Model-Checking Signal Transduction Networks through Decreasing Reachability Sets

Koen Claessen[1], Jasmin Fisher[2], Samin Ishtiaq[2], Nir Piterman[3], and Qinsi Wang[4]

[1] Chalmers University
[2] Microsoft Research Cambridge
[3] University of Leicester
[4] Carnegie Mellon University

Abstract. We consider model checking of *Qualitative Networks*, a popular formalism for modeling signal transduction networks in biology. One of the unique features of qualitative networks, due to them lacking initial states, is that of "reducing reachability sets". Simply put, a state that is not visited after i steps will not be visited after i' steps for every $i' > i$. We use this feature to create a compact representation of all the paths of a qualitative network of a certain structure. Combining this compact path representation with LTL model checking leads to significant acceleration in performance. In particular, for a recent model of Leukemia, our approach works at least 5 times faster than the standard approach and up to 100 times faster in some cases. Our approach enhances the iterative hypothesis-driven experimentation process used by biologists, enabling fast turn-around of executable biological models.

1 Introduction

Formal verification methods hold great promise for biological research. Over the years, various efforts have repeatedly demonstrated that the use of formal methods is beneficial for gaining new biological insights as well as directing new experimental avenues. Experimental biology is an iterative process of hypothesis-driven experimentation of a particular biological system of interest. The idea to boost this process using formal executable models describing aspects of biological behaviors, also known as Executable Biology, has proved to be successful in cell biology and shed new light on cell signalling and cell-cell communication.

Biological systems can be modelled at different levels of abstraction. On a cellular level, we usually consider the causality relations between molecular species (e.g., genes, proteins) inside the cell (collectively called signal transduction networks), and between cells (intercellular signalling). These can be described by various state-transition systems such as compositional state machines, Petri nets, and process calculi.

One successful approach to the usage of abstraction in biology has been the usage of Boolean networks [23]. Boolean networks call for abstracting the status of each modeled substance as either active (on) or inactive (off). Although a

very high level abstraction, it has been found useful to gain better understanding of certain biological systems [20, 22]. The appeal of this discrete approach along with the shortcomings of the very aggressive abstraction, led researchers to suggest various formalisms such as Qualitative Networks [21] and Gene Regulatory Networks [16] that allow to refine models when compared to the Boolean approach. In these formalisms, every substance can have one of a small discrete number of levels. Dependencies between substances become algebraic functions instead of Boolean functions. Dynamically, a state of the model corresponds to a valuation of each of the substances and changes in values of substances occur gradually based on these algebraic functions. Qualitative networks and similar formalisms (e.g., genetic regulatory networks [23]) have proven to be a suitable formalism to model some biological systems [3, 20, 21, 23].

Here, we consider model checking of qualitative networks. One of the unique features of qualitative networks is that they have no initial states. That is, the set of initial states is the set of all states. Obviously, when searching for specific executions or when trying to prove a certain property we may want to restrict attention to certain initial states. However, the general lack of initial states suggests a unique approach towards model checking. It follows that if some state is not reachable by exactly n steps, for some n, it will not be reachable by exactly n' steps, for every $n' > n$. These "decreasing" sets of reachable states allow to create a more efficient symbolic representation of all the paths of a certain length.

However, this observation alone is not enough to create an efficient model checking procedure. Indeed, accurately representing the set of reachable states at a certain time amounts to the original problem of model checking (for reachability), which does not scale. In order to address this we use an over-approximation of the set of states that are reachable by exactly n steps. We represent the over-approximation as a Cartesian product of the set of values that are reachable for each variable at every time point. The computation of this over-approximation never requires us to consider more than two adjacent states of the system. Thus, it can be computed quite efficiently. Then, using this over-approximation we create a much smaller encoding of the set of possible paths in the system.

Finally, an LTL formula is translated to an additional set of constraints on the set of paths. Our encoding is based on temporal testers [17].

We test our implementation on many of the biological models developed using Qualitative Networks. The experimental results show that there is significant acceleration when considering the decreasing reachability property of qualitative networks. In many examples, in particular larger and more complicated biological models, this technique leads to considerable speedups. The technique scales well with increase of size of models and with increase in length of paths sought for.

These results are especially encouraging given the methodology biologists have been using when employing our tools [2]. Typically, models are constructed and then compared with experimental results. The process of model development is a highly iterative process involving trial and error where the biologist compares a current approach with experimental results and refines the model until it matches current experimental knowledge. In this iterative process it is

important to give fast answers to queries of the biologist. We hope that with the speed ups afforded by this new technique, model checking could be incorporated into the routine methodology of experimental biologists using our tools.

1.1 Related work

Over the last decade, the usage of model checking has proven to be extremely useful for the analysis of discrete biological models (e.g., [4,5,9,12,13,21]), leading to numerous new biological findings. This is part of a general "trend" to adopt (and adapt) approaches from formal verification and formal methods for usage in biological modeling [1, 11, 12, 14].

Analysis of Boolean Networks and Qualitative Networks has been done either manually or with ad-hoc techniques. The main form of analysis applied is that of stability. That is, identifying in which states the system can remain even after very long executions. In particular, if all states of the system (or most of them) lead to the same loop and if that loop is small (in particular if it consists of one state) then the network is more "stable". Traditionally, analysis was done by manually inspecting the graphs of configurations that the system gives rise to [10, 15, 19].

Obviously, this approach severely limits the size of systems that can be analyzed. As qualitative networks define finite-state systems, the technology to support model checking through existing model checkers is widely available. Previous attempts to use standard model checking tools and techniques on these types of biological models have proven unsatisfactory [8]. Both SAT-based and BDD-based techniques could not scale to handle large models.

In previous work (*e.g.* [21], [8]) custom techniques have been developed for reasoning about the *final states* reached by biological models. The technique in [8] is related to our results as it uses a similar abstraction domain to reason about sets of states of a Qualitative Network. However, the approach in [8] is specifically tailored for stabilization and it was not clear how to apply it to model checking (or path representation). It is also more aggressive than the approach advocated here and scales to larger models.

In [18], the techniques in [8] are used for reasoning about hardware designs. They show how to use these techniques to handle a restricted subset of LTL.

1.2 Structure of the paper

The paper is structured as follows. In Section 2 we introduce qualitative networks and their usage through an example. In Section 3 we give the formal background and fix notations. Then, in Section 4 we explain the decreasing reachability concept and the abstraction that we use with it. We also include a brief explanation of the types of paths we are looking for. We then give some experimental results (Section 5). We conclude in Section 6.

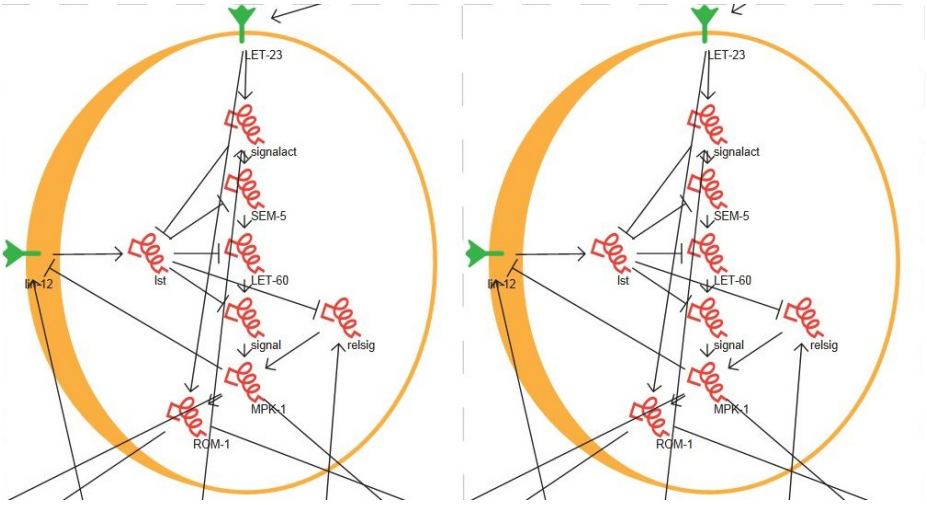

Fig. 1. A pictorial view of part of a model describing aspects of cell-fate determination during *C. elegans* vulval development [12]. The image shows two cells having the "same program". Neighboring cells and connections between cells are not shown.

2 Qualitative Networks Example

We start with an example giving some introduction to Qualitative Networks and the usage of LTL model checking in this context.

Figure 1 shows a model representing aspects of cell-fate determination during *C. elegans* vulval development [12]. The part shown in the figure includes three cells. Each cell in the model represents a *vulval precursor cell* and the elements inside it represent proteins whose level of activity influences the decision of the cell as to which part of the vulva the descendants of the cell should form. All cells execute the same program and it is the communication between the cells themselves as well as communication between the cells and additional parts of the model (i.e., external signals) that determine a different fate for each of the cells. Understanding cell-fate determination is crucial for our understanding of normal development processes as well as occasions where these go wrong such as disease and cancer. The pictorial view gives rise to a formal model expressed as a qualitative network [21]. Formal definitions are in the next section.

Each of the cells in the model includes executing components, for example `LET-60`, that correspond to a single variable. Each variable v holds a value, which is a number in $\{0, 1, \ldots, N_v\}$, where N_v is the *granularity* of the variable. Specifically, in Figure 1 all variables range over $\{0, 1, 2\}$. A *target function*, T_v, defined over the values of variables affecting v (i.e., having incoming arrows into v), determines how v is updated: if $v < T_v$ and $v < N_v$ then $v' = v + 1$, if $v > T_v$ and $v > 0$ then $v' = v - 1$, else v does not change. In a qualitative network all variables are updated synchronously in parallel.

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

$$T_v(w_1, w_2, \ldots w_{s+r}) = \max\left(0, \left\lfloor \frac{1}{s}\sum_{k=1}^{s} w_k - \frac{1}{r}\sum_{k=1}^{r} w_{s+k} + \frac{1}{2} \right\rfloor\right)$$

Graphically, this is often represented as an *influence graph* with $\rightarrow$ edges between each of $w_1, w_2, \ldots, w_s$ and v and $\multimap$ edges between each of $w_{s+1}, w_{s+2}, \ldots, w_{s+r}$ and v. More complicated target functions can be defined using algebraic expressions over $\{w_1, \ldots, w_{s+r}\}$. We refer the reader to [2, 3, 20, 21] for further details about other modeling options.

Specifically, in the model above, the target of `lst` is:

$$T_{\texttt{lst}} = min(2 - \texttt{signalact}, 1) * \texttt{lin-12}$$

This models activation by `lin-12` and inhibition by `signalact`. However, inhibition occurs only when `signalact` is at its maximal level (2). When inhibition is not maximal the target follows the value of `lin-12`. The target of `SEM-5` is:

$$T_{\texttt{SEM-5}} = max(0, 2 - ((2 - \texttt{signalact}) * (max(\texttt{lst} - 1, 0) + 1)))$$

This function means that `lst` inhibits `SEM-5` and `signalact` activates it. However, activation takes precedence: inhibition takes effect only in case that activation is not at its maximum value (2), and only when inhibition is at its maximum value (2). Otherwise, the target follows the value of its activator (`signalact`).

Models are analyzed to ensure that they reproduce behavior that is observed in experiments. A mismatch between the model and experimental observations signifies that something is wrong with our understanding of the system. In such a case, further analysis is required in order to understand whether and how the model needs to be changed. Models are usually analyzed by simulating them and following the behavior of components. A special property of interest in these types of models is that of *stability*: there is a unique state that has a self loop and all executions lead to that state [8, 21]. When a model does stabilize it is interesting to check the value of variables in the stabilization point. In addition, regardless of whether the model is stabilizing or not, model checking is used to prove properties of the model or to search for interesting executions. For the model in Figure 1 the following properties, e.g., are of interest.

– Do there exist executions leading to adjacent primary fates in which increase of LS happens after down-regulation of `lin-12`?
  This property is translated to an LTL formula of the following format:

  $$\theta \wedge FGf_{i,j} \wedge (\neg d_i U l_i) \wedge (\neg d_j U l_i),$$

  where θ is some condition on initial states, $f_{i,j}$ is the property characterizing the states in which VPCs i and j are both in primary fate, d_i is the property

that `lin-12` is low in VPC i, l_i is the property that d_i is high in VPC i, and d_j and l_j are similar for VPC j. This property is run in positive mode, i.e., we are searching for execution that satisfies this property.

– Is it true that for runs starting from a given set of states the sequence of occurrences leading to fate execution follows the pattern: MPK-1 increases to high level then `lin-12` is down-regulated, and then LS is activated.
This property is translated to an LTL formula of the following format:

$$\theta \implies F(m_i \wedge XF(l_i \wedge XFd_i)),$$

where θ is some condition on initial states, m_i is the property characterizing states in which VPC i has a high level of MPK-1, l_i is the property characterizing states in which VPC i has a low level of `lin-12`, and d_i is the property characterizing states in which VPC i has a high level of LS. This property is run in negative (model checking) mode, i.e., we are searching for executions falsifying this property and expecting the search to fail.

3 Formal Background

We formally introduce the Qualitative Networks (QN for short) framework and recall the definition of linear temporal logic (LTL for short).

Following [21], a *qualitative network* (QN), $Q(V,T,N)$, of granularity $N+1$ consists of variables: $V = (v_1, v_2 \dots v_n)$.[5] A state of the system is a finite map $s : V \to \{0,1,\dots N\}$. 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 $s = (d_1, d_2 \dots d_n)$, the *next state* $s' = (d'_1, d'_2 \dots d'_n)$ is computed by:

$$d'_i = \begin{cases} d_i + 1 & d_i < T_i(s) \text{ and } d_i < N, \\ d_i - 1 & d_i > T_i(s) \text{ and } d_i > 0, \\ d_i & \text{otherwise} \end{cases} \tag{1}$$

A target function of a variable v is typically a simple algebraic function, such as sum, over several other variables $w_1, w_2 \dots w_m$ (see, e.g., Section 2). 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. In the following, we use the term *network* to refer to a qualitative network.

A QN $Q(V,T,N)$ defines a state space $\Sigma = \{s : V \to \{0,1,\dots N\}\}$ and a transition function $f : \Sigma \to \Sigma$, where $f(s) = s'$ such that for every $v \in V$ we have $s'(v)$ depends on $T_v(s)$ as in Equation (1). For a state $s \in \Sigma$ we denote by $s(v)$ also by s_v. In particular, $f_v(s) = f(s)(v)$ is the value of v in $f(s)$. We say that a state s is *recurring* if it is possible to get back to s after a finite number of applications of f. That is, if for some $i > 0$ we have $f^i(s) = s$. As the state space

[5] For simplicity, we assume that all variables have the same range $\{0,\dots,N\}$. The extension to individual ranges is not complicated. Our implementation supports individual ranges for variables.

of a qualitative network is finite, the set of recurring states is never empty. We say that a network is *stabilizing* if there exists a unique recurring state s. That is, there is a unique state s such that $f(s) = s$, and for every other state s' and every $i > 0$ we have $f^i(s') \neq s'$. Intuitively, this means that starting from an arbitrary state, we always end up in a fixpoint and always the same one. A run of a QN $Q(V, T, N)$ is an infinite sequence $r = s_0, s_1, \ldots$ such that for every $i \geq 0$ we have $s_i \in \Sigma$ and $s_{i+1} = f(s_i)$.

We define linear temporal logic (LTL) over runs of qualitative networks as follows. For every variable $v \in V$ and every value $n \in \{0, 1, \ldots N\}$, we define an atomic proposition $v \bowtie n$, where $\bowtie \in \{>, \geq, \leq, <\}$. Let AP denote the set of all atomic propositions (for a network Q). The set of LTL formulas is:

$$\varphi ::= \mathsf{AP} \mid \varphi \vee \varphi \mid \neg\varphi \mid X\varphi \mid \varphi U \varphi$$

As usual, we introduce $\wedge$, $\rightarrow$, F, and G, as syntactic sugar.

An LTL formula φ is satisfied over a run $r = s_0, s_1, \ldots$ in location i, denoted $r, i \vDash \varphi$ according to the following:

- For $\varphi = v \bowtie n \in \mathsf{AP}$ we have $r, i \vDash \varphi$ if $s_i(v) \bowtie n$.
- For $\varphi = \neg\psi$ we have $r, i \vDash \varphi$ if it is not the case that $r, i \vDash \psi$.
- For $\varphi = \psi_1 \vee \psi_2$ we have $r, i \vDash \psi_1 \vee \psi_2$ if either $r, i \vDash \psi_1$ or $r, i \vDash \psi_2$.
- For $\varphi = X\psi$ we have $r, i \vDash \varphi$ if $r, i + 1 \vDash \psi$.
- For $\varphi = \psi_1 U \psi_2$ we have $r, i \vDash \varphi$ if there is $j \geq i$ such that $r, j \vDash \psi_2$ and for every $i \leq k < j$ we have $r, k \vDash \psi_1$.

We say that a run r satisfies an LTL formula φ, denoted $r \vDash \varphi$ if $r, 0 \vDash \varphi$. Given a Qualitative Network Q, we say that Q satisfies an LTL formula φ, denoted $Q \vDash \varphi$, if for every run r of Q we have $r \vDash \varphi$. In case that $Q \nvDash \varphi$ a *counter example* is a run r such that $r \nvDash \varphi$.

We use bounded model checking [7] for checking whether a qualitative network satisfies a given LTL formula φ. Intuitively, we search for a run of a certain structure (and length) that does not satisfy the formula by constructing a Boolean formula whose satisfiability corresponds to such a run. Searching for a counter example of length l means that we (1) create Boolean variables that represent the state of the system in l different time points, (2) add constraints that enforce that the transition of the qualitative network holds between every two consecutive time points, (3) add constraints that enforce that the transition of the qualitative network holds between the state at time $l - 1$ (last state) and some previous state (i.e., that the sequence of states ends in a loop), and (4) add Boolean variables and constraints that enforce satisfaction of the (negation of) the temporal property.

In order to create a Boolean encoding of the LTL formula we use a variant of the temporal testers approach in [17]. Specifically, for every temporal subformula (and every time point in the trace) we add a Boolean variable that tracks the truth value of the subformula at that time. The truth value of these variables are connected to the truth values of propositions (encoded through the state of the model) and truth values of other subformulas. In addition, we add constraints

Concrete Decreasing Reachability
1 $\Sigma_0 = \Sigma$;
2 $\Sigma_{-1} = \emptyset$;
3 $j := 0$;
4 while $(\Sigma_{j-1} \neq \Sigma_j)$ {
5 $\quad \Sigma_{j+1} = \Sigma_j \setminus \{s' \in \Sigma_j \mid \forall s \in \Sigma_j . s' \neq f(s)\}$;
6 $\quad j$++;
7 }
8 return $\Sigma_0, \ldots, \Sigma_{j-1}$;

Fig. 2. Computing decreasing reachability sets.

that enforce satisfaction of eventualities in the loop. In order to search for a trace that satisfies a certain LTL formula we add the encoding of the formula to the trace. Satisfiability then provides a run satisfying the formula. To prove that all runs up of a certain length satisfy a formula, we add the encoding of the negation of the formula to the trace. Unsatisfiability then provides a proof that no run (of the given length) satisfies the formula.

4 Decreasing Reachability Sets

A notable difference between QNs and "normal" transition systems is that QNs do not specify initial states. For example, for the classical stability analysis all states are considered as initial states. It follows that if a state s of a QN is not reachable after i steps, it is not reachable after i' steps for every $i' > i$. Thus, there is a decreasing sequence of sets $\Sigma_0 \supseteq \Sigma_1 \supseteq \cdots \supseteq \Sigma_l$ such that searching for runs of the network can be restricted to the set of runs of the form $\Sigma_0 \cdot \Sigma_1 \cdots (\Sigma_l)^\omega$. Here we show how to take advantage of this fact in constructing a more scalable model checking algorithm for qualitative networks.

Consider a Qualitative Network $Q(V, T, N)$ with set of states $\Sigma : V \to \{0, \ldots, N\}$. We say that a state $s \in \Sigma$ is reachable by exactly i steps if there is some run $r = s_0, s_1, \ldots$ such that $s = s_i$. Dually, we say that s is not reachable by exactly i steps if for every run $r = s_0, s_1, \ldots$ we have $s_i \neq s$.

Lemma 1. *If a state s is not reachable by exactly i steps then it is not reachable by exactly i' steps for every $i' > i$.*

The algorithm in Figure 2 computes a decreasing sequence $\Sigma_0 \supset \Sigma_1 \supset \cdots \supset \Sigma_{j-1}$ such that all states that are reachable by exactly i steps are in Σ_i if $i < j$ and in Σ_{j-1} if $i \geq j$. We note that the definition of Σ_{j+1} in line 5 is equivalent to the standard $\Sigma_{j+1} = f(\Sigma_j)$. However, we choose to write it as in the algorithm above in order to stress that only states in Σ_j are candidates for inclusion in Σ_{j+1}. Given the sets $\Sigma_0, \ldots, \Sigma_{j-1}$ every run $r = s_0, s_1, \ldots$ of Q satisfies $s_i \in \Sigma_i$ for $i < j$ and $s_i \in \Sigma_{j-1}$ for $i \geq j$. In particular, if $Q \not\models \varphi$ for some LTL formula φ then the run witnessing the unsatisfaction of φ can be searched for in this smaller space of runs. Unfortunately, the algorithm in Figure 2 is not feasible. Indeed, it amounts to computing the exact reachability sets of the QN Q, which does not scale well [8].

```
Abstract Decreasing Reachability
1  ∀v_i ∈ V . D_{i,0} := {0, 1, ... N};
2  ∀v_i ∈ V . D_{i,-1} := ∅;
3  j := 0;
4  while (∃v_i ∈ V . D_{i,j} ≠ D_{i,j-1}) {
5      foreach (v_i ∈ V) {
6          D_{i,j+1} := ∅;
7          foreach (d ∈ D_{i,j}){
8              if (∃(d_1, ..., d_m) ∈ D_{1,j} × ··· × D_{m,j} . f_v(d_1, ..., d_m) = d){
9                  D_{i,j+1} := D_{i,j+1} ∪ {d};
10             }
11         }
12     }
13     j++;
14 }
15 return ∀v_i ∈ V . ∀j' ≤ j . D_{i,j'};
```

Fig. 3. Over-approximating decreasing reachability sets.

In order to effectively use Lemma 1 we combine it with over-approximation, which leads to a scalable algorithm. Specifically, instead of considering the set Σ_k of states reachable at step k, we identify for every variable $v_i \in V$ the domain $D_{i,k}$ of the set of values possible at time k for variable v_i. Just like the general set of states, when we consider the possible values of variable v_i we get that $D_{i,0} \supseteq D_{i,1} \supseteq \cdots \supseteq D_{i,l}$. The advantage is that the sets $D_{i,k}$ for all $v_i \in V$ and $k > 0$ can be constructed by induction by considering only the knowledge on previous ranges and the target function of one variable.

Consider the algorithm in Figure 3. For each variable, it initializes the set of possible values at time 0 as the set of all values. Then, based on the possible values at time j it computes the possible values at time $j+1$. The actual check can be either implemented explicitly if the number of inputs of all target functions is small (as in most cases) or symbolically (see [6]). Considering only variables (and values) that are required to decide the possible values of variable v_i at time j makes the problem much smaller than the general reachability problem. Notice that, again, only values that are possible at time j need be considered at time $j+1$. That is, $D_{i,j+1}$ starts as empty (line 6) and only values from $D_{i,j}$ are added to it (lines 7–10). As before, $D_{i,j+1}$ is the projection of $f(D_{1,j} \times \ldots \times D_{m,j})$ on v_i. The notation used in the algorithm above stresses that only states in $D_{i,j}$ are candidates for inclusion in $D_{i,j+1}$.

The algorithm produces very compact information that enables to follow with a search for runs of the QN. Namely, for every variable v_i and for every time point $0 \leq k < j$ we have a decreasing sequence of domains

$$D_{i,0} \supseteq D_{i,1} \supseteq \cdots \supseteq D_{i,k}.$$

Consider a Qualitative Network $Q(V, T, N)$, where $V = \{v_1, \ldots, v_n\}$ and a run $r = s_0, s_1, \ldots$. As before, every run $r = s_0, s_1, \ldots$ satisfies that for every i and for every t we have $s_t(v_i) \in D_{i,t}$ for $t < j$ and $s_t(v_i) \in D_{i,j-1}$ for $t \geq j$.

We look for paths that are in the form of a lasso, as we explain below. We say that r is a *loop of length* l if for some $0 < k \leq l$ and for all $m \geq 0$ we have $s_{l+m} = s_{l+m-k}$. That is, the run r is obtained by considering a prefix of length $l-k$ of states and then a loop of k states that repeats forever. A search for a loop of length l that satisfies an LTL formula φ can be encoded as a bounded model checking query as follows. We encode the existence of l states $s_0, \ldots, s_{l-1}$. We use the decreasing reachability sets $D_{i,t}$ to force state s_t to be in $D_{0,t} \times \cdots \times D_{n,t}$. This leads to a smaller encoding of the states $s_0, \ldots, s_{l-1}$ and to smaller search space. We add constraints that enforce that for every $0 \leq t < l-1$ we have $s_{t+1} = f(s_t)$. Furthermore, we encode the existence of a time $l-k$ such that $s_{l-k} = f(s_{l-1})$. We then search for a loop of length l that satisfies φ. It is well known that if there is a run of Q that satisfies φ then there is some l and a loop of length l that satisfies φ. We note that sometimes there is a mismatch between the length of loop sought for and length of sequence of sets (j) produced by the algorithm in Figure 3. Suppose that the algorithm returns the sets $D_{i,t}$ for $v_i \in V$ and $0 \leq t < j$. If $l > j$, we use the sets $D_{i,j-1}$ to "pad" the sequence. Thus, states $s_j, \ldots, s_{l-1}$ will also be sought in $\Pi_i D_{i,j-1}$. If $l < j$, we use the sets $D_{i,0}, \ldots, D_{i,l-2}, D_{i,j-1}$ for $v_i \in V$. Thus, only the last state s_{l-1} is ensured to be in our "best" approximation $\Pi_i D_{i,j-1}$. A detailed explanation of how we encode the decreasing reachability sets as a Boolean satisfiability problem is given in [6].

5 Experimental Results

We implemented this technique to work on models defined through our tool BMA [2]. Here, we present experimental results of running our implementation on a set of benchmark problems. We collected a total of 22 benchmark problems from various sources (skin cells differentiation models from [8,21], diabetes models from [3], models of cell fate determination during *C. elegans* vulval development, a *Drosophila* embryo development model from [20], Leukemia models constructed by ourselves, and a few additional examples constructed by ourselves). The number of variables in the models and the maximal range of variables is reported in Table 1.

Our experiments compare two encodings. First, the encoding explained in Section 4, referred to as *opt* (for optimized). Second, the encoding that considers l states $s_0, \ldots, s_l$ where $s_t(v_i) \in \{0, \ldots, N\}$ for every t and every i. That is, in terms of the explanation in Section 4 for every variable v_i and every time point $0 \leq t \leq l$ we consider the set $D_{i,t} = \{0, \ldots, N\}$. This encoding is referred to as *naïve*. In both cases we use the same encoding to a Boolean satisfiability problem. Further details about the exact encoding can be found in [6].

We performed two kinds of experiments. First, we search for loops of length 10, 20, ⋯, 50 on all the models for the optimized and naïve encodings. Second, we search for loops that satisfy a certain LTL property (either as a counter example to model checking or as an example run satisfying a given property). Again, this is performed for both the optimized and the naïve encodings. LTL properties are considered only for four biological models. The properties were suggested

Model name	♯Vars	Range	Model name	♯Vars	Range
2var_unstable	2	0..1	Bcr-Abl	57	0..2
Bcr-AblNoFeedbacks	54	0..2	BooleanLoop	2	0..1
NoLoopFound	5	0..4	Skin1D_TF_0	75	0..4
Skin1D_TF_1	75	0..4	Skin1D	75	0..4
Skin2D_3cells_2layers_0	90	0..4	Skin2D_3cells_2layers_1	90	0..4
Skin2D_3cells_2layers_2	90	0..4	Skin2D_5X2_TF	198	0..4
Skin2D_5X2	198	0..4	SmallTestCase	3	0..4
SSkin1D_0	30	0..4	SSkin1D_TF_1	31	0..4
SSkin1D	30	0..4	SSkin2D_3cells_2layers	40	0..4
VerySmallTestCase	2	0..4	VPC_lin15ko	85	0..2
VPC_Non_stabilizing	33	0..2	VPC_stabilizing	43	0..2

Table 1. Number of variables in models and their ranges.

by our collaborators as interesting properties to check for these models. For both experiments, we report separately on the global time and the time spent in the SAT solver. All experiments were run on an Intel Xeon machine with CPU X7560@2.27GHz running Windows Server 2008 R2 Enterprise.

In Tables 2 and 3 we include experimental results for the search for loops. We compare the global run time of the optimized search vs the naïve search. The global run time for the optimized search includes the time it takes to compute the sequence of decreasing reachability sets. Accordingly, in some of the models, especially the smaller ones, the overhead of computing this additional information makes the optimized computation slower than the naïve one. For information we include also the net runtime spent in the SAT solver.

In Table 4 we include experimental results for the model checking experiment. As before, we include the results of running the search for counter example of lengths 10, 20, 30, 40, and 50. We include the total runtime of the optimized vs the naïve approaches as well as the time spent in the SAT solver. As before, the global runtime for the optimized search includes the computation of the decreasing reachability sets. The properties in the table are of the following form. Let I, $a - d$ denote formulas that are Boolean combinations of propositions.

- $I \rightarrow (\neg a)Ub$ – we check that the sequence of events when starting from the given initial states (I)satisfies the order b happens before a.
- $I \wedge FGa \wedge F(b \wedge XFc)$ – we check that the model gets from some states (I) to a loop that satisfies the condition a and the path leading to the loop satisfies that b happens first and then c.
- $I \wedge FGa \wedge F(b \wedge XF(c \wedge XFd))$ – we extend the previous property by checking the sequence b then c then d.
- $I \wedge FGa \wedge (\neg b)Uc$ – we check that the model gets from some states (I) to a loop that satisfies the condition a and the path leading to the loop satisfies that b cannot happen before c.
- $GFa \wedge GFb$ – we check for the existence of loops that exhibit a form of instability by having states that satisfy both a and b.

Length of loop	10				20				30			
	Global Time (Seconds)		Sat Time (Seconds)		Global Time (Seconds)		Sat Time (Seconds)		Global Time (Seconds)		Sat Time (Seconds)	
Model name	Naïve	Opt	Naïve	Opt	Naïve	Opt	Naïve	Opt	Naïve	Opt	Naïve	Opt
2var_unstable	6.92	0.78	0.21	0	0.46	0.54	0	0	0.51	0.57	0	0
Bcr-Abl	67.76	9.32	28.92	1.46	196.68	9.49	142.41	1.31	281.27	10.29	108.14	1.85
Bcr-AblNoFeedbacks	66.52	6.77	29.58	0.71	201.59	6.71	101.69	0.56	307.60	6.60	219.72	0.62
BooleanLoop	0.49	0.51	0	0	0.48	0.57	0.01	0	0.53	0.59	0.01	0.01
NoLoopFound	0.78	0.74	0.06	0.01	1.14	0.93	0.09	0.03	1.45	1.04	0.10	0.06
Skin1D_TF_0	136.21	140.78	122.85	127.47	218.52	80.33	191.06	55.23	127.28	96.49	86.05	60.06
Skin1D_TF_1	167.32	173.03	154.00	159.55	698.47	445.32	670.77	419.24	883.35	572.03	842.06	536.04
Skin1D	90.92	68.82	77.63	54.54	45.67	23.21	17.55	8.77	133.72	23.46	92.36	8.13
Skin2D_3cells_2layers_0	567.31	640.71	545.49	618.44	238.28	205.15	192.28	162.14	164.79	218.77	93.45	153.11
Skin2D_3cells_2layers_1	910.08	553.27	891.70	535.02	82.04	117.48	44.70	82.79	122.77	219.04	64.96	167.65
Skin2D_3cells_2layers_2	315.20	169.92	293.45	151.64	121.12	36.58	74.49	18.74	188.78	39.36	114.81	20.15
Skin2D_5X2_TF	511.31	223.93	459.38	182.65	1466.90	391.96	1378.80	353.06	1275.30	73.77	1135.25	35.83
Skin2D_5X2	343.96	85.64	300.03	56.71	721.58	57.20	630.92	28.46	965.24	48.26	828.12	16.83
SmallTestCase	0.53	0.54	0.01	0	0.54	0.73	0.01	0	0.60	0.54	0.01	0
SSkin1D_0	70.71	69.00	63.71	61.93	21.35	20.71	5.87	5.93	33.07	32.74	12.52	12.34
SSkin1D_TF_1	9.77	10.05	2.88	2.93	22.85	26.02	8.23	9.04	35.61	35.16	15.12	14.96
SSkin1D	145.28	146.74	138.61	139.76	32.00	33.38	18.29	18.51	33.89	33.80	13.57	13.49
SSkin2D_3cells_2layers	301.33	158.62	286.80	148.08	63.46	50.12	35.44	36.14	86.26	32.41	44.30	14.91
VerySmallTestCase	0.37	0.42	0	0	0.39	0.43	0.01	0	0.40	0.43	0.01	0
VPC_lin15ko	8.31	6.81	3.35	0.32	14.87	6.74	5.13	0.26	21.99	6.76	7.42	0.20
VPC_Non_stabilizing	3.43	3.40	0.85	0.26	6.02	3.95	1.23	0.29	9.35	4.87	2.10	0.62
VPC_stabilizing	3.31	4.79	0.74	0.14	5.84	4.79	0.99	0.18	9.10	4.67	1.92	0.14

Table 2. Searching for loops (10, 20, 30).

When considering the path search, on many of the smaller models the new technique does not offer a significant advantage. However, on larger models, and

Length of loop	40				50			
	Global Time (Seconds)		Sat Time (Seconds)		Global Time (Seconds)		Sat Time (Seconds)	
Model name	Naïve	Opt	Naïve	Opt	Naïve	Opt	Naïve	Opt
2var_unstable	0.54	0.60	0.01	0	1.05	0.64	0.01	0.01
Bcr-Abl	667.22	11.54	552.90	2.74	1019.68	11.94	869.56	2.76
Bcr-AblNoFeedbacks	574.61	6.79	316.07	0.64	857.17	6.90	719.21	0.69
BooleanLoop	0.54	0.60	0.01	0.01	0.59	0.66	0.01	0.01
NoLoopFound	1.90	1.15	0.23	0.04	2.23	1.34	0.22	0.05
Skin1D_TF_0	126.13	153.85	68.31	104.11	224.38	247.93	149.55	182.58
Skin1D_TF_1	108.84	160.72	52.01	112.33	167.86	290.97	91.13	228.46
Skin1D	122.73	29.39	64.99	12.84	259.75	34.04	182.50	16.09
Skin2D_3cells_2layers_0	391.08	325.43	293.83	237.89	470.89	663.87	341.24	545.49
Skin2D_3cells_2layers_1	196.99	271.98	118.22	202.01	476.94	557.09	366.61	464.88
Skin2D_3cells_2layers_2	413.13	44.06	314.95	23.75	445.78	47.51	308.71	25.18
Skin2D_5X2_TF	3067.08	93.15	2649.01	48.12	5135.87	82.38	3956.13	34.25
Skin2D_5X2	2403.53	47.69	2149.43	14.87	4025.83	56.86	3254.90	18.18
SmallTestCase	0.96	0.57	0.02	0	0.77	0.58	0.02	0
SSkin1D_0	44.81	42.03	13.52	13.37	58.09	57.45	22.64	21.91
SSkin1D_TF_1	43.97	46.26	15.88	16.13	60.46	60.52	22.77	24.35
SSkin1D	41.13	41.49	12.48	12.59	60.77	61.34	22.82	22.87
SSkin2D_3cells_2layers	117.64	42.86	50.82	20.36	157.07	51.19	80.54	22.95
VerySmallTestCase	0.48	0.44	0	0	0.81	0.67	0.01	0
VPC_lin15ko	27.04	6.94	7.34	0.20	45.70	7.14	20.78	0.23
VPC_Non_stabilizing	14.58	5.64	2.36	0.65	16.21	6.50	4.10	1.07
VPC_stabilizing	13.13	6.66	3.44	0.12	17.07	4.99	5.07	0.20

Table 3. Searching for loops (40, 50).

in particular the two dimensional skin model (Skin2D_5X2 from [21]) and the Leukemia model (Bcr_Abl) the new technique is an order of magnitude faster. Furthermore, when increasing the length of the path it scales a lot better than the naïve approach. When model checking is considered, the combination of the decreasing reachability sets accelerates model checking considerably. While the naïve search increases considerably to the order of tens of minutes, the optimized search remains within the order of 10 seconds, which affords a "real-time" response to users.

	Global Time (Sec)		Sat Time (Sec)		Ratio		
Model name	Naïve	Opt	Naïve	Opt	Global	Sat	
Bcr-Abl1	69.30	9.04	26.67	0.90	7.66	29.61	sat
Bcr-Abl1	188.13	12.21	87.70	1.42	15.40	61.47	sat
Bcr-Abl1	380.24	13.12	292.21	2.01	28.96	145.02	sat
Bcr-Abl1	648.02	12.37	349.70	2.30	52.38	151.87	sat
Bcr-Abl1	1005.37	11.52	588.34	2.17	87.19	270.93	sat
Bcr-Abl2	47.04	10.97	9.94	0.72	4.28	13.76	Unsat
Bcr-Abl2	136.48	8.62	41.04	0.75	15.82	54.66	Unsat
Bcr-Abl2	285.28	11.28	112.35	0.77	25.28	144.58	Unsat
Bcr-Abl2	561.65	9.29	443.91	0.80	60.41	553.83	Unsat
Bcr-Abl2	781.64	12.03	408.55	0.87	64.96	465.55	Unsat
Bcr-Abl3	48.64	8.47	9.54	0.83	5.74	11.45	Unsat
Bcr-Abl3	133.83	9.10	38.68	1.11	14.69	34.81	Unsat
Bcr-Abl3	283.73	9.45	106.61	1.16	30.01	91.28	Unsat
Bcr-Abl3	596.50	9.50	466.01	1.18	62.78	394.48	Unsat
Bcr-Abl3	853.53	10.05	480.77	1.36	84.89	351.99	Unsat
Bcr-Abl4	75.27	9.19	44.50	0.80	8.18	55.31	sat
Bcr-Abl4	202.06	9.95	143.49	1.53	20.30	93.50	sat
Bcr-Abl4	296.02	11.35	116.24	2.54	26.07	45.74	sat
Bcr-Abl4	740.39	11.00	621.41	1.96	67.24	316.19	sat
Bcr-Abl4	975.97	10.42	823.53	1.10	93.63	747.14	sat
Bcr-AblNoFeedbacks1	42.98	6.25	7.94	0.40	6.87	19.51	Unsat
Bcr-AblNoFeedbacks1	163.33	8.18	95.43	0.77	19.95	123.90	Unsat
Bcr-AblNoFeedbacks1	302.17	6.41	122.25	0.46	47.07	260.90	Unsat
Bcr-AblNoFeedbacks1	493.28	6.41	314.24	0.45	76.92	686.28	Unsat
Bcr-AblNoFeedbacks1	809.97	6.45	680.70	0.46	125.51	1461.69	Unsat
Bcr-AblNoFeedbacks2	44.88	6.39	6.59	0.40	7.01	16.27	Unsat
Bcr-AblNoFeedbacks2	117.96	6.34	20.98	0.39	18.58	53.61	Unsat
Bcr-AblNoFeedbacks2	312.73	7.59	231.87	0.46	41.18	500.00	Unsat
Bcr-AblNoFeedbacks2	527.40	6.31	423.61	0.39	83.46	1084.74	Unsat
Bcr-AblNoFeedbacks2	751.45	6.83	362.09	0.44	109.87	806.35	Unsat
Bcr-AblNoFeedbacks3	60.99	6.95	20.45	0.64	8.77	31.64	sat
Bcr-AblNoFeedbacks3	204.66	7.06	144.58	0.61	28.97	233.95	sat
Bcr-AblNoFeedbacks3	356.33	8.81	267.48	0.49	40.42	539.32	sat
Bcr-AblNoFeedbacks3	Time out	7.06	Time out	0.42	N/A	N/A	sat
VPC_non_stabilizing1	30.14	10.83	4.83	0.69	2.78	6.93	Unsat
VPC_non_stabilizing2	17.42	9.85	3.59	1.11	1.76	3.24	sat
VPC_non_stabilizing3	52.01	11.91	26.69	1.48	4.36	17.93	Unsat
VPC_non_stabilizing4	19.53	8.31	7.08	0.60	2.34	11.77	Unsat
VPC_stabilizing1	3.75	5.11	0.31	0.07	0.73	3.99	Unsat
VPC_stabilizing2	5.53	5.32	0.86	0.11	1.04	7.41	sat

Table 4. Model checking results.

6 Conclusions and Future Work

We have presented a new technique for model checking Qualitative Networks. Our technique utilizes the unique structure of Qualitative Networks to construct

"decreasing reachability sets". These sets form part of a compact representation of paths in the QN and lead to significant acceleration in an implementation of bounded model checking.

Our main aim is to use these methods in order to gain new biological insights. These aspects will be reported elsewhere; the work presented here is about the various techniques we have developed and evaluated. The tool has been made available to our biologist collaborators. We are working on adding LTL model checking as one of the supported analysis techniques in our tool BMA [2]. We find the experimental results very encouraging especially given the iterative development methodology biologists have been using when employing our tools [2]. As mentioned, our users "try out" several options and refine them according to results of simulation and verification. In this iterative process it is most important to be able to give fast answers to queries of the user. We hope that with the speed ups afforded by this new technique model checking could be incorporated into the workflow of using our tools.

We note that the encoding of target functions for highly connected variables is not efficient. Enumerating all possible options entails going over a large number of options. For example, a variable that has 8 inputs (or more) ranging over $\{0, 1, 2\}$ requires to enumerate $3^8 = 6561$ options for the transition table of one variable (not to mention adding 6561 Boolean variables per variable per state in the path – see [6]). We are currently working on alternative approaches to analyze the target function of a single variable to enable better encoding of it. We note that this is a problem also of stability analysis [8].

We intend to remove the initial states from "normal" transition systems and evaluate whether decreasing reachability sets could prove useful for other types of systems as well.

References

1. G. Batt, D. Ropers, H. de Jong, J. Geiselmann, R. Mateescu, M. Page, and 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:i19–i28, 2005.
2. D. Benque, S. Bourton, C. Cockerton, B. Cook, J. Fisher, S. Ishtiaq, N. Piterman, A. Taylor, and M. Vardi. BMA: Visual tool for modeling and analyzing biological networks. In *24th International Conference on Computer Aided Verification*, volume 7358 of *Lecture Notes in Computer Science*, pages 686–692. Springer, 2012.
3. A. Beyer, P. Thomason, X. Li, J. Scott, and J. Fisher. Mechanistic insights into metabolic disturbance during type-2 diabetes and obesity using qualitative networks. *T. Comp. Sys. Biology*, 12:146–162, 2010.
4. N. Bonzanni, E. Krepska, K. A. Feenstra, W. Fokkink, T. Kielmann, H. Bal, and J. Heringa. Executing multicellular differentiation: Quantitative predictive modelling of *C.elegans* vulval development. *Bioinformatics*, 25(16):2049–2056, 2009.
5. N. Chabrier-Rivier, C. M., V. Danos, F. Fages, and V. Schächter. Modeling and querying biomolecular interaction networks. *Th. Comp. Sci.*, 325(1):25–44, 2004.
6. K. Claessen, J. Fisher, S. Ishtiaq, N. Piterman, and W. Qinsi. Model-checking signal transduction networks through decreasing reachbility sets. Technical Report MSR-TR-2013-30, Microsoft Research, 2013.

7. E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.
8. B. Cook, J. Fisher, E. Krepska, and N. Piterman. Proving stabilization of biological systems. In *Verification, Model Checking, and Abstract Interpretation*, volume 6538 of *Lecture Notes in Computer Science*, pages 134–149. Springer, 2011.
9. D. Dill, M. Knapp, P. Gage, C. Talcott, K. Laderoute, and P. Lincoln. The pathalyzer: a tool for analysis of signal transduction pathways. In *1st Annual Recomb Satellite Workshop on Systems Biology*, 2005.
10. C. Espinosa-Sotoa, P. Padilla-Longoriab, and E. Alvarez-Buyllaa. A gene regulatory network model for cell-fate determination during arabidopsis thaliana flower development that is robust and recovers experimental gene expression profiles. *The Plant Cell*, 16(11):2923–39, 2004.
11. J. Fisher and T. Henzinger. Executable cell biology. *Nature Biotechnology*, 25(11):1239–49, 2007.
12. J. Fisher, N. Piterman, A. Hajnal, and T. Henzinger. Predictive modeling of signaling crosstalk during *c. elegans* vulval development. *PLoS Computational Biology*, 3(5):e92, 2007.
13. J. Heath. The equivalence between biology and computation. In *Computational Methods in Systems Biology*, volume 5688 of *Lecture Notes in Computer Science*, pages 18–25. Springer, 2009.
14. J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. In *Computational Methods in Systems Biology*, volume 4210 of *Lecture Notes in Computer Science*, pages 32–47. Springer, 2006.
15. F. Li, T. Long, Y. Lu, Q. Ouyand, and C. Tang. The yeast cell-cycle network is robustly designed. *Proc Natl Acad Sci U S A*, 101(14):4781–6, 2004.
16. A. Naldi, D. Thieffry, and C. Chaouiya. Decision diagrams for the representation and analysis of logical models of genetic networks. In *Computational Methods in Systems Biology*, volume 4695 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2007.
17. A. Pnueli and A. Zaks. On the merits of temporal testers. In *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *Lecture Notes in Computer Science*, pages 172–195. Springer, 2008.
18. S. Ray and R. Brayton. Proving stabilization using liveness to safety conversion. In *20th International Workshop on Logic and Synthesis*, 2011.
19. A. Samal and S. Jain. The regulatory network of e. coli metabolism as a boolean dynamical system exhibits both homeostasis and flexibility of response. *BMC Systems Biology*, 2(1):21, 2008.
20. L. Sanchez and D. Thieffry. Segmenting the fly embryo: a logical analysis fo the *pair-rule* cross-regulatory module. *Journal of Theoretical Biology*, 244:517–537, 2003.
21. M. Schaub, T. Henzinger, and J. Fisher. Qualitative networks: A symbolic approach to analyze biological signaling networks. *BMC Systems Biology*, 1(4), 2007.
22. I. Shmulevitch, R. Dougherty, S. Kim, and W. Zhang. Probabilistic boolean networks: A rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18(2):261–74, 2002.
23. R. Thomas, D. Thieffry, and M. Kaufman. Dynamical behaviour of biological regulatory networks—i. biological role of feedback loops and practical use of the concept of the loop-characteristic state. *Bull. of Math. Bio.*, 55(2):247–276, 1995.