Month 6 Milestone Report

The AutoMATES Team

2019-04-31

Contents

Note: This PDF has been automatically generated from a web version, available here: `https://ml4ai.github.io/automates/documentation/deliverable_reports/m6_milestone_report` *Please visit the web version for the best experience.*

Link to the PDF version of this report. (TODO - insert link to PDF)

1 Overview

This report summarizes progress towards AutoMATES milestones at the six month mark, emphasizing changes since the m5 report.

2 CodeExplorer Webapp

The webapp has been Dockerized and is available from Dockerhub. To pull the image and run the webapp locally, do the following commands:

```
docker pull adarshp/codex
docker run -p 4000:80 adarshp/codex:latest
```

Then navigate to http://127.0.0.1:4000 in your web browser.

3 Program Analysis

3.1 Refactoring of front-end:

The focus of work in the past month was on refactoring the "Fortran code -> XML AST -> Python Code" generation process. The refactoring was done in order address issues with the Open Fortran Parser (OFP) AST XML generation that impacted aspects of `translate.py` and `pyTranslate.py` (inefficiency and inconsistencies). The new program `rectify.py` fixes these issues, relieving `translate.py` from having to handle variations in OFP-derived AST representaiton, and focus on generating the pickle file for `pyTranslate.py`. `rectify.py` reduces size of the generated AST XML by 30% to 40% comared to the original XML. For example, the original `PETASCE_markup.xml` is 4,177 lines whereas the new XML is only 2,858 lines. This also speeds up AST processing, reducing the number of special cases required.

3.2 Enhancement of GrFN Specifications and added support for new FORTRAN language features in GrFN

- Continued progress in generating GrFN specification and associated lambda files; this includes abstracting the multi-module structure of the original

FORTRAN code into a multi-file GrFN JSON data structure. This phase we developed a strategy for representing modules and this is being implemented in the GrFN spec.

- The Python-to-GrFN converter now handles multi-module files more elegantly with a separate GrFN and lambda file for each FORTRAN module.
- Began refactoring and cleanup of the code-base of the Python-to-GrFN converter. This will allow for more seamless integration of newer FORTRAN language features to the GrFN JSON and lambda file.
- Laid groundwork for the automatic support for inline comments. This will not necessarily be integrated in the GrFN file and might be provided through other means as required.
- Started integration of PyTorch with the automatically generated lambda files. This integration permits much faster code execution.
- Continued discussion and brainstorming about how arrays and multi-dimensional data structures can be represented in GrFN among other features such as `DELETE`, `SAVE` and `GOTO`.

4 Text Reading

The team has been working on the following tasks (all in progress):

1. Analyzing the relevant scientific publications with two goals in mind:
    - finding additional relations to include in the taxonomy, e.g., calculations ("E0 is calculated as the product of Kcd and ETpm.") and non-precise parameter settings ("Rns and Rnl are generally positive or zero in value."); this work will help increase extraction coverage and will be done in close collaboration with the Model Analysis team to make sure the relations included in the taxonomy are relevant to the project needs;
    - finding additional test cases to include in the test set for the existing relations (including both positive and negative examples); this work will help increase precision and recall.

2. Working on the rules used for extraction: analyzing the output of the system for false positives and adding constraints on rules and actions used

for extraction to eliminate the false positives; this work will help increase precision; writing additional rules; this work will help increase recall;

3. Implementing the functionality to extract units; this work is needed to substitute the grobid-quantities unit extractor, which has shown to be inadequate for our needs—with grobid-quantities, units are only extracted when preceded by a value; the units we need to extract are frequently compound (multi-word) and are not consistently extracted by grobid-quantities (e.g., "kg ha-1 mm-1"). To this end we are including an entity finder based on a gazetteer in the current extraction system; Building a gazetteer (a lexicon) of units based on the Guide for the Use of the International System of Units (SI) (Thompson and Taylor, 2008) to be used with the gazetteer entity finder.

In the following weeks, the team will continue to work on the current tasks. The work on having the tests pass has been put on hold temporarily to make sure the tests reflect the needs of the downstream consumers (e.g., the Model Analysis team). After a meeting with the Model Analysis team, the potential test cases will be added to the current set of tests and the team will continue the work on having the tests pass.

5 Equation Reading

The UA team began the task of equation reading using the open-source `im2markup` model (Deng, Kanervisto & Rush, 2016). However, while the pre-trained model performed well on the authors' data, when used on data from other sources, even after using the same preprocessing pipeline, the decoded equations were not usable (please see detailed results in the month 5 report). Additionally, because of the way the code was written, any model trained on a GPU (which is necessary due to the complexity of the model and the amount of training) would require a GPU for inference as well. Finally, when trying to re-train the model on our data, version conflicts with library dependencies are continually causing the model to crash.

For these reasons, and also because we ultimately want to extend the model, the UA team has reimplemented the model in PyTorch, which allows for trained models to be used on either a CPU or a GPU. Additionally, the reimplementation was intentionally done in a modular way, such that components can be replaced/extended easily. The team plans to make use of this modularity in the near future to add online data augmentation (to reduce sensitivity to exact input format) and a final layer to the equation decoder which will choose the globally optimal equation, rather than greedily making decoding decisions at each time step.

The UA team is in the process of building a Singularity container to train the model on the UA HPC. Once that is completed, the team will ensure that their reimplementation can approximately reproduce the results of the original, and will then begin working on the needed extensions to improve model performance for our use case.

6 Model Analysis

6.1 Variable Domain Constraint Propagation

During phase 1 the model analysis team identified instances of input sets to the PETASCE Evapo-transpiration model that caused bound errors during evaluation. This is due to the bound constraints of certain mathematical functions used in PETASCE, such as `arccos` and `log`. These constraints can be buried deep within the model architecture and commonly place shared constraints on more than one variable value. Some examples of constraints found in the PETASCE model are shared below.

```fortran
RHMIN = MAX(20.0, MIN(80.0, EA/EMAX*100.0))    ! EMAX must not be 0
RNL = 4.901E-9*FCD*(0.34-0.14*SQRT(EA))*TK4    ! EA must be non-negative

! The inner expression is bounded by [-1, 1] and this bound propagates to
! expression of three variables.
WS = ACOS(-1.0*TAN(XLAT*PIE/180.0)*TAN(LDELTA))
```

Automating the process of sensitivity analysis for any input set will require static analysis to determine the set of domain constraints that can be used to validate an input. Then when running a model the first step will be to validate the input set to ensure that the inputs fit within the domains. For sensitivity analysis that includes a sampling step over a set of bounds this will require validation of the runtime bounds and possibly amending the sampling process for different sensitivity methods to ensure that samples are not taken from areas out of the constrained domain. Currently the MA team is working on a method to determine the bounds for a model by doing a forward/backward pass of bounds over the model, starting with calculating the range value interval constraints of variables in a forward pass and then using this information to calculate the variable domain value interval constraints of variables during the backward pass.

6.2 Senstivity Analysis of PETPT (Priestley-Taylor) model

Model analysis currently uses the computation of Sobol indices to measure the variance in the output as a function of the fluctuations in the input parameters.

The AutoMATES MA pipeline current uses the SALib python library, which provides functionality for three different sensitivity methods, Sobol, Fourier Amplitude Sensitivity Test (FAST), and Random Balance Design-Fourier Amplitude Sensitivity Test (RBD-FAST). This phase we explored using these models for sensitivity analysis. We compared the first order indices (S$_i$; i is the variable) across the PETPT and ASCE models.

(The Priestley-Taylor (PETPT) model is an example of an evapotranspiration (EO) model where the daily EO rate of a crop depends on the following independent variables - the maximum (TMAX) and minimum (TMIN) observable temperatures within a given day, daily solar radiation (SRAD), leaf area index (XHLAI), and soil albedo coefficient (MSALB).)

Our results indicate that for reasonable approximations of the input parameter bounds, S_i values are maximum for TMAX and minimum (zero) for XHLAI in all three methods. SRAD followed by MSALB have the most significant S_i's after TMAX while the sobol index for TMIN in the PETPT model is negligible. The Sobol method can compute the second order index (S_{ij}; for the variables i and j) as well. Interestingly, we find that while S_{XHLAI} is vanishingly small, the second order index of XHLAI and TMAX is significantly large. Again, S_{ij} for $(i, j) =$ (TMAX, SRAD) is non-negligible. In addition, the runtime for each of the sensitivity analysis methods is computed and compared for different sample sizes. It becomes evident that for large sample sizes ($\sim 10^6$), Sobol is two orders of magnitude slower than both FAST and RBD-FAST methods. In summary, TMAX is the most significant parameter and any significant change in its value will have the maximum effect on the EO rate. Our future goal is to extend our discussion of sensitivity analysis to the PETASCE model in which the EO rate depends on the independent variables discussed above as well as a new set of input parameters.

7 Code Summarization

The code summarization team has begun focusing our attention on templated generation for summarizing the computations and structure of our extracted scientific models from GrFN. To meet this end we have begun investigating an architecture proposed in recent NLP literature that deals with (Wiseman, Shieber & Rush, 2018). The idea behind using a templated generation method is that the source code we wish to summarize has well-specified control-flow structures that can be easily summarized via rules. Such structures as conditional statements, loops, indexed loops, and function calls are all well-defined components of the GrFN structure that allow for the construction of a skeletal summary. The neural generation methods can then be used to create a brief description of the actual computations being carried out in the function nodes that are contained within a GrFN structure. We are currently in the process of investigating what rule-based methods would be best for summarizing whole models or portions of models. Once we have developed templates from these rule-based methods we will begin experimenting with neural generation methods to fill function node computation descriptions within the templates.