simgi - A Stochastic Gillespie Simulator for Molecular Systems

Author: Markus Dittrich
email: haskelladdict at users dot sourceforge dot net
Version: 0.1 (05/30/2009)

Contents

Introduction

simgi is a fairly simple and straightforward stochastic simulator based on Gillspie's[1] direct method. **simgi** is implemented in pure Haskell, command line driven and comes with a flexible simulation description language called Simgi Model Generation Language (SGL). More information is available from the project summary page.

Status

The 0.1 release of **simgi** provides a fully functional simulator but should still be treated as an alpha version since several parts of the code are currently not fully optimal. This is particularly true for the random number generator which presently leverages the StdGen instance of RandomGen[2] and is probably not sufficient for large simulations in terms of random number quality. Later revisions of simgi will have a more sophisticated random number generator. Nevertheless, for small systems (such as the examples in the *Models/* directory) the current implementation should be sufficient.

Download

The current release of simgi can be downloaded here.

Compilation

Compilaton of **simgi** requires

- >=ghc-6.10
- >=gmp-4.3

To compile the documentation (not required), you will also need

- >=docutils-0.5
- latex, e.g., tetex or texlive

Building of **simgi** can be done either via

- the standard `make, make check, make install`
- or via cabal

Simgi Model Generation Language (SGL)

simgi simulations are described via Simgi Model Generation Language (SGL). The corresponding simulation files typically have an *.sgl* extension, but this is not enforced by the **simgi** simulation engine. A SGL file consists of zero or more descriptor blocks of the form

```
def <block name>

  <block content>

end
```

The (but see[3]) formatting of the input files is very flexible. In particular, neither newlines[4] nor extraneous whitespace matter. Hence, the above SDL block could also be written on a single line. However, it is strongly recommended to stick to a consistent and "visually simple" layout to aid in "comprehending" the underlying model.
Comments can be added to the SGL file and are parsed according to the Haskell language specs

- simple line comments begin with a `--` token and treat everything until the next newline as a comment, including valid SDL commands. Hence, SDL blocks containing line comments need to be separated by newlines in order to be parsed correctly.
- block comments begin with a `{-` token and end with a `-}` token. Everything within a comment block is ignored by the parser and block comments can be nested.

Currently, the SDL specs define the following block types with their respective block commands and block content:
parameter block: *<block name> = parameters*

The purpose of the parameter block is to describe the global simulation parameters. The following parameters are currently supported:

***time* = <*double*>** Maximum simulation time in seconds. Default is 0.0 s.

***outputIter* = <*Integer*>** Output will be kept in memory and written to the output file and stdout every *outputIter* iterations. Larger values should result in faster simulations but require more system memory. Default is to write output every 10000 iterations.
Note: *outputIter* only affects how often output is written to the output file, not how much is being accumulated during a simulation (see outputFreq parameter).

***outputFreq* = <*Integer*>** Frequency with which output is generated. Default is 1000.

***systemVol* = <*double*>** Volume of the simulation system in liters. This is needed to properly compute the reaction rates in molar units. If rates should rather be interpreted as reaction propensities (like in[1]) please set *systemVol* = *nil*. Default is a system volume of 1.0 liter.

***outputFile* = <*quoted string*>:** Name of the output file. This is the only required parameter in the parameter section. If not given, the simulation will terminate.

molecule block: *<block name> = molecules*

This block consist of a list of pairs of the form

```
<String> <Integer>
```

giving the name of each molecule and the number of molecules present initially. For example, the following molecule definition block defines molecules `A` and `B` with initial numbers of 100 and 200, respectively

```
def molecules
  A 100
  B 200
end
```

reaction block: *<block name> = reactions*

This block describes the reactions between molecules defined in the molecule block. Reactions are specified via

```
reactants -> product { rate expression }
```

Here, `reactants` and `products` are of the form

```
<Integer> <String> + <Integer> <String> + .....
```

In this expression, `<String>` is the reactant or product name as defined in the molecule block and `<Integer>` an optional integer specifying the stoichiometry. If `<Integer>` is not explicitly given, it is assumed to be 1.

The reaction rate can either be a fixed value of type `<Double>` or else an mathematical expression involving `<Double>`, molecule names, and the current simulation time. Hence, **simgi** rate expressions can be arbitrary complex functions of the instantaneous simulation time and the instantaneous numbers of any molecule in the model. The parser will interpret any string in the rate expression as a molecule name in a case sensitive fashion, a mathematical operator or function (see[5] for supported functions), or the special variable TIME which refers to the current simulation time. Hence, do **not** use any of the mathematical keywords as a molecule name; this leads to undefined behavior.

Here is an example reaction block for the two molecules `A` and `B` defined above:

```
define reactions
  2A + B -> A  { 10.0e-5 }
  B      -> A  { 2.0e-5 * A * exp(-0.5*TIME) }
end
```

In the first reaction, 2 `A` molecules react with one `B` to yield another `A` at a rate of 10.0e-5 1/(Mol s). The second reaction describes a decay of `B` back to `A` at a rate that is computed based on the instantaneous number of `A` molecules present and which decays exponentially with simulation time.

Internally, rate expressions are converted into a compute stack in RPN format which is evaluated at run-time. Even though this procedure is fairly efficient, there is some numerical overhead incurred at each iteration and the use of complicated rate expressions should therefore be avoided if possible.

Example Input Files

Below are several example input files detailing the use of SGL:

- Lotka-Volterra Model
- Brusselator Model
- Oregonator Model

These are also available in the *Models/* sub-directory in the source tree.

Bugs

Please report all bugs and feature requests to <haskelladdict at users dot sourceforge dot net>.

References

[1] Daniel T. Gillespie (1977). "Exact Stochastic Simulation of Coupled Chemical Reactions". The Journal of Physical Chemistry 81 (25): 2340-2361

[2] http://hackage.haskell.org/packages/archive/random/1.0.0.1/doc/html/System-Random#globalrng.html

[3] Since **simgi** currently is an alpha version there may be fairly drastic changes to the SDL specs in future releases until the first beta release.

[4] An exception to this rule are line comments starting with `--` which ingnore everything until the next newline.

[5] Rate expressions can contain any arithmetic expression involving the standard operators "+", "-", "*", "/", "^" (exponentiation), and the mathematical functions `sqrt, exp, log, log2, log10, sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, asinh, acosh, atanh, acosh, atanh, erf, erfc, abs`.