

Designing a Front-End for Bio-PEPA

Laurence Loewe

Centre for Systems Biology Edinburgh, University of Edinburgh, Edinburgh EH9 3JU, Scotland

Laurence.Loewe@ed.ac.uk

Abstract: We explore the possibilities for designing a front-end for Bio-PEPA that simplifies the construction of models for biologists. The semantics of the approach also allows direct ODE and SSA analyses. The structural similarities of problems in molecular systems biology and population biology suggest that a common language may serve both communities well.

In the opinion of some biologists, chemical reaction systems are easier to specify in terms of their reactions than in terms of their process algebra components [1]. Since both approaches can be used to express the same reaction systems, algorithms can be specified that translate one into the other [1]. The present work relies on such algorithms to design a new front-end language that allows the automatic construction of Bio-PEPA models [2].

Bio-PEPA models facilitate the use of Ordinary Differential Equations (ODEs), simulations that employ the Stochastic Simulation Algorithm (SSA) developed by Gillespie [3], numerical analyses of Continuous Time Markov Chains (CTMCs) and formal equivalences for further analysis of system properties [2]. By applying different algorithms the front-end language can also be used to directly specify ODEs, SSAs and possibly CTMCs. Such capabilities allow for extended consistency checks between different analyses and more freedom to choose analysis tools. Figure 1 summarizes the various possibilities of automated model conversion in this framework. A front-end for Bio-PEPA could reduce the time required for encoding a complex reaction scheme into a research grade model implementation and thus ease the access to the analytical power of Bio-PEPA tools [2].

Here a proposal is presented for the design of such a front-end language. Important principles guiding the design effort are reviewed before some key concepts and syntax constructs are presented. Abstract data structures that help with the translation and some automatable checks that can be applied to find errors in front-end models are given at the end.

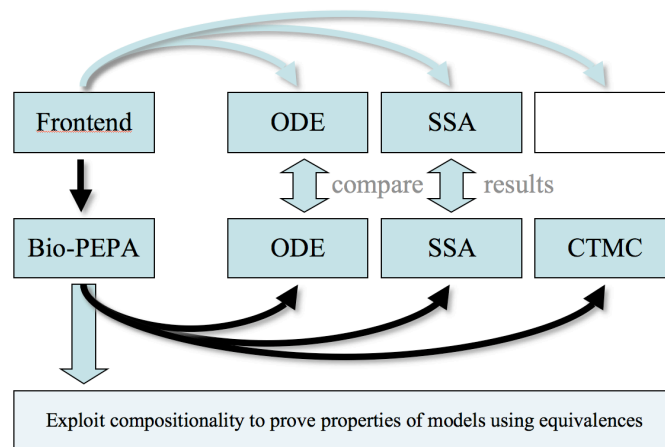


Figure 1: Modeling framework. Black arrows denote the main intended automatic flow of model information; narrow arrows denote a flow of information that is automatable in principle; wide grey arrows denote activities that cannot be automated easily at the moment.

Design principles

Language design inevitably requires striking a balance between various conflicting goals like making a language easy to learn and its ability to describe complex systems. The following guiding principles are proposed for the design of the front-end and are open to discussion.

1. **Bio-PEPA compatibility.** All statements in the front-end language have to respect the process algebra operators of Bio-PEPA and must be translatable into Bio-PEPA code.
2. **Reuse syntactic structures from the problem domains.** To increase the ease of use for biologists use chemical reactions to describe elementary actions of the system.
3. **Be as general as possible.** ODEs and SSAs are not only used in molecular systems biology, but also in epidemiology and population biology. It is desirable to choose keywords that allow for a natural description of problems from different subdisciplines.
4. **Support writing readable code.** While this could also be described as the overall goal, the front-end can contribute in a rather specific way by supporting long *SpeakingVariableNames* (as frequently used in well-documented code) *and* short *symbols* (as frequently used in mathematical equations). By allowing front-end identifiers that combine both, the front-end can support automatic translation into longer, better-documented models or shorter, mathematically more readable models.
5. **Keep together what belongs together.** When biologists want to understand a biochemical equation they need certain types of information. The language should support structures that allow for all this information to be put together effortlessly, minimizing the number of manual lookups required.
6. **Minimize syntactic clutter** without losing readability. Whenever syntactic structures can be omitted without becoming imprecise, omit, however not at the cost of readability. Features that are not needed are simply omitted instead of specifying their ‘non-use’.
7. **Support structures to hide complexity** by allowing one to define reusable building blocks (e.g. groups of rules that occur together at different locations).
8. **Support automated checks for common non-trivial errors.** For example (i) identifiers for rules have to be unique, (ii) long and short entity names have to be consistent and (iii) the total number of Bio-PEPA components that results from expanding valid front-end code may be limited to comply with limitations of analysis tools.

Problem domains and key concepts

There are two prominent areas in biology that can benefit from the modeling framework above: molecular systems biology and the modeling of population dynamics in epidemiology and ecology. ODEs and stochastic simulations have long been used in both and easier modeling can be expected to facilitate progress in both. Thus the front-end defines the following key concepts:

Type = *type* of molecular species or *type* of individual in a population. One *type* in each location produces one sequential Bio-PEPA component, one ODE and one SSA entity counter.

Rule = a possible transition of certain *types* into other *types*. A transition can be a chemical reaction or an individual-based event like birth or death. Each *rule* is named, written in the form of a chemical reaction and attached to a *law* that determines the rate at which the *rule* changes the system. The same rule in two different locations is treated as two different rules.

Law = a mathematical equation that takes some parameters and the *types* of a *rule* as input and produces the specific equations that describe the relevant rates of change in the system.

Location = a 2D or 3D compartment of specified size where all *types* in the compartment are assumed to be well-mixed. A location has specific *rules* for describing the behaviour of the *types* in it and for specifying transport to and from the immediate outer or neighbouring locations. To minimize the combinatorial explosion of equations resulting from many locations, not every *type* is defined in every

location, as some *types* may never be transported to some *locations* by the given *rules*. Locations can be nested, stacked in arrays or loosely connected. They can have types in surfaces or borders as well.

Model structure

Each model is described in one file that can *include* other files to reduce redundancy in large modeling enterprises. With a view to help organize computing projects, each model can specify a name, parameter combination, input parameters, output parameters and computing requests. Here is an overview model structure (only equivalents to the bold statements below are required in a minimal toy model):

```

model Name
parametercombination Name      # facilitates referring to different simulations
inputparameters { include Name } # all parameters varied in computing requests
outputparameters { include Name } # tell some underlying tools what to report
computingrequests { include Name } # tell tools what to compute automatically

laws { include Name } # define laws that are not MassAction (= MA =predefined)
environmentchanges { include Name } # arbitrary functions with time dependency
ruleset RulesetNameA { include Name } # for convenient referral to a set of rules
locationset LocsetNameA { include Name } # can contain other locations with rules
parameters { include Name
    # startconditions like initial concentrations or type counts
    # location parameters that can apply to several locations (e.g. volume)
    # environment parameters that are used in the environmentchanges
    # everything else that is fixed for a model and not specified in location
    # inputparameters override the parameters specified here
}

location UserGivenLocationName
    volume Value [Unit] {
        initial A count 100 [] # shortest possible arbitrary example
        initial B concentration 2 [nmol/L] # Units are for documentation
        rule Name { A + B --[ MA : r = 0.5 [1/s] ]--> C } # the simplest rule
        RulesetNameA # to add more rules, simply add rulesets
        LocsetNameA # add sublocations that can be nested many times
    } # end of model

```

Identifiers are italicized and those following “include” have to be defined elsewhere, either in the same file or in a different file (first look for a “define *IncludedName* {}” in the same file and if it does not exist, assume the identifier is a filename with the define statement). *Include* statements can be nested. Several *rulesets* and *locationsets* can be defined under different names at the highest level, where differently named sets can include the same *rules* and *locations*. Comments are denoted by #. A parameter is defined by

```
parameter Name = Value [Unit]
```

where *Value* can be a number, an *environmentchange* or an equation that includes other parameters and *Unit* is for documentation. All definitions are global except those given within a location; there

```
public parameter ... OR private parameter ...
```

defines parameters that are visible or invisible by default for all locations within this location, respectively. The same scope definitions work for *rules*, except that global rules will have to be declared as `public` in the most global location. Start conditions are specified in terms of counts or concentrations that are converted with the help of the size of the corresponding location:

```
initial Type@*@inner@outer@location count|concentration Value [Unit]
```

Here @ is used to separate nested locations and * indicates all locations at the corresponding level. If a starting condition is specified in the right location, no @ is needed.

Rules and Laws

Rules are named to make the parameters of their *laws* unique by using *rule* names and *location* names as prefixes. Since each *rule* is bound to a *location*, its name needs to be unique only within that *location*. To make *rules* globally unique, all nested *location* names are used as prefixes in the construction of the output model. Each rule specifies the *law* it follows and all ‘non-type’ parameters for that *law* are given either by referring to globally defined parameters or by including actual numbers directly, where parameters are separated by []. All ‘type’ parameters of a law are automatically defined by all reactants and their stoichiometry, which are accessible for building equations within the context of a law.

```
rule Name {  
  A + B --[ LawName : ParameterName1 = Number [Unit] Par2 = DefinedParam [...] ]--> C  
}
```

The *law* `MassAction`, abbreviated by `MA`, is built into the front-end and can be considered a group of laws considering all possible combinations of reactant stoichiometries as input. All other parameters required by the definition of the law are given between the colon and the end of the squared bracket.

Combo Names

A recurring problem in modeling is the specification of variable names. While mathematical equations are usually expressed in symbols that are as concise as possible to allow focusing on the structure of the equation, readable implementation code usually requires LongSpeakingNames to help coders identify quickly which entity is meant. Translating between these two notations can be cumbersome and error prone. To ease this process, the front-end will allow the specification of “combo names” that combine the best of both worlds. A combo name is an identifier that consists of two parts, which are separated by a syntactical structure known to the front-end:

```
LongSpeakingName__ShortName      for example:  mRNA_decay_rate__m1
```

where the long and short names are separated by two underscores (which are otherwise forbidden in identifiers). While long names may be expected to be unique within the group of long names, a short name in a complex model can by accident easily refer to two different things. The proposed structure allows for an internal check and for automated production of either readable or concise Bio-PEPA model definitions.

Support for rule-refinement

One can envision a feature, where the *law* that is usually specified within the arrow of a *rule* is expressed in the form of a small submodel of more elementary rules that take the ‘reactants’ as input and produce the ‘products’ of the superior rule. Since these sub-reactions do not interact with the outside world, one could envision Bio-PEPA to analyze them separately in an attempt to reduce state space without loss of precision.

Comparison to other work

This front-end is not the only rule-based system that can be used to describe chemical reactions. CMDL, as implemented by Dizzy (<http://magnet.systemsbiology.net/software/Dizzy/>), the kappa calculus [4] and BioCHAM (<http://contraintes.inria.fr/BIOCHAM/>) also use rules, but they do not support the translation into Bio-PEPA and use only a part of the analysis framework above. They also do not support combo names and rule-refinement, which may help in the analysis of some models.

Translation Data Structures and Automated Checks

While parsing a model, the front-end will construct a list of all `ElementaryTypes` and `ElementaryRules`. These contain the *short*, *long* and *combo* names for all *types* and *reactions* after making them unique by considering their *location* context, a process that can also be called “flattening”. This process allows also for automated consistency checks, as *short*, *long* and *combo* names have to give precisely the same list. Early rejection of inconsistencies by the front-end is expected to substantially reduce the cost of debugging models. Flattening will also reveal, how many sequential Bio-PEPA components will be required to describe the specified model and if some predetermined limit is exceeded, the front-end can issue a warning to avoid problems further down in the analysis workflow. `ElementaryTypes` are often affected by multiple rules. Since all influences are additive, these can be viewed as combinations of `ElementaryChanges`, which are used to compile the Bio-PEPA target code. Each `ElementaryChange` stores:

```
ElementaryChange {
  1 ElementaryType      # reactant or product
    - ComboName        # as specified in the front-end
    - LongName         # for model output where names are readable
    - ShortName        # for model output where equations are readable
  1 Equation           # quantifies size of change in FlatType
    - ElementaryRuleID + LawID + TermID in resulting equation
    - CodeStringFor Bio-PEPA
    - CodeStringFor Stochkit-SSA
    - CodeStringFor Matlab-ODE
    # each equation is stored as Long, Short and Combo notation
}
```

To produce the description of the sequential components of a corresponding Bio-PEPA model, one needs to compile one sequential Bio-PEPA component statement for each `ElementaryType`, which appropriately combines all the `ElementaryChanges` that affect the corresponding `ElementaryType`. To produce code that computes the propensity functions in SSAs, one needs to compile one statement for each elementary reaction. Other translations work analogously. To complete the translation process, each target requires additional code that specifies initial conditions, additional functions and other details, which are not discussed here.

Conclusion

An early draft for a possible front-end language for Bio-PEPA was presented. It provides some mechanisms to hide complexity in model definitions and to check their internal consistency. Feedback for the improvement of the specification of this front-end language is welcome.

Acknowledgements: I thank Jane Hillston for helpful comments on this manuscript. The Centre for Systems Biology at Edinburgh is a Centre for Integrative Systems Biology (CISB) funded by BBSRC and EPSRC, reference BB/D019621/1.

References

1. Guerriero ML, Heath JK, Priami C: **An Automated Translation from a Narrative Language for Biological Modelling into Process Algebra**. *Lecture Notes in Computer Science* 2007, **4695**:136-151.
2. Ciocchetta F, Hillston J: **Bio-PEPA: a framework for the modelling and analysis of biological systems**. *School of Informatics, University of Edinburgh* 2008, **EDI-INF-RR-1231**:<http://www.inf.ed.ac.uk/publications/report/1231.html>.
3. Gillespie DT: **Stochastic simulation of chemical kinetics**. *Annu Rev Phys Chem* 2007, **58**:35-55.
4. Danos V, Feret J, Fontana W, Harmer R, Krivine J: **Rule-based modelling of cellular signalling**. *Lecture Notes in Computer Science* 2007, **4703**:17-41.