Bridging the Gap between Deterministic and Stochastic Modeling with Automatic Scaling and Conversion

Pengyuan Wang

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science

Yang Cao, Chairman
Clifford A. Shaffer
Layne T. Watson
William T. Baumann

May 7, 2008
Blacksburg, Virginia

Keywords: stochastic simulation, pathway modeling, biomolecular regulatory networks

Bridging the Gap between Deterministic and Stochastic Modeling with Automatic Scaling and Conversion

Pengyuan Wang

Abstract

During the past decade, many successful deterministic models of macromolecular regulatory networks have been built. Deterministic simulations of these models can show only average dynamics of the systems. However, stochastic simulations of macromolecular regulatory models can account for behaviors that are introduced by the noisy nature of the systems but not revealed by deterministic simulations. Thus, converting an existing model of value from the most common deterministic formulation to one suitable for stochastic simulation enables further investigation of the regulatory network. Although many different stochastic models can be developed and evolved from deterministic models, a direct conversion is the first step in practice.

This conversion process is tedious and error-prone, especially for complex models. Thus, we seek to automate as much of the conversion process as possible. However, deterministic models often omit key information necessary for a stochastic formulation. Specifically, values in the model have to be scaled before a complete conversion, and the scaling factors are typically not given in the deterministic model. Several functionalities helping model scaling and converting are introduced and implemented in the JigCell modeling environment. Our tool makes it easier for the modeler to include complete details as well as to convert the model.

Stochastic simulations are known for being computationally intensive, and thus require high performance computing facilities to be practical. With parallel computation on Virginia Tech's System X supercomputer, we are able to obtain the first stochastic simulation results for realistic cell cycle models. Stochastic simulation results for several mutants, which are thought to be biologically significant, are presented. Successful deployment of the enhanced modeling environment demonstrates the power of our techniques.

Acknowledgements

I would like to express my sincere gratitude to all those who have made this thesis possible for me. I want to thank my advisor, Dr. Yang Cao, for his invaluable guidance, advice, and help during my graduate study. He is such a nice and excellent mentor both for academic study and for life. I would like to thank Dr. Layne T. Watson for his advice and honesty on my study. His extraordinary scientific spirit is what I should always learn and respect. My appreciation goes to Dr. Clifford A. Shaffer for his precious suggestions and help on my work and writing. His teaching and training will be part of me for the rest of my career. Very special thanks to Dr. William T. Baumann. I cannot finish this work without his tremendous help and encouragement. I also want to thank my group members and lab mates: Dr. John J. Tyson, Dr. Kathy Chen, Dr. Adrian Sandu, Dr. Ranjit Randhawa, Dr. Jason Zwolak, Jingwei Zhang, David Beck, Tae-Hyuk Ahn, and Rhonda Phillips for their kind support and help. I enjoyed the collaboration and friendship with all of you.

I want to thank my parents for their forever love, support and encouragement. I cannot be where I am without them. I am also very thankful to my roommates who make my life in Blacksburg and America enjoyable and memorable.

This work has been supported by a grant from the National Institutes of Health (GM078989). We also acknowledge the generous support from Virginia Tech's Advanced Research Computing facility for a grant of computing resources on System X.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Mathematical modeling of macromolecular regulatory networks in terms of ordinary differential equations (ODEs) has helped to shed light on the detailed workings of the cell cycle and other intracellular processes [9, 26]. However, deterministic continuous models fail to explain behaviors that are related to the stochastic nature of the underlying system. Inside the cell, gene expression and signaling pathways are often controlled by only a few molecules and thus molecular fluctuations may have important consequences in biology. For example, experiments show that fluctuations in gene expression, protein concentrations and other cellular components are responsible for divergence in genetic activities as well as cell phenotypes [11, 22, 29, 30, 35]. Therefore, deterministic models, which do not account for molecular fluctuations, do not always accurately describe the chemical dynamics of such systems.

To overcome this problem, modelers have started to employ stochastic methods, in particular to account for the differences in outcomes that appear in individual cells. The heart of such models are a set of chemical reactions. There are well-known stochastic simulation techniques for modeling networks of chemical reactions such as Gillespie's stochastic simulation algorithm (SSA) and its variants [7, 14, 15]. Since there currently exist models of value, it is natural to wish to run the existing models using stochastic simulation algorithms, even if those models were created with continuous, deterministic simulators in mind. Our vision is that in practice modelers will develop initial models using deterministic ODEs for their relative speed and simplicity, before progressing to a full stochastic simulation.

Unfortunately, ODE-based models are not cast in the right form for direct simulation using stochastic methods. The values of species in ODE models are usually written in concentration form, because concentrations are what experimenters measure in the lab. However, stochastic simulations require the model to be in terms of population because they account for individual molecules. Thus, a translation process must take place. For large complex models that could have over one hundred parameters, this process would be tedious and error prone to perform by hand. In this thesis we describe our progress toward automatic conversion of models within the JigCell modeling environment [20] and show how it helps modelers do stochastic simulations on complex macromolecular regulatory networks.

Gillespie's stochastic simulation algorithm uses propensity functions that account for the probabilities that reactions will fire. They take the place of rate laws (velocities) for the reactions in a deterministic formulation of the model. Though they have different physical meanings, these two formulations are closely connected and can be converted one to the other.

Our tools are constrained by the fact that they must represent both the deterministic and stochastic forms within the Systems Biology Markup Language (SBML) [18], which is the de facto standard within the systems biology modeling community today.

When first building deterministic models, normalized or scaled values for species concentrations are often used. This is because the actual value of a species' concentration or number of molecules is often unknown. Instead, arbitrarily scaled values that are sufficient to describe the abstract dynamics of the system are used. Typical concentrations of various species in a system are usually set to 1, regardless of their actual values, so they are also called normalized concentrations. The actual concentration corresponding to 1 unit of normalized concentration is called the characteristic concentration. Thus, before being converted to a stochastic model, the model should be converted to actual concentrations first, by choosing characteristic concentrations for all of the species and then scaling all the normalized values back to their actual values. This might appear to be a trivial issue, but there are two reasons why it is not. First, there is non trivial reasoning that must be applied to generate the correct scaling functions. Second, this scaling requires careful definition of units for all species. We think the scaling task can be elegantly solved by following the units of values in the model, which will be explained in detail in later part of this thesis.

Whether a model is written in terms of normalized concentration, real concentration, or population, it describes the same reaction network. Thus, essentially we are identifying the lost information that was not required for the ODE-based model, and changing the form of information describing the model. The key missing information typically is the definition of the units for various constants and state variables. Unit definitions include any scaling factors involved in the corresponding variables. Modelers make implicit assumptions for unit definitions, and many tools today do not support defining this information explicitly. During the conversion process, it is important to enforce units on every species and parameter, and to insure unit consistency inside the model. The explicit definition of units is a necessary step in building large and complex models.

When implementing the conversion function, we keep the following goals in mind. The toolkit should be smart enough to automatically infer the correct units for parameters and to check unit consistency within the model. It should be convenient for the modeler to input whatever information cannot be inferred. Conversion between concentrations and populations should be automatic once all necessary information is available. It should be flexible so that once converted, the model can be modified at will without any restriction that might be implied from the original deterministic model. In other words, the modeler should be free to add new features that are not modeled in the original deterministic model. JigCell now achieves all these goals, and thereby greatly reduces the work load when modelers move between deterministic and stochastic models. Our technique allows us to conveniently and efficiently implement stochastic simulations on significant models.

A flexible and efficient stochastic simulator, StochKit [24], is modified to carry out simulations on our models. We have developed an SBML to StochKit model file translator to automatically translate a model into the format defined and understood by StochKit so that the modeler can simulate the model directly from JigCell. The building of conversion functions and the model file translator are two major improvements to JigCell to upgrade it from a deterministic modeling environment to a deterministic and stochastic modeling environment.

A stochastic simulation usually requires significantly more computing resources than a deterministic simulation, and that simulation must be run thousands of times to produce ensemble averages. The ensemble is inherently parallelizable. We report here our initial results from using StochKit to run simulations of both a simplified and a more detailed (and therefore larger) budding yeast cell cycle model on System X [39], Virginia Tech's parallel supercomputer. We also compare the System X time to the time required by a deterministic simulation on the same reaction networks to contrast the difference in the computation time.

A direct conversion of a model from deterministic form to stochastic form is just the first step in building a realistic model. After successful simulations on the directly converted models we seek to make the stochastic model more realistic by adding various features not modeled in deterministic models, such as growing volume and random cell division. What's more, the power of a successful model lies in its abilities to accurately model both the wild type cell and various mutants. By doing simulations of realistic stochastic models of several mutants, we obtained some biologically significant and interesting results, such as probabilistic cessation of cycling, which demonstrates the power of our technique.

The main research objective for this thesis is to make it easy for modelers to develop stochastic models from existing deterministic models. The structure of this thesis is organized as follows: Chapter 2 introduces concepts in modeling regulatory networks and our modeling environment. Chapter 3 gives a theoretical introduction to the conversion process and explains how to achieve it using units. Chapter 4 applies the conversion process to implement units and model conversion in JigCell. Chapter 5 introduces the stochastic simulator StochKit and how we integrated it into JigCell by building a model file translator. Chapter 6 presents our stochastic simulation results on a simplified model and a full-sized cell cycle model and its mutants. Chapter 7 concludes the work in this thesis and mentions future research work on building realistic stochastic models.

Chapter 2

Regulatory Network Modeling

2.1 The Cell Cycle Model

A major challenge of contemporary cell biology is to understand the molecular mechanisms regulating complex biochemical systems such as eukaryotic cell reproduction. Modeling of such a system typically consists of many species interconnected by various kinds of reactions that represent the synthesis, degradation, activation, inhibition, and other biochemical activities hypothesized by modelers to correctly describe the system [9, 31]. Mathematical modeling of the dynamical properties of these reaction networks plays a key role in studying them [41, 42].

Figure 2.1 shows the wiring diagram of a simple regulatory network. In this graph, vertices represent substrates and products (collectively referred to as species), solid directed edges represent biochemical reactions, and dashed directed edges represent regulatory signals. The icon consisting of four small circles represents products of degradation. It is only "nothing" ($\emptyset$) in the sense that the model has no use for it.

This is a simplified version of a published cell cycle model [9]. The cell cycle is usually subdivided into four phases: G1, S, G2, M. The M phase is further divided into metaphase and anaphase. In this model, $CycB$

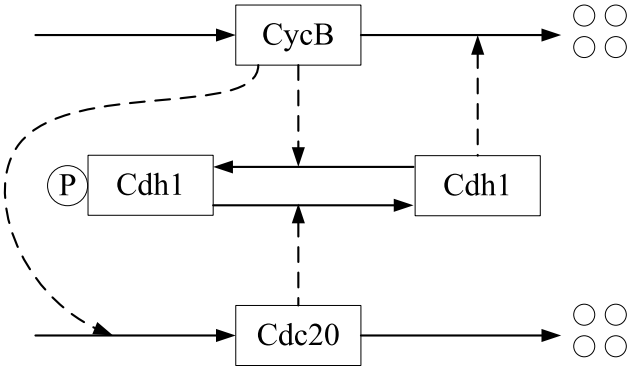


Figure 2.1: Network diagram of a simple regulatory network

is a central component in controlling cycling events, which drives the cell through S phase, G2 phase and up to metaphase. The activity of $CycB$ is regulated by $Cdh1$ and $Cdc20$. $CycB$ and $Cdh1$ are antagonistic proteins: $Cdh1$ destroys $CycB$ activity by degrading it, and $CycB$ inactivates $Cdh1$ by phosphorylating one of its subunits. $Cdc20$ activates $Cdh1$ and itself is activated by $CycB$. This small model is used as an example to illustrate our technique in this thesis.

From the diagram systems biologists can derive specific reactions (2.1–2.7). The substrates and products correspond to what is drawn in the diagram and the reaction rates are hypothesized by the modeler to be appropriate for the corresponding reaction. Mass action laws are usually used to model simple synthesis and degradation. For enzyme catalyzed reactions, Michaelis-Menten and Hill functions are often used as rate laws.

$$\emptyset \xrightarrow{\kappa_1 m} CycB \tag{2.1}$$

$$CycB \xrightarrow{\kappa_2'[CycB]_n} \emptyset \tag{2.2}$$

$$CycB + Cdh1 \xrightarrow{\kappa_2''[CycB]_n[Cdh1]_n} Cdh1 \tag{2.3}$$

$$Cdh1 \xrightarrow{\frac{\kappa_4[Cdh1]_n[CycB]_n}{\Gamma_4+[Cdh1]_n}} Cdh1_P \tag{2.4}$$

$$Cdh1_P \xrightarrow{\frac{\kappa_3''[Cdh1_P]_n[Cdc20]_n}{\Gamma_3+[Cdh1_P]_n}} Cdh1 \tag{2.5}$$

$$\emptyset \xrightarrow{\kappa_5'+\frac{\kappa_5''[CycB]_n^2}{{\Gamma_5}^2+[CycB]_n^2}} Cdc20 \tag{2.6}$$

$$Cdc20 \xrightarrow{\kappa_6[Cdc20]_n} \emptyset \tag{2.7}$$

In this model, parameter m represents the mass of the cell, which is directly proportional to cell size. The background synthesis rate of $CycB$ is also assumed to be proportional to cell size. The following notation is used through out this thesis. For any species S, $[S]_n$ denotes its normalized concentration, and $[S]$ denotes the real concentration. N_S denotes the number of molecules (population) of species S. κ and Γ are parameters of a normalized model. k and J are scaled versions of κ or Γ, respectively, used in the unnormalized model in terms of real concentrations.

Complex rate laws such as Michaelis-Menten and Hill functions are only approximations for more complex interactions going on within the cell. We refer to models using such complex rate laws as a "packed" formulation for the model. A stochastic model ideally will be defined by a collection of simple mass action (elementary) reactions, but it is also possible to stochastically simulate reactions that use more complex rate laws [5, 8, 33]. While this may somewhat underestimate the molecular noise, it suffices to demonstrate the validity of our approach.

2.2 Deterministic Formulation

From the reactions one can derive ordinary differential equations (ODEs) to formally define the temporal behavior of the system

$$\frac{dS_i}{dt} = \sum_{j=1}^{R} b_{ij} v_j, \qquad i = 1, \ldots, N,$$

where R is the number of reactions, N is the number of species, v_j is the reaction rate law of the jth reaction in the network, and b_{ij} is the stoichiometric coefficient of species i in reaction j ($b_{ij} < 0$ for substrates, $b_{ij} > 0$ for products, $b_{ij} = 0$ if species i takes no part in reaction j). The reaction rate of each reaction tells how fast the reaction fires. This formulation is deterministic—multiple simulations will produce exactly the same result. Thus this formulation can only account for average behavior of the system. However, this formulation is fast (in terms of modeling and simulating) and has been demonstrated to give good results for many models. A comprehensive, accurate, predictive model of cell cycle control in budding yeast using ODEs has only recently been created by the Tyson-Chen-Novak modeling group [9]. Below are the derived ODEs for the simplified model of Figure 2.1:

$$\frac{d[CycB]_n}{dt} = \kappa_1 m - (\kappa_2' + \kappa_2''[Cdh1]_n)[CycB]_n, \tag{2.8}$$

$$\frac{d[Cdh1]_n}{dt} = \frac{\kappa_3''[Cdc20]_n(1 - [Cdh1]_n)}{\Gamma_3 + (1 - [Cdh1]_n)} - \frac{\kappa_4[CycB]_n[Cdh1]_n}{\Gamma_4 + [Cdh1]_n}, \tag{2.9}$$

$$\frac{d[Cdc20]_n}{dt} = \kappa_5' + \kappa_5'' \frac{[CycB]_n^2}{\Gamma_5^2 + [CycB]_n^2} - \kappa_6[Cdc20]_n. \tag{2.10}$$

In this model, species' values are described in normalized concentration, which is what modelers usually do when building deterministic models. Besides avoiding the trouble of looking for characteristic concentrations, describing values in normalized concentration usually simplifies the modeling task. The equations assume that the normalized concentration of total $Cdh1$ is 1, so that the concentration of the phosphorylated form can be written as $[Cdh1_P]_n = 1 - [Cdh1]_n$.

After obtaining ODEs, the modeler could use a standard ODE solver to solve the equations for deterministic simulations.

2.3 Stochastic Formulation

Contrary to a deterministic formulation, a stochastic formulation tries to understand the far-from-average behavior of individual cells by accounting for noisy events in the system. The most popular way to do this is by using Gillespie's stochastic simulation algorithm (SSA). Gillespie's SSA assumes the system is well-stirred and the volume is fixed. In the SSA, the system is still formulated as N species that are in state $X \equiv (X_1, \ldots, X_N)$, and R reactions. However, instead of using reaction rate v_j for each reaction, there is a propensity a_j defined for each reaction, with the meaning that

$$a_j dt \equiv \text{Probability that reaction } j \text{ will occur in the container volume } V \text{ in } (t, t+dt), \text{ given that the system is in the state } (X_1, \ldots, X_N) \text{ at time } t.$$

So a_j describes the likelihood the reaction will happen. Besides that, a state change vector $\alpha_j \equiv (\alpha_{1j}, \ldots, \alpha_{Nj})$ is defined for each reaction with the meaning that

$$\alpha_{ij} \equiv \text{the change in the number of } S_i \text{ molecules caused by one } R_j \text{ reaction event.}$$

In other words, R_j induces $X \to X + \alpha_j$. (α_{ij}) is called the "stoichiometric matrix".

The SSA is a procedure for constructing sample paths or realizations of $X(t)$. The state of the system is updated by determining 1) the time to the next reaction, and 2) which reaction will occur next. With $p(\tau, j|X, t)$ defined by

$$p(\tau, j|X, t) \equiv \text{probability, given } X \text{ and } t, \text{ that the next reaction will occur in } [t+\tau, t+\tau+d\tau), \text{ and will be an } R_j \text{ reaction,}$$

one can prove that

$$p(\tau, j|X, t) = a_j(X) exp(-a_0(X)\tau), \text{ where } a_0(X) \equiv \sum_{j'=1}^{R} a_{j'}(X).$$

This implies that the time τ to the next reaction event is an exponential random variable with mean $1/a_0(X)$, and the index j of that reaction is an integer random variable with probability $a_j(X)/a_0(X)$. Therefore, the "direct" version of the SSA is:

1. With the system in state X at time t, evaluate $a_0(X) \equiv \sum_{j'=1}^{R} a_{j'}(X)$.
2. Draw two unit-interval uniform random numbers r_1 and r_2, and compute τ and j according to
   - $\tau = \frac{1}{a_0(X)} \ln(\frac{1}{r_1})$
   - $j =$ the *smallest integer* satisfying $\sum_{j'=1}^{j} a_{j'}(X) > r_2 a_0(X)$.
3. Replace $t \leftarrow t + \tau$ and $X \leftarrow X + \alpha_j$.
4. Record (X, t). Return to Step 1, or else end the simulation.

In the SSA the system evolves in a discrete way. Contrary to an ODE system where all the reactions happen all the time in a continuous and deterministic way, in the SSA at some time one reaction happens and at other times another reaction happens. Every firing of a reaction event will produce or eliminate some molecules of the corresponding species, so the value of the state variables should be in number of molecules rather than concentration.

The deterministic formulation and stochastic formulation are closely connected and can be converted from one to another. They both formulate the system using species and reactions, and some mathematical

expression (v_j or a_j) for each reaction. Although v_j and a_j have different physical meanings, they are closely related.

To explain this we need to first examine the formulation of a_j. In Gillespie's SSA $a_j \equiv c_j h_j$. For each reaction channel R_j the SSA assumes that there exists a scalar rate constant c_j such that

$$c_j dt + o(dt) \equiv \text{the probability that a particular combination of molecules from channel } R_j \text{ selected at the moment } t \text{ will react in the infinitesimal interval } [t, t+dt). \quad (2.11)$$

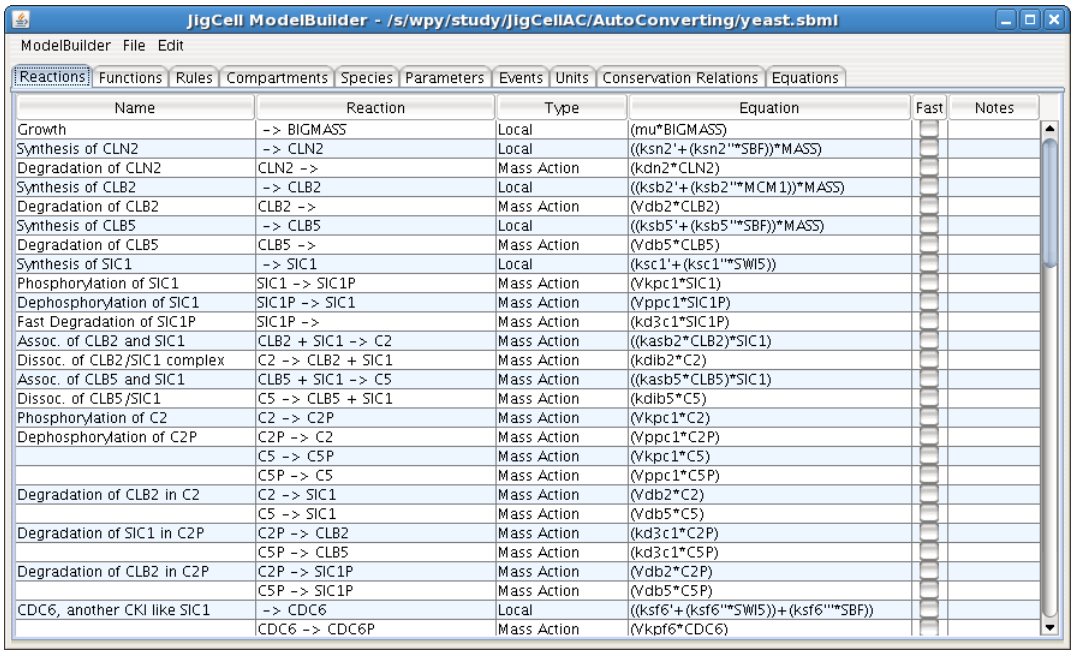
h_j is the total number of possible distinct combinations of molecules for a channel R_j. For a reaction like $S_1 + S_2 \rightarrow 2S_1$, $h_1 = X_1 X_2$ and for a reaction like $2S_1 \rightarrow S_1 + S_2$, $h_2 = \frac{X_1(X_1-1)}{2}$. Therefore, for these two reactions their propensity functions are $a_1 = c_1 X_1 X_2,$ and $a_2 = c_2 \frac{X_1(X_1-1)}{2}$.

For the same reactions, if we still express the value of the species in number of molecules, assuming elementary mass action reactions their deterministic reaction rate laws should be $\frac{k_1}{V} X_1 X_2$ and $\frac{k_2}{V} X_1 X_1$ (explained in detail in Chapter 3). It is shown in [14] that in practice $k_1 = V c_1$ and $k_2 = V \frac{c_2}{2}$. Thus, effectively for the two reactions their propensity functions are $a_1 = \frac{k_1}{V} X_1 X_2$ and $a_2 = \frac{k_2}{V} X_1 (X_1 - 1)$, roughly equal to their rate law forms. This applies to all elementary reaction types. That is, if the reaction has distinct substrates, then the format of the propensity function is the same as the reaction rate law based on population of species, and if the reaction has more than one identical substrate, then the format of the propensity function roughly shares the format of the reaction rate law based on populations of species. It differs slightly in the number of molecules due to the way that distinct pairs of molecular combinations are counted. This also applies to more complex, non-elementary reactions (refer to [5, 8, 33]). Thus, the major effort of deriving stochastic formulations is to convert the original deterministic model from concentrations to populations.

2.4 Modeling Environment

JigCell is our suite of software components intended to provide a problem solving environment for modeling molecular regulatory networks [2, 43].

The Model Builder in JigCell assists the user to translate a wiring diagram into a set of reactions facilitated by definitions of species, parameters, compartments, functions, rules, events, and units (every component has its direct SBML counterpart). It has a spreadsheet-like interface. Although JigCell was mainly used to construct deterministic models, its interface is suitable to construct both deterministic and stochastic models, because both models describe the network by a set of reactions and an algebraic expression (rate law or propensity function) for each reaction. The Model Builder's organization leads users to provide the complete information necessary, and its series of internal checks help to minimize errors in complex models. However, prior to this work it did not check or take advantage of arbitrary unit definitions, nor can it convert models. Figure 2.2 displays a screen shot of the Model Builder.



Name	Reaction	Type	Equation	Fast	Notes
Growth	-> BIGMASS	Local	(mu*BIGMASS)		
Synthesis of CLN2	-> CLN2	Local	((ksn2'+(ksn2''*SBF))*MASS)		
Degradation of CLN2	CLN2 ->	Mass Action	(kdn2*CLN2)		
Synthesis of CLB2	-> CLB2	Local	((ksb2'+(ksb2''*MCM1))*MASS)		
Degradation of CLB2	CLB2 ->	Mass Action	(Vdb2*CLB2)		
Synthesis of CLB5	-> CLB5	Local	((ksb5'+(ksb5''*SBF))*MASS)		
Degradation of CLB5	CLB5 ->	Mass Action	(Vdb5*CLB5)		
Synthesis of SIC1	-> SIC1	Local	(ksc1'+(ksc1''*SWI5))		
Phosphorylation of SIC1	SIC1 -> SIC1P	Mass Action	(Vkpc1*SIC1)		
Dephosphorylation of SIC1	SIC1P -> SIC1	Mass Action	(Vppc1*SIC1P)		
Fast Degradation of SIC1P	SIC1P ->	Mass Action	(kd3c1*SIC1P)		
Assoc. of CLB2 and SIC1	CLB2 + SIC1 -> C2	Mass Action	((kasb2*CLB2)*SIC1)		
Dissoc. of CLB2/SIC1 complex	C2 -> CLB2 + SIC1	Mass Action	(kdib2*C2)		
Assoc. of CLB5 and SIC1	CLB5 + SIC1 -> C5	Mass Action	((kasb5*CLB5)*SIC1)		
Dissoc. of CLB5/SIC1	C5 -> CLB5 + SIC1	Mass Action	(kdib5*C5)		
Phosphorylation of C2	C2 -> C2P	Mass Action	(Vkpc1*C2)		
Dephosphorylation of C2P	C2P -> C2	Mass Action	(Vppc1*C2P)		
	C5 -> C5P	Mass Action	(Vkpc1*C5)		
	C5P -> C5	Mass Action	(Vppc1*C5P)		
Degradation of CLB2 in C2	C2 -> SIC1	Mass Action	(Vdb2*C2)		
	C5 -> SIC1	Mass Action	(Vdb5*C5)		
Degradation of SIC1 in C2P	C2P -> CLB2	Mass Action	(kd3c1*C2P)		
	C5P -> CLB5	Mass Action	(kd3c1*C5P)		
Degradation of CLB2 in C2P	C2P -> SIC1P	Mass Action	(Vdb2*C2P)		
	C5P -> SIC1P	Mass Action	(Vdb5*C5P)		
CDC6, another CKI like SIC1	-> CDC6	Local	((ksf6'+(ksf6''*SWI5))+(ksf6'''*SBF))		
	CDC6 -> CDC6P	Mass Action	(Vkpf6*CDC6)		

Figure 2.2: Main interface of JigCell Model Builder showing reactions of a full-sized cell cycle model.

The JigCell Run Manager allows users to easily specify an ensemble of simulation runs, where each run typically corresponds to one form of the organism such as a mutant formed by a gene knockout. Each such run is specified by a set of parameter values and initial conditions, and a specification for the numerical simulation of the equations. The Run Manager calls external simulators to carry out simulation tasks. Prior to this work only deterministic simulators had been integrated into JigCell. By integrating StochKit [24] into the Run Manager we are able to run stochastic simulations from JigCell. StochKit is an efficient, extensible stochastic simulation package developed in C++. All simulation methods in StochKit are based on Gillespie's SSA [14] and tau-leaping [15] algorithms.

JigCell inputs and outputs models in the SBML language. SBML is a machine-readable format for representing models and it is usually encoded as an XML document. It breaks a model into components and each type of component in a model is described using a specific type of data object that organizes the relevant information. The top level of an SBML model definition consists of lists of the components [19]:

```
beginning of model definition
  list of function definitions
  list of unit definitions
  list of compartments
  list of species
  list of parameters
  list of rules
```

```
  list of reactions
  list of events
end of model definition
```

Not all components are listed above, and only relevant ones to this thesis are introduced. The meanings of these components are:

Function definition: A named mathematical function that may be used throughout the rest of a model.

Unit definition: A named definition of a new unit of measurement, or a redefinition of an existing SBML default unit. Named units can be used in the expression of quantities in a model.

Compartment: A well-stirred container of a particular type and finite size where species may be located. Every species in a model must be located in a compartment. In our model there is typically only one compartment—the cell.

Species: A pool of entities of the same species type located in a specific compartment. Species could be proteins, genes or any other kinds of reactants or products.

Parameter: A quantity with a symbolic name. It could be a reaction rate constant or any other constant or variable.

Rule: A mathematical expression added to the set of equations constructed based on the reactions defined in a model. There are three types of rules. An *assignment rule* defines how a variable's value is calculated from other variables. An *algebraic rule* constrains the value of a mathematical expression to be zero. A *rate rule* defines the rate of change of a variable, but it is not used in building our models. Rule is an extension on reaction rate equations. The set of rules in a model can be used with the reaction rate equations to determine the behavior of the model with respect to time.

Reaction: A statement describing some transformation, transport or binding process that can change the amount of one or more species. Reactions have associated kinetic rate expressions describing how quickly they take place in a deterministic formulation, or propensity functions in a stochastic formulation describing the likelihood they take place.

Event: A statement describing an instantaneous, discontinuous change in a set of variables of any type (species quantity, compartment size or parameter value) when a triggering condition is satisfied. It consists of a *trigger* to define the firing condition, a list of *event assignments* to set the values of variables, and optionally a *delay* to specify the lengh of time between when the event has fired and when the event's assignments are actually executed.

By supporting SBML as an input and output format, different software tools can all operate on an identical external representation of a model, enabling convenient exchange of models between systems biologists. While SBML is the de facto standard for models in systems biology, and while SBML is rapidly developing to support stochastic models, it still lacks certain key features. For example, SBML cannot describe stochastic events. SBML also has no direct method for allowing users to indicate whether a given model is in a deterministic or stochastic form. Fortunately, once a model is completely specified as described in the next chapter, it is suitable for simulation by both deterministic and stochastic simulators. Since StochKit requires

the input model to be written in a pre-defined C++ file format, an SBML to StochKit model file translator is needed.

2.5 Challenges in Converting the Model

The simple cell cycle model includes typical reaction types, namely mass action, Michaelis-Menten and Hill functions. Intuitively, the conversion of reaction rate from concentration based to population based could simply be a matter of mapping one type of rate law to the corresponding converted form. For instance, mass action with two reactants $k[S_1][S_2]$ should be mapped to $\frac{k}{V}N_{S_1}N_{S_2}$. However, in practice modelers often use non-standard rate laws. For example, Reaction 2.6 is essentially a combination of mass action with zero reactant (κ_5') and a Hill function ($\frac{\kappa_5''[CycB]_n^2}{{\Gamma_5}^2+[CycB]_n^2}$), but the program can hardly determine this. Instead, it would regard it as a non-typical arbitrary rate law. As another example, in SBML κ_2' of Reaction 2.2 could be given an assignment rule computing any expression, which breaks the assumed meaning of the mass action reaction type (where κ_2 should be a constant). This behavior is common in our larger models. Lastly, in SBML, it is possible to define a parameter as a "species", thus without checking units it is impossible to distinguish between species and parameters. Therefore, the only reliable way to parse arbitrary combinations of different reaction types is to follow the units of species and parameters.

The original deterministic model is written in terms of normalized concentrations, but lacks the following information necessary for conversion: the units, including scaling factors, of species; the units of parameters; and the initial volume of the cell. It is easy to enter the initial volume. However, since all species and parameters are inter-related by the reaction network, it is not easy to make sure all the units, once entered or changed, are consistent with each other. For example, if the modeler wants to change the characteristic concentration of $CycB$ by changing its scaling factor in the unit, he or she must remember to change the units of κ_1, κ_4 and Γ_5 correctly. It takes careful reasoning to recognize that there is no need to change κ_2', κ_2'' or κ_5'', though it seems that they are also connected with $CycB$. All of this would be a burden to determine by hand, but can be automated.

2.6 Prior Work

Previous works have demonstrated that stochastic modeling can describe molecular regulatory networks more accurately than deterministic modeling. [46] used a stochastic model to investigate the signaling pathway responding to stimuli in yeast. Their work shows that biochemical fluctuations can make noise-induced oscillations and account for variability in single cell measurements, which better conforms experimental observations. [34] built an excitable system driven by noise to explain transient cellular differentiation in *Bacillus subtilis*. [16] developed a stochastic model that quantitatively captures the means and distributions of the genetic expression of an engineered gene regulation system. This model also makes counterintuitive prediction about noise in protein expression levels, which is confirmed experimentally. [21] built a stochastic

hybrid model of the lactose regulation system of *E. coli* that captures important phenomena, such as spontaneous transitions of equilibrium, which cannot be described by continuous deterministic models. These works show that stochastic effects are important in modeling molecular regulatory networks.

In studying of stochastic modeling of yeast cell cycle control mechanisms, [36] studied the effects of stochasticity on a yeast cell cycle model proposed in [31] by rewriting the regulatory system as a system of stochastic differential equations in the form of chemical Langevin equations [45]. They found the introduced noise can account for certain experimental findings, such as the existence of quantized cycle times in a double mutant of fission yeast. They also observed noise-induced oscillations in their simulations. [36]'s inclusion of noise is simplified and arbitrary. [47] developed a more theoretical derivation of the chemical Langevin equations of the same reaction network [31] and studied the effects of cell size on stochastic dynamics of the cell cycle. These two works show that noise needs to be considered in modeling cell cycles. However, they used chemical Langevin equations which are only an approximation of the system and are more suitable for qualitatively accounting for the noise in the system. Moreover, their base model [31] is still a simplified modeling of the cell cycle reaction network.

There have been research efforts on yeast cell cycles using deterministic [23] and stochastic [12,37,48,49] Boolean networks, where each species only has two states. Although this approach has been demonstrated to be useful in studying dynamic properties of the system, it lacks the resolution to produce the result we want in this thesis, which can only be obtained in a state space of realistic values. Still, none of previous works has simulated a reaction network as detailed and realistic as what we will present here.

Many modeling tools have been developed to carry out deterministic and/or stochastic modeling and simulation [1,17,32,40,44]. However, none of them have addressed the problem of converting a deterministic model in normalized concentration to a stochastic model in population. COPASI [17] is one of the few programs to have dedicated functions to check the correctness of units of kinetic parameters and to make conversions between deterministic forms and stochastic forms. However, units in COPASI are fixed and users cannot design arbitrary units, thus there is no way to correctly input a normalized model whose units are scaled and non-typical. Besides, COPASI's checking on units is primitive and incomplete (for example, it doesn't check units of species), which is actually the consequence of rigid assumptions on default units. Virtual Cell [44] also has functions to deal with units and convert models. However, units in Virtual Cell are rigidly assumed too, so the modeler still cannot convert a model expressed in normalized concentration. Besides, its ability to convert rate law expressions is limited. We found that it cannot handle arbitrary expressions. To our knowledge, JigCell is the first modeling tool to take advantage of arbitrary units to flexibly scale and convert models from normalized concentration to population.

Chapter 3

Conversion Process

In this chapter we first introduce general rules to convert from deterministic to stochastic form, and then introduce how unit definitions provide the necessary information to accomplish the convertion task.

3.1 Model Conversion

In Gillespie's algorithm, the propensity function for each reaction replaces the reaction rate law in the ODE model, and is written in terms of numbers of molecules. They share the same expression except when a reaction contains two or more identical reactants, as described in Section 2.3. The slight difference in number of molecules in special cases can be solved straightforwardly, so here we will focus on how to convert a model in terms of normalized concentration to a model in terms of population.

For a species S, let c_S be the scaling factor between its normalized concentration and real concentration. Thus

$$[S] = c_S[S]_n. \tag{3.1}$$

Multiplying the concentration by volume will produce the population,

$$N_S = V[S]. \tag{3.2}$$

Here we define $V = (cell\ volume) \cdot (Avogadro\ number)$ to get N_S in terms of number of molecules instead of $moles$. Depending on the model, V could be a constant or a variable. c_s is a constant for a fixed species. Different species may have different values for c_s. Consider Reaction 2.3. Its contribution to the differential equation of $CycB$ is

$$\frac{d[CycB]_n}{dt} = \ldots - \kappa_2''[CycB]_n[Cdh1]_n, \tag{3.3}$$

where "$\ldots$" represents contribution to the ODE from other reactions. The first step is to convert it to a model in terms of real concentration.

$$\begin{aligned}
\frac{d[CycB]}{dt} &= \frac{dc_{CycB}[CycB]_n}{dt} = c_{CycB}\frac{d[CycB]_n}{dt} \\
&= \ldots - c_{CycB}\kappa_2''\frac{[CycB]}{c_{CycB}}\frac{[Cdh1]}{c_{Cdh1}} \\
&= \ldots - \frac{\kappa_2''}{c_{Cdh1}}[CycB][Cdh1] \\
&\doteq \ldots - k_2''[CycB][Cdh1], \text{ where } k_2'' \doteq \frac{\kappa_2''}{c_{Cdh1}}.
\end{aligned} \tag{3.4}$$

The conversion involves transformation of species and parameters, but the form of the rate law expression is kept, because essentially it is just a scaling of the system. Parameters are used to absorb any constants introduced during the conversion of species, and we shall see later that information for how to transform parameters is embedded in their units.

The second step is to convert the model to population. For any species S, the differential equation for population is

$$\frac{dN_S}{dt} = \frac{d(V[S])}{dt} = V\frac{d[S]}{dt} + [S]\frac{dV}{dt}. \tag{3.5}$$

Note although N_S represents population, it is not necessarily to be an integer here, because the physical meaning of the deterministic formulation is the average value of species.

For the fixed volume case, only the first term $V\frac{d[S]}{dt}$ is not zero and the conversion is similar to the first step as in (3.4). But when the volume is changing, additional terms are required to make the ODE for population complete and correct. The varying volume case will be introduced in Section 6.3. Here we only consider the fixed volume case. Then the model in terms of population would be

$$\begin{aligned}
\frac{dN_{CycB}}{dt} &= V\frac{d[CycB]}{dt} \\
&= \ldots - Vk_2''\frac{N_{CycB}}{V}\frac{N_{Cdh1}}{V} \\
&= \ldots - \frac{k_2''}{V}N_{CycB}N_{Cdh1}.
\end{aligned} \tag{3.6}$$

Therefore, when changing the model from normalized concentration to population, the rate law equation (or propensity function) for reaction 2.3 would be $\frac{k_2''}{V}N_{CycB}N_{Cdh1}$.

Generally speaking, let f_N, f_R, f_M be the rate law functions for a reaction in normalized concentration, real concentration and number of molecules. The rule for conversion is:

$$\frac{d[\hat{S}]_n}{dt} = \ldots + f_N(\vec{\kappa}, \overrightarrow{[S]_n}), \tag{3.7}$$

$$\frac{d[\hat{S}]}{dt} = \ldots + f_R(\vec{k}, \overrightarrow{[S]}) = \ldots + f_N(\vec{\kappa}, \frac{\overrightarrow{[S]}}{c_S})c_{\hat{S}}, \tag{3.8}$$

$$\frac{dN_{\hat{S}}}{dt} = \ldots + f_M(\vec{k}, \overrightarrow{N_S}, V) = \ldots + f_R(\vec{k}, \frac{\overrightarrow{N_S}}{V})V, \tag{3.9}$$

where $\vec{x}$ represents an array of all the parameters or species participating in the reaction, $\hat{S}$ is the species whose value will be changed by the reaction, and $c_{\hat{S}}$ is its scaling factor. If more than one species is changed by the same reaction, they must have the same scaling factor. The parameter in real concentration k is defined from the parameter in normalized concentration κ by absorbing all the introduced scaling factors as shown in (3.4), thus $f_R = f_N$, which means the form of the rate law is kept when converting from normalized concentration to real concentration.

3.2 Using Units to Facilitate Model Conversion

We think of the scaling factor as an integral part of a species' unit. In SBML, every species and parameter has a value field and a reference to a "Unit Definition" as its unit. Modelers can define arbitrary "Unit Definitions" from a combination of a list of "Units," where "Units" are transformations of pre-defined base units [18]. For example,

$$\{u\} = (m_1 10^{s_1}\{u_{b_1}\})^{x_1}(m_2 10^{s_2}\{u_{b_2}\})^{x_2}\ldots(m_n 10^{s_n}\{u_{b_n}\})^{x_n} \tag{3.10}$$

where $\{u\}$ is the "Unit Definition," $\{u_{b_i}\}$ is the base unit, $(m_i \cdot 10^{s_i}\{u_{b_i}\})^{x_i}$ is a transformation. This is a very broad definition of unit, which one can take advantage of to store the scaling factor and do model conversion.

For example, in a deterministic model with normalized concentration, one can assign the unit of $CycB$ to be $50 \cdot 10^{-9} \cdot Mole \cdot L^{-1}$, in other words, $50nM$. (In SBML, Unit Definition can only be defined from base units, not from other definitions, that is, it is not hierarchical. Thus, the model does not have a direct unit of $Molar$ or nM.) The meaning of this unit definition is that one unit of normalized concentration equals 50 times one unit of real concentration (which we assume to be nM). Thus, the scaling factor is 50.

Consistent units inside the model is a necessity for correct model conversion. There are three principles of unit consistency that should be followed.

First, every pair of quantities (a parameter, a species, or the result of a sub-expression) connected by $+$ or $-$ in any mathematical expression in the model should bear the same units. The mathematical expression could be a rate law in deterministic formulation, a propensity function in stochastic formulation or the expression in a rule. This is easy to understand because there is no physical meaning to add or subtract two

quantities in different units. Quantities with different units can be multiplied or divided and the resulting unit should be computed using the same operation (multiplication or division).

This implies that we can compute units in a mathematical expression, by computing

$$u_{f(A)} = f(u_A). \tag{3.11}$$

where u_q denotes the unit of quantity q, f is a function and A denotes all the arguments of the function. In other words, we can compute the result unit of an expression in a similar way as we compute its value.

The rules for computing units are: if two quantities are connected by addition or subtraction their units should be the same and the resulting unit is also the same; if two quantities are connected by multiplication, division or any power-law, the resulting unit is computed using the same operation; for other operators, units should be computed following the physical meaning of the definition of the operator. Regardless of the specific form of the mathematical expression, the resulting unit of any expression is always a power-law monomial of basic units (see Formula 3.10), just as with the units of individual quantities. For a detailed discussion of units please refer to [4].

Second, the units on both sides of any equation should be the same. This is because not only should values on both sides be equal, their meanings should also match. Particularly, the unit of the calculation result for the rate law expression must be equal to the supposed unit of the reaction rate, i.e.,

$$u_{\frac{dS}{dt}} = u_{f(A)}. \tag{3.12}$$

Here S could be any form of a species, be it in concentration or population, and f is the corresponding rate law function. Besides reaction rates, units on both sides of any assignment rule should also be the same.

Third, all species whose values are changed by the same reaction must have the same units (excluding enzymes or any modifiers because their values are not changed by the reaction). Actually this is a corollary to Equation 3.12. Suppose species S_1 and S_2's values are changed in the same reaction, from (3.11) and (3.12), we have

$$u_{\frac{dS_1}{dt}} = u_{f(A)} = u_{\frac{dS_2}{dt}} \tag{3.13}$$

$$\Rightarrow u_{S_1} = u_{S_2}. \tag{3.14}$$

When converting the model, instead of using different notations for different meanings of species and parameter values as shown in the previous section, we always use the same set of species and parameters. What will be changed are the units and values of species and parameters. For example, a model doesn't have two separate variables named $[S]_n$ and N_S. Instead, it only has a single species with a name "S". The conversion function's task is to change the unit and the value of S during conversion.

Let the unit of a species or parameter S be u_S and the value be v_S. In the first step of the conversion, we need to change the unit of a species from normalized concentration to real concentration by removing the scaling factor from the Unit Definition, and multiplying it to the value field of the species. If we think of the

unit and the value as a whole, we are only changing the form of the model.

To maintain the consistency of the units inside the model, the units of parameters should be changed as well. In most cases, the unit of a parameter is determined by all the other species participating in the reaction. According to Equation 3.11 and 3.12, we have

$$\frac{u_S}{u_t} = f(u_A). \tag{3.15}$$

Thus, we are able to infer the right unit of a parameter, or how to change the unit of a parameter when changing the units of the corresponding species. For example, from (3.4) we can conclude:

$$\frac{u_{CycB}}{u_t} = u_{k_2''} u_{CycB} u_{Cdh1}, \tag{3.16}$$

$$\Rightarrow u_{k_2''} = {u_{Cdh1}}^{-1} u_t^{-1}. \tag{3.17}$$

Similar to species, any scaling of a parameter's unit should be accompanied by the corresponding scaling of the parameters's value as well, otherwise the integrity of the model will be broken. Before the conversion, $u_{k_2''} = 50^{-1} nM^{-1} min^{-1}$. Converting to real concentration will change it to $u_{k_2''} = nM^{-1} min^{-1}$, and 50^{-1} will be multiplied to its value field. By examining units we are able to reason the correct scaling of parameters as we did in (3.4).

If there are multiple parameters appearing in the same term, usually only the product result of these units can be automatically inferred, and the modeler is responsible to split the units among multiple parameters. For example, from Reaction 2.1, it can be inferred that the unit of the reaction rate should be nM/min, i.e.,

$$u_{k_1} u_m = nM/min. \tag{3.18}$$

However, it is impossible for the program to know what should be correct unit of k_1 and m, and the modeler is responsible to tell the program that $u_{k_1} = nM/(min \cdot ng)$ and $u_m = ng$.

To convert the model from real concentration to population, since $N_S = V[S]$, we have:

$$v_{N_S} \cdot u_{N_S} = (v_V \cdot u_V)(v_{[S]} \cdot u_{[S]}) = (v_V \cdot v_{[S]})(u_V \cdot u_{[S]}) \tag{3.19}$$

$$\Rightarrow v_{N_S} = v_V v_{[S]}, \qquad u_{N_S} = u_V u_{[S]}. \tag{3.20}$$

So both the value and units of the species should be multiplied accordingly. By taking units into consideration we make sure that the converted model is consistent with the original model.

Instead of adjusting the parameters' units and values in response to changes made on species, here we choose to compensate for multiplication of species by V by adjusting the rate law expression, as exemplified

	Old		New	
Species	Value	Unit	Value	Unit
$CycB$	0.0535	$50nM$	48	molecules
$Cdh1$	0.0536	$50nM$	48	molecules
$Cdh1_P$	0.946	$50nM$	852	molecules
$Cdc20$	0.111	$50nM$	100	molecules

Table 3.1: Initial conditions of species in the model. Population is calculated using $V = 18molecules/nM$, that is, cell volume $= 30fL$. Note in this small model the characteristic concentrations for species are the same, which is not necessarily true for other models. It is only because for this simple model it is not necessary to find realistic characteristic concentrations, because the reaction network is not realistic after all. Rather, this simple model is served as a demonstration of our conversion process.

in (3.6) and (3.9), because V is a variable in the model.

To sum up, the converted model in terms of population is:

$$\emptyset \xrightarrow{k_1 mV} CycB \tag{3.21}$$

$$CycB \xrightarrow{k_2' N_{CycB}} \emptyset \tag{3.22}$$

$$CycB + Cdh1 \xrightarrow{\frac{k_2''}{V} N_{CycB} N_{Cdh1}} Cdh1 \tag{3.23}$$

$$Cdh1 \xrightarrow{\frac{k_4 N_{Cdh1} N_{CycB}}{J_4 V + N_{Cdh1}}} Cdh1_P \tag{3.24}$$

$$Cdh1_P \xrightarrow{\frac{k_3'' N_{Cdh1_P} N_{Cdc20}}{J_3 V + N_{Cdh1_P}}} Cdh1 \tag{3.25}$$

$$\emptyset \xrightarrow{k_5' V + \frac{k_5'' V N_{CycB}^2}{(J_5 V)^2 + N_{CycB}^2}} Cdc20 \tag{3.26}$$

$$Cdc20 \xrightarrow{k_6 N_{Cdc20}} \emptyset \tag{3.27}$$

The species and parameters before and after the conversion are listed in Tables 3.1 and 3.2.

Conversion takes two steps, as introduced above. The first step moves the scaling factor from the unit to the value field to change the model into real concentration. The second step changes the concentration of species into number of molecules by multiplying each species with V, and adjusting the rate law according to (3.9).

Parameter	Old		New	
	Value	Unit	Value	Unit
k_1	0.01	$50nM/(min \cdot ng)$	5	$nM/(min \cdot ng)$
m	1.5	ng	1.5	ng
k_2'	0.04	$1/min$	0.04	$1/min$
k_2''	1	$1/(50nM \cdot min)$	0.02	$1/(nM \cdot min)$
k_3''	10	$1/min$	10	$1/min$
k_4	35	$1/min$	35	$1/min$
k_5'	0.005	$50nM/min$	0.25	nM/min
k_5''	0.2	$50nM/min$	10	nM/min
k_6	0.1	$1/min$	0.1	$1/min$
J_3	0.04	$50nM$	2	nM
J_4	0.04	$50nM$	2	nM
J_5	0.3	$50nM$	15	nM

Table 3.2: Parameters in the model

Chapter 4

Model Conversion in JigCell

It is typical that modelers build their deterministic models with no unit information. It is difficult to input units correctly. Thus, we designed and implemented a tool in JigCell to facilitate this process. In most cases, the modeler only needs to input the units of species by obtaining the typical (characteristic) concentration of each species from the literature, and our program will automatically fill in the units of parameters for the modeler using Equation 3.15.

The conversion tool consists of two components. The first is *unit checking*, which checks unit consistency inside the model and automatically fills in the units of parameters whenever they can be inferred. The tool prompts the modeler to input any necessary information. The second is *model conversion*, which automatically converts the model by changing values and units of species and parameters and adjusting math expressions inside the model. A typical work flow is illustrated in Figure 4.1.

4.1 Units and Unit Checking

There are three conditions on which JigCell will check the unit consistency inside the model, as introduced in Section 3.2. If all of these are satisfied then the model is ready to be converted, and the conversion process will guarantee the units of the converted model are still consistent. In a brief,the three conditions are:

- Every pair of quantities (a parameter, a species, or the result of a sub-expression) connected by $+$ or $-$ in any mathematical expression in the model should bear the same units.
- The unit of the calculation result of the rate law expression must be equal to the unit of the reaction rate, as in Equation 3.15, and units on both sides of any assignment rule should be the same.
- All species whose values are changed by the same reaction must have the same units (excluding enzymes or any modifiers because their values are not changed by the reaction).

Since units are part of the definition in SBML, they are supported by the JigCell Model Builder (JCMB). However, when developing a deterministic model, units historically have not been used. According to the

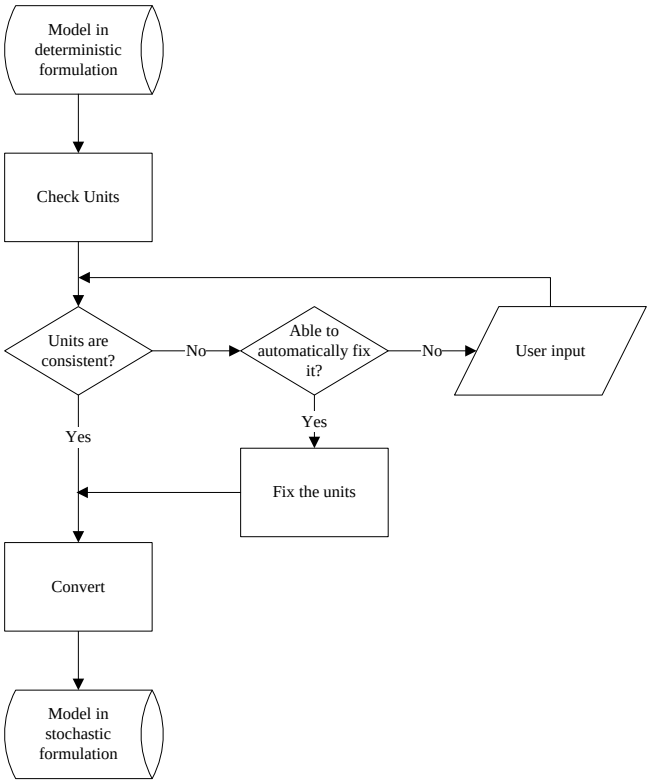


Figure 4.1: Model conversion work flow

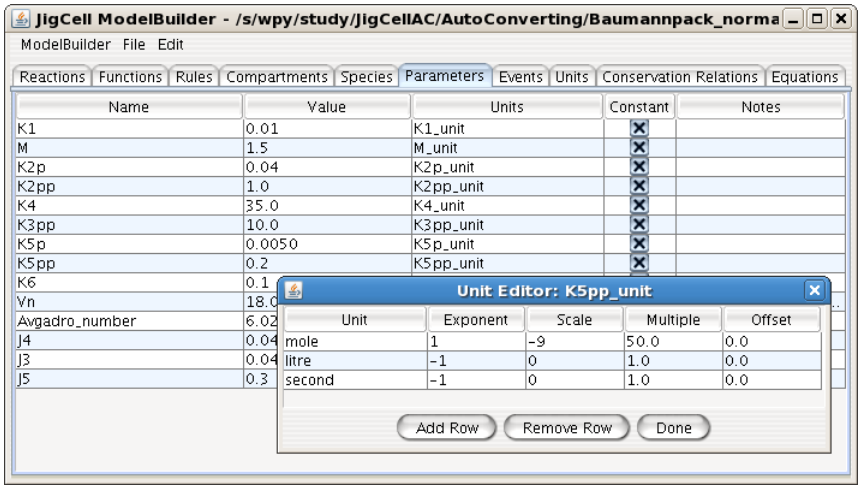


Figure 4.2: Editing unit for a parameter. Notice that a unit is defined from elementary base units.

SBML definition, units are defined separately from species or parameters, and they are referenced by species or parameters. To make units more accessible in JCMB we enable the user to edit unit definition directly for a parameter or species (Figure 4.2).

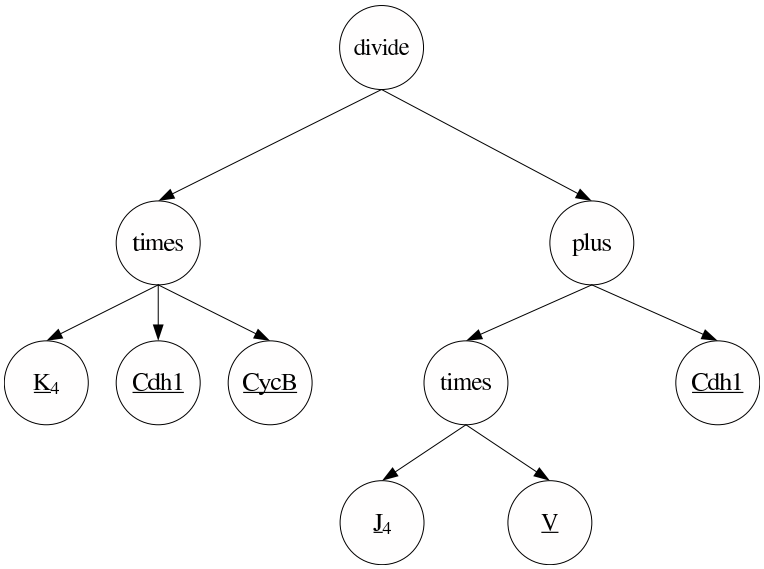


Figure 4.3: Math expression tree for reaction rate of (3.24).

When converting the model from deterministic to stochastic form, defining units that are correct and consistent with each other is the most important step. Units of species should be specified by the modeler, who hopefully can get the information from the literature. Although this part seems straightforward, it is easy for the modeler to make mistakes by violating the first and third conditions mentioned above. After species' units are input, parameters' units are deduced from the equations. In many cases, a parameter's unit can be automatically filled in by JCMB, or fixed in case it was wrong. The consistency of all units can be checked rigorously by JCMB to make sure the model is correct, which is the prerequisite to a correct conversion.

The unit-checking function will check every reaction and every rule using the conditions mentioned above. A unit is deduced from a rate law expression or rule expression and compared with the supposed unit of the reaction rate or target assignment variable. By using a recursive function, unit calculation can handle arbitrary algebraic expressions (operators are so far limited to addition, subtraction, multiplication, division, exponentiation and square root).

The math expressions in SBML are stored in a tree structure with internal nodes as operators and leaves as variables or constants. For example, Figure 4.3 illustrates the structure of math expression for (3.24)'s reaction rate. The unit calculation function works by making recursive operations on the tree nodes.

The core function to calculate the resulting unit of a math expression node is defined as

```
UnitDefinition concludeUnit (Node node, UnitDefinition criteria)
```

where *node* is the node of math expression to be processed and *criteria* is the supposed unit of this node. During the first call to this function the corresponding unit of the reaction rate or the assignment variable is given as *criteria*, and during successive recursive calls *criteria* is assigned by upper layer callers. A *null criteria* means there is no supposed unit for the current node to be processed. The return value of this function

is the resulting unit, or *null* if it encountered an error while parsing the node. The pseudo code for this function is:

```
\\unit (node) returns the unit of a leaf node,
\\i.e. the unit of a species or parameter

\\compare() is an assistant function
compare (node, criteria) {
    if (criteria == null)
        return unit (node);
    else if (unit (node) == criteria)
        return unit (node);
    else
        return null;
}

concludeUnit (node, criteria) {
    if (node == leaf node) {
        return compare (node, criteria);
    }

    else if (node == times) {
        for (i = 0; i < node.children.length; i++) {
            childrenUnits[i] = concludeUnit(node.children[i], null);
        }
        resultUnit = childrenUnits[0]*...*childrenUnits[node.children.length-1];
        return compare (resultUnit, criteria);
    }

    else if (node == divide) {
        childrenUnit[0] = concludeUnit(node.children[0], null);
        childrenunit[1] = concludeUnit(node.children[1], null);
        resultUnit = childrenUnit[0]/childrenUnit[1];
        return compare (resultUnit, criteria);
    }

    else if (node == minus || node == plus) {
        for (i = 0; i < node.children.length; i++) {
            childrenUnits[i] = concludeUnit(node.children[i], criteria);
        }
        for (i = 1; i < node.children.length; i++)  {
            if (childrenUnits[i-1] != childrenUnits[i])
```

```
                return null;
        }
        return compare (childrenUnits[0], criteria);
    }

    else if (node == power) {
        \\ the first child is the exponent
        \\ the second child is the base
        exponent = children[0];
        baseUnit = concludeUnit(node.children[1], power(criteria, 1/exponent));
        resultUnit = power(baseUnit,exponent);
        return compare(resultUnit, criteria);
    }

    else if (node == square root) {
        childUnit = concludeUnit(child, criteria*criteria);
        return compare( sqrt(childUnit), criteria);
    }

    else
        return null;
}
```

The pseudo code shown above illustrates the idea of calculating and comparing units. It can be enhanced to fix many cases of wrong units or infer the correct units during parsing of a math expression node. This function ensures that the first and second conditions of unit consistency are met. The only condition left to check is whether multiple species changed by the same reaction have the same units, which is rather straightforward to implement.

The unit checking results are reported to the modeler. For each error, JCMB will describe what the problem is and what actions (if any) JCMB has taken to fix it. It will also list all the variables involved with each error and let the modeler correct the units directly from the report panel. Figure 4.4 shows an example of unit checking results where all the species' units were correctly input but all the parameters' units, except m's and Γ_5's, were empty. All the problems were detected and fixed, and affected parameters were highlighted by JCMB. The reason why in this example m and Γ_5 were elaborately excluded is because m is known before hand to have the unit of mass but cannot be automatically inferred, and the unit checking function hasn't been able to automatically fix a case like Γ_5 (but unit calculation and verification are always working nevertheless). Figure 4.5 shows what will happen if the modeler did not assign $Cdh1$ and $Cdh1_P$ the same unit. JCMB found the problem and alerted the user by highlighting the problematic species. JCMB never tries to fix the unit of a species, because there is no basis for judging a species' correct unit, which instead should be obtained from the biological literature.

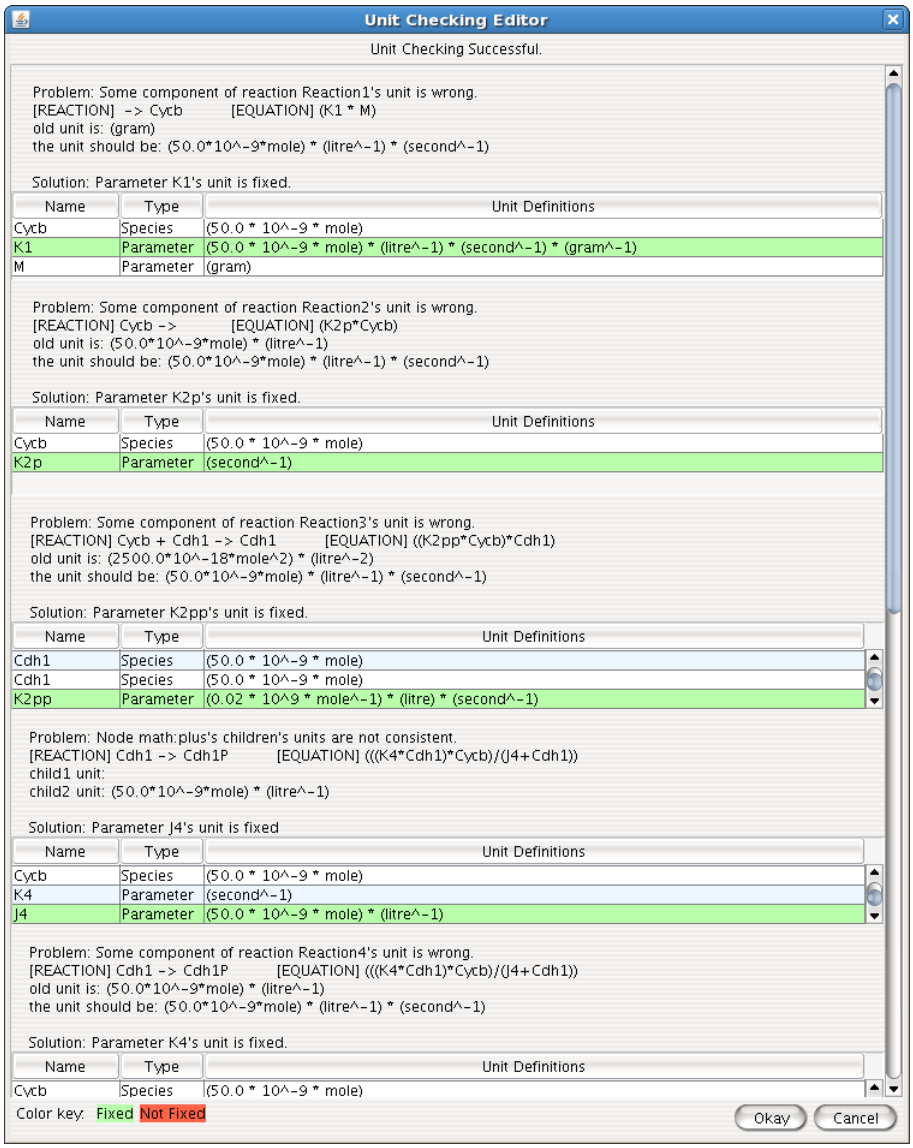


Figure 4.4: Unit checking report panel.

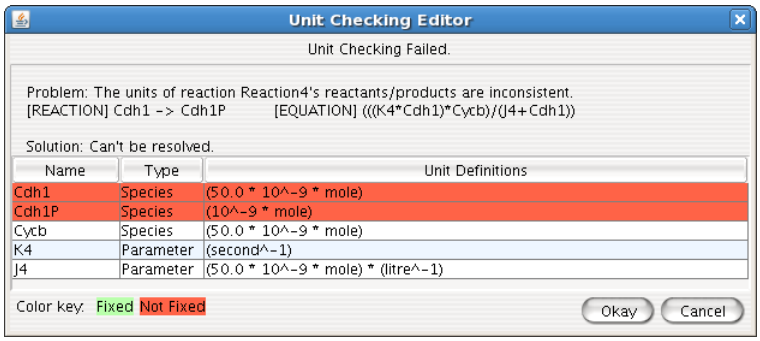
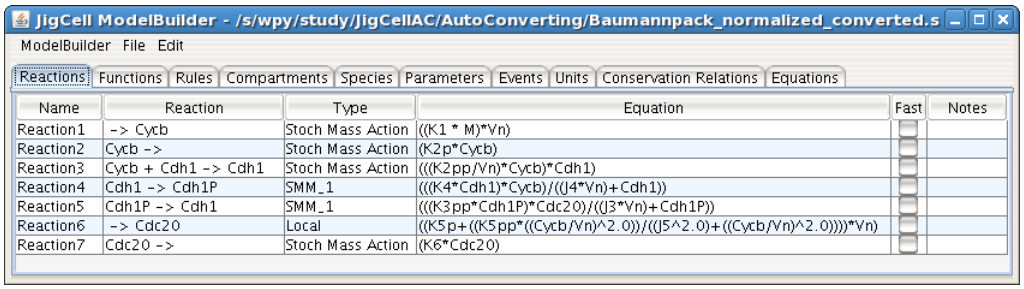


Figure 4.5: Unit checking report panel.



Name	Reaction	Type	Equation	Fast	Notes
Reaction1	-> Cycb	Stoch Mass Action	((K1 * M)*Vn)		
Reaction2	Cycb ->	Stoch Mass Action	(K2p*Cycb)		
Reaction3	Cycb + Cdh1 -> Cdh1	Stoch Mass Action	(((K2pp/Vn)*Cycb)*Cdh1)		
Reaction4	Cdh1 -> Cdh1P	SMM_1	(((K4*Cdh1)*Cycb)/((J4*Vn)+Cdh1))		
Reaction5	Cdh1P -> Cdh1	SMM_1	(((K3pp*Cdh1P)*Cdc20)/((J3*Vn)+Cdh1P))		
Reaction6	-> Cdc20	Local	((K5p+((K5pp*((Cycb/Vn)^2.0))/((J5^2.0)+((Cycb/Vn)^2.0))))*Vn)		
Reaction7	Cdc20 ->	Stoch Mass Action	(K6*Cdc20)		

Figure 4.6: The converted model.

With a built-in unit checking function, it is much easier for the modeler to input correct unit information. He/she can mostly concentrate on the units of species and let the program compute the complicated units of parameters. When the units cannot be fixed, the user can edit them right inside the reporting panel where the problem is reported.

4.2 Conversion

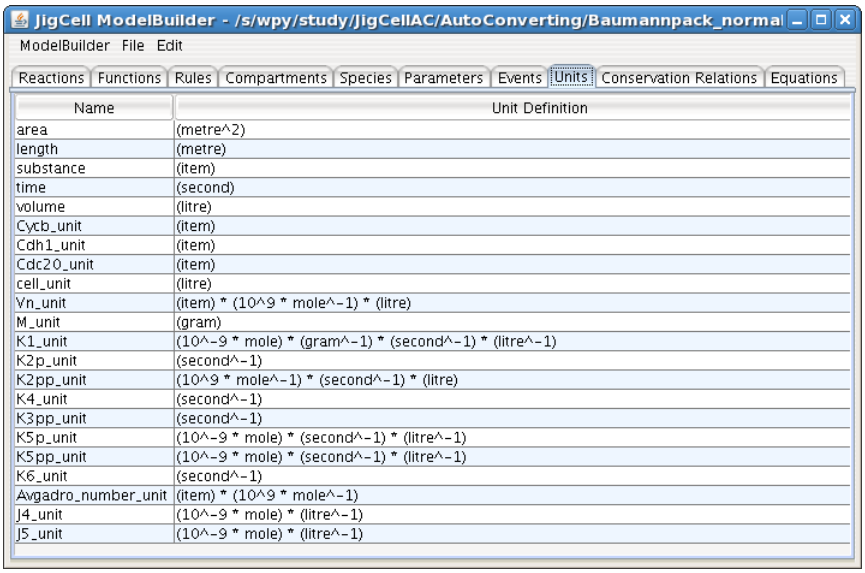
As introduced in Chapter 3, conversion requires two steps. In practice, after setting up the units and getting the volume, the conversion process is simply: 1) Change the form of species from normalized concentration to population by examining their units. 2) For any parameter that has normalized concentration (scaling factor) involved in its unit definition, change it to real concentration. 3) Change rate law equations by replacing every occurrence of a species S with $\frac{S}{V}$ and multiplying the whole rate law equation with V (refer to Equations (3.7-3.9)).

The conversion can be done automatically because: 1) the program can get all the information to convert the model, 2) the program can reason the correctness of the information, and 3) the conversion process can be done in polynomial time.

Figure 4.6 shows reactions in the converted model. There are several predefined types of reactions in JigCell, such as mass action and Michaelis-Menten. For those reaction types that JCMB already has knowledge about, their converted form can be directly coded into the program. For example, mass action with two reactants kS_1S_2 will be replaced by $\frac{k}{V}S_1S_2$. In JCMB, the corresponding forms for mass action and Michaelis-Menten are called "Stoch Mass Action" and "SMM". The direct replacement works faster and makes equations look nicer, however it cannot handle arbitrary expressions. Reaction 6 is a "Local" type reaction, which means it doesn't have a predefined reaction type, or in other words, its expression is arbitrary. For those kind of reactions, they will be converted using the generic rule (3.9).

Figure 4.7 shows the converted units. Notice species' unit definitions are "item", which means that they are in number of molecules. The system of units after the conversion is still consistent.

The whole conversion process is done by just clicking a menu command. Automatic conversion greatly reduces the work load for modelers.



Name	Unit Definition
area	(metre^2)
length	(metre)
substance	(item)
time	(second)
volume	(litre)
Cycb_unit	(item)
Cdh1_unit	(item)
Cdc20_unit	(item)
cell_unit	(litre)
Vn_unit	(item) * (10^9 * mole^-1) * (litre)
M_unit	(gram)
K1_unit	(10^-9 * mole) * (gram^-1) * (second^-1) * (litre^-1)
K2p_unit	(second^-1)
K2pp_unit	(10^9 * mole^-1) * (second^-1) * (litre)
K4_unit	(second^-1)
K3pp_unit	(second^-1)
K5p_unit	(10^-9 * mole) * (second^-1) * (litre^-1)
K5pp_unit	(10^-9 * mole) * (second^-1) * (litre^-1)
K6_unit	(second^-1)
Avgadro_number_unit	(item) * (10^9 * mole^-1)
J4_unit	(10^-9 * mole) * (litre^-1)
J5_unit	(10^-9 * mole) * (litre^-1)

Figure 4.7: The converted units.

4.3 Limitations

Note that events in the model are not processed by the conversion function. Events for deterministic simulation and stochastic simulation should be the same, but they are practically different. Since events are closely connected to the biochemical meaning of the model, it is hard to develop a general process to convert events. A more detailed discussion about events is introduced in chapter 6. However, for the simplified model there is no events so the automatically converted model can be simulated stochastically without any manual post-process.

SBML is about the mathematical description of models. In other words, no biochemical semantic is embedded. A program cannot determine directly from an SBML file what a reaction does or how a species relates to the biological literature. The conversion process does not change the biochemical meaning of the model. Both the deterministic and stochastic formulations describe exactly the same reaction network.

Further processing on models requires additional information. For example, volume is typically not modeled or considered when modelers are building a deterministic cell cycle model because everything is defined in concentrations. But a realistic stochastic model should treat volumes growing explicitly, as might for number of molecules. This also results in dilution of the degradation rates. If this is meant to be done by the program, it has to know which are degradation reactions and what are the rates. The program also has to know what species do not have synthesis or degradation reactions (because in the deterministic model their concentrations are conserved). This task requires consulting the biological meaning of the model which is not contained in the SBML definition.

Part of this problem could be resolved by using sboTerm (SBML's built-in support for System Biology Ontology [38]) and standard annotations to refer to external resources for additional mathematical and

biochemical semantics. For detailed information, refer to Chapter 5 and 6 of [19].

For this study, we only focus on the native SBML definition. Through analysis of units and mathematical expressions, JCMB makes it possible to automatically convert a model from a concentration based formulation to molecular populations. This is the first and foremost step in developing a stochastic model from an existing deterministic model. Chapter 6 will introduce our initial work on how to manually add new features to the converted model to make the stochastic model more realistic. In particular, we will discuss how stochastic models handle events.

Chapter 5

StochKit and the Model File Translator

5.1 StochKit

We use StochKit [24] to perform stochastic simulations. StochKit is an efficient and extensible stochastic simulation package developed in C++. It includes implementation for Gillespie's SSA, optimized SSA, and explicit, implicit and trapezoidal tau-leaping methods. We usually use the basic Gillespie's SSA in our simulations because it is the most accurate one.

The structure of StochKit is organized as shown in Figure 5.1

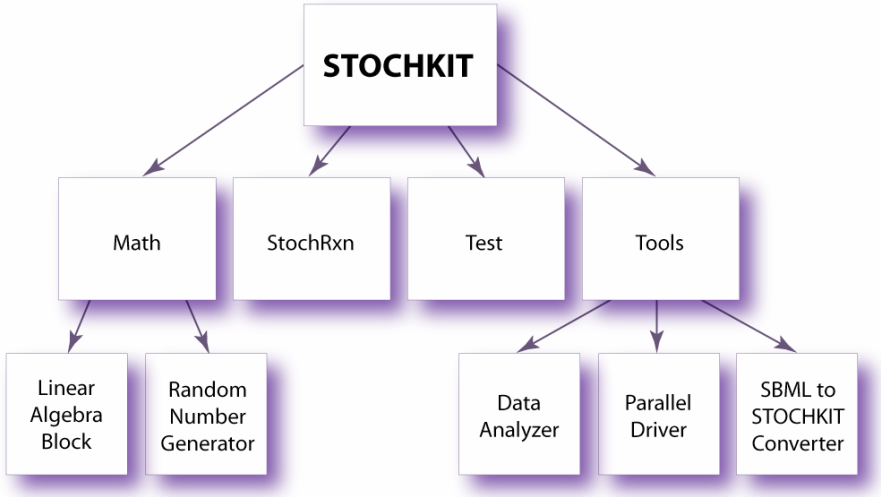


Figure 5.1: Structure of StochKit

Code is placed in directories with the same structure as Figure 5.1. The "Math" directory contains files and packages for underlying mathematical support, such as Vector and Matrix class definitions, internal Random Number Generator (RNG) implementation or external RNG package. Main algorithms are implemented in "StochRxn". "Tools" provides independent programs for file preparation, post-processing, or data analysis.

There is also an MPI interface for StochKit in that directory. "Test" contains models to be simulated. For each model there should be a driver file to link the model definition to the StochKit main program. Models are defined in a C++ file named "`ProblemDefinition.cpp`".

`ProblemDefinition.cpp` consists of several functions to return the data needed by the algorithm or carry out computations specific to the model:

- `Vector Initialize()` should return the initial condition vector.
- `Matrix Stoichiometry()` should return the stoichiometry matrix with $R \times N$ elements.
- `Vector Propensity(const Vector& x)` should compute propensity functions of all the reactions given the current state vector.

StochKit only defines interfaces for these functions and how they are implemented is up to the developer of the model.

Prior to this work StochKit was not able to handle events. Therefore we added two more functions in `ProblemDefinition.cpp`:

- `unsigned long int triggerActivated(const Vector& x)` will determine whether the condition for firing each event is met and return a `long int` type variable with each bit representing an event. For example, return value "5" ($0 \ldots 0101$) means only the first and the third event should be triggered.
- `int eventAction(Vector& x, unsigned long int fireRecord, double t)` will execute the events according to the record returned by `triggerActivated()`.

In the main algorithm the following piece of code is executed at every simulation step to implement events.

```
const TriggerActivatedFunc triggerActivated = reactions.triggerActivated();
const EventActionFunc eventAction = reactions.eventAction();
int fireRecord_old = 0;
int fireRecord = triggerActivated(x);
while (fireRecord_old != fireRecord) {
    eventAction(x,fireRecord,t);
    fireRecord_old = fireRecord;
    fireRecord = triggerActivated(x);
}
```

Firing one event could trigger another event, therefore there is a "while loop" to make sure all possible events are executed.

5.2 Model File Translator

The JigCell main program reads and writes models in SBML. However, StochKit requires the model to be presented in a specialized C++ format. Thus, an SBML to StochKit model file (`ProblemDefinition.cpp`) translator is needed. There are two motivations for this: 1) It frees the modeler from manually writing model files for StochKit and prevents human errors. 2) It makes it possible to integrate StochKit as a simulator in JigCell so that models can be stochastically simulated directly from JigCell.

The translator takes two steps. The first step reads the SBML file, parses it and stores the model in memory as a model tree. The second step outputs the model tree into the target C++ file (`ProblemDefinition.cpp`).

The first step is done using the jigcell.sbml2 package developed by previous JigCell group members [2,3]. This package can parse models conforming to the SBML level 2 standard. It has corresponding classes for every component in SBML such as Model, Reaction, Species etc. The model is stored in the memory with the same structure as it is in the SBML file.

In the second step the program will look through the model in memory and collect all the necessary information to generate needed functions in the C++ file. The function definitions in SBML are translated to inline C++ functions to be used in computing propensity functions. `Initialize()` is generated from species definitions. `Stoichiometry()` is generated from reaction definitions. `Propensity()` is generated by combining information from parameter, rule and reaction definitions in SBML. Event definitions in SBML are used to generate two event functions in the C++ file.

5.3 Integrating StochKit into JigCell

StochKit is integrated as a simulator in JigCell's Run Manager (JCRM). StochKit's main program has been compiled into libraries and placed under JCRM's directory. The translator is called to generate `ProblemDefinition.cpp`, and then JCRM issues shell commands to compile StochKit's driver program and execute the driver to do simulations. The output of StochKit's simulation is then read into JCRM and displayed in a plot panel.

Figure 5.2 shows StochKit as a selectable simulator in JCRM. Figure 5.3 shows the result of simulating the simplified model in JCRM. The simulation was done using the model directly converted by JCMB. Besides generating a single simulation trajectory, StochKit can also collect ensemble simulation results by doing multiple simulations. Figure 5.4 shows the ensemble result of 1000 simulations. Although it is possible to generate ensemble results in JCRM, it takes a much longer time to simulate than a single simulation. Thus while the internal support for stochastic simulation in JigCell is handy to use during model development stage, for serious simulations where thousands of runs are required to get good quality statistical data, parallel computation is needed.

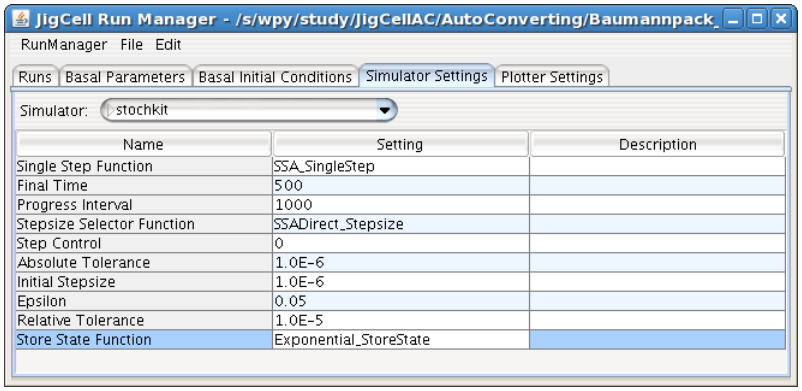


Figure 5.2: StochKit as a stochastic simulator in JCRM. Configuration options of the simulator are listed on the interface for the user to choose.

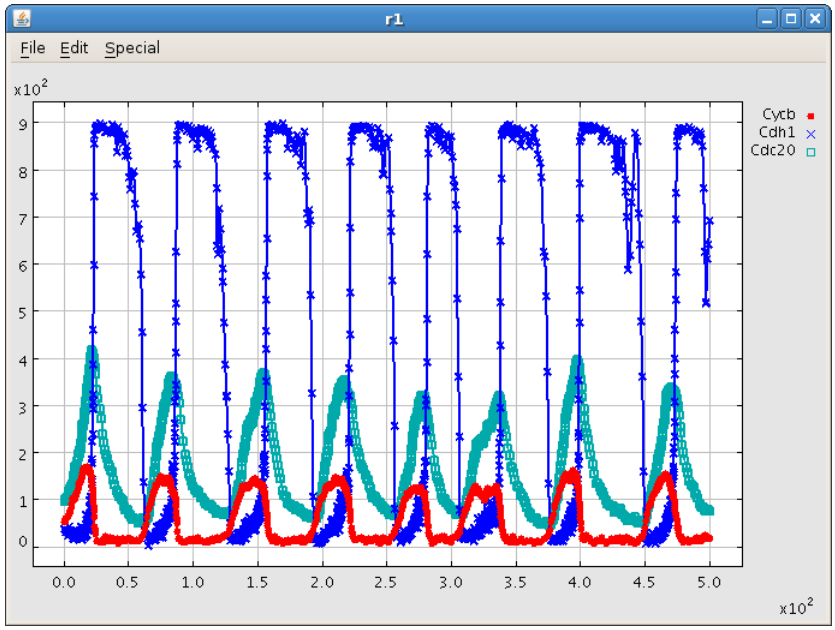


Figure 5.3: Displaying the stochastic simulation result in JCRM. The X axis is time and the Y axis is species population.

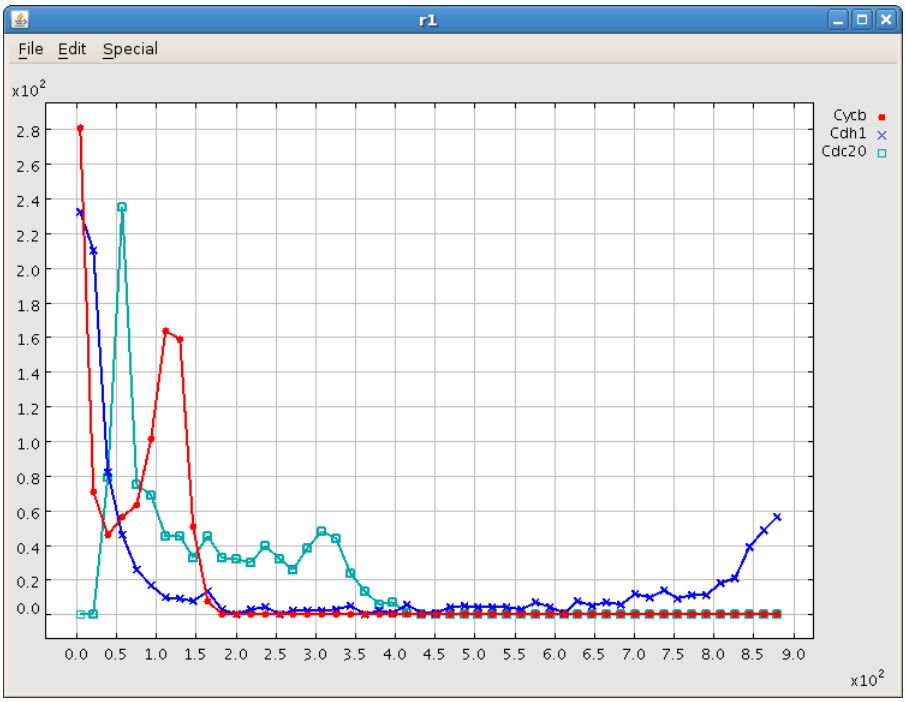


Figure 5.4: Displaying the ensemble simulation result in JCRM. This figure shows the distribution of species' populations at time = 200 min. The X axis is species population and the Y axis is frequency.

Chapter 6

Stochastic Simulation and Analysis

In this chapter we will first report the simulation results for two cell cycle models run on Virginia Tech's parallel supercomputer System X. The first is the simplified cell cycle model introduced above. The second is a full-sized budding yeast cell cycle model [9] consisting of around one hundred reactions. Both models are converted directly from the corresponding deterministic models. We then introduce our initial work on developing more realistic stochastic models and report simulation results on some mutants that are of great biological interest.

We did experiments using the basic SSA algorithm, which is considered an exact stochastic simulation method [14]. The basic SSA is usually slow compared with advanced ones. However, so far as we know, advanced simulation algorithms based on SSA will accelerate the simulation speed by less than an order of magnitude. So our simulation results give an indication of the computational resources needed to do stochastic simulation on the cell cycle model. For Gillespie's algorithm, the space complexity is $O(R)$, where R is the number of reactions.

StochKit can generate the trajectory of a single simulation or an ensemble result of many simulations. A single trajectory might be used to see the dynamics of the model, and the ensemble result is used to collect statistical data such as a histogram of the value of some variable at a certain time point. Any two cells that do not have a descendant relationship can be simulated simultaneously. In our ensemble simulations, we simulate a batch of cells starting from the same initial condition, so it can be easily paralled.

6.1 Events

In SBML, one can define events that change the state of the system. Events are triggered when some condition is met. For our simplified model, the mass of the cell is fixed and there are no events. But the full-sized model includes growing mass, and there are events defined to divide the cell or mark checkpoints within the cell cycle stages. Events for the deterministic model and corresponding stochastic model should match in terms of their biological meanings. However, there are differences due to the random nature of stochastic simulation, as illustrated in the following example. A typical deterministic event occurs when the value of

some variable is rising across a threshold:

```
if (A > threshold)
  then {event is triggered}
```

Here ">" means rising across a threshold, which is recognized at the right time point when the trigger condition makes the transition in value from "false" to "true".

To make the event work successfully in a stochastic simulation, it has to be rewritten to tolerate the situation where the value of A oscillates around the threshold. We did this by adding another threshold. For example:

```
if (A < minimum)
  then {minimum = A}
if (minimum < certain_low_value AND A > threshold)
  then {event is triggered; minimum = A}
```

Variable *minimum* is used to record the minimum value of A in the past, and is refreshed after every occurrence of the event. The condition for triggering the event is modified by additionally requiring the historical minimum value of A to be low enough. The idea is to ask for the value of A rising truly from a low level, not happening to pass the threshold by random oscillations.

In addition, the current version of SBML does not support using random numbers to describe random events. SBML uses a subset of MathML 2.0 [28] to represent mathematical expressions in the model, including event assignments. However, current MathML specification does not support describing random distributions yet. Thus event assignments cannot have random numbers. Therefore, our initial version of the stochastic cell cycle model has to divide the cell in a fixed fraction. Except for manually editing the events, i.e., adding extra thresholds as trigger conditions as exemplified above, all other parts of such a complex model are automatically converted by JigCell, including all the reactions, rules, species, parameters and their units.

6.2 Simulations on Directly Converted Models

We note that the directly converted model does not yet represent a fully realistic stochastic model, because 1) usually a realistic stochastic model also needs changes on the reaction network itself, such as adding extrinsic noise and synthesis and degradation terms for species whose values were conserved in the deterministic model, and 2) we haven't collected all the scaling factors (characteristic concentrations) for our models. However, we do have most of the scaling factors for the full-sized model, which are calculated from the data given in [10] and [13]. Among the 47 species in our model (a different activation status for a protein is considered a different species), five of them are actually introduced auxiliary variables (*BUD*, *IE*, *IEP*, *ORI*, *SPN*). *BUD*, *ORI* and *SPN* are introduced to represent signals that control the timing of bud emergence, onset of DNA synthesis, and cell separation. *IE* and *IEP* are two forms of a proposed intermediary enzyme

to facilitate the design of the network dynamics [9]. In addition, we do not yet know the characteristic concentrations for four of the species (*Cdh1*, *Cdh1i*, *Bub2* and *PPX*). For these nine species we arbitrarily set their scaling factors as 100, but all the others are computed with the unit information found in the published literature. The volume of the cell is set to be $50fL$. Some compromises were made during calculation because there is no simple one-to-one relation between our model and the published data. For example, $Cdc20$'s characteristic concentration is calculated using the value of mammalian cells, because no measurement was found in yeast. For another example, [13] found on the average $Cdc14$ is more abundant than $Net1$ in the cell. However, in our model we require that $Net1$ being slightly more abundant than $Cdc14$ (to keep $Cdc14$ inactive most of the time), so in our model we let them have same characteristic concentrations. But all in all, this is the most detailed stochastic model on the cell cycle mechanism. With such a model we can start to compare stochastic simulation results with experimental data.

The points of experiments in this section are 1) validate the (automatic) conversion of deterministic models to stochastic models, which is a necessary step in building realistic stochastic models, 2) demonstrate the computational resources needed to simulate these significant stochastic models and 3) explore statistical features in the simulations.

For the simplified model, we did 10,000 independent simulations of a cell starting from the same initial point (each simulation is called a run), and collected the state of the model at 200 minutes of simulation time. Figure 6.1 shows the distribution of the end point state. The stochastic simulation provides a much richer description of the cell state at this time than a deterministic simulation, which would produce a single value for each species. We used 100 worker processors (2.3 GHz PowerPC 970FX) in parallel, each doing 100 simulation runs. The total time for all the worker processors was 12,305 seconds. The simulation was well balanced. Ideally it should take $12305/100 = 123$ seconds of wall clock time. In reality the whole ensemble simulations took 145 seconds.

The full-sized model represents the current state-of-the-art in cell cycle simulation. We first compare the trajectories for deterministic and stochastic simulations on the converted model (see Figure 6.2 and 6.3). Conversion to actual populations allows the modeler to examine the performance of the model not only in terms of abstract dynamics but also in real population numbers that can be compared to experimental data for easier model verification. The figures show representative species of the model as well as the mass. The two formulations match well. Species populations range from below 100 molecules to over 10,000.

To examine statistical features, we did 10,000 independent simulations of the full-sized model starting from the same initial point, and collected the mass at birth after the third cell division. We discarded the first two divisions to winnow out the influence of the fixed initial condition. If the data is collected too early, the same fixed initial condition for all simulations might still have some global influence to the simulation results, and hereby introduce some unnecessary correlations between stochastic simulation results. Figure 6.4 shows the distribution of mass at birth. The average mass at birth is 1.20, compared with 1.21 from the deterministic simulation (both numbers are normalized). Coefficient of variation (CV) measures the dispersion of data points in a data series around the mean, and is defined as $\frac{\text{standard deviation}}{\text{mean}}$. In the experiment, CV of

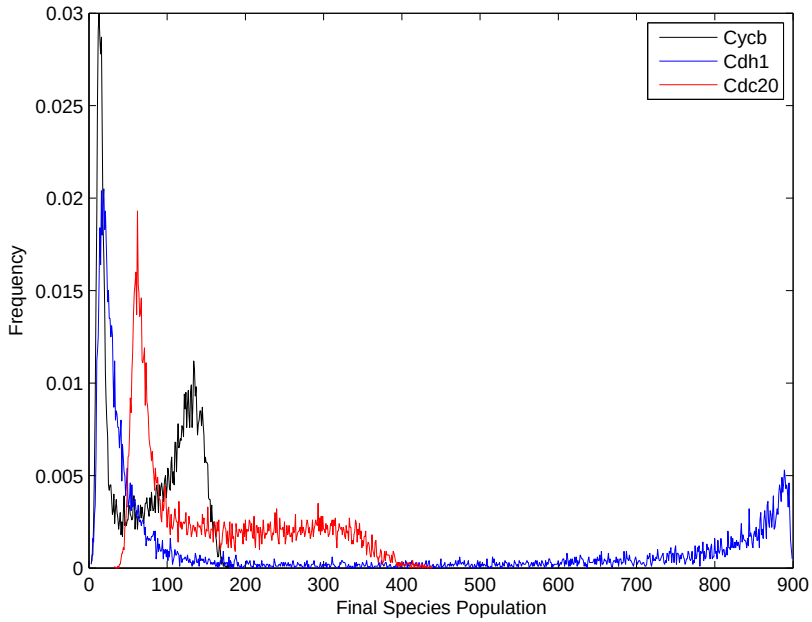


Figure 6.1: End point distribution for the simplified model

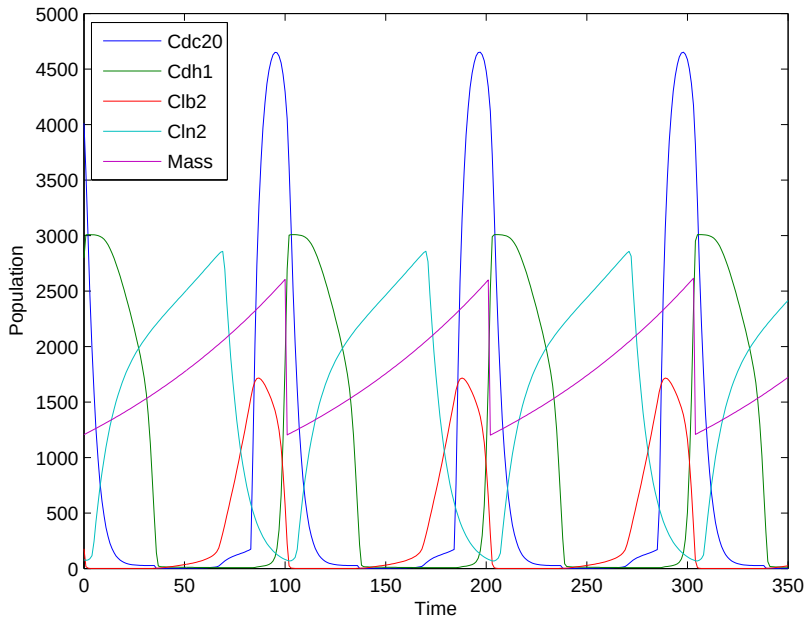


Figure 6.2: Deterministic simulation result of the converted full-sized model. (Mass is scaled 1000 times.)

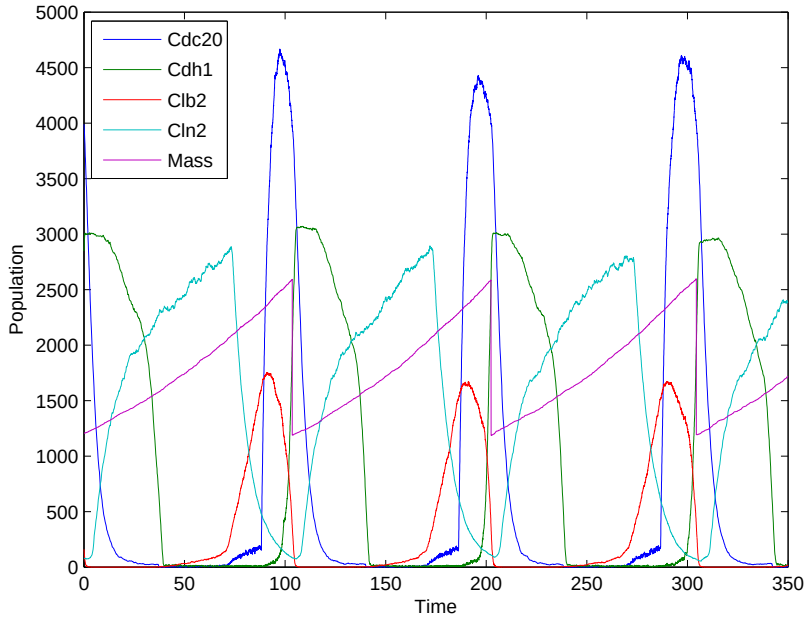


Figure 6.3: Stochastic simulation result of the converted full-sized model. (Mass is scaled 1000 times.)

mass at birth is 2.96%. Figure 6.5 shows the distribution of cycle length (the time between a cell's birth and division). Mean of cycle length is 101.3 min, compared with 102.0 from the deterministic simulation. CV of cycle length is 3.70%. The reason why we care about mass and cycle length is that they have biological significance and their wet lab experimental counterparts are easy to measure. The whole ensemble simulations took 3862 seconds of wall time. We used 100 worker processors in parallel, each doing 100 simulation runs. The total time for all the worker processors were 382,267 seconds.

Both the characteristic concentration and cell volume provided to the model are approximate. As a result, the estimated population of the stochastic model is not very accurate. We performed a sensitivity analysis to see how changes in the population of the system affect the simulation result. Our method is to uniformly scale the population of every species. Figure 6.6 illustrates how the result will change with population. The average of mass at birth (not drawn) is quite stable. As the population becomes larger, the average decreases extremely slowly from 1.21 to 1.20. Population has a larger impact on CV, which drops rapidly until the population reaches about 50%, after which it decreases slowly. Time spent on simulation grows linearly as the population gets larger. This result suggests the possibility that when the population of the system is very large one could save time by cutting the population while not hurting the statistical characteristics very much.

StochKit can be configured to use the standard C Library random number generator random() or the third-party random number generator SPRNG2.0 [27] (which is used by default in our experiments). We found that these two generators will produce different simulation results. Compared with SPRNG, random() produces a long tail of outliers, which represents some rare events (Figure 6.7). The average of mass is 1.21

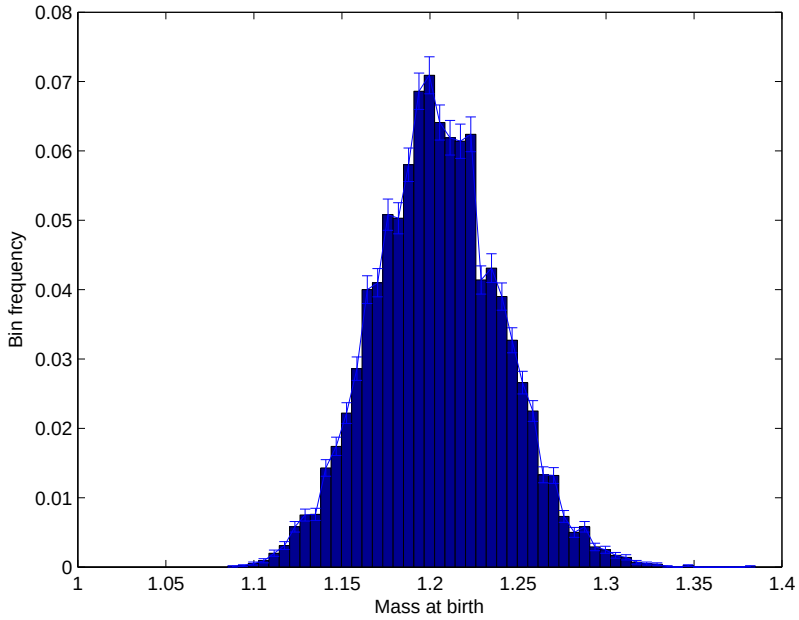


Figure 6.4: Histogram of mass at birth of the full-sized model.

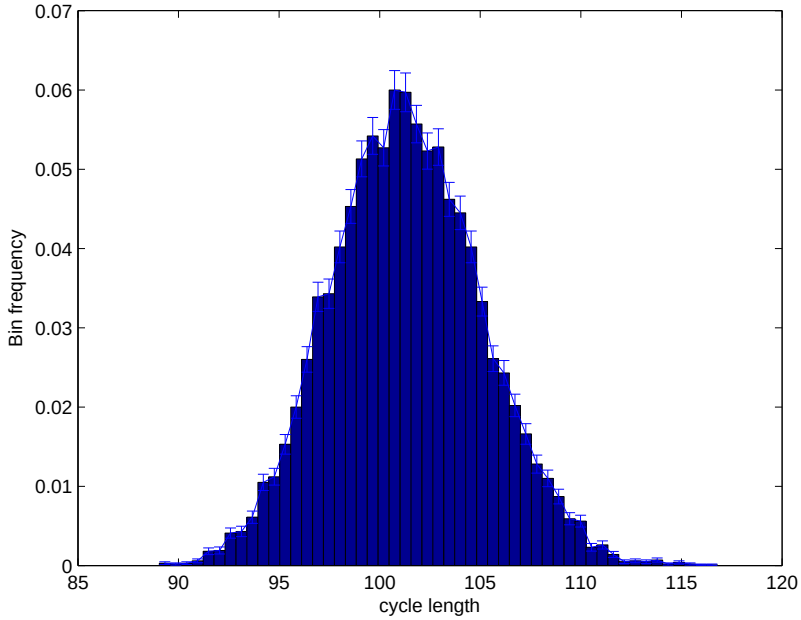


Figure 6.5: Histogram of cycle length of the full-sized model.

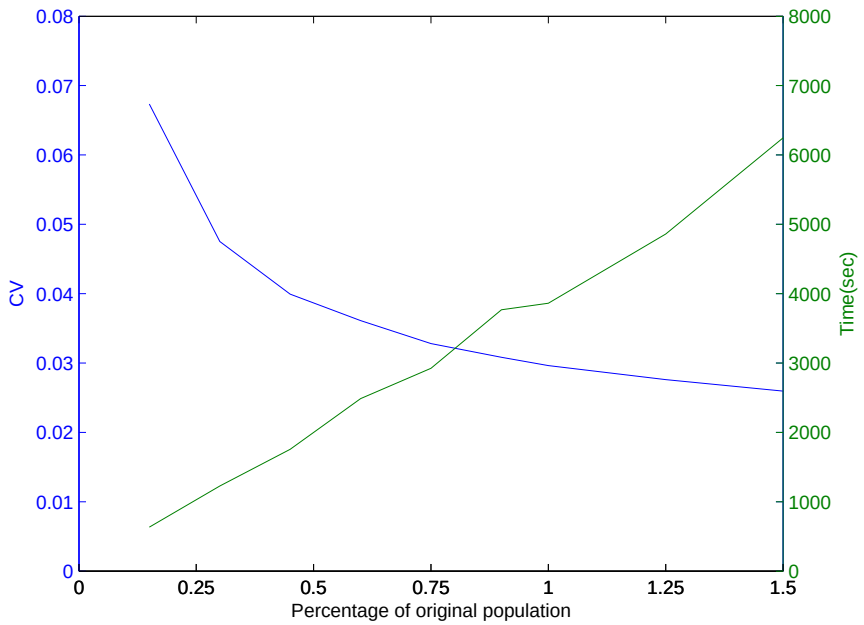


Figure 6.6: Coefficient of Variance of mass at birth and time spent on simulation on different populations.

Model	Stochastic Time			Deterministic Time
	Wall	Total	Avg./run	
Simplified	145	12305	1.23	0.029
Full-sized	3862	382267	38.2	0.311

Table 6.1: Time for stochastic and deterministic simulation (measured in seconds). For stochastic time columns, "Wall" means wall time for the simulation, "Total" means sum of times for all the worker processors, "Avg./run" is defined as $\frac{\text{total time}}{\text{number of runs}}$.

which is almost the same, but CV of mass at birth for random() becomes 6.02%, which is larger than for SPRNG. One might wonder whether SPRNG generates these same events after running long enough. We have done dozens of experiments on each random number generator with 10,000 runs for each simulation. Every time, random() will produce the outlier events, but SPRNG never has.

Table 6.1 shows timings for both versions of the cell cycle model, for both deterministic and stochastic simulations. LSODAR is used as the integrator for the deterministic simulation, and was run on an Intel Pentium IV 2.6 GHz CPU. The deterministic simulations use the same stopping criteria as the stochastic simulations (200 time units, or the third division). In these simulations, both the deterministic and stochastic models describe the same reaction network. Thus, the comparisons give some indication for the difference in the computational resources needed. Not surprisingly, even a single run of the stochastic simulation (run on a single processor of System X) takes much more time than the deterministic simulation. However, multiple

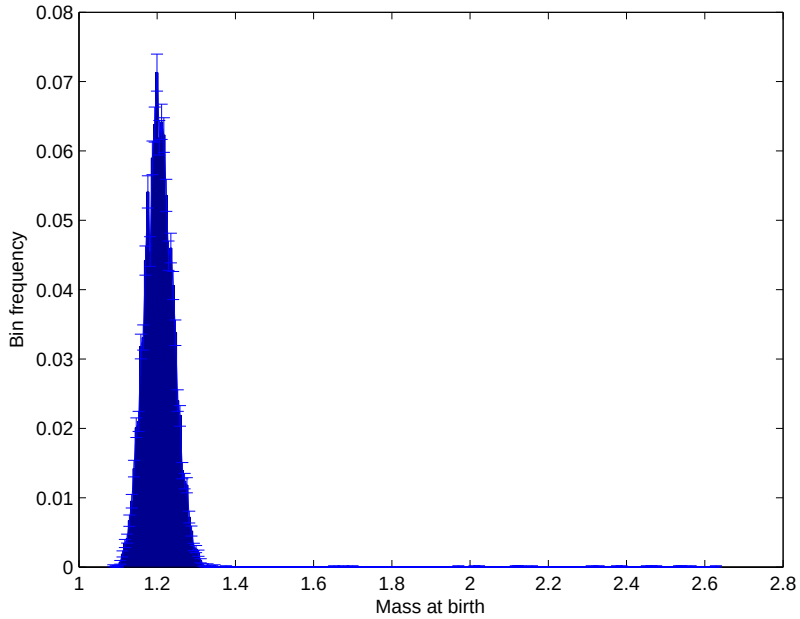


Figure 6.7: Histogram for mass at birth of the full-sized model generated using native random(). Figure is plotted using the same bin size as Figure 6.4. Notice the small number of outliers to the right side of the peak

runs of the stochastic simulation (to gather ensemble statistics) can be computed in parallel.

6.3 Improvements on the Directly Converted Model

As mentioned in the last section, the directly converted model is not fully realistic. We can take steps to make the model more biologically meaningful. Improvements to the model are applied consecutively in the order introduced below.

The first revision of the model happened when we found scaling factors for some species might be too high, which resulted in too large populations. Following biologists' suggestion we reduced certain (five in total) species' scaling factors. Three introduced variables (*BUD, ORI, SPN*) had their scaling factors reduced from 100 to 10. *Cdc20*'s scaling factor was changed from 300 to 150, and *Mad2*'s scaling factor was changed from 400 to 40. These changes resulted in a little bit of growth in the CV of mass at birth from 2.96% to 3.05% and the CV of the cycle length from 3.70% to 3.83%. There was no significant change in the average values. However, total simulation time was reduced from 382,267 seconds to 232,744 seconds.

Our second improvement focused on the growing volume. The deterministic model does not track volume. Instead, there is *mass* to represent the effect of the size of the cell. When converting to the stochastic model, we kept mass while adding volume. Volume in the original stochastic model was fixed. Therefore, in the directly converted model there were growing mass and fixed volume. However, mass and volume should

vary similarly. A realistic stochastic model should have a growing volume. Furthermore, according to biologists, the growth of volume is much more deterministic than the fluctuation of species' populations. Thus the growth of volume should be deterministic (in the directly converted model the growth of mass is simulated stochastically). Replacing mass with volume requires a small modification to the model. As stated in Equation 3.5, if the volume is changing there is an extra non-zero term $[S]\frac{dV}{dt}$ for the converted reaction rate in population. Assuming that volume is growing in an exponential way, i.e., $\frac{dV}{dt} = \mu V$, we get

$$[S]\frac{dV}{dt} = [S]\mu V = \mu N_S. \tag{6.1}$$

Thus, there is the addition of a source term, μN_S for a species' ODE. If the species we are considering has synthesis and degradation terms, then this source term can be lumped with the degradation term. For example:

$$\frac{d[S]}{dt} = k_s - k_d[S] \Rightarrow \frac{dN_S}{dt} = k_s V - (k_d - \mu)N_S. \tag{6.2}$$

In practice, we modified the model in the same way, i.e., we diluted the degradation rate constants by the volume growth rate constant μ to keep population growing with the volume.

To implement deterministic volume growth in a stochastic simulation we have to modify the simulation algorithm. In the classical Gillespie's SSA, the volume of the system is assumed to be fixed. However, if the volume is changing over time then the rate constant c, as defined in (2.11), becomes a time dependent variable because c is calculated from k using V (see Section 2.3). [25] suggested modifications to the SSA to accommodate varying volume, which keeps the framework of the SSA but requires evaluation of some functions that are hard to compute. An alternative approximate approach is to keep the whole SSA algorithm and update volume and rate constants only after each reaction event. In other words, volume and rate constants are not changed between two simulation steps. In [25] the authors suggested that if the growth rate of volume is much slower than the fastest chemical reaction, then the approximate approach will be accurate enough. In our model the maximum step size τ is around 0.003 minutes and during this time volume only changes 0.0024%. Therefore, we think the approximate approach to growing volume is accurate enough for our simulation. Note that now the volume V is one of the variables used to evaluate the propensity function a_j at each simulation step. The revised SSA algorithm with approximate volume growing is listed below, denoting $Y = (X, V)$:

1. With the system in state Y at time t, evaluate $a_0(Y) \equiv \sum_{j'=1}^{M} a_{j'}(Y)$.

2. Draw two unit-interval uniform random numbers r_1 and r_2, and compute τ and j according to

   - $\tau = \frac{1}{a_0(Y)} \ln(\frac{1}{r_1})$
   - $j =$ the *smallest integer* satisfying $\sum_{j'=1}^{j} a_{j'}(Y) > r_2 a_0(Y)$.

3. Replace $t \leftarrow t + \tau$, $X \leftarrow X + \alpha_j$ and $V \leftarrow V + \mu V \tau$ (since $\frac{dV}{dt} = \mu V$).

4. Record (Y, t). Return to Step 1, or else end the simulation.

This simulation yields a CV of mass at birth $= 2.87\%$ and a CV of cycle length $= 3.94\%$. Note that now the mass is calculated from volume. Again, average values of these variables are not affected significantly.

Our next improvement was to divide the cell in a random manner which is what happens in nature. In the previous model, when the cell divides, both the volume and every species are divided deterministically by multiplying the value before division with a fixed fraction F (which is around but not exactly equal to 0.5). Pseudo code for this simple formulation is:

```
V = V*F;  //volume division
S = S*F;  //species division
```

In the new version, volume will be multiplied with a Gaussian random number with a mean of F and a CV of 0.05. During the division, a more realistic model should decide which molecule will go to which daughter cell through a Bernoulli trial. Thus in the new model the value after division for each species is gotten from a binomial distribution with number of trials equal to the value before division and probability for each trial equal to the Gaussian random number drawn for division. Pseudo code for a stochastic formulation is therefore:

```
F = F*N(1,0.05^2);
V = V*F;      //volume division
S = B(S,F);   //species division
```

where N is the normal distribution RNG function and B is a binomial distribution RNG function. The addition of random numbers is not supported by SBML, so we did it directly on the C++ model file for StochKit. After this revision, CV of mass at birth becomes 6.16% and CV of cycle length 5.86%. Average values are not affected significantly. Notice that without random division, CV of mass at birth is always smaller than CV of cycle length, which conforms to real world experiments. However, after adding random division, CV of mass at birth is larger than CV of cycle length, which is expected because we are directly enforcing randomness on the divided mass. To winnow out the effect of this directly introduced randomness, we examined the mass before division, whose CV is 3.55%. We think the mass before division will be more comparable to experimental data.

After these improvements, we think the stochastic model is more realistic. It is the most detailed and realistic stochastic model on cell cycling that has been developed so far. The statistical data of mass and cycle length conforms reasonably well with experimental data.

6.4 Simulations on Mutants

The power of a successful model is not only in how well it can explain a single wild type strain but also in how well it can explain various mutants. A mutant is a change in genes from the wild type modeled by

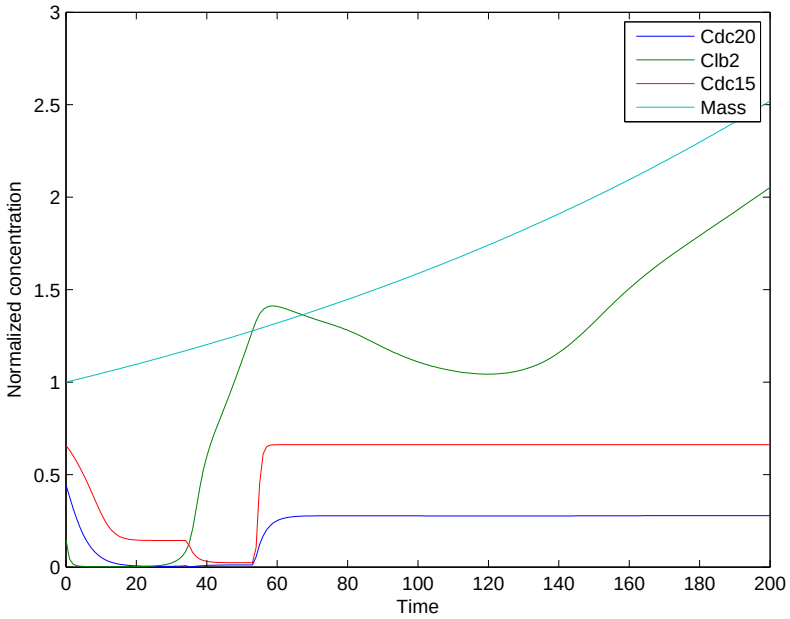


Figure 6.8: Time series deterministic simulation result on mutant *APC-A cdh1*Δ *in galactose*.

changing some parameters' values and some initial conditions. However the underlying reaction network and mechanism are kept the same. The original deterministic model can conform to the phenotypes of over 100 mutant strains. For 120 of 131 mutants, the model agrees well with observations. For stochastic simulations, we haven't done extensive experiments on every mutant tested before, but we have tested several interesting mutants because biologists think their behaviors are on the edge of being viable or inviable. Since deterministic simulations can only give a binary answer, modelers are curious to see their stochastic behaviors.

We have tested eight mutants so far. They are *cdh1*Δ*, APC-A, CLB2*Δ*, APC-A cdh1*Δ*, CLB2-db*Δ *clb5*Δ*, CLB1 clb2*Δ *cdh1*Δ*, APC-A sic1*Δ*, cdc20*Δ *pds1*Δ. For detailed information on these mutants, please refer to [6]. For some of these mutants whose phenotypes are different when growing in different mediums, such as glucose (default medium) or galactose, we did simulations modeling different mediums too, which is actually a matter of changing the parameter for "mass doubling time" (mdt) in the model. Among these mutants, two of them are particularly interesting and will demonstrate the power of stochastic simulations.

The first mutant is *APC-A cdh1*Δ *in galactose*. Experimental data suggests that this mutant should have partial viability. The viability of a mutant is defined by its ability to spawn a colony with more than a thousand cells from a single ancestor cell during a period of time. However, deterministic simulation predicts absolutely inviable—the cell cannot divide at all. Figure 6.8 shows the time series plot of the deterministic simulation.

Surprisingly, stochastic simulations on the same mutant told some different stories. Figure 6.9 shows two

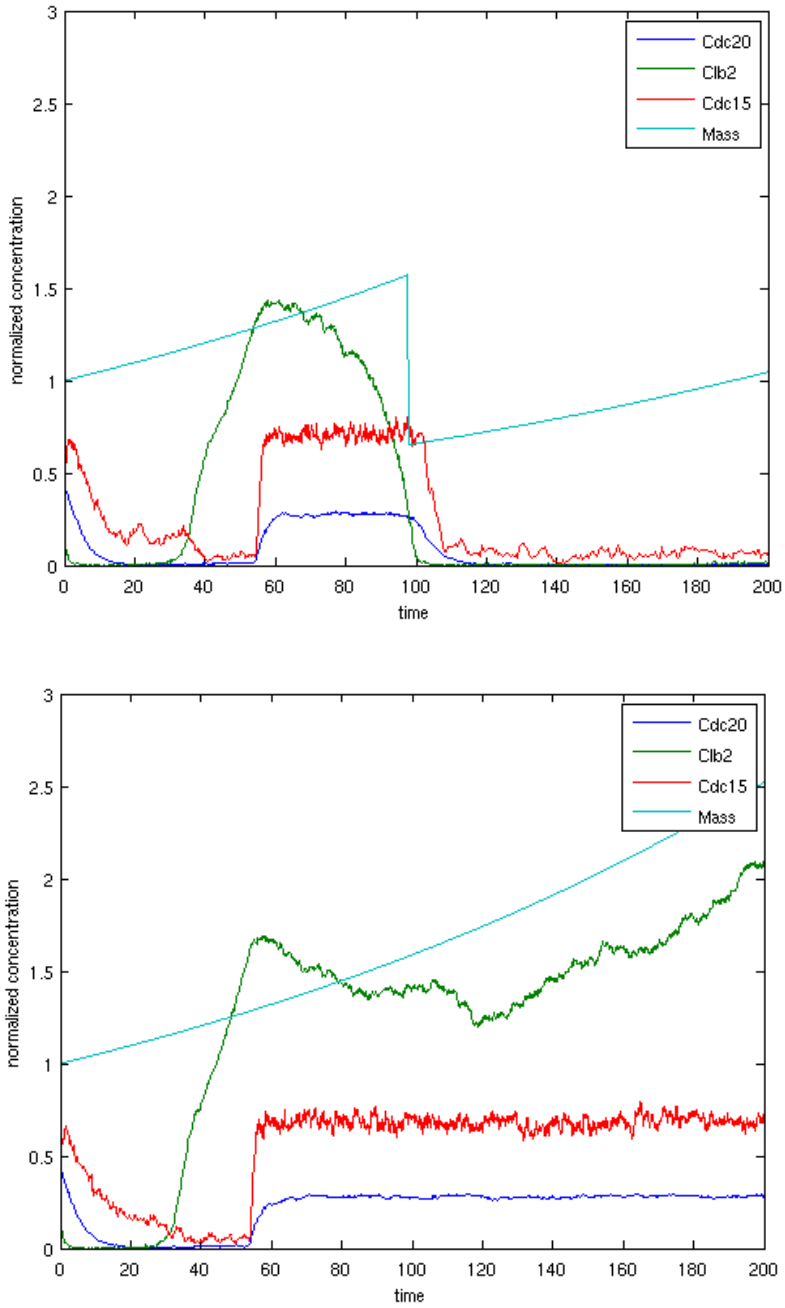


Figure 6.9: Time series stochastic simulation results on mutant *APC-A cdh1*Δ *in galactose*. The top and bottom plots are two instances of simulations on the same mutant. They displayed dramatical differences in their behaviors.

instances of stochastic simulations on exactly the same mutant, with one of them being not able to divide and the other divided at least once.

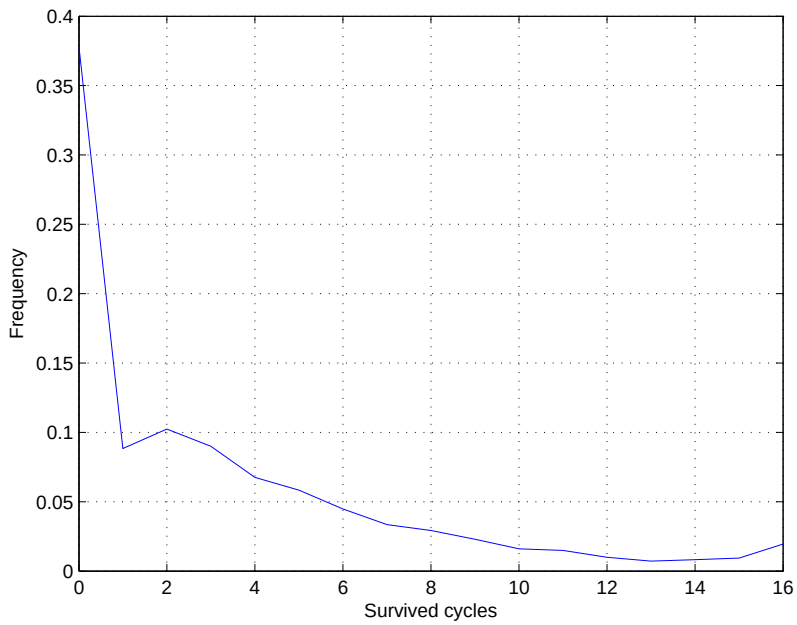


Figure 6.10: Survived cycles of mutant *APC-A cdh1*Δ *in galactose*. Ideally we should do the simulation as long as there is still a positive frequency of survived cycles. In reality we chose to set a cut off time at 3000 minutes to save simulation resource, which we think is long enough because the frequency at that time is already very close to zero. Thus cells who could still divide after the cut off time will be counted in the frequency for the number of cycles they have achieve before cut off, but there are only a small number of them.

A collection of 10,000 simulations revealed some statistical data for this mutant. Figure 6.10 plots the frequency of survived cycles for the mutant. Survived cycles are how many divisions the cell achieved before it ceased division. A cell dies when it ceases to divide and its mass grows over 5. Contrary to the deterministic simulation, different cells (different simulations) behave dramatically differently. Some cells died without any division and others died after 1, 2 or more divisions. This suggests the possibility for this mutant to be viable—a result closer to the real world data than the deterministic simulation result.

Let $F(n)$ be the frequency of surviving n cycles before ceasing division. The backward cumulative frequency $C(n)$ is defined as

$$C(n) = \sum_{n}^{\infty} F(n). \tag{6.3}$$

Figure 6.11 shows the backward cumulative frequency of the data in Figure 6.10 and plots the cumulative frequency on a logarithmic scale. A number n on the X axis should be interpreted as "survived n or more cycles". Biologically it means how many cells are still alive after that many divisions. Note the beginning of the curve is not very smooth, which we think is due to the effect of initial conditions. This number is distributed in an exponential curve, which implies that there is a fixed probability for a cell to continue

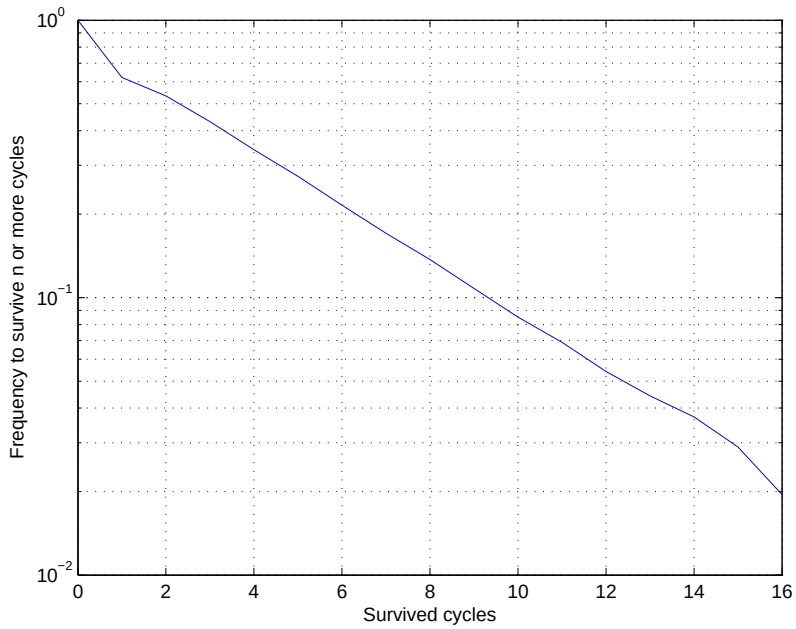


Figure 6.11: Backward cumulative of survived cycle frequency of mutant *APC-A cdh1*Δ *in galactose*.

division or stop division.

The second mutant is *CLB2-db*Δ *clb5*Δ *in galactose*. For this mutant, deterministic simulation predicts viability, and every cell will divide indefinitely. In stochastic simulation, however, different cells again behave differently. Figure 6.12 shows frequency of survived cycles. Figure 6.13 plots percentage of alive cells after each division. The distribution is an exponential curve.

Not all mutants' stochastic simulations are different from their deterministic counterparts. In the eight mutants we simulated so far, only the two mentioned above show significant differences from the deterministic simulation.

The possibility of viability for a mutant in the stochastic simulation can be calculated from the backward cumulative frequency data, which can then be compared with real world experimental data. This level of verification on the model can not be achieved through a deterministic model. While simulating on a wild type cell doesn't give much surprise, the power and usefulness of stochastic simulations are well demonstrated in simulating these mutants.

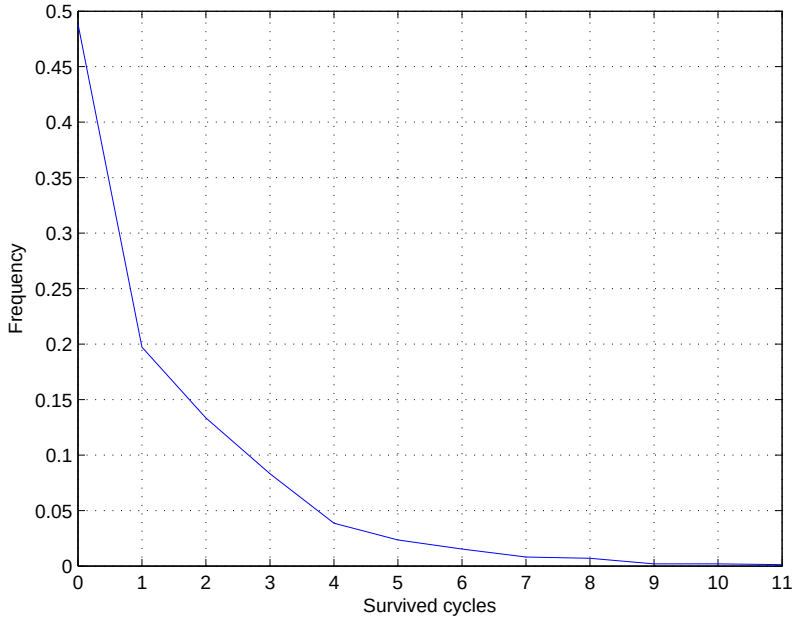


Figure 6.12: Survived cycles of mutant *CLB2-db*Δ *clb5*Δ *in galactose*

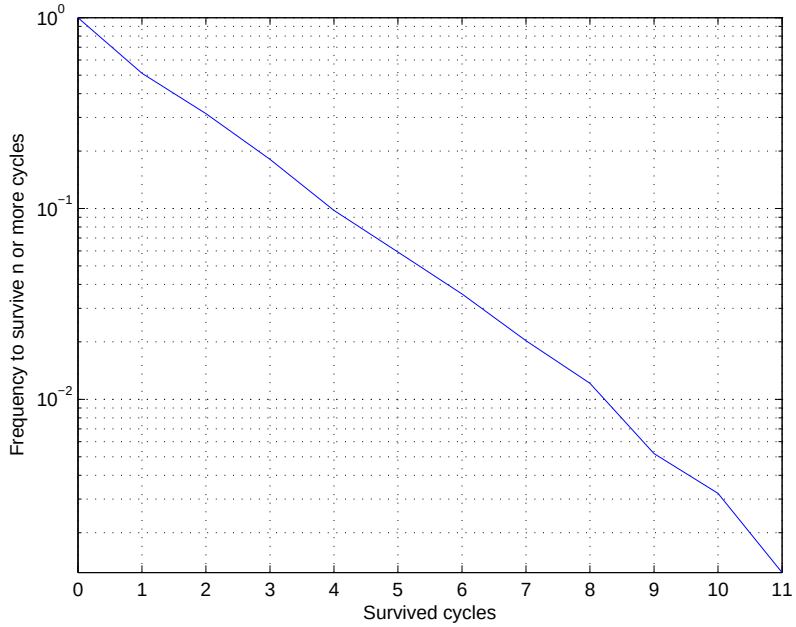


Figure 6.13: Backward cumulative of survived cycle frequency of mutant *CLB2-db*Δ *clb5*Δ *in galactose*

Chapter 7

Conclusions and Future Work

We investigated the relation between deterministic and stochastic formulations of molecular regulatory networks. Prior to this work modelers have to manually scale and convert a deterministic model in normalized concentration to a stochastic model in population. This procedure is tedious, error prone and even prohibitive to perform by hand for large and complex systems. The major contribution of this work in computer science is to introduce an automatic conversion process by analyzing units in the system, which greatly reduced the work load of the modeler. Our work also highlights the usefulness of supporting flexible unit definitions, which is usually neglected in current modeling tools.

It has been an open question for some time whether there are true differences between the "deterministic form" and the "stochastic form" of a model. Our work on conversion of models shows that a fully specified model contains the information necessary to perform both stochastic and deterministic simulations, and that traditionally modelers have merely taken shortcuts when defining the deterministic form. As this thesis has explained, much of our "conversion process" is in fact a requirement for full specification of all details of the model, including precise specification of units.

Support for stochastic simulation has been established in the JigCell modeling environment. This includes integration of a stochastic simulator and the addition of unit checking and model conversion functions. It requires little expertise in biology or modeling to perform the conversion. With the help of JigCell, stochastic cell cycle models were developed. The directly converted model is further improved by tuning characteristic concentrations, supporting volume growth and enabling random cell division.

We have performed the first stochastic simulations of a full-sized quasi-realistic cell cycle model that we are aware of, which demonstrates the usefulness of our technique. We collected runtime characteristics in simulating this significant model to help future development and simulation of molecular regulatory network models. Additionally, results for several mutants reveal the unique and important role of stochastic simulation in studying the noisy molecular level behavior of cell cycles.

In future work, more simulations and analysis of mutants should provide some insight into the behaviors of stochastic simulations. More work could be done to make the model and simulation more realistic. For

example, we plan to introduce mRNA in the model to make the reaction network more complete. We expect that the introduction of mRNA will bring more noise to the model and make it match better with experimental data. For another example, our current simulation disregards the mother cell and only tracks the daughter cell after each division. A more realistic simulation would simulate and collect statistics of every cell that is born during the simulation. Lastly, to what extent a modeling environment can help modelers to process models is an interesting and open issue.

Bibliography

[1] David Adalsteinsson, David McMillen, and Timothy Elston. Biochemical Network Stochastic Simulator (BioNetS): software for stochastic modeling of biochemical networks. *BMC Bioinformatics*, 5(1):24, 2004.

[2] N.A. Allen, L. Calzone, K.C. Chen, A. Ciliberto, N. Ramakrishnan, C.A. Shaffer, J.C. Sible, J.J. Tyson, M.T. Vass, L.T. Watson, and J.W. Zwolak. Modeling regulatory networks at Virginia Tech. *OMICS, J. of Integrative Biol.*, 7:285–299, 2003.

[3] Nicholas A. Allen, Clifford A. Shaffer, Naren Ramakrishnan, Marc T. Vass, and Layne T. Watson. Improving the development process for eukaryotic cell cycle models with a modeling support environment. *Simulation*, 79(12):674–688, 2003.

[4] G. I. Barenblatt. *Scaling*. Cambridge University Press, 2003.

[5] Jos A. M. Borghans, Rob J. de Boer, and Lee A. Segel. Extending the quasi-steady state approximation by changing variables. *Bulletin of Mathematical Biology*, 58(1):43–63, January 1996.

[6] Budding yeast model website. `http://mpf.biol.vt.edu/research/budding_yeast_model/pp/`.

[7] Y. Cao, H. Li, and L. Petzold. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. *J. Chem. Phys.*, 121:4059–67, 2004.

[8] Yang Cao, Daniel T. Gillespie, and Linda R. Petzold. The slow-scale stochastic simulation algorithm. *The Journal of Chemical Physics*, 122(1):014116, 2005.

[9] K.C. Chen, L. Calzone, A. Csikasz-Nagy, F.R. Cross, B. Novak, and J.J. Tyson. Integrative analysis of cell cycle control in budding yeast. *Mol. Biol. Cell*, 15(8):3841–3862, 2004.

[10] Frederick R. Cross, Vincent Archambault, Mary Miller, and Martha Klovstad. Testing a mathematical model of the yeast cell cycle. *Mol. Biol. Cell*, 13(1):52–70, January 2002.

[11] Michael B. Elowitz, Arnold J. Levine, Eric D. Siggia, and Peter S. Swain. Stochastic gene expression in a single cell. *Science*, 297(5584):1183–1186, 2002.

[12] Hao Ge, Hong Qian, and Min Qian. Synchronized dynamics and non-equilibrium steady states in a stochastic yeast cell-cycle network. *Mathematical Biosciences*, 211:132–152, January 2008.

[13] Sina Ghaemmaghami, Won-Ki Huh, Kiowa Bower, Russell W. Howson, Archana Belle, Noah Dephoure, Erin K. O'Shea, and Jonathan S. Weissman. Global analysis of protein expression in yeast. *Nature*, 425:737–741, October 2003.

[14] D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81:2340, 1977.

[15] D.T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *J. Chem. Phys.*, 115:1716, 2001.

[16] Nicholas J. Guido, Xiao Wang, David Adalsteinsson, David McMillen, Jeff Hasty, Charles R. Cantor, Timothy C. Elston, and J. J. Collins. A bottom-up approach to gene regulation. *Nature*, 439:856–860, February 2006.

[17] Stefan Hoops, Sven Sahle, Ralph Gauges, Christine Lee, Jurgen Pahle, Natalia Simus, Mudita Singhal, Liang Xu, Pedro Mendes, and Ursula Kummer. COPASI–a COmplex PAthway SImulator. *Bioinformatics*, 22(24):3067–3074, 2006.

[18] M. Hucka, A. Finney, H.M. Sauro, and 40 additional authors. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinfo.*, 19(4):524–531, 2003.

[19] Michael Hucka, Andrew Finney, Stefan Hoops, Sarah Keating, and Nicolas Le Novre. Systems Biology Markup Language (SBML) Level 2: Structures and Facilities for Model Definitions. Available at `http://sbml.org/Documents/Specifications/SBML_Level_2/Version_3/`, 2007.

[20] Jigcell website. `http://jigcell.biol.vt.edu`.

[21] A.A. Julius, A. Halasz, M.S. Sakar, H. Rubin, V. Kumar, and G.J. Pappas. Stochastic modeling and control of biological systems: the lactose regulation system of *Escherichia Coli*. *IEEE Transactions on Automatic Control*, 53:51–65, January 2008.

[22] Minoru S. H. Ko. Problems and paradigms: Induction mechanism of a single gene molecule: Stochastic or deterministic? *BioEssays*, 14:341–346, February 1992.

[23] Fangting Li, Tao Long, Ying Lu, Qi Ouyang, and Chao Tang. The yeast cell-cycle network is robustly designed. *Proceedings of the National Academy of Sciences of the United States of America*, 101(14):4781–4786, 2004.

[24] H. Li, Y. Cao, L. Petzold, and D. Gillespie. Algorithms and software for stochastic simulation of biochemical reacting systems. *Biotechnology Progress*, 2007.

[25] T. Lu, L. Tsimring, and J. Hasty. Cellular growth and division in the Gillespie algorithm. *Systems Biology*, 1(1):121–128, 2004.

[26] G. Marlovits, C.J. Tyson, B. Novak, and J.J. Tyson. Modeling m-phase control in *xenopus* oocyte extracts: the surveillance mechanism for unreplicated dna. *Biophys. Chem.*, 72:169–184, 1998.

[27] M. Mascagni and A. Srinivasan. Algorithm 806: SPRNG: A scalable library for pseudorandom number generation. *ACM Trans. Math. Soft.*, 26:436–461, 2000.

[28] Mathematical Markup Language (MathML) Version 2.0 (Second Edition). `http://www.w3.org/TR/MathML2/`, 2003.

[29] Harley H. McAdams and Adam Arkin. Stochastic mechanisms in gene expression. *Proc. Natl. Acad. Sci. USA*, 94(3):814–819, February 1997.

[30] Harley H. McAdams and Adam Arkin. It's a noisy business! genetic regulation at the nanomolar scale. *Trends in Genetics*, 15:65–69, February 1999.

[31] Bela Novak, Zsuzsa Pataki, Andrea Ciliberto, and John J. Tyson. Mathematical model of the cell division cycle of fission yeast. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 11(1):277–286, 2001.

[32] Stephen RamseY, David Orrell, and Hamid Bolouri. Dizzy: Stochastic simulation of large-scale genetic regulatory networks. *Journal of Bioinformatics and Computational Biology*, 3:415–436, April 2005.

[33] Christopher V. Rao and Adam Arkin. Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the Gillespie algorithm. *The Journal of Chemical Physics*, 118(11):4999–5010, 2003.

[34] Grol M. Sel, Jordi Garcia-Ojalvo, Louisa M. Liberman, and Michael B. Elowitz. An excitable gene regulatory circuit induces transient cellular differentiation. *Nature*, 440:545–550, March 2006.

[35] John L. Spudich and D. E. Koshland Jr. Non-genetic individuality: chance in the single cell. *Nature*, 262:467–471, August 1976.

[36] Ralf Steuer. Effects of stochasticity in models of the cell cycle: from quantized cycle times to noise-induced oscillations. *Journal of Theoretical Biology*, 228:293–301, June 2004.

[37] Gautier Stoll, Jacques Rougemont, and Felix Naef. Few crucial links assure checkpoint efficiency in the yeast cell-cycle network. *Bioinformatics*, 22(20):2539–2546, 2006.

[38] Systems Biology Ontology. `http://biomodels.net/SBO/`.

[39] System X website. http://www.arc.vt.edu/arc/SystemX/.

[40] K. Takahashi. *Multi-algorithm, multi-timescale cell biology simulation*. PhD thesis, Keio University, Japan, January 2004.

[41] John J. Tyson, Kathy Chen, and Bela Novak. Network dynamics and cell physiology. *Nature Reviews Molecular Cell Biology*, 2:908–916, December 2001.

[42] John J. Tyson, Attila Csikasz-Nagy, and Bela Novak. The dynamics of cell cycle regulation. *BioEssays*, 24:1095–1109, 2002.

[43] M. Vass, C.A. Shaffer, N. Ramakrishnan, L.T. Watson, and J.J. Tyson. The JigCell Model Builder: a spreadsheet interface for creating biochemical reaction network models. *IEEE/ACM Trans. on Comp. Biol. and Bioinf.*, 2005.

[44] Virtual Cell home page. http://www.vcell.org/login/login.html.

[45] N.G. von Kampen. *Stochastic processes in physics and chemistry*. Elsevier, Amsterdam, third edition, 2007.

[46] Xiao Wang, Nan Hao, Henrik G. Dohlman, and Timothy C. Elston. Bistability, stochasticity, and oscillations in the mitogen-activated protein kinase cascade. *Biophysical Journal*, 90:1961–1978, March 2006.

[47] Ming Yi, Ya Jia, Jun Tang, Xuan Zhan, Lijian Yang, and Quan Liu. Theoretical study of mesoscopic stochastic mechanism and effects of finite size on cell cycle of fission yeast. *Physica A: Statistical Mechanics and its Applications*, 387:323–334, January 2008.

[48] Yuping Zhang, Minping Qian, Qi Ouyang, Minghua Deng, Fangting Li, and Chao Tang. Stochastic model of yeast cell-cycle network. *Physica D: Nonlinear Phenomena*, 219:35–39, July 2006.

[49] Yuping Zhang, Huan Yu, Minghua Deng, and Minping Qian. Nonequilibrium model for yeast cell cycle. *Proceedings of Computational Intelligence and Bioinformatics, Part III*, 4115:786–791, August 2006.

Appendix A

Characteristic Concentrations of Species in the Full-sized Model

Table A.1 lists characteristic concentrations (scaling factors) of species in the full-sized buddying yeast cell cycle model. Values in the brackets are what later used in the improved model.

Species	Characteristic Concentration (nM)	Species	Characteristic Concentration (nM)
$Cln2$	40	$Swi5$	57.5
$Clb2$	40	BUD	100 (10)
$Clb5$	40	$Bub2$	100
$Sic1$	40	$C2, C2_P, F2, F2_P$	40
$Cdc6$	40	$C5, C5_P, F5, F5_P$	40
$Cdc20$	300 (150)	$CKIT$	40
$Cdh1$	100	$Cln3$	40
$Cdc14$	18	IE, IE_P	100
$Net1$	18	$Mad2$	400 (40)
$Esp1$	3.3	ORI	100 (10)
$Pds1$	3.3	SPN	100 (10)
$Cdc15$	8	PE	3.3
$Lte1$	10	PPX	100
SBF	110	$RENT$	18
$Mcm1$	100	$TEM1GDP, TEM1GTP$	20

Table A.1: Characteristic concentrations of species in the full-sized model.