ASKE TA2 Project KApEESH M11 Report

Alfredo Gabaldon, Natarajan C. Kumar, Andrew Crapo, Vijay Kumar.
GE Research

This work is supported by the Defense Advanced Research Projects Agency (DARPA) under Agreement No. HR00111990007.

This document contains the GE TA2 KApEESH project M11 Report on the team's progress over the last two+ months. It has two sections: the first section contains detailed system installation instructions and the second section provides the update on the project.

System Installation

This section describes how to install all the system components, including the DBN execution framework which is publicly available as a docker container.

Requirements:

- Java 8
- Docker
- Graphviz

SWI Prolog

1. Download SWI-Prolog from https://www.swi-prolog.org/download/stable and install it on your machine.
2. Add the path to the bin directory to the system path (Windows only).

DBN Specification Generation Services and DBN Execution Service

1. Clone the git repository at: https://github.com/GEGlobalResearch/DARPA-ASKE-TA2
2. cd into the directory `ASKE-Service`
3. Open the file `env.sh` in an editor of your choice and suitably modify the following configuration settings:
    a. DBN_PORT: The port on the host machine at which DBN service can be accessed. Also the port at which DBN service runs within its Docker container. Default value: 5050
    b. JSON_GEN_PORT: The port on the host machine at which the input json generation service for DBN can be accessed. Also the port at which this service runs within its Docker container. Default value: 46000
    c. FS_PATH: The full path to a local directory that will contain files (e.g., Python scripts containing 'external equations') that need to be accessed by the DBN service.

Bring up the services by executing the `run.sh` script: ./run.sh (On Windows, this requires Cygwin or Git bash). This will pull the corresponding Docker images from Docker Hub, and instantiate DBN service and the json generation services within their respective Docker containers.

Eclipse Dialog Interface

1. Download and install Eclipse for Java Developers, version 2018-12 or later, available from: https://www.eclipse.org/downloads/packages/
2. Download SADL v3.2.0.20191028_DARPA_ASKE_TA1 Update from: https://github.com/crapo/sadlos2/releases/tag/Answer-0.7
   a. Download the ZIP file under this release.
   b. From the Eclipse Help menu, select "Install New Software...", then click on "Add...".
   c. Click "Archive..." on the popup, browse to and Open the downloaded ZIP file. Then click on "Add".
   d. Select "SADL 3" and follow the prompts to complete the installation process. On newer versions of Eclipse, a popup will show an option to "Update my installation to be compatible with the items being installed". Select it and click on "Next".
3. Install ANSWER Dialog v1.0.0.20191028 from: https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/releases/tag/Answer-0.7 Follow the procedure used to install SADL in Step 2.
4. From the "Window" menu ("Eclipse" on MacOS), select "Preferences".
5. On the LHS of the popup, select "Dialog" and modify the URLs of the various services as necessary. If running the services locally on your machine, the default values should work. Make sure the "Use DBN" option is checked.
6. Also in Preferences, select "SADL", under "Translation Settings", check "Use indefinite and definite articles in validation and translation" and under "Type Checking Settings" make sure the last two options are checked: "Property range specification required" and "Type checking issues as warning only".
7. The Eclipse interface requires an environment variable GraphVizPath set equal to the path to the bin directory of Graphviz. On Windows, it can be set in the standard way. On MacOS, it is necessary to open a Terminal and execute the command:

   ```
   launchctl setenv GraphVizPath /<graphviz path>/bin
   ```

   so that all apps (in particular Eclipse) can access it. This must be done before running Eclipse.

Semantic Models and Assumption Checking Module

These steps create a sample project that includes semantic models from the Hypersonics domain, the assumption checking module, and a sample dialog file (demo.dialog).

1. Open Eclipse and under the File menu select "Import…".
2. In the popup, under "General" select "Existing Projects into Workspace" and click on "Next >".

3. Check “Select archive file” and browse to the folder “Sample-Projects” in the git repo https://github.com/GEGlobalResearch/DARPA-ASKE-TA2 (cloned earlier). Select the file ASKE.zip.

Project Update

This section describes progress on the system achieved since the M9 report.

Dialog Interface

The dialog interface (used by both TA1 and TA2 teams) was improved substantially. Apart from being more stable, its basic dialog functionality is expanded to keep track of the complete state of the dialog (questions and answers) and it is capable of detecting unanswered questions anywhere on the dialog page, which triggers the computation of answers. More specifically relevant to the KApEESH system, the dialog interface now captures variable context unit information provided in queries, and this information is passed on to the backend for ingestion and processing. Thus, the queries both include more information and appear more natural. We transitioned from the earlier query form, for example,

```
what is the ^value of TotalTemperature when the ^value of Altitude is 30000?
```

to the more natural looking and complete form:

```
what is the totalTemperature of some Air when the altitude of the Air is 30000 ft?
```

which explicitly captures the context around the variables as properties of an object of type Air and captures the units of the given input.

Another extension to the dialog interface involves importing knowledge graphs that are created after running a query. These knowledge graphs contain information around the query, the models compose to compute answers, the answers returned by the DBN execution framework, the results of assumption checking and any sensitivity results that were computed. They are linked to the background scientific knowledge through the use of concepts defined there and used, for example, to specify the types of the inputs and outputs using those concepts. In order for the user to be able to explore this information through further queries, the dialog interface automatically adds an import statement to the dialog page, make knowledge graph objects, such as a model id, recognizable by the dialog interface and hyperlinked to the objects’ definitions in the OWL-serialized knowledge graph files.

Assumption Checking in Composed Models

A composite model is essentially a set of equations that together can be used to compute a desired output from the given input variables. An equation is often defined to work under certain assumptions on the inputs and other context, as discussed in detail in previous reports.

We have implemented an assumption checking module that evaluate the assumptions made by a set of equations in a model to determine if they are satisfied. The current version deals with inequality assumptions and checks if the assumptions are self-consistent and also if they are consistent with any input values given in a query. The check is performed on every model composed to compute answers to a query. The result of the assumption satisfaction check is ingested into the query knowledge graph as a property of the corresponding computational graph. If the assumptions were not satisfied, the result includes a subset of the assumptions that led to the unsatisfied condition and are accessible to the user for consideration through the dialog interface.

Since the overall approach of the project functions under the assumption that scientific knowledge has been obtained through a noisy extraction process, hence there is a possibility that some of the assumptions are incorrect, we let the system exercise the model even if the assumptions are unsatisfied. The user can then consider the results from the model execution together with the results of the assumption check and confirm if the model is indeed inapplicable as indicated by the results. Alternatively, the user may discover that the assumptions need to be adjusted and proceed to edit the equations used in the model. For models that pass the assumption check, the system performs a sensitivity analysis as well. We discuss this in a later section.

At the time of writing of the M9 report, we were still exploring different approaches to implement assumption checking. At the time we were looking at approaches based on Constraint Handling Rules and on a semantic web language called SHACL. Our implementation is based on the former, which has worked well so far, and we plan to continue developing and expanding on this approach. Specifically, we would like to be able to handle combinations of inequalities, such as “altitude > 36150” with Boolean conditions such as “pressure is constant”.

Units Handling

As described earlier, the dialogue interface was extended to capture the units provided in a query. The dialogue interface generates a set of triples form the query in an intermediate form using properties from the UnittedQuantity class in the implicit model. Equations stored in the knowledge graph also specify units for inputs and outputs. When a model is composed to answer a query, the system gathers retrieves the model information, including units required by each of the equations in the model and units specified in the query, and sends the information to the DBN specification generation service. This service checks if there is unit misalignment between any equations. That is, it checks if the composed model includes an equation that requires an input value in some units, and the input is given in the query or is output from another equation in different units. The service will insert unit conversion code into the DBN specification wherever it finds a unit misalignment. Our implementation of unit conversion relies on the python library Pint[1] and the actual execution of the unit conversion code occurs as part of model execution in the DBN framework.

There were a number of alternatives to be considered to incorporate unit handling in the system, as described in the M9 report. The ideal scenario would be to have unit conversion deeply incorporated in the DBN execution framework at the implementation level, but this is not possible without a complete reimplementation of the DBN because it relies on third-part libraries. Implementing unit conversion as part of the DBN models, i.e., by adding unit conversion nodes to the network, did not work either and turned out to introduce more overhead than the solution we adopted. The approach we settled on works but it does have a number of drawbacks. The main one being that the inline unit conversion code can substantially slow down model computation in cases where the DBN performs sampling. On the other hand, a possible advantage of the approach is that, because unit alignment occurs at the knowledge layer level, we may be able to leverage this approach for alternative execution frameworks, such as TA1's TensorFlow-based K-Chain.

Query Answers in SADL

Previous versions of the system presented results from model executions in the form of plain text. Apart from less than ideal readability, the results could not be added to the semantic model of the dialog page, which prevented the user from executing follow-up queries by referring to objects in the results, e.g., referring to a model used to compute an answer. We have implemented functionality to present query answers, computed by executing models on the DBN execution framework, in the form of SADL controlled-English. This is done by translating parts of the knowledge graph generated from processing a query into SADL and adding the resulting SADL statements as a contribution of the system to the dialog. As a contrasting example, the answer to the earlier query was previously presented in the following textual form:

```
"Model","Variable","Mean","StdDev"
"CG_1572473460215","SpeedOfSound","410.3218249130797","5.684341886080802e-14"
"CG_1572473460215","MachSpeed","3.6734036127393423","2.077921448583046"
"CG_1572473460215","TotalTemperature","-319.3631230799508","223.02112482769076"
"CG_1572473460215","StaticTemperature","-70.0","0.0"
```

In the current version of the system, the same answer is presented in SADL:

```
CM: The CGExecution with compGraph CG_1572473460215
    has output (a SpeedOfSound with ^value 410.3218249130797,
                               with stddev 5.684341886080802e-14)
    has output (a MachSpeed with ^value 3.6734036127393423,
                            with stddev 2.077921448583046)
    has output (a TotalTemperature with ^value -319.3631230799508,
                                   with stddev 223.02112482769076)
    has output (a StaticTemperature with ^value -70.0, with stddev 0.0).
```

[1] https://pint.readthedocs.io/en/0.9/

As mentioned earlier, the dialog page now also automatically inserts import statements for the generated query knowledge graph, hence the knowledge graph elements in the query answer presented in SADL are recognized by the dialog interface as part of its overall knowledge graph, letting the user refer to those elements in follow-up queries. As described in other sections of this report, follow-up queries around models composed to answer a query is the approach we have adopted to avoid cluttering the dialog page with details that users may not want to see, while allowing them to explore those details if they do.

Sensitivity Analysis

The purpose of sensitivity analysis is to obtain information about the relative sensitivity of output of a model to the inputs. In the context of this project, this is particularly relevant since query answers are computed with models composed on-the-fly rather than with pre-existing models, and therefore more likely to bring about questions from the user about the models. Sensitivity analysis would help answer questions such as “how important is this input in the model?”.

We have described our approach for sensitivity analysis in detail in the M9 report and we have completed an implementation for our system. Sensitivity analysis is performed via a request to the DBN execution framework similar but separate from an initial model execution resulting from a query. Currently we perform the analysis automatically on models that pass the assumption satisfiability check and ingest the results of the analysis into the corresponding query’s knowledge graph. To avoid cluttering the dialog interface, we don’t immediately present the results of the analysis to the user as part of the original answer to a query. Instead, the user can explore the results by issuing follow-up queries using a model’s knowledge graph instance ID, which do appear in the initial answer to a query.

The current implementation of sensitivity analysis does not take into account inputs values given in a query. It computes the sensitivity of the outputs on all inputs in the model. In the next few months we plan to explore additional sensitivity analyses that can be performed on a model assuming some of the inputs are fixed.

In the next months we also plan to replace the automatic computation of sensitivity analyses by on-demand computation triggered by a user query of a form along the lines of “what is the sensitivity of model X?”.

Models with External Equations

Previous versions of the KApEESH system worked with equations with “inline” expressions, that is, equations whose definition in the knowledge graph included the actual equation expression (code) that would be executed as part of a composed model. We have now implemented model execution with “external” equations, that is, equations defined by a signature and an external piece of code (currently, only for python) that implements the equation. Composite models then can include a combination of equations of the two types and be executed by the DBN framework. This is particularly relevant because the GE TA1 team has the capability of extracting knowledge from code in python and in java and translated to python, which moves us closer to being able to use model knowledge in the form extracted by the TA1 system.

The current implementation requires that copies of the external functions reside in a directory accessible to the DBN execution framework. We plan to remove this limitation by modifying the model specifications we sent to the DBN framework to include any python external functions used by the model within the request.

Conclusions

We have made substantial progress toward removing many of the initial simplifying assumptions. The KApEESH system now performs unit conversion within a model where necessary and checks whether the underlying assumptions of the submodels are not in conflict. We have also extended the capabilities of the system to perform sensitivity analysis on the models to provide users with explanatory insight into the models composed to answer their queries. Query answers are now presented to the users on the dialog interface in truly semantic form, using concepts that link to the background domain knowledge in the knowledge graphs, letting the user query the answers and the models for further details. Finally, DBN models have been generalized to allow the execution of “external” functions as submodels in a DBN.
There are a number of tasks in our backlog for the next months. Including:

- Hypothesizing models from a dataset was function that was working in earlier versions of the system but is currently disabled after the extensions for unit handling and assumptions. We will bring this functionality back, expanding datasets to include unit information.
- Extend the dialog grammar to include sensitivity queries to enable sensitivity analysis on demand by the user.
- Unit conversion with external functions is also not functional yet and is a little more challenging due to restrictions of the DBN execution framework. This may require generating wrapping functions around the external equations.
- Expanding the capabilities of the assumption checking module to include inequalities and Boolean constraint combinations.
- Performing assumption checking after model execution using computed values. This is most interesting when a model fails to execute but partial results are available, which in turn depends on the DBN execution framework returning such results and thus involves more risk. One possible approach that does not require modification of the DBN framework is to execute subsets of a model in case the full model fails.
- Using assumption reasoning to generate hypotheses and trigger knowledge extraction is described in the M9 report. We need to add this functionality to the assumption checking module.
- Explore sensitivity analysis of a model with some inputs fixed. This task is of a more risky/exploratory nature to be considered.
- Incorporate K-CHAIN as an alternative option for model execution.