single trivial fixpoint $V$
    **return** stabilizing at fixpoint $V$

**Alg 2.** Lemma generation procedure GenLemmas

**input** $Q$ : QualitativeNetwork($N$)
**output** $V^{\min}, V^{\max}: V \to \{0, 1, \dots N\}$
  $\mathcal{F} \coloneqq \varnothing$;  $\mathcal{P} \coloneqq \varnothing$
  $\forall v \in V, v$ constant  . $V^{\min}(v) \coloneqq v \wedge V^{\max}(v) \coloneqq v$
  $\forall v \in V, v$ non-constant . $V^{\min}(v) \coloneqq 0 \wedge V^{\max}(v) \coloneqq N$
  **for all** non-constant variable $v \in Q$ **do**
    $\mathcal{F} \coloneqq \mathcal{F} \cup \{v\}$
  **while** $\mathcal{F} \neq \varnothing$ **do**
    $w \coloneqq$ pick a variable from $\mathcal{F}$
    **for all** variable $v \in$ outputs($w$) **do**
      **for all** lemma $l \in$ NewLemmas($v, V^{\min}, V^{\max}$) **do**
        $\mathcal{F} \coloneqq \mathcal{F} \cup \{v\}$
        $\mathcal{P} \coloneqq \mathcal{P} \cup \{l\}$
        update $V^{\min}(v), V^{\max}(v)$ with respect to $l$
  **return** $(V^{\min}, V^{\max})$

**Alg 3.** Lemmas generation procedure NewLemmas

**input** $v$ : variable
**input** $V^{\min}, V^{\max}: V \to \{0, 1, \dots N\}$
**output** $S$ - lemmas
  $\mathcal{S} \coloneqq \varnothing$
  $(w_1, w_2, \dots, w_m) \coloneqq$ inputs($v$)
  $p \coloneqq \big(\bigwedge_i V^{\min}(w_i) \le w_i \le V^{\max}(w_i)\big)$
  **for all** predicate $q \in \mathcal{L}$ over $v$ that strengthen $V^{\min}(v)$ or $V^{\max}(v)$ **do**
    $l \coloneqq (FG(p) \Rightarrow FG(q))$
    **if** PROVE$(l, Q|_v)$ **then**
      $S \coloneqq S \cup \{l\}$
  **return** $S$

**Alg 4.** Domain-specific fast lemma generation F-NewLemmas

**input** $v$ : variable
**input** $V^{\min}, V^{\max}: V \to \{0, 1, \dots N\}$
  $(w_1, w_2, \dots, w_m) \coloneqq$ inputs($v$)
  $p \coloneqq \big(\bigwedge_i V^{\min}(w_i) \le w_i \le V^{\max}(w_i)\big)$
  $T \coloneqq T_v(\times_i [V^{\min}(w_i), V^{\max}(w_i)])$
  **return** $\{FG(p) \Rightarrow FG(\min(T) \le v \le \max(T))\}$

$w_1, w_2 \ldots w_m$. Instead of guessing the influence of inputs under the constraints $V^{\min}$ and $V^{\max}$ on the output $v$, we compute it *exactly*. Namely, we compute the set $T$ of values of target function $T_v$ applied to all possible input combinations:

$$T = T_v([V^{\min}(w_1), V^{\max}(w_1)] \times [V^{\min}(w_2), V^{\max}(w_2)] \times \cdots \times [V^{\min}(w_m), V^{\max}(w_m)]),$$

thus obtaining a new approximation for $v$: $\min(T) \le v \le \max(T)$. In Theorem 2 [36] we prove that the lemmas generated by F-NewLemmas indeed hold.

The worst-case cost of the stabilization proving procedure using F-NewLemmas is $\mathcal{O}(n^2 N^{d+1})$ where the network has $n$ variables, of maximal indegree $d$ ($N^d$ results from generating input combinations). Since in all of our examples $N$ is small, this procedure works exceptionally fast (see experimental results in Section 4). If $N$ were larger, the procedure with NewLemmas could be in principle more efficient than F-NewLemmas.

*Search for counterexamples.* In Alg. 1, if the lemmas do not imply stabilization then the procedure Search is called to search for a counterexample, or exhaustively show that no counterexample exists. Search uses the bounds $V^{\min}$ and $V^{\max}$ computed earlier to limit the search space that is exhaustively explored.

Search is designed to find one of two types of counterexamples: multiple trivial fixpoints and non-trivial cycles. In the case of multiple fixpoints, Search encodes the problem of existence of at least two fixpoints of the system of length 1 as an instance of a formula satisfiability problem. A decision procedure is used to search for the existence of two different states: $(v_1, \ldots v_n)$ and $(w_1, \ldots, w_n)$ such that each of them is a fixpoint: $\forall i \in \{1 \ldots n\} \; . \; (v'_i = v_i \wedge w'_i = w_i) \quad \wedge$ $\exists i \in \{1 \ldots n\} \; . \; (v_i \neq w_i)$, where the next state, $v'$, is determined from $v's$ inputs by (1). We can ignore reachability here because the set of initial states is equal to the set of all possible state configurations. Note also that, for efficiency, we conjoin the system with extra constraints using $V^{\min}$ and $V^{\max}$: $\forall v \in \{v_1 \ldots v_n, w_1, w_n\} \; . \; V^{\min}(v) \le v \le V^{\max}(v)$. Experimentally we find that the information from the proved lemmas leads to tremendous speedups when searching for multiple fixpoints. Satisfiability of the query proves the existence of at least two different fixpoints. If it is unsatisfiable, the system is cyclic or terminating. In the next phase we search for a non-trivial cycle counterexample.

To find a non-trivial cycle we use bounded model-checking [30] together with the encoding of liveness to safety found in [31]. For efficiency, as we unroll the system $Q$ during bounded model checking, we conjoin the system with constraints on the values of the variables that come from the proven lemmas. Again we find that the information from the proved lemmas leads to tremendous speedups when searching for non-trivial cycles. Termination of the unrolling uses a naive diameter check [30], leading to a sound and complete technique. Fortunately we know only of toy examples where a search to the system's diameter is necessary.

## 4 Experimental results

We have implemented Alg. 1 in a tool called BioCheck, using Cadence SMV [32] as the implementation of `PROVE` and Z3 [33] as the decision procedure. The

| Model | N+1 | #variables | #edges | Model | N+1 | #variables | #edges |
|---|---|---|---|---|---|---|---|
| SSkinFxd | 4 | 25 | 45 | Skin3DFxd | 4 | 1200 | 2420 |
| SkinFxd | 4 | 60 | 90 | Skin3D | 4 | 1200 | 2420 |
| ESkin6Fxd | 4 | 72 | 108 | Diabetes8days | 3 | 75 | 148 |
| ESkin7Fxd | 4 | 84 | 126 | Diabetes15weeks | 3 | 75 | 148 |
| ESkin8Fxd | 4 | 96 | 144 | VPC4 | 3 | 48 | 92 |
| Skin2DFxd | 4 | 300 | 530 | VPC6 | 3 | 72 | 138 |
| Skin2D | 4 | 300 | 530 | PairRule(EctoEve) | 4 | 7 | 23 |

**Tab. 1.** Biological examples tested. $N+1$ indicates the granularity of the network; #variables and #edges represent the number of variables (nodes) and the number of interactions between variables (edges), respectfully, in the model. The skin models Skin2D and Skin3D contain bugs that our tool found for the first time. The repaired versions are suffixed with Fxd.

NewLemmas procedure is easily parallelized: the local lemmas are proved in batches rather than one-by-one. All experiments were performed on a PC equipped with 4GB memory and a quad-core Intel processor with hyper-threading.

*Biological systems tested.* Information about the examples used during our experimental evaluation can be found in Tab. 1. These models are variations on four base systems: skin, diabetes, VPC and pair-rule genes.

The mammalian epidermis model [12], SkinFxd, consists of 5 cells, each containing 12 variables. We tested a simplified version, SSkinFxd, where only 5 variables per cell directly relevant to stabilization were considered (Fig. 1). We also built elongated variants of this model: ones that consist of more than 5 cells, ESkin6-8Fxd, and ones that emulate multidimensional skin tissue. Skin2DFxd contains $4\times5$ cells (240 variables) and represents skin cross-section. Skin3DFxd consists of $4\times5\times5(=100)$ 3-D mesh of cells (1200 variables). Note that, using our tool, we are the first to find a bug in the skin model from [12] (Section 2).

The model of several molecular pathways operating in type-2 diabetes and chronic obesity [17], Diabetes, exists in two variants that differ in constants: a variant after 8 days or 15 weeks after mice started being fed a fatty diet.

The vulval precursor cells (VPC) model [18], is a model of cell fate determination in the formation of *C. elegans* vulva. The model VPC4 includes 4 cells. In nature, there are 6 precursor cells, but the model was reduced by its author to 4 cells to make analysis by other tools tractable. Our tool easily handles the extended model VPC6, which includes 6 cells.

We also tested a genetic regulatory network, PairRule, that models genes operating during segmentation in the *Drosophila* embryo [19], and a mutant of this network, PairRuleEctoEve, with ectopic expression of the *even-skipped* gene. Sanchez *et al.* [19] report the former model non-stabilizing and the latter stabilizing, which is confirmed by our results. As the *pair-rule* model is very small, the time to analyze it is negligible and is not included in the performance comparison.

*Results.* The comparison between our tool and existing tools is presented in Tab. 2. In this table we have compared the following tools:

– BC is our tool BioCheckimplementing Alg. 1 and 3 (NewLemmas).

| Model | Result | BC | FBC | NAIVE | TRM$^\alpha$ | SMV$_1^\alpha$ | SMV$_2^\alpha$ | QNB | SPN$^\alpha$ | VIS |
|---|---|---|---|---|---|---|---|---|---|---|
| SSKINFXD | proved | 3.8 | 0.0 | T | T | M | T | M | T | T |
| SKINFXD | proved | 9.0 | 0.0 | T | T | M | T | M | T | T |
| ESKIN6FXD | proved | 10.6 | 0.0 | T | T | M | T | M | T | T |
| ESKIN7FXD | proved | 12.9 | 0.0 | T | T | M | T | M | T | T |
| ESKIN8FXD | dispr. | 12.3 | 1.0 | 2.1 | T | M | M | M | T | T |
| SKIN2DFXD | proved | 50.3 | 0.0 | T | T | M | M | M | T | T |
| SKIN2D | dispr. | 56.5 | 13.1 | T | T | M | M | M | M | T |
| SKIN3DFXD | proved | 257.3 | 0.1 | T | T | V | M | V | M | T |
| SKIN3D | dispr. | 396.8 | 182.8 | T | T | V | M | V | M | T |
| DIABETES8DAYS | proved | 4.9 | 0.0 | T | T | M | T | M | T | T |
| DIABETES15WEEKS | proved | 5.2 | 0.0 | T | T | M | T | M | T | T |
| VPC4 | proved | 4.6 | 0.0 | T | T | T | T | M | T | T |
| VPC6 | proved | 7.0 | 0.0 | T | T | M | T | M | T | T |

**Tab. 2.** Comparison of our approach with other tools. BC is found in Alg. 1 in Section 3. FBC is the domain-specific version of BC using F-NEWLEMMAS instead of NEWLEMMAS. Runtimes are given in seconds. T indicates a timeout, where the threshold was set to 20mins. M represents an out-of-memory exception. The memory threshold was set to 4GB. V indicates tool failure after reporting too many variables.

- FBC is BIOCHECK using F-NEWLEMMAS (Alg. 4) instead of NEWLEMMAS.
- NAIVE is an implementation of bounded model checking using a diameter check as the termination condition, *i.e.* NAIVE($Q$) = SEARCH($\varnothing, \varnothing, Q$).
- TRM$^\alpha$ is the application of TERMINATOR [24] to solve a slightly different problem than stabilization (as stabilization itself is not encodable using LTL). For all the models that do stabilize, we test if the provided fixpoint is eventually reached. For those that do not guarantee stabilization we look for a non-trivial cycle. We use the symbol $\alpha$ to remind the reader that this application is not solving quite the same problem as stabilization.
- SMV$_1^\alpha$ and SMV$_2^\alpha$ are applications of Cadence SMV and NUSMV [34] respectively to the same problem as is used in TRM$^\alpha$.
- QNB is a tool from [12] that computes infinitely-often visited states in a network. For the comparison in Tab. 2, we could only use the tool that treats a system as a whole, rather than the version using the system's hierarchical structure to accelerate the whole-system reachable states computation. This acceleration-based technique has not been implemented. When applied manually to the example SKIN, on similar hardware, the acceleration-based technique took 21 mins. (see [12]). With some help by the author of the tool, we have established that the acceleration-based technique still would not be able to handle our larger examples.
- SPN$^\alpha$ is the application of Spin [35] on the same formulas as in TRM$^\alpha$.
- VIS is used in our experiments to symbolically compute the model's reachable state spaces, from which we look for a stable state.

Note that all previously known approaches fail to scale to the larger examples. For example, in the column TRM$^\alpha$ the encoding creates a program that, in essence, forces the liveness prover to find termination arguments for each possible path through the loop, which is a *very* large set (*e.g.* SKINFXD contains $3^{60}$ such

| Model | GenLemmas | Opt. GenLem. | Search | Proof / CEX Size | Model | GenLemmas | Opt. GenLem. | Proof Size |
|---|---|---|---|---|---|---|---|---|
| SkinFxd | 9.0 | 0.0 | | 177 | ESkin6Fxd | 10.6 | 0.0 | 212 |
| ESkin8Fxd | 11.3 | 0.0 | 1.0 | 2 / 74 | ESkin7Fxd | 12.9 | 0.0 | 251 |
| Skin2D | 43.4 | 0.0 | 13.1 | 2 / 215 | Skin2DFxd | 50.3 | 0.0 | 926 |
| Skin3D | 214.1 | 0.1 | 182.7 | 2 / 860 | Skin3DFxd | 257.3 | 0.1 | 3896 |
| VPC4 | 4.6 | 0.0 | | 75 | Diabetes8days | 4.9 | 0.0 | 132 |
| VPC6 | 7.0 | 0.0 | | 107 | Diabetes15weeks | 5.2 | 0.0 | 132 |
| PairRule | 4.8 | 0.0 | 1.6 | 2 / 4 | PairRuleEctoEve | 1.7 | 0.0 | 8 |

**Tab. 3.** Experimental details of application of our tool to the examples. Proof size is given as the number of implications in the proof, if the stabilization was proved. Otherwise counterexample size is given as "CEX size", which is cycle length and number of variables involved in the cycle. All times are given in seconds.

| Mesh (#cells) | #Variables | Optimized GenLemmas [s] | Mesh (#cells) | #Variables | Optimized GenLemmas [s] |
|---|---|---|---|---|---|
| $10 \times 10 \times 5$ | $6.0 \cdot 10^3$ | 0.8 | $75 \times 75 \times 5$ | $3.4 \cdot 10^5$ | 57.4 |
| $10 \times 20 \times 5$ | $1.2 \cdot 10^4$ | 1.6 | $100 \times 100 \times 5$ | $6.0 \cdot 10^5$ | 103.8 |
| $20 \times 20 \times 5$ | $2.4 \cdot 10^4$ | 3.6 | $100 \times 200 \times 5$ | $1.2 \cdot 10^6$ | 208.5 |
| $10 \times 50 \times 5$ | $3.0 \cdot 10^4$ | 4.5 | $200 \times 200 \times 5$ | $2.4 \cdot 10^6$ | 423.0 |
| $20 \times 50 \times 5$ | $6.0 \cdot 10^4$ | 9.8 | $100 \times 500 \times 5$ | $3.0 \cdot 10^6$ | 544.3 |
| $50 \times 50 \times 5$ | $1.5 \cdot 10^5$ | 25.0 | $200 \times 500 \times 5$ | $6.0 \cdot 10^6$ | $M$ |

**Tab. 4.** Performance of our tool FBC on scaled-up variants of the Skin3DFxd model. M represents an out-of-memory exception.

paths). For this reason, Terminator times out. In the case of SMV$^\alpha$, the SkinFxd has $4^{60}$ reachable states, which exceeds the typical limits of symbolic model checking tools. Note that unlike Naive, our implementation of Search with range restrictions does scale. This shows how the range restrictions that come from the lemmas help reduce the state space significantly.

Note that in the case of the non-stabilizing ESkin8Fxd algorithm, our lemma generation procedure performs worse than the naive method. This demonstrates (as mentioned in Section 1) that our lemma generation procedure could in cases hinder rather than help performance.

Table 3 contains more statistics about the results of our tool during the experimental evaluation. The optimized lemma generation procedure performs an order of magnitude faster than the one that uses a model checker. The size of the counterexamples found corresponds to the number of nodes in the network that haven't been fixed by the proof procedure (not shown); meaning that the proof procedure comes close to a counterexample.

In Tab. 4 we check how our proof procedure scales to larger examples. We run them on models containing up to $10^4$ cells (with or without bug) with a 10min timeout. The NewLemmas-based implementation does not time out on exactly one of these examples, In contrast, the F-NewLemmas-based implementation successfully verifies all but the $200 \times 500$ mesh model.

*Release.* We provide a *preliminary* packaging of the tool and benchmarks used at: `http://www.cs.vu.nl/~ekr/BioCheck` .

## 5 Conclusions

This paper reports on new advances in the area of formal analysis for biological models. We have addressed the open problem of scalable stabilization proving with a new sound and complete modular proof procedure. Our procedure takes advantage of the fact that, in practice, we can limit the set of possible modular proofs from which we search to those where the local lemmas are of a very restricted form. This leads to tremendous speedups, both for proving as well as disproving stabilization. It seems that it is the inherent robustness of the biological systems that makes our technique work so well—evolutionary developed systems remain naturally stable in the presence of timing and concentration variations.

*Future work.* While stabilization is a liveness property, safety techniques can in principle be used when checking finite-state systems. In future we might find useful adaptations of the circular proof rules (*e.g.* [9,10]) for the purpose of proving stabilization. Furthermore, it might be interesting to adapt our procedure to prove additional liveness properties beyond stabilization, and fair concurrent execution. Finally, the models that we have examined do not model certain important aspects, such as probabilities, aging or cell cycle which should fit in well with our current framework.

*Acknowledgments.* We thank M. Schaub for discussions about QNBuilder; C. Chaouiya for Drosophila models; S. Sankaranarayanan for clarifying the relation to tail invariants; and all the reviewers for their insightful comments.

## References

1. Fisher, J., Henzinger, T.A.: Executable biology. Proc. WSC, pp 1675–1682, 2006.
2. Bonzanni, N., Feenstra, A.K., Fokkink, W., Krepska, E.: What can formal methods bring to systems biology? In Proc. FM, LNCS 5850, pp 16–22, 2009.
3. Heath, J.: The equivalence between biology and computation. In Proc. CMSB, LNBI 5688, pp 18–25, 2009.
4. Fisher, J. *et al.*: Predictive modeling of signaling crosstalk during *C. elegans* Vulval Development. In PLoS CB, 3(5):e92, 2007.
5. Heath, J., Kwiatkowska, M., Norman, G., Parker G., Tymchyshyn, O.: Probabilistic model checking of complex biological pathways. CMSB'06, LNCS 4210:32–47.
6. Clarke, E., Faeder, J., Langmead, C., Harris, L., Jha, S., Legay, A.: Statistical model checking in *BioLab*: Applications to the automated analysis of T-Cell receptor signaling pathway. In Proc. CMSB'08, LNCS 5307, 231–250.
7. Chabrier-Rivier, N., Chiaverini, M., Danos, V., Fages, F., Schächter, V.: Modeling and querying biomolecular interaction networks. *Theo Comp Sci*,325(1):25-44,2004.
8. Zotin, A.I.: The stable state of organisms in thermodynamic bases of biological processes: Physiological Reactions and Adaptations, De Gruyter, 1990.
9. Jones, C.: Specification and design of (parallel) programs. IFIP Congr.'83, 321-32.
10. Pnueli, A.: In transition from global to modular temporal reasoning about programs. In Logics and Models of Concurrent Systems, pp 123-144, 1985.
11. Abadi, M., Lamport, L.: Composing specifications. TOPLAS 15(1):73–132, 1993.

12. Schaub, M. *et al.*: Qualitative networks: A symbolic approach to analyze biological signaling networks. In BMC Systems Biology, 1:4, 2007.
13. Thomas, R., Thieffry, D., Kaufman, M.: Dynamical behaviour of biological regulatory networks—I. Biological role of feedback loops and practical use of the concept of the loop-characteristic state. Bullet. of Math. Bio., 55(2):247–276, 1995.
14. Naldi, A., Thieffry, D., Chaouiya, C.: Decision diagrams for the representation oand analysis of logical models of genetic networks. LNBI 4695, pp 233–247, CMSB'07.
15. Halbwachs, N., Lagnier, F., Ratel, C.: Programming and verifying critical systems by means of the synchronous data-flow programming language LUSTRE. In IEEE Transactions on Software Engineering 18(9):785–793, 1992.
16. Pnueli, A.: The temporal logic of programs. In Proc. FOCS, pp 46–57, 1977.
17. Beyer, A., Fisher, J.: Unpublished results, 2009.
18. Beyer, A., *et al.*: Mechanistic insights into metabolic disturbance during type-II diabetes and obesity using qualitative networks. TCSB, LNBI 5945:146–162, 2010.
19. Sanchez, L., Thieffry, D.: Segmenting the fly embryo: a logical analysis fo the *pair-rule* cross-regulatory module. Journal of Theoretical Biology 224, pp 517–537, 2003.
20. Ropers, D., Baldazzi, V., de Jong, H.: Model reduction using piecewise-linear approximations preserves dynamic properties of the carbon starvation response in E. coli. IEEE/ACM Trans. on Comp. Bio. and Bioinf., vol. 99, preprint, 2009.
21. Ghosh, R., Tomlin, C.: Symbolic reachable set computation of piecewise affine hybrid automata and its application to biological modelling: Delta-Notch protein signalling. IEE Systems Biology, 1(1):170–183, 2004
22. Podelski, A., Wagner, S.: A sound and complete proof rule for region stability of hybrid systems. In HSCC, LNCS 4416, pp 750–753, 2007.
23. Oehlerking, J., Theel, O.: Decompositional construction of Lyapunov functions for hybrid systems. In HSCC, LNCS 5469, pp 276–290.
24. Cook, B., Gotsman, A., Podelski, A., Rybalchenko, A., Vardi, M.Y.: Proving that programs eventually do something good. In Proc. POPL, pp 265-276, 2007.
25. Moore, J.S.: A mechanically checked proof of a multiprocessor result via a uniprocessor view. In FMSD, 14(2):213228, 1999.
26. McMillan, K.: Circular compositional reasoning about liveness. In Proc. CHARME, LNCS 1703, pp 342–345, 1999.
27. Cousot, P. and Cousot, R.: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In Proc. POPL, pp 238–252, 1977.
28. Colón, M. and Sipma, H.: Practical Methods for Proving Program Termination. In Proc. CAV, pp 442–454, 2002.
29. Lowell, S. *et al*: Stimulation of human epidermal differentiation by delta-notch signalling at the boundaries of stem-cell clusters. Curr Biol. 4;10(9):491-500, 2000.
30. Clarke, E., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. In Proc. FMSD, 19(1):7–34, 2001.
31. Biere, A., Artho, C., Schuppan, V.: Liveness checking as safety checking. In Proc. FMICS, ENTCS 66(2):160–177, 2002.
32. McMillan, K.: Symbolic model checking (PhD thesis), Kluwer, 1993.
33. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In Proc. TACAS, LNCS 4963, pp 337–340, 2008.
34. Cimatti, A. *et al.*: NuSMV 2: An open-source tool for symbolic model checking In Proc. CAV, LNCS 2404, 2002.
35. Holzmann, G,: The SPIN model checker: Primer and ref. manual, Wesley, 2003.
36. Appendix. TODO.