Report
On
AllegroGraph RDFStore

Submitted
By

Ruku Roychowdhury
Shagun Kariwala

CSC 8711
Database and the Web
Spring 2011

Table of content

Introduction :-

Semantic web technology aim to build websites with sufficient self describing data so that computers can browse through them as easily humans can do.Allegrograph is a framework to develop semantic web applications.It is a high performance Graph Database to store this data and metadata in triple format. Allegrograph provides various query APIs to perform query on these triples. It also has its own built in RDFS++ reasoner but it is not complete.Allegrograph can be integrated to other powerful GUI based reasoner if more elaborated reasoning is required.

Installation :-

Allegrograph runs as a service. We have to install the server first and start the service. The latest version of Allegrograph Server, Allegrograph Server 4 runs natively on Linux x86-64 bit machine. It is advisable to to set up a Linux Virtual machine if we want to run it on other Operating Systems (Windows or MAC). Allegrograph 3.3 can run on 64-bit Mac, Windows, and Solaris, and all 32-bit systems.

Allegrograph server can work with a large variety of programming interfaces like Java, Common Lisp and Prolog to name a few. Future editions of Allegrograph will also have interfaces for C# and Ruby. In this project, we have explored various features of the Java API to create and operate on triple strores.

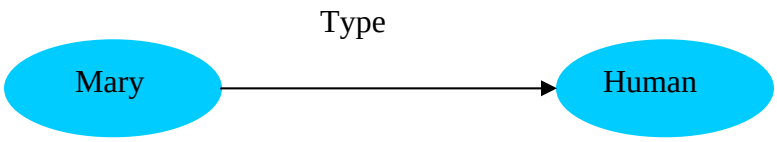
Allegrograph also provides a GUI based browser called “GRUFF” which is very easy to use. The details about it are discussed later.

Allegrograph Features :-

Allegrograph is a database for triple store.A triple can simply be described as three URIs. It can be easily broken down into three parts.

- subject
- predicate
- object

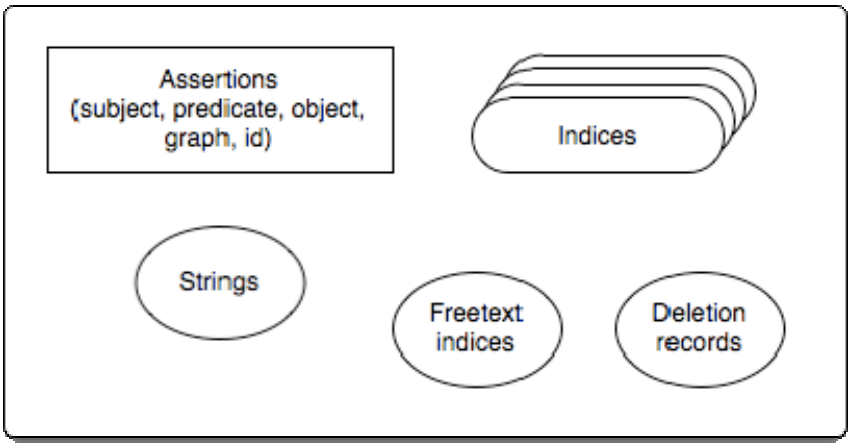
As for example,



In the above mentioned example, the subject object and predicate can be classified as follows:-

subject	**predicate**	**object**
Mary	Type	Human

Allegrograph data store adds one more piece of information named Graph to the standard set of subject, predicate and object. Graph is used to mention some additional useful information about the triple such as the context of the triple. Graph is useful for fast if we break down the graph data store into subgraphs. Allegrograph also appends an unique id with every triple.



Logical Structure of an AllegroGraph triple-store

Therefore, AllegroGraph actually stores quints. A triple in AllegroGraph contains 5 slots, the first three being subject (s), predicate (p), and object (o). The remaining two are a named-graph slot (g) and a unique id assigned by AllegroGraph. The id slot is used for internal administrative purposes, but can also be referred to by other triples directly.

The W3C proposal is to use the 'named-graph' slot for clustering triples. So for example, when loading a file with triples into AllegroGraph filename is used as the named-graph. This way, if there are changes to the triple file, user just has to update those triples in the named graph that came from the original file.

Federation:-

AllegroGraph supports queries with distributed databases. Multiple triple-stores, both local and remote can be grouped into a single virtual store. It allows thread-safe opening of multiple triple-databases from one application (for the read only parts of the database). Queries over multiple databases are easy with direct data access from applications. It also supports physical merging of databases.

Encoded Strings:-

All these are unique strings and it will be inefficient to store all the duplicated strings. Allegrograph assigns a 12 byte unique identifier called a Unique Part Identifier(UPI) to each unique string. The String Dictionary manages the association between the strings and UPI and prevent duplication.These UPI also saves the time and IO overhead for string retrieval.

Triple Indices:-

AllegroGraph employs a set of sorted indices to search a contiguous block of triples that are likely to match a specific query pattern.
These indices are identified by names that describe their organization. The default set of indices are called **spogi, posgi, ospgi, gspoi, gposi, gospi**, and **i**, where:

- S stands for the subject URI.
- P stands for the predicate URI.
- O stands for the object URI or literal.
- G stands for the graph URI.

- I stands for the triple identifier (its unique id number within the triple store).

SPOGI Index

The order of the letters denotes how the index has been sorted. For example, the **spogi** index contains all of the triples in the store, sorted first by subject, then by predicate, then by object, and finally by graph. The triple id number is present as a fifth column in the index. When we run a SPARQL query to find triples with a specific subject value, the **spogi** index find all of the triples with that subject value as a block. This block is further sorted by predicate, so if the predicate value is known we can immediately narrow the potential matches to a small range of triples.

POSGI Index

To find unknown subjects that have a specific predicate and object value, AllegroGraph uses the **posgi** index. All triples that have the same predicate are located together in that index, and are then sorted by object value.

OSPGI Index

If only the object value is known, we can use the **ospgi** index. It is organized by object, and from it we can rapidly retrieve the corresponding subject and predicate.

Graph Indices

The graph indices, **gspoi, gposi,** and **gospi**, are used when the triple store is divided into subgraphs. If the subgraph value is known then we can immediately take out all of its triples and then index by subject, predicate or object as needed.

I Index

The **i** index is special. It is simply a list of all triples sorted by id number. Its primary purpose is to make triple deletion fast when triples are deleted by id. The id alone is sufficient to identify the subject, predicate, object and graph values that let AllegroGraph delete the same triple from the other indices. Without the **i** index, AllegroGraph has to scan the other indices line-by-line to find the matching id numbers. This is very slow.
Triples can also be deleted by pattern-matching instead of id. That type of deletion is not influenced by the **i** index.

The standard seven indices are enabled when a triple store is created. User can customize this set, both by eliminating indices that his/her application will not use, and by requesting custom indices that match his/her more unusual triple patterns.

- For example if his/her application does not use subgraphs, the indices beginning with "g" will never be used. User can speed up indexing dramatically by eliminating these indices from the system.
- As a second example, if the user have a pattern that asks what graphs contain resources that are blue, he/shemight request a **pogsi** index which the system does not normally provide. If the user knows that the predicate is "color" and the object is "blue," then the **g** column of the **pogsi** index will contain a block of graph URIs that can be returned in one operation to match this triple pattern.

Java Programming API:-

Allegrograph provides a very powerful Java API to interact with the triple store. It provides built in methods for:-

- **Creating a triple store**
  Method Name: renew()
  Create an instance that accesses an AllegroGraph triple store. If the triple store exists, it is deleted and replaced with a new empty triple store.
  Triple stores are created with the methods access(), create(), open(), renew(), and replace().
  Parameters:
  Name: the name of the triple store or a pathname string that specifies the name and location of the triple store. If the name argument is a pathname string then the directory argument must be null.
  Directory: the directory where the data base does or will reside. If this argument is a non-null string, then the name argument must be a simple name.
  If the operation implies the deletion of a triple store, the entire directory is deleted.

- **Bulk Load the triple store from any RDF/NTriple file**
  Method Name: loadNtriples()
  Load a file of triple declarations. The triples are added in the null context of the triple store.
  Parameters:
  Name: A string that specifies the source of the triple declarations. This can be a file pathname or a file URI or a web URI. The data must be in the Ntriples format expected by the AllegroGraph server. All file pathnames are relative to the pathname defaults in the server. The file environment of the client application is irrelevant.

- **Add triples to the triple store**
  Method Name: addStatement()
  Add a Statement in the null context to the triple store.
  Parameters:
  Subject: Specifies subject of the triple
  Predicate: Specifies predicate of the triple
  Object: Specifies object of the triple

- **Retrieve all triples from triple store**
  Method Name:getStatements()
  Retrieve null context/graph statements from the triple store if we provide null for every parameter.
  Parameters:
  subject
  predicate
  object
- Retrieve a set of triples based on some condition
  getStatements()
  Retrieve all statements from the triple store which matches the subject, predicate,object values passedin the parameters.
  Parameters:
  Subject: Specifies subject of the triple we want to retrieve
  Predicate: Specifies predicate of the triple we want to retrieve
  Object: Specifies object of the triple we want to retrieve

- **Delete triples from triple store**
  Method Name: removeStatement()
  Remove all statement that match a pattern in the null context of the triple store.

Parameters:
s the subject pattern (Value, UPI, string, or array of same)
p the predicate pattern (Value, UPI, string, or array of same)
o the object pattern (Value, UPI, string, or array of same)
The s, p, or o, arguments may also be null or an empty string to denote a wild card.
Returns:
The number of triples deleted.

- **Close triple store**
  Method Name: closeTripleStore()
  Close an AllegroGraph instance. Since many other users may be using this same triple store, closing the AllegroGraph instance does not actually close the triple store but only synchronizes the store and invalidates this access instance unless this is the only instance accessing the store.

- **Psedocode for Querying Triple Store:-**

```
String query =
"SELECT ?s ?p ?o WHERE { ?s ?p ?o } ";

// Query the store and show the results
SPARQLQuery sq = new SPARQLQuery();
sq.setTripleStore(ts);
sq.setQuery(query);
AGSparqlSelect.doSparqlSelect(sq);
```

Gruff

Gruff is an interactive triple-store browser, query manager, and editor that is built on AllegroGraph. Information can be browsed as visual graphs of nodes and link lines that are layed out automatically, and also as tables of properties for particular nodes. Queries can be written textually as SPARQL or Prolog code, or designed graphically as diagrams of nodes and link lines. Data can be created and edited by filling in tables of property values. The various views and tools are tightly integrated to facility rapid browsing, querying, and editing.

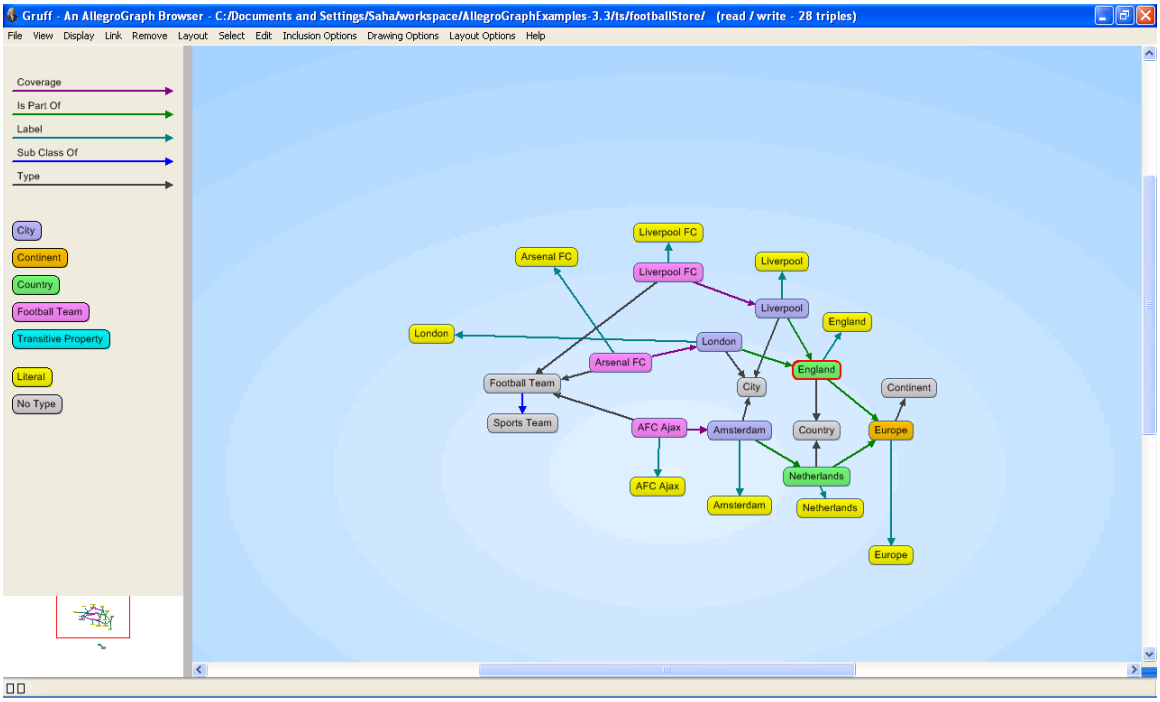


Fig 1: Graphical view of GRUFF

The graph view displays a "visual graph" of a subset of the nodes and links that are in the store that you are currently browsing as shown in Fig. 1. The nodes are automatically arranged (or "layed out") to make the relationships readable. A node in the graph view is a labeled box that represents a resource or literal in the store. A link is a straight line segment between two nodes that represents one or more of the triples that link those nodes.

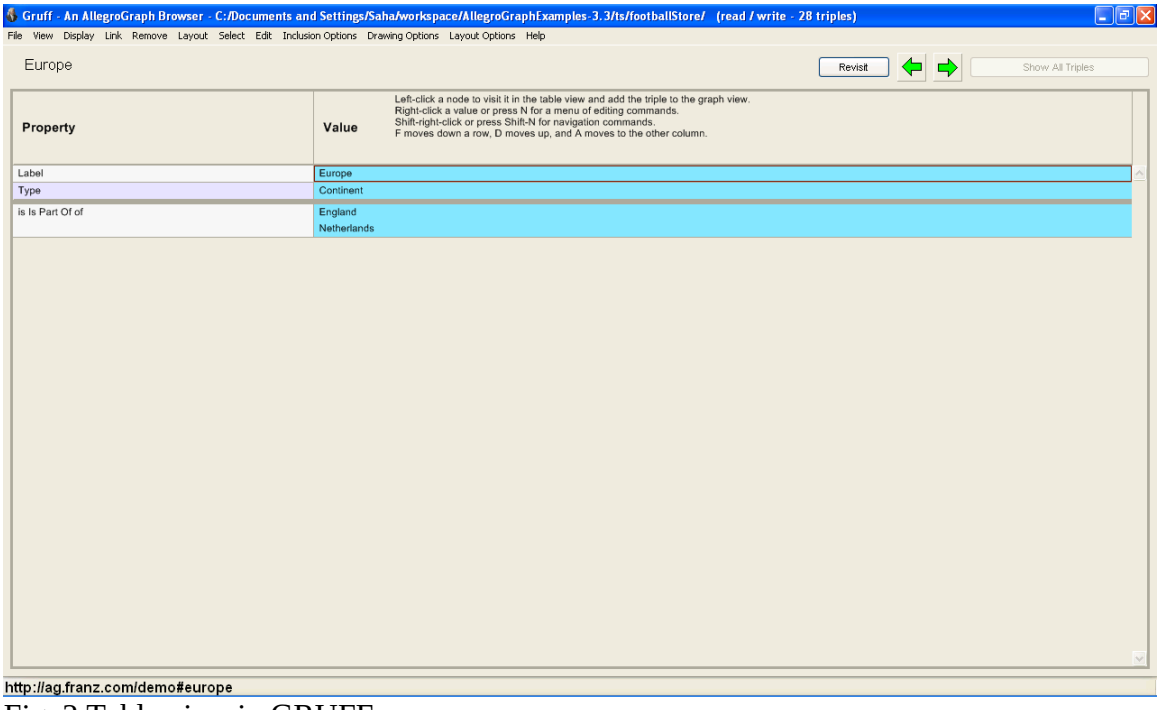


Fig. 2 Table view in GRUFF

The table view displays a table of all of the properties of a single node, which is known as "the displayed node" as shown in Fig. 2. You can browse from that node to linked

nodes in the usual hyperlink way, and edit the property values. Each row of the table represents a single triple that's in the store.

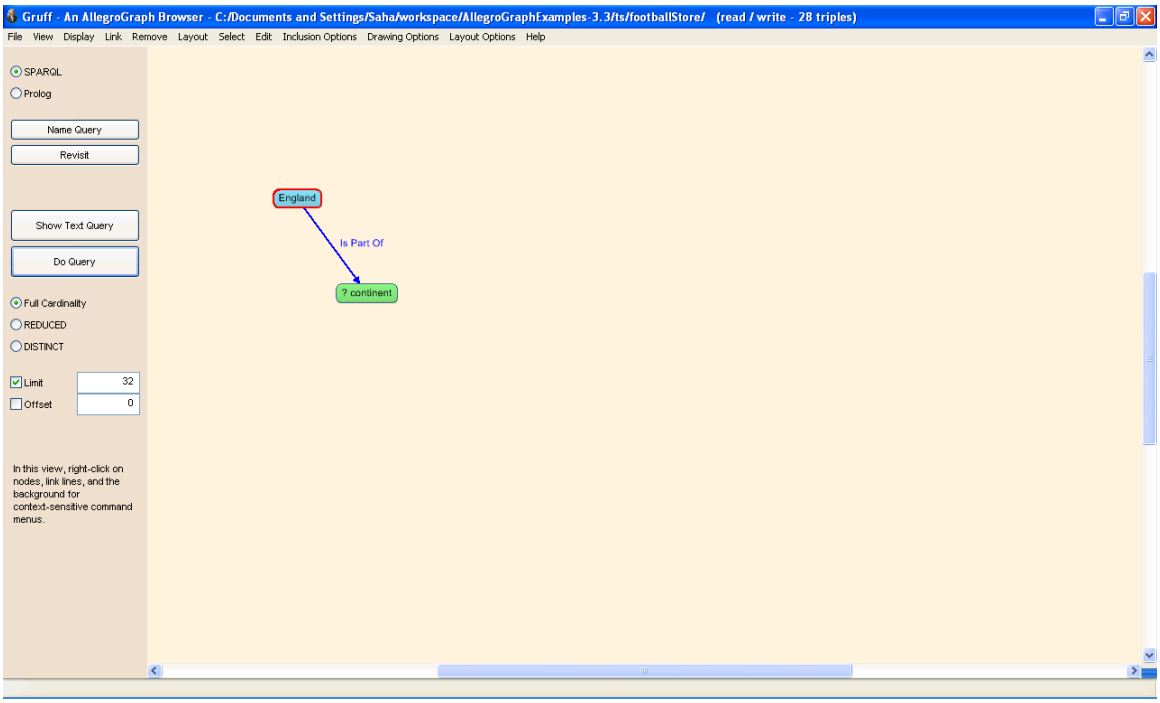


Fig. 3 Graphical Query view in GRUFF

The query view displays a view where you can do a SPARQL or Prolog query (as shown in Fig. 3 and see the results in a table. Nodes in the results table can then be viewed in detail in the table view, or added as nodes to the graph view. To do a query, first decide whether you want to do a SPARQL query or a Prolog query, and select the corresponding radio button at the upper left.

The graphical query view allows devising a query "visually" as a diagram. This is done by arranging node boxes and link lines that represent triple patterns in the query, where the triples patterns can contain variables as well as actual objects that are in the store. General and specialized filters can be specified as well. A SPARQL or Prolog query can then be automatically generated from the diagram and executed as usual in the query view.

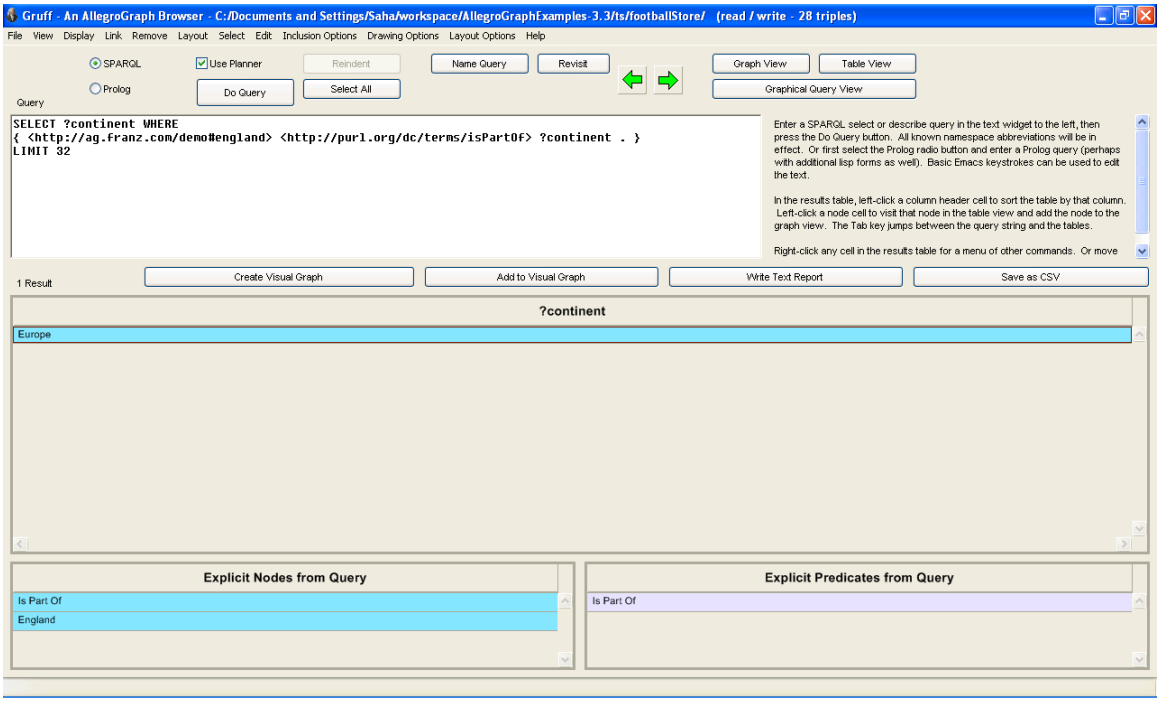


Fig. 4 Result returned by SPARQL query in GRUFF

It generates the text query as can be seen in Fig.4 in the top text area.

Recent Projects in Allegrograph:-

Allegrograph is used in many commercial, Open Source, Defense projects as a datastore. Some of the significant ones are:-

1. **DBPedia Germany**

DBpedia is a community effort to extract structured information from Wikipedia and to make this information available on the Web as Linked Data. DBPedia Deutschland is the German part of DBPedia. It extracts structured information from Wikipedia (Deutschland) and makes this information available on the Linked Date Web. DBPedia Deutschland is a joint project of the research group AKSW, lead by Dr. Soeren Auer at the University of Leipzig, and the research group Corporate Semantic Web, lead by Prof. Dr. Adrian Paschke at the Freie Universitaet Berlin.It comprises over 100 million facts extracted from Wikipedia Deutschland. This allows machines answering questions, such as "Which impressionists are born in Berlin?" or "Which chemical elements are contained in the periodic table?" DBPedia Deutschland runs an AllegroGraph triple store.

2. **GenomeWeb**

This project was an initiative of Pfizer to serve as a "real-world example" of semantic technology in the pharmacy setting.

3. **TweetLogic**

TweetLogic is a project to bring the Semantic Web concept to Tweeter Data. Allegrograph is also the storage component for this project.

Conclusion:-

AllegroGraph uses disk-based storage, enabling it to scale to billions of triples while maintaining high performance through indices. AllegroGraph supports SPARQL, RDFS++, and Prolog reasoning from Java applications. It has wide range of programming APIs to work with many different languages like Java, Python, Lisp, Prolog to name a few. Visual editors like GRUFF makes it easy for beginners to interact with the tool and makes data retrieval more pleasant.

References:-

- http://www.franz.com/agraph/services/conferences_seminars/Franz_Webinar_6-12-08.pdf
- http://www.franz.com/agraph/allegrograph/doc/server-installation.html -
- http://www.franz.com/agraph/support/documentation/v4/java-tutorial/java-tutorial-40.html
- http://www.franz.com/agraph/support/learning/
- http://www.franz.com/agraph/allegrograph/doc/learning/Building-in-Eclipse.html
- http://www.franz.com/agraph/allegrograph/doc/learning/index.html
- http://en.wikipedia.org/wiki/AllegroGraph
- http://www.iscb.org/cms_addon/conferences/cshals2009/presentations/GudivadaCFeb09.pdf
- http://jp.franz.com/base/seminar/2006-07-24/AllegroGraph-presentation-July06.pdf
- http://www.semantic-web-journal.net/content/new-submission-racerpro-knowledge-representation-and-reasoning-system
- http://www.snee.com/bobdc.blog/2009/04/getting-started-with-allegrogr.html