Model Checking in Biology

Jasmin Fisher and Nir Piterman

Abstract Model checking is a technique to check whether programs and designs satisfy properties expressed in temporal logic. Such properties characterize sequences of events. In recent years, model checking has become a familiar tool in software and hardware industries. One of the main strengths of model checking is its ability to supply counter examples: in case that the property is not satisfied by the model we get an execution exhibiting this failure. Counter examples are fundamental in understanding, localizing, and eventually fixing, faults. This, together with the relative ease of use of model checking, led to its adoption.

The success of model checking prompted system biologists to harness it to their needs. In this domain, the main usage is to have a model representing a certain biological phenomenon and to use model checking for one of two things. Either prove that the model satisfies a set of properties, i.e., reproduces a set of biological behaviors. Or to use model checking to extract interesting behaviors of the model by looking for a counter example to the property saying that this interesting behavior does not happen.

In this chapter we present the technique of model checking and survey its usage in systems biology. We take quite a liberal interpretation of what is model checking and consider also cases where the techniques underlying model checking are used for similar purposes in systems biology.

Jasmin Fisher
Microsoft Research Cambridge, Cambridge, UK, e-mail: jasmin.fisher@microsoft.com

Nir Piterman
University of Leicester, Leicester, UK e-mail: nir.piterman@leicester.ac.uk

1 Introduction

Biological systems are extremely complex reactive systems. They operate as highly concurrent programs with millions of entities running in parallel and communicating with each other under various environmental conditions. Understanding how living systems operate in such harmony and precision, and how this harmony is being disrupted in disease states, are key questions in biological and medical research. Due to their enormous complexity, the comprehension and analysis of living systems is a major challenge. Over the last decade various efforts to tackle this problem of enormous complexity concentrate on a new approach called *Executable Biology* focused on the construction and analysis of executable models describing biological phenomena (for a review see [19]).

Over the years, these efforts have demonstrated successfully how the use of formal methods can be beneficial for gaining new biological insights and even directing new experimental avenues. At the core of these models lies their ability to be analysed by *model checking* [15]. In the context of biological models, model checking can be used in two ways:

1. **Testing and comparing hypotheses.** Computational models represent hypotheses about molecular and cellular mechanisms that result in experimental data. Executions of these models can be used to check if a possible outcome of these mechanisms conforms to the data. Due to the nondeterministic nature of these models, each repeated execution may yield a different possible outcome. Therefore it is impossible to check by executing these models if all possible outcomes conform to the data. This, however, can be done by model checking, as model checking systematically analyzes all of the infinitely many possible outcomes of a computational model without executing them one by one. If model checking tells us (a) that all possible outcomes of the computational model agree with the experimental data, and (b) that all experimental outcomes can be reproduced by the model, then the model represents a mechanism that explains the experimental data. If (b) is violated, then the hypothesis that the computational model captures a mechanism for explaining the data is found to be wrong. In this case, either the model must be enriched as to produce the additional outcomes that are present in the data, or completely revised. If (a) is violated, then the situation is more interesting. In this case, the mechanistic hypothesis represented by the model may be wrong, and one may attempt to restrict the model as to not produce outcomes that are not supported by the data. Alternatively, the experimental data may be incomplete and not exhibit some possible observations that would show up if more data were collected. Thus, in case (a), model checking can offer suggestions for additional, targeted experiments that would either confirm or invalidate the mechanistic hypothesis represented by the computational model.
2. **Querying the behaviour of mechanistic models.** Once a model has been tested and compared against hypotheses, it can also be queried by searching for interesting executions. By stating that a certain property holds for all executions,

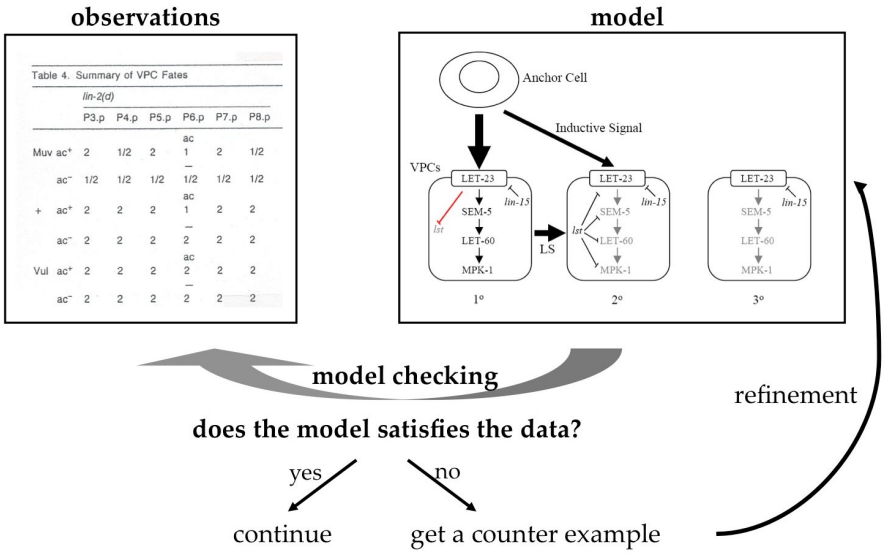


Fig. 1 Methodology of using model checking. One possible methodology for using model checking is by comparing mechanistic models to specifications. A formal model that represents a hypothetical understanding of the system under study is constructed (model). Results of experiments are formalized in the form of specifications (observations). Model checking is used to ensure that the model reproduces the experimental observations. Mismatch with experimental observations suggests that the model is lacking and should be refined by additional information. Match with experimental observations could lead to further querying and testing of the model to suggest further experimental studies.

or by stating that a certain property does not hold for all executions, we can either validate or falsify specific predictions about the behaviour of the model. By phrasing queries such as which molecular events may lead to specific cell behaviour, we can also determine what part of the execution allows this kind of events.

In this chapter we give an introduction to model checking and the techniques underlying it. This is in the hope that practitioners will understand better the techniques and what can be done with them. In particular, recent algorithmic progress shows that with good understanding of the underlying techniques further types of analysis can be accomplished using model checking techniques. We also give examples of instances where model checking was used in biological modeling to demonstrate a flavor of the results that can be achieved by using model checking. In particular, in some cases, usage of model checking led to new biological insights, shedding new light on signalling crosstalk.

2 What is Model Checking

In this section we introduce the mathematical concepts underlying model checking. Model checking is a technique that checks whether all the computations of a system satisfy a property. In order to be able to answer this question we have to create a formal model of the system and formally state the property. In this section we introduce the underlying formal concepts and the formal definition of model checking. We start by introducing transition systems, which will be used to represent the possible computations of a model. We then proceed to explain temporal logic and conclude with an explanation of what is model checking.

2.1 Transition Systems

The concept of a transition system is a fundamental concept of computation. Here, we are going to refer to a transition system representing some machine. However, in the context of this chapter, the machine is usually a model of some biological machinery. A transition system has states, which represent snapshots of the status of the machine, and transitions, which represent the possible change of status of the machine. For the sake of model checking additional annotations are required and these are usually put on the states in the form of propositions, which are basic facts about the world of the machine that could be either true or false in a given state.

A transition system is $\mathscr{T} = (S, T, S_0, \mathscr{P}, L)$ with the following components:

- $\mathscr{T}$ is the name of the transition system.
- S is the set of states of $\mathscr{T}$, every state $s \in S$ represents a possible snapshot of the machine. The set S_0 is a subset of S of initial states indicating in which states can the machine start an execution. For the purpose of presentation of model checking we are going to concentrate on transition systems with a finite number of states.
- T is the set of transitions formally represented as a set of pairs of states, i.e., $T \subseteq S \times S$. For states s and t, having $(s,t) \in T$ means that the machine can change from state s to state t.
- $\mathscr{P}$ is the set of facts that are observable in a state of $\mathscr{T}$. The labeling function L associates with every state the facts that are true in that state. Formally, $L : S \to 2^{\mathscr{P}}$ is a function that associates with every state the set of propositions that are true in it. By elimination, all other propositions are false in that state.

Given this notion of a transition system we are now ready to define what are computations of a transition system. Intuitively, a computation is a sequence of states that starts in an initial state and proceeds by taking permissible transitions. More formally, a path π is a sequence $s_0, s_1, \ldots$ such that all transitions are in T, i.e., for every $i \geq 0$ we have $(s_i, s_{i+1}) \in T$. A path $\pi = s_0, s_1, \ldots$ is a computation if in addition s_0 is initial, i.e., $s_0 \in S_0$. We note that paths and computations are infinite. Thus,

we assume that all states have at least one outgoing transition. This is a usual assumption in model checking as it makes it simpler to avoid many technicalities. One can reintroduce the concept of finite runs by adding a special *finish* state with a self loop and considering runs that end in an infinite sequence of visits to finish as finite. A computation or a path induces a sequence of labels representing the change in the status of the different facts about the machine that interest us. Instead of looking on the sequence $s_0, s_1, \ldots$, we could look on the sequence of labels $L(s_0), L(s_2), \ldots$, where $L(s_i)$ is the set of facts that are true in state s_i (and the rest are false). We call the sequence of labels a *run* and denote it by $r = L(s_0, s_1, \ldots)$. When the run is induced by a computation we say that it is *initialized*.

2.2 Temporal Logic

We now turn to the definition of a languages to give "interesting" properties about the transition system. Essentially, we would like to be able to describe qualities of the transition system or its computations. The ultimate goal (to be visited in the next subsection) is to check whether a transition system satisfies a given property. That is, whether the machine described by a transition system has this quality or not. We distinguish between two languages to describe properties of systems. The first, called *linear temporal logic* views a system as the sum of all of its possible computations. The second, called *computation tree logic* views the entire transition system as the embedding of the machine. Here we choose to expose both as each has its own advantages and both have been used in the context of biological modeling.

We start with *linear temporal logic* (LTL for short), which was introduced by Pnueli in the late 70's [29]. As it's name suggests it takes the first approach of viewing the system as the sum of all its computations. An LTL formula is going to use the basic facts about states (i.e., labels) and combine them in ways that say things about sequences. Then, we define when an LTL formula is satisfied by a single run. Finally, an LTL formula will be satisfied by the transition system if all possible runs of the transition system satisfy it.

As mentioned an LTL formula can use one of the basic facts in P. It can use one of the following operators.

- Next, denoted $\mathscr{X}$, a unary operator saying that its operand is true in the next state.
- Until, denoted $\mathscr{U}$, a binary operator saying that its first operand must hold until its second operand holds.
- Always (or globally), denoted $\mathscr{G}$, a unary operator saying that its operand is true in all future states (including current).
- Eventually (or finally), denoted $\mathscr{F}$, a unary operator saying that its operand is true in some future (or current) state.
- In addition, LTL uses the usual Boolean operators not, denoted $\neg$, conjunction, denoted $\wedge$, disjunction, denoted $\vee$, and implication, denoted $\rightarrow$.

Thus, an LTL formula is constructed hierarchically from simpler formulas. For example, the formula $\mathscr{G}(p \rightarrow \mathscr{X} q)$ uses the basic facts p and q and says that in all states of the computation if p holds in a state then q must hold in the next state. Similarly, $\mathscr{G}(p \rightarrow p \mathscr{U} q)$ says that whenever p holds, it must hold continuously until a later time when q holds.

We now make the intuition regarding when a formula holds in a computation formal. For that, we start with a definition of when a formula holds in a specific run in a specific location. The definition builds truth values according to the hierarchical structure of the formula. That is, basic facts (propositions) can be deduced from the label. Then, the truth of more complicated formulas is constructed from that of simpler formulas. Consider a run $r = l_1, l_2, \ldots$ over $\mathscr{P}$. That is, every l_i is a set of basic facts that are true at time i (and all facts in $\mathscr{P} - l_i$ are false at time i). The following list defines when an LTL formula formula φ is satisfied in r at time i, denoted $r,i \models \varphi$, and when it is not satisfied, denoted $r,i \not\models \varphi$.

- If φ is a proposition, then $r,i \models \varphi$ if $\varphi \in l_i$ and $r,i \not\models \varphi$ if $\varphi \notin l_i$.
- If φ is $\neg\psi$ then $r,i \models \varphi$ if $r,i \not\models \psi$ and $r,i \not\models \varphi$ if $r,i \models \psi$.
- If φ is $\psi_1 \wedge \psi_2$ then $r,i \models \varphi$ if $r,i \models \psi_1$ and $r,i \models \psi_2$ and $r,i \not\models \varphi$ if either $r,i \not\models \psi_1$ or $r,i \not\models \psi_2$.
- If φ is $\psi_1 \vee \psi_2$ then $r,i \models \varphi$ if either $r,i \models \psi_1$ or $r,i \models \psi_2$ and $r,i \not\models \varphi$ if $r,i \not\models \psi_1$ and $r,i \not\models \psi_2$.
- If φ is $\psi_1 \rightarrow \psi_2$ then $r,i \models \varphi$ if either $r,i \not\models \psi_1$ or $r,i \models \psi_2$ and $r,i \not\models \varphi$ if $r,i \models \psi_1$ and $r,i \not\models \psi_2$.
- If φ is $\mathscr{X}\psi$ then $r,i \models \varphi$ if $r,i+1 \models \psi$ and $r,i \not\models \varphi$ if $r,i+1 \not\models \psi$.
- If φ is $\psi_1 \mathscr{U} \psi_2$ then $r,i \models \varphi$ if there is a $k \geq i$ such that $r,k \models \psi_2$ and for every $i \leq j < k$ we have $r,j \models \psi_1$ and $r,i \not\models \varphi$ if for every $k \geq i$ such that $r,k \models \psi_2$ there is $i \leq j < k$ such that $r,j \not\models \psi_1 1$.
- If φ is $\mathscr{G}\psi$ then $r,i \models \varphi$ if for every $j \geq i$ we have $r,j \models \psi$ and $r,i \not\models \varphi$ if for some $j \geq i$ we have $r,j \not\models \psi$.
- If φ is $\mathscr{F}\psi$ then $r,i \models \varphi$ if for some $j \geq i$ we have $r,j \models \psi$ and $r,i \not\models \varphi$ if for all $j \geq i$ we have $r,j \not\models \psi$.

Finally, a system $\mathscr{T}$ satisfies an LTL formula φ, denoted $\mathscr{T} \models \varphi$, if for every run of the system we have $r,0 \models \varphi$. Otherwise, the system does not satisfy the formula, denoted $\mathscr{T} \not\models \varphi$.

We turn now to *computation tree logic* (CTL for short), which was introduced by Clarke and Emerson [14]. The name of the logic derives from viewing the transition system as producing a single *computation tree*, which we do not explain here. Unlike LTL, CTL is going to take an integrative view of the system. A formula is going to be either true or false for a state of the system. Like LTL, CTL is going to use the basic facts about states and combine them to properties about the system. It then combines information about the system by considering the paths that start in states and state properties of some or all these paths. A CTL formula is satisfied by the system if all initial states of the system satisfy it.

As mentioned CTL combines information about paths and about states. A CTL formula can use one of the basic facts in $\mathscr{P}$. Such basic facts are state formulas. It can use one of the following operators.

- Boolean operators $\wedge$, $\vee$, $\neg$, $\rightarrow$ can be applied to formulas as in LTL.
- The temporal operators next, until, always, and eventually are combined with path quantification E and A. Thus, CTL includes the unary operators $E\mathscr{X}$, $A\mathscr{X}$, $E\mathscr{G}$, $A\mathscr{G}$, $E\mathscr{F}$, and $A\mathscr{F}$ that can be applied to simpler formulas. The binary operators $E\,\mathscr{U}$ and $A\,\mathscr{U}$ combine two formulas. The E quantifier says "there exists a path" and the A quantifier says "for all paths". The meaning of the temporal part remains the same as in LTL. Thus, a formula like $A\mathscr{X}\,\psi$ all next states satisfy the property ψ. A formula like $E(\psi_1\,\mathscr{U}\,\psi_2)$ says that from a given state there is a path satisfying the property $\psi_1\,\mathscr{U}\,\psi_2$.

For example, the formula $A\mathscr{G}(p \rightarrow E\mathscr{X}\,q)$ says that every state where the fact p is true that is reachable from an initial state must have a successor where the fact q holds. Similarly, $A\mathscr{G}A\mathscr{F}\,p$ means that from every possible reachable state every way to continue the will eventually reach a state where the fact p is true.

We now make this intuition formal. As before, the definition builds truth values according to the hierarchical construction of the formula. Every state formula defines a set of states in which it is true. Path formulas are defined just like for LTL except that for the set of paths that start at a given state. Similar to the case of LTL, we denote by $\mathscr{T},s \models \varphi$ if a formula is satisfied in state s of $\mathscr{T}$ and $\mathscr{T},s \not\models \varphi$ otherwise.

- If φ is a proposition, then $\mathscr{T},s \models \varphi$ if $\varphi \in L(s)$ and $\mathscr{T},s \not\models \varphi$ otherwise.
- If φ is $\neg\psi$ then $\mathscr{T},s \models \varphi$ if $\mathscr{T},s \not\models \psi$ and $\mathscr{T},s \not\models \varphi$ if $\mathscr{T},s \models \varphi$. The definitions for formulas of the form $\psi_1 \wedge \psi_2$ and $\psi_1 \vee \psi_2$ is similar.
- If φ is $E\mathscr{X}\,\psi$ then $\mathscr{T},s \models \varphi$ if there is a successor t of s (i.e., $(s,t) \in T$) such that $\mathscr{T},t \models \psi$ and $\mathscr{T},s \not\models \varphi$ if for all successors t of s we have $\mathscr{T},t \not\models \psi$.
- If φ is $E(\psi_1\,\mathscr{U}\,\psi_2)$ then $\mathscr{T},s \models \varphi$ if there is some path $\pi = s_0,s_1,\ldots$ starting in s such that for some i we have $\mathscr{T},s_i \models \psi_2$ and for every $0 \leq j < i$ we have $\mathscr{T},s_j \models \psi_2$ and $\mathscr{T},s \not\models \varphi$ if for every path $\pi = s_0,s_1,\ldots$ starting in s and for every $i \geq 0$ if $\mathscr{T},s_i \models \psi_2$ then there is $0 \leq j < i$ such that $\mathscr{T},s_j \not\models \psi_1$.
- The definitions of other temporal connectives can be completed in a similar way by combining the path quantification with the definition from LTL.

We say that the transition system $\mathscr{T}$ satisfies a CTL state formula φ, denoted $\mathscr{T} \models \varphi$ if for every initial state s_0 we have $\mathscr{T},s_0 \models \varphi$. Otherwise, $\mathscr{T}$ does not satisfy φ, denoted $\mathscr{T} \not\models \varphi$.

2.3 Model Checking

Model checking is the process of checking whether a formula holds for a specific transition system. That is, given a transition system $\mathscr{T}$ and a formula φ (either

in CTL or LTL), to decide if the formula is satisfied by the system, i.e., whether $\mathscr{T} \models \varphi$. In the case of LTL if the answer is negative we would like to get a run of the system that exhibits the failure of the property. That is, produce an initialized run r such that $r,0 \not\models \varphi$.

The algorithmic approach towards model checking reduces this problem to a graph exploration problem. Essentially, we look on the transition system as a graph (sometimes with additional information) and deduce from analysis of this graph the correctness of the property. For both LTL and CTL we augment the transition system with auxiliary information. In the case of CTL the auxiliary information is by adding additional labels that tell us the truth values of simpler state formulas. In the case of LTL the algorithm is more complicated and the auxiliary information requires the addition of extra states and condition. Here we give a short exposition of the addition of labels that supports CTL model checking. A detailed exposition of LTL model checking is available for example in [2].

The algorithm for model checking CTL involves adding additional labels to the states of the transition system. Starting with a transition system $\mathscr{T} = (S, T, S_0, \mathscr{P}, L)$ and a CTL property φ we show how to add labels to $\mathscr{T}$. We assume that $\mathscr{T}$ already includes labels that tell us for every subformula of φ in which states it holds. Then, if φ is a Boolean combination of simpler operands then it is simple to deduce the truth of φ from the labels on states of $\mathscr{T}$. We simply add the label of φ to the states of $\mathscr{T}$. If φ is $A\mathscr{X}\psi$, then by assumption we have already included labels telling us when ψ is true in states of $\mathscr{T}$. It now suffices to iterate over all states of $\mathscr{T}$. If a state has a successor not marked by ψ then we do not change the label of such a state. If a state has all its successors marked by ψ then we add φ to the label of that state. The treatment of $E\mathscr{X}\psi$ is similar.

We now turn to the treatment of $E(\psi_1 \mathscr{U} \psi_2)$. As before we assume that the labeling of states includes the value of ψ_1 and ψ_2. We start by marking with $E(\psi_1 \mathscr{U} \psi_2)$ all states that are marked by ψ_2. We then remove from our consideration states that are not labeled by ψ_1. Indeed, such states cannot be labeled by $E(\psi_1 \mathscr{U} \psi_2)$. We then repeatedly go over all states and add the marking $E(\psi_1 \mathscr{U} \psi_2)$ to every state that has some successor marked by $E(\psi_1 \mathscr{U} \psi_2)$ (considering only states marked by ψ_1). Once no such states can be found this stage terminates. The description above sounds inefficient requiring repeated iteration over all the states. However, it can be implemented efficiently by iterating at most once over all possible transitions.

The treatment of $A(\psi_1 \mathscr{U} \psi_2)$ is very similar. As before, we mark with $A(\psi_1 \mathscr{U} \psi_2)$ all states that are marked by ψ_2 and remove from consideration all states not marked by ψ_1. Then, the repeated iteration adds the label $A(\psi_1 \mathscr{U} \psi_2)$ to states that have *all* their successors marked by $A(\psi_1 \mathscr{U} \psi_2)$.

The treatment of $A\mathscr{F}\psi$ and $E\mathscr{F}\psi$ is a special case of the two cases above, where we note that we can ignore the role of ψ_1. The treatment of $E\mathscr{G}\psi$ and $A\mathscr{G}\psi$ is by considering the equivalences $E\mathscr{G}\psi \equiv \neg A\mathscr{F}\neg\psi$ and $A\mathscr{G}\psi \equiv \neg E\mathscr{F}\neg\psi$. We treat states not labeled by ψ as labeled by $\neg\psi$.

The algorithm for model checking LTL formulas is similar to the above in the sense that we create an algorithm that searches for paths in the graph obtained from the transition system and an additional structure obtained from the LTL formula.

3 Usage of Model Checking in Biology

In this section we survey a few modeling efforts that used model checking for analysis. We choose to highlight results that use the techniques as described above and that, in our opinion, show how model checking can be beneficial for the analysis of biological models. In some cases, these studies also led to the discovery of new biological insights. Obviously, in such a short book chapter it is impossible to survey all studies using model checking and we apologize to colleagues whose work we could not review due to lack of space.

3.1 Insights into temporal aspects of signalling crosstalk during cell fate determination

Describing mechanistic models in biology in a dynamic and executable language offers great advantages for representing time and parallelism, which are important features of biological behaviours. Model checking can be used to ensure the consistency of computational models with biological data on which they are based. Fisher and colleagues [20] have previously developed a dynamic computational model describing the mechanistic understanding of cell fate determination during the earthworm *C. elegans* vulval development, which provides an important paradigm for studying animal development. Model checking analysis of this model has provided new insights into the temporal aspects of the cell fate patterning process and predicted new modes of interaction between the signalling pathways involved. These biological insights, which were also validated experimentally, further substantiate the usefulness of dynamic computational models to investigate complex biological behaviours.

Fisher et al. [20] have used model checking for two purposes. First, to ascertain that their mechanistic model reproduces the biological behaviour observed in different mutant backgrounds. For that, they have formalized the experimental results described in a set of papers and verified that all possible executions satisfy these behaviours. That is, regardless of the order of interactions from a given set of initial conditions, different executions always reproduced the experimental observations. Second, they also used model checking to query the behaviour of the model. By phrasing queries such as which mutations may lead to a stable or an unstable fate pattern, they analyzed the behaviour of the model. Once an unstable mutation was found, they determined what part of the execution allows this kind of mutations by disallowing different behavioural features of the model and checking when the instability disappears (see Fig. 2).

By using model checking to compare the executable model with existing experimental data, Fisher et al. predicted novel interactions in the signalling network governing vulval fate specification, in addition to predicting the outcome of perturbations that are difficult to test experimentally. These insights led to suggest a

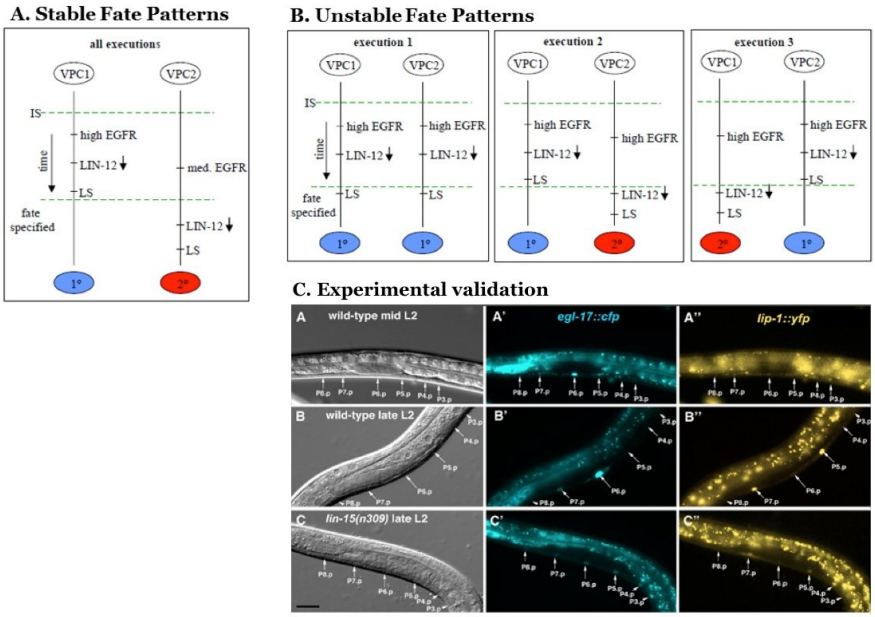


Fig. 2 Order of events in stable and unstable fate patterns and experimental validation. The upper panels show sequence diagrams. Time flows from top to bottom. Two events that appear on the same vertical line are ordered according to the time flow. The dashed lines synchronize the different vertical lines. All events that appear above a synchronization line occur before all events that appear below the synchronization line. The time-order between two events that appear on parallel vertical lines without a synchronization line is unknown. (A) Proposed sequence of events leading to a stable pattern. The left time line starts with a high inductive signal (IS) and the right time line (B) with a medium IS. (B) Three diagrams that represent possible sequences of events leading to different fate patterns. (C) Experimental validation of the loss of sequential activation in lin-15 mutants, as predicted by the computational model. The pictures visualize cell fate specification in C. elegans with blue and yellow fluorescent proteins (EGL-17::CFP and LIP-1::YFP) expressed during activation of the inductive and lateral signaling pathways, respectively. The upper and middle rows show examples of wild-type animals at mid and late L2 stage expressing the EGL-17 marker in P6.p and the LIP-1 marker in P5.p and P7.p, respectively. The lower row shows examples of a lin-15 mutant at the late L2 stage showing simultaneous expression of both markers in P5.p and P7.p. Reproduced from [20].

revised model with at least one additional negative feedback loop indicated by the inhibition arrow in Fig. 3.

Through model checking an executable model representing the crosstalk between Epidermal growth factor receptor (EGFR) and LIN-12/Notch signalling during *C. elegans* vulval development Fisher et al. gained new insights into the usage of these conserved signalling pathways that control many diverse processes in all animals. While many modelling efforts use simulations that allow investigating only a few possible executions, this work had emphasized the power of analyzing all possible executions using model checking.

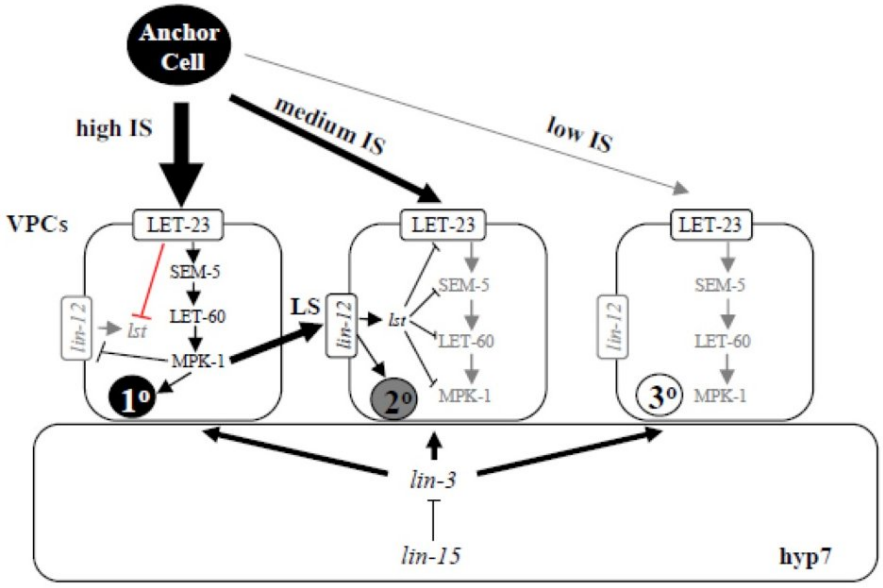
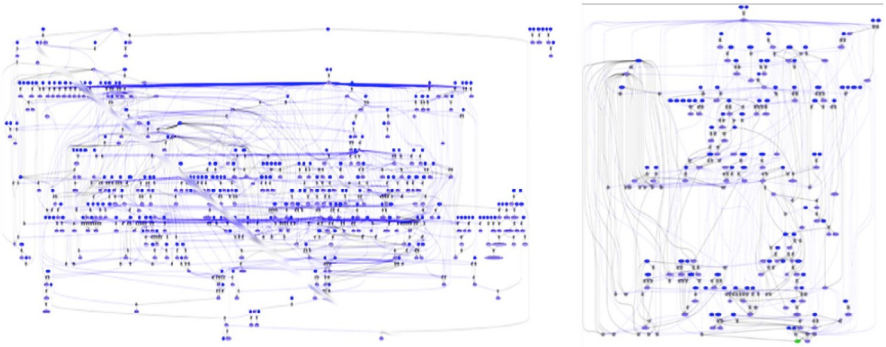


Fig. 3 Conceptual model for the signaling events underlying VPC fate specification. Diagrammatic mechanistic model for the signaling events underlying vulval precursor cell (VPC) fate specification. Inductive signal (IS), lateral signal (LS), cell fates: primary 1^0, secondary 2^0, tertiary 3^0. Reproduced from [20].

3.2 Symbolic analysis of biochemical networks

The idea to use computation tree logic (CTL) as a query language for biochemical networks was first introduced by Fages, Schächter and colleagues in 2004. Chabrier-Rivier et al. were the first to show the potential of using symbolic model-checking tools to evaluate CTL queries in the context of mammalian cell-cycle control and gene expression regulation [11, 12]. More recently, the Biochemical Abstract Machine (BIOCHAM) tool was introduced as an environment to model and analyse biochemical networks using model checking [10]. BIOCHAM is based on a rule-based language that allows the user to write models of biochemical networks and perform multi-level analysis, and a temporal logic language (CTL or LTL) used to formalize experimental data. BIOCHAM permits continuous or stochastic simulations, as well as model validation or revision according to a formal qualitative or quantitative specification. Consequently, BIOCHAM allows to verify that whenever an interaction or a molecule is added to the network, the global properties of the system (expressed in temporal logic) are conserved. In addition, it is possible to automatically search for parameter values that reproduce the specified behaviour of the system under different conditions.



(a) Egf stimulation net (b) Erk activation subnet

Fig. 4 Model of EGF stimulation using *Pathway Logic*. (a) An impression of the Pathway Logic Assistant (PLA) rendering of the model as a Petri net. (b) The subnet of all reactions relevant to activation of Erk in response to a stimulus by EGF is obtained by making Erk1 (and/or Erk2) a goal and asking PLA for the subnet. Reproduced from [33].

The *Pathway Logic* [18] is tightly related to the symbolic model checking approach as it is based on rewriting logic in order to model and analyze signal transduction and biochemical networks, and interpret experimental data. In Pathway Logic, biological molecules, their states, locations, and their role in molecular or cellular processes can be modelled at different levels of abstraction. An example of an EGFR pathway model as a Petri net is shown in Fig. 4a. Pathway Logic can be used to browse the model and ask for subnets or pathways satisfying goals of interest (Fig. 4b) [33].

The modelling of biochemical networks with concurrent transition systems is of a somewhat lower level than with Pathway Logic. Pathway Logic is more expressive as it can express algebraic properties of the components, such as the associativity of complexation. This capability can be used to infer the possible reactions of molecules from their logical structure.

3.3 Insights into signalling crosstalk during pancreatic cancer

Clarke and colleagues applied symbolic model checking to study temporal logic properties in a model of pancreatic cancer. This is the first in-silico model describing the crosstalk between six signalling pathways that have genetic alterations in all pancreatic cancers, with the aim to investigate apoptosis (programmed cell death), proliferation, and cell-cycle arrest. The signalling network model composed of the EGF-PI3K-P53, Insulin/IGF-KRAS-ERK, SHH-GLI, HMGB1-NFkB, RBE2F, WNT-b-Catenin, Notch, TGF b-SMAD, and apoptosis pathway verified temporal logic properties encoding behaviour related to cell fate, cell cycle, and oscillation of

expression level in key proteins. The model agreed well with experimental observations as well as suggested several properties to be tested by experimentally.

4 Underlying Techniques

In this section we revisit the algorithms for model checking and expose some of the techniques used to implement those algorithms efficiently. In general, we term these techniques as *graph representation* and *graph analysis* techniques. After exposition of these techniques we survey some results where these techniques were used for analysis of biological models. In this cases, the knowledge of the techniques used by model-checking tools for the effective analysis of transition systems were used to effectively analyze biological models.

4.1 Symbolic Transition Systems

In Section 2 we concentrated on the explicit representation of models. That is, every snapshot of the status of the system was treated individually. While this approach is very intuitive it has its limitations. Most importantly the size of models that can be handled. For example, a moderately sized model that represents the status of 30 substances, each represented as either active or inactive, has 2^{30} states, which is about one Billion. Approaches that call for direct drawing of the possible transitions of such a model for user inspection are hopeless. But even exploring each one of these states automatically will incur a significant time delay. Alternative approaches to represent the states and executions of systems have been developed [8, 9, 7]. Such approaches are generally termed *symbolic* and they raise the level of representation from that of single states to that of sets of states.

We explore an alternative *symbolic* representation of a transition system. A symbolic transition system is $\mathscr{T} = (V, \rho, \Theta)$ with the following components:

- V is a set of variables, each ranging over a fixed range $R(V)$. By writing formulas over the variables in V we can represent sets of states. For example, the formula $v_1 > 2 \wedge v_2 \leq 3$ represents all the states where the value of v_1 is more than 2 and v_2 is at most 3. A *valuation* σ of the system is an assignment of value to each variable $v \in V$ such that $\sigma(v) \in R(v)$. The language used for representing such formulas depends on the ranges of variables. Here, we assume that all variables range over the natural numbers or (small) sets of natural numbers. We restrict attention to formulas constructed by taking the normal arithmetic operations over natural numbers and variables, comparison between such expressions, and usage of conjunction, disjunction, and negation to combine such terms to large formulas. Thus, formulas can represent the set of valuations such that by assigning the value of the variables into the formula it evaluates to true. In order to represent transitions we use two copies of the variables. We take a

disjoint copy of the variables V and add primes to them V'. The primed variables represent the *next* values of the variables. Thus, by writing a formula like $v_1 = 1 \rightarrow v_1' = 2$ we impose the restriction that whenever v_1 is 1 it changes its value to 2. By writing formulas over $V \cup V'$ we can represent changes in the values of variables, or transitions between states of the model. Thus, a pair of valuations σ and σ' satisfies a formula over $V \cup V'$ if by assigning the values in σ to all the variables in V and the values in σ' to all the variables in V' the formula evaluates to true.

- Accordingly, Θ is a formula over the variables V representing the initial states and ρ is a formula over $V \cup V'$ representing the transitions.

It is simple to convert the symbolic representation to the explicit representation presented in Section 2. Given a symbolic transition system $\mathscr{T} = (V, \rho, \Theta)$ and a set of formulas $\varphi_1, \ldots, \varphi_n$ representing the basic propositions about the model we can construct the explicit transition system $\mathscr{T} = (S, T, S_0, \mathscr{P}, L)$, where S is the set of possible assignments to variables in V, i.e., all the valuations of $\mathscr{T}$, S_0 is the set of states/valuations satisfying the formula Θ, T is the set of pairs of states/valuations satisfying the formula ρ. Finally, the propositions $\mathscr{P}$ are $\{\varphi_1, \ldots, \varphi_n\}$ and L associates with a valuation σ all the propositions that hold true in that valuation.

It seems that we have made a meaningless change of notation. However, the truth could not be further. Now, we have a way of easily representing sets of states as formulas. Furthermore, by manipulating such formulas we can manipulate sets of states. For example, taking two sets of states represented by formulas ϕ_1 and ϕ_2 the formula representing their intersection is $\phi_1 \wedge \phi_2$ and the formula representing their union is $\phi_1 \vee \phi_2$. We can check set equivalence by testing formula equivalence and check whether the set of states represented by a formula is empty by checking whether the formula is satisfiable. By introducing quantification over variables we also can compute the set of successor states of a given set. That is, if ϕ represents a set of states and ρ is the transition relation over variables $V \cup V'$ then the following formula represent the set of states that can be reached from ϕ in one application of ρ:

$$unprime(\exists V.\phi \wedge \rho)$$

That is, the formula $\phi \wedge \rho$ represents the pairs of valuations (σ, σ') such that σ satisfies ϕ and σ' is a successor of σ as (σ, σ') satisfies ρ. Then, $\exists V.\phi \wedge \rho$ throws away the variables in V leaving a formula over V' that characterizes valuations over V' such that there is some value for the variables in V such that $\phi \wedge \rho$ holds for the pair. Exactly the states that are successors of states in ϕ. Finally, we have to translate every reference in $\exists V.\phi \wedge \rho$ to a variable in V' to refer to the same variable in V. This is the role of the *unprime*. It follows that we have a symbolic way to represent the application of the transition to a set of states. We denote this in short as $next_\rho(\phi)$. Similarly, $prev_\rho(\phi) \equiv \exists V'.(\rho \wedge prime(\phi))$, computes the set of predecessors of ϕ. The *prime* operator changes references to V to references to V' resulting in a formula that characterizes all the valuations over V' that satisfy ϕ. Then, adding ρ ensures that we characterize pairs satisfying the transition such that the second satisfies the formula ϕ. Throwing away the variables in V' we get the desired formula.

Now we need two additional tools. Suppose that all variables in V range over finite domains. Then, a variable v ranging over $\{1,\ldots,n\}$ can be represented by $log(n)$ Boolean variables. It follows that we can think about formulas over Boolean variables and in order to use the symbolic approach we need an efficient way to store, manipulate, and evaluate formulas over Boolean variables. Boolean Decision Diagrams (BDDs for short) [8] do exactly that. They are a canonical representation of Boolean formulas making comparison between such formulas very simple. Operations such as conjunction, disjunction, and negation can be implemented efficiently. Finally, existential quantification is done by translating it to disjunction.

The second tools is satisfiability solving. If variables range over finite domains we can translate them to Boolean variables as above and we need a solver for propositional formulas [7]. If variables range over infinite domains we need a theorem prover [31] or a SMT solver [23]. We are now in position to use the symbolic representation for analysis.

4.2 Symbolic Model Checking

The algorithms in Section 2 consisted of annotating states by additional markings corresponding to the CTL formulas holding in them. Here we use formulas to represent the same.

Suppose that we have computed a formula representing the set of states satisfying a CTL formula ψ. This is straight forward for propositions as they are already formulas representing sets of states. Consider a formula of the form $\varphi = E\mathscr{X}\psi$. Then, $prev(\psi)$ is the formula representing the set of states satisfying φ. The set of states satisfying a formula of the form $\varphi = A\mathscr{X}\psi$ is represented by $\neg prev(\neg\psi)$. We have already explained how to handle Boolean connectives and we turn now to the until operator. For a formula $E(\psi_1 \mathscr{U} \psi_2)$ we do the following inductive process. We start with $\phi_0 = \psi_2$, and then compute $\phi_{i+1} = \phi_i \vee (\psi_1 \wedge prev(\phi_i))$. We then compare ϕ_{i+1} to ϕ_i. If they are equivalent, the process has terminated and we have computed the formula representing the set of states satisfying $E(\psi_1 \mathscr{U} \psi_2)$, otherwise, we proceed with another step. Similarly, $A(\psi_1 \mathscr{U} \psi_2)$ is computed by iterating over $\phi_0 = \psi_2$ and $\phi_{i+1} = \phi_i \vee (\psi_1 \wedge \neg prev(\neg\phi_i) \wedge prev(true))$. The need in adding $prev(true)$ is to avoid adding states satisfying ψ_1 that have no successors at all, which obviously do not satisfy $A(\psi_1 \mathscr{U} \psi_2)$.

It turns out that in practice, symbolic model checking, in many cases, outperforms explicit model checking. These techniques combined with BDD representations allowed model checking of hardware designs to scale to systems composed of 10^{120} states and more [9].

4.3 Path Representation

In recent years efficient satisfiability solvers and SMT solvers have been developed [27, 17, 28]. These tools enabled a different approach to model checking. This approach creates a formula representing a set of executions of the model. By asking whether this formula is satisfiable we can search for paths of a certain length. Furthermore, by combining the formula representing executions with additional constraints on the states participating in such executions we can search for executions satisfying given conditions. For example, one could be looking for executions reaching a certain state or set of states. Alternatively, one could search for paths satisfying a sequence of conditions or evolving according to a certain pattern.

Consider a system $\mathscr{T} = (V, \rho, \Theta)$. In order to represent executions of $\mathscr{T}$ of length n we create n copies of V. That is, $V_0, \ldots, V_{n-1}$ are all copies of the variables in V each numbered with the location in the execution. A valuation to the variables in $V_0, \ldots, V_{n-1}$ is now a representation of n states of the system. We now create a formula representing executions of length n by translating Θ to Θ_0, which refers to the first copy V_0 instead of V and translating ρ to $\rho_{i,j}$, which refers to V_i instead of V and to V_j instead of V'. Thus, the formula $P_i \equiv \Theta_0 \wedge \bigwedge_{i=0}^{n-2} \rho_{i,i+1}$ is a formula over the variables $V_0, \ldots, V_{n-1}$. A satisfying assignment to the P_i is a sequence of states such that the first satisfies Θ (through Θ_0) and every pair of adjacent states satisfies ρ (through $\rho_{i,i+1}$). The formula $L_i \equiv \Theta_0 \wedge \bigwedge_{i=0}^{n-2} \rho_{i,i+1} \wedge \bigvee_{i=0}^{n-1} \rho_{n-1,i}$ is satisfiable if there exists a looping execution of length at most n.

4.4 Biological Model Analysis

We now survey results that take advantage of the techniques underlying model checking to improve analysis of biological models. Both apply to the analysis of discrete models that extend Boolean networks; Qualitative Networks (QNs, for short) [32] and Gene Regulatory Networks (GRNs, for short) [34]. We give a short informal introduction to these formalisms and explain how model checking techniques are used to analyze them.

We give a short exposition of QNs and how they give rise to transition systems. A model in *Qualitative Network*s includes variables that represent the concentration of proteins as a discrete value in a (small) fixed range. Changes in variable values are gradual allowing them to increase or decrease by at most 1 in every step of the system. Mathematically, a QN Q includes two components (V,T): The set $V = \{v_1, \ldots, v_n\}$ is a set of variables each ranging over a finite range $D(v_i) = \{j, \ldots, k\} \subseteq \mathbb{N}$, e.g., $\{0,1,2,3\}$. The set $T = \{T_1, \ldots, T_n\}$ include target functions for all variables. A *state* of Q is an assignment $s : V \to \mathbb{N}$ such that for every i we have $s(v_i) \in D(v_i)$. Let S denote the set of all possible states of the QN. Each *target function* $T_i \in T$ associates the target value of variable v_i for every state of the system. Formally, $T_i : S \to D(v_i)$. Intuitively, in state $s \in S$, variable v_i will change to get to the value $T_i(s)$, however will do so in increments/decrements of 1.

It follows that a QN Q gives rise to a transition system $\mathscr{T}_Q = (S, \Delta, S, \mathscr{P}, L)$, where the components of $\mathscr{T}_Q$ are as follows.

- S is the set of states as explained above and all states are initial.
- Δ is the set of transitions that associates with state s the successor s' such that for every variable $v_i \in V$ we have

$$s'(v_i) = \begin{cases} s(v_i)+1 & \text{If } T(s) > s(v_i) \text{ and } s(v_i)+1 \in D(v_i) \\ s(v_i)-1 & \text{If } T(s) < s(v_i) \text{ and } s(v_i)-1 \in D(v_i) \\ s(v_i) & \text{Otherwise} \end{cases}$$

- The set of propositions $\mathscr{P}$ is $v_i = j$ for $v_i \in V$ and $j \in D(v_i)$.
- The labeling L associates with a state s the propositions $v_i = s(v_i)$ for every $v_i \in V$.

Thus, the transition system updates all the variables in the system by allowing them to pursue their target by a change of at most 1. The basic facts labeling each state of the model are the values of the different variables.

One of the most interesting questions regarding qualitative networks has been that of *stabilization*. A network is called stabilizing if there is a unique state s such that $\Delta(s,s)$ and for every other state t it is impossible to get from t to itself by application of Δ. That is, for every $t_1, \ldots, t_n$ such that $\Delta(t_i, t_{i+1})$ for every $1 \leq i < n$ we have $t_1 \neq t_n$. The question of stabilization can be answered by computing the set of states that are visited along arbitrarily long paths. If that set has the size 1, then the network is stabilizing. This observation suggests the following algorithm for checking stabilization. Let $R_0 = S$ and for $i \geq 0$ let $R_{i+1} = \Delta(R_i) = \{s \mid \exists t \in R_i \text{ s.t. } \Delta(t,s)\}$. It is clear that $R_{i+1} \subseteq R_i$. It follows that if a state appears on a cycle in $\mathscr{T}_Q$, it remains in R_i for all i. It is equally easy to see that if a state s is not on a cycle in $\mathscr{T}_Q$ then for some i it does not appear in R_i. Then, by repeatedly computing R_i for increasing values of i one can find a set R_i such that $R_i = R_{i+1}$. It follows that this set R_i includes exactly the set of states that appear on cycles in $\mathscr{T}_Q$. Unfortunately, the straight forward computation of R_i has been elusive. In [32], it was suggested to iteratively compute R_i by abstracting parts of the system. This abstraction lead to a considerable increase in capacity of networks that can be analyzed compared with the naïve approach. In [16], it was suggested that instead of constructing an exact representation of the set R_i, it would be enough to consider subsets of R_i for which the range of each variable is a contiguous range of values. For example, in a system with two variables v_1 and v_2 ranging over $\{0, \ldots, 4\}$ the set that includes the two points $(0,1)$ and $(1,0)$ would be represented by the set of states in which $0 \leq v_1 \leq 1$ and $0 \leq v_2 \leq 1$. Then, the set R_{i+1} is the best set of this form that captures the transition of Q. The results in [16] show that this abstraction technique scales to an order of magnitude larger models than those that could previously be handled. This technique has been made available through the tool BMA [5].

We now turn to survey the usage of model checking in the analysis of GRNs. The transition structure of GRNs and QNs are very similar. The main difference is that in the context of GRNs the target functions are defined in terms of so called *parameters*. Somewhat simplifying presentation, the parameters are the actual value

of the target function per each possible value of the inputs. For example, consider a system with variables v_1, v_2, and v_3 all ranging over $\{0,1,2\}$. If v_1 is affected by v_2 and v_3 then the parameters are essentially the values written in the 3×3 table for all possible options for the values of v_2 and v_3. An entry in the table is the value of the target function of v_1 when v_2 and v_3 are in the appropriate values matching this table entry. One of the interesting usages of model checking in the context of GRNs is to try to narrow down the space of possible values of parameters. In this context, a GRN is given without concrete knowledge of the parameters but with additional dynamic behaviors that are exhibited in experimental results. These dynamic behaviors should be translated to temporal logic specifications that talk about the changes in variable values over time. Then, model checking can be used to check which parameter values allow the network to reproduce this behavior. We note that in this context, model checking is used dually to the way it was described above. Thus, instead of requiring that all behaviors for a given parameter value reproduce the specification we require that for an *admissible* parameter value there be a behavior reproducing the specification.

Technically, in order to enable this check, one has to augment the transition system with variables that represent the parameter values. The transition system encodes the fact that parameter values do not change over an execution and that the changes in the values of "core" GRN variables depend on the values of the parameter variables. Then, a model checking query can create a representation of the set of possible parameter values that allow a particular behavior. This set can be narrowed down by considering multiple dynamic behaviors. See, for example, recent work on such usage of Model Checking in the context of GRNs [6, 4, 3].

5 Probabilistic Model Checking

So far the formalism we used to represent models has been transition systems. Transition systems can represent possible transitions and possible executions, however, they have no quantitative information regarding the prevalence of these transitions. In order to include such information in our models we have to use richer formalisms that include such information. In this section we give a short exposition of continuous time Markov chains (CTMCs). These are models that have discrete states and cannot represent continuous changes in values of variables. However, they do capture probabilities in change from state to state and include also a representation of continuous time. The algorithms involved in the analysis of Markov chains are significantly more complex and we do not review them here. The interested reader is referred to [25, 26, 2].

5.1 Markov chains and Their Analysis

We extend the concept of a transition system to that of a continuous-time Markov chains (CTMCs). For that, we replace the transitions with probabilistic transitions. CTMCs are usually defined via higher level languages that resemble the symbolic representation of transition system. We try to keep the discussion as general as possible and avoid relating to such formalisms. We then introduce the temporal logic continuous stochastic logic (CSL) that is used to express properties of CTMCs.

A CTMC is $\mathscr{T} = (S, T, s_0, \mathscr{P}, L)$, where S and L are as in transition systems. The set of transitions $T : S \times S \to \mathbb{R}^+$ associates with every pair of states (s,t) a non-negative real number $T(s,t)$ corresponding to the *rate* of transition from state s to state t. There is one initial state $s_0 \in S$. The transition $T(s,t) = r > 0$ implies that it is possible to change from state s to state t. The change occurs within t time units with probability $1 - e^{-r \cdot t}$. If multiple transitions from s are possible, i.e., $T(s,t) > 0$ and $R(s,t') > 0$ for $t \neq t'$, then there is a *race* between the transition to t and the transition to t'. The *exit rate* from state s is the sum of positive rates leaving state s, that is $T(s) = \sum_t T(s,t)$. The probability that some transition occurs within t time units is $1 - e^{-T(s) \cdot t}$ and the probability to end up in state t'' is $R(s,t'')/T(s)$. One interesting features of probabilistic systems is that an experiment that has positive probability and is tried often enough will eventually succeed. Accordingly, if a state of the CTMC is revisited often enough every one of its successors will be visited often enough. This leads to the steady state behavior of a CTMC. In the long run, some of the states of the CTMC will be transient: the amount of time spent in them will be 0, and some recurrent: the amount of time spent in them will be greater than 0. Accordingly, the value $R_\infty(s,t)$ is the probability of having started in state s to be in state t in the long run.

We are now ready to define CSL, a version of temporal logic that is used to specify properties of CTMCs [1]. CSL extends CTL by adjusting it to continuous time and replacing path quantification by probabilistic quantification. As before CSL combines the basic facts with Boolean operators and special temporal operators.

- The temporal operators $\mathscr{X}$ and $\mathscr{U}$ are nested within probabilistic path quantification $[\cdot]_{\bowtie p}$, where $\bowtie \in \{>, \geq, <, \leq\}$ and $p \in [0,1]$. Furthermore $\mathscr{U}$ is extended by allowing to state an interval I in which the second formula is expected to happen. Thus, $[\mathscr{X}\phi]_{>0.5}$ means that the measure of paths that satisfy ϕ in the next state is more than 0.5. Similarly, $[\phi_1 \,\mathscr{U}^{[5,7]}\, \phi_2]_{\leq 0.2}$ means that the measure of paths that satisfy that ϕ_1 holds until ϕ_2 holds sometime between 5 and 7 time units from now is at most 0.2.
- The steady-state operator $S(\cdot)_{\bowtie p}$ states that in the long run the probability to be in states that satisfy the subformula must satisfy the probability condition.

As before we make this intuition more formal. Every formula defines a set of states in which it is true. Just like CTL and LTL we denote by $\mathscr{T}, s \models \varphi$ if formula φ is satisfied in state s of $\mathscr{T}$ and $\mathscr{T}, s \not\models \varphi$ otherwise.

- For propositions and Boolean operators the definitions is just as before.

- If φ is $[X\psi]_{\bowtie p}$ then $\mathscr{T},s \models \varphi$ if $prob_{\mathscr{T}}(\pi,\varphi) \bowtie p$. That is, if the probability of the set of paths that start in s and in the next state visit states that satisfy ψ satisfies the comparison with p.
- If φ is $[\psi_1 \mathscr{U} \psi_2]_{\bowtie p}$ then $\mathscr{T},s \models \varphi$ if $prob_{\mathscr{T}}(\pi,\varphi) \bowtie p$. That is, if the probability of the set of paths that start in s and satisfy $\psi_1 \mathscr{U} \psi_2$ such that ψ_2 is visited in the interval I satisfies the comparison with p.
- If φ is $S[\psi]_{\bowtie p}$ then $\mathscr{T},s \models \varphi$ if $\sum_{\mathscr{T},s' \models \psi} R_\infty(s,s') \bowtie p$. That is, if starting in s the long term probability of visiting states that satisfy ψ satisfies the comparison with p.

5.2 Biological Modeling with Markov Chains

Using CTMCs encoding of molecular networks is very direct. A typical model includes a certain set of species of molecules. A state of a model represents the number of molecules of each species (or the concentration level) and transitions correspond to interactions between substances [30].

More formally, a molecular model is defined over a set of substances V. For each substance $v \in V$ one defines the number of levels $l(v)$ of v, implicitly deciding whether v represents molecules explicitly or their concentration level (cf. [22, 13]). Furthermore, the model includes molecular interactions and their rates. For example, phosphorylation would correspond to one molecular species changing to another (non-activated to activate). Similarly, binding of two molecules would correspond to the number of the two simple forms decreasing and the number of the bound substance increasing. Unbinding would be treated dually. So in a state v, each molecular interaction would correspond to a transition that changes the numbers of substances in v according to one interaction taking place. The rate with which the interaction takes place would then depend on the number of possible molecular interactions. Thus, if an interaction is a change from one molecular type to another, its rate is a product of the rate of a single such interaction with the number of instances of the source type in v. If an interaction is a binding, its rate is a product of the number of possible pairs of molecules and thus is a product of the rate of a single such interactions with the number of instances of the first source and the number of instances of the second source. The disadvantage of this approach is that it leads to huge models with many possible transitions enabled from each state. This is especially true if molecules are modeled individually and not through concentration levels. This severely limits the size of models that can be analyzed by model checking. Still, the extensive analysis enabled by model checking makes it useful to analyze even models of bounded size.

One example of an application of such an analysis is the model by Heath et al. of the Mitogen-activated protein kinase (MAPK) [22]. In this paper a CTMC model of the Fibroblast growth factor (FGF) signalling pathway is constructed. Due to the scalability issue mentioned above the model is analyzed with very low number of copies of each molecules. The model is analyzed with questions such as: What is the

probability that a certain species is bound to another species at a given time t? What is the expected number of times that a molecule binds before degrading? The exact analysis of these questions sheds light on the roles of different molecules within the pathway. A similar (in terms of the technology involved) study of the MAPK pathway is available in [24].

Another example is the model of gp130/JAK/STAT signalling pathway using the concentration level approach [21]. Here, probabilistic model checking is used to ensure that the model is of sufficient quality. For example, model checking identifies that one of the substances can "run out" in the system and lead to no further molecular interactions being possible. This highlights the important role of this substance and the need in further modeling of the production of this substance. Then, analysis similar to that described above shows that a full phosphorylation of some of the substances is achieved with high probability.

6 Lessons Learned

To summarize, we highlight the main issues that we believe are important for the development of this approach:

- Model checking is a powerful technique for the analysis of programs. Over the last decade model checking has been successfully used for the analysis of biological models, providing novel insights into various cellular mechanisms and behaviors.
- Many tools providing implementations of model checking are experimental and academic in nature. This implies that users require certain expertise in underlying techniques and formalisms in order to use model checking. The development of more reliable and user-friendly tools, as well as approaches that facilitate the creation of models, could further encourage users with little to no formal background to use these tools for biological modeling and analysis.
- Biological models are somewhat different from software and hardware programs. This calls for the formal methods community to develop dedicated techniques and algorithms that are particularly tailored for the analysis of biological models, leading to improved capacity and efficiency when analyzing biological models.
- Model checking cannot stand on its own as a sole technique of analysis. It is crucial to combine multiple forms of analysis of the same model. One of the major obstacles to combine multiple forms of analysis is the lack of standardization in modeling languages. While SBML provides a standard for mathematical biological models, a similar language that supports further types of models is missing. Such a language could provide a cross-tool foundation for sharing and distributing models enabling analysis by multiple approaches.

References

1. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continuous-time markov chains. ACM Trans. Comput. Logic **1**(1), 162–170 (2000)
2. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
3. Barnat, J., Brim, L., Krejci, A., Safranek, D., Vejnar, M., Vejpustek, T.: On parameter synthesis by parallel model checking. IEEE/ACM Transactions on Computational Biology and Bioinformatics **9**(3), 693–705 (2012)
4. Batt, G., Page, M., Cantone, I., Goessler, G., Monteiro, P., de Jong, H.: Efficient parameter search for qualitative models of regulatory networks using symbolic model checking. Bioinformatics **26**(18), i603–i610 (2010)
5. Benque, D., Bourton, S., Cockerton, C., Cook, B., Fisher, J., Ishtiaq, S., Piterman, N., Taylor, A., Vardi, M.: BMA: Visual tool for modeling and analyzing biological networks. In: 24th International Conference on Computer Aided Verification, *Lecture Notes in Computer Science*, vol. 7358, pp. 686–692. Springer (2012)
6. Bernot, G., Comet, J.P., Richard, A., Guespin, J.: Application of formal methods to biological regulatory networks: extending thomas' asynchronous logical approach with temporal logic. Journal of Theoretical Biology **229**(3), 339–347 (2004)
7. Biere, A., Cimatti, A., Clarke, E., Fujita, M., Zhu, Y.: Symbolic model checking using SAT procedures instead of BDDs. In: Proc. 36st Design Automation Conf., pp. 317–320. IEEE Computer Society (1999)
8. Bryant, R.: Graph-based algorithms for Boolean-function manipulation. IEEE Transactions on Computing **C-35**(8), 677–691 (1986)
9. Burch, J., Clarke, E., McMillan, K., Dill, D., Hwang, L.: Symbolic model checking: 10^{20} states and beyond. In: Proc. 5th IEEE Symp. on Logic in Computer Science, pp. 428–439 (1990)
10. Calzone, L., Fages, F., Soliman, S.: BIOCHAM: an environment for modeling biological systems and formalizing experimental knowledge. Bioinformatics **22**(14), 1805–1807 (2006)
11. Chabrier, N., Fages, F.: Symbolic model checking of biochemical networks. In: Computational Methods in Systems Biology, *Lecture Notes in Computer Science*, vol. 2602, pp. 149–162. Springer-Verlag (2003)
12. Chabrier-Rivier, N., Chiaverini, M., Danos, V., Fages, F., Schächter, V.: Modeling and querying biomolecular interaction networks. Theoretical Computer Science **325**(1), 25–44 (2004)
13. Ciocchetta, F., Hillston, J.: Bio-PEPA: A framework for the modelling and analysis of biological systems. Theoretical Computer Science **410**(33–34), 3065–3084 (2009)
14. Clarke, E., Emerson, E.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Proc. Workshop on Logic of Programs, *Lecture Notes in Computer Science*, vol. 131, pp. 52–71. Springer-Verlag (1981)
15. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press (1999)
16. Cook, B., Fisher, J., Krepska, E., Piterman, N.: Proving stabilization of biological systems. In: Verification, Model Checking, and Abstract Interpretation, *Lecture Notes in Computer Science*, vol. 6538, pp. 134–149. Springer (2011)
17. Eén, N., Sörensson, N.: An extensible sat-solver. In: 6th International Conference on Theory and Applications of Satisfiability Testing, *Lecture Notes in Computer Science*, vol. 2919, pp. 502–518. Springer (2004)
18. Eker, S., Knapp, M., Laderoute, K., Lincoln, P., Meseguer, J., Sönmez, M.: Pathway logic: Symbolic analysis of biological signaling. In: Pacific Symposium on Biocomputing, pp. 400–412 (2002)
19. Fisher, J., Henzinger, T.: Executable cell biology. Nature Biotechnology **25**(11), 1239–49 (2007)
20. Fisher, J., Piterman, N., Hajnal, A., Henzinger, T.: Predictive modeling of signaling crosstalk during *c. elegans* vulval development. PLoS Computational Biology **3**(5), e92 (2007)
21. Guerriero, M.: Qualitative and quantitative analysis of a bio-pepa model of the gp130/jak/stat signalling pathway. Transactions on Computational Systems Biology XI **5750**, 90–115 (2009)

22. Heath, J., Kwiatkowska, M., Norman, G., Parker, D., Tymchyshyn, O.: Probabilistic model checking of complex biological pathways. Theoretical Computer Science **391**(3), 239 – 257 (2008)
23. Kroening, D., Strichman, O.: Decision Procedures: An Algorithmic Point of View. Springer (2008)
24. Kwiatkowska, M., Heath, J.: Biological pathways as communicating computer systems. Jouranl of Cell Science **122**, 2793–2800 (2009)
25. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, *Lecture Notes in Computer Science*, vol. 4486, pp. 220–270. Springer (2007)
26. Kwiatkowska, M., Norman, G., Parker, D.: Using probabilistic model checking in systems biology. SIGMETRICS Performance Evaluation Review **35**(4), 14–21 (2008)
27. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient sat solver. In: Proceedings of the 38th Design Automation Conference, pp. 530–535. ACM (2001)
28. de Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: 14th International Conference Tools and Algorithms for the Construction and Analysis of Systems, *Lecture Notes in Computer Science*, vol. 4963, pp. 337–340. Springer (2008)
29. Pnueli, A.: The temporal logic of programs. In: Proc. 18th IEEE Symp. on Foundations of Computer Science, pp. 46–57. IEEE press (1977)
30. Priami, C., Regev, A., Shapiro, E., Silverman, W.: Application of a stochastic name-passing calculus to representation and simulation of molecular processes. Information Processing Letters **80**(1), 25–31 (2001)
31. Robinson, A., Voronkov, A. (eds.): Handbook of Automated Reasoning. Elsevier (2001)
32. Schaub, M., Henzinger, T., Fisher, J.: Qualitative networks: A symbolic approach to analyze biological signaling networks. BMC Systems Biology **1**(4) (2007)
33. Talcott, C.: Pathway logic. In: Formal Methods for Computational Systems Biology, *Lecture Notes in Computer Science*, vol. 5016, pp. 21–53. Springer-Verlag (2008)
34. 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. Bulleting of Mathematical Biology **55**(2), 247–276 (1995)

7 Glossary

- *Reactive Systems* – A system that consists of parallel processes, where each process may change state in reaction to another process changing state. Biological systems are highly reactive (e.g., cells constantly send and receive signals and operate under various conditions simultaneously).
- *Formal methods* – A collection of methods that relate to formal logic to analyze computer systems and prove properties (written in formal logic) about such systems.
- *Model checking* – A technique for proving that systems have certain properties described in *temporal logic*. In case of proof failure, in most cases, the technique provides a *counter example* – an execution that violates the requirement.
- *Nondeterministic system* – A system that may have several possible reactions to the same stimulus. In biological systems, for example, we can observe various patterns of cell fate under the same genotype. Hence, nondeterministic models capture the diverse behavior often observed in biological systems by allowing different choices of execution, without assigning priorities or probabilities to each choice.
- *Property*/*Requirement*/*Specification* – A formal sentence describing some aspect of a program or system.
- *Transition system* – A computational model for a system. A *state* of a transition system describes the status of the world (restricted to the point of interest of the model/system under study). *Transitions*, which are connections between states describe, the possible changes to the world.
- *Temporal logic* – A specific formalism for describing properties of systems. Temporal logic describes possible evolutions of systems over time. Generally classified to *linear time* or *branching time* according to their view of a computation. In the linear time view a computation is a sequence of the states of the system. Nondeterministic systems have multiple possible computations. In the branching time view a computation is a tree like structure encompassing all possible options of the system. A nondeterministic system has one computation that resembles a tree. *Linear temporal logic (LTL)* and *Computation Tree Logic (CTL)* are examples of a linear time and a branching time logic often used in verification of computer systems.
- *Logic operators* – The combinators of simple logic formulas to more complicated ones. For example, an "and" operator combines two Boolean operands. *Binary operators* and *unary operators* operate on two or one operands respectively. In the context of temporal logic we distinguish between *Boolean operators*, which combine the truth values of formulas at a given point in time (e.g., "and", "or", or "not"), and *Temporal operators*, which combine the truth values of formulas in different time points.
- *Boolean networks* – Computational models that describe a biological system by referring to its components as either "active" or "inactive". Usually, each component relates to a certain protein. Components change their values according to positive and negative influences from other components.

- Petri nets – Computational models that describe the state of the world by associating a number of "tokens" with designated "places". "Transitions" prescribe how tokens can move from place to place leading to a general change of conformation of the system.
- Graph representation – Representing a transition system in the form of a mathematical graph. Nodes of the graph correspond to the states of the system and (directed) edges of the graph correspond to transitions of the system.
- Graph analysis – Applying algorithms on the graph representation of the system.
- Symbolic model checking – Applying the model checking technique by combining reasoning over sets of states instead of reasoning over individual states. Using such techniques model checking can scale to systems with a huge number of states that cannot be enumerated.
- prime operator – In a symbolic representation of a transition system this is our way of saying that a variable relates to the next time point and not the current one.
- Boolean Decision Diagrams (BDDs) – A specific technique for storing sets of states through relating to them as Boolean functions.
- Satisfiability solver – A tool that solves the question of whether a Boolean formula is satisfiable. A Boolean formula is a way of stating constraints over the values of Boolean variables. The formula is then satisfiable if there is a was to assign Boolean values to variables so that the formula evaluates to true.
- SMT solver – A tool that solves the question of whether a formula that combines Boolean parts and additional (theories) parts is satisfiable.
- Qualitative networks – An extension of Boolean Networks that allows more values to represent the possible status of each component and allows a more flexible way of describing how the values of components change over time. The changes in the values of variables are defined through so called target functions, which describe the value that the component aspires to get to. The value of the component then changes gradually until it attains this target.
- Fixed point – a value in a computation that does not change when applying to it some operation. This is used many times to describes states of a system that does not change anymore. Also, in algorithms that compute a set of states by applying a certain operation to them a fixpoint is a set of states that the operation does not change.
- Stabilization – One of the main properties checked for Boolean networks and Qualitative networks. Essentially, this is a property of the system which indicates that the system has exactly one fixpoint. That is, there is a unique stabilization state such that regardless of the starting values of the components in the network, after a long enough execution the stabilization state is reached and never changed anymore.
- Continuous time Markov chains (CTMCs) – A computational model combining discrete state transitions with continuous time and probabilities. As in general transition systems, the state of the world is described via a "state", however,

there is a probability distribution over the time that the system stays in the same state and in case of change to which state the system changes.