Knowledge-driven Automated model Execution, ExplanationS, and Hypothesis generation (KApEESH)

DARPA ASKE TA2 M3 report

Alfredo Gabaldon, Natarajan C Kumar, Andrew Crapo, Vijay Kumar

GE Research

Introduction

We report on progress made as of Month 3 by the GE Research team on DARPA ASKE TA2 project KApEESH (Knowledge-driven, Automated model Execution, Explanations, and Hypothesis generation). We describe the approaches taken in the different subtasks and problems that need to be solved by the system to answer a user query by assembling and executing a scientific model. During the last two months we have used a "thin slice" example from the hypersonic aerodynamics domain to drive the implementation of a prototype system. We use that example in this report to illustrate the different tasks and approaches. Throughout the document we have included links to source code and other sources in our Git repository, as well as links to third-party open source packages we use in our implementations.

Example scenario

We use an example to illustrate the prototype's components we have implemented or in the process of implementing and how they will work in the integrated system. While the example involves a small set of equations, it is complex enough to raise various questions that need to be addressed by the system.

The example comes from the Hypersonic aerodynamics domain we are using as the application domain and involves computations of air flow variables: Total temperature and total pressure computed from altitude and speed. The diagram in Figure 1 represents how variables influence each other. Static temperature and pressure are determined by the Altitude. The Speed of sound depends on the Static temperature. Mach speed depends on the Speed of sound and the Air speed of the moving object. Total temperature is determined by the Static temperature and the Mach speed. Similarly for Total pressure.

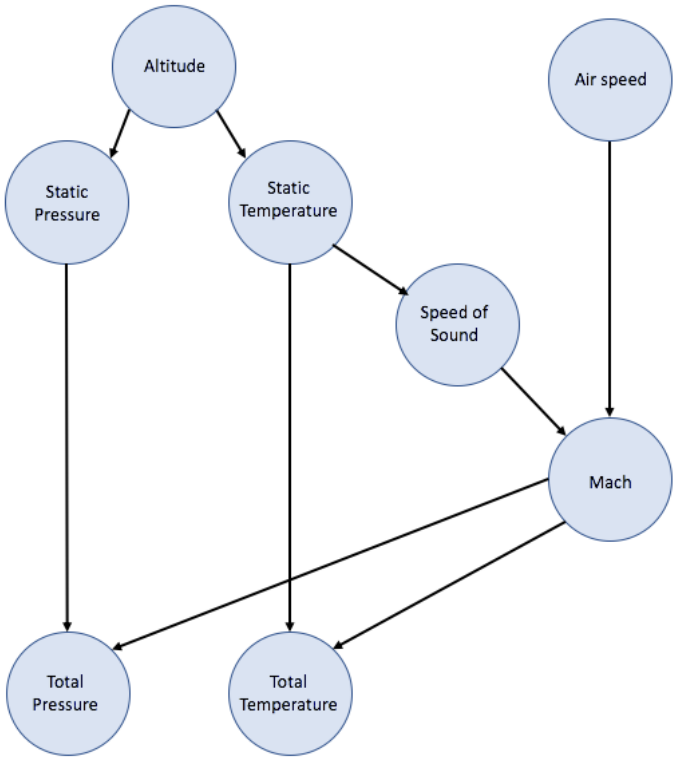

Figure 1: Air flow variable influence diagram

This example scenario is based on part of the code in the Engine simulator from the NASA Glenn Research Center's Guide to Hypersonics.

Knowledge and Network

The knowledge graph will contain knowledge extracted from code and other sources. For the example scenario, we created a domain model for the air flow variables shown in Figure 1 that includes specifications of equations for each concept in the diagram, except for Altitude and Air speed which are only used as inputs in the scenario. The model also specifies "DBNs" for each concept. A DBN concept in the semantic model simply provides additional information, such as a value range and a default distribution, about the concept variable. This additional information is useful in cases where the variable cannot be computed with the given inputs but can be treated as a random variable by the DBN Execution Framework (discussed in a later section). We will describe later a case where the Mach variable needs to be treated as a random variable in the example scenario and network.

Information such as the range of values and distribution may not be readily available for extraction from code and textual sources (the ASKE TA1 task). If such information is not available in the knowledge graph and it is needed for executing a computational graph, the system will request that additional information together with other DBN information that is typically required, such as 'number of samples', 'tracking time steps', etc., by prompting the user. Many of these parameters, however, will have default values. The system may also attempt to use parameters used for similar computational graphs previously executed.

Queries

For the example scenario, we have a number of queries that illustrate the functionality of the system. We show them here exactly as they are entered in the SADL Dialog Interface (described later).

What is the **^value** of **TotalTemperature** and the **^value** of **TotalPressure** when the **^value** of **Altitude** is 30000 and the **^value** of **AirSpeed** is 1000?

The user writes the query in SADL controlled-English, and the editor recognizes the concepts found in the knowledge graph mentioned in the query. The dialog editor uses highlighting to indicate the type of entity recognized, e.g., blue for concepts and green for properties. With this query, the system determines that it needs the equations for all the nodes shown in Figure 1.

Another query only asks for the total temperature:

What is the **^value** of **TotalTemperature** when the **^value** of **Altitude** is 30000 and the **^value** of **AirSpeed** is 1000?

In this case, the system determines that it does not need the equations for static and total pressure and constructs the necessary subgraph, which is used downstream to construct the DBN specification to be submitted to the DBN Execution Framework.

Consider another query where we again ask for the total temperature, but this time the user only provides the altitude:

What is the **^value** of **TotalTemperature** when the **^value** of **Altitude** is 30000?

In this case, the system determines that a speed input is absent and will have to be sampled. To minimize effort, the system decides to sample Mach values instead of sampling Air Speed. It will thus compute a subgraph that does not include Air Speed nor an equation for Mach, which is now an input whose values will be sampled according to the specified value range and distribution.

Components and Functionality

Here we describe the different components of the system, the tasks they perform, some design decisions, and their implementation status.

Knowledge Graph

The central repository of all information is stored as a knowledge graph, which is based on knowledge representation standards that emanated from Semantic Web research in the last few decades: OWL, RDF, and Sparql. Multiple storage systems (known as triplestores) are available both opensource and commercially. In our prototype system we use Apache Jena to access an in-memory knowledge graph. In a production system commercial alternatives such as Virtuoso (limited, opensource option also available) and Amazon's Neptune would be more suitable.

The information stored in the knowledge graph can be divided into three groups: domain knowledge, domain-independent knowledge, and meta-level knowledge. Domain knowledge includes knowledge extracted by ASKE TA1 systems from code, text, and documents. Domain-independent knowledge includes semantic models that are used to represent extracted knowledge but are independent of any particular subject matter and so can be reused across domains. Meta-level knowledge represents information about activity that occurs in the system, including queries that have been submitted by a user, scientific models that have been constructed in response to a query, results obtained from executing a model, and so on.

The domain-independent model includes definitions of entities like "UnittedQuantity," and "Equation". The semantic model for this part of the knowledge graph is hand-crafted in SADL and maintained as part of the overall system. The SADL deployment for ASKE includes it as an implicit model. A united quantity is essentially a variable in a scientific model, for example, the speed of sound, and has values and units as properties. An Equation specifies inputs (or arguments), outputs (return types), and an expression to be used for the actual computation. Knowledge extraction will sometimes result in an opaque piece of code for which we know the inputs and outputs, but not a precise equation expression. The code may also be an opaque data-driven model. For these cases, the semantic model includes the concept of an "External Equation", a type of Equation that links to a location where the code is stored and its identifier (a URI). Another concept defined here is the notion of a "Computational Graph" and a subclass for representing Dynamic Bayesian Network (DBN) information. A DBN in the semantic model links to a node (a variable), an equation or an external equation, a type (deterministic, stochastic, discrete, continuous, etc), a distribution (uniform, normal, Weibull, lognormal, exponential, beta, binomial, poisson, etc), and a range of values. Other than node and equation, these properties are treated as defaults and may be ignored, e.g., if the DBN is linked to a deterministic equation and values for all the inputs are provided, or overridden by the user before launching the execution. A DBN definition for a

root node (i.e., does not depend on other variables) would not have an equation linked to it. Otherwise the node variable corresponds to the output variable in the equation.

The domain-independent model encoded in SADL includes these statements:[1]

```
^Equation is a class
        described by input with values of type UnittedQuantity
        described by output with values of type UnittedQuantity
        described by expression with values of type string
        described by assumption with values of type Condition.

^ExternalEquation is a type of ^Equation
        described by externalURI with values of type anyURI
        described by location with values of type string.


DBN is a type of ComputationalGraph
        described by node with values of type UnittedQuantity
        described by hasEquation with a single value of type ^Equation
        described by hasModel with a single value of type ^ExternalEquation
        described by ^type with values of type NodeType
        described by distribution with a single value of type Distribution
        described by range with values of type Range .
```

Since we are working in parallel with ASKE TA1, we hand-crafted the domain-specific part of the knowledge graph with hypersonics knowledge around air flow (see Figure 1). It includes six equations/DBNs, one for each of the variables except for Altitude and Air speed, which in the example scenario are root nodes, i.e., do not depend on other variables. The Equation and DBN definitions for Static Temperature are:[2]

```
StaticTempEq is a type of ^Equation.
input of StaticTempEq has at most one value of type Altitude.
//assumption of StaticTempEq2 has value (a Condition (^value of Altitude <= 36152))
output of StaticTempEq only has values of type StaticTemperature.
expression of StaticTempEq always has value "518.6-3.56 * Altitude /1000.0".

StaticTempEq2 is a type of ^Equation.
input of StaticTempEq2 has at most one value of type Altitude.
output of StaticTempEq2 only has values of type StaticTemperature.
//assumption of StaticTempEq2 has value (a Condition (^value of Altitude > 36152) and (^value of Altitude =< 82345))
expression of StaticTempEq2 always has value "389.98".

StaticTempDBN is a type of DBN.
hasEquation of StaticTempDBN has at most one value of type {StaticTempEq or StaticTempEq2}.
range of StaticTempDBN always has value (a Range with lower -200 with upper 200).
distribution of StaticTempDBN always has value uniform.
```

Static temperature has multiple equations that apply for different values of Altitude. In general, equations will be associated with assumptions under which they are applicable and the system will need to check these assumptions to determine if a DBN using a particular equation can be chained together

[1] The current models, available from the Git repo through the link, may have already changed by the time this report is submitted.

[2] The definition of UnittedQuantity includes a property 'units' (hence the name) in addition to the 'value' property. Keeping track of units and maintaining their consistency in computations is of course a very important aspect of scientific modeling. Our system will represent and reason about units, but in this report we ignore units.

with other DBNs. Representing assumptions, which can potentially be very complex conditions, and accounting for them when constructing a computational graph, remains to be developed.

The final component of the knowledge graph is the meta-model, used for persisting queries, computational graphs, and execution events. The meta-model defines a class for "complex DBNs" (CDBN): a subtype of computational graph comprised of a set of subgraphs, each of which is another computational graph (a DBN or a KChain) but also linking to an output. Although DBNs link to an output variable, each subgraph will link to an output of the same type (same variable) but with an instantiated value and units. In other words, the CDBN will store the computed value for each node in the DBN, not only the leaf output node. These values of intermediate variables will be available for explanation and follow-up user queries. The concept CGExecution is used to represent execution events. Our preliminary definition includes start and end times, a link to the computational graph used (a CDBN), and a measurement of the accuracy of the results obtained. A class CGQuery defined in the meta-model captures queries submitted by a user or machine-generated. A CGQuery links to the inputs given in the query and each linking to the given value, an output, and an CGExecution instance. The query output should be the same as the output of the leaf subgraph in the computational graph.

The current meta-model includes definitions for the three types of object we wish to capture but is still very preliminary and subject to change. We anticipate further development down the line. We also intend to keep it general enough to accommodate computational graphs that are not based on DBNs. Another next step we plan on executing is to closer align our representation of scientific knowledge with the most recent representations used by the GE ASKE TA1 team.

SADL Dialog Interface

The user interface for the system is a controlled-English dialog editor that lets users enter queries and get answers interactively in a "chat" style dialog. In our experience, controlled-English has shown to be an effective way for humans to build and to read and understand semantic models.[3] In particular, the Semantic Application Design Language (SADL) implements a controlled-English grammar with the expressivity of OWL 1 plus qualified cardinality constraints from OWL 2, as well as rules. It also supports queries, along with tests, explanations, and other model maintenance aids. The SADL grammar and integrated development environment (IDE) is implemented using Xtext.[4] We have created, and will continue to extend, the Dialog language as an extension of the SADL language. The Dialog grammar enables the user to create a Dialog conversation in a Dialog Editor window and specify a knowledge graph to serve as the domain of discourse. Once created, the Dialog's OWL model, which extends the selected domain model with any new knowledge captured in the conversation, is usable for query and inference and, if saved by the user, is saved both as a file in the Eclipse project with the ".dialog" extension and as an OWL file that contains the domain reference and the new knowledge. New or modified content in the Dialog Editor is passed both to the UI component DialogAnswerProvider and the backend component JenaBasedDialogModelProcessor.

[3] See "Toward a Unified English-like Representation of Semantic Models, Data, and Graph Patterns for Subject Matter Experts", A Crapo and A. Moitra, International Journal of Semantic Computing, Vol. 7, No. 3 (2013), pp 215-236. Available at http://sadl.sourceforge.net/S1793351X13500025.pdf.

[4] Xtext is a framework for development of domain-specific languages.

The dialog interface is part of the SADL Eclipse frontend. This dialog interface is used by both ASKE TA1 and TA2 GE teams. For TA2, the focus is on queries that result in scientific model assembly and execution to compute the answer as well as on elaboration and follow-up queries. We have extended the grammar used by the interface to parse TA2 queries like those discussed earlier. However, the dialog interface will support general queries to the knowledge graph and make general domain knowledge accessible to the user in an easy to understand form. Through the dialog interface the user is also able to enter new concept definitions to be added to the domain knowledge. For example, if during the course of an interaction with the system the user discovers that a scientific concept she knows about is missing, the user will be able to add the definition to the knowledge graph by expressing it in SADL on the dialog interface. The system will then add this new concept definition to the knowledge graph and it will be immediately available as part of the rest of the domain knowledge. After entering the new piece of knowledge on the dialog interface, the user is able to simply resume the work she was doing, all this without the need to switch UIs or restart systems.

Next step for the dialog interface will be to continue to extend the grammar for other types of scientific model queries such as diagnostic queries. We also need to extend it to allow explanatory follow-up queries, which may need to refer implicitly or explicitly to previous queries, in addition to general queries about scientific concepts in the knowledge graph, which are already supported to some degree.

Online Model Assembly

When a user submits a query, the system accesses the knowledge graph looking for a set of equations that can be chained together to compute an answer from the given inputs. As an example, suppose the user submits the query

> What is the **^value** of **TotalTemperature** when the **^value** of **Altitude** is 30000?

Using the hypersonics domain knowledge, the system attempts to put together a set of equations that allow it to compute Total Temperature given Altitude. Let's use the network depicted in Figure 1 to illustrate how this works. The query was written in SADL and the SADL parser has recognized both the Total Temperature and the Altitude concepts in the knowledge graph. Using the equations in the knowledge graph, the system builds a graph of dependencies among variables and adds that information using a 'parent' property. For the example this graph corresponds to the graph depicted in Figure 1. The system will then find all the variables in the knowledge graph that Total Temperature depends on, shown in yellow and green in Figure 2.

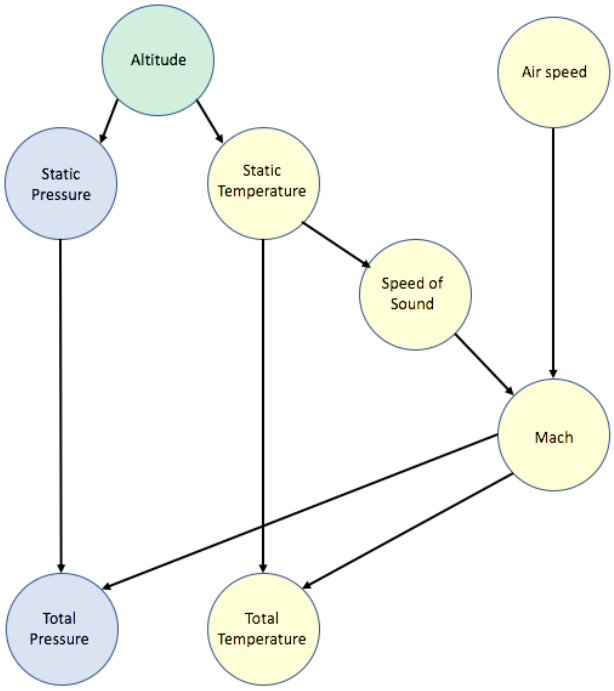

Figure 2 Ancestors of Total Temperature, inclusive.

Altitude and Air speed are root nodes, but only Altitude is a given as an input in the query. Yellow nodes are candidates for which an equation or model is needed to answer the query. In a second step, the system looks at descendants of the inputs given in the query and eliminates nodes that depend on variables that are not among the given inputs. In the example, this will remove Air speed and Mach. Total Temperature depends on Mach, so removing the Mach equation from the set means Mach will be treated as an input. Air speed, on the other hand, is just removed from the subnetwork because it is completely independent of the given inputs. In Bayesian terms, all other nodes are independent of Air speed given Mach. Since Mach is not a given input in the query, i.e., we don't have a value for it, the DBN execution framework treats it as a random variable and employs sampling to obtain values using the specified distribution for Mach. One of the advantages of the knowledge graph is that it allows very complex graph pattern-matching through the query language (Sparql). It allows us to build this computational subgraph using a single query (after the dependency graph has been inferred and added to the knowledge graph as explained earlier).

Finally, equations whose output is not required by any other equation are removed from the set. In the example this step removes the Speed of Sound equation since values of Mach will be sampled instead of computed and no other equation needs speed of sound as input. The resulting model for answering the query is shown in Figure 3.

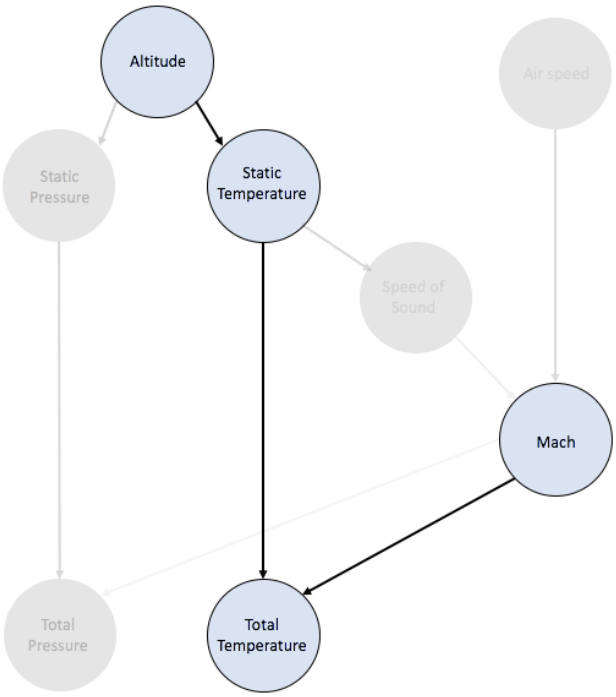

Figure 3 Final computational subgraph for Total Temperature given Altitude.

There are of course various situations here that the system will need to address. If a model is almost complete but is missing a few equations, this information should go back as feedback to ASKE TA1 systems and trigger knowledge extraction targeted at those specific variables. Similarly, the system may find a seemingly complete model for the desired output but one that doesn't use all the inputs provided in the query. Human interaction will be useful in this case as this may indicate that the user's query should be modified or alternatively that the model is incomplete. This approach will also have to be expanded for cases where an input is a data set and instead of an equation the system needs to find a data-driven model that can be applied given the characteristics of the data set.

One of the immediate next steps to execute on this part of the project is to incorporate reasoning about the assumptions under which equations are applicable. For instance, for Static temperature different equations are applicable depending on the value of Altitude. All the equations in the example scenario assume that the medium through which the object is moving is air. When constructing a scientific model by chaining together equations and other component models, the system must make sure that the equations are applicable under consistent assumptions.

Model Execution

Once the system computes a model for the desired output, it has to be translated into the format required by the DBN Execution Framework. This task is performed by the Model Execution Manager module which will retrieve from the knowledge graph the model assembled as described in the previous

section, translate it into the required json format, and launch the DBN execution through the REST service provided by the DBN Execution Framework.

This translation is carried out via a lightweight microservice layer that is currently in the works. Once the nodes and equations for the model are retrieved from the knowledge graph and assembled into SADL-specific result sets, they are first translated into some intermediate json tabular form, which then facilitates easy downstream translation into the format currently expected by the DBN execution framework. Such a set up not only provides flexibility should the current specifications of the DBN execution json format change, but also to serve multiple downstream model execution frameworks (besides DBN) in the future.

The DBN execution framework, which is based in Python, is set up be a Service in a straightforward manner using the Flask microframework that can process RESTful dispatches. A preliminary version of the Execution Service is provided in the repo here. The Model Execution Manager puts the json object together that represents the model as a DBN and sends it as a REST request to the DBN Execution Service. The DBN Execution Service takes the json as the input and evaluates the network, thereby executing the query, and returns the results back.

In the example queries shown above, the DBN execution service will return the computed total temperature and pressure when inputs are provided for the altitude and speed. When only altitude input is provided, the execution framework will use the established random distribution assigned to the Mach variable, and compute a probability distribution (or histogram in a discrete sense) of the total temperature and pressure. We have uploaded the input jsons to the DBN Execution Service in the github repository. Based on which nodes have input data associated with them, the DBN framework will automatically evaluate the overall system in the current examples of a prognostic query. The DBN framework can also effectively address other query scenarios: calibration query – the system will automatically update parameters in the model when data for outputs in the model are provided, sensitivity query – the system will automatically output the amount of variance contribution to the output from the uncertainty of each of the different inputs, and optimization query – the system will automatically run an optimization to answer queries such as what input settings or distributions will result in the maximum efficiency of an engine design or the smallest specific fuel consumption etc. We will demonstrate the different query cases during our alpha release of the platform.

The GE ASKE TA1 team is also using Flask to implement REST services for the execution module and we aim to develop our systems with enough compatibility to make it possible to switch execution frameworks with minimal effort in the future.

Query, Computational Graph, and Execution Results Ingestion

One of the processing tasks that are triggered when a user submits a query on the dialog interface is the ingestion of the query itself into the knowledge graph. This task is carried out by the JenaBasedDialogProcessor component, which is a backend component that is independent of the UI frontend components and could be used with a UI different from the Eclipse SADL dialog interface. When we persist a query, we essentially create a knowledge graph instance of the meta-model concept

CGQuery. This instance will be linked to scientific concepts used as inputs, e.g., Altitude, and the provided values or data sets. After further processing downstream, the query instance will also be linked to an instance of the meta-model concept CGExecution, that is, to the model execution event triggered by the query. For the example query used earlier, the CGQuery instance, in SADL, would be:

```
cgq1 is a CGQuery
        input (a Altitude with ^value 30000)
        output (a TotalTemperature)
        with execution (a CGExecution cge1 with compGraph cg1).
```

Note that the query instance cgq1 is linked to a CGExecution instance cge1 which in turn is linked to a computational graph instance cg1. These instances need may not be created and ingested into the knowledge graph at the same time. However, after the query-answering cycle is complete they will all be linked together.

The CGExecution concept, as we described earlier, will capture information about the execution event: start and end time, the computational graph used, and if applicable a measurement of the accuracy obtained by the model, for example, the root-mean-square error.

The computational graph would be persisted as an instance of the meta-model class CDBN, also described earlier. This object would be connected to the selected subgraphs assembled together. Each subgraph corresponds to a node in the dependency graph and therefore to a DBN and equation. The subgraph is also linked to the equation's output and value obtained. For our example, the computational graph instance, in SADL, would be:

```
cg1 is a CDBN
  subgraph sg1
  subgraph sg2
  subgraph sg3 .

sg1 is a SubGraph
        cgraph (a StaticTempDBN hasEquation (a StaticTempEq))
        output (a StaticTemperature ^value 483) .

sg2 is a SubGraph
        cgraph (a MachSpeedDBN distribution uniform)
        output (a MachSpeed ^value 1.0 ^value 1.5 ^value 2.0) .

sg3 is a SubGraph
        cgraph (a TotalTemperatureDBN hasEquation (a TotalTemperatureEq))
        output (a TotalTemperature ^value 2000 ^value 2500 ^value 3000) .
```

Each subgraph is linked to a DBN and either an equation or a distribution. Since Mach is sampled, the corresponding subgraph sg2 links to a MachSpeedDBN instance with a distribution. The distribution is persisted because the user may have overridden the default distribution. As for values, Static temperature is a deterministic node and has a single computed value. Mach speed was sampled, so it has multiple values, and so does Total temperature, which takes Mach as one of its inputs.

All this information is available in the knowledge graph after the execution of the model. The user will be able to look not only at the output value but if desired at the values of all intermediate nodes. The subgraph elements are also available for inspection. Note that the user can inspect intermediate nodes and submit follow-up questions about the scientific concepts used in the model.

Meta-reasoning

Having the queries and corresponding computational graphs and results stored in the knowledge graph allows the system to perform meta-reasoning, i.e., to reason over current and previous queries, models, and results. One way this is useful is to explain results to the user. Consider for example a query asking

What is the **^value** of **MachSpeed** when the **^value** of **Altitude** is 30000 and the **^value** of **AirSpeed** is 2000?

The user gets an answer of 2.947. Then she tries the same query with an altitude of 40000, which produces an answer of 3.028. Then she tries an altitude of 50000 and gets the same answer of 3.028. The user may wonder why an altitude of 40000 produces the same result as 50000. We want the user to be able to ask the system how the computational graphs used to answer the queries are different and find out that the computational graph for the first query is different from the computational graph for the second and third queries. The equation used to compute the Static temperature for the first query is 518.6 - 3.56 * *Altitude* / 1000, whereas for the second and third queries the Static temperature is a constant 389.98 K. The user could then use follow-up questions about hypersonics knowledge and obtain the explanation that the temperature in the lower stratosphere (between altitudes of 36152 and 82345 ft) is constant.

In an alternative scenario, the computational graph may include a node that uses a data-driven model instead of a deterministic equation. In this case, when the user requests an explanation for a surprising variation in the output, the system could launch a sensitivity analysis on the computational graph used to answer the user's queries. This would involve retrieving the computational graph and requesting the DBN execution framework to do a sensitivity analysis instead of the prognosis tasks previously done to answer the original user's queries.

We are currently working on designing the knowledge graph and semantic models to support this type of reasoning. Apart from the meta-model and ingestion of queries, computational graphs, and results, we need to extend the dialog interface grammar and implement algorithms to compare computational graphs.

Conclusion

We have concretized and fleshed out many of the components of the prototype system. We will continue developing the system in the coming months guided by our example scenario, incrementally expanding it as we implement new functionality.