May ASKE Milestone 7 Report for AMIDOL

Eric Davis[1], Alec Theriault[1], and Ryan Wright[1]

[1]Galois, Inc

1 Introduction

In this report we discuss recent extensions to AMIDOL's framework which seek to extend AMIDOL's Abstract Knowledge Layer and Formulation and Palette generation capabilities, and briefly discuss extensions to AMIDOL's ability to work with additional intermediate representations at the Structured Knowledge Layer, and support for additional backends for its Executable Knowledge Layer.

2 Formulations

AMIDOL's Abstract Knowledge Layer focuses on the representation, manipulation, and extension of problem formulations as part of the metamodeling process.

Definition 2.1. Formulation A formulation is a high-level domain model which represents and encodes semantic domain knowledge about a complex system. Formulations are the model-stack equivalent of a high level language and should focus on being accessible to domain scientists. While formal semantics may be implied by a formulation, the primary point of formulating a model is not writing down executable code or mathematical representations. It is writing down assumptions and knowledge which can, at a later stage, be used to infer these properties of a model.

Formulations should be represented in a form that is close to natural language, or natural representations, such as diagrams, used by domain scientists.

Formulations define the Abstract Knowledge Layer of the modeling stack.

The primary mechanism used for formulations in AMIDOL are Visual Domain Specific Ontological Languages (VDSOLs). VDSOLs attempt to leverage the semi-formal diagrams domain scientists use when describing and documenting systems; pairing them with formal mathematical and executable meaning. To accomplish this, a VDSOL is defined by a Formulation Palette,

Definition 2.2. Formulation Palette A formulation palette is a set $P = \{p_0, p_1, \ldots\}$ of palette elements which can be used to construct a semi-formal diagram with underlying mathematical and executable meaning.

Palettes form the core of the definition of a VDSOL and consist of the valid symbols which can be used to instantiate concrete occurances of a given palette element.

Definition 2.3. Palette Element A palette element is a type which can be instantiated in a formulation as a concrete occurance of the element. These instance elements must have unique names, and can be connected with arcs to define structured co-spans of instance elements which can be

compiled into AMIDOL's intermediate representation, providing formal mathematical meaning for a diagram expressed in a VDSOL.

A palette element is defined by the combination of a palette template T, and a definition in AMIDOL's intermediate representation R into the tuple (T, R)

An instance element is a palette element, combined with a unique identifier N and a set of parameters and constants C, (T, R, N, C). The set of parameters and constants are used to override generic parameters and constants within a palette element, specializing it to a particular modeling case where necessary.

Palette templates are abstract palette elements which have recently been added to AMIDOL in order to support palette extenstion, both automatically and manually.

Definition 2.4. Palette Template A palette template is a category defined by an input set, output set, and measure set, over an unknown or undefined AMIDOL IR. Palette templates are essentially container categories for compatible AMIDOL IR models whose input and output cardinality and typing match, making them interchangable.

Palette templates cannot be instantiated as a concrete instance element, but they can be subclassed by a palette element. Much like abstract classes or methods in object oriented programming, palette templates lack *implementation*, in this case a paired IR representation.

A Formulation in AMIDOL is a set of palette element instances composed together by a set of relations, $A \rightarrow B$ such that for the output cardinality of A and the input cardinality of B are the same, and have compatible types. Type relations allow restrictions on model composition which are richer than input/output cardinality; such as that used by all current AMIDOL palettes which states if A is a noun, B must be a verb, and if A is a verb, B must be a noun. Further restrictions could be used to apply types to individual inputs or outputs to restrict their composability to.

2.1 Formulation Paradigm

The process of model formulation in AMIDOL seeks to allow domain experts to construct metamodels in a novel way, using VDSOLs. These VDSOLs utilize an underlying intermediate abstract representation to give formal meaning to the intuitive process diagrams scientists and domain experts normally create. The intention is to remove the burden of having to code models explicitly, and enable 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.

While to date these formulations have all been through the use of diagrams, we are extending AMIDOL to support other types of formal languages that are natural for certain domains.

2.2 Formulation Languages

To explore other types of visual languages and diagrams, we are extending AMIDOL to support systems of stochastic chemical equations via standard notations for networks of chemical reactions of the form

$$A + B \xrightarrow{\beta} C$$

which is interpreted as "a molecule of A combines with a molecule of B to form a molecule of C with propensity β". We use the following formal grammar to represent stochastic chemical equations in AMIDOL as a formulation:

```
start   : line+

line    : equation
    | comment

comment   : "#" ANY

equation  : forwardequation
    | backwardequation
    | bothequation

forwardequation : rate "," reactants FORWARD_ARROW reactants
backwardequation: rate "," reactants BACKWARD_ARROW reactants
bothequation  : rates "," reactants BOTH_ARROW reactants

rates   : "(" rates ")"
    | forwardrate "," backwardrate
    | bothrate

forwardrate : rate
backwardrate  : rate
bothrate  : rate

rate    : "(" rate ")"
    | FLOAT
    | INT

reactants : reactants "+" reactant
    | reactant

reactant  : INT name
    | name

name    : POLAR
    | NONPOLAR

FORWARD_ARROW : /-->|==>|->|=>|>/
BACKWARD_ARROW  : /<--|<==|<-|<=|</
BOTH_ARROW  : /<-->|<==>|<->|<=>|<>/
FLOAT   : /\d+\.\d+/
INT   : /\d+/
NONPOLAR  : /[a-zA-Z_][a-zA-Z0-9_]*/
POLAR   : /[a-zA-Z_][a-zA-Z0-9_]*[+-]?/
WHITESPACE  : /[ \t\n\r\f\v]+/
ANY   : /[\d\w \t\r\f\v]+/

%ignore WHITESPACE
```

Listing 1: Stochastic Chemical Equation Grammar

AMIDOL is capable of reading networks of chemical reactions, like the following:

```
1.0,      A       -> B
2.0,      A + B <--> C + D
3.0, 4.0, A + B <--> C + D
```

Listing 2: Example Stochastic Chemical Equation

and generating an abstract syntax tree like that shown in Figure 1. This abstract syntax tree can

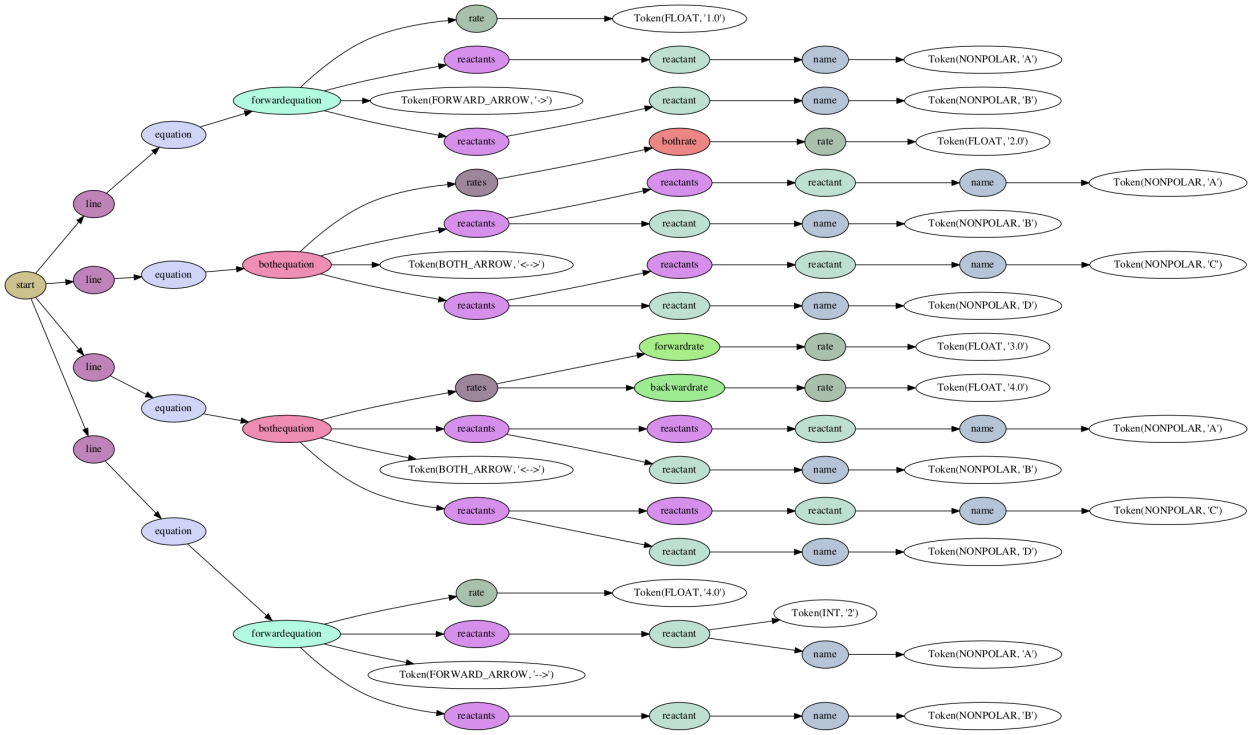

Figure 1: Example Parse Tree from Stochastic Chemical Grammar

then be used to generate a model in the AMIDOL IR, representing each reactant with a state variable, and constructing events from the identified equation nodes.

While designed to test the expressive capability of AMIDOL's Abstract Knowledge Layer in representing chemical equations, it can also be used to represent other models, like the SIR model, as follows:

```
beta / (S + I + R), S + I --> I
gamma,              I --> R
```

Listing 3: Example Stochastic Chemical Equation

3 Formulations from Models

In order to support a joint demo with GTRI and the University of Arizona, and the Knowledge Extraction workflow of moving up the stack, we have also been developing methods which allow us to automatically generate formulations in a visual language from models at the Structured Knowledge Layer. To support this, we have constructed a new module for AMIDOL which is able to translate Julia abstract syntax trees, like that shown in Figure 2, into AMIDOL's IR. These models are combined with grounding information to perform the novel process of Formulation Inference in AMIDOL.

3.1 Formulation Inference

Formulation Inference is a formal procedure in AMIDOL for automatically generating new palette elements from existing palette templates, using an ontology of domain specific terms.

```
(:function, (:call, :main, :β, :γ, :μ), (:block,
    (:macrocall, Symbol("@grounding"), nothing, (:block,
        (:call, :(=&gt;), :S, (:call, :Noun, :Susceptible, (:kw, :ontology, :Snowmed))),
        (:call, :(=&gt;), :I, (:call, :Noun, :Infectious, (:kw, :ontology, :ICD9))),
        (:call, :(=&gt;), :R, (:call, :Noun, :Recovered, (:kw, :ontology, :ICD9))),
        (:call, :(=&gt;), :λ₁, (:call, :Verb, :H3N2 infection)),
        (:call, :(=&gt;), :λ₂, (:call, :Verb, :H3N2 recovery))
      )),
    (:macrocall, Symbol("@reaction"), nothing, (:tuple, (:block,
          (:tuple, :λ₁, (:call, :+, :S, (:-&gt;, :I, (:block,
                  (:call, :*, 2, :I)
                ))))),
          (:tuple, :λ₂, (:-&gt;, :I, (:block,
                :R
              )))
        ), :λ₁, :λ₂)),
    (:(=), :Δ, (:vect, (:-&gt;, (:tuple, :S, :I), (:block,
            (:tuple, (:call, :-, :S, 1), (:call, :+, :I, 1))
          )), (:-&gt;, (:tuple, :I, :R), (:block,
            (:tuple, (:call, :-, :I, 1), (:call, :+, :R, 1))
          )))),
    (:(=), :ϕ, (:vect, (:-&gt;, (:tuple, :S, :I), (:block,
            (:&amp;amp;&amp;amp;, (:call, :&gt;, :S, 0), (:call, :&gt;, :I, 0))
          )), (:-&gt;, :I, (:block,
            (:call, :&gt;, :I, 0)
          )))),
    (:(=), :Λ, (:vect, (:(=), (:call, :λ₁, :S, :I), (:block,
            (:call, :/, (:call, :*, :β, :S, :I), (:call, :+, :S, :I, :R))
          )), (:(=), (:call, :λ₂, :I), (:block,
            (:call, :*, :γ, :I)
          )))),
    (:(=), :m, (:call, (:., :Petri, (:quote, #QuoteNode
            :Model
          )), :g, :Δ, :ϕ, :Λ)),
    (:(=), :d, (:call, :convert, :ODEProblem, :m)),
    (:(=), :soln, (:call, :solve, :m)),
    (:(=), :soln, (:call, :solve, :d))
  ))
)
```

Figure 2: Julia Abstract Syntact Tree for SIR Model

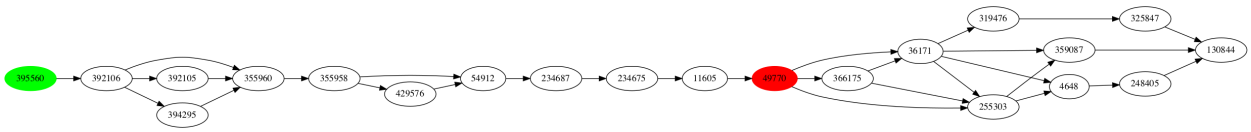

Figure 3: Ontology search for viral template from grounding “Influenza A virus subtype H3N2v”

Figure 3 shows a subset of the SNOMED CT ontology, representing elements of the ontology with nodes labeled with their SCTID. In this example, the green node represents the term "Influenza A virus subtype H3N2v" while the red node represents the term "Virus (organism)". Formulation inference assumes the existance of an annotated ontology, with nodes containing palette templates as annotations. In this example the "Virus (organism)" node has been annotated with the palette template for a viral infection verb.

For this example, assume we have a Julia AST representing the infection process, which has been transformed into AMIDOL's IR, along with the attached grounding "Influenza A virus subtype H3N2v". Formulation inference executes a search on a specified ontology, in this case SNOMED CT, and walks the parent relationships from those nodes which match the grounding, searching for a palette template. If a palette template is found, we check the category of the template against an inferred category from the IR representation of our code by constructing a *model dependency graph*.

Definition 3.1. Model Dependency Graph A model dependency graph of a model M is defined as a labeled graph $G_M = (V, A, L)$ where V is the set of vertices composed of three subsets $V_S \cup V_E \cup V_\Theta$, A is a set of arcs connecting two vertices such that arcs connect elements of the subset V_S to V_E, or vice versa, or connect an element of V_Θ to $V_S \cup V_E$, and L is a set of labels applied to elements of A from the set $\{\Phi, \Lambda, \Delta, \Theta\}$. The model dependence graph contains $v_i \in V_S$ for every state variable in the model, $v_j \in V_E$ for each event in the model, and $v_k \in V_\Theta$ for each measure defined in the model.

Arcs in A are defined such that, if an event e_j has enabling conditions that depend on state s_i, an arc labeled Φ exists from $v_i \to v_j$; if an event e_j has a state dependent rate defined in terms of s_i, an arc labeled Λ exists from $v_i \to v_j$; if an event e_j has a state transition function defined in terms of s_i and whose firing results in $s_i' < s_i$, an arc labeled Δ exists from $v_i \to v_j$ and if the firing results in $s_i' > s_i$, an arc labeled Δ exists from $v_j \to v_i$. Finally if a measure Θ_k exists and is defined over s_i, an arc labeled Θ exists from $v_k \to v_i$, and if the measure is defined over e_j, an arc labeled Θ exists from $v_k \to v_j$.

The model dependence graph is used to infer the category of a model, and checked against any template found in the ontology search. On a successful matching to a template, the IR is used to infer a new palette element which is added to the VDSOL palette, and can be instantiated into new instance elements.

If a template is not found, a generic template is created from the inferred category, and flagged for user attention and resolution.

4 The SNOMED Ontologies

In order to support our goals for the demo, and AMIDOL's new formulation inference capabilities, we have implemented an ontology ingestion pipeline for AMIDOL, and uploaded the SNOMED CT ontology. The SNOMED ontology consists of 466,612 concepts organized as a directed acyclic graph, with parent/child relationships represented. Terms in the ontology have a maximum of 3,762 children, each, and 36 parents. We annotated this ontology with palette templates for the SIR model palette.

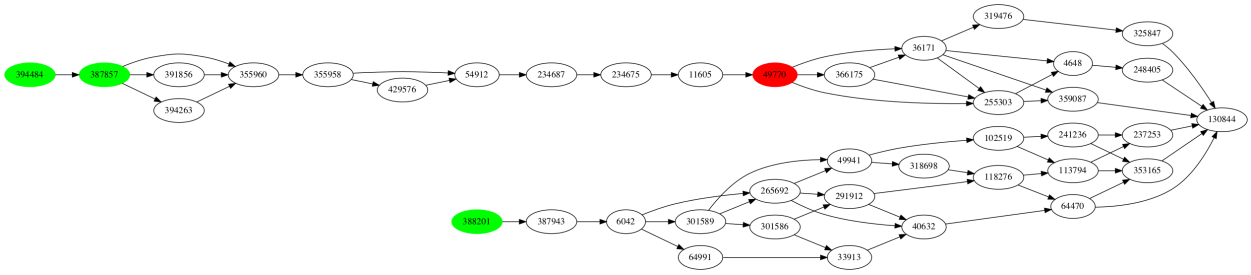

Figure 4: Ontology search for viral template from grounding “Influenza A (H1N1)”

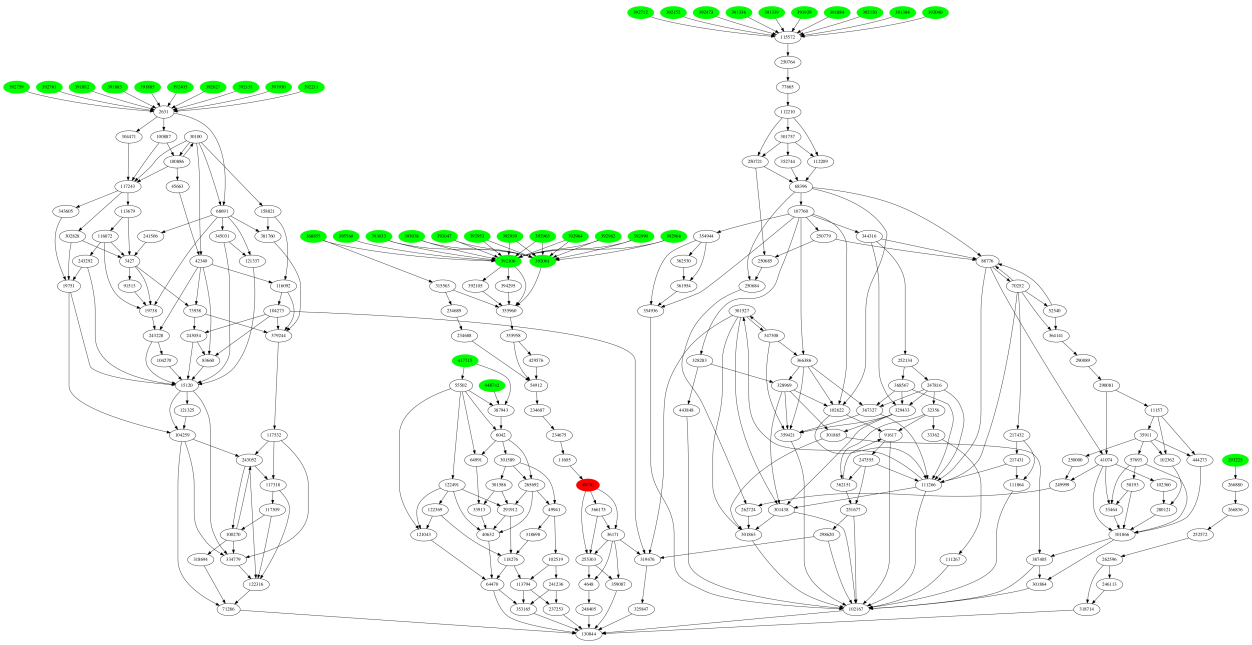

Figure 5: Ontology search for viral template from grounding “H3N2”

5 Examples

The following examples demonstrate the behavior of AMIDOL when performing palette inference from grounding. In all of the following examples, our goal was to find the palette template annotated at the node for “Virus (organism)”. Figure 4 shows the process of ontology search for a template beginning with the grounding “Influenza A (H1N1)”. Each green node represents a separate matching term, using substring matching. AMIDOL proceeded to follow all parent relationships, and displayed all annotated nodes in red, successfully recovering the palette template.

Figure 5 shows the results from searching on the grounding “H3N2”. Note the results are quite different than those presented in Figure 3, as far more nodes matched the term search, requiring a larger search of the ontology to find potential palette templates.

Figure **??** shows the same process for the grounding “Ebola”, also recovering the viral template.

Our initial experiments have been successful, and suggest the suitability for this algorithm for formulation inference using human-in-the-loop teaming with AMIDOL.

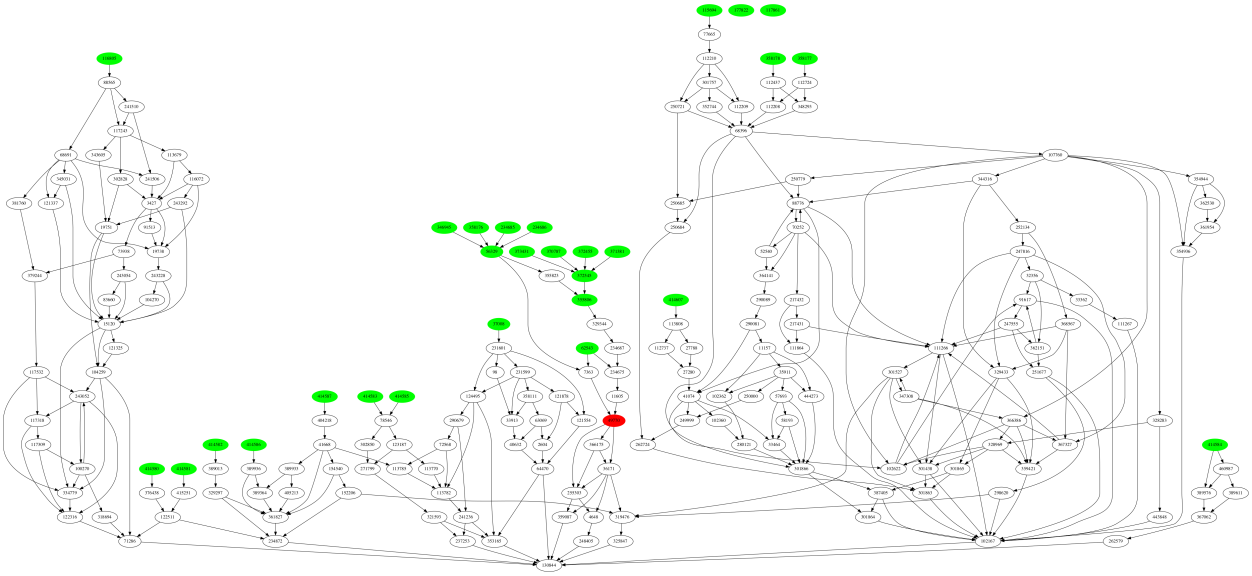

Figure 6: Ontology search for viral template from grounding “Ebola”

6 Other Recent AMIDOL Extensions

In addition to the major extensions described above, we have also extended AMIDOL to support new back-end solvers for the Executable Knowledge Layer, including agent-based models in Julia and Python, and ODE-based solvers in Julia.

7 Resources, web sites, etc.

The current AMIDOL source code, including example models and documentation, is available at the AMIDOL Github site `https://github.com/GaloisInc/AMIDOL`.