Antimony: a modular model definition language

Lucian P. Smith*, Frank T. Bergmann, Deepak Chandran and Herbert M. Sauro

Department of Bioengineering, University of Washington, Seattle, WA, USA

Received on March 23, 2009; revised on May 27, 2009; accepted on June 26, 2009

Advance Access publication July 3, 2009

Associate Editor: Limsoon Wong

ABSTRACT

Motivation: Model exchange in systems and synthetic biology has been standardized for computers with the Systems Biology Markup Language (SBML) and CellML, but specialized software is needed for the generation of models in these formats. Text-based model definition languages allow researchers to create models simply, and then export them to a common exchange format. Modular languages allow researchers to create and combine complex models more easily. We saw a use for a modular text-based language, together with a translation library to allow other programs to read the models as well.

Summary: The Antimony language provides a way for a researcher to use simple text statements to create, import, and combine biological models, allowing complex models to be built from simpler models, and provides a special syntax for the creation of modular genetic networks. The libAntimony library allows other software packages to import these models and convert them either to SBML or their own internal format.

Availability: The Antimony language specification and the libAntimony library are available under a BSD license from http://antimony.sourceforge.net/

Contact: lpsmith@u.washington.edu

1 INTRODUCTION

The Systems Biology Markup Language (SBML) (Hucka et al., 2003) and CellML (Lloyd et al., 2004) have facilitated the exchange and verification of computer models of biological systems by providing a standard format in which to codify models. Several model creation methods have been developed, including the GUI-based methods CellDesigner (Kitano et al., 2005), JDesigner (Bergmann et al., 2006) and ProMoT (Mirschel et al., 2009), text-based methods such as Jarnac (Sauro, 2000), little-b (Mallavarapu et al., 2009), PySCeS (Olivier et al., 2005), SBML-shorthand (Gillespie et al., 2006), ScrumPy (Poolman, 2006) and GEC (Pedersen and Phillips, 2009).

As researchers create larger and more complex models, it becomes more important to be able to combine those models, and test variations of the models in different contexts. In addition, modularity is one of the properties seen to emerge from biological systems (Hartwell et al., 1999).

In some contexts, modularity means a model with defined inputs and outputs, while a systems biologist may be interested in defining a subgroup of highly connected species or co-expressed proteins (Alon, 2003). ProMoT provides a graphical interface supporting modularity, both for grouping model elements and providing inputs and outputs. [ProMoT also defines a Lisp-based model definition language (MDL) with the same capabilities.] The modularity in GEC focuses on a particular problem domain with defined inputs and outputs. The modularity in little-b is programmatic, and can thus accommodate inputs and outputs as well as subgroup definitions. Antimony, like the proposed SBML Level 3 (Finney and Hucka, 2003) is primarily a tool for grouping like sets, but also provides an input/output interface.

Genetic networks are of particular interest to the synthetic biology community. Participants in the annual iGEM competition (Goodman, 2008) are encouraged to create and share genetic ‘parts’ (snippets of DNA in a particular format) which are combined to form the working systems the teams submit for consideration, but can then be recombined and rearranged by other teams for completely different systems. GEC is one of the first languages to focus on the design of genetic networks, and provides a way to specify a generic design that can be converted automatically into a particular implementation of that design. Similarly, Marchisio and Stelling’s extension to ProMoT’s MDL (Marchisio and Stelling, 2008) provides a modeling framework for modular genetic network design.

Antimony is unique in that it combines the relative simplicity of languages like Jarnac, PySCeS, and SBML-shorthand with the modularity of languages like little-b and ProMoT’s MDL, without requiring that the modeler learn a programming language like Python or Lisp. GEC does the same, though it focuses on one particular problem domain. The libAntimony library will allow Antimony-formatted files to be read by other software packages like PySCeS, allowing them to benefit from Antimony’s modularity without having to implement it directly. The Antimony/SBML converters further extend Antimony’s usefulness by allowing users to convert their modules into a ‘flattened’ form for use in their SBML-compliant software.

2 FEATURES

2.1 Simplicity

Models in Antimony are designed to be easy to create and easy to understand. Chemical reactions are written as

\[ J0: S1 + S2 \rightarrow S3; k1*S1*S2; \]

That is, the name of the reaction, lists of reactants and products separated by a ‘\(-\rightarrow\)’, and the reaction rate. Initial conditions and constants are set with an equals sign:

\[ k1 = 2.1; \]

*To whom correspondence should be addressed.
We define a 'module' simply as a model that can be incorporated, a glycolysis module named 'G' contained the species 'pyruvate', for example, the containing model would refer to the pyruvate as 'G.pyruvate'. To connect the module to the larger model in which it appears, elements of the model may be declared to be 'the same as' elements in the larger model, or elements of other submodules. For example, a 'glycolysis' module and a 'citric acid cycle' module might be connected in this way, using the 'is' keyword:

G: glycolysis();
CAC: citric_acid_cycle();
G.pyruvate is CAC.pyruvate.

The rule in the third line declares that the 'pyruvate' in the glycolysis model is the same as the 'pyruvate' in the citric_acid_cycle module. (Similar rules can be written for other shared elements such as ATP and NADH.)

Modules with defined inputs and outputs are also accommodated with a special syntax for modules, and using the same principle as using the 'is' keyword. Inputs and outputs are defined at module creation, and make it syntactically easier to connect elements inside modules to elements in the model as a whole:

G: glycolysis(glucose, pyruvate);
CAC: citric_acid_cycle(pyruvate, CO2);

Using this syntax, you no longer need to refer to sub-elements like 'G.pyruvate' and instead simply refer to it as 'pyruvate'.

2.3 Importing models

Other Antimony and SBML files may be imported and the module(s) they contain used as submodules. This provides a way to modularize and combine existing SBML models in advance of the implementation of Level 3. As an example, the following Antimony script is how one might start to combine two BioModels (Le Novère et al., 2006):

```antimony
import "BIOMD00000000172.xml"
import "BIOMD00000000171.xml"
model multiPyruvate()
  G: Pritchard2002_glycolysis(); //Prom 172
  PB: Hoefnagel2002_pyruvateBranches(); //Prom 17
  G.PYR is PB.pyruvate;
end
```

2.4 Genetic networks

Traditional models of networks containing genetic elements tend to ignore their genetic aspects, so our efforts here try to preserve that information by using a special syntax. Genetic elements are simply listed in sequence, connected with '.--'. This example shows a promoter, ribosome binding site, gene and terminator, in sequence: '~-P1--RBS1--G1--stop--'

A unique aspect Antimony brings to modeling genetic networks is that the rate law at any given element in a sequence is, by default, equal to the rate law at the element directly upstream of it. In the above example, if P1 has a defined rate law and RBS1 and G1 do not, the rate at both of those elements will be equal to the rate at P1.

Elements in genetic sequences may also modulate the rate of the element upstream by using an ellipsis ('...') in their own formula:

P1 = ... + k1*S1;
RBS1 = ...*0.8;

If nothing is found upstream of a sequence element, the upstream rate is assumed to be zero. In this case, the rate at G1 would be equal to '(0+k1*S1)*0.8'.

Genetic elements may also be duplicated in multiple strands, or within the same strand. 'RBS1', for example, might be placed upstream of multiple genes, or the same gene might be placed downstream of multiple promoters. In these more complicated cases, a distinction is made between the 'net' rate for the element (the total of all the rates where the element appears, such as the total rate of protein production for a duplicated gene) and the 'point' rate (the rate at the specific point in the genetic sequence where an element appears, regardless of whether that element appears elsewhere).

The name of a module that contains a single genetic sequence may be used as a proxy for the strand it contains. For example, the genetic 'part' F2620 (Canton et al., 2008) has as input the concentration of AHL, and as output the activity at the end of a DNA strand containing a promoter. An Antimony module defining this 'standard part' might be used as follows:

F2620: bba_f2620(AHL);
--F2620--GFP--

This syntax allows easy creation of a module containing a single strand of DNA defining a genetic network which would convert AHL concentration to GFP production.

In this way, as researchers develop standard genetic parts with known rates and relative rates, these rates may be easily modeled.

2.5 Other features

Antimony also features the ability to define: Elements as variable or constant:

```antimony
const species s1;
var species s2, s3;
J0: s1 + s2 -> s3; k1*(s1*s2/s3*s4);
```

Interactions:

```antimony
I0: s4 -> J0;
```

Compartments:

```antimony
compartments cell1, cell2 in cell, cell3 in cell;
```
A cross-platform library ‘libAntimony’ is available to parse and convert Antimony-formatted text files, written in Bison and C++ with language bindings for C, and supported on Linux, Windows, and MacOS X. It uses libSBML (Bornstein et al., 2008) to convert modules to and from the SBML format. Stand-alone programs are also provided that translate SBML models to Antimony, and translate ‘flattened’ Antimony models to SBML.

The library has additionally been incorporated into TinkerCell (http://tinkercell.com/), a GUI-based model visualization and simulation program produced by our lab. Antimony-formatted input files may be created in an editor window, then visualized and simulated.

To test Antimony’s robustness and to provide some concrete examples of working Antimony models, we translated all 211 curated models in the current BioModels database to Antimony and back to SBML again. We then simulated the original and round-tripped models using the roadRunner simulator (Bergmann and Sauro, 2006) and compared the results. All but two models had identical results or results within a reasonable range of each other. Model seven included a delayed event that did not translate to Antimony, and the large model 183 had small numerical differences for a single variable due to different species ordering that affected the conservation laws.

The libAntimony source code, documentation, precompiled binaries for Windows, the Antimony language specification, a tutorial, converters to and from SBML, and the translated BioModels are available from http://antimony.sourceforge.net/.

Events:
E1: at (time=3): s1 = 5;

Rate rules (differential equations):
u’ = a*(b*v-u);

Assignment rules:
Ptot := P1 + EP + 2*SPP.

As with other constructs, Antimony derives information when it can from these definitions. For example, if a species has a rate rule or assignment rule, it is defined as a boundary species. Similarly, elements that change as a result of an event are flagged as variable.

3 IMPLEMENTATION AND DISTRIBUTION
A cross-platform library ‘libAntimony’ is available to parse and convert Antimony-formatted text files, written in Bison and C++ with language bindings for C, and supported on Linux, Windows, and MacOS X. It uses libSBML (Bornstein et al., 2008) to convert modules to and from the SBML format. Stand-alone programs are also provided that translate SBML models to Antimony, and translate ‘flattened’ Antimony models to SBML.

The library has additionally been incorporated into TinkerCell (http://tinkercell.com/), a GUI-based model visualization and simulation program produced by our lab. Antimony-formatted input files may be created in an editor window, then visualized and simulated.

To test Antimony’s robustness and to provide some concrete examples of working Antimony models, we translated all 211 curated models in the current BioModels database to Antimony and back to SBML again. We then simulated the original and round-tripped models using the roadRunner simulator (Bergmann and Sauro, 2006) and compared the results. All but two models had identical results or results within a reasonable range of each other. Model seven included a delayed event that did not translate to Antimony, and the large model 183 had small numerical differences for a single variable due to different species ordering that affected the conservation laws.

The libAntimony source code, documentation, precompiled binaries for Windows, the Antimony language specification, a tutorial, converters to and from SBML, and the translated BioModels are available from http://antimony.sourceforge.net/.

4 FUTURE WORK
A modeling element currently under-utilized by Antimony is metadata such as units and annotation. Units are read from imported SBML models, and are used to prevent connecting elements with incompatible units, but there is no way to export this information to an Antimony file. Connecting elements could also be facilitated by annotation by making it easier to discover common elements of combined models. In the above model combining two BioModels, for example, if pyruvate was annotated in the same way in both models, they could be automatically connected, instead of having to connect them explicitly using the "G.PTR is PB.pyruvate" line. Both of these improvements are planned for a future release of Antimony, along with support for other SBML elements not currently included in Antimony such as delayed events and algebraic rules.

We also plan to develop new converters between Antimony and other languages, starting with CellML. One advantage of a direct converter between CellML and Antimony as opposed to relying on SBML to CellML converters is that the modularity present in CellML could be directly converted to Antimony modules.

ACKNOWLEDGEMENTS
We would like to thank Mark Poolman, Brett Olivier, Johann Rohwer and Janne Hofmeyr for comments on an early draft of the language specification.

Funding: National Institutes of Health (grant 1R01GM081070-01); National Science Foundation (grant 0527023-FIBRI).

Conflict of Interest: none declared.

REFERENCES