PHERL: an Emerging Representation Language for Patterns, Hypotheses, and Evidence

Kenneth Murray, Ian Harrison, John Lowrance, Andres Rodriguez, Jerome Thomere, and Michael Wolverton

SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
{murray harrison lowrance rodriguez thomere wolverton} @ ai.sri.com

Abstract

Managing and analyzing information remains an enduring and essential challenge for intelligence professionals. Link analysis tools are beginning to be adopted in the intelligence community as part of the community's analysis toolset. However, since a wide variety of types of analyses is possible, even within just the link analysis field, no one tool is likely to be sufficient to complete an analysis. Instead, analysts will be required to use a suite of tools to accomplish their tasks. This motivates the need for interchange languages between tools, so they can be used in coordinated workflows to meet the needs of the intelligence analyst. We are developing PHERL, an interchange language for link analysis tools, which is designed to support the sharing of patterns and hypotheses (e.g., pattern-match results). This paper presents our preliminary work and solicits contributions of representation requirements from the link analysis community.

Introduction

Managing and analyzing information remains an enduring and essential problem for intelligence professionals. The vast volume of data and speed at which it must be considered motivates using new and sophisticated tools for searching, filtering, and evaluating information. Included in these emerging information management tools are *link analysis* tools. These include tools for discovering patterns of interest in relational data, as well as tools for using patterns to detect links between disparate data points ("connecting the dots"). No one tool or algorithm will span the full needs of the intelligence analyst: instead, a suite of tools is often needed, used in a serial or parallel workflow, transforming and augmenting raw data, and then filtering and analyzing the data sufficiently to enable the analyst to answer particular questions posed by the ultimate consumer of the intelligence.

This situation motivates the need to develop a shared representation of hypotheses, patterns and evidence/data, such that the tools can be coordinated and integrated with one another. We are working within a community of technology developers to support the development of a *link analysis toolset* for the intelligence community. Within this community previous work has focused on developing a common data schema. This succeeded in enabling all tools to participate within a single evaluation program. A later thrust on developing a common representation language for patterns and hypotheses (e.g., candidate pattern matches) had only limited success, due in part to its overemphasis on pattern representation and the lack of attention applied to representing hypotheses. At its core, representing a hypothesis involves capturing how some data matches a pattern, so that one cannot consider hypothesis representation as an adjunct to pattern representation: they are intimately tied to one another

Based on our previous experiences, we are now developing a Pattern, Hypothesis, and Evidence Representation Language (PHERL, pronounced "furl"), which is designed to be an interlingua to support communication among link analysis tools. The goal of this paper is to present preliminary work on PHERL and to encourage other developers in the link analysis community to contribute their representation requirements so that they can be supported in the emerging representation language.

We begin with a discussion of use cases that motivate our representation requirements. These use cases are set within the context of supporting intelligence professionals, but we believe that most of these features will be applicable outside that domain. We then follow with the requirements that we have derived from our understanding of what is required to support these use cases. Finally, we present examples of our pattern representation language.

Use Cases

Integration of link analysis tools is most likely to succeed at the data level, which implies that the output from one tool (hypotheses) is understandable as input to another

tool. In the uses cases below, we identify two classes of what "understanding" means. The first is that hypotheses from one tool essentially become data sources for another tool. In this meaning, hypotheses can be seen as augmented data. Examples of such hypotheses, in the domain of the intelligence community, might be determination that two identities are aliases of one another. In the second meaning, one link analysis tool may have only been able to (or was only asked to) produce hypotheses where a part of a pattern is matched. Another tool then uses this hypothesis and transforms it into a new query to be applied to data. In this second meaning, the query results (hypotheses) used as input to the second tool are not simple data (e.g., RDF triples, database records), but are partially instantiated pattern matches from the first tool. The second tool's job is to extend and build on the first tool's results.

1. Application-Based Use Cases

A number of potential use cases involving link analysis tools have been identified for applications in the intelligence community. All of these use cases can be used in a standing query mode or an *ad hoc* mode, as a response to a one-off request for information (RFI).

1.1 Message filtering. An analyst arrives at work and runs message-filter queries over the 10,000 incoming messages that have arrived overnight. Those messages relevant to the different standing queries are added to the analyst's inbox for further consideration and analysis.

A new RFI comes in requesting an assessment of the evolving leadership power struggle in a particular terrorist group. The analyst develops a query pattern and puts it into the standing query list so that relevant messages will be culled from the overnight message stream and added to the analyst's inbox.

A new RFI comes in requesting an assessment of the changes in a group's intent during the last 90 days. The analyst develops a query pattern and runs it against the message archive to retrieve relevant messages.

1.2 Mapping out a social network. An RFI arrives on an analyst's desk, asking to locate person X, who has disappeared. The analyst develops a social network map for person X and then queries for communications from/to person X. The analyst iterates, developing a richer social network map. Based on this the analyst is able to identify a list of people who may know person X, and based on the communication traffic, have a better handle on where X is possibly located.

1.3 Mapping out capability models. An RFI arrives on an analyst's desk, asking to assess the weapons of mass destruction (WMD) capability of a particular threat group. A predefined set of capability model queries is run, looking at prior activities, the individual and collective capabilities of the group members, the resources possessed, and transactions and communications with others connected with WMD resource or capability acquisition.

1.4 Detecting terrorist plan prior to attack. An RFI arrives on an analyst's desk, asking to track a particular group. The analyst uses predefined threat patterns to query transaction and communication databases for known group members. Returned results are potentially instantiated plans, each identifying where the group members may be in their planning and suggesting where to focus resources for further investigation and response.

1.5 Incremental, prospective alerts. The analyst is asked to monitor streaming, time-stamped data for evidence of terrorist activity, giving notification when the data provides "suggestive" evidence of a viable threat and providing updates as subsequent evidence adds to, or detracts from, the certainty and maturity of the possible threat.

2. Workflow-Based Use Cases

A number of potential use cases have been identified that capture anticipated workflows involving link analysis tools.

2.1 Pattern-refinement cycle. An analyst uses a pattern editor to create a pattern to detect some *situation of interest*. The analyst then begins an incremental *test and modify* refinement cycle that involves matching the current pattern against some data, using a link discovery tool, reviewing the results of the test using a hypothesis visualization tool, and then modifying the pattern (Wolverton et al. 2005). This pattern-refinement cycle repeats until the pattern performs as desired.

2.2 Merged hypotheses from several tools. In all of the use cases above, a single tool might have been unable to return all the results, or may have only been tasked to return results about a part of the overall query (e.g., one tool is used for a quantitative, probabilistic analysis, and another tool for a discreet, qualitative analysis). The partial results must be merged to produce the final results. Result generation by the subordinate tools might involve a parallel or serial process. Hypothesis merging likewise might require a single process (for parallel result generation) or multiple processes (for serial result generation).

2.3 Pipeline. The results from one tool may be used directly, or in a preprocessed form, as input to another tool, as part of an overall serial workflow process

2.4 Blackboard. Query results may be posted to a central area (blackboard) and used opportunistically by other tools (knowledge sources); for example, further analysis for a compelling but not yet conclusive hypothesis could be triggered when designated data conditions become satisfied.

Three Tiers of Representation

Our analysis of the workflow use cases led us to consider three distinct tiers of representation in support of link analysis.

- The first tier captures the pattern; this is used by tools that discover patterns in data, pattern-editing tools that are used by intelligence analysts, and pattern-matching tools that perform link detection.

- The second tier captures the hypothesis; this includes both a query and an answer. The query includes the pattern, the dataset to match the pattern against, and the tool to use. The answer includes the data bindings, if any, that are matched to the pattern by the tool. Hypotheses are produced by link discovery tools and are used by hypothesis visualization tools, and tools operating at the third tier (see below).

- The third tier captures how hypotheses can be integrated and used as evidence within larger analyses. This includes hypothesis management tasks such as merging hypotheses, ranking and comparing competing hypotheses, evaluating hypotheses over time as data and context change, defining and supporting a "hypothesis life cycle," and using link analysis hypotheses as evidence within higher-level structured argumentation tools, for example, SEAS (Lowrance et al. 2001). Although there has been little work on support for hypothesis management in our toolset community to date, PHERL should be informed by these potential applications and designed to accommodate them.

The following sections discuss representing the pattern and hypothesis tiers and very preliminary considerations for representing the evidence tier.

Representing Patterns

The cornerstone of link analysis is the pattern. We consider explicit, declarative representations for patterns (e.g., vs. algorithmic or neural-network descriptions).

Pattern Representation Requirements

The pattern representation should support capturing the following types of information:

1. The pattern definitions must have clear semantics (see **Representing Patterns with Prototypes** below).

2. Pattern terms, including:

2.1 Constant terms, including strings, numbers, and ontology-defined constant terms, such as classes and instances.

2.2 Variables:

2.2.1 Optional variables. Some variables may be designated as *optional* (vs. *required)*. For optional variables, bindings need not be found to establish an answer; for required variables, bindings must be found.

2.2.2 Boolean variables. Some variables may be designated as *boolean* (vs. *value*). For boolean variables, bindings must be found but the identity of the binding is deemed uninteresting, and different bindings need not be distinguished in distinct answers. Bindings of normal (e.g., value) variables are included as usual (each distinct binding warrants a distinct pattern match in the answer). For example in (child ?x ??y) the "y" variable is deemed a boolean variable (in this case, indicated by the "??" prefix), while the "x" variable is a normal (value) variable. The answer would include each binding for ?x such that some child (i.e., some binding of ??y) was found, and the answer would not include multiple children of a single binding for ?x. This distinction is useful when some/any binding must be found for a variable to warrant some finding (e.g., that someone is a parent) but alternative bindings are of no interest.

2.2.3 Binding-inclusion directives. OWL-QL (Fikes, Hayes, and Horrocks 2003) identifies distinctions (e.g., *must-bind*, *may-bind* and *don't-bind*) for capturing the desired binding properties of variables. The bindings to must-bind variables are always reported (i.e., required valued variables). The bindings to may-bind variables are reported opportunistically, when they are available (i.e., optional, value variables). The bindings to don't-bind variables are never reported.

2.2.4 Group variables: qGraph (Blau, Immerman, and Jensen 2002) identifies "group" variables, where all bindings for a designated variable are grouped into each distinct answer rather than each binding generating a distinct answer. For the example above, this would return an answer for each parent that included all known children for that parent.

2.3 Computed terms: functions (called "column functions" in GraphLog, see Eigler 1994) over other terms or their binding sets that compute some value, for example, to count the number of members in a threat group, to compute the maximum deposit to a bank account.

3. Pattern relations: predicates applied to pattern terms, or simple regular expressions over predicates applied as predicates to pattern terms; for example, in GraphLog, *before** might denote the transitive closure of predicate

before (Consens, Mendelzon, and Ryman 1994). Patterns may include two types of relations:

3.1 Data relations: requirements expressed as predicates over variables that bind to data elements.

3.2. Constraints: requirements that refer to constant terms, such as numbers, strings, ontology classes or instances, and so on, or computed terms rather than only to variables that bind to data elements. For example:

3.2.1 Type constraints: classes used to restrict which data elements can bind to a variable.

3.2.2 Cardinality constraints: the min or max number of data elements that bind to a variable or subpattern.

3.2.3 Relational constraints: predicates relating variables or computed terms to constants or computed terms.

Constraints can be designated as requiring either that each binding is consistent with (i.e., is not known to violate) the constraint condition, or that the binding complies with (i.e., is known to satisfy) the constraint condition.

4. Hierarchical patterns: (possibly recursive) subpatterns within patterns (Wolverton and Thomere 2005, Eigler 1994).

Pattern elements are defined as pattern terms, pattern relations, and subpatterns.

5. Disjunction and conjunction over pattern relations and subpatterns.

6. Negation, denoting bindings must not be found for a variable or a relation or subpattern or that a constraint is not deemed satisfied (Yeh, Porter, and Barker 2003).

7. Optional pattern elements (i.e., bindings are optional for designated variables or relations or subpatterns).

8. A pattern-specific search plan (e.g., that indicates the order of variables or relations or subpatterns in which bindings are sought).

9. Belief (e.g., probability) that a matched variable, or relation, or subpattern warrants determining that a match has been found for the pattern as a whole.

10. Meta-data

10.1 Ontology: the ontology used to express this pattern (e.g., the predicates, the classes).

10.2 Inception attribution: the author, creation date, project or case or RFI, and comment.

10.3 Modification attribution: for each pattern update, the date, author, project, comment, and new version number.

10.4 A serialized handle (e.g., URI), allowing patterns to be archived, retrieved, and distributed as collections or individually.

10.5 The urgency of responding to a candidate match.

10.6 The performance of the pattern (e.g., precision, recall, cost) on test data with known ground truth or with prior evidence after further investigation of the hypothesis.

10.7 The pattern classes of the pattern (discussed below).

10.8 The queries (hypotheses) in which a pattern has been used.

13. Aggregation: patterns can be grouped into libraries or collections (e.g., that share meta-data, such as author).

14. Inter-pattern relations, for example, to denote that one pattern specializes another, or that one provides confirming or disconfirming evidence for another.

These representation requirements do not legislate that every link analysis tool in the developing toolset supports all features (e.g., certainty, negation, hierarchical composition). But because some tools may include support for a given feature, the interlingua representation should support the feature in order to facilitate pattern sharing between those tools.

Representing Patterns with Prototypes

Often we want to capture knowledge by describing a representative example. An example is often easier for a pattern author to conceive and articulate than are rules expressed in first-order logic (FOL). The example-level representation comprises simple ground-atomic formulas (GAFs); these are direct and compact; they are free of the extra symbols (e.g., variables) and syntax (e.g., quantifiers) and the indirection required to capture knowledge using quantified rules. Because they are simple, direct, and compact, they are easier to present (e.g., as a graph) and easier for subject-matter experts to author and edit (Chaudhri et al. 2004, Pool et al. 2003). However, an example description does not by itself support making inferences involving other data; supporting their application to other data is of course the reason for the extra symbols and syntax, the variables and the quantifiers, in rules. But is it possible to integrate the simplicity of instance-level descriptions and the clear semantics of quantified rules? One approach is to start with an example description and then to add a small extension to that example description that allows it to function as FOL rules and so support the desired inferences with other data. We call the extended example a prototype: axioms capturing an instance-level example description plus some additional

axioms that establish the prototype status of the example such that prototype axioms collectively are equivalent to a designated set of FOL rules and so capable of supporting inferences by applying to other data, other examples. Here, we describe a prototype-based representation scheme for capturing patterns being developed for PHERL.

The prototype scheme is similar to and extends the support for prototypes provided in KM (Clark and Porter 2004, Clark et al. 2001), which captures necessary conditions for class (the first of five types of prototypes described below). Our approach extends this with support capturing sufficient conditions for class membership. It also supports capturing necessary and sufficient criteria for conditions other than class membership (e.g., predicates). While prototypes for classes and predicates support inference, the scheme also provides prototypes for queries to support data filtering. The prototype scheme supports explaining the application of sufficient prototypes to data by identifying what terms in the prototype pattern match what terms in the data. It supports the hierarchical use of prototypes in two ways. A prototype can be explicitly authored in terms of one or more subprototypes. Alternatively, a predicate over terms in a prototype can be satisfied by chaining on any sufficient prototypes defined for that predicate. It also supports representing optional terms and optional axioms appearing within a prototype.

Class Prototype, Necessary Conditions. The prototype describes an arbitrary instance of the class. More specifically, the prototype formula (i.e., the set of axioms that describe the "example") captures properties that hold for every instance of the class. One term designates the class instance; it is interpreted as a universally qualified variable. All other variables are interpreted as existentially quantified. Figure 1 presents a graphically depicted pattern that captures a necessary condition for the class *Person*: *every person has a mother*:

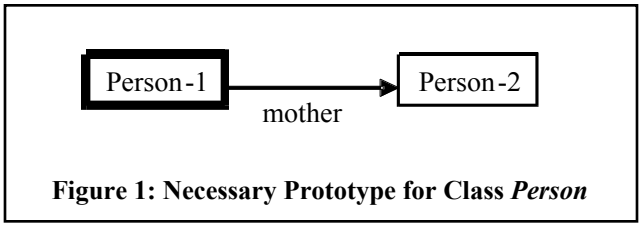

Figure 1: Necessary Prototype for Class *Person*

The class instance, Person-1, is identified visually by the heavy node outline; the type constraint is indicated by the name of the node.

This pattern denotes the following FOL sentence:

```
(forall ?x (implies (isa ?x Person)
                    (exists ?y (and (isa ?y Person)
                                    (mother ?x ?y)))))
```

This pattern is captured in PHERL as

```
(and (isa Person-1 Person)
     (isa Person-2 Person)
     (mother Person-1 Person-2)
     (isa Person-1 Class-Prototype)
     (isa Person-1 Necessary-Prototype)
     (isa Person-2 Prototype-Term)
     (class-prototype Person Person-1)
     (prototype-term-required Person-1 Person-2)
     (prototype-lhs Person-1 (isa Person-1 Person)))
```

Prototype-Terms are interpreted as variables within the "prototype rule"; their quantifications are determined by the type of the prototype in which they appear (as indicated by the prototype-term-required literal). Class-Prototype is a subclass of Prototype-Term so its instances are also treated as variables. The conjuncts that do not involve such prototype-defining vocabulary (e.g., in this example, the first three conjuncts) make up the "instance-level example" and are included in the graphical display of the prototype.

Class Prototype, Sufficient Conditions. The prototype example captures one condition that warrants establishing class membership. In this type of prototype, all variables are deemed to be universally quantified; one such variable is designated as the instance being classified. The pattern of Figure 2 captures a sufficient condition for the class *Parent*: *any person that has a child is a parent.*

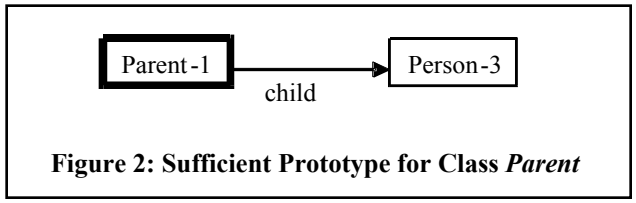

Figure 2: Sufficient Prototype for Class *Parent*

This pattern denotes the following FOL sentence:

```
(forall ?x,??y (implies (child ?x ??y)
                        (isa ?x Parent)))
```

This pattern is captured in PHERL as

```
(and (isa Parent-1 Person)
     (isa Person-3 Person)
     (child Parent-1 Person-3)
     (isa Parent-1 Class-Prototype)
     (isa Parent-1 Sufficient-Prototype)
     (isa Person-3 Boolean-Prototype-Term)
     (class-prototype Parent Parent-1)
     (prototype-term-required Parent-1 Child-2)
     (prototype-rhs Parent-1 (isa Parent-1 Parent)))
```

As a Boolean-Prototype-Term, *Person-3* is interpreted as a boolean variable. The "instance-level example" again comprises the first three conjuncts.

Predicate Prototype, Necessary Conditions. The prototype example captures (one set of) conditions that can be established whenever the predicate is established. In this type of prototype, each argument position of the predicate is associated with a prototype term, all of those argument terms that are variables are interpreted as universally quantified, and all other variables are interpreted as existentially quantified. For example, the pattern of Figure 3 captures a necessary condition for the predicate *wife*: *when a man has a wife then the wife was the bride in a wedding in which he was the groom.*

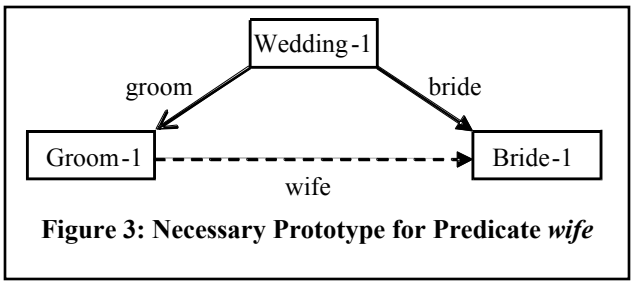

Figure 3: Necessary Prototype for Predicate *wife*

The antecedent edge (i.e., the predicate for which the condition is necessary) is identified visually here as a dashed line. This pattern denotes the following FOL sentence:

```
(forall ?x,?y
    (implies (wife ?x ?y)
             (exists ?z (and (isa ?z Wedding-Event)
                             (bride ?z ?y)
                             (groom ?z ?x)))))
```

This pattern is captured in PHERL as

```
(and (wife Groom-1 Bride-1)
     (bride Wedding-1 Bride-1)
     (groom Wedding-1 Groom-1)
     (isa Wedding-1 Wedding-Event)
     (isa Wedding-1 Prototype-Term)
     (isa Bride-1 Prototype-Term)
     (isa Groom-1 Prototype-Term)
     (isa Wife-P1 Predicate-Prototype)
     (isa Wife-P1 Necessary-Prototype)
     (predicate-prototype wife  Wife-P1)
     (prototype-term-required Wife-P1 Wedding-1)
     (prototype-term-required Wife-P1 Bride-1)
     (prototype-term-required Wife-P1 Groom-1)
     (prototype-lhs Wife-P1 (wife Groom-1 Bride-1)))
```

Predicate Prototype, Sufficient Conditions. The prototype example captures one condition that warrants establishing the predicate. In this type of prototype each argument position of the predicate is associated with a prototype term and all variables are interpreted as universally quantified. For example, the pattern of Figure 4 captures a sufficient condition for the predicate *alias*: *when two distinct identities share a social security number they are aliases.*

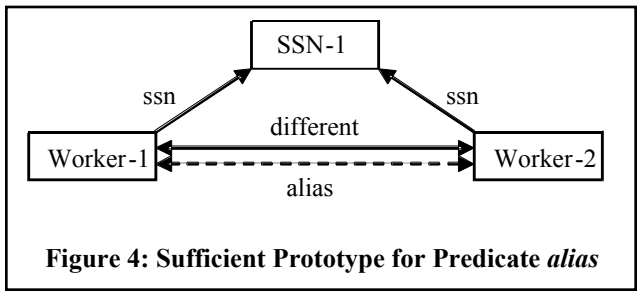

Figure 4: Sufficient Prototype for Predicate *alias*

The consequence edge is identified visually here as a dashed line. This pattern denotes the following FOL sentence:

```
(forall ?x,?y,?z (implies (and (ssn ?x ?z)
                               (ssn ?y ?z)
                               (different ?x ?y))
                          (alias ?x ?y)))
```

This pattern is captured in PHERL as

```
(and (alias Worker-1 Worker-2)
     (ssn Worker-1 SSN-1)
     (ssn Worker-2 SSN-1)
     (different Worker-1 Worker-2)
     (isa Alias-P1 Predicate-Prototype)
     (isa Alisa-P1 Sufficient-Prototype)
     (isa Worker-1 Prototype-Term)
     (isa Worker-2 Prototype-Term)
     (isa SSN-1 Prototype-Term)
     (predicate-prototype alias Alias-P1)
     (prototype-term-required Alias-P1 Worker-1)
     (prototype-term-required Alias-P1 Worker-2)
     (prototype-term-required Alias-P1 SSN-1)
     (prototype-rhs Alias-P1 (alias Worker-1 Worker-2)))
```

Query Prototype. The prototype example captures some situation of interest (which need not be defined as necessary or sufficient for any predicate); i.e., it captures a query. In query prototypes, all variables are interpreted as existentially quantified. For example, the pattern of Figure 5 captures the query: *what transfers have occurred of $1000 or more, and to whom, from a member of al Qaeda?*

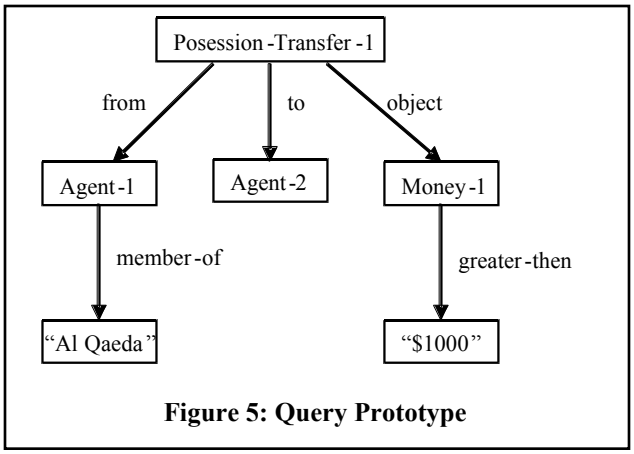

Figure 5: Query Prototype

The constraints are visually identified by the quoted node names. This pattern denotes the following FOL query sentence:

```
(exists ?w,?x,?y,?z
   (and (member Al-Qaeda ?w)
        (isa ?x Possession-Transfer-Event)
        (from ?x ?w)
        (to ?x ?y)
        (object ?x ?z)
        (> ?z $1000)))
```

This pattern is captured in PHERL as

```
(and (member Al-Qaeda Agent-1)
     (from Transfer-1 Agent-1)
     (to Transfer-1 Agent-2)
     (object Transfer-1 Money-1)
     (> Money-1 $1000)
     (isa Transfer-1 Possession-Transfer-Event)
     (isa Funding-P1 Query-Prototype)
     (isa Transfer-1 Prototype-Term)
     (isa Agent-1 Prototype-Term)
     (isa Agent-2 Prototype-Term)
     (isa Money-1 Prototype-Term)
     (prototype-term-required Funding-P1 Transfer-1)
     (prototype-term-required Funding-P1 Agent-1)
     (prototype-term-required Funding-P1 Agent-2)
     (prototype-term-required Funding-P1 Money-1))
```

Each of these five classes of prototypes supports a class of patterns (e.g., patterns for establishing class membership, patterns for establishing a predicate holds over some data). They support an intuitively meaningful graphical display for patterns while capturing the clear semantics of their corresponding FOL rules.

Representing Hypotheses

Representation Requirements

In our toolset community, a hypothesis is defined to be a query (e.g., a pattern, matched against a dataset by a link discovery module) and an answer (e.g., zero, one, or more matches to the pattern among the accessible data). As an ideal, the representations of a query and answer should include all necessary information to re-run the query (using the same pattern, module, and data) to obtain the same answer. This is an ideal because, for example, some aspects of some modules may be nondeterministic. In support of this ideal, the hypothesis representation requirements should capture the following information.

1. Representing the Pattern (see above)

2. Datasets(s) used for answering the query

2.1 The accessible datasets (e.g., knowledge bases, databases).

2.2 Dataset classes defined by some criteria on properties of the datasets (e.g., use all datasets compliant with a given ontology or that were produced by a given set of sources).

2.3 Data accessibility restrictions, including:

2.3.1 Temporal windows (e.g., start-time, end-time).

2.3.2 Spatial windows (e.g., within a given zip code).

2.3.3 Security/privacy authorizations.

The data classes and data-access restrictions can be used as a subquery to dynamically select the datasets for use in answering the primary query.

2.4 The dataset preprocessing specifications.

2.5 Query premises (Fikes, Hayes, and Horrocks 2003): these enable *"if…then…"* queries, such as *"if Muffet-Muktbar is an alias for Shirin-Khatami then what data now matches pattern Social-Network-P212?"*

3. Module(s) to be used for answering the query

3.1 The module(s), including submodule(s), to be used, for example, the LAW tool using the GEM pattern matcher (Thomere et al. 2004).

3.2 Parameter settings that configure and control the behavior of the search module; these might include:

3.2.1 Support for a query-specific search plan that indicates, for example, the order in which to look for variable bindings.

3.2.2 A function used to measure match quality.

3.2.3 Query-answer termination parameters:

3.2.3.1 A maximum number of matches to return in total and/or how many to return in any one time frame.

3.2.3.2 A time cutoff: the latest time by which to send results, for example, return results in 10 minutes, or return results by March 13th 2005, 0700EST.

3.2.3.3 A search-state cutoff: the maximum number of search states that can be considered.

3.2.3.4 A search-time cutoff: the maximum runtime allowed for finding matches.

3.2.3.5 A search-cost cutoff: the maximum total pattern match cost permitted.

3.2.3.6 A confidence cutoff: the maximum uncertainty permitted.

3.3 A specification of the answer format (see below).

4. Answer format(s): some alternatives

4.1 The answer data is organized as a two-dimensional table, such that each column corresponds to one pattern variable and each row (aka a "binding vector") corresponds to one match of the pattern (e.g., see SPARQL).

4.2 The answer data is organized as a list or set of "matches," each having the same structure as the query pattern, with variables replaced by their bindings.

4.3 The answer data is organized as a list or set of a specified formula (e.g., an "answer pattern") referencing variables from the query pattern (Fikes, Hayes, and Horrocks 2003).

4.4 Inclusion or exclusion conditions for binding vectors (e.g., that define when bindings are deemed redundant and omitted from answer).

4.5 A sorting predicate by which to order the binding vectors that compose the answer.

4.6 The answer data includes meta-data beyond the bindings of pattern variables, such as

4.6.1 A (typically module-specific) match score.

4.6.2 Data describing the certainty of individual or sets of bindings.

4.6.3 Explanations or derivations of bindings.

4.6.4 The provenance of data and rules used to derive the bindings (e.g., source, authoritativeness, see Pinheiro da Silva, McGuinness, and Fikes 2004).

4.6.5 Properties of the datasets accessed (e.g., completeness or correctness estimates, ontology, owner).

4.7 Batching or streaming the answer data.

5. Maintaining and organizing hypotheses

5.1 Queries and answers and hypotheses can be aggregated into related collections (e.g., that share meta-data, such as author or urgency or ontology).

5.2 Queries and answers and hypotheses may each have associated meta-data, such as

5.2.1 Ontology

5.2.2 Inception and modification attribution: date, author, project, comment, and version number.

5.2.3 A serialized handle (e.g., URI) allowing hypotheses to be archived, retrieved and distributed as collections or individually.

5.2.4 Access control information.

5.2.5 The hypotheses in which a query (or answer or pattern or module) has appeared; the structured arguments (see below) in which each hypothesis has appeared as evidence.

5.2.6 Performance of the hypothesis; e.g., cost, or correctness on test data with known ground truth or with post hoc evidence from further investigation of the hypothesis.

5.2.7 Hypotheses have associated degree of belief, at both gross hypothesis level and pattern variable binding level. This supports inexact pattern matching by modules, for example, as embodied in SRI's LAW system (Wolverton et al. 2005, Ruspini, Thomere, and Wolverton 2004).

5.2.8 There can be annotations at any level of a hypothesis. Annotations can be used, for example, to record agent (tool or user) comments.

5.3 Hierarchical hypotheses

5.3.1 Hypotheses (queries) can be made up of subhypotheses (subqueries), which could have been generated by different modules over different data sources. Different, nonintegrated data sources can occur in practice, where the datasets have different information-theoretic properties (Tierno 2005, Fournell and Tierno 2005); the query can be broken into pieces, each piece running against one of the nonintegrated datasets and corresponding to a subquery with corresponding subhypotheses.

5.3.2 Hypotheses can be used as input to modules to perform queries on previous query results. For instance, a module may be able to answer only a part of the original query; that is, it may find bindings for variables in some subsets of the query formula literals and return those as well as the unanswered formula literals so that some other module (or some other data, perhaps becoming available later in time) may be used to complete the answer (e.g., KSL's JTP system; see Fikes, Frank, and Jenkins 2003).

5.4 Hypotheses can be extended or modified (e.g., getting additional bindings as new data becomes available, extending or correcting providence information as warranted by the evolving context).

5.5 Hypotheses can have variable bindings unbound, that is, to support unanswered parts of a query.

5.6 Syntax and presentation

5.6.1 Human-friendly: for example, a text-based or graph-based form of the hypothesis language, which can be read and written by human users of the language, as well as by machines. This may require representing presentation-specific properties (e.g., node position, edge color).

5.6.2 Network-friendly: for example, an XML-based syntax for sharing over the network.

Pattern and Hypothesis Interlingua for Sharing and Display

PHERL is being developed as an interlingua to support communication among tools that include patterns and hypotheses among their inputs and outputs. One motivation for a common pattern representation language is that patterns developed by or for one tool can be used by other tools. A second motivation is that a single presentation can be supported for display to the user, for pattern browsing and editing. This enables the user to learn a single pattern display and editing interface rather than having to develop proficiency with each tool's idiosyncratic display and interface. Supporting PHERL as a pattern interlingua requires developing translators to and from the PHERL and the native pattern representation languages of the tools, such as GEM (Thomere et al. 2004, Wolverton et al. 2005) and qGraph (Blau, Immerman, and Jensen 2002). Preliminary (partial) translators between the pattern representation of PHERL and representations used by several tools developed in our toolset community have been developed.

Representing Evidence

The third representation tier in support of link analysis is the evidential tier. This tier captures how link analysis hypotheses can be merged together or related in various ways, for example, to support the evidence relevant to some particular case as it evolves over time, to combine both competing and supporting evidence within a structured argument in support of a range of intelligence needs, from the relatively low-level tasks of tracking scenarios to relatively high-level tasks of developing policy based on the best current interpretations of a complex set of factors and hypotheses. Our development of this tier is less mature than the prior two, and we present some preliminary requirements.

Representation Requirements

1. Hypotheses, as well as being an output in their own right, can be used as evidence within a higher-level hypothesis (hierarchical hypotheses); for example, SRI's structured argumentation tool, SEAS (Lowrance et al. 2001), can use low-level pattern match hypotheses from SRI's link discovery tool, LAW, as evidence.

2. Groups of hypotheses within evidence structures can be annotated with relationships and dependencies (e.g., as providing supporting or opposing evidence for some particular outcome or conclusion, or as sets of competing, alternative hypotheses for describing a situation, or as the next group in a chronological series of hypotheses).

3. Evidence structures can be annotated with meta-data:

3.1 Inception and extension or modifications attributions (e.g., time, date, author, project or RFI, comment, version number).

3.2 Urgency, maturity of the threat or policy issue.

3.3 Access restrictions and an access log of who has reviewed the evidence structure and when.

4. Evidence structures can be aggregated into containers (e.g., related collections that share meta-data).

5. Evidence structures and their hypotheses can be annotated with triggers for data conditions that warrant reconsideration or further consideration (e.g., triggering an RFI from an intelligence consumer).

Conclusions

Previous work in link analysis has developed interchange languages for patterns and, to a lesser extent, hypotheses (Harrison 2002, Blau et al. 2002). However, to date no work has looked at representing all three tiers of possible interchange between link analysis tools, namely, at the pattern, hypothesis, and evidence levels. It is our contention that one must consider all three tiers and how tools might coordinate with one another (e.g., in a web-services oriented environment) in order to develop a rich, extensible, and flexible interlingua for link analysis. Supporting only one tier (e.g., patterns) might facilitate the ability to use a single visualization tool (e.g., Analyst's Notebook), but it does not address the broader issues of sharing and reusing hypotheses derived by different tools.

Today, analysts must manually manipulate output from one tool to another, or alternatively develop specialized direct tool-to-tool translators. A common link analysis interlingua, such as PHERL, would eliminate this requirement, and allow a suite of specialized tools to be brought to bear on the challenging problems faced by the intelligence community.

Acknowledgments

This research was supported under Air Force Research Laboratory (AFRL) contract number F30602-01-C-0193. Peter Yeh provided valuable comments on this work.

References

Analyst's Notebook, http://www.i2.co.uk/Products/Analysts_Notebook/default.asp

Blau, H., Immerman, N., and Jensen, D. 2002. A Visual Language for Querying and Updating Graphs. University of Massachusetts Amherst Computer Science Technical Report 2002-037.

Chaudhri, V., Murray, K., Pacheco, J., Clark, P., Porter, B., and Hayes, P. 2004. Graph-based Acquisition of Expressive Knowledge. In Proceedings of the European Knowledge Acquisition Workshop (EKAW'04).

Clark, P. and Porter, B. 2004. KM 2.0: Users Manual. http://www.cs.utexas.edu/users/mfkb/RKF/km.html.

Clark, P., Thompson, J., Barker, K., Porter, B., Chaudhri, V., Rodriguez, A., Thomere, J., Mishra, S., Gil, Y., Hayes, P., and Reichherzer, T. 2001. Knowledge Entry as the Graphical Assembly of Components. In Proceedings of the First International Conference on Knowledge Capture (K-Cap'01).

Consens, M., Mendelzon, A., and Ryman, A. 1994. Looking at the Relations Among Software Objects, Technical Report November 10, 1994, Department of Computer Science, University of Toronto.

Eigler, F. 1994. Translating GraphLog to SQL. In Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research.

Fikes, R., Frank, G., and Jenkins, J. 2003. JTP: A System Architecture and Component Library for Hybrid Reasoning. KSL Report 03-01, Stanford University. http://ksl.stanford.edu/KSL_Abstracts/KSL-03-01.html.

Fikes, R., Hayes, P., and Horrocks, I. 2003. OWL-QL: A Language for Deductive Query Answering on the Semantic Web. KSL Report 03-14. Stanford University. http://ksl.stanford.edu/KSL_Abstracts/KSL-03-14.html

Fournelle, C. and Tierno, J. 2005. Link Analysis Technologies for Differing Data Sources. In Proceedings of the AAAI Spring Symposium on AI Technologies for Homeland Security.

Harrison, I. 2002. PatternML Schema Design http://www.ai.sri.com/~law/schemas/2002/07/patternML.htm

Lowrance, J., Harrison, I., and Rodriguez, A. (2001). Capturing Analytic Thought. In Proceedings of the First International Conference on Knowledge Capture.

Pinheiro da Silva, P., McGuinness, D. L., and Fikes, R. E. 2004. A Proof Markup Language for Semantic Web Services. KSL Report 04-01. Stanford University.

Pool, M., Murray, K., Fitzgerald, J., Mehrotra, M., Blythe, J., Kim, J., Chalupsky, H., Miraglia, P., Schneider, D., Schrag, R., and Russ, T. 2003. Evaluating SME-Authored COA Critiquing Knowledge. In Proceedings of the Second International Conference on Knowledge Capture (K-Cap'03).

Ruspini, E., Thomere, J., and Wolverton, M. 2004. Database Editing Metrics for Pattern Matching. In Proceedings of the IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety (CIHSPS-04).

SPARQL Query Language For RDF. W3C Working Draft http://www.w3.org/TR/rdf-sparql-query/

Thomere, J., Harrison, I., Lowrance, J., Rodriguez, A., Ruspini, E., and Wolverton, M. 2004. Helping Intelligence Analysts Detect Threats in Overflowing, Changing and Incomplete Information. In Proceedings of the IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety.

Tierno, J. 2005. Performance Analysis and Prediction for Data Mining Systems. In Proceedings of the AAAI Spring Symposium on AI Technologies for Homeland Security.

Wolverton, M., Harrison, I., Lowrance, J., Rodriguez, A., and Thomere, J. 2005. Supporting the Pattern Development Cycle in Intelligence Gathering. In Proceedings of the First International Conference on Intelligence Analysis (IA-05).

Wolverton, M. and Thomere, J. 2005. The Role of Higher-Order Constructs in the Inexact Matching of Semantic Graphs. Proceedings of the AAAI Workshop on Link Analysis.

Yeh, P., Porter, B., and Barker, K. 2003. Using Transformations to Improve Semantic Matching. In Proceedings of the Second International Conference on Knowledge Capture (K-Cap'03).