OBO format generator

1. Introduction

This document describes how to setup the java files for generating an ontology file in OBO format from a Unified Medical Language System (UMLS) database (MySQL) installation. The OBO file can be used as an ontology for text-mining purposes.

2. OBO format

The Open Biomedical Ontologies (OBO) format was proposed by the OBO Foundry to create controlled vocabularies for use across different biological and medical domains. It is maintained by the National Center for Biomedical Ontology (www.bioontology.org).

The OBO format is an ontology representation language. The format is similar to the tag-value format of the Gene Ontology definitions file, and has important features such as human readability, ease of parsing, extensibility, and minimal redundancy.

The OBO format structure consists of a header section where header tags are defined. The remaining part contains type definitions and terms. Each term has a unique identifier, name, namespace, and a definition (optional), and may have zero or more typed relationships with other terms. A term may have multiple synonyms. Each synonym has a type, a scope, and one or more links to its original resources. More details of the OBO format can be found here:
http://oboformat.googlecode.com/svn/trunk/doc/GO.format.obo-1_4.html

To distinguish between terms that belong to semantic groups, semantic types, and concepts in the UMLS, we defined three namespaces:

- umls_semantic_group: Terms that belong to the UMLS semantic groups (such as Anatomy, Disorders, Objects, etc.).
- umls_semantic_types: Terms that belong to the UMLS semantic types (such as Food, Enzyme, Vitamin, etc.). Terms within this group are organized hierarchically.
- umls_term: Terms that belong to UMLS concepts and are grouped by their CUIs.

Semantic information can be found at the following URL:
http://semanticnetwork.nlm.nih.gov/SemGroups/SemGroups.txt

To generate the initial terminology in OBO format, we made use of the OBO format library (https://code.google.com/p/oboformat/). This library provides functions for manipulating OBO Documents.

3. Terminology generation

Download the UMLS version 2014AA (or newer) and store its content in a local MySQL database. UMLS can be downloaded at the following link: http://www.nlm.nih.gov/research/umls/licensedcontent/umlsknowledgesources.html

Using the OBO format library, create an empty instance of an OBODoc object, which is then filled with header information, UMLS semantic type and group definitions, and all the concepts and terms.

3.1. Generating the header

The OBO header contains meta-data such as creation date, format version, and synonym type definitions. OBO files can be in multiple languages for preferred terms and for synonyms. For example, the two synonym types for English are defined as "PREF_ENG" and "SYN_ENG".

3.2 Generating the semantic types and groups

We defined a relation type "has_semantic_group" to indicate the relationships between semantic types and semantic groups. Data about the semantic types and groups were gathered from http://semanticnetwork.nlm.nih.gov/SemGroups/SemGroups.txt. Information about the relationships between the semantic types is available in table STRTRE1 of the UMLS2014AA release. The hierarchical structure between semantic types was extracted with the following query:

```
SELECT * from SRSTRE1 WHERE UI2 = 'T186'
```

in which "T186" denotes the "is-a" relationships. The java code (function 'addSemanticGroupandType') is found in OBOData.java.

3.3. Generating the concepts and terms

We used the guidelines provided by the NLM (http://www.nlm.nih.gov/research/umls/implementation_resources/query_diagrams/er1.html) to find all information associated with a particular UMLS concept (CUI). In particular, we used information from the following UMLS tables: MRCONSO (preferred terms and synonyms), MRDEF (concept definitions), MRSTY (semantic types of concepts), and MRREL (concept relationships). We filter out terms with less than two or more than 99 characters. We collect all concepts with English preferred terms that are not suppressible (terms in the UMLS are marked "suppressible" if they are obsolete or considered of low quality):

```
SELECT * from MRCONSO WHERE LAT ='ENG' and SUPPRESS ='N' and ispref='Y' and TS='P' order by CUI
```

in which LAT is the language and TS is the term status.

We then performed for each concept the following queries to collect information of that concept.

1. Find all definitions related to a given concept:

SELECT * FROM MRDEF WHERE cui = '<CUI>'

2. Find all semantic types related to a given concept:

SELECT * FROM MRSTY WHERE cui = '<CUI>'

We defined a relation type "has_semantic_type" to indicate relate concepts to their corresponding semantic types. The three above MySQL queries are found in the function 'init()' in the DataProvider.java.

3. Find all non-suppressible synonyms related to a given concept:

SELECT * from MRCONSO WHERE cui= '<CUI>' and LAT in ('ENG','GER','DUT') AND SUPPRESS='N'

4. Find "is_a" relationships associated with a given concept:

SELECT * from MRREL WHERE cui1='<CUI>' and REL='PAR'

in which REL indicates the type of the relationship, in this case PAR (parent, or is_a).

To remove possible cyclic relationships, we recursively checked for each concept its parent concepts up in the hierarchy. If the given concept and an ancestor have the same CUI, a cyclic relationship is present. To break the cycle, the is_a relationship with the parent of the given concept is automatically removed (RemoveCycle.java).

Using all the retrieved information, the terminology in OBO format can be generated.

4. Generating the OBO file

1. Unzip the folder containing all the java files and jar files. The three main java files are DataProvider.java, OBOData.java, and RemoveCycle.java.
2. The three java files depends on a two jar files (included in the zip file): Oboformat-all.jar and the mysql connector mysql-connector-java-5.1.23-bin.jar.
3. In the file DataProvider.java, set 'DriverManager.getConnection' (line 51) in class init() to the correct UMLS configuration (make sure the UTF8 field is set correctly!).
4. In the other two files, OBOData and RemoveCycle,on the top there are global variables that have to be set to the correct path to run the java files.
5. In public class DataProvider() (line 40), specify which languages to include (e.g., langMap.put("ENG", "en");). One line per language.
6. Run OBOData.java to create the OBO file. We advise a minimal amount of 8Gb or RAM to run this script.

Acknowledgment

This software was developed as part of the SEMCARE project (grant no. 611388) under the European Union's Seventh Framework Programme for research, technological development and demonstration activities.