Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 232 (2009) 39–53
www.elsevier.com/locate/entcs

Studying Irreversible Transitions in a Model of Cell Cycle Regulation

Paolo Ballarini [a,1,2], Tommaso Mazza [a,1,2], Alida Palmisano [a,1,2] and Attila Csikasz-Nagy [a,1,2]

[a] *The Microsoft Research - University of Trento Centre for Computational and Systems Biology Piazza Manci 17, 38100 Trento, Italy*

Abstract

Cells life follows a cycling behaviour which starts at cell birth and leads to cell division through a number of distinct phases. The transitions through the various cell cycle phases are controlled by a complex network of signalling pathways. Many cell cycle transitions are irreversible: once they are started they must reach completion. In this study we investigate the existence of conditions which lead to cases when irreversibility may be broken. Specifically, we characterise the elements of the cell cycle signalling network that are responsible for the irreversibility and we determine conditions for which the irreversible transitions may become reversible. We illustrate our results through a formal approach in which stochastic simulation analysis and model checking verification are combined. Through probabilistic model checking we provide a quantitative measure for the probability of irreversibility in the "Start" transition of the cell cycle.

Keywords: Budding Yeast, Cell Cycle, Probabilistic Model Checking, BlenX, Stochastic Simulation.

1 Introduction

The cell division cycle is a coordinated set of processes by which a cell replicates all its components and divides into two nearly identical daughter cells. The eukaryotic cell cycle is driven by an underlying molecular network which centers around complexes of cyclin dependent kinases (Cdk's) and their regulatory cyclin partners. Active Cdk/cyclin (CycB) complexes can induce critical cell cycle processes by phosphorylating target molecules [16]. The activity of this complex can be regulated in several ways, one of which is the controlled degradation of the cyclin subunit. The Anaphase Promoting Complex (APC) with help from the regulatory protein Cdh1 labels cyclins for degradation at the end of the cell cycle. Interestingly, Cdk/CycB

[1] The authors would like to thank Ivan Mura for comments and discussions about stochastic modeling.
[2] Email: {ballarini,mazza,palmisano,csikasz}@cosbi.eu

doi:10.1016/j.entcs.2009.02.049

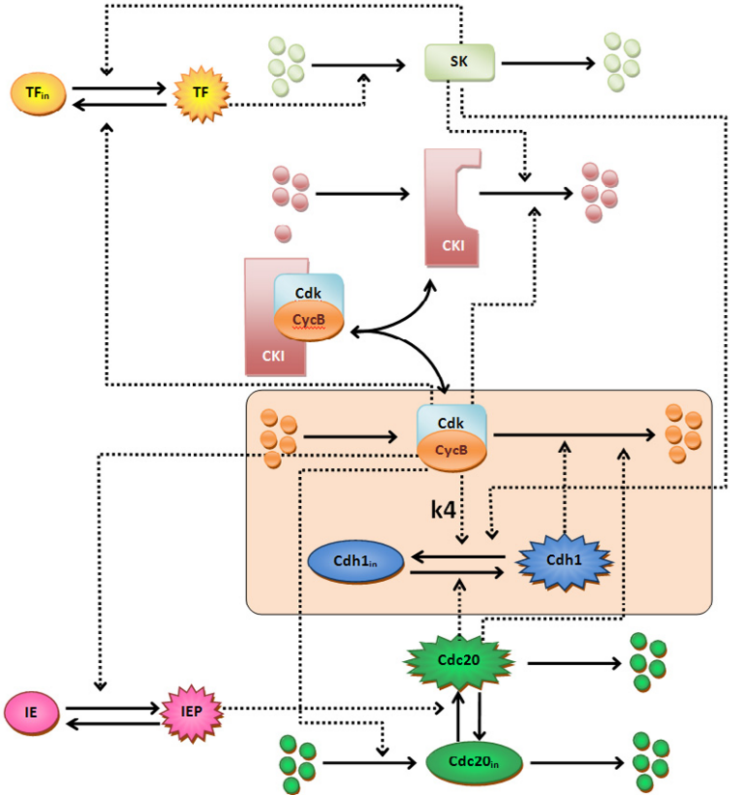


Fig. 1. **Simple budding yeast cell cycle model.** Solid arrows represent transitions, dashed arrows stand for regulatory interactions. The core Cdk/CycB - Cdh1/APC module is highlighted. We also point to the reaction rate which we change during the analysis and that is responsible for the irreversibility of the cycle.

complex can phosphorylate and inactivate Cdh1 proteins, which leads to an antagonistic relation between Cdk/CycB and Cdh1/APC [23].

After cell division the newborn cells are in G1 phase, with high Cdh1 activity, and no Cdk/CycB complex present. As the cell grows it starts to produce cyclin that binds to Cdk and this complex phosphorylates Cdh1. As cell growth proceeds eventually enough Cdk/CycB complexes are produced to inactivate Cdh1. This leads to a further increase in Cdk/CycB level, since cyclin degradation is slowed down after Cdh1 got inactivated. This increased Cdk/CycB activity can induce DNA replication, thus the cells enter into S-phase. This event, when Cdk/CycB activity abruptly increases and the cells enter into S-phase is called "Start" transition of the cell cycle [17].

Mathematical models have been investigating the dynamics of the interactions that drive this transition [4,19]. It has been proposed that the positive feedback loop that is the result of this antagonism between Cdk/CycB and Cdh1/APC (also

called double negative feedback, since the two negative effects bring together a positive loop) can create bistability and hysteresis in the system [4]. Furthermore it has been proposed that this simple module can be responsible for the irreversibility of the Start transition of the cell cycle [20]. Experiments proved the existence of bistability in the cell cycle of budding yeast cells [6], but the irreversibility of this transition was never tested yet.

In this paper we present an analysis of this module by application of both probabilistic model checking and stochastic simulations on a simple budding yeast cell cycle model. More specifically we show that irreversibility can be removed from the system by weakening the positive feedback loop. The remainder of the paper is organised as follows: we first briefly describe a model of the budding yeast cell cycle (Sec. 2) concentrating, in Sec 2.1, on one of its basic mechanism, namely the Cdk/CycB-Cdh1/APC interaction, which is what we focus our irreversibility study on. In Section 3 we present probabilistic model checking and describe a quantitative analysis of the stochastic model obtained through verification of probabilistic logical queries. In Section 4 we discuss results obtained through stochastic simulation of the detailed model of the cell cycle that is containing the core mechanism analysed in the previous sections. We summarise our contribution in the conclusive section.

2 A model of budding yeast cell cycle regulation

During normal cell cycles of the budding yeast the two stable states of the presented bistable switch (G1 with low Cdk/CycB and high Cdh1/APC activity; S/G2/M, with high Cdk/CycB and low Cdh1/APC activity) are alternating. There are several assisting molecules that help the switch to turn back and forth between these stable states [3]. A starter kinase (SK) helps Cdk/CycB to turn off Cdh1/APC and to remove CKI (another inhibitor of Cdk/CycB) at the G1/S transition. On the other hand Cdc20 helps to reactivate Cdh1/APC and remove Cdk/CycB activity at the end of the cell cycle (see Figure 1). In addition there are further regulators (transcription factors (TF), intermediary enzymes (IE), etc ...) that play roles in robust cell cycle regulation.

2.1 Cdk/CycB - Cdh1/APC interactions in the core of cell cycle regulation

At the core of the cell cycle regulation stands the antagonistic interaction between Cdk/CycB and Cdh1/APC (Fig 2). This switch makes the decisions on commitment to start the cell cycle [3]. The wiring diagram of Figure 2 was turned to ODE's by Novak and Tyson [12]. The dynamic of the real valued variables X and Y (representing the concentration of Cdk/CycB and Cdh1/APC complexes respectively) are described by the following linear ordinary differential equations, taken from [12]:

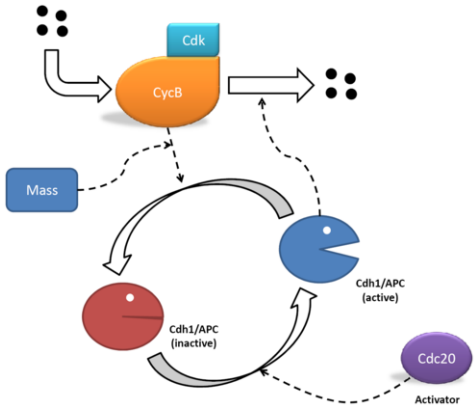


Fig. 2. **Antagonism between Cdk/CycB complex and the Cdh1/APC.** Cdh1/APC (denoted Y in equation (2)) induces the degradation of cyclin while Cdk/CycB (denoted X in equation (1)) inactivated Cdh1/APC by phosphorylation. Increase in cell mass helps to concentrate Cdk/CycB into the nucleus, where it acts on Cdh1/APC, thus following [12] we assume that increase in cell size elevates Cdk/CycB's efficiency to inactivate Cdh1/APC.

$$\frac{dX}{dt} = k_1 - (k_2' + k_2'' \cdot Y)X \tag{1}$$

$$\frac{dY}{dt} = \frac{(k_3' + k_3'' \cdot A)(1-Y)}{J_3 + (1-Y)} - \frac{k_4 \cdot m \cdot X \cdot Y}{J_4 + Y} \tag{2}$$

$$\frac{dm}{dt} = \mu \cdot m \left(1 - \frac{m}{m_*}\right) \tag{3}$$

Note that an unitary (constant) total concentration of Cdh1/APC complex is assumed, thus $(1-Y)$ represents the inactive amount of Cdh1/APC. Furthermore, A indicates the amount of the *activator* (Cdc20) protein which influences activation of Cdh1/APC. For simplicity, in this model, we assume A constant. Finally m denotes the cell mass and its dynamic behaviour is driven by equation (3), where μ is the growing factor and m_* is the maximum level that the mass can reach.

3 Model verification through probabilistic model checking

We apply *model checking* [5] as a means to verify *irreversibility* related properties of the Cdk/CycB-Cdh1/APC module of the cell cycle.

With model checking a (discrete-state) model of a system is developed, and relevant properties are stated in terms of temporal logic formulae. Algorithms exist that take as inputs both the model M and a formula ϕ and return either a positive answer, if ϕ is satisfied by M (denoted $M \models \phi$) or a negative one if that is not the case (denoted $M \not\models \phi$). The peculiarity of model checking is that

```
// Budding yeast CELL-CYCLE

ctmc // stocastic model
const double k4=35;
const int n=4; // power in rate of Cdc20 reaction
const double alpha=0.00302023; //conversion factor from continuous concentrations to discrete
const double mu=0.01; // rate of Mass growth
const int noise_d=2;

// initial population
const int Y_init=42;
const int X_init=1;
const int m_init=50;
const int A=42; //424;

// maximum population
const int Y_max=42;
const int X_max=42;
const int m_max=50;
```

Table 1
PRISM model's constants. Discretized quantities translate to integer constants: *noise_d* represents the desired level of noise and is set to 2, which is roughly 5% of maximum signal level

verification of ϕ against M is achieved through an exhaustive exploration of the model state space, hence the outcome of model verification is *exact*, as opposed to the *approximated* results obtained through model simulation. The obvious downside of model checking, is that, due to the complexity of many real systems, the resulting model dimension blows up to the point that model checking becomes untreatable.

Model checking techniques can be classified according to the type of model they refer to and to the query language they adopt. A broad taxonomy may distinguish between *non-probabilistic* model checking, such as LTL and CTL model checking, and *probabilistic* model checking, such as PCTL [14] and CSL [1,2] model checking. Since the nature of the models we develop in this work is inherently stochastic, we focus on probabilistic/stochastic model checking, which we briefly introduce in the following sections. We first shortly describe the class of stochastic processes we have considered in our modelling effort, namely Continuous Time Markov Chains (CTMCs), pointing out some relevant steps for the derivation of a stochastic model of the cell-cycle regulatory network described in Figure 2.

3.1 On the Markovian model of the Cdk/CycB-Cdh1/APC module of the cell cycle

CTMCs are a well established form of discrete-state stochastic processes largely used for modelling and analysis of many different types of systems. Pratically speaking a CTMC model can be thought of as a graph whose states correspond to variables' value and whose transitions indicate the dynamic of the modeled system. In a CTMC, transitions are labelled with real valued numbers, representing the rate of an exponentially distributed delay (the time consumed by the transition to take place). Once a CTMC model is developed then it can be analysed in several manners. Classical steady-state and transient analysis, provides information about the system evolution, respectively, *in the long run* (steady-state), or with respect to a specific instant of time (transient analysis). If the model is too large, stochastic simulation can be applied to derive relevant statistics. For a detailed description of CMTC models the reader is referred to one the many books, such as, for

```
module cyclinB
_X : [0.. X_max] init X_init;
_X_inc_count : [0.. noise_d] init 0;
_X_dec_count : [0.. noise_d] init 0;
increasing_macro_X : bool init true;
decreasing_macro_X : bool init true;

[synth_X] true & _X_inc_count<noise_d − 1 → (k1/alpha) : _X′=(min(_X + 1, X_max))
                & (_X_inc_count′=min(_X_inc_count + 1, noise_d))
                & (_X_dec_count′=max(_X_dec_count − 1, 0));
[synth_X] true & _X_inc_count{≥}noise_d − 1 → (k1/alpha) : _X′=(min(_X + 1, X_max))
                & (increasing_macro_X′=true)
                & (decreasing_macro_X′=false)
                & (_X_inc_count′=0) & (_X_dec_count′=0);

[selfdeg_X] (_X>0) & _X_dec_count<noise_d − 1 → k2_1 ∗ _X : _X′=(max(_X − 1, 0))
                & (_X_dec_count′=min(_X_dec_count + 1, noise_d))
                & (_X_inc_count′=max(_X_inc_count − 1, 0));
[selfdeg_X] (_X>0) & _X_dec_count{≥}noise_d − 1 → k2_1 ∗ _X : _X′=(max(_X − 1, 0))
                & (_X_dec_count′=0) & (_X_inc_count′=0)
                & (decreasing_macro_X′=true) & (increasing_macro_X′=false);

[deg_X] (_X>0) & _X_dec_count<noise_d − 1 → k2_2 ∗ alpha ∗ _X : _X′=(max(_X − 1, 0))
                & (_X_dec_count′=min(_X_dec_count + 1, noise_d))
                & (_X_inc_count′=max(_X_inc_count − 1, 0));

[deg_X] (_X>0) & _X_dec_count{≥}noise_d − 1 → k2_2 ∗ alpha ∗ _X : _X′=(max(_X − 1, 0))
                & (_X_dec_count′=0) & (_X_inc_count′=0)
                & (decreasing_macro_X′=true) & (increasing_macro_X′=false);

[deactivation_Y] (_X>0) → _X : true;
endmodule
```

Table 2
PRISM code for species X (CycB): noise-free increment/decrement are recorded on boolean flags *increasing_macro_X* and *decreasing_macro_X* according to synthesis (transitions [*synth_X*]) and degradation (transitions [*selfdeg_X*] and [*deg_X*]) of X.

example [22].

Discretization of the continuous model. Starting from the ODEs (1) and (2) we have derived a CTMC model of the Cdk/CycB-Cdh1/APC module which we coded into the PRISM probabilistic model checker [15]. Coding into discrete (finite) states of the continuous quantities X and Y is obtained through a discretization step through which the initial concentrations of X and Y, as described in [12], are turned into discrete number of molecules in the following way: if C_S is the concentration of species S and M_S the number of molecules of species S then $C_S = \alpha \cdot M_S$, where $\alpha = \frac{1}{N_A \cdot V \cdot 10^{-6}}$, where V is the (average) volume of the cell nucleus (which is assumed to be $V = 0.7043188 \cdot 10^{-15}$, corresponding to roughly 1.67% of the initial average cell volume which for the budding yeast is equal to $42fl$) and N_A the Avogadro number. Such conversion, details of which can be found in [18], produces the following (discrete) initial values: $X_{init} = 0$, $Y_{init} = 424$. In order to limit the state-space explosion of the CTMC model, we scaled those initial values by an order of magnitude, hence in our model we consider: $X_{init} = 0$, $Y_{init} = 42$. Reaction rates have been adjusted accordingly through a re-scaled value of α.

Noise sensitive model. We have provided our CTMC model with means to keep track of the signal noise level. Noise can be thought of as a fluctuation, within a given threshold, around the current level of signal (i.e. the level of molecules). Keeping track of signal noise allows to account for "noise-free" variations in the level of molecules of a given species [3]. For that purpose we equipped our CTMC model with a parameter *noise_d* (see Table 1) which is set to the desired level of noise. As a result, a sequence of transitions in the CTMC model is recorded as an actual increasing/decreasing path only when it consists of at least *noise_d* consecutive increasing/decreasing transitions. In practical terms this is achieved by means of a specific coding in the PRISM language. Each module representing the behaviour of a species, say species X, is equipped with a pair of boolean flags, *increasing_macro_X* and *decreasing_macro_X*, which allows us to keep track of incresing, respectively, decreasing trends in species X. Furthermore a pair of *counters* (i.e. *inc_X_count*, *dec_X_count*) are used to determine whether increments/decrements in X exceed the considered noise level (see, for example, a sample of the PRISM code for the CycB module in Table 2). In such way (noise-free) monotonic trends for species X can be straightforwardly captured by means of temporal logic formulae such as (4) and (5). We briefly introduce the basic of the stochastic temporal logic for expressing properties of CTMC models, before discussing formulae (4) and (5) which we used to capture monotonic executions of the model.

3.2 Expressing properties of the cell-cycle in probabilistic temporal logic

The *Continuous Stochastic Logic* (CSL) [1,2] is a language for stating properties referring to CTMC models. A CSL formula is built upon a set of atomic propositions (AP), combined through a number of logical connectives: the classical propositional logic *conjunction* ($\wedge$), *disjunction* ($\vee$) and *negation* ($\neg$) plus two *probabilistic* operators $S_{\trianglelefteq p}$ for *steady-state* formulae and $P_{\trianglelefteq p}$ for *path* formulae (with $\trianglelefteq \in \{\leq, <, >, \geq\}$ and $p \in [0,1]$). If a and b are atomic propositions representing basic properties of a system (for example *"$a \equiv$ the number of Cdk/CycB molecules is n"*, and *"$b \equiv$ the cell mass is below m"*) then the state formula $\phi \equiv a \wedge b$ identifies those states of the model in which both a and b are satisfied. System's dynamics is captured in CSL by means of two basic *path operators*: Next ($X^{\leq t}$) and Until ($U^{\leq t}$). A CSL Next formula, $P_{\leq p}(X^{\leq t} a)$ is satisfied in a state s of a CTMC model if (and only if) the probability of reaching, within t, a successor of state s in which the property a is satisfied, is not greater than p. Similarly, a CSL Until formula, $P_{\leq p}(b\ U^{\leq t}\ a)$ is satisfied in state s if the probability measure of those evolutions starting in s and reaching, within time t, a future a-state through a sequence of b-states is bounded by p (which essentially represents the probability that the number of molecules of Cdk/CycB becoming equal to n without the cell mass exceeding m). The formal syntax of CSL formulae is as follows:

[3] in fact noise fluctuations are not relevant for the sake of our analysis and should be disregarded.

$$\phi := a \mid \top \mid \neg\phi \mid \ \phi \wedge \phi \mid \mathcal{S}_{\trianglelefteq p}(\phi) \mid \mathcal{P}_{\trianglelefteq p}(\varphi)$$
$$\varphi := X^I \ \phi \mid \phi \ U^I \phi$$

where $a \in AP$ is an atomic proposition, $\top$ is the truth value *true*, $\trianglelefteq \in \{<, \leq, >, \geq\}$, $I \subseteq \mathbb{R}_{\geq 0}$ is a non empty (time) interval, $\mathcal{S}$ denotes the *steady-state* operator (i.e. it refers to the steady-state measure of probability) and $\mathcal{P}$ denotes paths' measure of probability.

Relaying on the expressiveness of the CSL language we formulate a number of properties which capture relevant features of the cell-cycle CTMC model. Irreversibility in the Cdk/CycB-Cdh1/APC module of the cell-cycle corresponds to the monotonic trend of both X (Cdk/CycB) and Y (Cdh1/APC). In the initial state X is low ($X = 0$) whereas Y is high ($Y = 42$) , when the cell's mass reaches a certain threshold then Y gets inactivated thus X starts growing. We formally characterise monotonicity of Y and X with the following CSL path formulae:

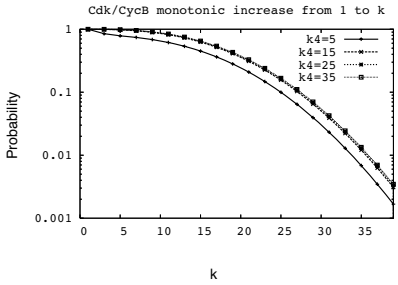
Probability of Monotonic decrease of Cdh1/APC. *"What is the probability that the number of molecules of Cdh1/APC decreases monotonically until the value k is reached?"*

$$P_{=?}[(decreasing\_Y \ U \ (Y = k)] \tag{4}$$

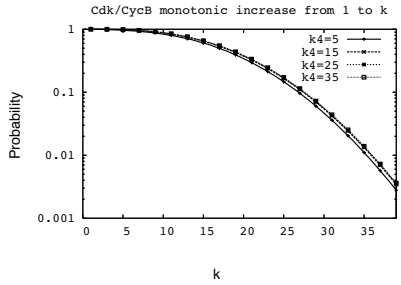
Probability of Monotonic increase of Cdk/CycB. *'What is the probability that the number of molecules of Cdk/CycB increases monotonically until the value k is reached?"*

$$P_{=?}[(increase\_X \ U \ (X = k)] \tag{5}$$

Verification of such formulae through the PRISM tool provides us with a quantification of the likelihood of monotonicity (hence irreversibility) to be maintained (for X and Y) up until the value of k (which can be made varying in $[0, 42]$). Figure 3 and Figure 4 show the results of model checking verification for formulae (4) and (5), respectively, as a function of the reaction rate k_4 (see Figure 1). The original value is $k_4 = 35$. Different boundaries for cell mass growth have also been considered (see plots in Figure 3(a) and Figure 3(b) for the monotonic increase of X and plots in Figure 4(a) and Figure 4(b) for the monotonic decrease of Y). Generally speaking results depicted in Figure 3 and Figure 4 show, as expected, that decreasing the influence of the Cdk/CycB dimer on the inactivation of Cdh1/APC (i.e. through lowering of the rate k_4) decreases the likelihood of a monotonic trend (thus of irreversibility) of both X and Y. Furthermore, by comparing Figure 4 and Figure 3, we observe the existence of a slight asymmetry between the monotonicity of Y and X, which is: it is more likely for X to span upwards the whole interval $[0, 42]$, than for Y to span downwards $[42, 0]$.

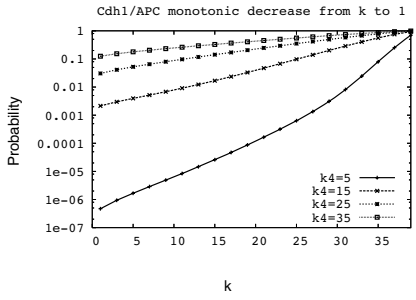


(a) variable cell mass:$M_{init} = 50$,$M_{max} = 150$

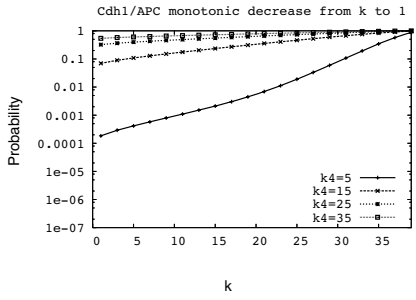


(b) variable cell mass:$M_{init} = 100$,$M_{max} = 200$

Fig. 3. Monotonic increase of Cdk/CycB with cell mass growth.



(a) variable cell mass: $M_{init} = 50$, $M_{max} = 150$



(b) variable cell mass: $M_{init} = 100$, $M_{max} = 200$

Fig. 4. Monotonic decrease of Cdh1/APC with with cell mass growth.

4 Stochastic simulation of the detailed model

In order to "validate" the results obtained through probabilistic model checking we use a more detailed version of the cell cycle regulatory network which we considered in Sec. 3. By means of the *Beta Workbench* [8], a computational tool based on the BlenX modelling language for biological systems, we developed a model of the wild-type network [12] as depicted in Figure 1.

The resulting BlenX code obtained through the translation of the ODEs in [12] is contained in Appendix A. A detailed description of the BlenX language and of the model building procedure is out of the scope of this paper; here we just summarize the sub-set of the BlenX language needed for the understanding of the code of the presented model. We refer the reader to [8,9] for a detailed description of the language and its modeling approach.

The basic metaphor that BlenX relies on is that a *biological entity* (i.e. a component that is able to interact with other components to accomplish some biological

functions) is represented by a *box* in BlenX. A box has an interface (its set of *binders*) and an internal structure that drives its behaviour (see Fig. 5).

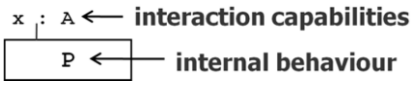


Fig. 5. Boxes as abstractions of biological entities.

For example, in a box modeling a protein, binders may represent *sensing* and *effecting domains*. Sensing domains are the places where the protein receives signals, effecting domains are the places that a protein uses for propagating signals, and the *internal structure* codifies for mechanism that transforms an input signal into a protein conformational change, which can result in the activation or deactivation of another domain.

The exchanging of signals can happen between boxes whose binders have a certain degree of affinity, which codes the strength of their interaction.

The basic primitives of the language that are used to build the model in Appendix A are summarized in graphical form in Fig. 6. Besides the action in the figure, we can specify events of the form: "`when`(*conditions*) *verb*", where the action *verb* is triggered when *conditions* are satisfied. The *verb* can be one among `split`, `new` or `delete`, modeling respectively the substitution, creation, and deletion of boxes in the system (see Fig. 6). Conditions, in the models presented here, are in the form of "*entity_name : : rate_function*", whose meaning is that with the rate *rate_function* the action after the condition is triggered on the entity *entity_name*.

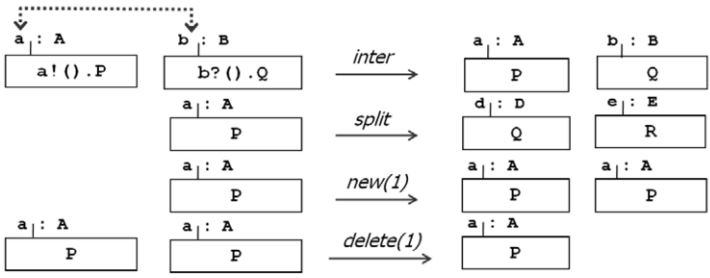


Fig. 6. Intuitive behaviour of some BlenX primitives. Each row represents one of the primitives used in our translation. The first primitive code the interaction between two boxes, through the exchange of an input/output signal (input is in the form of b?() and the output is in the form a!()). The meaning of the last three events is explained in the text.

Rate functions can be declared using real numbers that will be used as base rate for the elementary mass action law, or arbitrary functions (e.g. a sigmoidal response) that are useful when a box represents an aggregated process or when the precise mechanism of interaction between entities is not known.

Using this sub-set of the BlenX language, we were able to built the executable model in Appendix A of the wild-type cell-cycle in Figure 1. On that model we performed Multiple stochastic simulative Replications in Parallel (*MRIP*).

MRIP approaches are frequently used to speed up simulations by working out independent replications of the same stochastic trajectory on multiple computers. Each run is calculated starting from a *seed* chosen among a stream of pseudorandom

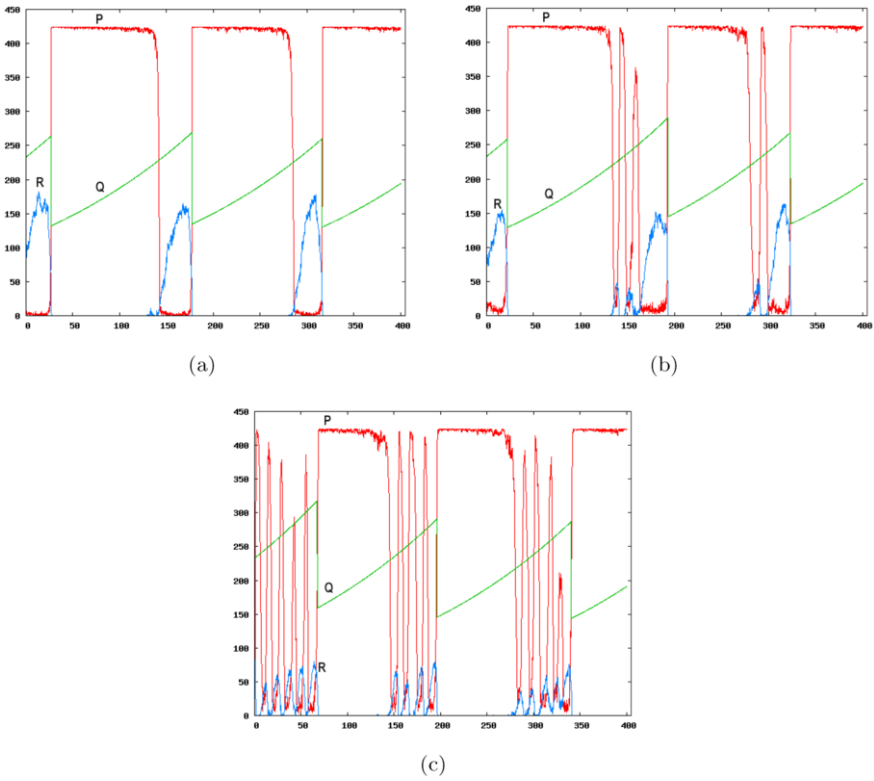


Fig. 7. Sample simulations of the wild type model depicted in Fig. 1. Each of the plot shows 3 curves: (P) Cdh1/APC, (Q) mass, (R) Cdk/CycB. All the simulations have been equally sampled and then 2000 points (a point every 0.2 seconds) have been plotted. The 7(a) plot corresponds to the original model ($k4 = 35$), the 7(b) plot comes from a model with $k4 = 15$ and the 7(c) plot is related to a model with $k4 = 5$.

numbers obtained with the *leap-frog* technique [10] by splitting linear congruential generators. Such a seed guarantees that the resulting trajectories are approximatively uncorrelated. So doing, more observations can be collected during a given time interval than running a single replication on one computer within the same period of time [11,13,7].

We generated 8 identical models except for the kinetic parameter $k4$, that accounts for the inactivation of Cdh1/APC by Cdk/CycB and that we systematically decreased from 35 to 0, step 5. Therefore, to guarantee the trustworthiness and the statistical accuracy of the following analyses, we ran a batch of 100 simulations for each new parameter value to the amount of 800 simulations. In Fig. 7, we show three sample simulations with decreasing $k4$ parameter. The simulations with the original set of parameters (Fig. 7(a)) is reproducing the solutions of the original ODE model in [12], a part from the stochastic noise. At a first glance, it is evident that even a small change of the parameter makes less stable the supposed irreversible Cdh1 decreasing activity (curve P in each graph). Such activity results in quick and sustained oscillations of high amplitude waves.

Moreover, the Fig. 7(c) shows that the more $k4$ is decreased, the less stable is the phase transition, i.e. one obtains an increasing number of oscillations of the

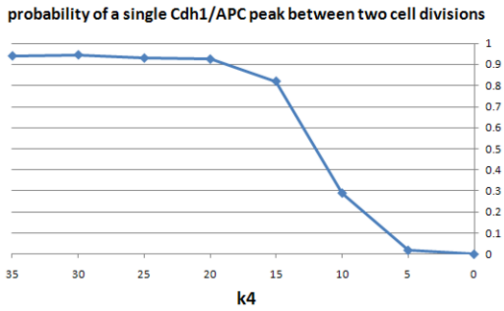


Fig. 8. probability of a single Cdh1/APC peak between two cell divisions

Cdh1 concentration per cell cycle.

In this context, in order to figure out how much sensitive is the irreversibility of the Cdh1/APC inactivation to the parameter of interest, we statistically inspected the 800 simulations results, counting the number of oscillations observed during a complete cycle. The results depicted in Fig. 8 show the percentage of cases with *only one* Cdh1/APC inactivation per cell cycle. They show that the probability of an irreversible behaviour is decreasing, as the parameter is decreased.

5 Conclusion

Our results provide a quantitative measure for the irreversibility of the Start transition of the cell cycle, reached by probabilistic model checking of the Cdk/CycB - Cdh1/APC core module of cell cycle regulation. We show that by weakening the strength of the positive feedback loop (by reducing $k4$) the irreversibility is getting lost. With stochastic simulations of a budding yeast cell cycle model (that includes the auxiliary regulators of this module) we show that indeed the above mentioned parameter variations can perturb the irreversibility of the Start transition of the cell cycle. Future directions include extensions of such methodology (i.e. quantitative analysis of cell cycle irreversibility based on probabilistic model checking verification) to analyze the role that other reactions/rates have in the irreversibility of the cell cycle. A CTMC model of the cell cycle wild-type network of signal (Figure 1) is currently being studied.

References

[1] Aziz, A., K. Sanwal, V. Singhal and R. Brayton, *Model-checking continuous-time Markov chains*, ACM Transactions on Computational Logic **Vol. 1** (2000), pp. pp. 162–170.

[2] Baier, C., B. Haverkort, H. Hermann and J.-P. Katoen, *Model-checking algorithms for continuous-time Markov chains*, IEEE Trans. on Software Eng. **Vol. 29** (2003), pp. pp. 524–541.

[3] Chen, K., L. Calzone, A. Csikasz-Nagy, F. Cross, B. Novak and J. Tyson, *Integrative analysis of cell cycle control in budding yeast*, Mol Biol Cell **15** (2004).

[4] Chen, K., A. Csikasz-Nagy, B. Gyorffy, J. Val, B. Novak and J. Tyson, *Kinetic analysis of a molecular model of the budding yeast cell cycle*, Mol Biol Cell **11** (2000).

[5] Clarke, E., O. Grumberg and D. Peled, "Model Checking," MIT Press, 1999.

[6] Cross, F., V. Archambault, M. Miller and M. Klovstad, *Testing a mathematical model for the yeast cell cycle*, Mol Biol Cell **13** (2002), pp. 52–70.

[7] Dematté, L. and T. Mazza, *On Parallel Stochastic Simulation of Diffusive Systems*, Proceedings of the sixth International Conference on Computational Methods in Systems Biology (CMSB2008) **LNBI 5307** (2008), pp. 191 – 210.

[8] Dematté, L., C. Priami and A. Romanel, *The Beta Workbench: a computational tool to study the dynamics of biological systems*, Briefings in Bioinformatics (2008).

[9] Dematté, L., C. Priami and A. Romanel, "The BlenX Language: a tutorial," Number 5016 in LNCS, Springer-Verlag, 2008 pp. 313–365.

[10] Entacher, K., A. Uhl and S. Wegenkittl, *Linear congruential generators for parallel monte-carlo: the leap-frog case*, Monte Carlo Methods and Applications **4** (1998), pp. 1–16.

[11] Ewing, G., D. McNickle and L. Pawlikowski, *Multiple replications in parallel: Distributed generation of data for speeding up quantitative stochastic simulation*, Proceedings of the 15th Congress of Int. Association for Matemathics and Computer in Simulation (1997).

[12] Fall, P., E. Marland, J. Wagner and J. Tyson, "Computational Cell Biology," Interdisciplinary Applied Mathematics, 1999.

[13] Glynn, P. and P. Heidelberger, *Analysis of parallel replicated simulations under a completion time constraint*, ACM TOMACS **1(1)** (1991), pp. 3–23.

[14] Hansson, H. A. and B. Jonsson, *A framework for reasoning about time and reliability*, in: *Proc. 10th IEEE Real -Time Systems Symposium* (1989), pp. 102–111.

[15] Kwiatkowska, M. Z., G. Norman and D. Parker, *Probabilistic symbolic model checking with prism: A hybrid approach*, in: *TACAS '02: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (2002), pp. 52–66.

[16] Morgan, D., *Principles of cdk regulation*, Nature (1995), pp. 131–134.

[17] Morgan, D., *The cell cycle: Principles of control*, New Science Press (2006).

[18] Mura, I. and A. Csikasz-Nagy, *Stochastic Petri Net Extension of a Yeast Cell Cycle Model*, Journal of Theoretical Biology (2008 (in press)), doi:10.1016/j.jtbi.2008.07.019.

[19] Novak, B., A. C.-N. B. Gyorffy, K. Nasmyth and J. Tyson, *Model scenarios for evolution of the eukaryotic cell cycle*, Phil Trans R Soc Lond B **Vol. 353** (1998), pp. 2063–2076.

[20] Novak, B., J. Tyson, B. Gyorffy and A. Csikasz-Nagy, *Irreversible cell-cycle transitions are due to systems-level feedback*, Nat Cell Biol **9** (2007), pp. 724–728.

[21] Palmisano, A., I. Mura and C. Priami, *From Odes to Language-based, executable models of Biological Systems*, Proceedings of Pacific Symposium on Biocomputing 2009 (PSB 2009) (To appear, 2009).

[22] Stewart, W. J., "Introduction to numerical solution of Markov Chains," Princeton, 1994.

[23] Zachariae, W. and K. Nasmyth, *Whose end is destruction: cell division and the anaphase-promoting complex*, Genes Dev **13** (1999), pp. 2039–2058.

A Appendix: BlenX code for the cell cycle model

BlenX code for the [12] cell cycle model. The model is composed by three files: the first defines the model, the second declares the functions and the third defines the types definitions. Each element is defined through the `let` constructor. In the model file, each protein is represented by a `bproc` and its dynamic behaviour is coded by a series of events (coded with the `when` constructor). The rates of the different reactions are recorded in the function definition file, as constants (`const`), variables (`var`) or more complex mathematical functions (`function`). For a more detailed description of the translation from ODE to BlenX, we refer the reader to [21].

```
//-------------------------------------------------------------------------------------------
//MODEL DEFINITION FILE

[steps = 5000, delta = 0.2]

let CYCBT: bproc = #(x,CYCBT)[ nil ];
when(CYCBT:: d_dtCYCBT_1) new(1);               when(CYCBT:: d_dtCYCBT_2) delete(1);
when(CYCBT:: d_dtCYCBT_3) delete(1);            when(CYCBT:: d_dtCYCBT_4) delete(1);

let CDH1: bproc = #(y,CDH1)[ nil ];             let CDH1_IN : bproc = #(y_in,CDH1_IN) [ nil ];
when(CDH1_IN :: d_dtCDH1_1 ) split(Nil, CDH1);  when(CDH1_IN :: d_dtCDH1_2 ) split(Nil, CDH1);
when(CDH1 :: d_dtCDH1_3 ) split(Nil, CDH1_IN);  when(CDH1 :: d_dtCDH1_4 ) split(Nil, CDH1_IN);

let CDC20_IN : bproc = #(a,CDC20_IN)[ nil ];    let CDC20_A : bproc = #(a,CDC20_A)[ nil ];
when(CDC20_IN :: d_dtCDC20_IN_1 ) new(1);       when(CDC20_IN :: d_dtCDC20_IN_2 ) new(1);
when(CDC20_IN :: d_dtCDC20_IN_5 ) delete(1);
when(CDC20_IN :: d_dtCDC20_IN_4 ) split(Nil, CDC20_A);
when(CDC20_A :: d_dtCDC20_A_2) split(Nil,CDC20_IN);
when(CDC20_A :: d_dtCDC20_A_3) delete(1);

let IEP : bproc = #(y,IEP)[ nil ];              let IEP_IN : bproc = #(y_in,IEP_IN) [ nil ];
when(IEP_IN :: d_dtIEP_1 ) split(Nil, IEP);     when(IEP :: d_dtIEP_2 ) split(Nil, IEP_IN);

let CKIT : bproc = #(x,CKIT )[ nil ];
when(CKIT :: d_dtCKIT_1 ) new(1);               when(CKIT :: d_dtCKIT_2 ) delete(1);
when(CKIT :: d_dtCKIT_3 ) delete(1);            when(CKIT :: d_dtCKIT_4 ) delete(1);

let SK : bproc = #(x,SK)[ nil ];
when(SK :: d_dtSK_2) new(1);                    when(SK :: d_dtSK_3) delete(1);

let TF : bproc = #(y,TF)[ nil ];                let TF_IN : bproc = #(y_in,TF_IN) [ nil ];
when(TF_IN :: d_dtTF_1) split(Nil, TF);         when(TF_IN :: d_dtTF_2) split(Nil, TF);
when(TF :: d_dtTF_3) split(Nil, TF_IN);         when(TF :: d_dtTF_4) split(Nil, TF_IN);

when ( : mCycB -> 0.2, mCycB <- 0.1 :  ) update (m, mass_div);

run 25 CKIT || 97 CYCBT || 39 SK || 5 CDH1 || 419 CDH1_IN || 0 CDC20_A ||
    24 CDC20_IN || 40 IEP || 384 IEP_IN || 15 TF || 409 TF_IN
//-------------------------------------------------------------------------------------------

//-------------------------------------------------------------------------------------------
//FUNCTION DEFINITION FILE

let J15 : const =  0.01;        let J16 : const =  0.01;
let J3 : const =  0.04;         let J4 : const =  0.04;
let J5 : const =  0.3;          let J7 : const =  0.0010;
let J8 : const =  0.0010;       let alpha : const = 0.00236012;
let k1 : const =  0.04;         let k10 : const =  0.02;
let k11 : const =  1.0;         let k12p : const =  0.2;
let k12s : const =  50.0;       let k12t : const =  100.0;
let k13s : const =  1.0;        let k14 : const =  1.0;
let k15p : const =  1.5;        let k15s : const =  0.05;
let k16p : const =  1.0;        let k16s : const =  3.0;
let k2p : const =  0.04;        let k2s : const =  1.0;
let k2t : const =  1.0;         let k3p : const =  1.0;
let k3s : const =  10.0;        let k4 : const =  35.0;
let k4p : const =  2.0;         let k5p : const =  0.005;
let k5s : const =  0.2;         let k6 : const =  0.1;
let k7 : const =  1.0;          let k8 : const =  0.5;
let k9 : const =  0.1;          let keq : const = 1000.0;
let mstar : const = 10.0;       let mu : const =  0.005;
let n : const =  4.0;

let SIGMA : function = alpha*|CYCBT| + alpha*|CKIT|+1/keq;

let alphaDimer : function = alpha * |CYCBT| -
                 ( (2*alpha*|CYCBT|*alpha*|CKIT|)/(SIGMA + sqrt(SIGMA*SIGMA
                 - 4*alpha*|CYCBT|*alpha*|CKIT|)) );

let d_dtCDC20_A_1 : function = (k7*alpha*|IEP|*|CDC20_IN|)/(J7+alpha*|CDC20_IN|);
let d_dtCDC20_A_2 : function = (k8*|CDC20_A|)/(J8+alpha*|CDC20_A|);
let d_dtCDC20_A_3 : function = k6*|CDC20_A|;

let d_dtCDC20_IN_1 : function = k5p/alpha;
let d_dtCDC20_IN_2 : function = (k5s)/(alpha*(1+pow((J5/(m*alphaDimer)),n)));
let d_dtCDC20_IN_3 : function = (k8*|CDC20_A|)/(J8+alpha*|CDC20_A|);
let d_dtCDC20_IN_4 : function = (k7*alpha*|IEP|*|CDC20_IN|)/(J7+alpha*|CDC20_IN|);
let d_dtCDC20_IN_5 : function = k6*|CDC20_IN|;

let d_dtCDH1_1 : function = (k3p*(|CDH1_IN|))/(J3+alpha*|CDH1_IN|);
```

```
let d_dtCDH1_2 : function = (k3s*alpha*|CDC20_A|*|CDH1_IN|)/(J3+alpha*|CDH1_IN|);
let d_dtCDH1_3 : function = (k4*m*alphaDimer*|CDH1|)/(J4+alpha*|CDH1|);
let d_dtCDH1_4 : function = (k4p*alpha*|SK|*|CDH1|)/(J4+alpha*|CDH1|);

let d_dtCKIT_1 : function = k11/alpha;
let d_dtCKIT_2 : function = k12p*|CKIT|;
let d_dtCKIT_3 : function = k12s*|SK|*alpha*|CKIT|;
let d_dtCKIT_4 : function = k12t*m*alphaDimer*|CKIT|;

let d_dtCYCBT_1 : function = k1/alpha;
let d_dtCYCBT_2 : function = k2p*|CYCBT|;
let d_dtCYCBT_3 : function = k2s*alpha*|CDH1|*|CYCBT|;
let d_dtCYCBT_4 : function = k2t*alpha*|CYCBT|*|CDC20_A|;

let d_dtIEP_1 : function = k9*m*alphaDimer*|IEP_IN|;
let d_dtIEP_2 : function = k10*|IEP|;

let m(0.1): var = mu * m * (1 - m/mstar) init 0.7040450659379;
let mass_div : function = m / 2;
let mCycB : var = m * alphaDimer ;

let d_dtSK_2 : function = k13s*|TF|;
let d_dtSK_3 : function = k14*|SK|;

let d_dtTF_1 : function = (k15p*m*|TF_IN|)/(J15+alpha*|TF_IN|);
let d_dtTF_2 : function = (k15s*alpha*|SK|*|TF_IN|)/(J15+alpha*|TF_IN|);
let d_dtTF_3 : function = (k16p*|TF|)/(J16+alpha*|TF|);
let d_dtTF_4 : function = (k16s*m*alphaDimer*|TF|)/(J16+alpha*|TF|);
//-------------------------------------------------------------------------------------------

//-------------------------------------------------------------------------------------------
//TYPE DEFINITION FILE
{ CDH1, CDH1_IN, CYCBT, CDC20_A, IEP_IN, CDC20_IN, IEP, SK, TF,  TF_IN, CKIT }
//-------------------------------------------------------------------------------------------
```