SBML Semantic Test Suite
User Guide and Reference

Andrew Finney
afinney@caltech.edu

Last update: 29 June 2007

Contents

1 Introduction

This document describes how to use the SBML semantic test suite.

1.1 Aims and Scope of the Semantic Test Suite

The semantic test suite provides a set of valid SBML models each with a simulated time course data. This test suite is designed to be used by developers to check that their simulators produce results that are consistent with the SBML standard and thus with each other. Thus the overall aim of the semantic test suite is to facilitate the consistent interpretation of SBML models.

1.2 Semantic Test suite Documentation and Download

This document does not provide details of specific models in the test suite. This is provided by the suite's pages on the SBML wiki, see `http://sbml.org/wiki/`. However Section 2 describes in general terms how the suite is structured. The suite and scripts to automate test application are available from the SBML project CVS repository on SourceForge (`http://sourceforge.net/projects/sbml`.

2 Generic Structure of Semantic Test Suite

The test suite is divided into categories with a top level directory for each category. In addition there are two other top level directories: `docs` and `bin` containing documentation and automation scripts, respectively. Each category directory contains a subdirectory for each individual test. The only exception to this is the `basic-reactions` category which is divided into models each of which has a set of individual variant test directories.

2.1 Content of Each Test Directory

Each test directory contains the following files:

- `<model>-l2.xml` the SBML Level 2 test model
- `<model>.CSV` comma separated times series simulation results. The first line of this file doesn't contain data but identifies the columns of data. The first column is time. The subsequent columns are all species concentrations. Described in more detail in Section 3.2.2.
- `<model>.html` documentation on the model.
- `<model>.GIF` linear scale plot of species concentrations against linear time scale.
- `<model>-log.GIF` log scale plot of species concentrations against linear time scale.
- `<model>-diagram.GIF` diagram of model reaction network.
- `<model>.m` Mathematica script used to generate the above files from the test model. This script assumes that Mathematica has automatically loaded the MathSBML package. (See `http://sbml.org/software/mathsbml`).
- `<model>.test` file for driving test automation. Described in more detail in Section 3.2.2.

The `<model>` file name is designed to be unique across the suite and is composed from the category and test names.

3 Automation

3.1 User Guide

3.1.1 Platform and Environment requirements

The suite automation scripts run in a UNIX style environment. On Windows this means that you need to install Cygwin and then run commands in the Cygwin bash shell.

3.1.2 Writing a Simulator Wrapper

The test suite comes with a set of scripts to enable the testing of a given simulator. Before you can start running automated tests you must write a script or program that 'wraps' your simulator. The wrapper script should be written so that one of the script runs one simulation by the wrapped simulator. Such a wrapper should take the following arguments in order:

- the path of the SBML file containing the model
- the simulation time in seconds
- the number of simulation time steps. For each time step a row of simulation output should be generated. The time interval between simulation steps is the simulation time divided by the number of steps.
- the filename to use when storing the simulation output.
- the temporary directory in which to insert any temporary files
- the remaining arguments are the names of the species whose concentration values should be placed in the simulation output.

The simulation output must be in ASCII comma separated value (CSV) format (described in more detail in Section 3.2.2. The first row can contain any header information you like and will be ignored. The second row should contain initial condition data for $t = 0$. There should then be one row output for each time step. For example if the test requires 50 time steps then the simulation output should consist of a CSV file containing 52 rows. The first column of the file should be the simulation time for the timestep. The remaining columns should be species concentrations at that time in the order given in the arguments to the wrapper. The `*.CSV` files in the test directories are examples of the format required. An example wrapper script for MathSBML is `bin/wrappers/mathSBML-wrapper.bsh`.

To test your simulator wrapper, first make sure the Test Suite's `bin` directory is on the PATH and locate a specific `*.test` file. You can then run that test on the simulator with the command `test.bsh <wrapper> <*.test file>`. No output means that the command was successful.

3.1.3 Running Through the Complete Test Suite

Once you have a wrapper you can test your simulator by running the tests in the suite. The simplest way to do this is to change directory to the top of the test suite and type `./runtests.bsh -vwrapper="<wrapper>"`. If you add the argument `-vhaltOnFailure="true"` then the test sequence will stop when the first failure is encountered.

3.1.4 Error Messages from Failed Tests

When running test scripts one of three error messages will be generated:

- `<model>.CSV and testout.csv have different numbers of lines` This indicates that the wrapper is not interpreting the number of time steps correctly and not generating the correct number of rows in the simulation output.
- `time fields out of sync on line <line number> in file <model>` This indicates that the time column of the simulation output is incorrect because for example the time is being incorrectly calculated or the file is incorrectly formatted.
- `species <n> values <x> and <y> don't match at <time> on line <line number> in file <filename>` This message indicates that one of the species concentration values generated by the wrapper isn't close enough to the original results generated from the model. This normally indicates that either
  - the wrapped simulator is not interpreting the SBML contained in the model correctly;
  - the wrapped simulator is not as accurate as it could be; or
  - the original simulator is not as accurate as it could be.

  `n` is the number of the first species that doesn't match on the given line. The species are indexed in the order given in the corresponding test file. The file given in the message is a `*.CSV.join` file whose format is decibed in Section 3.2.2.

When any of the above error messages are generated they are followed by a very short description of the test and the URL of detailed documentation on the test.

3.1.5 Running your own test sets

It is possible to run subsets of the test suite in a custom order with a custom categorization. Simply make a copy of the `bin/testlist.txt` file and edit the copy to create your test list file. The format of this file should be self evident however see Section 3.2.2 for the details. Your test list file must have at least one category and there should at least one test in each category. A category declaration should precede the tests in that category. Tests can be placed in any order. Categories don't have to correspond to the test suite directory structure.

Once you have edited your test list file you can run the test sequence declared in that file by placing the `bin` directory on the `PATH` and invoking the command `runthroutests.awk -vwrapper="<wrapper>" <test`

`list file>`. If you add the argument `-vhaltOnFailure="true"` then the test sequence will stop when the first failure is encountered.

3.2 Reference

This section describes in detail the scripts that comprise the test automation system and the file formats that they use.

3.2.1 Scripts

All the following scripts are located in the `bin` directory of the Test Suite.

compare.awk Argument: <.join file>

Runs through a join file (see Section 3.2.2 for more details on the file format), produced by the `cvs-comparator.bsh` script, and compares the columns. Outputs errors if there is significant numerical differences between the 2 sets of data.

createargs.awk Argument: <.test file>

Extracts simulation configuration data from the given test file and returns a string for use as arguments to the simulation wrapper. Outputs a string of the form: `<time> <steps> testout.csv temp <species1> <species2> ... <speciesN>`

csv-compartor.bsh Arguments: <CSV File A> <CSV File B>

Compares the content of the 2 given files. Outputs errors if either file contains different numbers of line or if there is significant numerical differences between the 2 sets of data.

findmissingtest.bsh Argument: <test list file>

Compares the content of the test list file with the test files that exist in the test suite. Must be executed from the top level test suite directory. Outputs those files that are in the test suite but are not in the given test list. This output is in a form that can be pasted into another test list file.

generateMessage.awk Argument: <.test file>

Extracts model information from the given test file and outputs an message based on it.

number.awk Argument: <.CSV file>

Outputs a space separated version of the given file and inserts an additional first column which is the number of the record.

runthroutests.awk Arguments: `-vwrapper="`<simulator wrapper>`"` `[-vhaltOnFailure="`<boolean value>`"]` <test list file>

Runs the test sequence given in the given test list file using the given simulator wrapper. If `haltOnFailure` is set to 'true' then the sequence stops on the first test that fails. Each test should have an associated `*.CSV` and `*-l2.xml` file both with the same name and located in the same directory. This script must be run from the top level directory of the test suite.

test.bsh Arguments: <simulator wapper> <.test file>

Runs the given test using the given simulator wrapper. This script expects a `*.CSV` file and a `*-l2.xml` file both in the same directory and having the same name as the test file. The wrapper is passed the path to the `*-l2.xml` file together with simulation control arguments from the `.test` file. The results of the simulation are compared with the content of the `*.CSV` file.

The following script is located in the top level directory of the test suite

runtests.bsh Arguments: `-vwrapper="`<simulator wrapper>`"` `[-vhaltOnFailure="`<boolean value>`"]`

Runs the complete test suite against the given simulator wrapper. This script must be run from the top level directory of the test suite.

3.2.2 File Formats

*.csv or *.CSV

These files contain time series simulation output data in comma separated value format (which is readable for example by Microsoft Excel). The first row should ignored (it usually contains column headings for subsequent rows). The remaining rows comprise the times series data starting with a row for $t = 0$. The first column contains the simulation time for each row. Each remaining column contains the concentration value for a given species. The order of these columns follows that given in the corresponding `*.test` file.

*.join

These files contain the 'join' (simple combination) of two `*.num` files. The join is by the first column of the `*.num` files. The result is a space separated values file in which the columns have the pattern <row number> <time> <species 1>...<species N> <row number> <time> <species 1>...<species N>. The first row can be ignored and just contains column headings.

*.num

These files are simply a space separated values version of a comma separated values file with an additional first column which is the row number.

*.test

A `*.test` file represents a single test of a SBML simulator. A `<name>.test` file always has an associated model file and 'correct' simulation results file both in the same directory with respective corresponding names `<name>-l2.xml` and `<name>.CSV`. The `*.test` file itself contains information on how to execute the simulation test together with information on the purpose of test and/or the key features of the associated model.

A `*.test` file consists of a series of records where each record is on a separate line. Each record begins with a keyword which indicates the type of the record. The rest of the record is separated from the keyword by at least one space. The keywords and the corresponding record content are:

- TIME <simulation time in seconds>
- STEPS <number of simulation steps> - excludes the first data set at $t = 0$
- SPECIES <space separated sequence of species identifiers> - corresponds to the columns of the associated CSV file and indicates the required column order of the CSV file generated by a simulator wrapper
- URL <web page name (last part of URL) which describes test> - Currently this contains the SBML wiki page name for the page containing information on this specific test. (A complete URL is not used so that information on the test suite can be easily relocated.)
- REM <text> - short summary text describing the purpose of the test and or specific content of the associated model.

A `*.test` file must contain at least one record of each type. A `*.test` file can contain any number of REM records but only one record each of the other types. The order of REM records is significant: the text in these fields is combined to form a complete text message. The order of the other record types is not significant.

test list

A test list file defines of a sequence of simulation tests which are divided into categories. A `*.test` file consists of a series of records where each record is on a separate line. Each record begins with a keyword which indicates the type of the record. The rest of the record is separated from the keyword by at least one space. The keywords and the corresponding record content are:

- TEST <test file> - Path of a `*.test` file relative to the top level directory of the test suite.

- CATEGORY <category name>

A category record should precede the test records for tests placed in that category. A test list file must have at least one category record and there should at least one test record immediately following each category record. Test records can be placed in any order. Categories don't have to correspond to the test suite directory structure.