SemanticModels.jl: A Julia Package for Scientific Model Augmentation

Micah Halter[1], Sreenath Raparti[1], Kun Cao[1], Christine Herlihy[1], and James Fairbanks[1]

[1]Georgia Tech Research Institute, Atlanta, USA

ABSTRACT

`SemanticModels.jl` is a package that takes advantage of metamodeling and meta-programming to automate model augmentation and creation. We chose Julia for our project because of the powerful type system, the efficient internal abstract syntax tree (AST), and the embedded domain specific languages (DSL) that emerge as a result through the multiple dispatch mechanism for all libraries in the Julia ecosystem [1]. The dynamic type inference and method generation has significant and powerful downstream effects which enables dynamic model manipulation with efficient code generation. These DSLs allow for strong theoretical category definitions and classifications of models to define conceptually sound universal rewrite rules for any given model class.

Keywords

Modeling Frameworks, Meta-programming, Category Theory

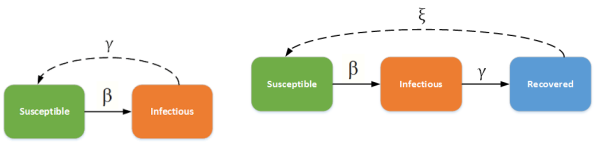
1. Methodological Approach

Introduction. `SemanticModels.jl` facilitates several metamodeling tasks by detecting and exploiting the implicit relationships between the semantically rich, natural language-based representations of scientific knowledge found in academic papers, and the relatively semantically sparse, but modular and precise representations found in code. We provide a generalized model structure to support metamodeling tasks that may be exploratory, iterative, and/or inter-disciplinary in nature. Our software can augment scientific workflows in support of a wide range of objectives, from model space exploration and hypothesis generation to model selection in complex workflows.

Language Selection. Julia breaks the mold of other common scientific computing languages such as `R`, `Python`, and `MATLAB` by emphasizing features such as multiple dispatch and meta-programming. Meta-programming is the ability to write programs that generate programs, usually, by representing code as a data structure within the program. This facilitates more generalized routines to modify, generate, and transform programs. Julia meta-programming is much easier than generating code for a static compiler, and because of the Julia just-in-time compiler, yields faster executions than a dynamic language like Python.

Model Augmentation. The susceptible-infected-susceptible (SIS) model shown in Figure 1a is one of the simplest models of infectious disease. Susceptible individuals, S, are infected by infected individuals, I, at a per-capita rate βI, and infected individuals recover at a per-capita rate γ to become susceptible again. An extension of this model is the SIR model outlined in Figure 1b, the SIR model adds a new state, R, for individuals that have recovered from their infection [2] and are no longer susceptible to new infections. The core of functionality shown in Figure 1c ingesting and manipulating previously written modeling scripts. For example, taking a code that implements an SIS model and transforming it into an SIR model using meta-programming. In the `model` function, the AST of a program is indexed for specific features that are known to be important for dynamical systems such as the states that the population can occupy, the rate relationships, and the state transition functions. SemanticModels.jl captures the meaningful structures of a code with links to how a library defines a class of problems and enables high level program transformations that scientists recognise as *perturbing the model*. In Figure 1c, when we look at the programmatic replacement for the dS term, the AST is searched for the definition of `dS`. The `add` function takes in a newly defined state with a name, transition function, and initial value and places each of those components in the appropriate location within the AST. The new model is solved by generating a new problem instance along with the necessary methods and calling the library solver functions.

Conclusion. `SemanticModels.jl` aims to leverage scientific knowledge currently trapped in the body of scientific code. While open source code is a prerequisite for reproducible research, it is not sufficient towards understanding complex modeling code. We present an automated system for extracting information from, reasoning about, and augmenting computational models. This framework enables the extension of models with new parameters and components, leveraging the dynamic nature of the Julia compiler.



(a) The flow of a SIS model. (b) The flow of a SIR model.

```
# read model into SemanticModels
sis = parsefile("epicookbook/src/SISModel.jl");
sis = model(ExpODEModel,sis);
# redefine a flux (rate) within the model
replace(sis.funcs[1],:dS,:(-β * S * I))
# define a new state and add it to our model
dR = Definition(:dR,:(γ * I),0)
sis.funcs[1] = add(sis.funcs[1],dR)
```

(c) SIS to SIR using SemanticModels.jl

Fig. 1: An example of model manipulation.

2. References

[1] J. Bezanson, A. Edelman, S. Karpinski, and V. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[2] S. Frost, A. Walsh, and J. Thompson. Epicookbook: A cookbook of epidemiological models, 2018.