Chapter 1
Formal methods for checking the consistency of biological models

Allan Clark, Vashti Galpin, Stephen Gilmore, Maria Luisa Guerriero and Jane Hillston

Abstract Formal modelling approaches such as process algebras and Petri nets seek to provide insight into biological processes by using both symbolic and numerical methods to reveal the dynamics of the process under study. These formal approaches differ from classical methods of investigating the dynamics of the process through numerical integration of ODEs because they additionally provide alternative representations which are amenable to discrete-state analysis and logical reasoning. Backed by these additional analysis methods, formal modelling approaches have been able to identify errors in published and widely-cited biological models. This paper provides an introduction to these analysis methods, and explains the benefits which they can bring to ensuring the consistency of biological models.

1.1 Introduction

Modelling complex systems on a computer allows us to investigate the rich dynamics of phenomena which are difficult or impossible to study at first hand. Making and analysing models may open the door to understanding but the insights and understanding gained depend crucially on the accuracy of the model and the legitimacy of its assumptions. Accurate modelling of complex systems often leads to difficult computational problems where inherent complexities of the problem such as multi-scale populations and widely-separated reaction rates present genuine technical challenges for robust numerical software. When grappling with these challenges

Allan Clark, Stephen Gilmore, Maria Luisa Guerriero and Jane Hillston
Centre for Systems Biology at Edinburgh, The University of Edinburgh,
Edinburgh EH9 3JU, Scotland, UK.
e-mail: {A.D.Clark, S.Gilmore, Jane.Hillston}@ed.ac.uk, mguerrie@inf.ed.ac.uk

Vashti Galpin
Laboratory for Foundations of Computer Science, The University of Edinburgh,
Edinburgh EH8 9AB, Scotland, UK, e-mail: Vashti.Galpin@ed.ac.uk

it is important not to lose sight of the fact that the quality of the insights obtained from the modelling depend critically on the quality of the model and that – in addition to carrying the burden of computing robust numerical results – modellers must also shoulder the burden of creating accurate biological models.

Adding to the difficulty of the problem, it is very easy to introduce simple errors which may have subtle effects which are extremely difficult to detect. For example, writing down the wrong variable in a differential equation may give rise to a model whose results look plausible (for example, there are no negative concentrations or other non-physical results) but which are essentially meaningless. From the perspective of a traditional differential equation integrator we have an entirely valid system of equations and a well-posed initial value problem; it is simply that it does not capture the phenomenon under study.

Domain-specific modelling languages such as process algebras and Petri nets specifically tailored for biology are helpful here because they define and enforce rules about the internal consistency of models which can allow simple modelling errors to be detected automatically. In this way, these languages can prevent the computational analysis of non-well-formed models and thereby – in some cases – stop erroneous conclusions being derived from erroneous models.

Evidence for the effectiveness of these methods can be seen when formal modelling is applied retrospectively to published models and at that point a previously-unknown error is discovered. It is possible for these errors to be either in the model itself, or in the computational analysis which was carried out in order to reveal the dynamics of the underlying biological process. An example of an error of the former kind in a model of the TNFα-mediated NFκ-B signal transduction pathway was uncovered using the techniques in [1]. An example of an error of the latter kind was uncovered as reported in [2] where Gillespie simulation is used in co-operation with continuous deterministic simulation to reveal a discrepancy which is traced to an incorrect use of a numerical integrator.

Two of the most important motivations for modelling a biological system are: (i) to identify gaps in the existing knowledge of the system, and (ii) to generate new insights and understanding without the need to perform laboratory experiments. In the former case we would work through validation where we try to discover whether the behaviour of the model agrees with current biological knowledge. In the latter case we investigate specific hypotheses via computational analysis instead of laboratory work.

Although these aspects are closely interconnected, there are existing computational and mathematical techniques which provide features particularly suitable to tackle one or the other. The process of identifying inconsistencies within models is an important phase which should always be performed before any conclusion is drawn from the results of the analysis of the model. Here, we focus on the use of analysis techniques which have their roots in formal language theory and program analysis. These range from static analysis and control flow analysis through to invariant generation, graph analysis and bisimulation. These methods enable us to identify flaws in models: both errors due to unknowns or incorrect hypotheses in the biological knowledge which are then unwittingly encoded in the model, leading to a

flawed model; and errors which are introduced during model construction such that the model does not faithfully represent current biological understanding.

Several formal methods have been developed (or adapted) in order to model and analyse biological systems, including Petri nets [3]; rewriting systems such as membrane systems [4] and Kappa [5]; and process algebras such as the biochemical stochastic π-calculus [6], Bio-PEPA [7] and the Continuous π-calculus [8]. Most of these languages are equipped with a discrete stochastic semantics, and some also allow for a continuous deterministic interpretation. The analysis techniques which are available differ for the various formalisms. For some of these languages it is possible to employ verification techniques such as model-checking.

These kinds of computational models can either be analysed statically via techniques which work at the level of the model structure, or can be dynamically executed via stochastic simulation [9] to produce time-course trajectories of amounts of the participating species. For languages which have a deterministic interpretation, numerical solution of the associated set of ordinary differential equations (ODEs) can be also performed, together with the various mathematical methods available for the analysis of ODE systems such as bistability, bifurcation, and continuation analysis. Existing modelling platforms such as the Bio-PEPA Eclipse Plug-in [10] allow modellers to perform model experimentation including parameter sensitivity analysis, components knock-down, and dose-response experiments.

1.2 Static Analysis

Viewed as a formal text, a biological model contains *definitions* of constituents of the model such as reaction rate constants, kinetic laws, initial concentrations, and chemical species; and it contains *uses* of these definitions. One simple check of self-consistency in the model is to determine that all definitions are used, and that everything which is used has been defined. This type of checking falls within the domain of *static analysis* because it can be performed without executing the model (via simulation or otherwise). The benefits of static analysis are enormous: a vast range of simple modelling errors can be easily and automatically caught at low computational cost.

Automatic static analysis seems such a simple and sensible check that it may be surprising to learn that not all programming languages enforce a static analysis check. For example, the Python programming language [11] does not and so a biological model implemented in Python has had less thorough automatic checking than a model implemented in Bio-PEPA or Snoopy [12] where a static analysis check is automatically enforced for every version of every model. Similar remarks apply to biological models coded directly in MATLAB [13]. Both Python and MATLAB have separate, optional static analysis tools (PyLint and M-Lint) which may be used by modellers, but are not required.

Static analysis based on the structure of a model is the first step which allows us to identify a number of errors, ranging from syntax errors such as trivial typos in

variable names to more subtle omissions of species behaviour. Static analysis can also be used in order to verify the presence or absence of deadlocks in the model behaviour and to clarify the causal and temporal relations between events.

Most static analysis checks relate to the internal consistency of a model. They generally do not require quantitative information such as kinetic rates and molecular concentrations. The use of high-level modelling languages helps modellers to reduce potential sources of errors by allowing them to define mnemonic names for system components, reaction kinetic laws and parameters. In addition to reducing the chance of introducing trivial modelling errors, the use of named definitions instead of numerical vectors for variables and parameters makes it possible to automatically perform a number of internal self-consistency checks as we will see.

In order to be considered valid, a formal model does not have to be only syntactically correct, but it must also satisfy a set of predefined plausible and common constraints. Formal languages are often domain-specific, thus allowing the notion of plausible and common constraints to be tailored to the specific domain. For example, as discussed below, static analysis checks on dependencies of kinetic laws on reactants in Bio-PEPA models will warn that simulations may produce negative results. A general purpose programming language or numerical computing environment will never warn about this because negative results might be legitimate for some modelling problems in other domains.

Throughout the remainder we illustrate some of the concepts with features of the Bio-PEPA language, as an example of a text-based formalism, and Petri nets, as an example of a graphical formalism.

1.2.1 The reagent and reaction-centric views of a model

A biochemical model can be viewed in one of two orthogonal ways which we call the reagent-centric and the reaction-centric views. In the former, for each reagent we list the set of reactions in which the reagent is involved. Conversely, the reaction-centric view displays, for each reaction, the set of reagents which are involved in that reaction and the associated effect that the reaction has on the population of that reagent. The BIOCHAM language [14] uses the reaction-centric view. In Bio-PEPA models are constructed in the reagent-centric view and the reaction-centric view is generated automatically by the software, in addition to some annotations on the reagent-centric view to be discussed below.

Providing both views of a model is important because they have complementary sets of advantages. The reagent-centric view is appropriate when scrutinising the behaviour of a particular component. Biologists are often trained to read reaction definitions and this view can assist the detection of errors such as misplaced reactants or products.

This is a strength of high-level modelling languages such as process algebras and Petri nets: they give us more than one view of a model. In contrast, a programming

language model such as a system of differential equations implemented in C or Python gives only a single view.

1.2.2 Missing and unused definitions

For a textual modelling language, such as Bio-PEPA, a straightforward static analysis check scrutinises all definitions of names and considers if any are not subsequently used. There is also a converse check to ensure that any names used have indeed been defined. These checks are fast to perform and catch simple errors made by modellers, commonly misspelt names as well as missing definitions. Whenever a definition is missing, this is considered as an error from which the software cannot recover and evaluation of the model is disallowed. When a definition remains unused, the software can still evaluate the model, but presents the user with a clear warning that something is possibly wrong. For example, in the Bio-PEPA tools these checks are applied to definitions of rate constants, kinetic laws and biochemical species.

Whenever a rate function or initial concentration uses a constant which lacks a definition this is likely to be an error on the part of the modeller, either they have forgotten to provide such a definition or the constant name has been misspelt at the point of use. Similarly, if a chemical species is given an initial concentration or molecule count but is not involved in any reaction as reactant, modifier or product then the model is incomplete – perhaps a component definition has been forgotten. Alternatively a name which is misspelled at the point of use will be detected as a missing definition and an unused definition.

An unused constant definition is likely to be caused by an error in either a rate function or an initial concentration or molecule count. An unused rate function is possibly missing behaviour in a species definition or perhaps the species definition is missing entirely. Finally, an unused species definition may signal that the modeller has forgotten to set the initial concentration or molecule count for one of the species in the model.

In the context of Petri nets, a graphical notation, a similar problem may arise if the model is not strongly connected, i.e. if there is not a directed path between every pair of nodes in the Petri net. A disconnected Petri net implies that there are two or more independent submodels. If static analysis reports that a model is not connected, this could be the desired model (in which case the distinct submodels can be analysed independently) but it could be due to reactions or species omitted in error.

1.2.3 Kinetic dependency analysis

Although missing and unused definitions can catch some simple errors made by the user, some similar errors may escape such analysis. There are two basic analyses which the Bio-PEPA Eclipse Plug-in performs over rate functions to catch further errors. The two analyses performed check that any species P, whose population affects a given rate function r, has a corresponding behaviour for r as a reactant, activator, inhibitor or general modifier in the definition of P. When this is not the case, the software brings this to the attention of the modeller since it is likely that the modeller has forgotten to add the reaction-behaviour to the species definition. Since we cannot know the role in which it should be added (reactant, activator, etc.) the modeller must be notified. The second analysis can be seen as the converse of this, it checks that for every reaction r for which a given species P is defined to be a reactant, activator or inhibitor then the corresponding rate function for r includes a reference to the population of P. Again, if this is not the case then it is likely that the model erroneously includes reaction r behaviour in the definition of P or has an incorrect definition for the rate function for r. These types of errors may lead to the amount of a species P undergoing inappropriate or insufficient updates as the model is exercised.

Kinetic dependency analysis does not guarantee that species are being used effectively in a kinetic law. For example, it is possible to construct pathological functions such as $(k \times E) + (P - P)$ which formally depends on the value of P because the symbol P occurs in the expression, but which uses P in such a way that its current value has no impact on the result of the function—which will always be equal to $k \times E$. It is not possible to detect all such false dependencies statically and so kinetic dependency analysis helps to detect when species have accidentally been omitted from function expressions but it can never provide a guarantee that their values have been used effectively.

1.2.4 Boundary nodes, sources, sinks and input/output paths

Simple analysis of the model can determine its boundaries and its interactions with its environment. For example, a Petri net without boundary nodes is a self-contained closed system.

We can consider the interface to a model in terms of both reactions (transitions in a Petri net model) and species (places in a Petri net). In general an input to the model is termed a *source* while an output is termed a *sink*. A source reaction is a reaction which has no reactants and at least one product (e.g. synthesis reaction). A sink reaction is one which has no products and at least one reactant (e.g. degradations). The reaction $r_1 \stackrel{def}{=} P + S \longrightarrow$ is an example of a sink reaction as mass is consumed without any being produced.

Similarly, a species is considered a source if it is involved in at least one reaction as a consumed reactant and no reactions as a product. Conversely, a species sink is a species which is involved in at least one reaction as a product and no reactions as a consumed reactant. In Bio-PEPA the species S with definition: $S \stackrel{def}{=} r_1 \downarrow S + r_2 \downarrow S$ is a source species as it can be consumed by the reactions r_1 and r_2 but it is never produced.

The presence of boundary nodes is not in itself an error because these kinds of species and reactions are perfectly valid in general open systems, but they can also be the cause of unintended behaviour. For instance, a Petri net with source species cannot be live and might lead to undesired deadlocks once the source node's initial amount is consumed. Similarly, a Petri net with source transitions cannot be bounded because its products could grow unboundedly.

The dual views offered by the Bio-PEPA software allow appropriate source/sink annotations to be added to reaction and species definitions. These annotations can provide useful error detection information to the modeller. Reaction source and sinks in particular denote that mass is not conserved by the model, although this may be intentional. Additionally the user can be warned if a source species has an initial population of zero, since in this case it will never have non-zero population and the reactions associated with it will never occur (assuming that the rate of the reactions do depend in a meaningful way on the population of the source species).

When boundary nodes are not caused by errors, but instead represent inputs or outputs with the environment, they can provide additional insight into the behaviour of the model: their identification, supplemented with the minimal sequences of reactions that link a given source/sink combination, illustrates the flow of mass through the model as an input/output behaviour. The input/output behaviour informs modellers about which sources influence which sink, and what is the effect on the overall model of the sequence of reactions leading from source to sink. For instance, consider a signalling pathway that describes the signalling cascade which, starting from a constant influx of a ligand, leads to the production of one target protein. This will have a source action and one sink species. When dealing with complex interconnected pathways, the input/output behaviour can help in understanding causal dependencies and in abstracting from the behaviour of part of the system.

1.3 Structural analysis

For graphical formalisms, such as Petri nets, a complementary method of investigating internal consistency is to view the model as a graph and consider it in terms of graph-theoretic concepts such as connectedness, reachability, paths, and cycles. An extensive body of work on structural analysis of models comes from Petri net-based techniques so we will discuss this type of analysis in Petri net terms.

Petri nets are graphical models of concurrent systems which contain *places* and *transitions*. Places contain *tokens* and firing a transition moves tokens from one place to another place. In the context of biological modelling, a Petri net is an au-

tomaton whose places represent molecular species and whose transitions represent reactions transforming reactants into products. Places can only ever be connected to transitions, and transitions to places—thus every Petri net defines a *bipartite* graph. Arcs are weighted, and their weights specify the stoichiometric coefficients of reactants. Places contain an arbitrary number of tokens which represent the current molecule count of each biochemical species. The current state of the system, termed the *marking* of the Petri net, is given by the number of tokens on each place.

The behaviour of a Petri net is defined by a firing rule, which specifies when transitions are enabled (if there are enough tokens for the involved reactants) and what is their effect (the changes in the number of tokens for the involved species).

Petri nets build on well-established mathematical foundations, which, in addition to the static and structural analysis techniques discussed here, support transient and steady-state analysis of the dynamic behaviour. Reachability analysis can be used to identify parts of the model which are not connected; boundedness analysis can be used to ensure uncontrolled growth of molecules is not possible; and invariant analysis can identify violations of the law of conservation of mass. See [15] for more details and examples.

1.3.1 Structural concepts in Petri nets

A number of structural properties have been defined for Petri nets. All of these can be checked statically, because they are based solely on the structure of the Petri net without consideration of the initial marking. Here we give only an informal overview; for more details and their formal definitions see for instance [16]. These common properties are often valid for biological models. If they are not satisfied, it can be an indication of an error in the model specification (though not necessarily). Some concepts, such as *pure* Petri nets and *ordinary* Petri nets, simply allow models to be categorised — this can provide a validity check to the modeller. For example, an ordinary Petri net is one in which all arcs are equal to 1 (which implies all stoichiometric coefficients are 1). Thus if the model is identified as being an ordinary Petri net but the model should include a homodimerization it is an indication that something is wrong. A pure Petri net is one in which there is no pair of nodes which are connected in both directions. This excludes models in which the same species is both a reactant and a product of a reaction.

Other concepts are related to the possible behaviours of the model when it is executed, i.e. when the Petri net is given a marking. For example a Petri net is considered to be *bounded* if the maximum number of tokens which can be on any place in the net is bounded by a constant. In biological terms this means that the amount of a species cannot grow without limit. A net is said to be *structurally bounded* if it is bounded for any initial marking. In some circumstances this property can be determined without executing the model and exploring its state space. A Petri net is *conservative* if for each transition the sum of the weights of the incoming arcs is equal to the sum of weights of the outgoing arcs (i.e. the model does not contain any

reaction which does not preserve the total number of molecules, such as complex formation reactions). Conservative Petri nets are always structurally bounded.

Another way of characterising nets is in terms of the conflict and causality structures in operation within the model. For example a Petri net is termed *static conflict free* if there are no two transitions which share an input place. In terms of a biological model this means that there is no competition between reactions for reactants. Again, identifying such properties can be a source of validation – or otherwise – for the model.

1.3.2 Invariants

For biochemical models it is commonly the case that the modeller wishes to respect conservation of mass, so that the total quantity of matter at any time throughout the simulation is the same as the total quantity at the start. This can be characterised as an *invariant* of the model, a weighted sum of species quantities which remains constant.

Where the model contains source and/or sink reactions, mass will, of course, not be conserved. Such reactions are generally an abstraction, the products of a source reaction do not suddenly materialise from nothing and nor do the reactants of a sink reaction disintegrate into nothing. However the real reactants (of source reactions) or products (of sink reactions) are outside the scope of the model. A sink reaction could also represent a transportation to a place outside the scope of a model. For example, an intracellular model may represent a movement to the extracellular environment by the use of a sink reaction. Similarly, a movement from the extracellular environment into the cell may be represented by a source reaction.

However, even for models in which the entire mass of the system is not conserved, we expect there to be local conservation. When mass is conserved within a set of components we call this an invariant, since the sum of the values of all the members of the invariant – weighted by some suitable coefficients – will remain constant throughout the simulation of the model. Invariant analysis is a well-established technique of structural analysis of Petri nets, and can also be applied to textual notations such as Bio-PEPA.

The stoichiometric information about the reaction network, captured as the *incidence matrix* for a Petri net, defines a system of linear inequalities. *Fourier-Motzkin elimination* can find both real and integer solutions to such a system of linear inequalities. These solutions are the weights which are used in the invariants which hold over the species of the system. These are termed *P-invariants* in the context of Petri nets. We can compute a minimal generating set of invariants with a version of the Fourier-Motzkin method which produces only integer solutions [17]. The algorithm works on a matrix representation of the stoichiometric information for the model. This has as columns the reactions of the model and as rows the species. Each element reflects the change in population of the given species when the given reaction is fired.

When the matrix is transposed, such that the rows become the reactions and the columns are the species, performing the same Fourier-Motzkin method over the transposed matrix we compute a new set of invariants. This new set of invariants is called the reaction invariants, or reaction loops (termed *T-invariants* in Petri nets). A reaction invariant consists of a set of reaction names with an integer coefficient associated with each reaction name. Such an invariant states that if this exact set of reactions is fired, the number of times indicated by each associated coefficient – in any order – then the model will be returned to the same state it was in at the start of the reaction invariant sequence.

The Bio-PEPA software computes both state and reaction invariants. Additionally the user can temporarily eliminate any of the reactions. This means that the chosen reactions are ignored for the purposes of the invariant analysis. In particular then the modeller may ignore all sink and source reactions in the model. If this is done then the entire set of species in the model should be covered by a list of invariants (which may be summed to create a single invariant which covers all of the species in the model). Where this is not the case, this indicates that somewhere in the model, mass is not conserved by a sequence of (non-source/sink) reactions. This probably indicates an error and the modeller should inspect their model carefully and either be able to repair it or explain why the conservation of mass is not observed.

$$
\begin{array}{rl}
a : & A \rightarrow B + C \\
b : & B \rightarrow D \\
c : & C \rightarrow D \\
d : & D \rightarrow A \\
prod : & \rightarrow A \\
degr : & D \rightarrow
\end{array}
\qquad
\begin{array}{l}
A \stackrel{def}{=} a{\downarrow}A + d{\uparrow}A + prod{\uparrow}A \\
B \stackrel{def}{=} b{\downarrow}B + a{\uparrow}B \\
C \stackrel{def}{=} c{\downarrow}C + a{\uparrow}C \\
D \stackrel{def}{=} d{\downarrow}D + b{\uparrow}D + c{\uparrow}D + degr{\downarrow}D
\end{array}
$$

Fig. 1.1 An incorrect model which is not covered by invariants even if the source reaction '*prod*' and the sink reaction '*degr*' are ignored.

Consider the model in Figure 1.1. This model will not be covered by any state invariants, however it does contain two reaction loops: $a + b + c + d + degr$ and $a + b + c + (2 \times degr) + prod$. The first of these is suspicious because it includes a sink reaction without a corresponding source reaction. Hence we employ our tactic of ignoring source and sink reactions and computing the set of invariants that this implies. This shows us that there are no invariants and this is highly indicative of an error in modelling.

In this particular case the correction could be that the reaction a should be modified to consume two molecules of A as in: $A + A \rightarrow B + C$ or the reaction d could be modified to consume two molecules of D as in: $D + D \rightarrow A$.

This is not an exhaustive list of possible corrections and in general there are many possible ways to correct this problem. However, the static analysis of invariant

coverage has highlighted the possibility of an error and this hopefully will help to ensure that less time is spent analysing incorrect models.

1.4 Verification of behavioural properties

As valuable as static and structural analysis are, there are some properties of models which cannot be assessed in this way. These instead require the model to be exercised to find all the possible configurations or states that it can reach. These correspond to the possible (although some of them may be unlikely) behaviours of the system. Such properties are termed *behavioural properties*, and generally are able to tell us more about the dynamics of the system.

One point of interest may be the extent to which behaviour in the model persists, or whether it reaches a state where no further reactions are possible (often termed *deadlock*). In Petri nets this is known as *liveness*: A Petri net is *live* if, for every transition, it is possible from any state to reach a state where this transition is enabled. A live Petri net is deadlock-free (i.e. the corresponding system does not have any state where no reaction is possible).

Classical techniques for checking the behaviour of models over a bounded state-space are considered in theoretical computer science under the heading of *model checking*. These techniques use efficient algorithms and data structures to determine whether logical formulae characterising desired (or undesired) behaviour are satisfied by a model. The presence of desired behaviour shows that the model is live, and that sequences of reactions can lead the model to good states. The absence of undesired behaviour shows that the model is safe, and that no sequence of reactions can lead the model to bad states. Both liveness and safety are desirable qualities for a model to possess.

Exact discrete-state model-checking where the complete state-space is generated can have applications in the modelling of biological processes (see [18]) but very often the memory needed to store the reachable state-space of a biological model exceeds the memory capacity of any computer system which we can access. Approximate statistical model-checking [19, 20] can be used instead and can give numerical results which are in very good agreement with those which are computed using more expensive techniques [21]. In this method, exhaustive generation of the reachable state-space is replaced by investigation of numerous trajectories over the state-space generated by simulation. Exact numerical solution of the underlying Markov chain is replaced by the execution of an ensemble of sufficiently many Monte Carlo simulations to approximate the measure of interest, and the probability of satisfaction of the logical formulae of interest is reported together with a confidence interval on the result.

1.4.1 Trace-based validation and model-checking

Novel techniques for detecting errors in models are now working not on the models themselves, but on their outputs generated through simulation. These complement static and structural methods beautifully because they consider the simulation results which are the output from a modelling study, not the model used as input. Such a perspective can allow these techniques to detect errors in simulators, as well as errors in models.

The Traviando trace analyser [22] is a discrete-event simulation trace analyser which provides graphical techniques to inspect and manipulate simulation trace output and to compute statistical results. These results include counts of reaction events by category and can allow the modeller to discover that their simulation run has not been long enough to allow some reactions to fire, or to discover that they only fire a small number of times. Either of these might indicate an error in the model, caught by trace analysis.

Another novel and promising technique integrates model-checking and trace analysis and can be applied even in the continuous domain to the results of numerical integrators. Fages [23], Donaldson [24] and others describe model checkers which inspect simulation outputs and evaluate quantified logical formulae over a single simulation trace (in contrast to the statistical model-checking approach, which requires an ensemble of traces generated from Monte Carlo simulations conducted in the discrete molecular regime).

When working in the continuous domain the output of a model is a deterministic simulation utilising continuous sure variables. This contrasts strongly with a stochastic simulation – which uses discrete random variables – because a single stochastic simulation run can be very far from the average-case behaviour of the model, and thus conclusions drawn from a single stochastic simulation can rarely give definitive insights into behaviour. In contrast, a time-course output from a continuous deterministic simulation returns sure trajectories for each of the chemical species in the model and thus carries more information which can be investigated in model-checking.

1.4.2 Equivalence relations

Once we are working at the level of the state space generated by a model as well as verification of behavioural properties, we can also consider whether alternative models of the same system are in some sense equivalent. At the static level this may be carried out by identifying an isomorphism between the constructs of the model — essentially showing that models are equivalent because they are made up of equivalent components. However, at the underlying level of the state space, often termed the *labelled transition system*, much richer and more flexible notions of equivalence can be defined.

A *bisimulation* (and a semantic equivalence, more generally) is a way to assess whether two different labelled transition systems have the same behaviour. A bisimulation is a symmetric relation between states of two labelled transition systems that requires that any two states in the relation both have transitions with similar enough behaviour, as captured by the labels of transitions, and the states that are a result of a pair of transitions are again in the bisimulation relation [25]. This captures the idea that the way the two models progress is the same for both models because states have transitions that match, as do the states that are targets of those transitions.

In the context of process algebras such as Bio-PEPA a variety of different bisimulations have been defined depending on what information about transitions is included in the labels [26]. In the original definition of bisimulation [25], equality between labels is required. However, it is possible to relax this requirement to allow for different notions of behaviour, as exemplified by the work on g-bisimulation in Bio-PEPA [27]. This applies a function g to the labels on transitions and two labels are determined to be similar enough whenever the function g gives the same value for both labels. This is a powerful mechanism as it allows identification of selected reaction names, and selection of information about the reaction that the transition represents. The function g is chosen by the modeller to express the information of interest when considering behaviour of the two systems. For example, a weak form of this equivalence relation, together with invariant analysis, is used to establish the equivalence of two previous models of the MAPK signalling cascade activated by EGF receptors [28, 29, 27].

1.5 Conclusions

As memorably expressed by Box and Draper [30], "all models are wrong, but some are useful". Models are built in order to help us to improve our understanding of systems and processes: if we already had perfect understanding then these models would not be needed. Our current imperfect understanding is necessarily encoded in the model. On top of this, all models simplify and abstract from details which are believed to be inessential in order that they can be tractable and usable for mathematical analysis. The belief that these details were inessential could be misplaced, and simply one part of our imperfect understanding. Because of this, and by their nature, we can never expect models of biological processes to be "correct".

However, we can – and should – expect our models to be consistent. If we intended to make a closed model where mass is conserved then we have an invariant which we expect to hold—computing the species invariants of our model allows us to check that mass is conserved and eliminates one possible source of error in our encoding. Similarly, if we have defined a reaction in our reaction network to involve species E and S then the kinetic law for the rate of that reaction should depend on E and S, and only on E and S. If not, then we have a likely source of error in the model.

Errors such as these may seem like simple carelessness but all of the evidence which we have seen seems to suggest that errors in the construction of formal models are very similar in nature to the errors which occur when writing a computer program. These are very rarely profound misunderstandings and are more likely to be simple mistakes such as a forgotten parameter or a forgotten function [31]. Nevertheless, the impact of such errors should not be underestimated.

Domain-specific modelling languages specialised to biological modelling are much more helpful in enabling simple errors to be caught automatically than are general-purpose numerical computing platforms. By building static analysis techniques into their modelling tools, domain-specific modelling languages can check that models are consistent, for every model, and for every revision of that model. State-of-the-art modelling platforms run inexpensive static analysis procedures every time that a model is saved after an edit has been made. This 'always-on' supervision helps modellers to find flaws in their models early, before analysis results are computed.

At best the methods which we have considered in this paper can help us to produce biological models which are internally consistent, with all parameters, kinetic laws and species used in the way in which we intended. Internal consistency in our models can never make them right, but lack of consistency will make them wrong.

Acknowledgements

Clark, Guerriero, Gilmore and Hillston are supported by the Centre for Systems Biology at Edinburgh. The Centre for Systems Biology at Edinburgh is a Centre for Integrative Systems Biology (CISB) funded by BBSRC and EPSRC, reference BB/D019621/1. The authors benefited from an introduction to invariant generation by Peter Kemper during his time as a SICSA Distinguished Visiting Fellow.

References

1. A. Clark, S. Gilmore, J. Hillston, and P. Kemper. Verification and testing of biological models. In *Proceedings of the Winter Simulation Conference (WSC)*, pages 620–630, December 2010.
2. M. Calder, A. Duguid, S. Gilmore, and J. Hillston. Stronger computational modelling of signalling pathways using both continuous and discrete-state methods. In *Proceedings of the Fourth International Conference on Computational Methods in Systems Biology (CMSB'06)*, volume 4210 of *Lecture Notes in Computer Science*, pages 63–77, 2006.
3. W. Reisig. *Petri Nets : an Introduction*. Springer-Verlag New York, Inc., 1985.
4. G. Păun. *Membrane Computing: An Introduction*. Springer-Verlag New York, Inc., 2002.
5. V. Danos and C. Laneve. Formal molecular biology. *Theoretical Computer Science*, 325(1):69–110, 2004.
6. C. Priami, A. Regev, W. Silverman, and E. Shapiro. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80(1):25–31, 2001.

7. F. Ciocchetta and J. Hillston. Bio-PEPA: a Framework for the Modelling and Analysis of Biological Systems. *Theoretical Computer Science*, 410(33–34):3065–3084, 2009.
8. M. Kwiatkowski and I. Stark. The continuous π-calculus: A process algebra for biochemical modelling. In *Proceedings of the Sixth International Conference on Computational Methods in Systems Biology (CMSB'08)*, number 5307 in Lecture Notes in Computer Science, pages 103–122. Springer-Verlag, 2008.
9. D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977.
10. A. Duguid, S. Gilmore, M. L. Guerriero, J. Hillston, and L. Loewe. Design and development of software tools for Bio-PEPA. In *Proceedings of the Winter Simulation Conference (WSC'09)*, pages 956–967. IEEE Press, 2009.
11. Python Web site. www.python.org, 2011.
12. C. Rohr, W. Marwan, and M. Heiner. Snoopy, a unifying Petri net framework to investigate biomolecular networks. *Bioinformatics*, 26(7):974–975, 2010.
13. MATLAB. `http://www.mathworks.com/`.
14. L. Calzone, F. Fages, and S. Soliman. BIOCHAM: an environment for modeling biological systems and formalizing experimental knowledge. *Bioinformatics*, 22(14):1805–7, July 2006.
15. M. Peleg, I. Yeh, and R. Altman. Modeling biological processes using Workflow and Petri Net models. *Bioinformatics*, 18(6):825–837, 2002.
16. M. Heiner, D. Gilbert, and R. Donaldson. Petri Nets for Systems and Synthetic Biology. In *Formal Methods for Computational Systems Biology (SFM'08)*, volume 5016 of *Lecture Notes in Computer Science*, pages 215–264. Springer-Verlag, 2008.
17. J. Martinez and Manual Silva. A Simple and Fast Algorithm to Obtain All Invariants of a Generalized Petri Net. In *Selected Papers from the First and the Second European Workshop on Application and Theory of Petri Nets*, pages 301–310. Springer-Verlag, 1982.
18. M. L. Guerriero. Qualitative and Quantitative Analysis of a Bio-PEPA Model of the Gp130/JAK/STAT Signalling Pathway. *Transactions on Computational and Systems Biology XI*, 5750:90–115, 2009.
19. K. Sen, M. Viswanathan, and G. Agha. On statistical model checking of stochastic systems. In *Computer Aided Verification*, volume 3576 of *Lecture Notes in Computer Science*, pages 266–280. Springer-Verlag, 2005.
20. H. L. S. Younes and R. G. Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Information and Computation*, 204(9):1368–1409, 2006.
21. F. Ciocchetta, S. Gilmore, M. L. Guerriero, and J. Hillston. Integrated simulation and model-checking for the analysis of biochemical systems. In *Proceedings of the Third International Workshop on Practical Applications of Stochastic Modelling (PASM'08)*, volume 232 of *Electronic Notes in Theoretical Computer Science*, pages 17–38. Elsevier, 2009.
22. P. Kemper and C. Tepper. Automated trace analysis of discrete-event system models. *IEEE Transactions on Software Engineering*, 35(2):195–208, 2009.
23. F. Fages and A. Rizk. On the analysis of numerical data time series in temporal logic. In *Proceedings of Computational Methods in Systems Biology (CMSB'07)*, volume 4695 of *Lecture Notes in Computer Science*, pages 48–63. Springer, 2007.
24. R. Donaldson and D. Gilbert. A model checking approach to the parameter estimation of biochemical pathways. In *Proceedings of Computational Methods in Systems Biology (CMSB'08)*, volume 5307 of *Lecture Notes in Computer Science*, pages 269–287, 2008.
25. R. Milner. *Communication and concurrency*. Prentice Hall, 1989.
26. V. Galpin and J. Hillston. A semantic equivalence for Bio-PEPA based on discretisation of continuous values. *Theoretical Computer Science*, 412(21):2142–2161, 2011.
27. V. Galpin. Equivalences for a biological process algebra, 2011. Submitted for publication.
28. B. Schoeberl, C. Eichler-Jonsson, E. D. Gilles, and G. Muller. Computational modeling of the dynamics of the MAP kinase cascade activated by surface and internalized EGF receptors. *Nature Biotechnology*, 20:270–375, 2002.
29. Y. Gong and X. Zhao. Shc-dependent pathway is redundant but dominant in MAPK cascade activation by EGF receptors: a modeling inference. *FEBS Letters*, 554:467 – 472, 2003.
30. G. Box and N. Draper. *Empirical Model-Building and Response Surfaces*. Wiley, 1987.
31. D. E. Knuth. The errors of TEX. *Software Practice and Experience*, 19:607–685, July 1989.