Computational Techniques and Tools for Systems Biology with Application to Lipidomics

Chuan Hock Koh[1,2] and Limsoon Wong[2]

[1]NUS Graduate School for Integrative Sciences and Engineering, Singapore 117597

[2]School of Computing, National University of Singapore, Computing Drive, Singapore 117417

ABSTRACT

Motivation:

Due to technological advancements in molecular biology such as genome sequencing and high-throughput measurements, there is increasing interest in system biology. It is postulated that it would be a necessary (if not mandatory) step to build an *in-silico* model of the pathway under investigation as part of a biologist's protocol in future. This is because there are numerous advantages in having a mathematical model such as rapid prediction of the effects of manipulating experiments' settings, reaffirming knowledge of the working of the pathways, hypothesis testing, to list a few. There are four main steps in building a model; viz., construction, verification, calibration and validation. The calibration step is where parameters such as concentration of molecules and reaction rates of the model are estimated. A huge amount of effort had already been invested into this important and challenging process. However, past and current works mainly focus on obtaining a simple optimal parameter set that best fit given a set of data, ignoring the robustness of biological systems and inaccuracies of experimental data. Another step that is gaining attention recently is the validation process where the model's behavior is checked to ensure that it conforms to a set of given properties. The automated validation process is becoming increasingly important as larger and more complex biological pathways are being modeled, which renders manual validation tedious if not impossible. Current works in this area are still immature, as they either waste resources (offline-based), is limited to a restricted class of models, or do not provide reliable results in every situation.

Result:

For the calibration process, we have built a pragmatic application (DA 1.0) that offers solutions to the problem of applying parameter estimation in practice. For example, to overcome the problem of limited experimental observed time-points due to current experimental techniques and/or project funding, DA 1.0 is built with the ability to increase the number of time-points by means of smoothing and re-sampling. More importantly, DA 1.0 serves as a foundation step for us to develop an approach to do parameters' distribution estimation instead of single optimal parameter estimation. The approach that we are currently developing will take into consideration the fact that biological systems are inherently robust, and that experimental data is noisy.

In the area of automated validation, we have proposed a new algorithm that always produces reliable (statistically backed) results and have incorporated that algorithm into a program (MIRACH 1.0). Furthermore, MIRACH deploys the more efficient online (on-the-fly) model checking approach and we have shown that the timesaving gains by using MIRACH easily surpasses 400% compared to the offline-based counterparts. Lastly, MIRACH is able to support any model written in the widely used SBML or CSML formats.

Conclusion:

Kell (2007) believes that good systems biology models of metabolism and metabolomics will be easier to build than gene or protein networks because changes in the metabolome is usually amplified relative to the transcriptome and the proteome. Hence, we are confident that tools we have developed would be able to provide assistance to the lipidomics project in Singapore.

1 INTRODUCTION

"To understand complex biological systems requires the integration of experimental and computational research – in other words a systems biology approach," (Kitano, 2002) The major reason for the increasing interest in systems biology is due to the technological advancement in molecular biology such as genome sequencing and high-throughput measurements. These technological advancements have enabled quantitative data to be obtained at a system level. Using these information, mathematical models of the biological system that is under investigation can be built.

There are a variety of reasons why the mathematical model of biological system under study should be built (Kell, 2006). 1) Using experimental facts to validate the model built from the knowledge of the workings of the biological system, the accuracy of that very knowledge can be reaffirmed. 2) By analyzing the model, the parts that contribute most to some properties of interest can be identified. 3) Using the *in-silico* model, the effects of manipulating experiments could be predicted rapidly. 4) Design the next experiment that would gain the most insight into the biological system. Currently, the building of mathematical models is not part of a biologist's protocol, but it is widely postulated that it would be necessary in the years to come (Kitano, 2002; Kell, 2006).

There are a series of steps involved in building a mathematical model; viz., construction, verification, calibration and validation (Aldridge *et al*., 2006). Model construction is the step where the network of the model is being drawn. This step is typically done manually and based on prior knowledge from various databases such as Kyoto Encyclopedia of Genes and Genomes (Kanehisa and Goto, 2000), Reactome (Vastrik *et al*., 2007) and WikiPathways (Pico *et al*., 2008) etc. Naturally, this process will become cumbersome and error prone as the model complexity and size increases. The second step would be model verification, which is to ensure that the model is accurately translated from prior knowledge and the underlying structure is reasonable. This step is required as the model construction step is prone to errors. Furthermore, it is also possible that prior knowledge from various databases might contain structurally illogical errors. Model calibration is the process of estimating the parameter values of models such that it fits the experimental data. These involve solving an inverse problem and are known to be ill-conditioned and multimodal (Moles *et al*., 2003). The final step in building a mathematical model is model validation. In this step, the model is evaluated to ensure that it satisfies some known constraints or properties that the biological system exhibits. No existing software or framework is able to perform all the above steps, leading to users having to use a mixture of software. This in itself is another problem since different packages are usually incompatible.

Currently, I am attached to Human Genome Center, Institute of Medical Science, University of Tokyo, as a visiting graduate student. In this laboratory, they have already spent over ten years in research and developing technologies to perform some of the above steps. Cell Illustrator was written to facilitate model construction and simulation (Nagasaki *et al*., 2010). Cell Illustrator is a user-friendly program that supports graphical model construction and simulation via Hybrid Functional Petri nets with extension (Nagasaki *et al*., 2004). In an effort to reduce errors in model construction steps, they have been working on automating the translation of prior knowledge to simulation models (Nagasaki *et al*., 2008). In order to provide a framework for model verification, they proposed a system dynamics centered ontology called Cell System Ontology (Jeong *et al*., 2007). Cell System Ontology has the ability to do semantic validation and automatic reasoning to check the consistency of models. Finally, for model calibration, they presented some work on using data assimilation to do parameter estimation (Nagasaki *et al*., 2006; Tasaki *et al*., 2006). However, as mentioned, parameter estimation is a difficult problem and these works are not without their limitations.

Model calibration or parameter estimation and model validation or model checking are currently the lacking technologies in that laboratory. Parameter estimation is critical in modeling the dynamics of biological pathways. It is at the same time a very challenging problem due to the large search space, the existence of multiple local minimums and the experimental data being limited and noisy. As for model validation, it is only gaining attention recently and currently available approaches have several drawbacks. To this end, my current work in University of Tokyo focuses on developing approaches to perform parameter estimation and model validation. In the completion of these algorithms, we would have developed a framework that can handle all the necessary steps in building a pathway model with ease. Equipped with that, I would eventually like to apply that in the building of a lipid-related pathway.

In the following sections, I will describe the work that I have done for the past two years of my PhD candidature and plans for the remaining two years. Section 2 will describe the work accomplished thus far, current state-of-art approaches and our direction for parameter estimation. Section 3 will provide details on the pitfalls of current model validation approaches and my proposed and implemented method in overcoming these problems. Finally, I will briefly discuss how I can deploy these tools in the modeling and simulation of lipid-related pathways.

2 PARAMETER ESTIMATION

Parameter estimation plays a key role in the understanding of complex biological pathways. Having the dynamics would allow us to build a quantitative model, which in turn will shed light on the underlying mechanisms. Parameter estimation is known to be a nonlinear programming problem and because of this, local optimization problems will not be able to obtain satisfactory results as they are likely to converge quickly towards the local optimal. Therefore, deterministic and stochastic global optimization methods are usually employed. Another problem in parameter estimation is scalability, because as the number of unknown parameters increase, the search space also grows exponentially, rendering numerous algorithms unfeasible.

2.1 Related Work

Moles *et al*. (2003) compared several global optimization methods of different nature such as deterministic methods, adaptive stochastic methods and evolutionary computation methods. Using simulation data from a model that consists of 36 kinetics parameters and 8 ordinary differential equations (ODEs), they concluded that only stochastic algorithms, which use evolution strategies, are able to

obtain good results. However, this comes at the price of a high computational cost.

Therefore, recent efforts combine global and local approaches to do parameters estimation (Rodriguez-Fernandez *et al*., 2005; Balsa-Canto *et al*., 2008). Using these hybrid approaches, the rapid convergence of local methods reduces the computational cost significantly while the robust global algorithms prevent getting trapped in local minimum, thus producing satisfactory results. Basically, these hybrid methods proceed as follows; first, scan the full search space with a global algorithm until it hits a switching point, where it will switch to a local algorithm to find the optimal solution in the local vicinity. Naturally, determination of the switching point would be crucial to the success of a hybrid method. Rodriguez-Fernandez *et al*. (2005) has shown that different switching points potentially lead to different results.

Koh *et al*. (2006) suggested an interesting approach from an entirely different direction. They proposed to first break a large pathway into small sub-pathways before performing parameter estimation on each sub-pathway independently. This leads to significant reduction in search space since each sub-pathway would have much lesser parameters to be estimated and the search space is exponentially proportional to number of parameters to estimate. The advantages of this approach is two fold; the computational time needed would be greatly reduced and with a smaller search space, almost any algorithm can be used since some algorithms are only be feasible for small problems such as deterministic and exhaustive methods.

2.2 Data Assimilation

Most current parameter estimation algorithms aims to estimate a single best parameter that could best fit the given data. However, we believe this is inappropriate since it is known that current experimental data is noisy. There are generally two types of noise in data; viz., observation noise and system noise. Observation noise is due to experimental and/or measurement error. This can be reduced as better experimental protocols are devised and more precise equipments are built. System noise is inherently part of a biological system. "All cell components display intrinsic noise due to random births and deaths of individual molecules and extrinsic noise due to fluctuations in reaction rates" (Paulsson 2004). Therefore, instead of eliminating or ignoring noise, we have chosen to embrace it. Thus, it is more appropriate to do parameter's distribution estimation rather than parameter estimation.

Data Assimilation (DA) is a computational approach that estimates unknown parameters in a pathway model using time-course information. Particle filtering, the underlying method used, is a well-established statistical method which approximates the joint posterior distributions of parameters by using sequentially-generated Monte Carlo samples.

2.3 Particle Filter

Nagasaki *et al*. (2006) introduced this approach to systems biology and successfully used it to perform estimation on the circadian clock model. The approach in brief is as follows; given a pathway model, the list of parameters to be estimated and N observed time-points, a set of M particles is drawn either randomly or through user-specified distribution. At each time point N, the M particles will be re-sampled with a probability directly proportional to a fitness score. The fitness score is computed as a function of difference between the simulated and observed data at each time point. A higher score is given to particles with simulated results closer to the observed results. At the end of the algorithm, users are given the distribution plot of the M particles' values.

The power of DA largely depends on two factors; 1) The number of observed time-points, 2) Size of particles and number of parameters to be estimated. From a statistical point of view, the more time-points observed (higher frequency and longer duration), the higher the accuracy would be. However, current experimental techniques and project funding often limit the number of time-points. Also, the size of particles should also be exponentially proportional to the number of parameters to be estimated in order to obtain a high accuracy. However, if the number of particles is too large, it is likely to cause either out of memory error or slow running time on standard desktop computers.

We have developed an application (DA 1.0) to offer some practical solutions to the two problems above (Koh *et al*., 2010). Since observed time-points are often limited, this could potentially reduce the estimation accuracy. To overcome this limitation, DA 1.0 has the ability to increase the frequency of time points by means of smoothing and re-sampling (Fig 1b). In some cases, it is possible that users do not have any experimental data but have an idea of how a particular biological entity would behave with respect to time, either gleaned from literature or simply testing out a hypothesis. To handle such cases, we have also made it simple to draw expression plots within that application.

Another important factor that affects estimation accuracy is the seed size (number of particles). However, setting a large seed size would cause the program to run slowly. Therefore, we suggest for users to do repeat runs using medium seed size (Fig 1d) and to adjust the possible range after each run using the distribution plot of the parameters (Fig 1e).

2.4 Ongoing work

We are currently working on scaling particle filters up for larger problems i.e. problems with many unknown parameters, hence resulting in a huge search space. One intuitive way is to simply use supercomputers to allow the use of large seed size (Nakamura *et al.*, 2009). Another possible approach would be to use the decomposition method proposed by Koh *et al*. 2006. However, the efficiency of the method is largely dependant on the structure of the investigated pathway. As of now, we would like to develop a more general approach, and is working to integrate global optimization methods with particle filter inspired by the hybrid approaches as mentioned earlier in Section 2.1.

3 MODEL CHECKING

Model checking is an automated process to formally verify a system's behavior with respect to a set of properties. It is a widely used technique for validating circuit designs (Biere *et al*., 2003). In model checking, the properties to be verified are first written in a temporal logic. After which, the reachable states of the system are traversed (and maybe recorded) in order for a model checker to determine if the system satisfies those properties. As larger and more complex biological pathways are being modeled, the manual validation of these models becomes tedious if not impossible.

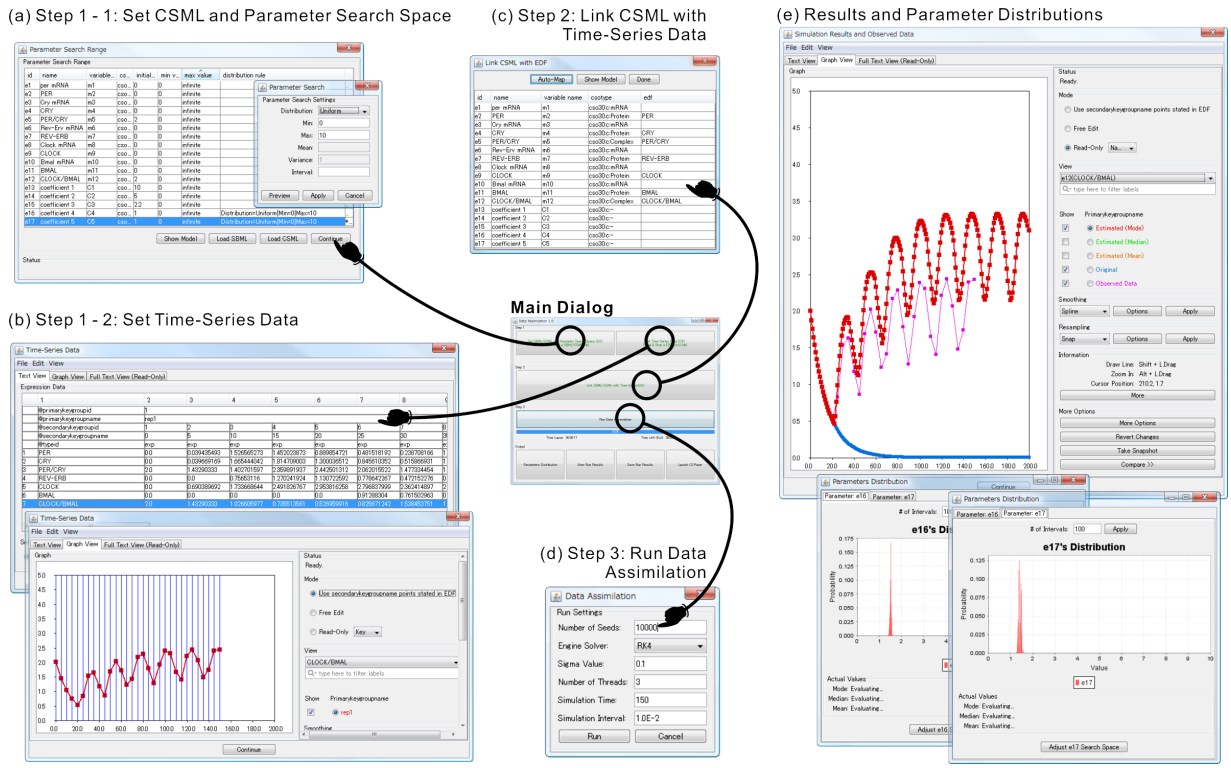


Fig. 1. a) This step is to load the model file (CSML or SBML) and define the distribution and range for parameters that users wish to estimate. b) This step is to input the observed time-series data. Accepted formats include EDF, CSV and TSV. Functions such as smoothing and sampling are included to improve the quality of observed data for better estimation results. c) This step is needed to pair the model entities with observed data. An auto-map function is available to match corresponding entities and observed data with same names. d) A variety of settings for the particle filter and simulation are enabled to allow for flexibility based on the user's needs. e) After running the particle filter algorithm, the simulation runs results using estimated parameters will be plotted for ease of comparison between the original and fitted models. The parameters' distribution plot is also displayed.

Therefore, there is a growing interest in the development and application of model checking algorithms to biological pathway models.

3.1 Related Work

Clarke *et al*. (2008) introduced BioLab, an algorithm to verify properties written in probabilistic bounded linear temporal logic, using the BioNetGen modeling framework. A nice feature of BioLab is that it uses a statistical approach in deciding the minimum number of traces needed to verify a property as the program runs. Younes and Simmons, (2002) proposed this statistical approach to accept or reject a property with a bounded error rate using the minimum number of samples. However, there is a major pitfall in this approach as it depends heavily on a parameter- indifference region- which must be given by the user. Setting that parameter too high would potentially have unbound error rate while setting that parameter too low would significantly increase the number of samples needed. It is impossible to set the optimal indifference region, as that would mean knowing if the property should be accepted or rejected. In this case, there would be no need to run a model checker to validate the property in the first place.

Donaldson and Gilbert (2008a) proposed another temporal logic called probabilistic LTL with numerical constraints (PLTLc). They went on to couple it with a genetic algorithm and used it for parameter estimation. In their method, they simply sampled a fixed number of simulations and validated a property based on those samples. They showed that with 10,000 samples, their results are very close to the results obtained via exact methods using empirical data. Obviously, that fixed number of samples (10,000) is not a generalization across all models. Furthermore, despite being approximative, no statistical measure is computed for their results. Nevertheless, there is an interesting feature in their approach, which is their proposed temporal logic, PLTLc, which allows for properties to be written with a free variable (or numerical constraint). Having free variables allow for a richer way to sum up a set of properties, but they are more complicated to write and interpret (Heiner *et al*., 2009).

In general, there are two approaches in the analysis of stochastic systems; viz., exact or approximative. While exact methods are able to provide high accuracy in theory, they are almost impossible to use in quantitative and stochastic biological models in practice, as there are usually a huge if not infinite number of states unless for simple discrete models. Hence, both authors (Clarke *et al.*, 2008; Donaldson and Gilbert, 2008a) used the approximative approach. Due to the nature of the approximative approach, statistics should be used to give a confidence measure in the results returned. However, Donaldson and Gilbert (2008a) did not use any

statistics to compute their results' confidence; and while Clarke *et al*. (2008) deployed a statistical approach, it did not work under all scenarios. In addition, a common drawback of the two methods is that they are both based on the offline model checking approach. That is, all simulation runs must be completed and all traversed states recorded before model checking can commence. This wastes CPU and storage resources if the decision of validity or rejection for the simulation can be determined early in its execution. Naturally, this wastage is magnified and perhaps even becomes intractable when the complexity of the model increases (more parameters to record and longer running time) and/or the total number of simulation runs needed increases significantly. One example where the total number of simulation runs can increase significantly is when model checking is coupled with a genetic algorithm to perform parameter estimation (Donaldson and Gilbert, 2008b). In this instance, checking is performed for every member/model at each generation/iteration.

Online or on-the-fly model checking, on the other hand, carries out model checking during the simulation run. Thus, a full state graph need not be recorded and simulation runs are only executed for as long as a decision needs to be made. Jard and Jeron (1990) introduced on-the-fly model checking for finite state programs that can be represented by a finite state graph; and in 2007, Troncale *et al*. (2007) implemented a real-time model checker for Timed Hybrid Petri Nets (THPN). However, that model checker could only support THPN, which is a sub-class (reduced expressiveness) of Hybrid Functional Petri Nets (HFPNs) (Matsuno *et al*., 2000; Nagasaki *et al*., 2004).

3.2 Statistical Online Model Checker

We have built a model checker that has the advantages of the different approaches mentioned above, but without the limitations that weigh them down. To do so, we first propose a new algorithm for model checking, which always returns a definite result that is statistically backed. Next, we define a new temporal logic, Probabilistic Linear Temporal Logic with Statistics (PLTLs), to be used with this algorithm. Finally, we develop a program, MIRACH 1.0, that implements our algorithm and PLTLs. MIRACH is integrated with the HFPNe simulation engine (Nagasaki *et al*., 2010) so as to deploy the efficient online checking approach.

To appeal to the Systems Biology Markup Language (SBML) community (Hucka *et al*., 2003), we have included a SBML2CSML converter (http://csmlpipeline.hgc.jp/). As of now, MIRACH is able to support any models written in commonly used formats such as SBML (L2v1) and CSML (http://www.csml.org/).

The two main contributions of our work are the proposed algorithm that always produces a definite and reliable (statistically backed) result and the implement of the efficient online (on-the-fly) checking. These contribute towards a more reliable and efficient model checker, which is critical to support the ever-growing pathway models' size and complexity.

3.2.1 Temporal Logics

Several different temporal logics have been proposed and used for model checking in biological pathway models. Troncale *et al*. (2007), proposed Continuous Time Evolution Logic (CTEL) for use with their Timed Hybrid Petri Nets (THPN). Clarke *et al*. (2008) presented Probabilistic Bounded Linear Temporal Logic for their BioLab algorithm. Donaldson and Gilbert (2008a) introduced Probabilistic Linear Temporal Logic with numeric constraints (PLTLc). It combines LTL in probabilistic settings (Baier, 1998) and LTL with numeric constraints (Fages and Rizk, 2007). In their implementation, free variables (or numeric constraints) are limited to the integer domain of 0 to positive infinity. Although having free variables allow for a richer way to sum up a set of properties, they are more complicated to write and interpret (Heiner *et al*., 2009). For MIRACH 1.0, we have decided to extend PLTL to PLTLs to better suit our approach, because it is sufficient for stochastic model checking in general and is easy to write and interpret.

The syntax of PLTLs is defined in Table 1 and it allows for the use of filter constructs. For a property in the form ϕ {AP}, ϕ will be checked from the state that AP is satisfied rather than from the default initial state. Note that PLTLs allows AP to contain temporal logic operators (X, F, G, R, U). We also allow the use of formulas without probabilistic operators (i.e. in pure LTL). This is useful when the model is deterministic.

The semantics of PLTLs is defined over the finite sets of finite paths through system's state space, obtained by repeated simulation runs of HFPN models. A property contains two components, the probabilistic operator and the linear temporal logic statement. For each simulation run, the temporal logic statement is evaluated to a boolean truth value, and the probability of the temporal logic statement holding true is decided based on the whole set of simulation runs performed using statistical approaches.

For the probability operator component, there are two distinct operators; $P_{\Diamond}$ is the inequality comparison of the probability of the property holding true and $P_{=?}$ returns the value of the probability of the property holding true with a confidence interval. The semantics of the temporal logic operators are described in Table 2. Concentrations of biochemical species in the model are denoted by [variableName]. We also define a special variable, [time], to represent simulation time.

We also provide the ability to define functions of two different natures: functions that return a real number and functions that return a boolean value. An example of the real number function is d([variableName]) which returns the subtracted value of [variableName] between time *i* and time *i-1*. By convention, the value of the d([variableName]) at time 0 is 0. One example of a boolean function is similarAbsolute(Value *a*, Value *b*, Value ε). This function returns true if $|a - b| \leq \varepsilon$ else it returns false. Please see MIRACH's documentation for details and examples on usage of implemented functions.

Table 1. PLTLs Syntax

ψ	::=	$P_{\Diamond\Omega}$ *(LTL)* | $P_{=?\Theta}$ *(LTL)* | *LTL*
LTL	::=	ϕ *{AP}* | ϕ
ϕ	::=	*X* ϕ | *G* ϕ | *F* ϕ | ϕ *U* ϕ | ϕ *R* ϕ | ¬ ϕ | ϕ && ϕ | ϕ || ϕ => ϕ | *AP*
AP	::=	*AP*||*AP* | *AP*&&*AP* | *AP*=>*AP* | *Value comp Value* | $Value_{boolean}$
Value	::=	*Value op Value* | *[variableName]* | $Function_{numeric}$ | *Integer* | *Real*
$Value_{boolean}$	::=	*true* | *false* | $Function_{boolean}$
comp	::=	== | != | >= | > | < | <=
op	::=	+ | - | * | / | ^
Ω	::=	*(θ, α, β, max)* | *(θ, δ, α, β, max)* | *(θ, δ, α, β, γ, max)*
Θ	::=	*(confidence, max)* | *(confidence, CI, max)*

with $\Diamond \in \{<, <=, >, >=\}$, *θ denotes the value to be compared against; δ denotes the half-width of the indifference region; α denotes the type-I error rate (false negative rate;, β denotes the type-II error rate (false positive rate); γ denotes the probability of an undecided results; max denotes the maximum number of simulation run;, confidence denotes the confidence level; CI denotes the confidence interval.*

Table 2. Semantics of temporal operators

Operator	Explanation
$X\,\phi$	ϕ must be true at the next time point.
$G\,\phi$	ϕ must always be true.
$F\,\phi$	ϕ must be true at least once.
$\phi_1 U\,\phi_2$	ϕ_1 must be true until ϕ_2 becomes true; ϕ_2 must become true eventually.
$\phi_1 R\,\phi_2$	ϕ_2 must be true until and including the time point ϕ_1 becomes true; if ϕ_2 never true, ϕ_1 must always be true.

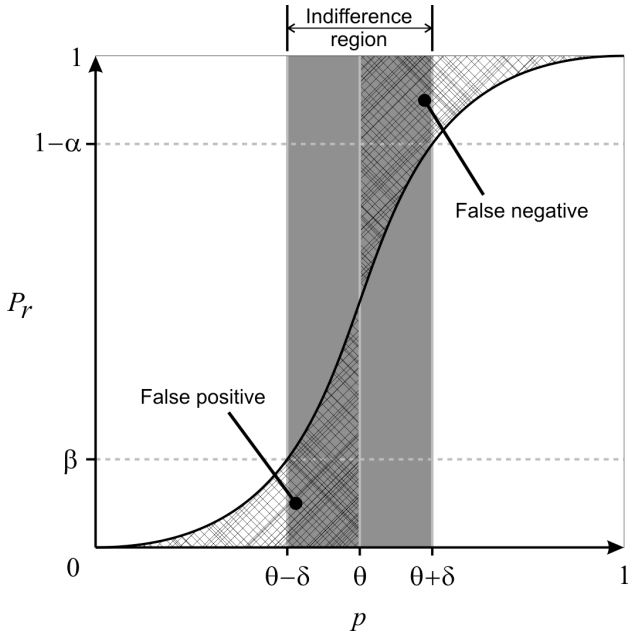


Fig. 2. Pr is the probability of accepting $P_{\geq\theta}\Phi$ as true whereas p is the actual probability of Φ holding true. Note that false positives and false negatives are not bounded when p is inside the indifference region

3.2.2 Statistical Hypothesis Testing

Younes *et al*. (2002) devised an approach based on statistical hypothesis testing to verify probabilistic properties in the form, $P_{\lozenge}\,\Phi$, where $\lozenge \in \{<, <=, >, >=\}$, which Clarke *et al*., (2008) deployed in a systems biology context. In brief, they relaxed the standard hypothesis test of H_0: $p \geq \theta$ vs. H_1: $p < \theta$ to H_0: $p \geq p_0$ vs. H_1: $p \leq p_1$ by introducing $p_0 = \theta + \delta$ and $p_1 = \theta - \delta$ so that α (type-I error rate) and β (type-II error rate) could be bounded independently. The region (p_1, p_0) is known as the indifference region and δ, the half-width of the indifference region. They further showed that by using Wald's sequential probability ratio test (Wald, 1945), they could determine after each sample whether another sample is required or a hypothesis could be accepted with the prescribed strength using available samples (Younes and Simmons, 2002; Younes *et al*., 2006a). The main limitation of this approach is that if $|p - \theta| < \delta$, then it gives no bound on the error rate (Fig. 2); i.e. p is inside the indifference region. It is worth noting that this approach does not give any likelihood as to whether p is inside or outside the indifference region. This implies that although the test promises to bound the error rate for p outside of the indifference region, there is no indication if p lies outside the indifference region.

In another publication, Younes (2006b) proposed another procedure that would bound the error rate whenever a definite result is given. This procedure could be seen as using two simultaneous acceptance tests as described above, H_0: $p \geq \theta$ vs. H_1: $p \leq \theta - \delta$ with strength $<\alpha, \gamma>$ and H'_0: $p \geq \theta + \delta$ vs. H'_1: $p \leq \theta$ with strength $<\gamma, \beta>$. γ represents the bound on the probability of an undecided result. A property, $P_{\geq\theta}\,\Phi$ is accepted as true only if both H_0 and H'_0 are accepted. Likewise, the property is accepted as false only if both H_1 and H'_1 are accepted. In other cases, the property is deemed as undetermined; e.g. H_0 is accepted but H'_0 is not accepted. This new procedure overcomes the limitation of unbounded error (if p is inside indifference region) in the previous approach. However, it comes at the expense of an increased sample size needed to make a decision, given the same α, β and δ (Younes, 2006b), and more importantly, the chance of an undecided result after resources have been used to do sampling.

In both methods proposed above, the selection of δ is critical. If δ is set too small, the sample size required will be very large as it is inversely proportional to δ^2. If δ is set too large, it will result in either an unbounded error rate (if the first approach is used) or an undecided result (if the second approach is used). Therefore, we propose an algorithm that does not require an indifference region to be predetermined by the user in the next sub-section.

3.2.3 Our proposed algorithm

We propose an algorithm that dynamically selects an indifference region by improving on the procedure described in the previous sub-section (Younes, 2006b). The algorithm is as follows; we first assume δ to be a sufficiently large value, currently set as 0.05. Then we proceed using two simultaneous acceptance-sampling tests. However, whenever an undecided result is given, we reduce δ by ½ and check if a definite result can be given or another sample is needed or a further reduced δ is required. We continue this process until a definite result is produced.

This algorithm has two advantages. Firstly, a predetermined δ is not required and hence the number of samples required will adjust to the difficulty of the problem; i.e. depending on how close p is to θ. Secondly, this algorithm always gives a definite result if sufficient samples are given, and that result is guaranteed to be error bounded.

However, to always provide a definite result with its error bounded has a drawback. That is, if p is very close to θ, δ needs to be reduced to a very small value such that $\delta < |p - \theta|$. As mentioned, if δ is very small, the sample size required to determine a result will be very large or in the worst case where $p = \theta$, this algorithm will not terminate. Hence, in the next sub-section, we further improve on this proposed algorithm by allowing a limit on the sample size to be set; thus ensuring that the program completes in a user acceptable runtime.

3.2.4 Limiting the number of samples generated

By limiting the sample size, we can bound the runtime of the program but we can no longer bound the error rate. So we compute a p-value, which serves as a measure of confidence of the result. The algorithm is as follows; as before, we first assume δ to be a large value, currently set as 0.05. Then we proceed using two simultaneous acceptance-sampling tests and whenever an undecided result is

given, we reduce δ by ½ and check if a definite result can be given or another sample is needed or further reducing δ is required. We continue this process until a definite result is given or the sample size limit is reached. If a definite result is reached before the sample size limit, then the error rate is guaranteed to be bounded. Otherwise, if the sample size limit is reached, we compute the p-value for both hypothesis (H_0: $p \geq \theta$ vs. H_1: $p < \theta$) and accept the hypothesis with the lower p-value.

The method for computing the p-value is presented in Younes (2005) and is given as Equation 1 here. The p-value for H_0: $p \geq \theta$ is $1 - F(d; n, \theta)$ and the p-value for H_1: $p < \theta$ is $F(d; n, \theta)$ where d is the number of successes (or true) and n is the total number of samples.

$$F(d;n,\theta) = \sum_{i=0}^{d} \binom{n}{i} \theta^i (1-\theta)^{n-1} \quad (1)$$

The time complexity of approximative model checking depends on two main factors: sample size and time used for each sample. Now that we have demonstrated how to achieve the minimum sample size to satisfy user requirements, we will discuss in the next sub-section how simulation time of each simulation run can be reduced to further increase the efficiency of our model checker.

3.2.5 Online Sample Verification

There are two different approaches---viz., offline and online---to obtain the truth-value of a simulation run. The offline approach does verification after obtaining the full set of simulation traces while the online approach checks as the simulation runs. The main advantage of offline model checking is that it can perform checking as long as the traces can be obtained. This permits the validation of existing traces and non-computational models such as biological experiments results. Online model checking, on the other hand is always faster because the simulation only needs to be run for as long as needed to verify the properties. Since we are developing a model checker for *in-silico* simulation models, it is more efficient to use the online approach provided we are able to integrate it with a simulation engine.

HFPNs are versatile and expressive Petri nets for modeling and simulation of complex biological processes. Numerous large and complex models have already been built using HFPNs (Troncale *et al*., 2006). Therefore, we integrate MIRACH 1.0 with the HFPNe simulation engine (Nagasaki *et al*., 2010) to perform online verification. At each time step of a simulation run, each LTL statement that has yet to be accepted or rejected is checked. A LTL statement is removed once its truth-value can be determined and the simulation stops when all LTL statements have been determined or the predetermined termination simulation time has been reached. We present below the pseudo codes of our online algorithm:

```
for(time = 0; time ≤ terminationTime; time = time + timeStep){
  if(LTLSet.size() == 0){
    break; //end simulation since all LTL statements have been determined
  }
  foreach LTL in LTLSet{
    if(APisNotDefined() || (APisDefined() && checkIsAPSatisfied())){
      result = check(LTL);
      if(result == TRUE || result == FALSE){
        //Remove LTL from LTLSet since it has been determined
        RemoveFromLTLSet(LTL);
      }else{ //result is UNKNOWN at current time point
        //Log involved species in LTL for current time
        StoreValues(involvedSpeciesValues, time);
      }
    }
  }
}
```

The simulation model defines terminationTime and timeStep. Stored values using StoreValues(involvedSpeciesValues, time) are used in check(LTL) and the implementation of check(LTL) is different for each of the five temporal logic operators (X, F, G, U, R).

3.3 Comparison with other Model Checkers

3.3.1 Comparison of Hypotheses Testing Algorithms

Clarke *et al.* (2008) implemented an approach, initially introduced by Younes *et al.* (2002), to probabilistic model checking for biological pathway models, where there is no undecided result (given sufficient samples) but the error rate is not always bounded. Another paper by Younes (2006b) proposed an algorithm that always bounds the error rate whenever a definite result is returned but this algorithm does not always return a definite result (even when sufficient samples are given). We name these two algorithms as Younes algorithm A (Younes *et al*., 2002) and Younes algorithm B (Younes, 2006b) for ease of reference. To compare our proposed algorithm against these two algorithms, we set up a simple experiment.

We use a uniform random generator that produces real numbers in the range of [0,1]. We attempt to validate $P_{>=0.8}$ with p set to 0.7, 0.78 and 0.79 to represent three different levels of difficulty- easy, medium and hard, respectively. If the generated value is greater than p, then that sample is accepted as true, and false otherwise. Using this approach, we know the true probability, p, and can determine if a response from an algorithm is correct. Therefore, we can compare the performance of these algorithms under different scenarios. For each scenario, we repeat the experiment 10,000 times with α (type-I error rate) and β (type-II error rate) fixed at 0.0001. We expect the number of errors to be less than or equal to 0.0001 * 10,000 = 1 (or at least close to 1) if the error rate is to be bounded. We further limit the sample size used by our algorithm to 10,000, as we believe any realistic application should have a limit on the CPU resource. A total of three tables are obtained, one for each algorithm.

Table 3. Younes algorithm A (Younes et al., 2002)

	p = 0.7	**p = 0.78**	**p = 0.79**
δ = 0.04			
No. of Errors	0	73	**626**
No. of Undecided	0	0	0
Avg sample size	181.2	820.7	1350.5
δ = 0.01			
No. of Errors	0	0	0
No. of Undecided	0	0	0
Avg sample size	737.5	3676.3	7311.4
δ = 0.001			
No. of Errors	0	0	0
No. of Undecided	0	0	0
Avg sample size	7377.5	36911.7	**73715**

No. of Errors show how many mistakes were made in the 10,000 experiments. No. of Undecided indicates how many experiments returned undecided results. Avg sample size compute the average samples used in each of the 10,000 experiments.

Table 3 clearly shows how crucial the selection of δ is for Younes algorithm A. If δ is too large, the error rate is not bounded (e.g. with p = 0.79 and δ = 0.04, error rate is at 6.26% instead of the requested 0.01%). On the other hand, if δ is too small, then the number of samples needed to make a decision grows rapidly (e.g. with p = 0.79 and δ = 0.001, the average number of samples needed is over 70,000). Indeed, if the correct δ is chosen, the error rate is bounded and minimum samples are used. However, it is rare for users to be able to choose the correct δ since we would need to know p, the true probability to do so; and if we actually knew that then there is really no need to do any hypothesis testing in the first place. It should be noted that Younes algorithm A does not provide information on when the error rate is bounded or not. This implies that the user may come to a false conclusion that the result is bounded with a certain error rate when it is actually not.

Table 4. *Younes algorithm B (Younes, 2006b)*

δ = 0.04	**p = 0.7**	**p = 0.78**	**p = 0.79**
γ = 0.1			
No. of Errors	0	0	0
No. of Undecided	0	**7561**	**9913**
Avg sample size	487.8	2571.3	1191.2
γ = 0.01			
No. of Errors	0	0	0
No. of Undecided	0	**6211**	**9890**
Avg sample size	493.6	4738.9	2185.1
γ = 0.001			
No. of Errors	0	0	0
No. of Undecided	0	**5206**	**9900**
Avg sample size	491.1	7016.2	3168.3
No. decided by p-value	0	2814	9379
Errors by p-value	0	0	77
Avg p-value (wrong decision)	NA	NA	**0.3744**
Avg p-value (correct decision)	NA	0.0008	**0.0379**
γ = 0.001			
No. of Errors	0	0	0
No. of Undecided	0	0	0
Avg sample size	428.3	7610.1	9852.0
No. decided by p-value	0	3079	9521
Errors by p-value	0	0	70
Avg p-value (wrong decision)	NA	NA	**0.3866**
Avg p-value (correct decision)	NA	0.0009	**0.0389**

No. decided by p-value shows how many experiments were decided by comparing p-values. Errors by p-value indicate the number of wrong decisions made by accepting the hypothesis with lower p-value when the sample size limit is reached. Avg p-value (wrong decision) computes the average p-value of those that we use p-value to decide and was wrong. Avg p-value (correct decision) computes the average p-value of those that we use p-value to decide and was right.

From Table 4, we can see that Younes algorithm B does indeed always bound the error rate. However, it comes at the expense of a large number of undecided results. Over 50% were undecided results when p = 0.78 and it goes beyond 90% when p = 0.79. Note that this means the algorithm uses up computation resources and in the end returns an undecided result, which is undesirable. We should also note the introduction of a new parameter γ in this algorithm where it is meant to limit the probability of undecided results but it comes at the expense of higher sample size.

Table 5. Our Proposed Algorithm

	p = 0.7	**p = 0.78**	**p = 0.79**
γ = 0.1			
No. of Errors	0	0	0
No. of Undecided	0	0	0
Avg sample size	422.6	6927.2	9814.4
No. decided by p-value	0	2452	9435
Errors by p-value	0	0	58
Avg p-value (wrong decision)	NA	NA	**0.3850**
Avg p-value (correct decision)	NA	0.0010	**0.0387**
γ = 0.01			
No. of Errors	0	0	0
No. of Undecided	0	0	0
Avg sample size	428.3	7290.3	9825.4

As shown in Table 5, our proposed algorithm always gives a definite result. When it is unable to bound the error rate of the result because the sample size limit has reached, it computes and accepts the hypothesis that has a lower p-value. It may seem that accepting a hypothesis based on p-value often results in mistakes. However, if we look at the average p-value where correct and wrong decisions are made, it is clear that wrong decisions are often made when the p-value is close to 0.4 and correct decisions are made when the p-value is below 0.04. Therefore, it is not difficult for a human to discern when to suspect or trust the result given a p-value.

Another feature of our algorithm is that the γ parameter is less influential as we do not have undecided results. Furthermore, γ is found to be optimal at 0.1 (Table 5). Hence, we set it to 0.1 as default in MIRACH so that users have one less parameter to worry about.

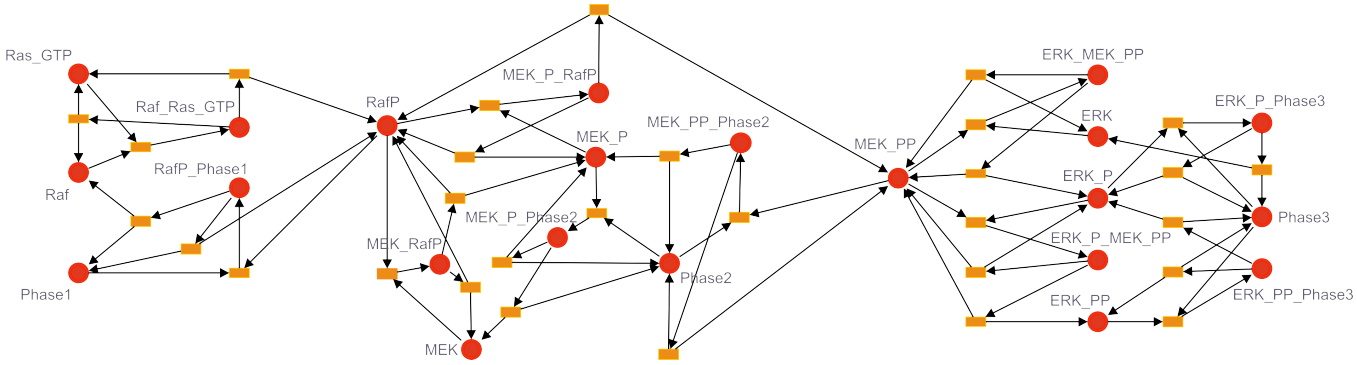
3.3.2 Online vs. Offline

It is not difficult to appreciate that an online approach is almost certainly more efficient than offline in terms of time efficiency since it only runs as long as it needs to and does not read and write to the hard disk. However, to prove precisely how much faster is challenging and subjective as there are numerous factors affecting it. Apart from factors such as the programming language used, actual implementation, model to be checked on, hard disk access speed, etc., it largely depends on the properties written and also how they are formulated. For instance, if those properties were written such that they could be determined early in the simulation in most of the runs, then the amount of time saved would be large. On the other hand, if the properties were written such that they could only be verified at the last time point of most simulation runs, then the savings would be reduced. Therefore, to show that there is no bias towards the online approach in our comparison, we assume the worst-case situation that the simulation run will continue even when all properties have been determined.

One offline model checker similar to MIRACH is MC2(PLTLc) by Donaldson and Gilbert (2008a). Both model checkers are writ-

ten in Java and supports PLTL. Therefore, we use MC2(PLTLc) to illustrate the difference between online and offline checkers.

To run model checking using MC2(PLTLc), two steps are required. First, obtaining the simulation results, then running MC2(PLTLc) to apply checking. MIRACH 1.0 also encompasses two steps---first, the initialization stage where models will be converted to CSML (if needed) and the integration of properties to be checked into the model. The second step will be checking, which is concurrent with the simulation run.



To compare between the two model checkers, we need a sample model that can be run in both. We have chosen a SBML model (Fig. 3) by Levchenko *et al*. (2000) that was used by Donaldson and Gilbert (2008a) as an example in their paper. This model is chosen because, among the few models they described, this is one model where most of the properties can be modified to our defined PLTLs easily (from PLTLc) and at the same time was the most complex model presented.

Levchenko *et al*. (2000) used this model to investigate the influence of scaffold proteins on mitogen-activated protein kinase. The core of the model consists of ordinary differential equations (ODEs) that describe sequential phosphorylation and dephosphorylation of MAPK cascade kinases and their corresponding phosphatases. For comparison, we are using the SBML model, Levchenko_4000.xml and for the properties, we concatenate C.queries (Table 6) and S1_4.queries together and obtain a total of eight properties. All files are downloaded from the MC2(PLTLc) website.

Table 6. *C.queries in PLTLs syntax*

	PLTLs Query
C1	$P_{>=}$(0.99,0.0001,0.0001,1000)((([MEK_PP] < 0.001) && ([ERK_PP] < 0.0002)) U ([RafP] > 0.06))
C2	$P_{>=}$0.99,0.0001,0.0001,1000((([RafP] > 0.06) && ([ERK_PP] < 0.0002)) => ((([RafP] > 0.06) && ([ERK_PP] < 0.0002)) U ([MEK_PP] > 0.004)))
C3	$P_{>=}$0.99,0.0001,0.0001,1000((([RafP] > 0.06) && ([MEK_PP] > 0.004)) => ((([RafP] > 0.06) && ([MEK_PP] > 0.004)) U ([ERK_PP] > 0.0005)))

Table 7. *MIRACH vs. MC2(PLTLc) using Levchenko model*

	100 Samples	1000 Samples
MIRACH		
Initialization	6.85 (0.24)	6.86 (0.31)
Simulation and Checking	5.34 (0.20)	40.74 (0.90)
Total Time	12.19	**47.6**
MC2(PLTLc)		
Run simulation and log results	12.14 (0.40)	107.95 (1.52)
Load results and check	10.13 (0.29)	88.58 (1.11)
Total Time	22.27	**196.53**

100 Samples indicates that 100 simulation runs were executed (similarly for 1000 samples). Results shown are in seconds and are the average of 20 repeated runs. The number in brackets is the standard deviation. All runs are performed on a laptop with 1.6GHz dual core processor and 2G RAM running on Linux.

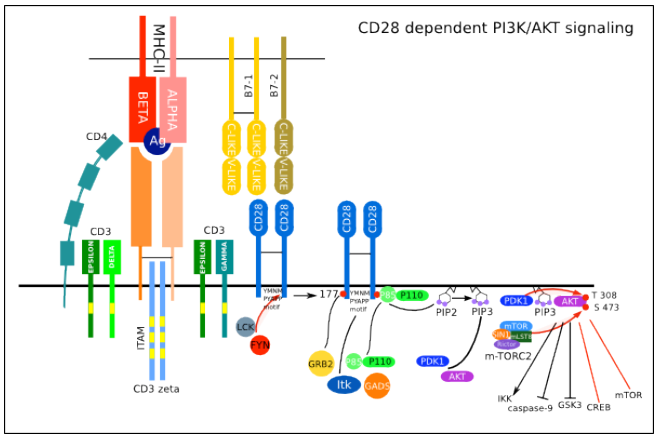
From Table 7, we see that MIRACH outperforms MC2(PLTLc) and the time saved increases with sample size. When comparing the runtime for 1000 samples, the time saved by using MIRACH is already 400%. As seen in Section 3.1, to obtain good confidence in the approximation of PLTLs, the sample size required could easily exceed thousands, likely leading to an even greater amount of time saved using MIRACH as compared to MC2(PLTLc). The main reason that slows MC2(PLTLc) down is the logging and reading of results to and from the hard disk. It should be noted that we only logged the species that were used in the properties which is only a total of three for this example. Naturally, the logging and reading time will increase significantly as the properties involve more species.

It is also worth noting that we are currently considering the worse case performance of MIRACH as we "force" each simulation run to complete, regardless as to whether LTL could have been decided early in the simulation.

Another performance measure is the minimum memory requirement. Precise memory requirements depend on several factors such as the model used and properties to be checked. The memory requirement of online checking is likely to be higher than offline checking because the offline method does not carry out checking and simulation concurrently. As described in Section 2.3.1, in the checking step, MIRACH needs to store the values of involved species in memory (RAM) when a LTL cannot be decided (neither TRUE nor FALSE) at that time point. However, even in an extreme case, where there are 100 species involved and that property cannot be decided for 100,000 time points, the additional memory (RAM) needed is still less than 80MB (100 x 100,000 x 8 bytes). Note that this memory space used will be freed once that particular simulation ends and will not increase with the number of simulation runs.

3.4 Ongoing work

Currently, we are working to improve on an *in-silico* model that investigates cell fate determination of gustatory neurons of Caenorhabditis elegans previously published by my laboratory in Tokyo (Saito *et al*., 2006). We do this by deriving properties from several recently published biological papers on this pathway. The motivation of this current work is to demonstrate the practicality of MIRACH and also to obtain an improved version of the model from which we hope to gain new insights into the pathway.



4 UPCOMING PLANS

My attachment to Tokyo University will end in June 2011. In this remaining year in Tokyo, I hope to finish up the development of the parameter estimation method. Together with the developed model checker by me and other technologies by Tokyo University, we would have a framework to build and assign dynamics to any biological pathway of interest. Therefore, in the 4th year of my PhD, I wish to apply this framework to provide assistance to the lipidomics project in Singapore.

In particular, we can utilize the Reactome (www.reactome.org) pathway database, which is an expert-authored and peer-reviewed database that includes more than 2900+ human proteins, 2900+ reactions and 4400+ literature citation. Using this database and our framework, we can quickly build a pathway model based on literature and customize it to suit our goals. For example, there is this pathway named the CD28 dependent PI3K/Akt signaling for Homo sapiens (Figure 4) in Reactome. This pathway model would be very closely link to the work we previously initiated with Shahid Noor (from Professor Wenk's group). We can build on this model and further the collaboration work with Shahid.

REFERENCES

Aldridge,B.B. *et al.* (2006) Physicochemical modeling of cell signalling pathways. *Nature Cell Biology*, **8**, 1195-1203.

Biere,A. *et al.* (2003) Bounded Model Checking. Advances in Computers, **53**.

Balsa-Canto,E. *et al*. (2008) Hybrid optimization method with general switching strategy for parameter estimation. *BMC Systems Biology*, **2**:26.

Clarke,E.M. *et al.* (2008) Statistical Model Checking in BioLab: Applications to the Automated Analysis of T-Cell Receptor Signaling Pathway. *In Proc. CSMB 2008*, 231–250.

Donaldson,R. and Gilbert,D. (2008a) A Monte Carlo Model Checker for Probabilistic LTL with Numerical Constraints. Technical report, University of Glasgow, Department of Computing Science.

Donaldson,R. and Gilbert,D. (2008b) A Model Checking Approach to the Parameter Estimation of Biochemical Pathways. *In Proc. CSMB 2008*, 269–287.

Heiner,M. *et al.* (2009) Extended Stochastic Petri Nets for Model-Based Design of Wetlab Experiments. *In Proc. CMSB 2009*, 138–163.

Hucka,M. *et al.* (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics* **19**, 524–531.

Jard,C. and Jeron,T. (1990) On-Line Model-Checking for Finite Linear Temporal Logic Specifications. *Lecture Notes in Computer Science*, **407**, 189–196.

Jeong,E. *et al*. (2007) Cell System Ontology: Representation for Modeling, Visualizing, and Simulating Biological Pathways. *In Silico Biology*, **7**, 0055.

Paulsson,J. (2004) Summing up the noise in gene networks. *Nature*, **427**, 415-418.

Kanehisa, M. and Goto, S. (2000) KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Research*, **28**, 27-30.

Koh,C.H. *et al*. (2010) DA 1.0: Parameter Estimation of Biological Pathways using Data Assimilation approach. *Bioinformatics*, EPub May 26.

Koh,G. *et al*. (2006) A decompositional approach to parameter estimation in pathway modeling: a case study of the Akt and MAPK pathways and their crosstalk. *Bioinformatics*, **22**, 271-280.

Kell,DB. (2006) Metabolomics, modeling and machine learning systems. *FEBS Journal*, **273**, 873-894.

Kell,DB. (2007) Metabolomic biomarkers: search, discovery and validation. *Expert Review Molecular Diagnostic*, **7**(4). 329-333.

Kitano,H (2002) Computational Systems Biology. *Nature*, **420**, 206–210.

Levchenko, A. *et al*. (2000) Scaffold proteins may biphasically affect the levels of mitogen-activated protein kinase signaling and reduce its threshold properties. *Proc Natl Acad Sci USA*, **97**(11), 5818–5823.

Matsuno,H. *et al*. (2000) Hybrid Petri net representation of gene regulatory network. *Pacific Symposium on Biocomputing*, **5**, 341–352.

Moles,C.G. *et al*. (2003) Parameter Estimation in Biochemical Pathways: A Comparison of Global Optimization Methods. *Genome Research*, **13**, 2467-2474.

Nagasaki,M. *et al.* (2004) A versatile Petri net based architecture for modeling and simulation of complex biological processes. *Genome Informatics*, **15**(1), 180–197.

Nagasaki,M. *et al*. (2006) Genomic data assimilation for estimating Hybrid Functional Petri Net from time-course gene expression data. *Genome Informatics*, **17**(1), 46–61.

Nagasaki,M. *et al*. (2008) Systematic Reconstruction of TransPath Data into Cell System Markup Language. *BMC Systems Biology*, **2**:53.

Nagasaki,M. *et al.* (2010) Cell Illustrator 4.0: A computational platform for systems biology. *In Silico Biology* **10**, 0002.

Nakamura,K. *et al*. (2009) Parameter estimation of *in silico* biological pathways with particle filtering towards a petascale computing. *Pacific Symposium on Biocomputing*, **14**, 227–238.

Pico,A.R. *et al*. (2008) WikiPathways: Pathway Editing for the People. *PLoS Biology*, **6**(7).

Rodriguez-Fernadez,M. *et al*. (2005) A hybrid approach for efficient and robust parameter estimation in biochemical pathways. *BioSystems*, **83**, 248-265.

Tasaki,S. *et al*. (2006) Modeling and estimation of dynamic EGFR pathway by data assimilation approach using time series proteomic data. *Genome Informatics*, **17**(2), 226–238.

Troncale,S. *et al*. (2006) Modeling and simulation with hybrid functional petri nets of the role of interleukin-6 in haematopoiesis. *In Proc PSB 2006*.

Troncale,S. *et al.* (2007) Validation of Biological Models with Temporal Logic and Timed Hybrid Petri Nets. *Conf Proc IEEE Eng Med Biol Soc*, 2007, 4603-4608.

Vastrik,I. *et al*. (2007) Reactome: A Knowledge Base of Biologic Pathways and Processes. *Genome Biology*, **8**:R39.

Wald,A. (1945) Sequential tests of statistical hypotheses. Annals of Mathematical Statistics **16**(2), 117–186.

Younes,H.L.S. *and* Simmons,R. (2002) Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling. *Lecture Notes in Computer Science*, **2404**, 223–235.

Younes,H.L.S. (2005) Probabilistic Verification for "Black-Box" Systems. *Lecture Notes in Computer Science*, **3576**, 253–265.

Younes,H.L.S. *et al.* (2006a) Numerical vs statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer*, **8**, 216–228.

Younes,H. (2006b) Error Control for Probabilistic Model Checking. *Lecture Notes in Computer Science*, **3855**, 142–156.