Analyzing Military Intelligence Using Interactive Semantic Queries

Rod Moten
Data Fusion and Analytics
Sotera Defense Solutions
Aberdeen Proving Grounds, MD, USA
rod.moten@soteradefense.com

Abstract— **We describe a strategy for performing semantic searches for analyzing military intelligence. Our strategy allows the analyst and the query engine to work together to reduce a complex query into simpler queries. The answers for the simpler queries are combined into answers for the original query. The queries can be refined using rules defined by the analyst or analytics created by a data scientist. Our strategy uses an alternative approach to semantic modeling than the state-of-the-art approaches based on OWL. OWL is an implementation of a branch of mathematical logics designed specifically for semantic modeling called description logics. Our strategy uses a branch of mathematical logics called type theory. We use type theory because of the long history of developing systems based on type theory for reasoning interactively. We demonstrate with an example how the strategy can be used to answer questions posed by analysts that couldn't be answered using conventional methods.**

Keywords: semantic search; military intelligence; analytics; type theory; ontology; semantic modeling; interactive theorem proving

I. INTRODUCTION

"The Army is working closely with the intelligence community and other Defense Department partners, including the Navy, in developing cloud-based systems for battlefield intelligence."[1] The goal of the U.S. Army is to fulfill theater intelligence requirements using these systems as much as possible [2]. For example, suppose an analyst created a hypothesis that a family within an Afghan village is responsible for several IEDs. The analyst may use the Cloud to determine which families have connections to hostile organizations. The data may be in the Cloud that *directly links* a family to a hostile organization. For example, suppose the Sadat Baba family [2] is a member of the village and intelligence data contains the triple (Sadat Baba shares-profit Taliban). In the triple, Sadat Baba is the subject, shares-profit is the predicate and Taliban is the object. On the other hand, the intelligence data may only contain data that *indirectly links* the family to a hostile organization. These links have to be inferred either deductively or inductively from the data. For example, it may be possible to infer that Sadat Baba and the Taliban have common interests because both Sadat Baba and the Taliban attacked the Dalazak family. This could be inferred by applying the following rule. If a family and an external organization attack another family in the same tribe or village, then the external organization and the attacking family have common interests. Or it could be inferred using network analysis because Sadat Baba and Taliban both are linked to Dalazak by the same relationship within the same subgraph.

The current state-of-the-art for military intelligence analysis focuses on the analyst using visual aids and various retrieval techniques, such as faceted search and querying, to perform this inference manually [3]. Military intelligence analysis systems should focus on developing strategies for reducing the manual work performed by analyst by incorporating more automated methods. The effort for searching for data can be reduced if the query engine automated some of the work the analysts performed manually. This means the query engine would need to have analytics that automate some of the inductive and deductive reasoning performed by the analyst.

Some of the analytics used in a query or search may be different from the analytics used in ETL (extraction, transformation, and load). In ETL, analytics are used primary for entity and feature extraction, entity resolution, and entity fusion [4]. In this case, the analytics aren't used to answer a query posed by an analyst. The analytics we are interested in, such as multi-relational link predication [5] [6], occur after ETL. The analytics will be performed after the data has been mapped into a graph-like structure, such as RDF or DRIF [7], [8]. Therefore, the query engine will need the ability to express the behavior of the analytic in terms of the underlying semantic network.

The behavior of an analytic can be specified as the logical implication of the postcondition from the precondition. The precondition is a logical statement that must be satisfied in order for the analytic to produce valid output. The postcondition is a logical statement that describes the characteristics of the concepts and relationships produced by the analytic. In the simplest case, the specification of an analytic could be defined as $\phi \rightarrow \psi$, where ϕ is the precondition and ψ is the postcondition. For example, the precondition of the analytic for associating a family and an external organization would be 'for any $?f \in \text{Family}$ and $?n \in \text{National Organization}$ there exists a $?g \in \text{Family}$ such that $(?f \text{ attack } ?g)$ and $(?n \text{ attack } ?g)$'. The postcondition would be '$(?f \text{ common-interests } ?n)$'.

The examples above may mislead the reader to believe that first-order logic would be sufficient for expressing the precondition and postcondition. However, first-order logic doesn't support cases when analytics can operate on a set of concepts and roles. For example, consider an analytic that uses numeric calculations to determine relationships, such as common neighbor algorithms [9]. Such an analytic only cares about relationships or edges between nodes. So it can operate on any concept. For example, if the analytic uses a common neighbors algorithm for determining the common-interests relationship, then the precondition would be 'for any $?f \in ?F$and $?n \in ?N$ there exists a $?g \in ?F$such that $(?f\ ?A\ ?g)$ and $(?n\ ?A\ ?g)$'. In this precondition, we parameterized the concepts for the attackers, Family and National Organization, and we parameterized the attack relationship. We can also parameterize the postcondition. For example, a data scientist may be able to improve the analytic to infer a stronger relationship between the attackers using additional information about them. In this case, the postcondition would be 'there exists a $?Q \subseteq \text{common-interests}$ such that $(?f\ ?Q\ ?n)$'.

The query engine could use the specification of an analytic as a rule for solving an intelligence requirement. If the specification of an analytic is $\phi \rightarrow \psi$ and the intelligence requirement can be reduced to ϕ, then the solution to the intelligence requirement would be the concepts and relationships that the analytic produces that satisfy ψ.

In this paper, we present a strategy for analyzing intelligence data using an *interactive query language*. When a user specifies a query, the query engine solves the query by refining it into new queries. If any of the new queries cannot be answered, it asks the user to assist it. The user assists the query engine by specifying an analytic or rule that can solve the query or reduce the query into new queries. This process continues until all queries are answered or until there is a query that cannot be answered. The query engine keeps track of this process and combines the answers from the generated queries into an answer for the original query. At the heart of the query language is the type theory TT-IQ, *Type Theory for Interactive Querying*. The query engine for TT-IQ consists of a framework that allows a data scientist to define analytics that can be included in query processing and for analysts to add new rules.

This paper is outlined as follows. First, we present work related to our strategy. Then, we use an example to demonstrate interactive querying. Next we give an overview of TT-IQ. Finally, we conclude the paper with a discussion of our strategy.

II. Related Work

Our strategy is similar to approaches that use a semantic network or ontology for refining queries. These approaches, such as QUICK [10], LISQL [11], and query rewriting [12] use semantic information to enhance a query supplied by a user. These approaches use a semantic network and stepwise refinement to create semantic queries. Our approach, on the other hand, uses stepwise refinement for query execution.

Researchers at GMU have spent over 15 years developing strategies that could be used for interactive querying [13]–[17]. Their approaches use heuristics to perform inductive and deductive reasoning. They also use machine learning to find new rules to add to the knowledge base. Our strategy support inductive and deductive reasoning except we use proof-theoretic methods used in interactive theorem provers. Theorem provers, such as NuPrl [18], Coq [19], and Isabelle [20], use interactive methods for developing formal mathematical proofs. In these systems, the assertions are *type judgments*. Type judgments are logical statements that ask which objects belong to a specific type. The types can be defined to resemble logical statements, such as $P \wedge Q \rightarrow R$. We use this same technique in our query language. However, our type theory differs from the state-of-the-art in order to support semantic modeling.

K-DTT [21] and S-DTT [22] are type theories that use an *extensional* approach to semantic modeling. Description logics also use an extensional approach to semantic modeling. This means that A-Box statements or an external source, such as a database, has to be used to determine that an object belongs to a concept. In TT-IQ, concept membership is *intensional*. In other words, we determine whether an object belongs to a concept based on how the object was constructed. Therefore there is no need for A-Box statements or external sources to determine concept membership.

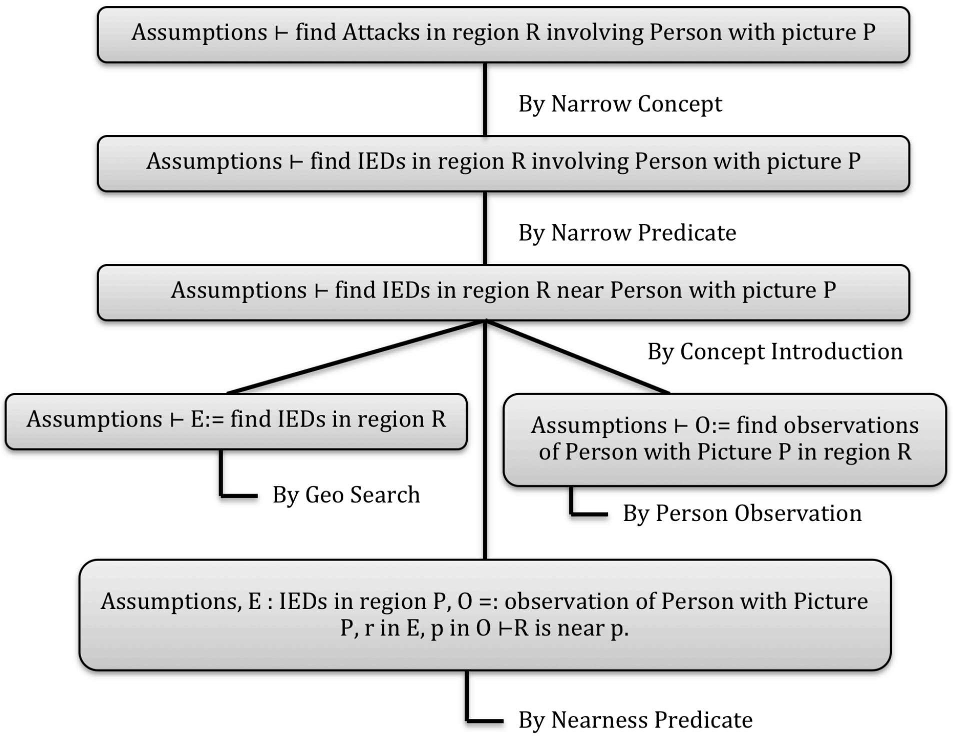
Type theory isn't the only method of using higher-order logic for semantic modeling. Classical higher-order logics have been used for semantic modeling as well [23]–[26]. However, the query languages used in these approaches do not allow a user to define analytics to be incorporated into the query engine.

Rule-based approaches for semantic search, such as Tuple-Generating Dependencies [27], have the capability to incorporate rules specified by a user into the query answering process. However, these approaches are implemented using first-order reasoning techniques from logic programming. As a result, they will not be able to support domain metamodeling. Our strategy, on the other hand, will support finding concepts and relationships that meet specific criteria.

III. Interactive Querying

In this section, we give an overview of how interactive querying is performed. Interactive querying is analogous to proving a type judgment. Informally, a typing judgment consists of a goal and a set of assumptions. The goal is the assertion we want to prove. The assumptions represent facts and statements from the knowledge base. Therefore, it consists of logical statements about the semantic meaning of concepts and relationships.

Fig. 1 shows an example proof created from an interactive query for attacks in a region that may have involved a specific person. We assume the analyst only knows that the person has a set of features observed by an interrogator, such as facial marks, height, and the number of a cell phone belonging to the detainee. This query is stated as a judgment that appears as the root of the proof tree. Here we state the judgments informally in natural language. The assumptions in the judgments in Fig. 1 contain a definition of a type representing the concepts Attack and Person; an object representing the features of the detainee;



Assumptions ⊢ find Attacks in region R involving Person with picture P
By Narrow Concept
Assumptions ⊢ find IEDs in region R involving Person with picture P
By Narrow Predicate
Assumptions ⊢ find IEDs in region R near Person with picture P
By Concept Introduction
Assumptions ⊢ E:= find IEDs in region R
By Geo Search
Assumptions ⊢ O:= find observations of Person with Picture P in region R
By Person Observation
Assumptions, E : IEDs in region P, O =: observation of Person with Picture P, r in E, p in O ⊢R is near p.
By Nearness Predicate

Fig. 1. Example proof created from an interactive query

features of the detainee; declarations of the predicate symbols, such as involving; and definitions of data types that represent cell phone numbers and regions. The assumptions also contain a taxonomy of properties and features. The taxonomy is defined as a partial ordering on predicate symbols and attribute names.

The query engine determines which *tactics* can be applied based on whether the conclusion of a tactic matches the goal of a judgment. The query engine uses the *type compatibility relationship* [28] of TT-IQ to perform matching. When two types $T$and T' are compatible, it means that an object in T can be converted to an object in T' and vice versa. Type compatibility is an extension of the subtyping relationship in TT-IQ. We give an overview of subtyping in "Overview of TT-IQ".

Each tactic has a conclusion and zero or more antecedents. When a tactic matches a judgment, the query engine creates new judgments for each antecedent. For example, the Narrow Concept tactic is a built-in tactic that replaces a type T in a term with a subtype of T. This tactic has two antecedents. One antecedent is a judgment that asserts the replacement type is a subtype of the substituted type. The other antecedent is the same as the conclusion except all occurrences of the supertype are replaced with the subtype. Fig. 1 only shows the second antecedent because the first antecedent can be proved automatically using TT-IQ's subtyping relationship.

In practice, the query engine will only show the judgments that require assistance from the user. There may be multiple tactics that can be applied to a judgment. The compatibility relationship can rank the tactics that match best. However, if two tactics have the same rank, then the user will need to select the tactic to apply.

Narrow Predicate is similar to Narrow Concept, except Narrow Predicate replaces a predicate symbol with another predicate symbol. Therefore, Narrow Predicate has antecedents to prove that the replacement predicate is a *sub-property* of substituted predicate. This requires two antecedents: an antecedent to prove that the replacement property is a sub-property of the existing property and an antecedent to prove that the type of the sub-property is a subtype of the type of the super-property. The first antecedent has to be proved using the taxonomy of the properties and features. The tactic also has a third antecedent that contains the sub-property instead of the super-property. The proof tree in Fig. 1 shows the judgment produced from the third antecedent. In particular, it shows "involving" replaced with "near".

The judgments produced by Concept Introduction illustrate the need for *dependent judgments*. Normally in type theory and in sequent calculus, judgments of the same parent are independent of each other. However, when using interactive proofs to query for data, terms created on one branch could be used in a judgment on a different branch. Notice that two of the goals produced by Concept Introduction contain $\coloneqq$. This special constructor informs the query engine to create a reference to the term that satisfies the type on the right hand side of$\coloneqq$. At some point, a tactic will be invoked that uses an analytic to create objects or find objects in the knowledge base to bind to the reference.

The Geo Search tactic uses an analytic to bind a reference to a collection of terms that are within a specific region. Geo

Search uses the goal as the criteria to search for objects within a specific region in the knowledge base. More specifically, Geo Search has a conclusion that has a type that contains one attribute, location. The type of location is the supertype of all types that could be used as the criteria for a geospatial search, such as KML. This type matches any type that has an attribute that is compatible with KML and the name of the attribute is interchangeable with location. The taxonomy determines which attributes are interchangeable with location. The tactic uses this attribute as the search criteria. In practice, the tactic may also require a time range.

Person Observation is a tactic that is created by a data scientist. In other words, it is an analytic that creates concepts inductively. This means it creates the definition of a type by generalizing existing data. Let's assume that Person Observation examines SIGINT data for calls originating form or made to the telephone number of the cellphone in the possession of a detainee. The time and location of the each call is used as an *observation point* of the person. In theory, a new triple is added to the knowledge base that links the person to the time and location of the call. In practice, the query engine may not create the triples. Instead, the query engine may define a way to create the triples on demand without altering the knowledge base unless directly instructed to do so. This approach is essential for cloud-scale data because it doesn't perform any destructive modifications if the analyst wants to back out changes. Instead the query engine could have a local cache containing the new triples. The modifications could be made permanent only when explicitly specified by the analyst or a data scientist.

The antecedent and conclusion of a tactic based on an analytic is determined by the precondition and postcondition of the analytic. The system uses the precondition as the antecedent and the postcondition as the conclusion. For Person Observation the postcondition states that there exists a concept P where each object is a Person that has the features of the detainee and whose locations correspond to the locations of the cell phone. The precondition requires that there exists a person in the knowledge base who has the features of the detainee. To satisfy the precondition, the analyst will need to add the detainee to the knowledge base. The precondition also requires that we can determine the locations of the cell phone. We assume another analytic will determine this location. For brevity, assume that the query engine can prove this automatically. As a result, it isn't included in Fig. 1.

We envision a suite of tactics that discover relationships between an entity and an event. These tactics use analytics that can create new relationships. In other words, they can add new triples to the knowledge base. Nearness Predicate belongs to this suite. Nearness Predicate uses an analytic that creates new triples using the near predicate. In other words, it creates triples of the form (P near E) where P is an entity and E is an event.

IV. Overview of TT-IQ

In this section, we give the formal definition of TT-IQ. Due to space limitations, we omit some rigor that would be found in a normal presentation of type theory.

Both types and objects are terms. We define the terms T and t as follows.

$$\begin{aligned} T, t \coloneqq\ & \bot \mid s | r | x | X | f(t_1, \ldots, t_n) | t.a | t.\mathbf{size} \\ & |\mathbf{tt}|\mathbf{ff}|(a_1 = t_1, \ldots, a_n = t_n)|[t_1, \ldots, t_n\,] \\ & \mid T \wedge T' | T \vee T' | T \rightarrow T' | \neg T | P(T_1, \ldots, T_n)|\mathbf{prop} \\ & |\mathcal{S}\ |\mathcal{R}|a_1{:}\,T_1 \times \ldots \times a_n{:}\,T_n | T^* | \{x{:}\,T|T'\} \\ & |\forall x{:}\,T.T' | \exists x{:}\,T.T'\ |\forall X \leq T.T'|\ \exists X \leq T.T'|\ \hat{X} \end{aligned}$$

In the definition of terms, s and r range over strings and numbers respectively; a ranges over attribute names; and X and x range over variables. The term $\bot$ represents null. The terms **tt** and **ff** represent true and false, respectively. The terms of the form $(a_1 = t_1, \ldots, a_n = t_n)$ represent records. Each $a_i = t_i$ in a record represents an attribute where a_i is the name of the attribute and t_i is the value of the attribute. The terms of the form $t.a$ represent selecting the value of an attribute whose name is a from a record t. Terms of the form $[t_1, \ldots, t_n\,]$ represent lists. Terms of the form $t.\mathbf{size}$ represent the number of elements in the list t. P and f range over predicate symbols and function symbols, respectively. $\mathcal{S}$ is the type of strings and $\mathcal{R}$ is the type of numbers. We call the terms s, r, x, $f(t_1, \ldots, t_n)$, $t.a$, $t.\mathbf{size}$, $\mathbf{tt}$, $\mathbf{ff}$, $(a_1 = t_1, \ldots, a_n = t_n)$, and $[t_1, \ldots, t_n\,]$ *objects*. We call all of the other terms, such as $\{x{:}\,T|T'\}$ and $\exists x{:}\,T.T'$, types.

Terms of the form $a_1{:}\,T_1 \times \cdots \times a_n{:}\,T_n$ represent record types and terms of the form T^* represent list types. The type **prop** represent the type that contains types that represent logical formulas, such as $T \wedge T'$ and $P(T_1, \ldots, T_n)$. Any type created using terms in **prop** will also be in **prop**. For example grt $(5,4) \wedge$ grt $(7,3)$ is a member of **prop**. Terms of the form $\{x{:}\,T|T'\}$ represent set types. Intuitively a set type $\{x{:}\,T|T'\}$ represents a list of objects of type T where each member of the list makes the type representing the logical formula T' true. We call $\hat{X}$ a *reference.* We use $\hat{X}$ to support injecting terms created by an analytic running in a separate subsystem. Notice there are two kinds of quantifiers, those that range over objects, $\forall x{:}\,T$ and $\exists x{:}\,T$, and those that range over types, $\forall X \leq T$ and $\exists X \leq T$. In the quantifiers that range over objects, $x{:}\,T$ means x is a member of type T. So, $\forall x{:}\,T$ means for all terms x that are members of the type T. In the quantifiers that range over types, $X \leq T$ means X is a subtype of T. So, $\forall X \leq T$ means for all types X that are subtypes of T.

Notice that types may contain objects. For example, if grt is a predicate symbol that represents greater than equal to and abs is a function symbol that represents absolute zero, then $\{x{:}\,\mathcal{R}|\text{grt}(\text{abs}(x), 0)\}$ is a type that contains the objects 0 and $\text{abs}(x)$.

Given any two terms t and t' and a variable x, $t[t'/x]$ is the term produced by replacing all free occurrences of x in t with t'. For example, $f(g(x,y),x)[3/x]$ is $f(g(3,y),3)$. For two terms t and T, we write $t{:}\,T$ to mean that t inhabits the type T or t is a member of the type T. For example, $3{:}\,\mathcal{R}$ and $[3,4,5]{:}\,\mathcal{R}^*$. We give some of the rules of TT-IQ for determining which terms inhabit a type in Fig. 4.

$$\frac{t_1 \rightharpoonup t_1' \quad \cdots \quad t_n \rightharpoonup t_n'}{(a_1 = t_1, \ldots a_n = t_n) \rightharpoonup (a_1 = t_1', \ldots, a_n = t_n')} (\textsc{Rec.} \rightharpoonup)$$

$$\frac{t \rightharpoonup (a_1 = t_1, \ldots, a_n = t_n) \quad a = a_i}{t.a \rightharpoonup a_i} (\textsc{Field Selection} \rightharpoonup)$$

$$\frac{t_1 \rightharpoonup t_1' \quad \cdots \quad t_n \rightharpoonup t_n'}{[t_1, \ldots, t_n] \rightharpoonup [t_1', \ldots, t_n']} (\textsc{List} \rightharpoonup)$$

$$\frac{t \rightharpoonup [t_1, \ldots, t_n]}{t.\mathbf{size} \rightharpoonup n} (\textsc{List Size} \rightharpoonup)$$

$$\frac{\Im(f(t_1, \ldots, t_n)) \mapsto t}{f(t_1, \ldots, t_n) \rightharpoonup t} (\textsc{Function Application} \rightharpoonup)$$

$$\frac{\Im(P(T_1, \ldots, T_n)) \mapsto t}{P(T_1, \ldots, T_n) \rightharpoonup t} (\textsc{Predicate Application} \rightharpoonup)$$

$$\frac{\Im(\hat{X}) \mapsto t}{\hat{X} \rightharpoonup t} (\textsc{Reference} \rightharpoonup)$$

Fig. 2. Evaluation rules for terms that do not evaluate to themselves.

The specification of an analytic can be generalized as follows. The specification will need to contain universal quantifiers to allow the query engine to pass in types and objects to the analytic. If the analytic takes in m types and n objects, then the specification will need the quantifiers $\forall X_1 \leq U_1. \cdots. \forall X_m \leq U_m$ and $\forall x_1{:}T_1. \cdots. \forall x_n{:}T_n$. We abbreviate these as $\forall \vec{X} \leq \vec{U}$ and $\forall \vec{x}{:}\vec{T}$, respectively. An analytic may output types and objects. If an analytic generates k types and l objects, then the specification will need to contain existential quantifiers $\exists Y_1 \leq U_1' \cdots \exists Y_k \leq U_k'$ and $\exists y_1{:}T_1'. \cdots. \exists y_l{:}T_l'$. We abbreviate these as $\exists \vec{Y} \leq \overrightarrow{U'}$ and $\exists \vec{y}{:}\overrightarrow{T'}$, respectively. The specification will also contain the precondition and the postcondition of the analytic. We denote these respectively as ϕ and ψ. The general term that represents the specification of an analytic is specified in (1).

$$\forall \vec{X} \leq \vec{U}. \forall \vec{x}{:}\vec{T}. \left(\phi \rightarrow \exists \vec{Y} \leq \overrightarrow{U'}. \exists \vec{y}{:}\overrightarrow{T'}. \psi\right) \tag{1}$$

In practice, the number of inputs and outputs of an analytic will be small. For example, the specification of Geo Search, an analytic that finds objects that occur within a region, is as follows.

$$\begin{aligned} &\forall X \leq \{\text{location:KML}\}. \forall r{:}\text{KML}.\mathbf{true} \\ &\qquad \rightarrow \exists e{:}X^*. \forall t{:}\, e.\text{inRegion}(r, t) \end{aligned} \tag{2}$$

In the specification of an analytic, the X's and Y's in (1) represent concepts and the x's and y's represent individuals. An analytic may also take relationships as input and output relationships. The relationships an analytic takes as input are defined by ϕ, and the relationships an analytic produces are defined by ψ. For example, the Geo Search analytic outputs a relationship $\text{inRegion}(r, t)$. The domain is defined by the type of r which is KML and the range is all subtypes of {location:KML}. Geo Search doesn't take any relationships as input.

Traditionally, the definition of a type theory includes rules for evaluating terms. As a result, most type theories in the literature are definitions of statistically typed functional programming languages. TT-IQ has rules for evaluation. We list a subset of the rules in Fig. 2. TT-IQ should not be considered a functional programming language. TT-IQ does not contain a construct for creating functions. Instead, TT-IQ defines a means for injecting functions and relations into it that are executed by an external subsystem or programming language. We represent the external subsystem or programming language as an *interpreter*. For any term t, an interpreter maps t to a term t'. We use $\Im$ to denote an interpreter. We write $\Im(t) \mapsto t'$ to mean that $\Im$ interprets t as t'. The interpreter is used to evaluate the application of function symbols and predicate symbols. For a term t, we write $t \rightharpoonup t'$ to mean t evaluates to t'. The rules $\textsc{Function Application} \rightharpoonup$ and $\textsc{Predicate Application} \rightharpoonup$ indicate that the application of a function symbol or a predicate symbol to a sequence of terms is equal to the interpretation of the application of the function symbol or predicate symbol. These rules do not require that the arguments of the function symbol and predicate symbol be evaluated before passing them to the interpreter. As a result, the interpreter can determine the mode of evaluation, such as lazy evaluation or eager evaluation. The evaluation rule $\textsc{Reference} \rightharpoonup$ illustrates the ability of the interpreter to manage storage of instance data. Intuitively $\textsc{Reference} \rightharpoonup$ means that a reference to a concept evaluates to the set of individuals in the concept. The concept is defined by the type T and $\hat{X}$ is a pointer to an index, graph or other data structure that contains members of the concept.

A paramount feature of TT-IQ is subtyping. TT-IQ uses a unique feature to typing that is required for semantic modeling. Traditionally, subtyping on record types is the same as class inheritance. In other words, a record type T is a subtype of T' if all the attributes in T' are also attributes in T. This means that for every attribute named a in T' there is attribute a in T. However, this definition of subtyping ignores the semantic meaning of attribute names. Two attributes can have different names, but mean the same thing. As a result, TT-IQ defines subtyping so that attribute names do not have to be syntactically the same, but semantically the same. Actually, in TT-IQ attribute names do not need to be equivalent, but one attribute name has to be subsumable by the other. The definition of subtyping in TT-IQ uses a partial ordering on attribute names to define subtyping. Due to space limitations we are unable to define the complete definition of subtyping. We do show the subtyping rule for record types below.

$$\frac{n \geq m \quad \text{for } i = 1, \ldots, m \quad a_i \sqsubseteq a_i' \quad T_i \leq T_i'}{a_1{:}T_1 \times \cdots \times a_n{:}T_n \leq a_1'{:}T_1' \times \cdots \times a_m'{:}T_m'}$$

In "Interactive Querying" we showed examples of tactics. A tactic is a program or script that returns a *proof tree.* A proof tree represents a proof of a *type judgment.* Intuitively a type judgment asserts that a term belongs to a type. A type judgment has the form $\Gamma \vdash T{:}T'$ where Γ represents an *environment* and $T{:}T'$ represents the assertion that T inhabits

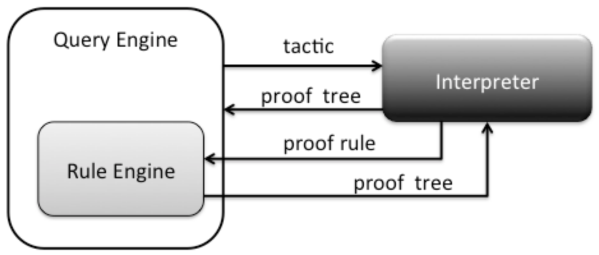


Fig. 3. Interaction between the query engine, the rule engine, and the interpreter.

T'. The environment, Γ, of a type judgment consists of type assignments, terms, predicate symbols and function symbols. It also consists of pairs of predicate symbols and attribute names. These pairs form a partial ordering, $\sqsubseteq$. If $A_1, \dots, A_n$ are type assignments of terms, predicate symbols, and function symbols, then $\sqsubseteq; A_1, \dots, A_n$ is an environment.

Only the rule engine can execute a proof rule. The rule engine is a subcomponent of the query engine. Given a rule, a proof tree, and judgment in the proof tree, the rule engine will apply the rule to the judgment. The result will be a proof tree that uses the antecedent of the rule to create children of the judgment. Fig. 3 illustrates the relationship between the query engine, the rule engine and the interpreter.

TT-IQ does not define the language in which tactics are written. An interpreter performs evaluation of tactics. Detailed discussion of the language for creating tactics is outside the scope of this paper. In this paper it suffices to say that a tactic takes as input a typing judgment and outputs a proof tree. The root of the tree has to be the input judgment. The tactic has to use proof rules to create the proof tree.

Fig. 4 contains a few proof rules for TT-IQ. We use $\exists_{\leq}$ ELIMINATION to specify that an analytic will retrieve the individuals of a concept represented as T or a subtype of T. The rule uses a concept reference so that we can postpone selection of the analytic until we have a judgment whose type matches the postcondition of an analytic. We use ANAYLIC EVAL to prove judgments that require an analytic. ANAYLIC EVAL generates concepts and individuals that satisfy a condition specified as a type. The condition can represent a relationship or the search criteria for a query. The condition is specified as the type T in ANAYLIC EVAL. An analytic whose postcondition *matches* T is used to generate the concepts and individuals that satisfy the condition specified as T. In the rule, the analytic is f and its postcondition is ψ. The top-right hypothesis is used to establish that T matches ψ if replacing the free variables in ψ produces a term that is a supertype of T. The terms that replace the free variables are the inputs to the analytic, $\vec{V}$ and $\vec{v}$, and the outputs of the analytic, $\vec{W}$ and $\vec{w}$. $\vec{V}$ is a sequence of types $V_1, \dots, V_m$ and $\vec{v}$ is a sequence of objects $v_1, \dots, v_n$. Likewise $\vec{W}$ is a sequence of types $W_1, \dots, W_k$ and $\vec{w}$ is a sequence of objects $w_1, \dots, w_l$. Intuitively, $\vec{V}$ and $\vec{v}$ are the concepts and individuals required by the analytic to produce the concepts and individuals $\vec{W}$ and $\vec{w}$ that satisfy the relationship defined by ψ. All free variables in each V_i and v_j are declared in Γ. Each W_i and each w_j do not have any free variables.

Since $\vec{V}$ and $\vec{v}$ represent the input to the analytic, we need to verify they satisfy the precondition ϕ. The three hypotheses of ANAYLIC EVAL on the left achieve this. The top two hypotheses are judgments to verify that the inputs to the analytic have the correct types. The last of the three hypotheses verifies that ϕ is true with all of its free variables replaced with inputs to the analytic. The top hypothesis on the right is used to verify that the postcondition is true for the inputs and outputs of the analytic. The second hypothesis from the top on the right indicates that the output of f on $\vec{V}$ and $\vec{v}$ produces $\vec{W}$ and $\vec{w}$. Recall from FUNCTION APPLICATION $\rightharpoonup$ in Fig. 2 that an interpreter is used to produce $\vec{W}$ and $\vec{w}$. The hypothesis on the bottom-right indicates that each W_i evaluates to a list of terms of type Y_i and that each w_j evaluates to a term that is of type T_j'. This hypothesis shows that we intend to represent concepts as list of terms of a specific type. The $\exists_{\{\}}$ REWRITE rule makes use of the fact that for any term t, if $t\!:\{y\!:T|T'\}$ then $t\!:T$ and

$$\frac{\Gamma, \hat{X} \leq T \vdash \phi[\hat{X}/X]\!:\!\mathbf{prop}}{\Gamma \vdash \exists X \leq T\!:\!\mathbf{prop}} \ (\exists_{\leq} \text{ ELIMINATION})$$

$$\frac{\Gamma \vdash \exists X \leq T.\,(\forall y\!:X.\,T') \wedge T''\!:\!\mathbf{prop}}{\Gamma \vdash \exists X \leq \{y\!:T|T''''\}.\,T''\!:\!\mathbf{prop}} \ (\exists_{\{\}} \text{ REWRITE})$$

$$\frac{\begin{array}{cc} \Gamma \vdash \vec{V} \leq \vec{U} & \Gamma \vdash T \leq \psi[\vec{W}/\vec{Y}][\vec{w}/\vec{y}][\vec{v}/\vec{x}][\vec{V}/\vec{X}] \\ \Gamma \vdash \vec{v}\!:\vec{T} & f(\vec{V},\vec{v}) \rightharpoonup [\vec{W},\vec{w}] \\ \Gamma \vdash \phi[\vec{v}/\vec{x}][\vec{V}/\vec{X}]\!:\!\mathbf{prop} & W_i \rightharpoonup t_i \ \mathit{and}\ \Gamma \vdash t_i\!:Y_i^* \quad w_j \rightharpoonup t_j' \ \mathit{and}\ \Gamma \vdash t_j'\!:T_j' \end{array}}{\Gamma, f\!:\forall \vec{X} \leq \vec{U}.\forall \vec{x}\!:\vec{T}.\big(\phi \to \exists \vec{Y} \leq \vec{U'}.\exists \vec{y}\!:\vec{T'}.\psi\big), \Gamma' \vdash T\!:\!\mathbf{prop}} \ (\text{ANAYLIC EVAL})$$

$$\frac{\Gamma \vdash S \leq T \quad \Gamma \vdash \exists X \leq S.\,T'\!:\!\mathbf{prop}}{\Gamma \vdash \exists X \leq T.\,T'\!:\!\mathbf{prop}} \ (\text{NARROW TYPE})$$

Fig. 4. A subset of the type rules of TT-IQ

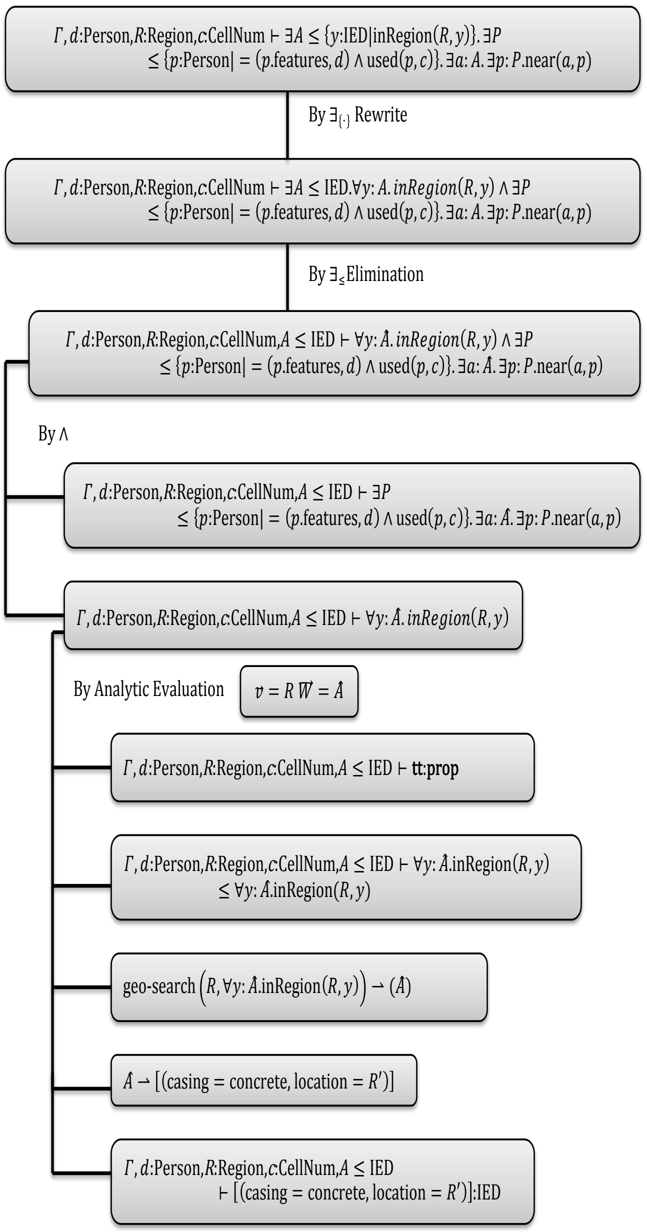

Γ, d:Person, R:Region, c:CellNum ⊢ ∃A ≤ {y:IED|inRegion(R, y)}. ∃P ≤ {p:Person| = (p.features, d) ∧ used(p, c)}. ∃a: A. ∃p: P.near(a, p)
By ∃{·} Rewrite
Γ, d:Person, R:Region, c:CellNum ⊢ ∃A ≤ IED.∀y: A. inRegion(R, y) ∧ ∃P ≤ {p:Person| = (p.features, d) ∧ used(p, c)}. ∃a: A. ∃p: P.near(a, p)
By ∃≤ Elimination
Γ, d:Person, R:Region, c:CellNum, A ≤ IED ⊢ ∀y: Á. inRegion(R, y) ∧ ∃P ≤ {p:Person| = (p.features, d) ∧ used(p, c)}. ∃a: Á. ∃p: P.near(a, p)
By ∧
Γ, d:Person, R:Region, c:CellNum, A ≤ IED ⊢ ∃P ≤ {p:Person| = (p.features, d) ∧ used(p, c)}. ∃a: Á. ∃p: P.near(a, p)
Γ, d:Person, R:Region, c:CellNum, A ≤ IED ⊢ ∀y: Á. inRegion(R, y)
By Analytic Evaluation
v = R W̄ = Á
Γ, d:Person, R:Region, c:CellNum, A ≤ IED ⊢ tt:prop
Γ, d:Person, R:Region, c:CellNum, A ≤ IED ⊢ ∀y: Á.inRegion(R, y) ≤ ∀y: Á.inRegion(R, y)
geo-search (R, ∀y: Á.inRegion(R, y)) ⇀ (Á)
Á ⇀ [(casing = concrete, location = R′)]
Γ, d:Person, R:Region, c:CellNum, A ≤ IED ⊢ [(casing = concrete, location = R′)]:IED

Fig. 5. Proof tree illustrating ∃ rules and analytic evaluation.

$T[t/y]$:**prop**.

Recall in Fig. 1, we used the Narrow Concept tactic to replace Attack with IED. Narrow Concept contains code to select the appropriate concept to use as the subtype for the Narrow Type rule. After it finds a subtype, it asks the rule engine to apply Narrow Type to a target type judgment in a proof tree. The rule engine creates a new judgment using the antecedent of Narrow Type as a template. The new judgment is added as a child of the target type judgment. The rule engine returns the new proof tree to Narrow Concept and Narrow Concept returns the proof tree to the query engine. Fig. 5 contains a portion of the proof tree created from the Geo Search and Concept Introduction tactics in Fig. 1.

V. Discussion

A. *Believability*

A query engine for intelligence data needs to support various levels of believability. We can support believability in TT-IQ by using *type modality [29]*. In other words, we can annotate type with a modality operator that represents a level of believability, such as **certain**, **likely**, **not-likely**, and **impossible**. Then an analyst could prove a hypothesis represented by the type T is likely to occur as the judgment $\Gamma \vdash T^{\textbf{likely}}$: **prop** or disprove it by proving the judgment $\Gamma \vdash T^{\textbf{impossible}}$: **prop.**

B. *Too complex for an analyst*

The formalism of TT-IQ and the method of reasoning employed by TT-IQ may be too complicated for an analyst. We don't expect the analyst to specify queries in the formal language of TT-IQ, but in natural language similar to that used in Fig. 1. We could employ a technique similar to that used in [14] to allow end users to specify queries using natural language.

C. *Non-determinism and Subtyping*

Since a type may have multiple subtypes, a tactic that finds or creates a subtype of a type could be nondeterministic. In other words, the tactic may not produce the same subtype for the same supertype. As a result, the query engine could produce different results for the same query over the same data. We can resolve non-determinism by asking the user to select the appropriate subtype. This approach would be similar to faceted search. The query engine would require the end user to select from a list of subtypes to use as a candidate to narrow the search space.

D. *Implementation of TT-IQ*

Currently, we are in the planning stage of creating an implementation of TT-IQ. We plan to create an implementation of TT-IQ using Coq and R. Coq will provide the interactive reasoning capability. R will be used as the language and runtime for defining and executing analytics. We anticipate having to add a library to R to provide a more seamless interaction with RDF than the existing R libraries.

We consider this implementation of TT-IQ a proof-of-concept implementation. We plan to use this implementation to conduct research to address usability issues and determine strengthens and weaknesses of interactive semantic querying over automatic semantic querying.

The analyst workstation will contain an analytic framework that would provide an interface to support contribution of analytics written in a wide range of languages, such as MATLAB, C, Java, and Python.

VI. Conclusion

In this paper, we showed how to apply techniques from ITPs (interactive theorem provers) to analyze military intelligence. Users of ITPs apply small programs called tactics in an iterative fashion to construct a proof. We demonstrated how tactics could be used to answer semantic queries interactively. Furthermore, we showed how to incorporate analytics that use machine learning, knowledge discovery, or network analysis into the querying process.

A. *Future Work*

In "Believability", we eluded to an approach to handle uncertainty. Future work should investigate this approach. Also, we should consider how to incorporate the approach in [15] into our type system.

In the future, we would like to investigate how to implement interactive querying in an existing military intelligence cloud system, such as the DCGS-A Cloud. We believe our approach to querying would be a good fit for the semantic enhancement approach adopted by the DCGS-A Cloud.

ACKNOWLEDGMENT

Rod Moten thanks Andy Zhao of Sotera Defense Solutions and Dr. Aaron Stump of the University of Iowa on their valuable feedback on the ideas expressed in this paper.

REFERENCES

[1] Wylie Wong, "The Army Brings the Cloud to the Battlefield," *FedTech Magazine*, 31-Jul-2013. [Online]. Available: http://www.fedtechmagazine.com/article/2013/07/army-brings-cloud-battlefield. [Accessed: 22-Aug-2013].

[2] S. Miakhel, "Understanding Afghanistan: The importance of tribal culture and structure in security and governance," *Asian Survey*, vol. 35, no. 7, 1995.

[3] B. Ulicny, G. M. Powell, C. J. Matheus, M. Coombs, and M. M. Kokar, "Priority Intelligence Requirement Answering and Commercial Question-Answering: Identifying the Gaps," DTIC Document, 2010.

[4] LTG Keith Alexander, BG Mike Ennis, Robert L. Grossman, James Heath, Russ Richardson, Glenn Tarbox, and Eric Sumner, "Automating Markup of Intelligence Community Data: A Primer," *Defense Intelligence Journal*, no. 12–2, pp. 83–96, 2003.

[5] D. Davis, R. Lichtenwalter, and N. V. Chawla, "Supervised methods for multi-relational link prediction," *Soc. Netw. Anal. Min.*, vol. 3, no. 2, pp. 127–141, Jun. 2013.

[6] D. Davis, R. Lichtenwalter, and N. V. Chawla, "Multi-relational Link Prediction in Heterogeneous Information Networks," in *2011 International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2011, pp. 281–288.

[7] S. David, M. Tatiana, H. Alan, C. Shaun, and S. Barry, "Integration of Intelligence Data through Semantic Enhancement," in *Proceedings of the Sixth International Conference on Semantic Technologies for Intelligence, Defense, and Security*, Fairfax, VA, USA, pp. 6–13.

[8] B. Smith, T. Malyuta, W. S. Mandrick, C. Fu, K. Parent, and M. Patel, "Horizontal Integration of Warfighter Intelligence Data: A Shared Semantic Resource for the Intelligence Community," in *Proceedings of the Seventh International Conference on Semantic Technologies for Intelligence, Defense, and Security*, 2012.

[9] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[10] G. Zenz, X. Zhou, E. Minack, W. Siberski, and W. Nejdl, "Interactive Query Construction for Keyword Search on the Semantic Web," in *Semantic Search over the Web*, R. De Virgilio, F. Guerra, and Y. Velegrakis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 109–130.

[11] S. Ferré, A. Hermann, and M. Ducassé, "Combining Faceted Search and Query Languages for the Semantic Web," in *Advanced Information Systems Engineering Workshops*, C. Salinesi and O. Pastor, Eds. Springer Berlin Heidelberg, 2011, pp. 554–563.

[12] H. Pérez-Urbina, E. Rodrıguez-Dıaz, M. Grove, G. Konstantinidis, and E. Sirin, "Evaluation of query rewriting approaches for OWL 2," in *Proc. of the Joint Workshop on Scalable and High-Performance Semantic Web Systems (SSWS+ HPCSW 2012)*, 2012, vol. 943.

[13] G. Tecuci, *Building intelligent agents: an apprenticeship multistrategy learning theory, methodology, tool and case studies*. San Diego: Academic Press, 1998.

[14] G. Tecuci, M. Boicu, C. Boicu, D. Marcu, B. Stanescu, and M. Barbulescu, "The Disciple–Rkf Learning and Reasoning Agent," *Computational Intelligence*, vol. 21, no. 4, pp. 462–479, 2005.

[15] M. Boicu, D. Marcu, G. Tecuci, and D. Schum, "Cognitive Assistants for Evidence-Based Reasoning Tasks," in *2011 AAAI Fall Symposium Series*, 2011.

[16] G. Tecuci, D. Schum, M. Boicu, D. Marcu, and Hamilton, Benjamin, "TIACRITIS System and Textbook: Learning Intelligence Analysis through Practice," in *Proceedings of the Fifth International Conference on Semantic Technologies for Intelligence, Defense, and Security*, 2010.

[17] G. Tecuci, D. Marcu, M. Boicu, D. Schum, and K. Russell, "Computational Theory and Cognitive Assistant for Intelligence Analysis," in *Proceedings of the Sixth International Conference on Semantic Technologies for Intelligence, Defense, and Security – STIDS 2011*, Fairfax, VA, USA, 2011, pp. 66–75.

[18] S. F. Allen, M. Bickford, R. L. Constable, R. Eaton, C. Kreitz, L. Lorigo, and E. Moran, "Innovations in computational type theory using Nuprl," *Journal of Applied Logic*, vol. 4, no. 4, pp. 428–469, Dec. 2006.

[19] *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. .

[20] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer, 2002.

[21] R. Dapoigny and P. Barlatier, "Using a Dependently-Typed Language for Expressing Ontologies," in *Knowledge Science, Engineering and Management*, H. Xiong and W. B. Lee, Eds. Springer Berlin Heidelberg, 2011, pp. 257–268.

[22] R. Dapoigny and P. Barlatier, "Formal foundations for situation awareness based on dependent type theory," *Information Fusion*, vol. 14, no. 1, pp. 87–107, Jan. 2013.

[23] G. De Giacomo, M. Lenzerini, and R. Rosati, "Higher-Order Description Logics for Domain Metamodeling.," in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011, pp. 183–188.

[24] F. A. Lisi, "A Declarative Modeling Language for Concept Learning in Description Logics," in *Inductive Logic Programming*, F. Riguzzi and F. Železný, Eds. Springer Berlin Heidelberg, 2013, pp. 151–165.

[25] Z. Abedjan and F. Naumann, "Improving RDF Data Through Association Rule Mining," *Datenbank Spektrum*, vol. 13, no. 2, pp. 111–120, Jul. 2013.

[26] C. Benzmüller and A. Pease, "Higher-order aspects and context in SUMO," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 12–13, pp. 104–117, Apr. 2012.

[27] M.-L. Mugnier, "Ontological query answering with existential rules," in *Proceedings of the 5th international conference on Web reasoning and rule systems*, Berlin, Heidelberg, 2011, pp. 2–23.

[28] Moten, Rod, "Using a Type-theoretic Approach to Resolve Heterogeneity in Data Fusion," in *Proceedings of the National Symposium on Sensor and Data Fusion 2012*, Washington, D.C., 2012.

[29] A. Nanevski, F. Pfenning, and B. Pientka, "Contextual modal type theory," *ACM Trans. Comput. Logic*, vol. 9, no. 3, pp. 23:1–23:49, Jun. 2008.