Systems Biology

Statistical Online Model Checker for Biological Pathway Models

Chuan Hock Koh[1,2,3], Masao Nagasaki[3,*], Ayumu Saito[3], Chen Li[3], Limsoon Wong[2] and Satoru Miyano[3]

[1]NUS Graduate School for Integrative Sciences and Engineering, Singapore 117597
[2]School of Computing, National University of Singapore, Computing Drive, Singapore 117417
[3]Human Genome Center, Institute of Medical Science, University of Tokyo, 4-6-1 Shirokanedai, Minato-ku, Tokyo 108-8639, Japan

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT

Motivation: Model checking is playing an increasingly important role in systems biology as larger and more complex biological pathways are being modeled. However, current available model checkers designed for biological pathway models are often offline-based (which waste resources), limited to a restricted class of models, or do not provide reliable results in every situation.

Result: First, we propose a new algorithm that always produces a definite and reliable (statistically backed) result. Next, we design a new temporal logic, Probabilistic Linear Temporal Logic with Statistic (PLTLs), to be used with this algorithm. Finally, we develop a program, MIRACH 1.0 that implements our algorithm and PLTLs while concurrently supporting any model written in commonly-used formats such as CSML and SBML. MIRACH is integrated with a simulation engine, enabling efficient online (on-the-fly) checking. The inadequacies of current model checking algorithms when compared to MIRACH in terms of reliability are clearly demonstrated through a simple yet representative experiment. In another experiment, by using the Levchenko *et al.* (2000) model, we reveal that time-saving gains by using MIRACH easily surpass 400% compared to the offline-based counterpart.

Availability and Implementation: MIRACH 1.0 was developed using Java and thus would be executable on any platform installed with JDK 6.0 (not JRE 6.0) or later. MIRACH 1.0 along with its source codes, documentation and examples are available at http://sourceforge.net/projects/mirach/ under the LGPLv3 license.

Contact: masao@ims.u-tokyo.ac.jp

1 INTRODUCTION

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

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. 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 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 biological systems in practice, as there are usually an infinite number of states. 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 do 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

*To whom correspondence should be addressed

© Oxford University Press 2005

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).

In this paper, we present an efficient and reliable model checker. It has the advantages of the different approaches mentioned above, but without the limitations that weigh them down. We first propose a new algorithm for model checking, which always returns a definite result that is statistically backed. 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 increase MIRACH 1.0's appeal to the Systems Biology Markup Language (SBML) community, a SBML2CSML (http://csmlpipeline.hgc.jp/) converter is included. Hence, MIRACH 1.0 could do checking for pathway models written in SBML (L2v1) (Hucka *et al*., 2003) and CSML formats (http://www.csml.org/).

2 METHODS

To build a model checker, one of the first steps is defining a formalism to express the rules to be checked. We have chosen PLTLs for reasons that we will describe shortly in Section 2.1. Due to the stochastic and complex nature of biological systems, we have also chosen to use approximative model checking. In this approach, there are two approximations. First, only a finite subset of the model's behavior is sampled to estimate the probability of a property being true or false. Thus, statistical approaches should be deployed to measure the confidence of results. We propose a new algorithm to overcome current limitations in this area, which we will discuss in Section 2.2. The next approximation is that, since the simulation run is finite, we can only approximate the truth-value of a property's temporal logic statement from the given limited run results. Hence, to have a good approximation, it is best to run the simulation until a steady state or until the properties could be determined. Notably, the offline model checking approach is therefore not well suited as it does not carry out checking when the simulation is running. In Section 2.3, we will describe our online (on-the-fly) algorithm to check the validity of a property's temporal logic statement as the simulation runs. Finally, MIRACH's implementation and its usage instructions can be found in its documentation.

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, we have decided to extend PLTL to PLTLs so that it will better suit our approach. We have chosen PLTL to build upon 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; $P_{\lozenge}$ 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.

Furthermore, we 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	::=	$P_{\lozenge\Omega}$ *(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 *=>* 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 $\lozenge \in$ *{<, <=, >, >=}, 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 ϕ	ϕ must be true at the next time point.
G ϕ	ϕ must always be true.
F ϕ	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_1 never become true, ϕ_2 must always be true.

2.2 Hypothesis Testing and Statistical Estimation

Clarke *et al*. (2008) implemented the SPRT (sequential probability ratio test) algorithm for hypothesis testing without undecided results to validate a probabilistic temporal logic statement statistically. The advantage of this approach is that it uses the minimum number of executions necessary to determine a given property. However, there are three limitations. Firstly, by using only hypothesis testing, it is not easy to handle the =? operator; that is, to estimate the percentage in which the temporal logic statement is true. In order to do that, one has to use statistical estimation. Another limitation in using this approach is that the user must define an indifference region (Fig. 1). The decision on the indifference region is crucial. The expected number of samples required is large if it is too small, since the complexity of the algorithm is inversely proportional to the width of the indifference region (Younes *et al*., 2006b). On the other hand, if the indifference region is chosen to be too large, the chance of the "true" distribution being in the indifference region increases; consequently, the error rate becomes unbounded (Younes *et al*., 2006b). Furthermore, without putting a limit on the number of samples, there is a chance of an extremely long running time if the required sample size is exorbitant--- i.e. indifference region set too low and/or very low error rate requested.

We design MIRACH 1.0 to be efficient and at the same time overcome the above limitations.

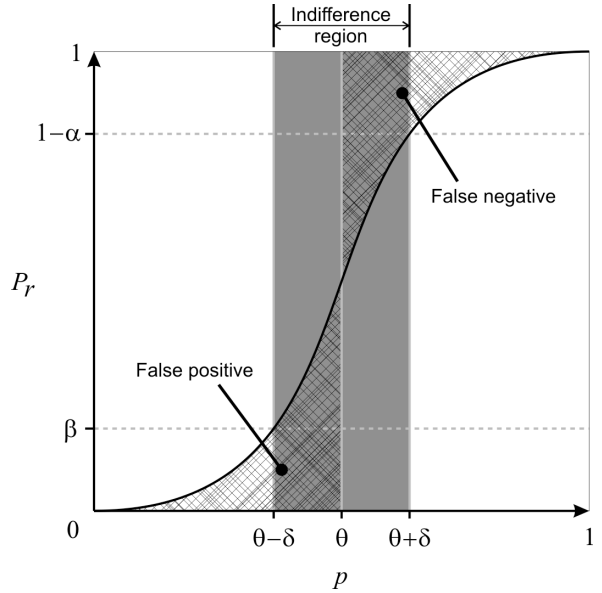

Fig. 1. Pr is the probability of accepting $P_{\geq\theta}\Phi$ as true whereas p is the actual probability of Φ holding true. Note that false positives and false negatives are not bounded when p is inside the indifference region.

2.2.1 The $P_{=?}$ Operator

As mentioned earlier, in approximative model checking, we are only estimating the probability that the property holds in a stochastic model since we are only sampling a subset of the model's behavior. Therefore, we should give a confidence interval instead of just returning a single value. We have chosen to use Wilson interval (Wilson, 1927) to compute the confidence interval instead of the simpler normal approximation interval for two reasons. First, normal approximation is known to perform badly when the sample probability is close to 0 or 1 and fail completely when it is 0 or 1. Due to this, one cannot devise a sequential sampling algorithm that stops sampling once the confidence interval falls below a certain value, as the confidence interval will always be 0 if the current sample proportion is 0 or 1. Wilson interval (Equation 1) does not have these limitations and allows us to give more flexibility to our model checker.

$$\frac{\hat{p} + z_{\alpha/2}^2 \pm z_{\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z_{\alpha/2}^2}{4n^2}}}{1 + \frac{z_{\alpha/2}^2}{n}} \quad (1)$$

$\hat{p}$ = No. of true samples / n, n = No. of samples, $z_{\alpha/2}$ denotes the variate value from the standard normal distribution such that the area to the right of the value is $\alpha/2$

2.2.2 *With and without undecided results*

Younes *et al*. (2002) devised an approach based on statistical hypothesis testing to verify probabilistic properties in the form, $P_{\diamond}\Phi$, where $\diamond \in \{<, <=, >, >=\}$, 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. 1); 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 subsection.

2.2.4 Without a predetermined indifference region

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 subsection, 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.

2.2.5 Limiting the number of samples generated

By limiting the sample size, we can bound the runtime of the program but we might not be able to always bound the error rate. So we compute a p-value, which will serve 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 2 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.

$$F(d;n,\theta) = \sum_{i=0}^{d} \binom{n}{i} \theta^i (1-\theta)^{n-1} \quad (2)$$

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.

2.3 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 RESULTS

The two quantifiable contributions of this paper are the proposed algorithm, which can dynamically select an indifference region, and the implementation of the more efficient online approach in model checking. Therefore, in this section, we will demonstrate the performance of MIRACH 1.0 against previous works from these angles.

3.1 Probabilistic model checking

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 a randomly generated value is lesser 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

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
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.

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 humans 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.

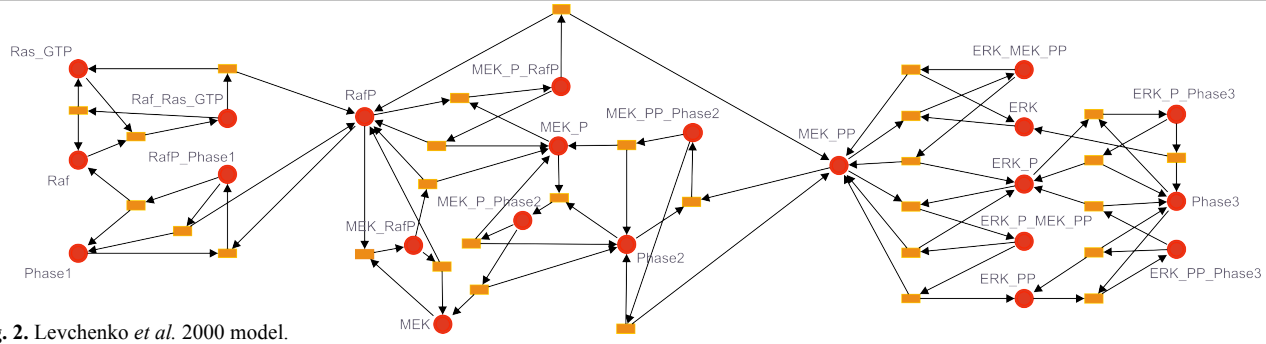

Fig. 2. Levchenko *et al.* 2000 model.

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 written 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. 2) 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.

4 CONCLUSION

In this paper, we have presented a model checker, MIRACH 1.0, for biological pathway simulation models. First, we have proposed a new algorithm that automatically adjusts the sample size needed according to the difficulty of the problem and always gives a definite result which either guarantees a bounded error rate (decided by user) or, when that is not possible (due to sample size limit), gives a p-value as a measure of confidence to the user. With the option of setting a limit on the sample size, users need not worry about non-termination (when p is equal to θ) or an extremely long runtime (when p is extremely close to θ). Another major contribution is the implementation of the more efficient online approach that saves a significant amount of time. Furthermore, MIRACH is integrated with the HFPNe simulation engine, an expressive and powerful framework for defining biological pathway models (Troncale *et al*., 2006). For coverage, MIRACH supports commonly used formats such as CSML and SBML. Lastly, in an attempt to enable easy extension and contribution to MIRACH, we have uploaded its source along with documentation and examples so that other researchers can easily modify and improve MIRACH to suit their projects. With the community effort, we believe MIRACH will become a much better software than it already is.

We have on-going projects using MIRACH. Firstly, we are working to improve on an *in-silico* model that investigates cell fate determination of gustatory neurons of Caenorhabditis elegans (Saito *et al*., 2006) which we previously published by deriving properties from several recently published biological papers. Also, we are currently integrating MIRACH into our own framework, CSMLPipeline, for ease of use in our future projects. Lastly, we are working on a nice graphical user interface so that MIRACH will be even simpler and more appealing to use.

With increasingly large and complex pathway models such as the virtual human and virtual mouse models (Kitano, 2010) to be built, having an efficient and reliable model checker like MIRACH to verify the validity of these models will certainly bring about momentous impact.

ACKNOWLEDGMENTS

We are thankful to Sharene Lin for her help rendered in the manuscript preparation and proofreading of this manuscript. We would also like to thank Hakan Younes, Andre Fujita and Yamaguchi Rui for their many helpful discussions and suggestions.

Funding: Singapore National Research Foundation grant NRF-G-CRP-2997-04-082(d) (to L.W. and C.H.K., in parts); National University of Singapore NGS scholarship (to C.H.K., in parts).

Conflict of Interest: none declared.

REFERENCES

Baier,C. (1998) On Algorithmic Verification Methods for Probabilistic Systems. *Habilitation thesis*, University of Mannheim.

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

Clarke,EM. *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.

Fages,F. And Rizk,A. On the Analysis of Numerical Data Time Series in Temporal Logic. *In Proc. CMSB 2007*, 48–63.

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.

Kitano,H. (2010) Grand challenges in systems physiology. *Frontiers in Physiology*, **1**, 1–3

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.

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

Saito,A. *et al*. (2006) Cell Fate Simulation Model of Gustatory Neurons with MicroRNAs Double-Negative Feedback Loop by Hybrid Functional Petri Net with Extension. *Genome Informatics*, **17**(1), 100–111.

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.

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

Wilson,E. (1927) Probable Inference, the Law of Succession, and Statistical Inference. *Journal of the American Statistical Association*, **22**, 209–212.

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