Simgi - A Stochastic Gillespie Simulator

Markus Dittrich

02/08/2010

This manual covers simgi v0.3 (released 02/09/2010).

Contents

Introduction

simgi is a small but efficient stochastic simulator based on Gillespie's direct method and uses a 64 bit implementation of the Mersenne Twister algorithm as pseudo random number source. simgi is command line driven and features a powerful and flexible model generation language.

Download

The current release of simgi (version 0.3) can be obtained here in both source and binary formats.

Installation and Compilation

The simgi binary packages for 32 bit and 64 bit Linux should run on any recent distribution with *libgmp* installed.

simgi is written in pure Haskell and compilation requires a working ghc Haskell compiler as well as the following additional packages:

- >=ghc–6.10
- >=gmp–4.3
- >=mersenne-random-pure64
- bc (for the test suite only)

The optional document generation requires

- >=pandoc–1.4
- latex (e.g. texlive)

simgi can be build in one of two ways

1) the standard *make, make check, make install*

2) via cabal

Simgi Model Generation Language Syntax

General Syntax

simgi simulation models are described via *Simgi Model Generation Language (SGL)* inside a plain text ASCII file. Syntactically, each *SGL* file consists of a number of blocks each describing a certain aspect of the simulation such as variables, parameters, molecules, or reactions. Each block has the following structure

```
<block name>

  <block content>

end
```

Please note that there is no need to put block name and content description on separate lines even though it is highly recommended to do so for ease of reading.

Even though syntactically *SGL* does not require blocks to be arranged in any specific order, semantically, each identifier used in a given block has to be defined when first used. Hence, the molecule definition block has to be placed before the reaction definition block.

Comment on Syntax Notation

In the syntax specification below, the following notation is used

- If a syntactic expression can contain either one of a number of options A, B, C, ... this is specified as `( A <|> B <|> C <|> ..)`.
- If a syntactic element A is optional, it is enclosed in brackets `[ A ]`.
- If a syntactic element B can repeat zero or more times it is enclosed in curly braces `{ A }`.
- Literal braces, parenthesis, etc. are always enclosed in single quotes, e.g, '(' or '}'.

Please note that this notation does not apply in the examples given, which are always meant to be literal code examples.

Comments

Comments inside SGL follow the standard Haskell convention. Multiline comments can be wrapped inside {— —}. Single line comments start with — and ignore everything until the next newline. For example,

```
{-- this is
    a multiline
    comment
--}

foo = bar   -- this is a single line comment
```

Identifiers

SGL identifiers have to start with a lower or uppercase letter followed by any number of lower or uppercase letters, digits, or underscores. Please note that identifiers can not be any of the keywords or mathematical functions available in simgi.

Numerical Identifiers and Statement Blocks

Inside SGL, some identifiers are assigned numerical values. Examples are variables, initial molecule numbers, reaction rates, or event definitions. A numerical value can either be a literal *Double* value or a *statement block*. The latter is a mathematical expression enclosed in curly braces that evaluates to a *Double* literal either at parse-time or at run-time. Whether a *statement block* is evaluated at parse or run-time depends on the definition block in which it occurs as detailed in the description for each block below.

statement blocks which are evaluated at parse-time may contain only mathematical expression involving *Double* literals and variable values. *statement blocks* which are evaluated at run-time can in addition contain the instantaneous counts of molecules as well as the current simulation time accessible via the keyword *TIME*. These two types of *statement blocks* are referred to as *parse-time statement blocks* and *run-time statement blocks*, respectively.

Assuming that *foo* and *bar* are variables, the following are valid statement blocks

Example:

```
{ 3.0*foo + bar^2 }                -- parse-time statement block

{ 3.0*exp(-foo/TIME) + bar*TIME }  -- run-time statement block
```

Inside *statment blocks* simgi supports the use of the following mathematical functions: `sqrt, exp, log, log2, log10, sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, asinh`

Note 1: Depending on their numerical complexity, *run-time statement blocks* incur a computational overhead and should therefore be avoided if possible.

Note 2: SGL statements which expect an *Integer* value instead of a *Double* will use `floor()` to for rounding.

Variable Definition Block

<block name> = variables

This block allows the definition of variables which can then be used inside any *statement block* in the remainder of the SGL file. Variable assignments are of the form

<variable name> = (*Double* <|> *parse-time statement block*)

Since the variable block will be evaluated only after it has been fully parsed, variables which depend on other variables via *parse-time statement blocks* do not have to be defined in order.

Note: Users have to ensure that variable definitions do not contain circular references since this may lead to infinite evaluation loops.

Parameter Definition Block

<block name> = parameters

The parameter block defines the main simulation control parameters. It can be left out and all parameters will then assume their default values. Available parameter options are

- **time** = (*Double <|> parse-time statement block*)

  Maximum simulation time in seconds. Default is 0.0 s.

- **outputBuffer** = (*Integer <|> parse-time statement block*)

  Output will be kept in memory and written to the output file and stdout every *outputBuffer* iterations. Larger values should result in faster simulations due to reduced I/O but will require more system memory. Default is to write output every 10000 iterations.

  Note: The value of *outputBuffer* only affects the chunk size in which output is written to the output file, not how much output is actually generated during a simulation (see outputFreq parameter).

- **outputFreq** = (*Integer <|> parse-time statement block*)

  Iteration frequency with which output is generated. Default is to generate output every 1000 iterations.

- **systemVol** = (*Double <|> parse-time statement block <|> nil*)

  Volume of the simulation system in dm^3 nor *nil*. Unless *nil* is specified, reaction rates are interpreted in molar units. If *nil* is given instead, rates are interpreted as reaction propensities (see 1). The default is a system volume of 1.0 dm^3.

Molecule Definition Block

<block name> = molecules

The molecule definition block is used to declare all molecular species present in the simulation and assign initial molecule counts to each species. Molecule assignments are of the form

<molecule name> = (*Integer <|> parse-time statement block*)

Note: In contrast to many ODE simulation packages, simgi requires the specification of molecule numbers not concentrations.

Example:

```
molecules
  A = 100
  B = { 10 * someVar }
end
```

Reaction Definition Block

<block name> = reactions

The reaction definition block is the heart of the simgi model and describes the dynamics of the underlying chemical system. Rate expressions are of the general form

$$a_1\ R_1 + \ldots + a_N\ R_N \text{ -> } b_1\ P_1 + \ldots + b_M\ P_M \mid \text{<rate>} \mid$$

where

$$\text{<rate>} = (\textit{Double} \text{ <|> } \textit{run-time statement block})$$

and a_i, R_i, b_j, P_j are the stoichiometric coefficients and names of reactants and products, respectively. If any of the stoichiometric coefficients is omitted it is assumed to be equal to 1.

Example:

```
reactions
  2A + 3B  -> C  | 1e-4 |
  10B + 4C -> D  | { 2.0 * exp(-A/TIME) } |
end
```

Here, the rate for the second reaction is given by a *run-time statement block* and exponentially decays as a function of the instantaneous concentration of species A and simulation time.

Event Definition Block

<block name> = events

Events allow users to interact with a simulation at run-time. The event block consists of a list of event statements of the form

['('] *trigger expression* [')'] => '[' *action expression list* ']'

Here, *trigger expression* defines when an event takes place and *action expression* specifies the action triggered by the event. During each iteration of the simulation each *trigger expression* will be evaluated and if *True* all actions in the associated *action expression list* will be executed.

A trigger expression consists of one or more trigger primitives combined via the boolean operators `&&` (*AND*) and `||` (*OR*)

> *trigger expression* = *trigger primitive* { (&& <|> ||) *trigger primitive* }

Trigger primitives each consist of two *run-time statement blocks* or *Double* literals combined via a relational operation

> *trigger primitive* = ['('] (*run-time statement block* <|> *Double*) (== <|> < <|> > <|> <= <|> =>) (*run-time statement block* <|> *Double*) [')']

An *action expression* consists of a comma separated list of *action primitives*

> *action expression* = *action primitive* { , *action primitive* }

where each *action primitive* is an assignment statement of the form

> (*variable* <|> *molecule name*) = (*Double* <|> *run-time statement block*)

Example:

```
events
  A == 100  => [ A = {A/100} ]
  (A == 100 && B == 0)  => [A = 10, B = {A/10}]
  A == 10 || C == 50 => [C = 10, A = {A+C*TIME}]
end
```

Output Definition Block

<block name> = output

This block defines the name of the output file and the type of simulation output that will be produced and written to it. Presently, simgi will only generate a single output file and produce a separate column for each output item requested. Available options are

- **outputFile** = *String*

  Name of the output file. If this option is not given no output is produced.

- '[' *String* <|> *run-time statement block* { , *String* <|> *run-time statment block* } ']'

  List of variables to be output. Users can either provide the name of a variable or molecule, or any *run-time statement block*. In addition, the simulation time and iteration number can be output via the special keywords TIMES and ITERATIONS. The order in which items are punched to the output file is the same as the one in which they are listed.

Note: Data is produced only every *outputFreq* iterations as defined in the *Parameter Definition Block*.

Example:

```
output
  outputFile = "someFile.dat"
  [ TIMES, A, B, {A*B/10}, ITERATIONS]
end
```

Input Samples

Both the binary and source distributions for simgi contain a *Models/* directory with several example SGL input files for a variety of systems.

Contact and Bugs

Please report bugs to Markus Dittrich <haskelladdict.at.users.sourceforge.net>. The author would also like to encourage users to email comments, suggestions, and questions.

Copyright and License

References

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