|galois|

September ASKE Milestone Report for AMIDOL

Eric Davis, Alec Theriault, and Ryan Wright

Galois, Inc.
421 SW 6th Ave., Ste. 300
Portland, OR 97204

Contents

1 Milestone 8: Interim Code Release **1**

2 Milestone 9: Interim Status Report **2**

2.1 Machine-Generation and Evaluation of Expert Hypothesis 2

2.2 Formal Rules for Composing the AMIDOL IR 4

2.3 Interpretation of Model Outputs to Identify Novel Characteristics or Flaws 6

3 Milestone 10: Scripted Demonstration Prototype and Initial Performance Benchmarks **7**

3.1 Extracting a Model from Julia Source Code 7

3.2 Comparing Model Results to Experimental Data 8

3.3 Initial Performance Benchmarks 9

1. Milestone 8: Interim Code Release

Recent work on AMIDOL has benefited from collaborations with our colleagues from GTRI and the University of Arizona, leading to expanded scope and capabilities in our tool. We have released the next version of AMIDOL at its github site (`https://github.com/GaloisInc/AMIDOL/`) under the BSD 3-Clause "New" or "Revised" License.

AMIDOL has been expanded in several fundamental ways, including new support for direct model comparisons. Building towards a rich capability to compare the results of comparable measures for multiple models, AMIDOL current supports comparing measures from:

- The same model.
- The same model run with different solvers.
- Different models.
- Models and real data.

We expect to continue to expand these capabilities with a full algebra on measures to support more sophisticated comparisons and tests.

AMIDOL has also been extended to provide functionality for lifting code and groundings, implemented in Julia, into AMIDOL's VDSOL-based **abstract knowledge layer**. Julia source files are first transformed into the AMIDOL **structural knowledge layer** using our process algebra-based IR. These representations are then divided into separate elements, and AMIDOL performs **palette-** and **formulation-inference** to create a VDSOL palette, and a formulation using that palette. Palette elements use grounding information included in the Julia source to try and augment VDSOL elements with visual representations and composition rules by searching over an annotated version of the SNOMED CT ontology.

AMIDOL has also been expanded to support a richer IR. The updated AMIDOL IR supports general input predicates (conditions which must be met for an event to fire and change state, the equivalent of inhibator arcs in Petri-nets). AMIDOL has also extended its IR with support for general output predicates, allowing for side-effects of an event firing to be represented by arbitrary functions. AMIDOL has expanded its state-sharing mechanism to allow composition by general rules defined by double-push-out diagrams given in applied category theory.

AMIDOL's UI has also been extended to improve the user experience, and satisfy the needs of the new palette and formulation inference procedures. VDSOL palette elements now have a modification work-flow that can be followed through the GUI. The General UI/UX was also updated to allow for multiple client connections at a single time, separating the state of the client by user. This was done to facilitate broader demos to multiple audiences, and to better enable us to share our results with the community.

2. Milestone 9: Interim Status Report

2.1. Machine-Generation and Evaluation of Expert Hypothesis

One of the primary goals of the ASKE program is to remove the burden of working with **executable knowledge** (i.e. source code in a given language, or compiled for a targeted architecture) and **structured knowledge** (i.e. mathematics outside of a specific

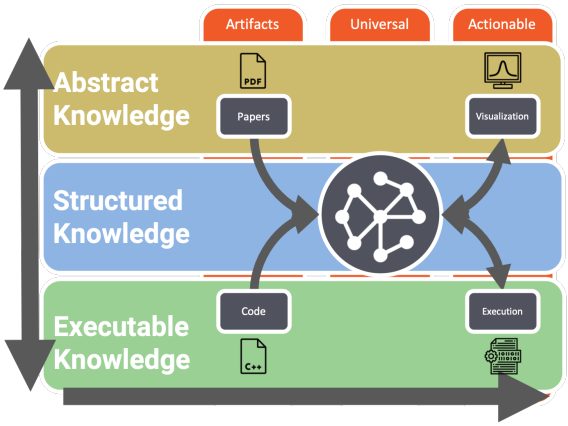


Figure 1: The ASKE Modeling Stack

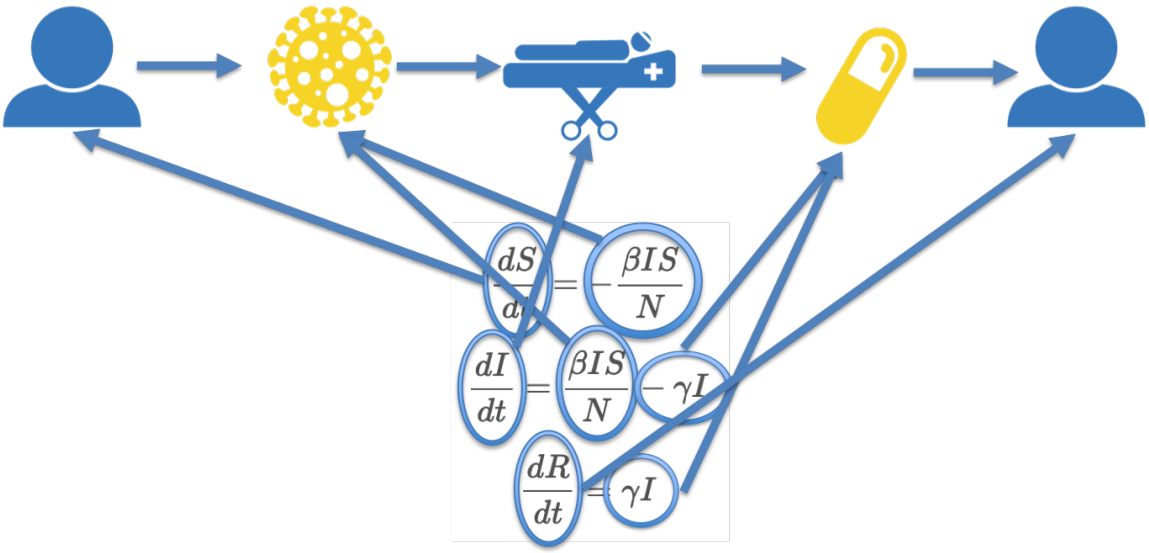


Figure 2: Mapping abstractions to Visual Domain Specific Languages (VDSOLs).

domain context) by enabling domain experts and scientists to interact with an **abstract knowledge** layer, as shown in Figure 1. Current state-of-the-art methods require domain scientists to essentially be experts in, not only domain science, but also in mathematics, statistics, software engineering, even possibly machine learning and data science, all in addition to expertise in their own domain. This is an unacceptable burden which creates unrealistic expectations on mastery of these core skills, and leads to poor implementations, bugs, errors, and long development times resulting in applications, simulations, and models with poor performance and the potential for serious errors, inaccuracies, and correctness problems.

One of the primary ways AMIDOL addresses these issues is in its use of Visual Domain Specific Ontological Languages (VDSOLs) for **model and measure formulation**. VD-

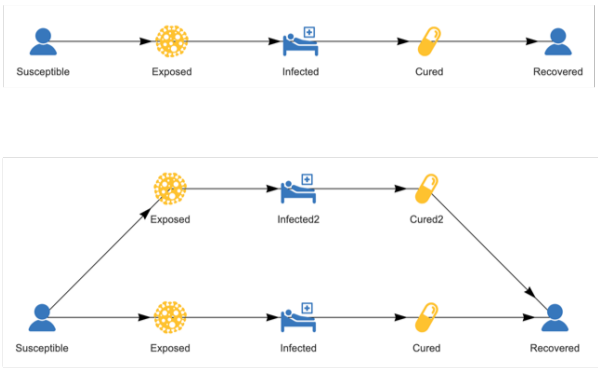


Figure 3: Example Formulations of two models in two separate languages.

SOLs help achieve these goals by lowering the barrier for entry associated with formal modeling languages. VDSOLs allow the expression of rich mathematical concepts using visual diagrams for systems and processes, but back these visual diagrams with a rich mathematical language, the AMIDOL IR, and execution semantics. This approach enables domain experts to build models of complex systems which are easier to maintain, validate, and verify, and avoid common pitfalls of monolithic and hand-coded implementations.

Figure 2 shows an example of a model in an example formulation palette for AMIDOL, and the mapping of its mathematical meaning to a system of differential equations that represents the same model. AMIDOL can automatically translate the diagram into the provided system of equations, and solve the resulting system for measures defined on the formulation.

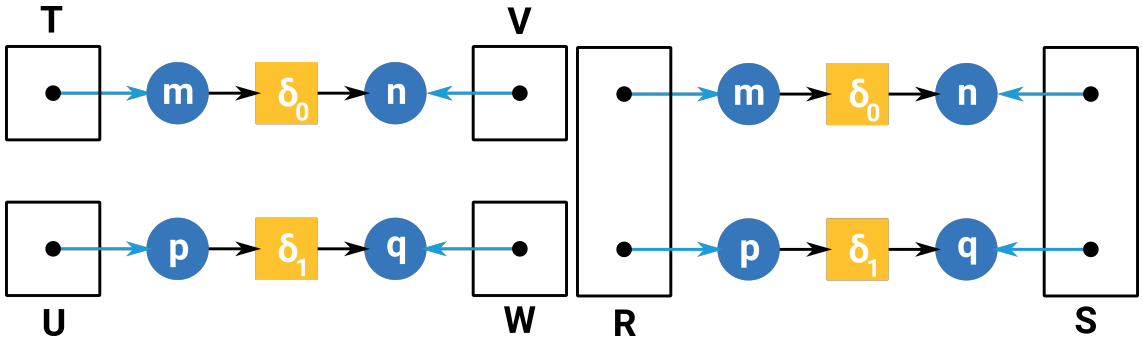
The purpose of VDSOLs is not to be prescriptive or restrictive, limiting users to a single language or representation. Figure 3, for instance, shows two separate models (SIR and SIIR model respectively on the top and the bottom) implemented in two separate VDSOLs. Those on the left using a directed graph-based VDSOL with noun and verb separations; and those on the right using a language of stochastic reactions much like `DiffEqBiological.jl`. Each pair of models represents the same underlying system and, in fact, has the same representation in the AMIDOL IR. VDSOLs allow scientists and domain experts to express models in a familiar syntax, and also to lift representations in the AMIDOL IR into a familiar syntax, automatically, even when they were originally authored in a different language.

AMIDOL's process of formulation inference even allows new palettes and languages to be constructed given extant code, and annotations of a domain-specific ontology, such as SNOMED CT.

2.2. Formal Rules for Composing the AMIDOL IR

We have extended our previously provided AMIDOL IR with formal rules for composition of **atomic models** in a VDSOL, formed of arbitrary sets of state-variables and events in the AMIDOL IR. This formal set of rules allows us not only to formalize the state-

|galois|



(a) Two AMIDOL IR representations prior to parallel composition.

(b) Two AMIDOL IR representations after parallel composition.

Figure 4: Example of parallel composition in the AMIDOL IR.

sharing process mentioned in prior reports, but also provides formal rules for composing models in formulated palettes, and checking compositions to ensure they are correct.

To do so, we utilize existing proofs from Applied Category Theory. We consider any model in the AMIDOL IR as a process algebra represented by the morphism

$$P : X \nrightarrow Y.$$

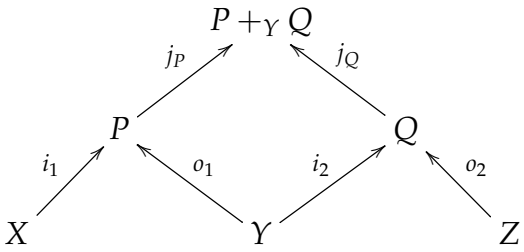
We treat this representation as an open graph. We build three rules for composing elements of the AMIDOL IR providing for **parallel** and **serial composition**, and a rule for **substitution**.

Parallel composition. Parallel composition of two models in the AMIDOL IR is accomplished by tensoring two models P and Q via disjoint union,

$$P \oplus Q : X_0 \oplus Y_1 \nrightarrow Y_0 \oplus Z_1.$$

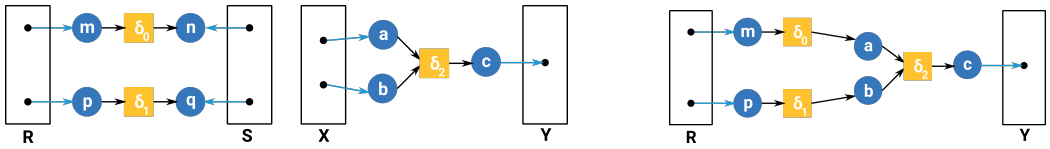
This results, as shown in Figure 4 in a new symmetric monodial category which is also a valid model in the AMIDOL IR.

Serial composition. Serial composition of two models in the AMIDOL IR is accomplished by the application of the cospan from X to Z resulting in $P \odot Q : X \nrightarrow Z$,



The results of this process are shown in Figure 5.

Substitution rules. Lastly, we provide a rule for substitution, this is usful for automatic model variation, as well as for our process of palette formulation, wherein specific models are generated from an annotated ontology to match "signatures" of models using



(a) Two AMIDOL IR representations prior to serial composition.

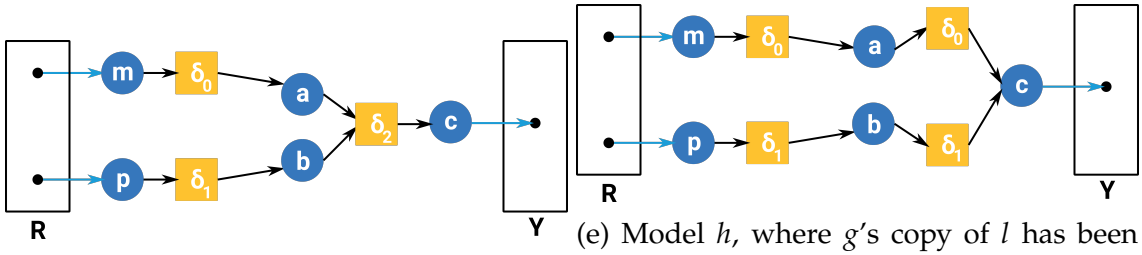
(b) Two AMIDOL IR representations after serial composition.

Figure 5: Example of serial composition in the AMIDOL IR.

(a) Original model, l.

(b) Model invariants, k.

(c) Substituted model, r.



(d) Model g, which contains a copy of l.

(e) Model h, where g's copy of l has been replaced by r.

Figure 6: AMIDOL IR's substitution rule, illustrated.

AMIDOL IR models extracted from found code. Substitutional rules utilizes double-push-out diagrams,

$$\begin{array}{ccccc} l & \longrightarrow & k & \longleftarrow & r \\ \downarrow & & \downarrow & & \downarrow \\ g & \longrightarrow & d & \longleftarrow & h \end{array}$$

Where intuitively l contains the original model, r contains the substituted model, and k contains the invariants in l and r. The model g then represents a model which contains a "copy" of l, and h is equivalent to g with its copy of l substituted with r, with d describing the invariants.

Figure 6 shows this process with an example model.

2.3. Interpretation of Model Outputs to Identify Novel Characteristics or Flaws

Substitution provides us with a way to reason about the compatibility, or incompatibility, of two models defined in the AMIDOL IR, giving us better ability to help diagnose potential errors or problems in model formulation, or to find possible model variations of interest using similar proofs from Applied Category Theory. For instance, provided we have $P : X \nrightarrow Y$ and $Q : X \nrightarrow Y$,

|galois|

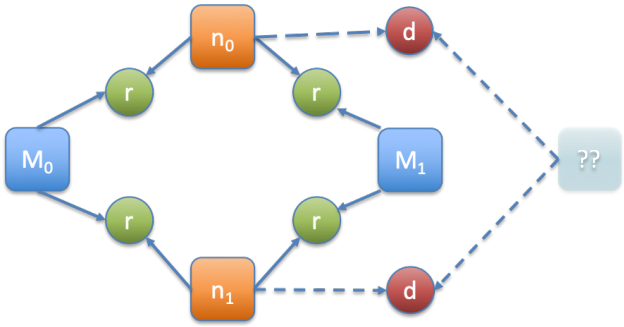
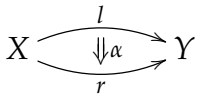


Figure 7: Representing Model Comparison in the ASKE Modeling Stack with the Results Graph



allows us to reason about the compatibility of P and Q, namely that two models in the AMIDOL IR are compatible if a substitution morphism $\alpha : P \nrightarrow Q$ can be constructed that follows our substitution rules, and maintains as invariants X and Y.

We are working to further extend AMIDOL to allow for easy model comparison through the implementation of capabilities to perform algebra on the results of model execution, using a database to store the results of model execution and measure evaluation, represented in Figure 7.

AMIDOL's graph database will consist of a set of nodes, $M = \{M_0, M_1, \ldots\}$ representing models, a set of measures $N = \{n_0, n_1, \ldots\}$ representing measures on models, and a set of results $R = \{r_0, r_1, \ldots\}$ representing the results obtained by evaluating a model with a given measure. Edges that form a path $M_i \rightarrow r_j \rightarrow n_k$ indicate that r_j is the result of evaluating M_i with measure n_k. Model comparison is then simply identifying multiple models who all have a path to a given measure of length two. Real data is represented by creating an unknown model node, representing the "model" of the real world, and with data as the result r_j for a given measure n_k.

3. Milestone 10: Scripted Demonstration Prototype and Initial Performance Benchmarks

Included below is both our demonstration script, and a link to a demon video which walks through this example. Please refer to this video if any step is confusing.

3.1. Extracting a Model from Julia Source Code

1. Load the AMIDOL UI (`http://amidol.galois.com/`) in a web browser (we recommend Google Chrome, it is not well tested in other browsers).

2. Click the "LOAD JULIA CODE" button in the top-right of the browser window.

3. Choose a file containing julia code to upload. Our demo includes a file called SIR.julia (`https://github.com/GaloisInc/AMIDOL/blob/master/examples/SIR.julia`) which is the basis for this demonstration. After the code is loaded (which will take a couple seconds), graph icons and arrows representing the Julia code will appear in the main section of the browser window.

4. Drag nodes to position them as desired.

5. Double click a node in the graph to rename it.

6. Double click a palette icon at the top of the window (not in the graph) to change its properties. E.g.:

   (a) Double click the Orange "question mark" icon to edit it.

   (b) Use the dropdown menu labeled "Icon:" to assign a new icon like "Cure".

   (c) Click the color swatch (likely colored black by default) next to this "Icon:" dropdown to choose a new color.

   (d) When finished, click the "UPDATE EXISTING ELEMENT" button at the bottom of the popup.

   (e) You should see the palette and graph have been updated to reflect your new choices.

7. Click the "EXECUTE" button in the top-right to compile the model into an executable and visualize the results.

3.2. Comparing Model Results to Experimental Data

1. Load the AMIDOL UI (`http://amidol.galois.com/`) in a web browser (we recommend Google Chrome, it is not well tested in other browsers).

2. Click the "Reset" button in the top-right of the browser window (between the undo and redo buttons to clear out previous simulation results.

3. Next, we'll load up a pre-baked model of an SIR model with two parallel infection states

   (a) Download the SIIR.json file (`https://raw.githubusercontent.com/GaloisInc/AMIDOL/master/examples/SIIR.json`).

   (b) Click on the "Load model file" button and select the file you just downloaded.

4. Click the "EXECUTE" button in the top-right to compile the model into an executable and visualize the results.

5. Now, open up the model comparison page (`http://amidol.galois.com/compare.html`). Here, we are going to compare the total infected population to some (synthetic) reference data

6. Click on "Add to comparison" and enter "Sum infected (model)"
   (a) Click on the three dots in the "Data traces" column to add a data trace whose name looks like "⟨some-date⟩_Infected_scipy_⟨n⟩". This is the first infected population.
   (b) Click on the three dots again to add a data trace whose name looks like "⟨some-date⟩_Infected2_scipy_⟨n⟩". This is the second infected population.
7. Now, we will load up some data meant to represent statistics on the total infected population.
   (a) Download the SIIR_infection_rates_demo.json data file (`https://raw.githubusercontent.com/GaloisInc/AMIDOL/master/examples/reference-data/SIIR_infection_rates_demo.json`).
   (b) In the "Upload new data trace" section, enter "reference dataset", browse and select the file you just downloaded, and click "Upload trace"
8. Click on "Add to comparison" but this time call the trace "Total infected (reference)"
   (a) Click on the three dots in the "Data traces" column and select the data trace case "reference dataset" (aka. the data you just uploaded)
9. Finally, click on "Plot the comparison" to bring up a graph that compares the total infected population that your model predicted with the one that was measured.

3.3. Initial Performance Benchmarks

In order to understand the impact of AMIDOL and the ASKE modeling stack on performance, we focus on metrics that enable us to compare the improvements to development, generalizability, and portability afforded by our novel methods, while demonstrating code that is at least as performable as the state-of-the-art, when performing program synthesis with AMIDOL to produce new **executable knowledge**.

AMIDOL provides mechanisms for automated lifting of extant source code into easily modifiable formulations in VDSOLs designed to speed the development, debugging, and other iterative processes surrounding model development. When benchmarking the time AMIDOL took to lift code to its IR, and formulate an IR, we found the following run times using models from epidemiology:

Julia Source File	Time to Extract
SIR.julia	3400ms
SIRS.julia	3800ms
SIIR.julia	4800ms

AMIDOL was able to transform formulations in its VDSOL to the AMIDOL IR, and then synthesize novel Python and Julia source code in negligable time, resulting in executables in Python and Julia that executable as fast as state-of-the-art, correct, implementations implemented with optimal algorithms for the Gillespie Method, and ODE solution. These results are promising, as they show AMIDOL provides capabilities to automatically extract, generalize, and resynthesize executable models from source code, without error, vastly simplifying the normal software development cycle for scientific code.

Future studies will need more detailed measures of normal software development times for scientific and domain experts, and detailed measures of error rates, their impact on the software development cycle, as well as user studies using AMIDOL to implement similar models.