BIOINFORMATICS APPLICATIONS NOTE Vol. 20 no. 16 2004, pages 2829–2831
doi:10.1093/bioinformatics/bth271

MathSBML: a package for manipulating SBML-based biological models

Bruce E. Shapiro[1,*], Michael Hucka[2], Andrew Finney[3] and John Doyle[2]

[1]Jet Propulsion Laboratory, California Institute of Technology, Mail Stop 126-347, 4800 Oak Grove Drive, Pasadena, CA 91109, USA, [2]Control and Dynamical Systems, California Institute of Technology, Pasadena, CA 91125, USA and [3]Science and Technology Research Center, University of Hertfordshire, Hatfield, UK

Received on December 22, 2003; revised on March 7, 2004; accepted on April 1, 2004
Advance Access publication April 15, 2004

ABSTRACT

Summary: MathSBML is a Mathematica package designed for manipulating Systems Biology Markup Language (SBML) models. It converts SBML models into Mathematica data structures and provides a platform for manipulating and evaluating these models. Once a model is read by MathSBML, it is fully compatible with standard Mathematica functions such as NDSolve (a differential-algebraic equations solver). MathSBML also provides an application programming interface for viewing, manipulating, running numerical simulations; exporting SBML models; and converting SBML models in to other formats, such as XPP, HTML and FORTRAN. By accessing the full breadth of Mathematica functionality, MathSBML is fully extensible to SBML models of any size or complexity.

Availability: Open Source (LGPL) at http://www.sbml.org and http://www.sf.net/projects/sbml.

Contact: bshapiro@caltech.edu

Supplementary information: Extensive online documentation is available at http://www.sbml.org/mathsbml.html. Additional examples are provided at http://www.sbml.org/software/mathsbml/bioinformatics-application-note

MathSBML is an open-source package for Mathematica (Wolfram, 2003) that facilitates working with systems biology markup language (SBML) models (Hucka *et al*., 2003, http://bioinformatics.oupjournals.org/cgi/content/abstract/19/4/524). It supports both the SBML Level 1 (Hucka *et al*., 2003) and SBML Level 2 (Finney and Hucka, 2003) standards as defined at http://www.sbml.org. SBML is a software-independent format for representing computational models of biological systems that is currently supported by over 50 different software tools. MathSBML allows investigators to explore SBML models using the full range of features available in Mathematica, which includes an exhaustive mathematical environment capable of supporting all SBML features including the solution of differential-algebraic equations and discontinuous events. Mathematica is one of several platforms widely used by biological modelers that is available in many academic and commercial environments [over 500 US colleges and universities have site licenses (A.de Laix, personal communication)], and MathSBML provides full model interoperability with this environment as well as a candidate reference implementation of SBML. MathSBML has a complete Applications Programming Interface (API) for model manipulation; and includes simple function points for model-based event-driven simulation, model exploration, plotting and file import and export.

The core module of MathSBML is SBMLRead; the main function of SBMLRead is to convert the model into a Mathematica rule list and produce all the differential equations derived thereof in a format suitable for further user-manipulation. Various options are available that allow the user to produce an interpretive listing of the model or immediately attempt to run a simulation and plot the results. The following example shows the use of SBMLRead:

$$\text{m = SBMLRead[}\textit{filename, options}\text{];} \qquad (1)$$

The data structure returned by SBMLRead for an SBML Level 2 model can be represented schematically as

{ SBMLODES→{*variable*′[t]==*expression*, *variable*′[t]==*expression*, …},
SBMLParameters→{*id*→*value*, *id*→*value*,…},
SBMLIC→{ *variable*[0]==*value*, *variable*[0]==*value*, …},
SBMLSpecies→{*id*, *id*,…},
SBMLAlgebraicRules→{*variable*[t]==*expression*, *variable*[t]==*expression*, …},
SBMLUnitDefinitions→{*unit*→*expression*, *unit*→*expression*,…},

*To whom correspondence should be addressed.

SBMLUnitAssociations→{*variable*→*unit*, *variable*→*unit*,…},
SBMLReactions→{*reactant*+*reactant*+…→*product*+ *product*+…, …},
SBMLFunctions→{*id*→Function[{*arguments*}, *expression*], …},
SBMLNameIDAssociations→{*name*→*id*, *name*→*id*,…},
SBMLEvents→{*id*→{"trigger" →*expression*,
'delay' →*expression*,
'events' →{*id*→*expression*, *id*→*expression*,…}},
id→…},
SBMLModelName→*name*,
SBMLNumericalSolution→*numericalSolution*
};

Here *id* and *name* are the SBML id and name fields of the corresponding SBML parameter, species, function or event; *variable* is the Mathematica name of a model variable; *reactant* and *product* or the id's of the corresponding fields in SBML reactions; and *expression* is the corresponding algebraic (for ODE's, rules, units, functions, event delays and actions) or logical expression (for event triggers). The arrow ('→' is normally represented by Unicode 62754) can be represented in Mathematica by the '->' key combination. The format is slightly different for an SBML Level 1 file. Model variable names in Mathematica are identical to the values of their SBML Level 2 id or SBML Level 1 name field, except that the reserved character '_' is replaced in Mathematica with '$ ' (or with any other character the user specifies); the reverse translation is performed during file export. No conflicts can arise from this because '$ ' is not part of the SBML character set for identifiers. Variables are stored in a model-specific scope (referenced by a unique Mathematica context) to provide maximum extensibility. If the user requests a numerical simulation in the invocation to SBMLRead, the model representation is passed on to SBMLNDSolve (see next paragraph) and the result is assigned to SBMLNumericalSolution. The MathSBML program documentation provides more detail on this.

This model data structure [e.g. m in Equation (1)] can be directly processed by standard Mathematica functions such as NDSolve. For example, if the model includes values for rate constants and initial conditions, its DAEs can be integrated in time from $t = 0$ to $t = 10$ with the command

```
s = NDSolve[Join[SBMLODES /. m, SBMLIC /. m],
        SBMLSpecies /. m, {t, 0, 10}];
```

To reduce syntactical confusion, the following wrapper for NDSolve provides the same result:

```
s = SBMLNDSolve[m,10, solveOptions];
```

where *solveOptions* is any valid option list for NDSolve. The output of SBMLNDSolve (as with NDSolve) a list of Mathematica rules for the variable concentrations as InterpolatingFunctions.

Events are handled by stopping the simulator when an event is triggered, applying the requested action, and restarting the simulation with appropriately modified initial conditions. In this case, the return value of SBMLNDSolve is a list of InterpolationSets that encapsulate the InterpolatingFunctions for each solution time interval. SBMLNDSolve interpolates backwards from the inexact stopping time produced by the StoppingTest option (of NDSolve) to determine the exact event trigger time. InterpolationSet is an extension to Mathematica provided by MathSBML to represent solutions that cross event points.

Each InterpolatingFunction returned by SBMLNDSolve can be plotted using Plot; e.g. to plot model variable x from the solution s one would type

```
Plot[x[t] /. s, {t, 0, 10}, plotOptions];
```

Here *plotOptions* is a list of valid options for Plot. Since plotting of different variables in Mathematica requires repeated calls to Plot, MathSBML provides added functionality with SBMLPlot. For example to plot variables $x_1, x_2, \ldots$ from $t = t_1$ to $t = t_2$ with the same *plotOptions*,

```
SBMLPlot[s, {x1, x2, ...}, {t1, t2}, plotOptions];
```

The argument s passed to SBMLPlot is the data structure returned by either SBMLNDSolve or NDSolve. The function is capable of plotting across events.

MathSBML also includes a complete API for *ad hoc* model creation, manipulation, plotting and export. These commands allow users to add, modify or remove single SBML elements from the model; users may also create a completely new model or start from a pre-existing SBML file. API commands are formatted textually (i.e. they do not require use of Mathematica palettes or its extended keyset) so that they can be called directly by other programming languages using J/Link (Gayley, 2003, http://www.wolfram.com/solutions/mathlink/jlink/documentation/), or by other Mathematica-based simulators such as Cellerator (Shapiro *et al*., 2003, http://bioinformatics.oupjournals.org/cgi/content/abstract/19/5/677). Models can be iteratively evaluated with SBMLNDSolve and manipulated with the API; when satisfied, users can save the modified (or created) model as a new SBML file. Models can also be exported in other formats such as XPP (Ermentrout, 2002), LSODI-compatible FORTRAN (Hindmarsh, 1980, http://www.netlib.org/alliant/ode/prog/lsodi.f) or HTML. The XPP and FORTRAN files contain all the differential/algebraic equations that are implied by model reactions and rules. HTML files contain tabular listings of all model variables, initial conditions, units, etc.

Tools for additional language and simulator compatibility are listed on the sbml.org website; e.g. on-line tools are available there for model validation, visualization and conversion from SBML Level 1 in to SBML Level 2. Users can also write their own model import, manipulation or export routines for any software package by linking with libSBML, an open-source C-library providing an SBML API and language bindings for Java, Python, MATLAB and others.

MathSBML is open source, platform-independent and freely downloadable from Sourceforge (http://www.sf.net/projects/sbml). It can run on any platform or operating system that has Mathematica version 4.1 or higher installed; differential-algebraic equations require Mathematica 5.0 or higher. MathSBML is extensively documented; available options for all functions and detailed examples are accessible in the usual way (e.g. by typing? *function-name* from within Mathematica) as well as online at http://www.sbml.org/mathsbml.html. The supplementary material includes examples illustrating MathSBML features for three well-known models: the Repressilator (Elowitz and Leibler, 2000, http://www.nature.com/cgi-taf/DynaPage.taf?file=/nature/journal/v403/n6767/abs/403335a0_fs.html), a three-stage oscillating MAP-kinase network with negative feedback (Kholodenko, 2000) and a simple mitotic oscillator with event-triggered cell division (Tyson, 1991).

Support for higher levels of SBML will be added to MathSBML as the standards become available.

ACKNOWLEDGEMENTS

Valuable suggestions during the development of MathSBML were provided by Ben Bornstein. The research described in this paper was carried out at the California Institute of Technology, and was supported by the JST/ERATO Kitano Symbiotic Systems Project and the US National Science Foundation.

REFERENCES

Elowitz,M.B. and Leibler,S. (2000) A synthetic oscillatory network of transcriptional regulators. *Nature*, **403**, 335–338.

Ermentrout,B. (2002) *Simulating, Analyzing, and Animating Dynamical Systems: A Guide to XPPAUT for Researchers and Students*. PA Society of Industrial and Applied Mathematics.

Finney,A. and Hucka,M. (2003) Systems biology markup language: level 2 and beyond. *Biochem. Soc. Trans.*, **31**, 1472–1473.

Gayley,T. (2003) J/Link User Guide.

Hindmarsh,A.C. (1980) LSODE and LSODI, two new initial value ordinary differential equation solvers. ACM SIGNUM Newsletter, 15:10-11.

Hucka,M., Finney,A., Sauro,H.M., Bolouri,H., Doyle,J.C., Kitano,H., Arkin,A.P., Bornstein,B.J., Bray,D., Cornish-Bowden,A. *et al*., (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, **19**, 513–523.

Kholodenko,B.M. (2000) Negative feedback and ultrasensitivity can bring about oscillations in the mitogen-activated protein kinase cascades. *Eur. J. Biochem.*, **267**, 1583–1588.

Shapiro,B.E. Levchenko,A., Wold,B.J., Meyerowitz,E.M. and Mjolsness,E.D. (2003) Cellerator: extending a computer algebra system to include biochemical arrows for signal transduction modeling. *Bioinformatics*, **19**, 677–678.

Tyson,J.J. (1991) Modeling the cell division cycle: cdc2 and cyclin interactions. *Proc. Natl Acad. Sci.*, USA, **88**, 7328–7332.

Wolfram,S. (2003) *The Mathematica Book*, 5th edn. Wolfram Media, Inc.