# Machine Learning Biochemical Networks from Temporal Logic Properties

Laurence Calzone, Nathalie Chabrier-Rivier,
François Fages, Sylvain Soliman

Firstname.Lastname@inria.fr
Projet Contraintes, INRIA Rocquencourt,
BP105, 78153 Le Chesnay Cedex, France.
http://contraintes.inria.fr/

**Abstract.** One central issue in systems biology is the definition of formal languages for describing complex biochemical systems and their behavior at different levels. The biochemical abstract machine BIOCHAM is based on two formal languages, one rule-based language used for modeling biochemical networks, at three abstraction levels corresponding to three semantics: boolean, concentration and population; and one temporal logic language used for formalizing the biological properties of the system. In this paper, we show how the temporal logic language can be turned into a specification language. We describe two algorithms for inferring reaction rules and kinetic parameter values from a temporal specification formalizing the biological data. Then, with an example of the cell cycle control, we illustrate how these machine learning techniques may be useful to the modeler.

## 1 Introduction

One promise of systems biology is to model biochemical processes at a sufficiently large scale so that the behavior of a complex system can be predicted under various conditions. The language approach to systems biology aims at designing formal languages for describing biochemical mechanisms, processes and systems at different levels of abstraction. The pioneering use in [1] of the $\pi$-calculus process algebra for modeling cell signalling pathways, has been the source of inspiration of numerous works in the line of process calculi [2–4] and their stochastic extensions [5].

Recently, the question of formalizing the biological properties of the system has also been raised, and formal languages have been proposed for this task, most notably using temporal logics in either boolean [6, 7], discrete [8–10] or continuous models [11, 12].

The biochemical abstract machine BIOCHAM [13, 14] has been designed as a simplification of the process calculi approach using a language of reaction rules that is both more natural to the biologists and well suited to apply model-checking techniques [15]. The rule-based language is used for modeling biochemical networks at three abstraction levels corresponding to three formal semantics:

boolean, concentration and population. In the boolean case, where one reasons only about the presence or absence of molecules, the reaction rules are highly non-deterministic rewriting rules. This setting is similar to Pathway Logic [6] and Petri nets. In the concentration (resp. population) semantics, the rules are equipped with kinetic expressions which provide a continuous dynamics with Ordinary Differential Equations (ODEs) (resp. continuous time Markov chains), somewhat similarly to the bio-calculus [16, 17]. One striking feature of this multi-level approach is that in the three cases, temporal logic can be used to formalize the biological properties of the system, and verify them by model-checking techniques.

Turning the temporal logic language into a specification language for expressing the observed behavior of the system opens the way to the use of machine learning techniques for completing or correcting such formal models semi-automatically. There has been work on the use of machine learning techniques, such as inductive logic programming [18], to infer gene functions [19], metabolic pathway descriptions [20, 21] or gene interactions [8]. However learning biochemical reactions from temporal properties is quite new, both from the machine learning perspective and from the systems biology perspective.

In this paper, we first describe the basic BIOCHAM language of biochemical processes with its three semantical levels, and the temporal logic language used for formalizing the biological properties of the system. We then present a structural learning algorithm for inferring reaction rules, or more generally model revisions, from a temporal specification of the system at the boolean abstraction level. Next, we detail a similar algorithm for finding kinetic parameter values from a temporal specification at the concentration abstraction level. These algorithms and their implementation in BIOCHAM[1] are illustrated in Sect. 6 on a model of the cell cycle control after Qu et al. [22], with examples of model revision, refinement and parameter search. Finally, we conclude on the merits of this approach, its limits and on some perspectives for future work.

## 2 BIOCHAM's Description Language of Biochemical Processes

### 2.1 Syntax

BIOCHAM rules represent biomolecular reactions between chemical or biochemical compounds, ranging from small molecules to proteins and genes. The syntax of the formal objects involved, and their reactions, is given by the following (simplified) grammar:

[1] BIOCHAM is a free software implemented in Prolog and distributed under the GPL license. It is downloadable on the web at `http://contraintes.inria.fr/BIOCHAM`. The data used in this paper are available at `http://contraintes.inria.fr/BiochamLearning/TCSB`

object = molecule | molecule :: location
molecule = name | molecule-molecule |molecule~{name,...,name}
reaction = solution => solution | kinetics *for* solution => solution
solution = _ | object | number*object | solution + solution

The basic object is a molecular compound. Thanks to the "::" operator, it can be given a precise location, which is a simple name representing a (fixed) compartment, such as the nucleus, the cytoplasm, the membrane, etc. The binding operator "–" is used to represent complexations and other forms of intermolecular bindings. The alteration operator "~" makes it possible to attach to a compound a set of modifications, such as the set of phosphorylated sites of a protein. For instance, `A~{p}` denotes a phosphorylated form of the compound `A`, and `A~{p}-B` denotes its complexation with `B`.

Reaction rules transform one formal solution into another one. The following abbreviations are used: `A =[C]=> B` for the catalyzed reaction `A+C => C+B`, and `A <=> B` for the reversible reaction equivalent to the two symmetrical reactions `A => B` and `B => A`. The constant "_" represents the empty solution. It is used for instance in protein *degradation rules*, like `A => _`, and in *synthesis rules*, like `_ =[G]=> A` for the synthesis of `A` by the (activated gene) catalyst `G`. The other main rule schemas are *(de)complexation rules*, like `A + B => A-B` for the complexation of `A` and `B`, *(de)phosphorylation rules*, like `A =[B]=> A~{p}` for the phosphorylation of `A` catalyzed by the kinase `B`, and *transport rules*, like `A::nucleus => A::cytoplasm` for the transport of `A` from the nucleus to the cytoplasm.

Reactions can be given kinetic expressions. For instance,
`k*[A]*[B] for A=[B]=>A~{p}` specifies a mass action law kinetics with parameter $k$ for the reaction. These expressions can be written explicitly, allowing any kinetics, or using shortcuts like `MA(k)` for a Mass Action law with parameter $k$, or `MM(Vm,Km)` for a Michaelian kinetics.

For an example of a complete model, we refer to the Appendix A, which contains a transcription of the model of the cell cycle control of Qu et al. [22], explained in Sect. 6.

BIOCHAM also offers a rich language of patterns and constraints [23] used to denote sets of objects or reaction rules, in a concise manner. A rule pattern basically replaces, in a rule expression, some objects by variables, written `$A, $B`, and adds constraints on the values these variables can take. For instance, the constraint `$A diff $B` imposes that the two molecules are different, or `$A more_phos_than $B` imposes that `$A` is more phosphorylated than `$B`. The precise description of patterns is however beyond the scope of this article, and we refer to [14] for details. Patterns are used to write large models in a concise manner, and to specify the kinds of rules to learn as explained in Sect. 4.

### 2.2 Boolean Semantics

The most abstract semantics of BIOCHAM rules is the boolean one. In that semantics, one associates to each BIOCHAM object a boolean variable representing its presence or absence in the system. Reaction rules are then interpreted

as an *asynchronous transition system* over states defined by the vector of boolean variables. A rule such as `A+B=>C+D` defines four possible transitions corresponding to the complete or incomplete consumption of the reactants `A` and `B`. Such a rule can only be applied when both `A` and `B` are present in the current state. In the next state, `C` and `D` are then present, while `A` and `B` can either be present (partial consumption) or absent (complete consumption). This choice is made in a non-deterministic fashion, in order to *over-approximate* all the possible behaviors of the system. The transition system is asynchronous, only one rule is applied at a given time, in order to represent the basic biological phenomena such as competitive inhibition, where a reaction "hides" another one. On the other hand, the hypothesis of synchrony (i.e. all rules that can be fired in a given state are fired simultaneously) would not faithfully represent the competition between reactions.

The boolean semantics is thus highly non-deterministic. The temporal evolution of the system is modeled by the succession of states in a path, and the different possible behaviors by the non-deterministic choice of a transition at each step.

The Appendix B shows a standard graphical representation of the model in Appendix A where the degree of non-determinism can be roughly estimated visually by the number of edges outgoing from the circular nodes representing the molecules.

Formally, the boolean semantics of BIOCHAM rules is defined via a *Kripke structure* $K = (S, R)$ where $S$ is the set of states defined by the vector of boolean variables, and $R \subseteq S \times S$ is the transition relation between states, supposed to be total (i.e. $\forall s \in S, \exists s' \in S$ s.t. $(s, s') \in R$). When the transition relation defined by the reaction rules of a BIOCHAM model is not total, it is automatically completed by loops on states having no successor. A path in $K$, starting from state $s_0$ is an infinite sequence of states $\pi = s_0, s_1, \cdots$ such that $(s_i, s_{i+1}) \in R$ for all $i \geq 0$. In the following, we will denote by $\pi^k$ the path $s_k, s_{k+1}, \cdots$.

### 2.3 Concentration Semantics

The concentration semantics is a more concrete semantics, where one associates to each BIOCHAM object a real number representing its concentration. Reaction rules are interpreted with their kinetic expressions by a set of nonlinear ordinary differential equations (ODE)[2]. Formally, to a set of BIOCHAM reaction rules $E = \{e_i \texttt{ for } S_i \texttt{ => } S'_i\}_{i=1,...,n}$ with variables $\{x_1, ..., x_m\}$, one associates the system of ODEs :

$$dx_k/dt = \sum_{i=1}^{n} r_i(x_k) * e_i - \sum_{j=1}^{n} l_j(x_k) * e_j$$

[2] The kinetic expressions in BIOCHAM can actually contain conditional expressions, in which case the reaction rules are interpreted by a deterministic hybrid automaton. For the sake of simplicity, we shall not detail this more general setting here.

where $r_i(x_k)$ (resp. $l_i$) is the stoichiometric coefficient of $x_k$ in the right (resp. left) member of rule $i$.

Given an initial state, i.e. initial concentrations for each of the objects, the evolution of the system is deterministic, and numerical integration algorithms compute a time series describing the temporal evolution of the system variables. The integration methods actually implemented in BIOCHAM are the adaptive step-size Runge-Kutta method and the Rosenbrock implicit method for stiff systems, which both produce simulation traces with variable time steps.

Figures 1, 2 and 3 in Sect. 6 show graphical views of the result of such computations in the model of Qu et al. with different parameter values.

### 2.4 Population Semantics

The population semantics is the most realistic semantics but also the most difficult to compute. This semantics associates to each BIOCHAM object an integer representing the number of molecules in the system. Rules are interpreted as a continuous time Markov chain where transition probabilities are defined by the kinetic expressions of BIOCHAM reaction rules.

Stochastic simulation techniques [24] compute realizations of the process. The results are generally noisy versions of those obtained with the concentration semantics. However, in models with, for instance, very few molecules of some kind, qualitatively different behaviors may appear in the stochastic simulation, and thus justify the recourse to that semantics in such cases. A classical example is the model of the lambda phage virus [25] in which a small number of molecules, promotion factors of two genes, can generate an explosive multiplication (lysis) after a more or less long period of passive wait (lysogeny).

In the population semantics, for a given volume $V_i$ of the location where the reaction occurs, a concentration C is translated into a number of molecules $N = C \times V_i \times K$, where $K$ is Avogadro's number. The kinetic expression $e_i$ for the reaction $i$ is converted into a transition rate $\tau_i$ (giving a transition probability after normalization) as follows [25]:

$$\tau_i = e_i \times (V_i \times K)^{(1-\sum_{k=1}^{m} l_i(x_k))} \times \prod_{k=1}^{m} (!l_i(x_k))$$

where $l_i$ is the stoichiometric coefficient of the reactant $x_k$ in the reaction rule $i$. In particular we have:

- $\tau_i = e_i$ for reactions of the form `A =>...`,
- $\tau_i = \frac{e_i}{V_i \times K}$ for reactions of the form `A+B=>...`,
- $\tau_i = 2 \times \frac{e_i}{V_i \times K}$ for reactions of the form `A+A=>...`,
- etc.

## 3 BIOCHAM's Description Language of Biological Properties

A second language based on temporal logic is used in BIOCHAM to formalize the biological properties of a system. The temporal logics CTL (*Computation Tree*

*Logic*), LTL (*Linear Time Logic*) and PLTL (*Probabilistic LTL*) with numerical constraints are used in the three semantics respectively. They are used to express in a formal manner the biological properties of the system that the model is supposed to capture. These properties correspond to the biological experiments (observations and measures done in wild-life or mutated organisms, etc.) that are used to build and validate the model.

Their formalization makes it possible to consider these properties as a *specification* to be preserved through operations of model correction, refinement, simplification or composition. The possibility to verify automatically such a specification with model-checking techniques opens the way to the curation, comparison and re-use of different models in a much more systematic fashion than what the current state-of-the-art permits.

### 3.1 CTL for the Boolean Semantics

The *Computation Tree Logic* CTL$^*$ [15] is an extension of classical logic that allows reasoning about an infinite tree of state transitions. It uses operators about branches (non-deterministic choices) and time (state transitions). Two path quantifiers $A$ and $E$ are thus introduced to handle non-determinism: $A\phi$ meaning that $\phi$ is true on all branches, and $E\phi$ that it is true on at least one branch. The time operators are $F, G, X, U$ and $W$; $X\phi$ meaning $\phi$ is true at the next transition, $G\phi$ that $\phi$ is always true, $F\phi$ that $\phi$ is eventually true, $\phi\ U\ \psi$ meaning $\phi$ is always true until $\psi$ becomes true, and $\phi\ W\ \psi$ meaning $\phi$ is always true until $\psi$ might become true. In this logic, $F\phi$ is equivalent to $true\ U\ \phi$, $\phi\ W\ \psi$ to $(\phi\ U\ \psi)|G\phi$, and the following duality properties hold: $!(EF(\phi)) = AG(!\phi)$, $!(E\ \phi\ U\ \psi) = A(!\psi\ W\ !\phi)$ and $!(E\phi\ W\ \psi) = A(!\psi\ U\ !\phi)$, where ! denotes negation.

In the CTL fragment of CTL$^*$ used in BIOCHAM for implementation reasons, each temporal operator must be preceded by a path operator, and each path operator has to be immediately followed by a temporal operator. Table 1 defines the truth value of a formula in a Kripke structure where states are defined by boolean variables.

A BIOCHAM model $\mathcal{M}$ is defined by a set of rules and a set of possible initial states. The rules of $\mathcal{M}$ define the Kripke structure $K$ associated to the model. As the initial state may not be completely defined, we distinguish between the truth of a formula $\phi$ in all initial states of $\mathcal{M}$, noted $M \models Ai\ \phi$ (or simply $M \models \phi$), and the truth of $\phi$ in some initial state of $\mathcal{M}$, noted $M \models Ei\ \phi$.

We refer to [7, 26, 13, 23], Sect. 6.3 and to Appendix A for examples of biological properties of a system expressed by CTL formulae. Some of the most used CTL formulae are abbreviated in BIOCHAM as follows:

- `reachable(P)` stands for $EF(P)$;
- `steady(P)` stands for $EG(P)$;
- `stable(P)` stands for $AG(P)$;
- `checkpoint(Q,P)` stands for $!E(!Q\ U\ P)$;
- `oscil(P)` stands for $AG((P \Rightarrow EF\ !P) \wedge (!P \Rightarrow EF\ P))$.

| | |
|---|---|
| $s \models \alpha$ | iff $\alpha$ is a propositional formula true in the state $s$, |
| $s \models E\psi$ | iff there exists a path $\pi$ starting from $s$ s.t. $\pi \models \psi$, |
| $s \models A\psi$ | iff for all paths $\pi$ starting from $s$, $\pi \models \psi$, |
| $s \models !\psi$ | iff $s \not\models \psi$, |
| $s \models \psi \ \& \ \psi'$ | iff $s \models \psi$ and $s \models \psi'$, |
| $s \models \psi \mid \psi'$ | iff $s \models \psi$ or $s \models \psi'$, |
| $s \models \psi \Rightarrow \psi'$ | iff $s \models \psi'$ or $s \not\models \psi$, |
| | |
| $\pi \models \phi$ | iff $s \models \phi$ where $s$ is the first state of $\pi$, |
| $\pi \models X\psi$ | iff $\pi^1 \models \psi$, |
| $\pi \models F\psi$ | iff there exists $k \geq 0$ s.t. $\pi^k \models \psi$, |
| $\pi \models G\psi$ | iff for all $k \geq 0$, $\pi^k \models \psi$, |
| $\pi \models \psi \ U \ \psi'$ | iff there exists $k \geq 0$ s.t. $\pi^k \models \psi'$ and $\pi^j \models \psi$ for all $0 \leq j < k$. |
| $\pi \models \psi \ W \ \psi'$ | iff either there exists $k \geq 0$ s.t. $\pi^k \models \psi'$ and $\pi^j \models \psi$ for all $0 \leq j < k$, or for all $k \geq 0$, $\pi^k \models \psi$. |
| $\pi \models !\psi$ | iff $\pi \not\models \psi$, |
| $\pi \models \psi \ \& \ \psi'$ | iff $\pi \models \psi$ and $\pi \models \psi'$, |
| $\pi \models \psi \mid \psi'$ | iff $\pi \models \psi$ or $\pi \models \psi'$, |
| $\pi \models \psi \Rightarrow \psi'$ | iff $\pi \models \psi'$ or $\pi \not\models \psi$, |

**Table 1.** Inductive definition of the truth value of a CTL* formula in a given state $s$ or path $\pi$, for a Kripke structure $K$.

– `loop(P,Q)` stands for $AG((P \Rightarrow EF\ Q) \wedge (Q \Rightarrow EF\ P))$.

Without strong fairness assumption, it is worth noting that the last two abbreviations are actually necessary but not sufficient conditions for oscillations. The correct formula for oscillations is indeed a CTL* formula that cannot be expressed in CTL: $EG((P \Rightarrow F\ !P) \wedge (!P \Rightarrow F\ P))$ [27].

The abbreviations can be used inside CTL formulae. For instance, the formula `reachable(steady(P))` expresses that the steady state denoted by formula $P$ is reachable, or the formula `AG(!P -> checkpoint(Q,P))` expresses that $Q$ is a checkpoint for $P$ not only in the initial state but in all reachable states.

### 3.2 LTL with Numerical Constraints for the Concentration Semantics

The *Linear Time Logic*, LTL is the fragment of CTL* that uses only temporal operators. A first-order version of LTL is used to express temporal properties about the molecular concentrations in the simulation trace. A similar approach is used in the DARPA BioSpice project [11]. The choice of LTL is motivated by the fact that the concentration semantics given by ODEs is deterministic, and there is thus no point in considering path quantifiers.

The version of LTL with arithmetic constraints we use, considers first-order atomic formulae with equality, inequality and arithmetic operators ranging over real values of concentrations and of their derivatives. For instance `F([A]>10)`

expresses that the concentration of `A` eventually gets above the threshold value 10. `G([A]+[B]<[C])` expresses that the concentration of $C$ is always greater than the sum of the concentrations of $A$ and $B$. Oscillation properties, abbreviated as `oscil(M,K)`, are defined as a change of sign of the derivative of $M$ at least $K$ times:
`F((d[M]/dt > 0) & F((d[M]/dt < 0) & F((d[M]/dt > 0)...)))`. The abbreviated formula `oscil(M,K,V)` adds the constraint that the maximum concentration of $M$ must be above the threshold $V$ in at least $K$ oscillations.

For practical purposes, some limited forms of quantified first-order LTL formulae are also allowed. As an example of this, constraints on the periods of oscillations can be expressed with a formula such as `period(A,75)`, defined as $\exists t\ \exists v\ F(Time = t\ \&\ [A] = v\ \&\ d([A])/dt > 0\ \&\ X(d([A])/dt < 0)\ \&\ F(Time = t + 75\ \&\ [A] = v\ \&\ d([A])/dt > 0\ \&\ X(d([A])/dt < 0)))$ where $Time$ is the time variable.

Note that the notion of *next state* (operator $X$) refers to the state of the following time point computed by the (variable step-size) simulation, and thus does not necessarily imply real-time neighborhood. Nevertheless, for computing for instance local maxima as in the formula above, the numerical integration methods do compute the relevant time points with a good accuracy.

### 3.3 PLTL with Integer Constraints for the Population Semantics

For the stochastic semantics, it is natural to consider the PCTL logic [28] which basically replaces the path operators of CTL, $E$ and $A$, by the operator $P_{\bowtie p}$, which represents a constraint $\bowtie_p$ on the probability that the formula under $P_{\bowtie p}$ is true. For instance, $A(\psi\ U\ \psi')$ becomes $P_{\geq 1}(\psi\ U\ \psi')$, i.e. the probability that $\psi\ U\ \psi'$ is realized is 1. The atomic formulae considered here are first-order formulae with arithmetic constraints, ranging on integers representing numbers of molecules.

However, for efficiency reasons explained in Sect. 5.3, a fragment of PCTL formulae, called PLTL, in which the $P_{\bowtie p}$ operator can only appear once as head of the formula, is actually considered in BIOCHAM.

## 4 Learning Reaction Rules from CTL Properties

The description language of biological properties based on temporal logic can be used to *specify* the expected behavior of a system, and to let a machine learning algorithm automatically search for possible model revisions. In BIOCHAM, the structural learning of reaction rules relies on the boolean semantics. A CTL specification is thus used to formalize the expected temporal properties of the model, in the process of learning rules.

### 4.1 Symbolic Model-Checking Algorithm

The learning algorithm necessitates to check the truth of CTL formulae, and makes use of counter-examples to direct the search of corrections to the model.

In BIOCHAM, the CTL formulae are evaluated through an interface to the symbolic model-checker NuSMV [29]. NuSMV also computes counter-examples in the form of pathways, which is used, for instance, in the search process to revise the model.

The performances obtained on a large model of the cell cycle control after Kohn's map [30], involving 800 rules and 500 variables, have been shown to be of the order of a few tenths of seconds to compile the model, and check simple CTL formulae [26]. These performances are nevertheless far below those obtained classically with NuSMV on much more strongly structured models of circuits or programs. One characteristic of the boolean models of BIOCHAM is their very high degree of non-determinism. This may lead to performance problems on small size models having a high number of parallel pathways, which is the case for instance in the MAPK model of [31]. These difficulties are recognized in the symbolic model-checking community [32] and biochemical networks provide interesting examples of problems with a high circuit width [33]. On the other hand, the most efficient model-checking tools based on a representation with Petri nets assume a 1-boundedness condition stating that at most one token can be present in a place [34], which is generally not satisfied in biochemical networks.

### 4.2 Learning One Rule

The boolean semantics of BIOCHAM can be used straightforwardly to search for one rule to add to, or suppress from, a model in order to satisfy a CTL specification. The search can be restricted by providing rule patterns with constraints.

The command `learn_one_addition(reaction_pattern,spec_CTL)` enumerates each instance of the rule pattern provided as first argument, that, when added to the model, is sufficient to satisfy the CTL specification given as second argument (or given implicitly by the command `add_spec` if there is no second argument). The time complexity grows linearly in the number of instances of the rule pattern, which is thus the main complexity factor added to the checking of the CTL specification.

The command `learn_one_deletion(reaction_pattern,spec_CTL)` enumerates each instance of the rule pattern that is sufficient to delete from the model to satisfy the CTL specification. Here, the time complexity added to the checking of the CTL specification is linear in the number of rules in the model.

It is worth noting that, by iterating the search of rules that can be deleted in a model while preserving its specification, one can easily implement a command `reduce_model(spec_CTL)` to compute a *minimal* subset of rules satisfying a given specification.

### 4.3 Learning Several Rules

In order to guide the automatic search for the addition and deletion of several rules, the CTL formulae can be divided into three classes. The ECTL class regroups CTL formulae that do not contain the $A$ operator (i.e. no positive

occurrence of $A$ or negative occurrence of $E$). The ACTL class regroups CTL formulae that do not contain the $E$ operator. The other formulae are regrouped in the UCTL class. The reason for this classification is that, in order to satisfy a false ACTL formula it is necessary to *delete* rules from the model, whereas to satisfy a false ECTL formula, it is necessary to *add* rules to the model:

**Proposition 1.** *Let $K = (S, R)$ and $K' = (S, R')$ be two Kripke structures such that $R \subseteq R'$. For any ECTL formula $\phi$, if $s \not\models_{K'} \phi$ then $s \not\models_K \phi$. For any ACTL formula $\phi$, if $s \not\models_K \phi$ then $s \not\models_{K'} \phi$.*

*Proof.* Let us prove the proposition for ECTL formulae by contrapositive, i.e. if $s \models_K \phi$ then $s \models_{K'} \phi$. The proof is done by induction on the size of the proof of $s \models_K \phi$, with $\phi$ supposed to be in negation-free form, except inside propositional formulae, and using only the temporal operators $X$, $G$ and $U$, thanks to the equivalences and duality properties given in section 3.1.

If $\phi = \alpha$ is propositional, we have $\alpha$ is true in the state $s$, and since $s$ is also a state of $K'$ we get $s \models_{K'} \phi$.

If $\phi = \psi_1$ & $\psi_2$ (resp. $\psi_1$ | $\psi_2$) then we have by definition $s \models_K \psi_1$ and (resp. or) $s \models_K \psi_2$, by induction hypothesis we get $s \models_{K'} \psi_1$ and (resp. or) $s \models_{K'} \psi_2$ and we can conclude that $s \models_{K'} \phi$.

The only remaining case is when $\phi = E\psi$ with $\psi$ starting with a temporal operator. We know that there exists a path $\pi$ of $K$ such that $\pi \models_K \psi$. Note that since $R \subseteq R'$, $\pi$ is also a path in the structure $K'$. Now,

- if $\psi = X\psi'$, we have $\pi^1 \models_K \psi'$, by induction hypothesis, $\pi^1 \models_{K'} \psi'$ and since $\pi$ is a path of $K'$ we get $\pi \models_{K'} X\psi'$, q.e.d.
- if $\psi = G\psi'$, we have $\pi^n \models_K \psi'$ for all $n \geq 0$, by applying the induction hypothesis to all states $\pi^n$, we get $\pi^n \models_{K'} \psi'$ for all $n \geq 0$, and since $\pi$ is a path of $K'$ we get $\pi \models_{K'} G\psi'$, q.e.d.
- if $\psi = \psi' \; U \; \psi''$, we have $\pi^n \models_K \psi''$ for some $n \geq 0$ and $\pi^m \models_K \psi'$ for all $0 \leq m < n$, by applying the induction hypothesis to all $\pi^m$ for $0 \leq m \leq n$ we get, $\pi^n \models_{K'} \psi''$, and $\pi^m \models_{K'} \psi'$ for all $0 \leq m < n$, thus $\pi \models_{K'} \psi' \; U \; \psi''$, q.e.d.

For an ACTL formula $\phi$, we once again prove the proposition by contrapositive and induction on the proof of $s \models_{K'} \phi$. The only difference with the above proof is when we get to the case $\phi = A\psi$, we have to prove that $\pi \models_K \psi$ for all paths $\pi$ starting from $s$ in $K$ but since they are all paths of $K'$ we can apply the induction hypothesis and the same reasoning as above applies. We get $s \models_K \phi$, q.e.d. □

Following this proposition, model revision algorithms allowing to add and delete several rules in a model, can be defined by fixing strategies for choosing the next CTL formula to satisfy, and for searching the rules to add to, or delete from the the model, according to the class of the formula.

In BIOCHAM, the command

```
revise_model(reaction_pattern, spec_CTL)
```

implements such a model revision algorithm. It enumerates sets of rule additions and deletions that are sufficient to satisfy the specification. The CTL specification, as well as a reaction pattern for the rules allowed to be added or deleted, can be provided as arguments in the command.

To satisfy an unsatisfied ECTL formula, the algorithm tries to add one rule among the instances of the pattern, and checks it by model-checking. To satisfy an unsatisfied ACTL formula, the algorithm searches for rule deletions only, among the pattern instances that are in the set of the rules provided by the model-checker as a counter-example. To satisfy an unsatisfied UCTL formula, the algorithm tries both adding and removing rules, as described formally below. More than one set of additions/deletions being possible at each step, the algorithm implements a backtracking procedure to explore all choices.

ECTL formulae are treated first, UCTL second and then ACTL. However, the deletions necessary to satisfy an ACTL formula are allowed to dissatisfy some ECTL or UCTL formulae, in which case they are treated again with the constraint not to dissatisfy the ACTL formulae already satisfied.

Formally, we present the *model revision algorithm* by a non-deterministic transition system over configurations of the form $[Q_t, Q, R]$ where:

- $R$ is the current set of reaction rules.
- $Q_t = (E_t, U_t, A_t)$ is the set of *treated* CTL formulae that are true in $R$. These formulae are sorted in three groups: the set $E_t$ of ECTL formulae, the set $U_t$ of unclassified (UCTL) formulae, and the set $A_t$ of ACTL formulae.
- $Q = (E, U, A)$ is the set of *untreated* CTL formulae, sorted similarly in three groups of ECTL, UCTL and ACTL formulae.

By a slight abuse of notation, we write $R \models \phi$ if the rule set $R$ satisfies the CTL formula $\phi$. The rules considered for addition (noted $r$ in the transitions E' and U' below) are instances of the rule pattern. The rule sets considered for deletion (noted $Re$ in transitions U" and A' below) are chosen among the rules computed by the model-checker as a counter-example to the formula.

The initial configuration is the configuration $[(\emptyset, \emptyset, \emptyset),\ (E, U, A),\ R]$, that is, we start with a model $R$ and an untreated CTL specification. The model revision algorithm performs the following transitions:

**E:** $[(E_t, U_t, A_t),\ (E \cup \{e\}, U, A),\ R] \rightarrow [(E_t \cup \{e\}, U_t, A_t),\ (E, U, A),\ R]$
if $R \models e$

**E':** $[(E_t, U_t, A_t),\ (E \cup \{e\}, U, A),\ R] \rightarrow [(E_t \cup \{e\}, U_t, A_t),\ (E, U, A),\ R \cup \{r\}]$
if $R \not\models e$ and $\forall f \in \{e\} \cup E_t \cup U_t \cup A_t \quad R \cup \{r\} \models f$

**U:** $[(E_t, U_t, A_t),\ (\emptyset, U \cup \{u\}, A),\ R] \rightarrow [(E_t, U_t \cup \{u\}, A_t),\ (\emptyset, U, A),\ R]$
if $R \models u$

**U':** $[(E_t, U_t, A_t),\ (\emptyset, U \cup \{u\}, A),\ R] \rightarrow [(E_t, U_t \cup \{u\}, A_t),\ (\emptyset, U, A),\ R \cup \{r\}]$
if $R \not\models u$ and $\forall f \in \{u\} \cup E_t \cup U_t \cup A_t \quad R \cup \{r\} \models f$

**U":** $[(E_t, U_t, A_t),\ (\emptyset, U \cup \{u\}, A),\ R \cup Re] \rightarrow [(E_t, U_t \cup \{u\}, A_t),\ (\emptyset, U, A),\ R]$
if $R \cup R_e \not\models u$ and $\forall f \in \{u\} \cup E_t \cup U_t \cup A_t \quad R \models f$

**A:** $[(E_t, U_t, A_t),\ (\emptyset, \emptyset, A \cup \{a\}),\ R] \rightarrow [(E_t, U_t, A_t \cup \{a\}),\ (E_p, U_p, A),\ R]$
if $R \models a$

**A':** $[(E_t \cup E', U_t \cup U', A_t), (\emptyset, \emptyset, A \cup \{a\}), R \cup Re] \rightarrow [(E_t, U_t, A_t \cup \{a\}), (E', U', A), R]$
if $R \cup Re \not\models a$, $\forall f \in \{a\} \cup E_t \cup U_t \cup A_t \;\; R \models f$ and $\forall g \in E' \cup U' \; R \not\models g$.

**Proposition 2 (termination and correctness).** *Given a CTL specification $(E, U, A)$, the model revision algorithm terminates in at most $|A| * |E \cup U|$ transitions. Moreover, if the terminal configuration is of the form $[(E_t, U_t, A_t), (\emptyset, \emptyset, \emptyset), R]$ then the revised model $R$ satisfies the specification $(E_t, U_t, A_t)$ which is equivalent to the initial specification $(E, U, A)$.*

*Proof.* The correctness of the algorithm comes from the fact that each transition maintains only true formulae in the satisfied set, which can be trivially checked for each transition. Moreover, the rules merely exchange formulae between the untreated and satisfied parts. The complete CTL specification is thus preserved in the union of the satisfied and untreated sets. Therefore if the algorithm terminates in a configuration containing no untreated formulas, $[(E_t, U_t, A_t), (\emptyset, \emptyset, \emptyset), R]$, its rule set $R$ satisfies the specification $(E_t, U_t, A_t)$ that is equivalent to the original specification.

The termination of the algorithm is proved by considering the lexicographic ordering over the couple $< a, n >$ where $a$ is the number of untreated ACTL formulae, and $n$ is the number of untreated ECTL and UCTL formulae. The transitions **E**, **E'**, **U**, **U'**, and **U"** do not change $a$ but strictly decrease $n$. The transition **A** strictly decreases $a$ (and does not change $n$) while the transition **A'** strictly decreases $a$. Hence, the complexity measure strictly decreases at each transition, and, as $n$ is bounded by the initial number of ECTL and UCTL formulae, there is at most $a * n$ transitions. □

This algorithm thus finitely enumerates model revisions that satisfy a given specification. However, this algorithm is incomplete for two reasons. First, the satisfaction of an ECTL or an UCTL formula is searched by adding only one rule to the model (transition E' and U'), whereas several rules might be needed. If this is the case for all ECTL and UCTL formulae, the algorithm terminates in a failed configuration containing unsatisfied formulae. Second, in the Kripke structure associated to the BIOCHAM model, the accessibility relation is completed into a total relation (which is necessary for the definition of Kripke structures). When a rule is deleted, another rule (loop) may thus be automatically added, thereby invalidating the assumptions of proposition 1.

This algorithm is therefore a heuristic algorithm which may fail in cases where the CTL specification is satisfiable. It is nevertheless usable in practice as illustrated in Sect. 6, and has been actually designed from our practical experience.

The search for model revisions is obviously a very computation intensive process that can be controlled here with the patterns given for the rules allowed to be added or deleted. These patterns limit the branching factor of the search tree explored by the algorithm, while the depth is bounded by the product of the number of ACTL formulae and the number of other formulae in the specification.

## 5 Learning Parameter Values from Constraint LTL Properties

In the same manner as for CTL, one can consider the LTL formulae in the concentration semantics as a specification of the expected behavior of a numerical model. Using constraint model-checking techniques, this specification can be checked for correctness. When it is not satisfied, a search algorithm can be used to revise the parameter values.

### 5.1 Constraint LTL Model-Checking Algorithm

Under the hypothesis that the initial state is completely defined, numerical integration methods (such as Runge-Kutta or Rosenbrock methods) provide a discrete simulation trace. This trace constitutes a linear Kripke structure in which LTL formulae can be interpreted. Since constraints refer not only to concentrations, but also to their derivatives, we consider traces of the form

$$(< t_0, x_0, dx_0/dt >, < t_1, x_1, dx_1/dt >, ...)$$

At each time point, $t_i$, the trace associates the concentration values of the $x_i$'s and the values of their derivatives $dx_i/dt$. It is worth noting that in variable step size integration methods, the step size $t_{i+1} - t_i$ is not constant and is computed following an estimation of the error made by the discretization.

To verify a temporal property $\phi$ within a finite time horizon, one can thus use the following trace-based constraint model-checking algorithm:

1. compute a finite simulation trace;
2. label each trace point by the atomic sub-formulae of $\phi$ that are true at this point;
3. add sub-formulae of the form $F\phi$ (resp. $X\phi$) to the predecessors (resp. immediate predecessor) of a point labeled with $\phi$;
4. add sub-formulae of the form $\phi_1\ U\ \phi_2$ to the points preceding a point labeled with $\phi_2$ as long as $\phi_1$ holds;
5. add sub-formulae of the form $G\phi$ to the last state if it is labeled by $\phi$, and to the predecessors of the points labeled by $G\phi$ as long as $\phi$ holds.

Being limited to finite simulation traces, and since $G\phi = !F(!\phi)$, we choose to label by $G\phi$ the last state if it satisfies $\phi$.

The rationale of this algorithm is that the numerical trace contains enough relevant points, and in particular those where the derivative changes abruptly, to correctly evaluate temporal logic formulae. This has been very well verified in practice with various examples of published mathematical models.

### 5.2 Parameter Search

One can use constraint LTL model-checking to design a *generate and test* algorithm for finding parameter values such that a given LTL specification is satisfied.

A set of parameters, together with intervals of possible values and a precision parameter, are input to an enumeration algorithm. All value combinations are then scanned with a step size corresponding to the given precision, until the specification is satisfied. The syntax of the command is:

```
learn_parameter(parameters, ranges, precision, spec_LTL, time)
```

where the last argument is the time horizon of the simulation.

This search procedure actually replicates and automates part of what the modeler currently does by hand: trying different parameter values, between bounds that are thought reasonable, or computed by other methods such as bifurcation diagrams, in order to obtain behaviors in accordance with the experimental knowledge. BIOCHAM provides a way to explore much faster this parameter space, once the effort for formalizing the expected behavior in LTL is done.

The main novel feature of this method is its capability to express and combine in LTL both qualitative and quantitative constraints on the expected behavior of the model. Section 6.5 shows an example where this algorithm already provides interesting solutions for a two-parameter search.

The computational complexity grows linearly in the number of combinations of parameter values to try, that is in $O(d^n)$ where $n$ is the number of parameter values to find and $d$ the number of values to try for each parameter. The difficulty to use better search algorithms than generate-and-test (such as local search or simulated annealing for instance) comes from the criterion of satisfaction of LTL formulae which is naturally boolean and for which a multi-valued measure of satisfaction can hardly be defined.

### 5.3 Constraint PLTL Model-Checking Algorithm

The existing probabilistic model-checking tools, like that of PRISM [35], do not handle well highly non-deterministic examples, nor those where variables have a large domain as it is the case in BIOCHAM's population semantics. This led us to actually consider the PLTL fragment of PCTL formulae in which the $P_{\bowtie_p}$ operator can only appear once as head of the formula, and to use a Monte-Carlo method as done in the APMC system [36].

To evaluate the probability of realization of the underlying LTL formula, BIOCHAM samples a certain number of stochastic simulations using standard algorithms like that of Gillespie [24] or of Gibson [37]. The outer probability is then estimated by counting. It is worth noting that this method provides a real estimate of realization of the LTL formula, whereas PCTL expresses the boolean satisfaction of a probability constraint ($\bowtie_p$) over the formula.

In principle, the Monte-Carlo algorithm can thus be used for model-checking and learning along the same lines as in the concentration semantics. However, both the stochastic simulation process and the model-checking process are computationally more expensive than in the concentration semantics by several orders of magnitude. For this reason, our machine learning approach is currently not practical in the stochastic population semantics.

## 6 Example of the Cell Cycle Control

In this section, we illustrate the use of our machine learning techniques based on temporal logic specifications, through an example of the cell cycle control developed by Qu et al. ([22]).

### 6.1 The Model of Qu et al. Transcribed in BIOCHAM

The model of Qu et al. describes the transitions between the different phases of the cell cycle. The cell cycle of somatic cells is usually composed of two alternate phases: the DNA synthesis (S phase) and the chromosome segregation and mitosis (M phase). These phases are separated by gap phases (G1 and G2) that ensure that one phase is finished before the other one starts.

The model of Qu presents the dynamics of one transition, either G1/S or G2/M, depending on the type of cyclin used in the model, i.e. `CycE` or `CycB` respectively. Here we choose to deal with the G2/M transition and thus with the B-type cyclin.

In this model, the cyclin is continuously synthesized and degraded, according to the following rules:

```
k1                          for _=>CycB.
k2*[CycB]                   for CycB=>_.
k2u*[APC]*[CycB]            for CycB=[APC]=>_.
```

The cyclin associates with a cyclin-dependent kinase called `CDK` to form a complex which plays a major role in the control of the transition.

```
(k3*[CDK]*[CycB],k4*[CDK-CycB~{p1,p2}]) for CDK+CycB<=>CDK-CycB~{p1,p2}.
```

Qu's model details the network of proteins that regulates the activity of the complex. This activity depends on the phosphorylation state of the complex, the presence of an inhibitor and the availability of the cyclin component. For that reason, Qu considered three positive (1-3) and one negative (4) feedback loops :

1. the active form `CycB-CDK~{p1}` is inactivated by the `Wee1` kinase, itself inactivated by `CycB-CDK~{p1}`;

   ```
   [Wee1]*[CycB-CDK~{p1}]    for CycB-CDK~{p1}=[Wee1]=>CycB-CDK~{p1,p2}.
   cw*[CycB-CDK~{p1}]*[Wee1] for Wee1=[CycB-CDK~{p1}]=>Wee1~{p1}.
   ```

2. similarly, the Cdc25 phosphatase (`C25~{p1,p2}`) activates `CycB-CDK~{p1}` which in turn activates Cdc25;

   ```
   k5u*[C25~{p1,p2}]*[CycB-CDK~{p1,p2}]
         for CycB-CDK~{p1,p2}=[C25~{p1,p2}]=>CycB-CDK~{p1}.
   cz*[CycB-CDK~{p1}]*[C25]
         for C25=[CycB-CDK~{p1}]=>C25~{p1}.
   cz*[CycB-CDK~{p1}]*[C25~{p1}]
         for C25~{p1}=[CycB-CDK~{p1}]=>C25~{p1,p2}.
   ```

3. when present, an inhibitor, CKI, binds to the active form of the complex to form an inactive trimer (CKI-CycB-CDK~{p1}) that is recognized for degradation when phosphorylated by CycB-CDK~{p1} itself;

```
(k14*[CKI]*[CycB-CDK~{p1}],k15*[CKI-CycB-CDK~{p1}])
      for CKI+CycB-CDK~{p1}<=>CKI-CycB-CDK~{p1}.
ci*[CycB-CDK~{p1}]*[CKI-CycB-CDK~{p1}]
      for CKI-CycB-CDK~{p1}=[CycB-CDK~{p1}]=>(CKI-CycB-CDK~{p1})~{p2}.
k16*[(CKI-CycB-CDK~{p1})~{p2}]
      for (CKI-CycB-CDK~{p1})~{p2}=>CDK.
```

4. CycB-CDK~{p1} activates APC which is responsible for its degradation.

```
(([CycB-CDK~{p1}]^2)/(a^2+([CycB-CDK~{p1}]^2)))/tho
      for _=[CycB-CDK~{p1}]=>APC.
k7u*[APC]*[CycB-CDK~{p1}] for CycB-CDK~{p1}=[APC]=>CDK.
```

The Appendix A contains the complete transcription of Qu's model, with the list of kinetic parameter values, and with the definition of an initial state. In the initial state, the cyclin-dependent kinase CDK, the Wee1 kinase and the cyclin inhibitor CKI are present with respective concentrations 200, 1 and 1, all the other molecules being absent (concentrations set to 0).

In addition, a BIOCHAM model can contain a formal specification in temporal logic, accounting for the relevant biological properties captured by the model. For instance, the activation of CycB-CDK~{p1}, the oscillatory behavior of the active form of the complex, or the compulsory role of CycB-CDK~{p1} in APC activation are formalized as follows:

```
reachable(CycB-CDK~{p1}).
oscil(CycB-CDK~{p1}).
checkpoint(CycB-CDK~{p1},APC).
```

The command genCTL(CTLpattern) can also be used to automatically generate a specification from the reaction rules. This specification contains, for each compound, all the reachable, oscil, and checkpoint properties (or any other properties specified by a pattern) that are true in the model. Appendix C shows the specification generated from the rules in this example.

### 6.2 Boolean Abstraction

Qu's model was conceived to specifically study numerical aspects of the cell cycle. Reasoning about such models at a boolean level may lead to some problems. For instance, a fast and a slow reaction are treated equally in the boolean semantics whereas their kinetics defines which one is preponderant. A property revealing the difference between these two reactions would then be invalid in the boolean semantics, which can be checked by BIOCHAM. In that case, the model needs to be adapted.

For example, in most organisms, Cdc25 (more precisely here C25~{p1,p2}) is necessary for the activation of the complex CycB-CDK~{p1}. This property

can be checked in the numerical model by blocking `C25~{p1,p2}` (setting to 0 the kinetic parameters of the rules producing this compound). In the numerical model, the activation of `CycB-CDK~{p1}` is done by a (fast) reaction involving the phosphatase `C25~{p1,p2}` and by a background reaction (slower by one order of magnitude). In the boolean semantics however, the property `checkpoint(C25~{p1,p2},CycB-CDK~{p1})` is false since the second pathway does not use `C25~{p1,p2}`.

That property can be enforced by adding it as a specification and by revising the model. The command `revise_model` produces the following output[3]:

```
biocham:add_spec(Ai(AG(!(CycB-CDK~{p1})
                         ->checkpoint(C25~{p1,p2},CycB-CDK~{p1})))).
biocham: revise_model.
Success
Time: 441.00 s
40 properties treated
Modifications found:
  Deletion(s):
k5*[CycB-CDK~{p1,p2}] for CycB-CDK~{p1,p2}=>CycB-CDK~{p1}.
k15*[CKI-CycB-CDK~{p1}] for CKI-CycB-CDK~{p1}=>CKI+CycB-CDK~{p1}.
  Addition(s):
```

The solution found consists in deleting two rules: the first one corresponding to the background activation of the complex `CycB-CDK~{p1}` and the second one corresponding to another pathway using the dissociation of the inactive trimer. With this modification, the revised model satisfies the complete CTL specification.

### 6.3 Rule Inference

The automatic search for rules can also be used to complete an erroneous model. To illustrate this, let us delete the rule of `CDK-CycB~{p1}` activation by `Cdc25`: `CDK-CycB~{p1,p2}=[C25~{p1,p2}]=>CDK-CycB~{p1}`.
and let the system recover the rule from the specification. After deletion, the model does not verify the CTL specification anymore. When asking to revise the model, the system recovers the deleted rule:

```
biocham: revise_model.
Success
Time: 76.00 s
40 properties treated
Modifications found:
  Deletion(s):
  Addition(s):
CycB-CDK~{p1,p2}=[C25~{p1,p2}]=>CycB-CDK~{p1}.
```

[3] The CPU times indicated in this paper have been obtained on a Pentium IV, 1,7GHz, 1GB RAM, under Linux.

When asking more precisely to learn one rule (as described in Sect. 4.2), the system tests 464 rules, in 25 seconds, and returns only one possible answer in this example. To reduce the time of search (and the number of outputs if several are found), one can also make the rule pattern more precise, like `learn_one_addition(dephosphorylation)`, where the dephosphorylation pattern is `$A~? =[?]=> $A`. In that case, fewer rules (80) are tested, and the missing rule is found in less than 5 seconds.

### 6.4 Model Refinement

The automated method described for learning one rule and revising models can also be used in an interactive fashion to refine a model, for instance by adding a molecule to the model.

Let us include the protein `CycE` in Qu's generic cell cycle in order to reproduce the G1/S transition. New properties are formulated and added to the specification. As it is the case for `CycB`, we assume that `CycE` is only active in a complex with another cyclin-dependent kinase, called for our purpose `CDKp`, and the complex can be inactivated by an inhibitor `CKI`:

```
Ai(reachable(CycE-CDKp)),          Ai(oscil(CycE-CDKp)),
Ai(reachable(CKI-CycE-CDKp)),      Ai(oscil((CycE-CDKp)~{p1})),
Ai(reachable((CycE-CDKp)~{p1})), Ai(loop(CycE-CDKp,(CycE-CDKp)~{p1})),
```

Rules for synthesis and degradation of `CycE` and `CDKp` are added. The specification is no longer verified and with `revise_model`, BIOCHAM is asked to propose some rules to correct the model.

```
biocham: revise_model.
Success
Time: 158.00 s
6 properties treated
Modifications found:
  Deletion(s):
  Addition(s):
CDKp+CycE=>CDKp-CycE.
CKI+CDKp-CycE=>CDKp-CKI-CycE.
CDKp-CycE=>(CDKp-CycE)~{p1}.
(CDKp-CycE)~{p1}=>CDKp-CycE.
```

The rules found are quite simple and constitute a first step for refining the model. This first solution found is followed by roughly $30^4$ other solutions corresponding to 30 different ways to achieve the same results for each rule. For instance the inhibition of `(CDKp-CycE)~{p1}` performed by dephosphorylation in the last rule, can be performed as well by other complexation, or degradation rules. One way to restrict the combinatorics of these revisions is thus to refine the rule patterns given for the search. Another way is to augment the specification if more knowledge about the proteins and their activity is available.

### 6.5 Parameter Search

Now to illustrate the parameter search method, let us consider the estimation of the two parameters $k1$ and $k5u$. These parameters, studied in lengthy details in Qu's article, correspond respectively to the parameter for `CycB` synthesis, and to the rate of the `CycB-CDK~{p1}` activation by `C25~{p1,p2}`. When set to 0, the numerical simulation exhibits a stable steady state, as shown in Fig. 1.

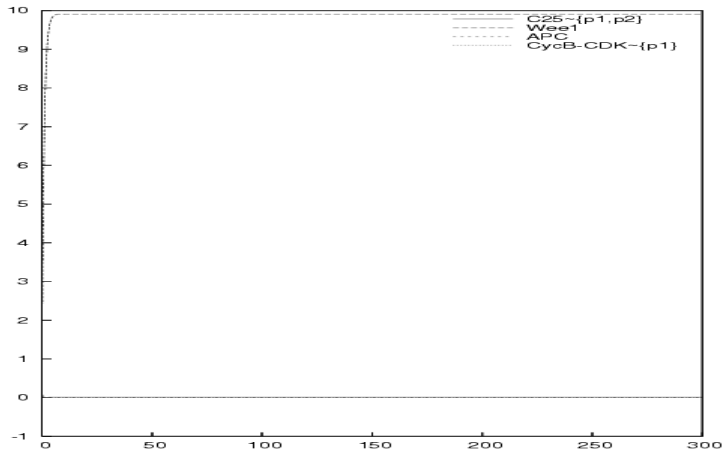


**Fig. 1.** Behavior obtained with parameter values: $k5u$=0 and $k1$=0

In order to find satisfactory values, the desired oscillation property may be expressed as follows: the concentration of the complex `CycB-CDK~{p1}` oscillates at least twice in the time horizon of 300 time units, with peaks above the threshold value 10. Based on its LTL formalization, a search is done for the two parameters $k5u$ and $k1$ on a domain of (0,10) and (0,500) respectively, among 20 different possibilities for each parameter:

```
biocham: learn_parameters([k5u,k1],[(0,10),(0,500)], 20,
                          oscil(CycB-CDK~{p1},2,10.0),300).
First values found that make oscil(CycB-CDK~{p1},2,10.0) true:
parameter(k5u,0.5).
parameter(k1,350).
Search time: 21.54 s
```

BIOCHAM proposes the first solution that satisfies the specification, here $k5u$=0.5 and $k1$=350. The simulation is depicted in Fig. 2.

The specification can be further refined to enforce a period of approximately 65 time units:

```
biocham: learn_parameters([k5u,k1],[(0,10),(0,500)], 20,
```

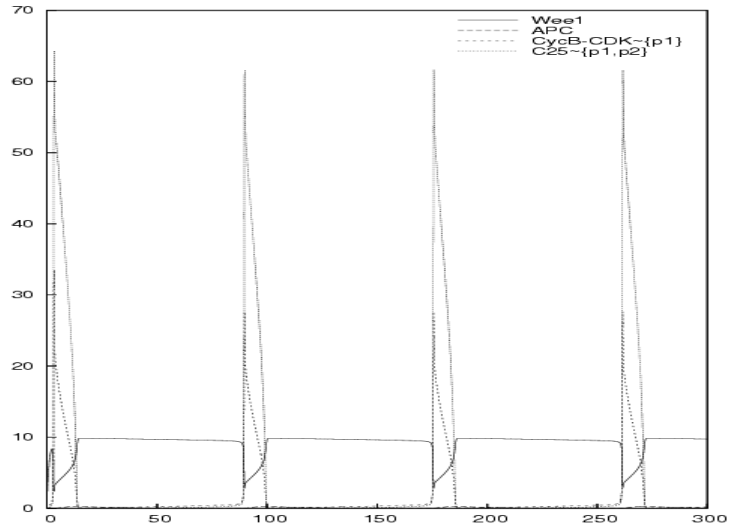


**Fig. 2.** Behavior obtained with parameter values $k5u$=0.5, $k1$=350 found in response to the query `oscil(CycB-CDK~{p1},2,10.0)` over a time horizon of 300 t.u.

```
                        period(CycB-CDK~{p1},65), 300).
First values found that make period(CycB-CDK~{p1},65) true:
parameter(k5u,2).
parameter(k1,150).
Search time: 236.37 s
```

These values produce Fig. 3, and are actually close to the values published by Qu et al ($k5u$=1 and $k1$=300 with an exact period of 67.65 time units). It is worth noting that they have been produced in response to a request containing both qualitative (oscillatory behavior) and quantitative (period and threshold values) aspects, without overspecifying the expected curve of concentration values.

## 7 Conclusion

Temporal logic is a powerful formalism for expressing the biological properties of a living system, such as state reachability, checkpoints, stability, oscillations, etc. This can be done both qualitatively and quantitatively, by considering first-order temporal logic formulae with numerical constraints.

In this paper we have shown how temporal logic can be turned into a specification language of the behavior of a biochemical system, and how model-checking and machine learning techniques can be jointly used to infer reaction rules, or more generally model revisions, in order to satisfy a temporal logic specification. More specifically we have described a model revision algorithm which, given a boolean model and a CTL specification, enumerates a set of solutions in the form

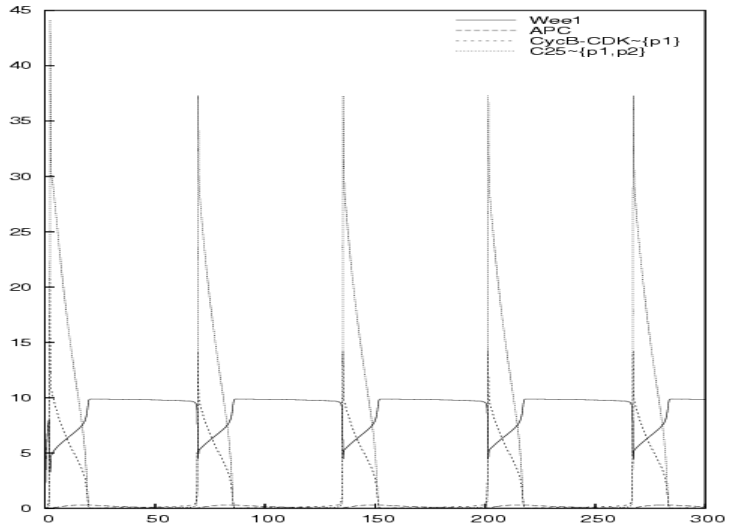


**Fig. 3.** Behavior obtained with parameter values $k5u$=2, $k1$=150 found in response to the query `period(CycB-CDK~{p1},65)` over a time horizon of 300 t.u.

of sets of rules to add to, or delete from, the model in order to satisfy the specification. The depth of the search tree explored by this algorithm is bounded by $a*n$ where $a$ is the number of ACTL formulae and $n$ is the number of other CTL formulae in the specification. The source of incompleteness of this algorithm has been analyzed, leaving place for further improvements.

Similarly, we have presented an algorithm which, given a numerical model, a value range for some parameters and an LTL specification with numerical constraints over concentrations, searches for parameter values satisfying the specification.

These algorithms have been illustrated with different scenarios of specification, rule inference, model refinement and parameter learning in an example of the cell cycle control after Qu et al. [22].

These first results implemented in BIOCHAM are quite encouraging and motivate us to go further in the direction of the formal specification of biological systems and in the improvement of the search algorithms. They also show some limitations concerning the boolean abstraction level used to reason about biochemical systems. There are many ways to define a boolean model from a numerical model. The boolean abstraction currently used in BIOCHAM is a strong abstraction as it simply forgets the kinetic expressions, and considers all possible transitions in an equal setting. Less crude abstractions are however possible and would be technically more effective. We are currently investigating these abstractions, as well as their formal relationship, in the framework of abstract interpretation.

**Acknowledgments**

This work has been partly supported by the European STREP project APRIL II[4]. We are especially grateful to Stephen Muggleton, Luc de Raedt, David Gilbert, Vlad Vyshemirsky and Monika Heiner for fruitful discussions.

## References

1. Regev, A., Silverman, W., Shapiro, E.Y.: Representation and simulation of biochemical processes using the pi-calculus process algebra. In: Proceedings of the sixth Pacific Symposium of Biocomputing. (2001) 459–470
2. Cardelli, L.: Brane calculi - interactions of biological membranes. In Danos, V., Schächter, V., eds.: CMSB'04: Proceedings of the second Workshop on Computational Methods in Systems Biology. Volume 3082 of Lecture Notes in BioInformatics., Springer-Verlag (2004) 257–280
3. Regev, A., Panina, E.M., Silverman, W., Cardelli, L., Shapiro, E.: Bioambients: An abstraction for biological compartments. Theoretical Computer Science **325** (2004) 141–167
4. Danos, V., Laneve, C.: Formal molecular biology. Theoretical Computer Science **325** (2004) 69–110
5. Phillips, A., Cardelli, L.: A correct abstract machine for the stochastic pi-calculus. Transactions on Computational Systems Biology (to appear) Special issue of BioConcur 2004.
6. Eker, S., Knapp, M., Laderoute, K., Lincoln, P., Meseguer, J., Sönmez, M.K.: Pathway logic: Symbolic analysis of biological signaling. In: Proceedings of the seventh Pacific Symposium on Biocomputing. (2002) 400–412
7. Chabrier, N., Fages, F.: Symbolic model cheking of biochemical networks. In Priami, C., ed.: CMSB'03: Proceedings of the first Workshop on Computational Methods in Systems Biology. Volume 2602 of Lecture Notes in Computer Science., Rovereto, Italy, Springer-Verlag (2003) 149–162
8. Bernot, G., Comet, J.P., Richard, A., Guespin, J.: A fruitful application of formal methods to biological regulatory networks: Extending thomas' asynchronous logical approach with temporal logic. Journal of Theoretical Biology **229** (2004) 339–347
9. Batt, G., Bergamini, D., de Jong, H., Garavel, H., Mateescu, R.: Model checking genetic regulatory networks using gna and cadp. In: Proceedings of the 11th International SPIN Workshop on Model Checking of Software SPIN'2004, Barcelona, Spain (2004)
10. Calder, M., Vyshemirsky, V., Gilbert, D., Orton, R.: Analysis of signalling pathways using the prism model checker. In Plotkin, G., ed.: CMSB'05: Proceedings of the third Workshop on Computational Methods in Systems Biology. (2005)
11. Antoniotti, M., Policriti, A., Ugel, N., Mishra, B.: Model building and model checking for biochemical processes. Cell Biochemistry and Biophysics **38** (2003) 271–286
12. Calzone, L., Chabrier-Rivier, N., Fages, F., Soliman, S.: A machine learning approach to biochemical reaction rules discovery. In III, F.J.D., ed.: Proceedings of Foundations of Systems Biology and Engineering FOSBE'05, Santa Barbara (2005) 375–379

[4] `http://www.aprill.org`

13. Fages, F., Soliman, S., Chabrier-Rivier, N.: Modelling and querying interaction networks in the biochemical abstract machine BIOCHAM. Journal of Biological Physics and Chemistry **4** (2004) 64–73
14. Chabrier, N., Fages, F., Soliman, S.: BIOCHAM's user manual. INRIA. (2003–2006)
15. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
16. Nagasaki, M., Onami, S., Miyano, S., Kitano, H.: Bio-calculus: Its concept and molecular interaction. In: Proceedings of the Workshop on Genome Informatics. Volume 10. (1999) 133–143
17. Nagasaki, M., Onami, S., Miyano, S., Kitano, H.: Bio-calculus: Its concept, and an application for molecular interaction. In: Currents in Computational Molecular Biology. Volume 30 of Frontiers Science Series. Universal Academy Press, Inc. (2000) This book is a collection of poster papers presented at the RECOMB 2000 Poster Session.
18. Muggleton, S.H.: Inverse entailment and progol. New Generation Computing **13** (1995) 245–286
19. Bryant, C.H., Muggleton, S.H., Oliver, S.G., Kell, D.B., Reiser, P.G.K., King, R.D.: Combining inductive logic programming, active learning and robotics to discover the function of genes. Electronic Transactions in Artificial Intelligence **6** (2001)
20. Angelopoulos, N., Muggleton, S.H.: Machine learning metabolic pathway descriptions using a probabilistic relational representation. Electronic Transactions in Artificial Intelligence **7** (2002) also in Proceedings of Machine Intelligence 19.
21. Angelopoulos, N., Muggleton, S.H.: Slps for probabilistic pathways: Modeling and parameter estimation. Technical Report TR 2002/12, Department of Computing, Imperial College, London, UK (2002)
22. Qu, Z., MacLellan, W.R., Weiss, J.N.: Dynamics of the cell cycle: checkpoints, sizers, and timers. Biophysics Journal **85** (2003) 3600–3611
23. Chabrier-Rivier, N., Fages, F., Soliman, S.: The biochemical abstract machine BIOCHAM. In Danos, V., Schächter, V., eds.: CMSB'04: Proceedings of the second Workshop on Computational Methods in Systems Biology. Volume 3082 of Lecture Notes in BioInformatics., Springer-Verlag (2004) 172–191
24. Gillespie, D.T.: General method for numerically simulating stochastic time evolution of coupled chemical-reactions. Journal of Computational Physics **22** (1976) 403–434
25. Gibson, M.A., Bruck, J.: A probabilistic model of a prokaryotic gene and its regulation. In Bolouri, H., Bower, J., eds.: Computational Methods in Molecular Biology: From Genotype to Phenotype. MIT press (2000)
26. Chabrier-Rivier, N., Chiaverini, M., Danos, V., Fages, F., Schächter, V.: Modeling and querying biochemical interaction networks. Theoretical Computer Science **325** (2004) 25–44
27. Batt, G.: Validation de modèles qualitatifs de réseaux de régulation génique : une méthode basée sur des techniques de vérication formelle. PhD thesis, Université Joseph Fourier - Grenoble I (2006)
28. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing **6** (1994) 512–535
29. Cimatti, A., Clarke, E., Enrico Giunchiglia, F.G., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: Nusmv 2: An opensource tool for symbolic model checking. In: Proceedings of the International Conference on Computer-Aided Verification, CAV'02, Copenhagen, Danmark (2002)
30. Kohn, K.W.: Molecular interaction map of the mammalian cell cycle control and DNA repair systems. Molecular Biology of the Cell **10** (1999) 2703–2734

31. Schoeberl, B., Eichler-Jonsson, C., Gilles, E., Muller, G.: Computational modeling of the dynamics of the map kinase cascade activated by surface and internalized egf receptors. Nature Biotechnology **20** (2002) 370–375
32. Wang, D., Clarke, E.M., Zhu, Y., Kukula, J.: Using cutwidth to improve symbolic simulation and boolean satisfiability. In: IEEE International High Level Design Validation and Test Workshop 2001 (HLDVT 2001). (2001) 6
33. Berman, C.L.: Circuit width, register allocation, and reduced function graphs. Research Report RC 14127, IBM (1988)
34. Murata, T.: Petri nets: properties, analysis and applications. Proceedings of the IEEE **77** (1989) 541–579
35. Kwiatkowska, M.Z., Norman, G., Parker, D.: Prism 2.0: A tool for probabilistic model checking. In: st International Conference on Quantitative Evaluation of Systems (QEST 2004), IEEE Computer Society (2004) 322–323
36. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Proceedings of the 5th Verification, Model Checking and Abstract Interpretation (VMCAI 2004). Volume 2937 of Lecture Notes in Computer Science., Springer-Verlag (2004) 73–84
37. Gibson, M.A., Bruck, J.: Efficient exact stochastic simulation of chemical systems with many species and many channels. Journal of Physical Chemistry **104** (2000) 1876–1889

## A BIOCHAM Model of the cell cycle control after [22]

```
% Cell cycle Qu et al. 2003 Biophysical journal

% Cyclin
k1                              for _=>CycB.
k2*[CycB]                       for CycB=>_.
k2u*[APC]*[CycB]                for CycB=[APC]=>_.
(k3*[CDK]*[CycB],k4*[CycB-CDK~{p1,p2}])
                                for CDK+CycB<=>CycB-CDK~{p1,p2}.
k5*[CycB-CDK~{p1,p2}]           for CycB-CDK~{p1,p2}=>CycB-CDK~{p1}.
k5u*[C25~{p1,p2}]*[CycB-CDK~{p1,p2}]
                          for CycB-CDK~{p1,p2}=[C25~{p1,p2}]=>CycB-CDK~{p1}.
k6*[CycB-CDK~{p1}]              for CycB-CDK~{p1}=>CycB-CDK~{p1,p2}.
[Wee1]*[CycB-CDK~{p1}]          for CycB-CDK~{p1}=[Wee1]=>CycB-CDK~{p1,p2}.
k7*[CycB-CDK~{p1}]              for CycB-CDK~{p1}=>CDK.
k7u*[APC]*[CycB-CDK~{p1}]       for CycB-CDK~{p1}=[APC]=>CDK.

% Cdc25
k8                              for _=>C25.
k9*[C25]                        for C25=>_.
k9*[C25~{p1}]                   for C25~{p1}=>_.
k9*[C25~{p1,p2}]                for C25~{p1,p2}=>_.
bz*[C25]                        for C25=>C25~{p1}.
cz*[CycB-CDK~{p1}]*[C25]        for C25=[CycB-CDK~{p1}]=>C25~{p1}.
az*[C25~{p1}]                   for C25~{p1}=>C25.
bz*[C25~{p1}]                   for C25~{p1}=>C25~{p1,p2}.
cz*[CycB-CDK~{p1}]*[C25~{p1}]   for C25~{p1}=[CycB-CDK~{p1}]=>C25~{p1,p2}.
```

```
az*[C25~{p1,p2}]              for C25~{p1,p2}=>C25~{p1}.

% Wee1
k10                           for _=>Wee1.
k11*[Wee1]                    for Wee1=>_.
k11*[Wee1~{p1}]               for Wee1~{p1}=>_.
bw*[Wee1]                     for Wee1=>Wee1~{p1}.
cw*[CycB-CDK~{p1}]*[Wee1]     for Wee1=[CycB-CDK~{p1}]=>Wee1~{p1}.
aw*[Wee1~{p1}]                for Wee1~{p1}=>Wee1.

% APC
(([CycB-CDK~{p1}]^2)/(a^2+([CycB-CDK~{p1}]^2)))/tho
                              for _=[CycB-CDK~{p1}]=>APC.
[APC]/tho                     for APC=>_.

% CKI
k12                           for _=>CKI.
k13*[CKI]                     for CKI=>_.
(k14*[CKI]*[CycB-CDK~{p1}],k15*[CKI-CycB-CDK~{p1}])
                              for CKI+CycB-CDK~{p1}<=>CKI-CycB-CDK~{p1}.
bi*[CKI-CycB-CDK~{p1}]    for CKI-CycB-CDK~{p1}=>(CKI-CycB-CDK~{p1})~{p2}.
ci*[CycB-CDK~{p1}]*[CKI-CycB-CDK~{p1}]
          for CKI-CycB-CDK~{p1}=[CycB-CDK~{p1}]=>(CKI-CycB-CDK~{p1})~{p2}.
ai*[(CKI-CycB-CDK~{p1})~{p2}]
                           for (CKI-CycB-CDK~{p1}){p2}=>CKI-CycB-CDK~{p1}.
k16*[(CKI-CycB-CDK~{p1})~{p2}]
                              for (CKI-CycB-CDK~{p1})~{p2}=>CDK.
k16u*[APC]*[(CKI-CycB-CDK~{p1})~{p2}]
                              for (CKI-CycB-CDK~{p1})~{p2}=[APC]=>CDK.

parameter(k1,300).  parameter(k5u,1).
parameter(k2,5).    parameter(k3,0.15). parameter(k4,30).
parameter(k5,0.1).  parameter(k6,1).    parameter(k7,10).
parameter(k8,100).  parameter(k9,1).    parameter(k10,10).
parameter(k11,1).   parameter(k12,10).  parameter(k13,1).
parameter(k14,1).   parameter(k15,1).   parameter(k16,2).
parameter(k2u,50).  parameter(k7u,0).   parameter(k16u,25).
parameter(a,4).     parameter(tho,25).  parameter(az,10).
parameter(aw,10).   parameter(ai,10).   parameter(bz,0.1).
parameter(bw,0.1).  parameter(bi,0.1).  parameter(cz,1).
parameter(cw,1).    parameter(ci,1).

% Initial state
present(CDK,200).   present(Wee1,1).   present(CKI,1).
make_absent_not_present.
```

## B Graphical View of the Model

The Fig. 4 shows a bipartite graph representation of Qu's model, where the reaction rules are in rectangular boxes, the molecules are in circular boxes, the hard lines materialize the reactants and products of the reactions, and the dashed lines materialize the catalysts.

## C CTL Specification

```
% Simple specification of Qu's model generated by genCTL
add_specs({
Ei(reachable(CycB)),
Ei(reachable(!(CycB))),
Ai(oscil(CycB)),

Ei(reachable(APC)),
Ei(reachable(!(APC))),
Ai(oscil(APC)),
Ai(AG(!(APC)->checkpoint(CycB-CDK~{p1},APC))),

Ei(reachable(CDK)),
Ei(reachable(!(CDK))),
Ai(oscil(CDK)),

Ei(reachable(CycB-CDK~{p1,p2})),
Ei(reachable(!(CycB-CDK~{p1,p2}))),
Ai(oscil(CycB-CDK~{p1,p2})),
Ei(reachable(CycB-CDK~{p1})),
Ei(reachable(!(CycB-CDK~{p1}))),
Ai(oscil(CycB-CDK~{p1})),

Ei(reachable(C25~{p1,p2})),
Ei(reachable(!(C25~{p1,p2}))),
Ai(oscil(C25~{p1,p2})),
Ai(AG(!(C25~{p1,p2})
    ->checkpoint(C25~{p1},C25~{p1,p2}))),
Ei(reachable(C25)),
Ei(reachable(!(C25))),
Ai(oscil(C25)),
Ei(reachable(C25~{p1})),
Ei(reachable(!(C25~{p1}))),
Ai(oscil(C25~{p1})),

Ei(reachable(Wee1)),
Ei(reachable(!(Wee1))),
Ai(oscil(Wee1)),
Ei(reachable(Wee1~{p1})),
Ei(reachable(!(Wee1~{p1}))),
Ai(oscil(Wee1~{p1})),
```

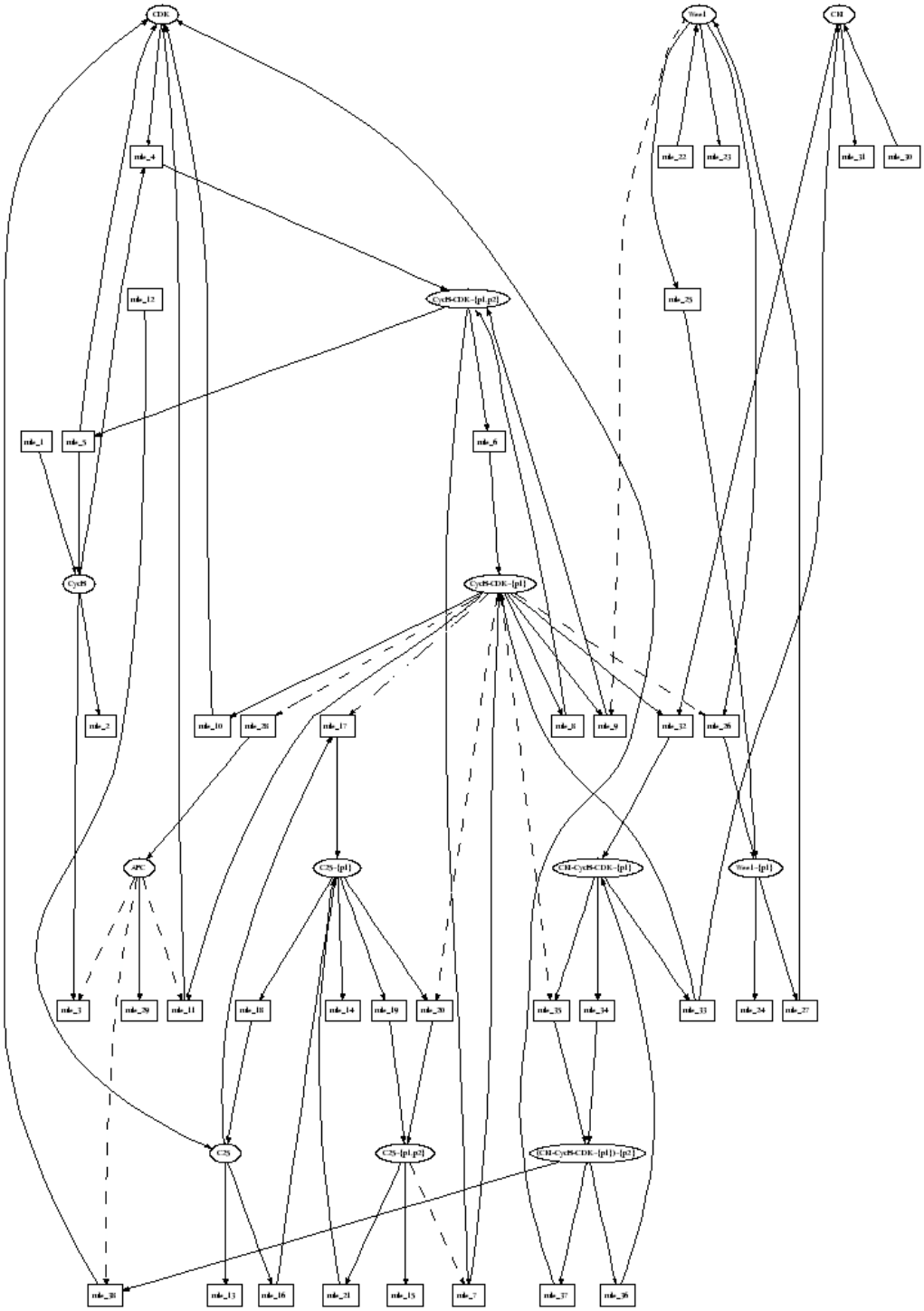


**Fig. 4.** Graphical view of Qu's model.

```
Ai(AG(!(Wee1~{p1})->checkpoint(Wee1,Wee1~{p1}))),

Ei(reachable(CKI)),
Ei(reachable(!(CKI))),
Ai(oscil(CKI)),
Ei(reachable(CKI-CycB-CDK~{p1})),
Ei(reachable(!(CKI-CycB-CDK~{p1}))),
Ai(oscil(CKI-CycB-CDK~{p1})),
Ei(reachable((CKI-CycB-CDK~{p1})~{p2})),
Ei(reachable(!((CKI-CycB-CDK~{p1})~{p2}))),
Ai(oscil((CKI-CycB-CDK~{p1})~{p2})),
Ai(AG(!((CKI-CycB-CDK~{p1})~{p2})
    ->checkpoint(CKI-CycB-CDK~{p1},CKI-CycB-CDK~{p1})~{p2}))}).
```