Transforming the EFO Inferred Ontology for MetaMap

Willie Rogers

January 14, 2014

1 Purpose

The purpose of this document is to provide an example of the processing necessary to allow for entity recognition of Resource Description Format (RDF) concepts, in this case Experimental Factor Ontology (EFO) concepts, using MetaMap.

2 Finding the concepts and synonyms using SPARQL queries

Each SPARQL query first finds the node containing the concept identifier in this case the node with the isa relation (rdf:type) "owl:Class" if the same node has a rdfs:label (preferred name) then add the subject and object to the result graph:

```
SELECT ?s ?o { ?s rdf:type owl:Class . ?s rdfs:label ?o }
```

Similarly, the following SPARQL query does the same for owl:Class instances which also have efo:alternative terms:

```
SELECT ?s ?o { ?s rdf:type owl:Class . ?s efo:alternative_term ?o }
```

We have one graph for concepts and another for synonyms. Each graph is indexed in a dictionary by the identifier, one dictionary for concepts and one for synonyms.

3 Generating concept and source identifiers from URIs

The function `abbrev_uri` is used to produce a concept identifier from the Uniform Resource Identifier (URI) of the concept in a form that is usuable by MetaMap.

```
http://www.ebi.ac.uk/efo/EFO_0003549 -> EFO_0003549

http://www.ebi.ac.uk/chebi/searchId.do?chebiId=CHEBI:153671 ->  CHEBI_153671

http://purl.org/obo/owl/CL#CL_0001003 -> CL_0001003
```

4 Generating the UMLS format tables (RRF)

When producing the Concept Names and sources File, MRCONSO.RRF, all of the keys in the concept dictionary are traversed. For each concept record for the label and, if present, any synonyms are produced. For example, the concept EFO_0003549 "caudal tuberculum" has the following MRCONSO.RRF records generated:

```
$ grep EFO_0003549 MRCONSO.RRF
EFO_0003549|ENG|P|L00004389|PF|S00015577|Y|A00015591||||EFO|PT|http://www.ebi.ac.uk/efo/EFO_000
EFO_0003549|ENG|S|L00015225|SY|S00019475|Y|A00019506||||EFO|PT|http://www.ebi.ac.uk/efo/EFO_000
EFO_0003549|ENG|S|L00015226|SY|S00019476|Y|A00019507||||EFO|PT|http://www.ebi.ac.uk/efo/EFO_000
```

The Semantic Types file, MRSTY.RRF, has one record for each semantic type assigned to a concept:

```
$ grep EFO_0003549 MRSTY.RRF
EFO_0003549|T205|A0.0.0.0.0.0|Unknown|AT0000000||
```

The Source Information file, MRSAB, contains information used by MetaMap when restricting to a particular set of sources, or excluding them.

```
$ cat MRSAB.RRF
C4000000|C4000000|GO|GO|http://purl.org/obo/owl/GO#|||||||0|3|2||||ENG|ascii|Y|Y||
C4000001|C4000001|EFO|EFO|http://www.ebi.ac.uk/efo/|||||||0|5222|3501||||ENG|ascii|Y|Y||
C4000002|C4000002|UO|UO|http://purl.org/obo/owl/UO#|||||||0|135|76||||ENG|ascii|Y|Y||
C4000003|C4000003|NCBITaxon|NCBITaxon|http://purl.org/obo/owl/NCBITaxon#|||||||0|77|46||||ENG|
C4000004|C4000004|SPAN|SPAN|http://www.ifomis.org/bfo/1.1/span#|||||||0|1|1||||ENG|ascii|Y|Y||
C4000005|C4000005|CL|CL|http://purl.org/obo/owl/CL#|||||||0|713|617||||ENG|ascii|Y|Y||
C4000006|C4000006|oboInOwl|oboInOwl|http://www.geneontology.org/formats/oboInOwl#|||||||0|1|1|
C4000007|C4000007|OBO|OBO|http://purl.obolibrary.org/obo/|||||||0|68|42||||ENG|ascii|Y|Y||
C4000008|C4000008|PATO|PATO|http://purl.org/obo/owl/PATO|||||||0|5|3||||ENG|ascii|Y|Y||
C4000009|C4000009|SNAP|SNAP|http://www.ifomis.org/bfo/1.1/snap#|||||||0|8|7||||ENG|ascii|Y|Y||
C4000010|C4000010|CHEBI|CHEBI|http://www.ebi.ac.uk/chebi/searchId.do?chebiId=|||||||0|0|0||||E
```

The Concept Name Ranking File, MRRANK.RRF, provides information to the filtering program `filter_mrconso` needed when determining which records in MRCONSO.RRF should be removed because they are redundant.

```
$ cat MRRANK
0400|EFO|PT|N|
0399|EFO|SY|N|
0398|GO|PT|N|
0397|GO|SY|N|
0396|NCBITaxon|PT|N|
0395|NCBITaxon|SY|N|
0394|CL|PT|N|
0393|CL|SY|N|
0392|CHEBI|PT|N|
0391|CHEBI|SY|N|
```

```
0390|OBO|PT|N|
0389|OBO|SY|N|
0388|oboInOwl|PT|N|
0387|oboInOwl|SY|N|
0386|UO|PT|N|
0385|UO|SY|N|
0384|SNAP|PT|N|
0383|SNAP|SY|N|
0382|PATO|PT|N|
0381|PATO|SY|N|
0380|SPAN|PT|N|
0379|SPAN|SY|N|
```

5 Generating the UMLS format tables (ORF)

When producing the Concept Names File, MRCON, all of the keys in the concept dictionary are traversed. For each concept record for the label and, if present, any synonyms are produced. For example, the concept EFO_0003549 "caudal tuberculum" has the following MRCON records generated:

```
$ grep EFO_0003549 MRCON
EFO_0003549|ENG|P|L0004578|PF|S0004578|caudal tuberculum|0|
EFO_0003549|ENG|S|L0004579|SY|S0004579|posterior tubercle|0|
EFO_0003549|ENG|S|L0004580|SY|S0004580|posterior tuberculum|0|
```

Each record contains the Concept Identifier, language, term status, lexical unique identifier (unused), term status, string unique identifier, term string, and Least restriction level (unused).

Similarly, for each record in MRCON there is a corresponding record in the Vocabulary Sources File, MRSO:

```
$ grep EFO_0003549 MRSO
EFO_0003549|L0004578|S0004578|EFO|PT|http://www.ebi.ac.uk/efo/EFO_0003549|0|
EFO_0003549|L0004579|S0004579|EFO|SY|http://www.ebi.ac.uk/efo/EFO_0003549|0|
EFO_0003549|L0004580|S0004580|EFO|SY|http://www.ebi.ac.uk/efo/EFO_0003549|0|
```

Each record contains the Concept Identifier, lexical unique identifier (unused), string unique identifier, Source identifier, term type in source, unique identifier in source (in this case the fully specified URI), and source restriction level (unused).

The Semantic Types file, MRSTY, has one record for each semantic type assigned to a concept:

```
$ grep EFO_0003549 MRSTY
EFO_0003549|T205|unkn|
```

The Source Information file, MRSAB, contains information used by MetaMap when restricting to a particular set of sources, or excluding them.

```
$ more MRSAB
C4000000|C4000000|SPAN|SPAN|http://www.ifomis.org/bfo/1.1/span#|||||||0|1|1||||ENG|ascii|Y|Y|
C4000001|C4000001|CL|CL|http://purl.org/obo/owl/CL#|||||||0|713|617||||ENG|ascii|Y|Y|
C4000002|C4000002|oboInOwl|oboInOwl|http://www.geneontology.org/formats/oboInOwl#|||||||0|1|1|
C4000003|C4000003|GO|GO|http://purl.org/obo/owl/GO#|||||||0|3|2||||ENG|ascii|Y|Y|
C4000004|C4000004|OBO|OBO|http://purl.obolibrary.org/obo/|||||||0|68|42||||ENG|ascii|Y|Y|
C4000005|C4000005|PATO|PATO|http://purl.org/obo/owl/PATO|||||||0|5|3||||ENG|ascii|Y|Y|
C4000006|C4000006|EFO|EFO|http://www.ebi.ac.uk/efo/|||||||0|5222|3501||||ENG|ascii|Y|Y|
C4000007|C4000007|SNAP|SNAP|http://www.ifomis.org/bfo/1.1/snap#|||||||0|8|7||||ENG|ascii|Y|Y|
C4000008|C4000008|UO|UO|http://purl.org/obo/owl/UO#|||||||0|135|76||||ENG|ascii|Y|Y|
C4000009|C4000009|CHEBI|CHEBI|http://www.ebi.ac.uk/chebi/searchId.do?chebiId=|||||||0|0|0||||E
C4000010|C4000010|NCBITaxon|NCBITaxon|http://purl.org/obo/owl/NCBITaxon#|||||||0|77|46||||ENG|
```

The Concept Name Ranking File, MRRANK, provides information to the filtering program `filter_mrconso` needed when determining which records in MRCON and MRSO should be removed because they are redundant.

```
$ more MRRANK
0400|EFO|PT|N|
0399|EFO|SY|N|
0398|GO|PT|N|
0397|GO|SY|N|
0396|NCBITaxon|PT|N|
0395|NCBITaxon|SY|N|
0394|CL|PT|N|
0393|CL|SY|N|
0392|CHEBI|PT|N|
0391|CHEBI|SY|N|
0390|OBO|PT|N|
0389|OBO|SY|N|
0388|oboInOwl|PT|N|
0387|oboInOwl|SY|N|
0386|UO|PT|N|
0385|UO|SY|N|
0384|SNAP|PT|N|
0383|SNAP|SY|N|
0382|PATO|PT|N|
0381|PATO|SY|N|
0380|SPAN|PT|N|
0379|SPAN|SY|N|
```

6 Acknowledgements

Thanks to Tomasz Adamusiak for exposing how the EFO inferred data was organized.

7 APPENDIX

7.1 A Sample EFO concept

Original Data for Concept `EFO_0003549` (URI:http://www.ebi.ac.uk/efo/EFO_0003549) in n3 format:

```
:EFO_0003549 a owl:Class;
   rdfs:label "caudal tuberculum"^^<http://www.w3.org/2001/XMLSchema#string>;
   :alternative_term "posterior tubercle"^^<http://www.w3.org/2001/XMLSchema#string>,
      "posterior tuberculum"^^<http://www.w3.org/2001/XMLSchema#string>;
   :bioportal_provenance "Brain structure which is part of the diencephalon and is
      larger than the dorsal thalamus and ventral thalamus. From Neuroanatomy of the
      Zebrafish Brain.[accessedResource: ZFA:0000633][accessDate: 05-04-2011]"
          ^^<http://www.w3.org/2001/XMLSchema#string>,
      "posterior tubercle[accessedResource: ZFA:0000633][accessDate: 05-04-2011]"
          ^^<http://www.w3.org/2001/XMLSchema#string>,
      "posterior tuberculum[accessedResource: ZFA:0000633][accessDate: 05-04-2011]"
          ^^<http://www.w3.org/2001/XMLSchema#string>;
   :definition "Brain structure which is part of the diencephalon and is larger
      than the dorsal thalamus and ventral thalamus. From Neuroanatomy of the
      Zebrafish Brain."^^<http://www.w3.org/2001/XMLSchema#string>;
   :definition_citation "ZFA:0000633"^^<http://www.w3.org/2001/XMLSchema#string>;
   :definition_editor "Tomasz Adamusiak"^^<http://www.w3.org/2001/XMLSchema#string>;
   rdfs:subClassOf :EFO_0003331.
```

7.2 Source code for EFO data set creator

7.2.1 efo_extract.py

This module requires RDFLIB (http://code.google.com/p/rdflib/) (version RDFlib 4.1.0). The program generates both ORF and RRF versions of DATA. The RRF versions are used by the 2013 release of Data File Builder and the ORF versions are used by the 2011 release of Data File Builder.

```
"""
 /rhome/wjrogers/studio/python/rdf/efo_extract.py, Mon Jan  9 14:05:05 2012, edit by Will Roger

Extract concepts and synomyms from EFO_inferred_v2.18.owl and generate
UMLS format tables for use by MetaMap's Data File Builder.


Original Author: Willie Rogers, 09jan2012
"""
from rdflib import ConjunctiveGraph, Graph
from rdflib import Namespace, URIRef
from string import join
from readrdf import readrdf
```

```
import re
import sys

from mwi_utilities import normalize_ast_string

efo_datafile = '/usr/local/pub/rdf/EFO_inferred_v2.18.owl'
EFO = Namespace("http://www.ebi.ac.uk/efo/")
RDFS = Namespace("http://www.w3.org/2000/01/rdf-schema#")
RDF = Namespace('http://www.w3.org/1999/02/22-rdf-syntax-ns#')
OWL = Namespace("http://www.w3.org/2002/07/owl#")
BFO = Namespace("http://www.ifomis.org/bfo/1.1/snap#MaterialEntity")

mmencoding='ascii'

prefixdict = { 'http://www.ebi.ac.uk/efo/' : 'EFO',
               'http://purl.org/obo/owl/GO#' : 'GO',
               'http://purl.org/obo/owl/NCBITaxon#' : 'NCBITaxon',
               'http://purl.org/obo/owl/CL#' : 'CL',
               'http://www.ebi.ac.uk/chebi/searchId.do?chebiId=' : 'CHEBI',
               'http://purl.obolibrary.org/obo/' : 'OBO',
               'http://www.geneontology.org/formats/oboInOwl#' : 'oboInOwl',
               'http://purl.org/obo/owl/UO#' : 'UO',
               'http://www.ifomis.org/bfo/1.1/snap#' : 'SNAP',
               'http://purl.org/obo/owl/PATO' : 'PATO',
               'http://www.ifomis.org/bfo/1.1/span#' : 'SPAN', }
prefixlist = [ 'http://www.ebi.ac.uk/efo/',
               'http://purl.org/obo/owl/GO#',
               'http://purl.org/obo/owl/NCBITaxon#',
               'http://purl.org/obo/owl/CL#',
               'http://www.ebi.ac.uk/chebi/searchId.do?chebiId=',
               'http://purl.obolibrary.org/obo/',
               'http://www.geneontology.org/formats/oboInOwl#',
               'http://purl.org/obo/owl/UO#',
               'http://www.ifomis.org/bfo/1.1/snap#',
               'http://purl.org/obo/owl/PATO',
               'http://www.ifomis.org/bfo/1.1/span#', ]
srclist = [ 'EFO', 'GO', 'NCBITaxon', 'CL', 'CHEBI', 'OBO', 'oboInOwl',
            'UO', 'SNAP', 'PATO', 'SPAN']
semtypes = [('T045','Genetic Function','genf'),
            ('T028','Gene or Genome','gegm'),
             ('T116','Amino Acid, Peptide, or Protein','aapp'),]

def list_id_and_labels(graph):
    for s,p,o in graph:
        if p.__str__ == 'http://www.w3.org/2000/01/rdf-schema#label':
            print(s,p,o)

def query_labels(graph):
```

```
    ns = dict(efo=EFO,rdfs=RDFS)
    return graph.query('SELECT ?aname ?bname WHERE { ?a rdfs:label ?b }',
                       initNs=ns)

def query_efo_type(graph, typename):
    ns = dict(efo=EFO, rdfs=RDFS)
    return graph.query('SELECT ?aname ?bname WHERE { ?a efo:%s ?b }' % typename,
                       initNs=ns)

def query_efo_concepts(graph):
    ns = dict(owl=OWL, rdf=RDF)
    return graph.query('SELECT ?s { ?s rdf:type owl:Class }', initNs=ns)

def query_efo_concept_labels(graph):
    " return all labels (concepts) from graph (EFO_inferred_v2.18.owl)"
    ns = dict(owl=OWL, rdf=RDF, rdfs=RDFS)
    return graph.query('SELECT ?s ?o { ?s rdf:type owl:Class . ?s rdfs:label ?o }',
                       initNs=ns)

def query_efo_concept_synonyms(graph):
    " return all alternative_terms (synonyms) from graph (EFO_inferred_v2.18.owl)"
    ns = dict(owl=OWL, rdf=RDF, efo=EFO)
    return graph.query('SELECT ?s ?o { ?s rdf:type owl:Class . ?s efo:alternative_term ?o }',
                       initNs=ns)

def escape_re_chars(prefix):
    " escape regular expression special characters in url"
    return prefix.replace('?','\?').replace('#','\#')

def abbrev_uri_original(uri):
    " remove prefix from uri leaving unique part of identifier "
    for prefix in prefixlist:
        m = re.match(r"(%s)(.*)" % escape_re_chars(prefix),uri)
        if m != None:
            return m.group(2).replace(':','_')
    return uri

def abbrev_uri(uri):
    " remove prefix from uri leaving unique part of identifier "
    for prefix in prefixlist:
        # print('prefix = ''%s''' % prefix)
        m = uri.find(prefix)
        if m == 0:
            newuri = uri[len(prefix):].replace(':','_')
            if newuri.find(':') >= 0:
                print("problem with abbreviated uri: %s" % newuri)
            return newuri
    return uri
```

```
def get_source_name_original(uri):
    " derive source name from uri "
    for prefix in prefixdict.keys():
        m = re.match(r"(%s)(.*)" %  escape_re_chars(prefix),uri)
        if m != None:
            return prefixdict[m.group(1)]
    return uri

def get_source_name(uri):
    " derive source name from uri "
    for prefix in prefixdict.keys():
        m = uri.find(prefix)
        if m == 0:
            return prefixdict[uri[0:len(prefix)]]
    return uri


def collect_concepts(graph):
    """
    Return dictionaries (maps) of concepts and synonyms
    (alternative_terms) from results of SPARQL queries
    """
    cdict = {}
    conceptresult=query_efo_concept_labels(graph)
    serialnum = 1
    for row in conceptresult:
        key = tuple(row)[0].__str__()
        if cdict.has_key(key):
            cdict[key].append(row)
        else:
            cdict[key] = [row]
    syndict = {}
    synonymresult = query_efo_concept_synonyms(graph)
    for row in synonymresult:
        key = tuple(row)[0].__str__()
        if syndict.has_key(key):
            syndict[key].append(row)
        else:
            syndict[key] = [row]
    return cdict,syndict

def is_valid_cui(cui):
    return re.match(r"[A-Za-z]+[\_]*[0-9]+" , cui)

def gen_mrcon_original(graph,filename):
    """
    Generate UMLS format MRCON table.
```

```
    return rows of the form:
    EFO_0003549|ENG|P|L0000001|PF|S0000001|caudal tuberculum|0|
    EFO_0003549|ENG|S|L0000002|SY|S0000002|posterior tubercle|0|
    EFO_0003549|ENG|S|L0000003|SY|S0000003|posterior tuberculum|0|
    """
    conceptresult=query_efo_concept_labels(graph)
    fp = open(filename,'w')
    serialnum = 1
    for row in conceptresult:
        if is_valid_cui(abbrev_uri(tuple(row)[0].encode(mmencoding, 'replace'))):
            fp.write("%s|ENG|P|L%07d|PF|S%07d|%s|0|\n" % (abbrev_uri(tuple(row)[0].encode(mmenc
                                                          serialnum, serialnum,
                                                          tuple(row)[1].encode(mmencoding, 'rep
            serialnum = serialnum + 1
    fp.close()

def gen_mrcon(filename, cdict={}, syndict={}, strdict={}, luidict={}):
    """
    Generate UMLS format MRCON (concepts) table.

    return rows of the form:
    EFO_0003549|ENG|P|L0000001|PF|S0000001|caudal tuberculum|0|
    EFO_0003549|ENG|S|L0000002|SY|S0000002|posterior tubercle|0|
    EFO_0003549|ENG|S|L0000003|SY|S0000003|posterior tuberculum|0|
    """
    fp = open(filename,'w')
    serialnum = 1
    for key in cdict.keys():
        for row in cdict[key]:
            if is_valid_cui(abbrev_uri(tuple(row)[0].encode(mmencoding, 'replace'))):
                term = tuple(row)[1].encode(mmencoding, 'replace').strip()
                lui = get_lui(luidict,term)
                if lui == None:
                    sys.stderr.write('gen_mrso: LUI missing for prefname %s\n' % (term))
                sui = strdict.get(term)
                if sui == None:
                    sys.stderr.write('gen_mrcon:SUI missing for preferred name %s\n' % (term))
                fp.write("%s|ENG|P|%s|PF|%s|%s|0|\n" % (abbrev_uri(tuple(row)[0].encode(mmencod
                                                        lui, sui, term))
                serialnum = serialnum + 1
        if syndict.has_key(key):
            if is_valid_cui(abbrev_uri(tuple(row)[0].encode(mmencoding, 'replace'))):
                for row in syndict[key]:
                    term = tuple(row)[1].encode(mmencoding, 'replace').strip()
                    lui = get_lui(luidict,term)
                    if lui == None:
                        sys.stderr.write('gen_mrso: LUI missing for synonym %s\n' % (term))
```

```
                    sui = strdict.get(term)
                    if sui == None:
                        sys.stderr.write('gen_mrcon:SUI missing for synonym %s\n' % (term))
                    fp.write("%s|ENG|S|%s|SY|%s|%s|0|\n" % (abbrev_uri(tuple(row)[0].encode(mme
                                                            lui, sui, term))
                    serialnum = serialnum + 1
    fp.close()

def gen_mrso(filename, cdict={}, syndict={}, strdict={}, luidict={}):
    """
    Generate UMLS format MRSO (sources) table.

    EFO_0003549|L0000001|S0000001|EFO|PT|http://www.ebi.ac.uk/efo/EFO_0003549|0|
    EFO_0003549|L0000002|S0000002|EFO|SY|http://www.ebi.ac.uk/efo/EFO_0003549|0|
    EFO_0003549|L0000003|S0000003|EFO|SY|http://www.ebi.ac.uk/efo/EFO_0003549|0|
   """
    fp = open(filename,'w')
    serialnum = 1
    for key in cdict.keys():
        for row in cdict[key]:
            if is_valid_cui(abbrev_uri(tuple(row)[0].encode(mmencoding, 'replace'))):
                term = tuple(row)[1].encode(mmencoding, 'replace').strip()
                lui = get_lui(luidict,term)
                if lui == None:
                    sys.stderr.write('gen_mrso: LUI missing for prefname %s\n' % (term))
                sui = strdict.get(term)
                if sui == None:
                    sys.stderr.write('gen_mrso: SUI missing for prefname %s\n' % (term))
                fp.write("%s|%s|%s|%s|PT|%s|0|\n" % (abbrev_uri(tuple(row)[0].encode(mmencoding
                                                    lui, sui,
                                                    get_source_name(tuple(row)[0].encode(mmenc
                                                    tuple(row)[0].encode(mmencoding, 'replace'
                serialnum = serialnum + 1
        if syndict.has_key(key):
            for row in syndict[key]:
                if is_valid_cui(abbrev_uri(tuple(row)[0].encode(mmencoding, 'replace'))):
                    term = tuple(row)[1].encode(mmencoding, 'replace').strip()
                    lui = get_lui(luidict,term)
                    if lui == None:
                        sys.stderr.write('gen_mrso: LUI missing for synonym %s\n' % (term))
                    sui = strdict.get(term)
                    if sui == None:
                        sys.stderr.write('SUI missing for synonym %s\n' % (term))
                    fp.write("%s|%s|%s|%s|SY|%s|0|\n" % (abbrev_uri(tuple(row)[0].encode(mmenco
                                                        lui, sui,
                                                        get_source_name(tuple(row)[0].encode(m
                                                        tuple(row)[0].encode(mmencoding, 'repl
                    serialnum = serialnum + 1
```

```
    fp.close()


def gen_mrconso(filename, cdict={}, syndict={}, auidict={}, strdict={}, luidict={}):
    """
    Generate UMLS RRF format MRCONSO (concept+sources) table

    cui|lat|ts|lui|stt|sui|ispref|aui|saui|scui|sdui|sab|tty|code|str|srl|suppress|cvf

    EFO0003549|ENG|P|L0000001|PF|S0000001|Y|A0003549||||EFO|PT|http://www.ebi.ac.uk/efo/EFO_000
    """
    fp = open(filename,'w')
    serialnum = 1
    for key in cdict.keys():
        for row in cdict[key]:
            uri = tuple(row)[0].encode(mmencoding, 'replace').strip()
            if is_valid_cui(abbrev_uri(uri)):
                sab = get_source_name(uri)
                term = tuple(row)[1].encode(mmencoding, 'replace').strip()
                aui = auidict.get((term,sab))
                if aui == None:
                    sys.stderr.write('gen_mrconso:AUI missing for preferred name %s,%s\n' % (te
                lui = get_lui(luidict,term)
                if lui == None:
                    sys.stderr.write('gen_mrso: LUI missing for prefname %s\n' % (term))
                sui = strdict.get(term)
                if sui == None:
                    sys.stderr.write('gen_mrconso:SUI missing for preferred name %s\n' % (term)
                fp.write("%s|ENG|P|%s|PF|%s|Y|%s||||%s|PT|%s|%s|0|N||\n" % \
                             (abbrev_uri(uri),lui,sui,
                              aui,sab,uri,term))
                serialnum = serialnum + 1
        if syndict.has_key(key):
            uri = tuple(row)[0].encode(mmencoding, 'replace').strip()
            if is_valid_cui(abbrev_uri(uri)):
                for row in syndict[key]:
                    sab = get_source_name(uri)
                    term = tuple(row)[1].encode(mmencoding, 'replace').strip()
                    aui = auidict.get((term,sab))
                    if aui == None:
                        sys.stderr.write('gen_mrconso:AUI missing for synonym %s,%s\n' % (term,
                    lui = get_lui(luidict,term)
                    if lui == None:
                        sys.stderr.write('gen_mrso: LUI missing for synonym %s\n' % (term))
                    sui = strdict.get(term)
                    if sui == None:
                        sys.stderr.write('gen_mrconso:SUI missing for synonym %s\n' % (term))
                    fp.write("%s|ENG|S|%s|SY|%s|Y|%s||||%s|PT|%s|%s|0|N||\n" % \
```

```
                                (abbrev_uri(uri),lui,sui,
                                 aui,sab,uri,term))
                    serialnum = serialnum + 1

    fp.close()

def get_semantic_typeid(uri):
    """ return semantic type id for uri, currently all uris belong to
    the unknown semantic type. """
    return 'T205'

def get_semantic_typeabbrev(uri):
    """ return semantic type abbreviation for uri, currently all uris
    belong to the unknown semantic type."""
    return 'unkn'

def get_semantic_typename(uri):
    """ return semantic type name for uri, currently all uris
    belong to the unknown semantic type."""
    return "Unknown"

def get_semantic_typetree_number(uri):
    """ return semantic tree number for uri, currently all uris
    belong to the unknown semantic type."""
    return "A0.0.0.0.0.0"

def get_semantic_typeui(uri):
    """ return semantic tree number for uri, currently all uris
    belong to the unknown semantic type."""
    return "AT0000000"

def gen_mrsty(filename, cdict={}, syndict={}):
    """ Generate UMLS ORF format MRSTY (semantic type) table.  Currently,
    all of the concepts are assigned the semantic type "unkn". """
    fp = open(filename,'w')
    for key in cdict.keys():
        for row in cdict[key]:
            if is_valid_cui(abbrev_uri(tuple(row)[0].encode(mmencoding, 'replace'))):
                fp.write("%s|%s|%s|\n" % (abbrev_uri(tuple(row)[0].encode(mmencoding, 'replace'
                                          get_semantic_typeid(tuple(row)[0].encode(mmencoding,
                                          get_semantic_typeabbrev(tuple(row)[0].encode(mmencodi
    fp.close()

def gen_mrsty_rrf(filename, cdict={}, syndict={}):
    """ Generate UMLS ORF format MRSTY (semantic type) table.  Currently,
    all of the concepts are assigned the semantic type "unkn".

    MRSTY.RFF contaons lines like
```

```
    C0000005|T116|A1.4.1.2.1.7|Amino Acid, Peptide, or Protein|AT17648347||
    C0000005|T121|A1.4.1.1.1|Pharmacologic Substance|AT17575038||
    C0000005|T130|A1.4.1.1.4|Indicator, Reagent, or Diagnostic Aid|AT17634323||
    C0000039|T119|A1.4.1.2.1.9|Lipid|AT17617573|256|
    C0000039|T121|A1.4.1.1.1|Pharmacologic Substance|AT17567371|256|
    """
    fp = open(filename,'w')
    for key in cdict.keys():
        for row in cdict[key]:
            if is_valid_cui(abbrev_uri(tuple(row)[0].encode(mmencoding, 'replace'))):
                fp.write("%s|%s|%s|%s|%s||\n" % (abbrev_uri(tuple(row)[0].encode(mmencoding, 'r
                                                 get_semantic_typeid(tuple(row)[0].encode(mmenco
                                                 get_semantic_typetree_number(tuple(row)[0].enco
                                                 get_semantic_typename(tuple(row)[0].encode(mmen
                                                 get_semantic_typeui(tuple(row)[0].encode(mmenco
    fp.close()


def gen_mrsat(filename, cdict={}, syndict={}, auidict={}, strdict={}, luidict={}):
    """ Generate UMLS format MRSAT (Simple Concept and String
    Attributes) table.  Currently, empty. """
    fp = open(filename,'w')
    fp.close()

def gen_mrsab(filename,cdict={},syndict={}):
    """ Generate UMLS format MRSAB (Source Informatino) table.
    Currently, empty. """
    cui_index = 4000000
    fp = open(filename,'w')
    for k,v in prefixdict.items():
        rcui=vcui='C%7s' % cui_index
        vsab=rsab=v
        son=k
        sf=vstart=vend=imeta=rmeta=slc=scc=ssn=scit=''
        srl='0'
        if len(cdict) > 0:
            # count concepts that belong to source
            cres=filter(lambda x: re.match(r"(%s)(.*)" % k,x[0].__str__()), cdict.items())
            cfr='%d' % len(cres)
            if len(syndict) > 0:
                # count synonyms that belong to source
                sres=filter(lambda x: re.match(r"(%s)(.*)" % k,x[0].__str__()), syndict.items()
                tfr='%d' % (len(cres)+len(sres))
            else:
                tfr='%d' % len(cres)
        else:
            cfr=tfr=''
        cxty=ttyl=atnl=''
```

```
        lat='ENG'
        cenc='ascii'
        curver=sabin='Y'
        fp.write('%s\n' % \
                    join((vcui,rcui,vsab,rsab,son,sf,vstart,vend,
                          imeta,rmeta,slc,scc,srl,tfr,cfr,cxty,
                          ttyl,atnl,lat,cenc,curver,sabin,ssn,scit),'|'))
        cui_index = cui_index + 1
    fp.close()

def gen_mrrank(filename):
    """ Generate UMLS format MRRANK (Concept Name Ranking) table. """
    ttylist = ['PT', 'SY']
    pred = 400
    fp = open(filename, 'w')
    for sab in srclist:
        for tty in ttylist:
            fp.write('%04d|%s|%s|N|\n' % (pred,sab,tty))
            pred = pred - 1
    fp.close()

def print_result(result):
    for row in result:
        print("%s|%s" % (tuple(row)[0],tuple(row)[1]))

def write_result(result, filename):
    f = open(filename, 'w')
    for row in result:
        f.write(('%s\n' % join((tuple(row)[0],tuple(row)[1]), '|')).encode(mmencoding, 'replace
    f.close()

def gen_mrcon_list(cdict={}, syndict={}):
    """
    return rows of the form:
    EFO_0003549|ENG|P|L0000001|PF|S0000001|caudal tuberculum|0|
    EFO_0003549|ENG|S|L0000002|SY|S0000002|posterior tubercle|0|
    EFO_0003549|ENG|S|L0000003|SY|S0000003|posterior tuberculum|0|
    """
    mrconlist = []
    serialnum = 1
    for key in cdict.keys():
        for row in cdict[key]:
            if is_valid_cui(abbrev_uri(tuple(row)[0].encode(mmencoding, 'replace'))):
                # "%s|ENG|P|L%07d|PF|S%07d|%s|0|\n"
                mrconlist.append((abbrev_uri(tuple(row)[0].encode(mmencoding, 'replace')),'ENG'
                                  'L%07d' % serialnum, 'PF', 'S%07d' % serialnum,
                                  tuple(row)[1].encode(mmencoding, 'replace'),'0',''))
                serialnum = serialnum + 1
```

```
        if syndict.has_key(key):
            for row in syndict[key]:
                if is_valid_cui(abbrev_uri(tuple(row)[0].encode(mmencoding, 'replace'))):
                    # "%s|ENG|S|L%07d|SY|S%07d|%s|0|\n"
                    mrconlist.append((abbrev_uri(tuple(row)[0].encode(mmencoding, 'replace')),'
                                      'L%07d' % serialnum, 'SY', 'S%07d' % serialnum,
                                      tuple(row)[1].encode(mmencoding, 'replace'),'0',''))
                    serialnum = serialnum + 1
    return mrconlist

def gen_mrconso_list(cdict={}, syndict={}, auidict={}):
    """
    return rows of the form:
    EFO_0003549|ENG|P|L0000001|PF|S0000001||||caudal tuberculum|0|
    EFO_0003549|ENG|S|L0000002|SY|S0000002|posterior tubercle|0|
    EFO_0003549|ENG|S|L0000003|SY|S0000003|posterior tuberculum|0|
    """

    # mrconlist = []
    # serialnum = 1
    # for key in cdict.keys():
    #     for row in cdict[key]:
    #         if is_valid_cui(abbrev_uri(tuple(row)[0].encode(mmencoding, 'replace'))):
    pass

def gen_strdict(cdict, syndict):
    """ Generate dict of concept and synonym triples mapped by string. """
    strdict = {}
    for triplelist in cdict.values():
        for triple in triplelist:
            strkey = triple[1].__str__()
            if strdict.has_key(strkey):
                strdict[strkey].append(triple)
            else:
                strdict[strkey] = [triple]
    for triplelist in syndict.values():
        for triple in triplelist:
            strkey = triple[1].__str__()
            if strdict.has_key(strkey):
                strdict[strkey].append(triple)
            else:
                strdict[strkey] = [triple]
    return strdict

def gen_nmstrdict(cdict, syndict):
    """ Generate dict of concept and synonym triples mapped by nomalized string. """
    strdict = {}
    for triplelist in cdict.values():
```

```
        for triple in triplelist:
            strkey = normalize_ast_string(triple[1].__str__())
            if strdict.has_key(strkey):
                strdict[strkey].append(triple)
            else:
                strdict[strkey] = [triple]
    for triplelist in syndict.values():
        for triple in triplelist:
            strkey = normalize_ast_string(triple[1].__str__())
            if strdict.has_key(strkey):
                strdict[strkey].append(triple)
            else:
                strdict[strkey] = [triple]
    return strdict

def gen_strdict_histogram(strdict):
    """ Generate histrogram of lengths of string dictionary values. """
    histogram = {}
    for v in strdict.values():
        key = '%d' % len(v)
        if histogram.has_key(key):
            histogram[key] += 1
        else:
            histogram[key] = 1
    return histogram

def gen_strdict_listsizedict(strdict):
    sizedict = {}
    for k,v in strdict.items():
        key = '%d' % len(v)
        if sizedict.has_key(key):
            sizedict[key].append(k)
        else:
            sizedict[key] = [k]
    return sizedict

def gen_aui_dict(cdict=[],syndict=[],auiprefix='A',offset=0):
    """ A simple way to generate atom unique identifiers (AUIS):

    1. Generate list of strings + vocabulary source from ontology.
    2. Sort list
    3. assign auis in descending order of sorted list.


    cdict: concept dictionary
    syndict: synonym dictionary
    auiprefix: prefix for Atom identifiers,
               usually "A" for standalone DataSets,
```

```
                should be "B" for dataset to be used with UMLS.
                A can be used if range new identifier space is outside
                of existing UMLS atom identifier space.
    offset=start of range for identifiers, default is zero
    """
    aset=set([])
    auidict={}
    for cstr in cdict.keys():
        if cstr == 'http://www.ebi.ac.uk/efo/EFO_0000694':
            print("%s -> %s" % (cstr,'is SARS '))
        prefterm = cdict[cstr][0][1].strip().encode(mmencoding, 'replace')
        sab = get_source_name(cstr.encode(mmencoding, 'replace'))
        if prefterm == 'SARS':
            print('%s --> pref: %s,%s' % (cstr, prefterm, sab))
        aset.add((prefterm, sab))
        if syndict.has_key(cstr):
            for row in syndict[cstr]:
                synonym = row[1].strip().encode(mmencoding, 'replace')
                if synonym == 'SARS':
                    print('%s --> syn: %s,%s' % (cstr, synonym, sab))
                sab = get_source_name(cstr.encode(mmencoding, 'replace'))
                aset.add((synonym, sab))
    alist = [x for x in aset]
    alist.sort()
    i = offset
    for atom in alist:
        auidict[atom] = '%s%08d' % (auiprefix,i)
        i = i + 1
    return auidict

def gen_sui_dict(cdict=[],syndict=[],suiprefix='S',offset=0):
    """ A simple way to generate String Unique Identifiers(SUIS):

    1. Generate list of strings + vocabulary source from ontology.
    2. Sort list
    3. assign auis in descending order of sorted list.

    cdict: concept dictionary
    syndict: synonym dictionary
    suiprefix: prefix for string identifiers,
              usually "S" for standalone DataSets,
              should be "T" for dataset to be used with UMLS,
              "S" can be used if range new identifier space is outside
              of existing UMLS string identifier space.
    offset=start of range for identifiers, default is zero
    """
    sset=set([])
    suidict={}
```

```
    for cstr in cdict.keys():
        if cstr == 'http://www.ebi.ac.uk/efo/EFO_0000694':
            print("%s -> %s" % (cstr,'is SARS '))
        prefterm = cdict[cstr][0][1].strip().encode(mmencoding, 'replace')
        sset.add(prefterm)
        if prefterm == 'SARS':
            print('%s --> pref: %s' % (cstr, prefterm))
    for cstr in syndict.keys():
        if cstr == 'http://www.ebi.ac.uk/efo/EFO_0000694':
            print("%s -> %s" % (cstr,'is SARS '))
        for row in syndict[cstr]:
            synonym = row[1].strip().encode(mmencoding, 'replace')
            sset.add(synonym)
            if synonym == 'SARS':
                print('%s --> syn: %s' % (cstr, synonym))
    slist = [x for x in sset]
    slist.sort()
    i = offset
    for mstring in slist:
        suidict[mstring] = '%s%08d' % (suiprefix,i)
        i = i + 1
    return suidict

def gen_lui_dict(cdict=[],syndict=[],luiprefix='L',offset=0):
    """ A simple way to generate Lexical Unique Identifiers(SUIS):

    1. Generate list of strings + vocabulary source from ontology.
    2. Sort list
    3. assign auis in descending order of sorted list.

    cdict: concept dictionary
    syndict: synonym dictionary
    suiprefix: prefix for string identifiers,
              usually "L" for standalone DataSets,
              should be "M" for dataset to be used with UMLS,
              "L" can be used if range new identifier space is outside
              of existing UMLS string identifier space.
    offset=start of range for identifiers, default is zero
    """
    nasset=set([])
    luidict={}
    for cstr in cdict.keys():
        if cstr == 'http://www.ebi.ac.uk/efo/EFO_0000694':
            print("%s -> %s" % (cstr,'is SARS '))
        prefterm = cdict[cstr][0][1].strip().encode(mmencoding, 'replace')
        nasset.add(normalize_ast_string(prefterm))
        if prefterm == 'SARS':
            print('%s --> pref: %s' % (cstr, prefterm))
```

```
    for cstr in syndict.keys():
        if cstr == 'http://www.ebi.ac.uk/efo/EFO_0000694':
            print("%s -> %s" % (cstr,'is SARS '))
        for row in syndict[cstr]:
            synonym = row[1].strip().encode(mmencoding, 'replace')
            nasset.add(normalize_ast_string(synonym))
            if synonym == 'SARS':
                print('%s --> syn: %s' % (cstr, synonym))
    naslist = [x for x in nasset]
    naslist.sort()
    i = offset
    for nasstring in naslist:
        luidict[nasstring] = '%s%08d' % (luiprefix,i)
        i = i + 1
    return luidict

def get_lui(luidict, mstring):
    """ get lui for un-normalized string from lui dictionary """
    return luidict.get(normalize_ast_string(mstring), 'LUI unknown')

def print_couples(alist):
    for el in alist:
        print("%s: %s" % (el[0].__str__(),el[1].__str__()))

def process(rdffilename):
    print('reading %s' % rdffilename)
    graph=readrdf(rdffilename)
    print('finding concepts and synonyms')
    cdict,syndict = collect_concepts(graph)
    print('Generating Atom Unique Identifier Dictionary')
    auidict = gen_aui_dict(cdict,syndict)
    print('Generating String Unique Identifier Dictionary')
    suidict = gen_sui_dict(cdict,syndict)
    print('Generating Lexical Unique Identifier Dictionary')
    luidict = gen_lui_dict(cdict,syndict)

#rrf
    print('generating MRCONSO.RRF')
    gen_mrconso('MRCONSO.RRF',cdict,syndict,auidict,suidict,luidict)


#orf
    print('generating MRCON')
    gen_mrcon('MRCON',cdict,syndict,suidict,luidict)
    print('generating MRSO')
    gen_mrso('MRSO',cdict,syndict,suidict,luidict)

#both rrf and orf
```

```
    print('generating MRSAB')
    gen_mrsab('MRSAB.RRF',cdict,syndict)
    print('generating MRRANK')
    gen_mrrank('MRRANK.RRF')
    print('generating MRSAT')
    gen_mrsat('MRSAT.RRF',cdict,syndict)
    print('generating MRSTY')
    gen_mrsty('MRSTY',cdict,syndict)
    print('generating MRSTY.RRF')
    gen_mrsty_rrf('MRSTY.RRF',cdict,syndict)

if __name__ == "__main__":
    print('reading %s' % efo_datafile)
    process(efo_datafile)
```

7.2.2 readrdf.py

```
"""
readrdf.py -- rdf utilities
"""
from rdflib import Graph
from string import join
import sys

def readrdf(filename):
    graph = Graph()
    graph.parse(filename)
    return graph

def print_triples(graph):
    for s,p,o in graph:
        print s,p,o

def print_piped_triples(graph):
    for s,p,o in graph:
        print join([s,p,o], '|')

def doit(filename):
    graph = readrdf(filename)
    print_triples(graph)
    return graph

def save_graph(graph, filename, format='n3'):
    f = open(filename, 'w')
    f.write(graph.serialize(None, format))
    f.close()

def write_piped_triples(graph, filename):
```

```
    f = open(filename, 'w')
    for s,p,o in graph:
        f.write(('%s\n' % join([s,p,o], '|')).encode('utf-8'))
    f.close()
```

7.2.3 mwi_utilities.py

```
"""
  MetaWordIndex Utilities.

  Currently only contains a Python implementation of the Prolog
  predicate normalize_meta_string/2.

  Translated from Alan Aronson's original prolog version: mwi_utilities.pl.

  Created: Fri Dec 11 10:10:23 2008

  @author <a href="mailto:wrogers@nlm.nih.gov">Willie Rogers</a>
  @version $Id: MWIUtilities.java,v 1.2 2005/03/04 16:11:12 wrogers Exp $
"""

import sys
import string
import nls_strings
from metamap_tokenization import tokenize_text_utterly, is_ws_word, ends_with_s

def normalize_meta_string(metastring):
    """
    Normalize metathesaurus string.

    normalizeMetaString(String) performs "normalization" on String to produce
    NormalizedMetaString.  The purpose of normalization is to detect strings
    which are effectively the same.  The normalization process (also called
    lexical filtering) consists of the following steps:

       * removal of (left []) parentheticals;
       * removal of multiple meaning designators (<n>);
       * NOS normalization;
       * syntactic uninversion;
       * conversion to lowercase;
       * replacement of hyphens with spaces; and
       * stripping of possessives.

    Some right parentheticals used to be stripped, but no longer are.
    Lexical Filtering Examples:
    The concept "Abdomen" has strings "ABDOMEN" and "Abdomen, NOS".
    Similarly, the concept "Lung Cancer" has string "Cancer, Lung".
    And the concept "1,4-alpha-Glucan Branching Enzyme" has a string
```

```
    "1,4 alpha Glucan Branching Enzyme".

    Note that the order in which the various normalizations occur is important.
    The above order is correct.
    important; e.g., parentheticals must be removed before either lowercasing
    or normalized syntactic uninversion (which includes NOS normalization)
    are performed.

    @param string meta string to normalize.
    @return normalized meta string.
    """
    try:
        pstring = remove_left_parentheticals(metastring.strip())
        un_pstring = nls_strings.normalized_syntactic_uninvert_string(pstring)
        lc_un_pstring = un_pstring.lower()
        hlc_un_pstring = remove_hyphens(lc_un_pstring)
        norm_string = strip_possessives(hlc_un_pstring)
        return norm_string.strip()
    except TypeError,te:
        print('%s: string: %s' % (te,metastring))


left_parenthetical = ["[X]","[V]","[D]","[M]","[EDTA]","[SO]","[Q]"]

def remove_left_parentheticals(astring):
    """ remove_left_parentheticals(+String, -ModifiedString)

    remove_left_parentheticals/2 removes all left parentheticals
    (see left_parenthetical/1) from String. ModifiedString is what is left. """
    for lp in left_parenthetical:
        if astring.find(lp) == 0:
            return astring[len(lp):].strip()
    return astring

def remove_hyphens(astring):
    """ remove_hyphens/2 removes hyphens from String and removes extra blanks
    to produce ModifiedString. """
    return remove_extra_blanks(astring.replace('-', ' ')).strip()

# def remove_possessives(tokens):
#     """ remove_possessives/2 filters out possessives
#         from the results of tokenize_text_utterly/2. """
#     if len(tokens) > 2:
#         if is_ws_word(tokens[0]) & (tokens[1] == "'") & (tokens[2] == "s"):
#             return tokens[0:1] + remove_possessives(tokens[3:])
#         else:
#             return tokens[0:1] + remove_possessives(tokens[1:])
#     elif len(tokens) > 1:
```

```
#         if is_ws_word(tokens[0]) & (tokens[1] == "'") & ends_with_s(tokens[0]):
#             return tokens[0:1] + remove_possessives(tokens[2:])
#         else:
#             return tokens[0:1] + remove_possessives(tokens[1:])
#     return tokens

# def remove_possessives(tokens):
#     modtokens=[]
#     for token in tokens:
#         if token[-2:] == "'s":
#             modtokens.append(token[:-2])
#         elif token[-2:] == "s'":
#             modtokens.append(token[:-1])
#         else:
#             modtokens.append(token)
#     return modtokens

# def remove_possessives(tokens):
#     """ remove_possessives/2 filters out possessives
#         from the results of tokenize_text_utterly/2. """
#     if len(tokens) > 1:
#         if is_ws_word(tokens[0]) & (tokens[1] == "'"):
#             if tokens[0].endswith('s'):
#                 return tokens[0:1] + remove_possessives(tokens[2:])
#             elif len(tokens) > 2:
#                 if (tokens[2] == "s"):
#                     return tokens[0:1] + remove_possessives(tokens[3:])
#                 else:
#                     return tokens[0:1] + remove_possessives(tokens[1:])
#             else:
#                 return tokens[0:1] + remove_possessives(tokens[1:])
#         else:
#             return tokens[0:1] + remove_possessives(tokens[1:])
#     return tokens

def is_quoted_string(tokens, i):
    pass

def is_apostrophe_s(tokens, i):
    if i+2 < len(tokens):
        if is_ws_word(tokens[i]) & (tokens[i+1] == "'") & (tokens[i+2] == "s"):
            if i+3 < len(tokens):
                if string.punctuation.find(tokens[i+3][0]) >= 0:
                    return False
            return True
    return False
```

```
def is_s_apostrophe(tokens, i):
    if i+1 < len(tokens):
        if is_ws_word(tokens[i]) & tokens[i].endswith('s') & (tokens[i+1] == "'"):
            return True
    return False

def remove_possessives(tokens):
    """ remove_possessives/2 filters out possessives
        from the results of tokenize_text_utterly/2.

    EBNF for possessives using tokenization of original prolog predicates:

    tokenlist -> token tokenlist | possessive tokenlist ;
    quoted_string -> "'" tokenlist "'" ;
    possessive --> apostrophe_s_possessive | s_apostrophe_possessive ;
    apostrophe_s_possessive -> alnum_word "'" "s" ;
    s_apostrophe_possessive -> alnum_word_ending_with_s "'" ;
    """
    i = 0
    newtokens = []
    while i < len(tokens):
        if is_apostrophe_s(tokens, i):
            newtokens.append(tokens[i])
            i+=3
        elif is_s_apostrophe(tokens, i):
            newtokens.append(tokens[i])
            i+=2
        else:
            newtokens.append(tokens[i])
            i+=1
    return newtokens

def strip_possessives(astring):
    """ strip_possessives/2 tokenizes String, uses
    metamap_tokenization:remove_possessives/2, and then rebuilds StrippedString. */ """
    tokens = tokenize_text_utterly(astring)
    stripped_tokens = remove_possessives(tokens)
    if tokens == stripped_tokens:
        return astring
    else:
        return string.join(stripped_tokens,'')


# def strip_possessives(astring):
    # if astring.find("'s") >= 0:
    #     rstring2 = astring.replace("'s","|dps|")
    #     rstring1 = rstring2.replace("'s","")
    #     rstring0 = rstring1.replace("|dps|","'s")
```

```
    # else:
    #     rstring0 = astring.replace("'s","")
    # return rstring0.replace("s'","s")

def remove_extra_blanks(astring):
    """ remove extra inter-token blanks """
    return string.join(astring.split())

def normalize_ast_string(aststring):
    """ similar to normalize_meta_string except hyphens are not removed
     normalizeAstString(String) performs "normalization" on String to produce
    NormalizedMetaString.  The purpose of normalization is to detect strings
    which are effectively the same.  The normalization process (also called
    lexical filtering) consists of the following steps:

       * syntactic uninversion;
       * conversion to lowercase;
       * stripping of possessives.

    """
    un_pstring = nls_strings.syntactic_uninvert_string(aststring)
    lc_un_pstring = un_pstring.lower()
    norm_string = strip_possessives(lc_un_pstring)
    return norm_string

if __name__ == '__main__':
    # a test fixture to test normalizeMetaString method.
    if len(sys.argv) > 1:
        print('"%s"' % normalize_meta_string(string.join(sys.argv[1:], ' ')))

# fin: mwi_utilities
```

7.2.4 metamap_tokenization.py

```
""" metamap_tokenization.py -- Purpose:  MetaMap tokenization routines

  includes implementation of tokenize_text_utterly """

def lex_word(text):
    token = text[0]
    i = 1
    if i < len(text):
        ch = text[i]
        # while (ch.isalpha() or ch.isdigit() or (ch == "'")) and (i < len(text)):
        while (ch.isalpha() or ch.isdigit()) and (i < len(text)):
            token = token + ch
            i+=1
            if i < len(text):
```

```
                ch = text[i]
    return token, text[i:]


def tokenize_text_utterly(text):
    tokens = []
    rest = text
    try:
        ch = rest[0]
        while len(rest) > 0:
            if ch.isalpha():
                token,rest = lex_word(rest)
                tokens.append(token)
            else:
                tokens.append(ch)
                rest = rest[1:]
            if len(rest) > 0:
                ch = rest[0]
        return tokens
    except IndexError, e:
        print("%s: %s" % (e,text))

# tokenize_text_utterly('For 8-bit strings, this method is locale-dependent.')

def is_ws_word(astring):
    return astring.isalnum() & (len(astring) > 1)

def ends_with_s(astring):
    return astring[-1] == 's'
```

7.2.5 nls_strings.py

```
"""
File:     nls_strings.py
Module:   NLS Strings
Author:   Lan (translated to Python by Willie Rogers
Purpose:  Provide miscellaneous string manipulation routines.
Source:   strings_lra.pl
"""
import lex
import sets
import sys
from metamap_tokenization import tokenize_text_utterly

def normalized_syntactic_uninvert_string(pstring):
    """ normalized version of uninverted string. """
    normstring = normalize_string(pstring)
    normuninvstring = syntactic_uninvert_string(normstring)
```

```
    return normuninvstring

def normalize_string(astring):
    """ Normalize string. Elminate multiple meaning designators and
    "NOS" strings.  First eliminates multiple meaning designators
    (<n>) and then eliminates all forms of NOS. """
    string1 = eliminate_multiple_meaning_designator_string(astring)
    norm_string = eliminate_nos_string(string1)
    if (len(norm_string.strip()) > 0):
return norm_string;
    else:
return astring

def eliminate_multiple_meaning_designator_string(astring):
    """ Remove multiple meaning designators; method removes an expression
    of the form <n> where n is an integer from input string.  The modified
    string is returned. """
    try:
        if (astring.find('<') >= 0) & (astring.find('>') > astring.find('<')):
            if astring[astring.find('<') + 1:astring.find('>')].isdigit():
                return astring[0:astring.index('<')] + eliminate_multiple_meaning_designator_st
            else:
                return astring
        else:
            return astring
    except ValueError, e:
        sys.stderr.write('%s: astring="%s"\n' % (e,astring))
        sys.exit()

def eliminate_nos_string(astring):
    """ Eliminate NOS String if present. """
    norm_string0 = eliminate_nos_acros(astring)
    return eliminate_nos_expansion(norm_string0).lower()

nos_strings = [
    ", NOS",
    " - NOS",
    " NOS",
    ".NOS",
    " - (NOS)",
    " (NOS)",
    "/NOS",
    "_NOS",
    ",NOS",
    "-NOS",
    ")NOS"
]
# "; NOS",
```

```
def eliminate_nos_acros(astring):

    # split_string_backtrack(String,"NOS",Left,Right),
    charindex = astring.find("NOS");
    if charindex >= 0:
        left = astring[0:max(charindex, 0)]
        right = astring[charindex+3: max(charindex+3, len(astring))]
        if ((len(right) != 0) and right[0].isalnum()) and ((len(left) != 0) and left[len(left)-
            charindex = astring.find("NOS", charindex+1)
            if charindex == -1:
                return astring;

    for nos_string in nos_strings:
        charindex = astring.find(nos_string)
        if charindex >= 0:
            left2 = astring[0: max(charindex, 0)]
            right2 = astring[charindex+len(nos_string):max(charindex+len(nos_string),len(astrin

            if nos_string == ")NOS":
                return eliminate_nos_acros(left2 + ")" + right2)
            elif nos_string == ".NOS":
                return eliminate_nos_acros(left2 + "." + right2);
            elif (nos_string == " NOS"):
                if len(right2) > 0:
                    if right2[0].isalnum():
                        return left2 + " NOS" + eliminate_nos_acros(right2)
                if not abgn_form(astring[charindex+1:]):
                    return eliminate_nos_acros(left2 + right2)
            elif nos_string == "-NOS":
                return eliminate_nos_acros(left2 + right2)
            else:
                return eliminate_nos_acros(left2 + right2);
    return astring;


abgn_forms = sets.Set([
    "ANB-NOS",
    "ANB NOS",
    "C NOS",
    "CL NOS",
    # is this right?
    # C0410315|ENG|s|L0753752|PF|S0970087|Oth.inf.+bone dis-NOS|
    "NOS AB",
    "NOS AB:ACNC:PT:SER^DONOR:ORD:AGGL",
    "NOS-ABN",
    "NOS ABN",
```

```
    "NOS-AG",
    "NOS AG",
    "NOS ANB",
    "NOS-ANTIBODY",
    "NOS ANTIBODY",
    "NOS-ANTIGEN",
    "NOS ANTIGEN",
    "NOS GENE",
    "NOS NRM",
    "NOS PROTEIN",
    "NOS-RELATED ANTIGEN",
    "NOS RELATED ANTIGEN",
    "NOS1 GENE PRODUCT",
    "NOS2 GENE PRODUCT",
    "NOS3 GENE PRODUCT",
    "NOS MARGN",
])

def abgn_form(astr):
    """
    Determine if a pattern of the form: "NOS ANTIBODY", "NOS AB", etc.
    exists.  if so return true.
    """
    return astr in abgn_forms

def syntactic_uninvert_string(astring):
    """ invert strings of the form "word1, word2" to "word2 word1"
        if no prepositions or conjunction are present.

        syntactic_uninvert_string calls lex.uninvert on String if it
        contains ", " and does not contain a preposition or
        conjunction."""
    if contains_prep_or_conj(astring):
return astring
    else:
return lex.uninvert(astring)


prep_or_conj = sets.Set([
    # init preposition and conjunctions
    "aboard",
    "about",
    "across",
    "after",
    "against",
    "aka",
    "albeit",
    "along",
```

```
    "alongside",
    "although",
    "amid",
    "amidst",
    "among",
    "amongst",
    "and",
    # "anti",
    "around",
    "as",
    "astride",
    "at",
    "atop",
    # "bar",
    "because",
    "before",
    "beneath",
    "beside",
    "besides",
    "between",
    "but",
    "by",
    "circa",
    "contra",
    "despite",
    # "down",
    "during",
    "ex",
    "except",
    "excluding",
    "failing",
    "following",
    "for",
    "from",
    "given",
    "if",
    "in",
    "inside",
    "into",
    "less",
    "lest",
    # "like",
    # "mid",
    "minus",
    # "near",
    "nearby",
    "neath",
    "nor",
```

```
    "notwithstanding",
    "of",
    # "off",
    "on",
    "once",
    # "only",
    "onto",
    "or",
    # "out",
    # "past",
    "pending",
    "per",
    # "plus",
    "provided",
    "providing",
    "regarding",
    "respecting",
    # "round",
    "sans",
    "sensu",
    "since",
    "so",
    "suppose",
    "supposing",
    "than",
    "though",
    "throughout",
    "to",
    "toward",
    "towards",
    "under",
    "underneath",
    "unless",
    "unlike",
    "until",
    "unto",
    "upon",
    "upside",
    "versus",
    "vs",
    "w",
    "wanting",
    "when",
    "whenever",
    "where",
    "whereas",
    "wherein",
    "whereof",
```

```
    "whereupon",
    "wherever",
    "whether",
    "while",
    "whilst",
    "with",
    "within",
    "without",
    # "worth",
    "yet",
])


def contains_prep_or_conj(astring):
    for token in tokenize_text_utterly(astring):
        if token in prep_or_conj:
            return True
    return False



nos_expansion_string = [
    ", not otherwise specified",
    "; not otherwise specified",
    ", but not otherwise specified",
    " but not otherwise specified",
    " not otherwise specified",
    ", not elsewhere specified",
    "; not elsewhere specified",
    " not elsewhere specified",
    "not elsewhere specified"
  ]

def eliminate_nos_expansion(astring):
    """ Eliminate any expansions of NOS """

    lcString = astring.lower()
    for expansion in nos_expansion_string:
        charindex = lcString.find(expansion)
        if charindex == 0:
            return eliminate_nos_expansion(lcString[len(expansion):])
        elif charindex > 0:
            return eliminate_nos_expansion(lcString[0:charindex] +
                                           lcString[charindex + len(expansion):])

    return astring
```

7.2.6 lex.py

```
import sys

COMMA = ','
SPACE = ' '


def uninvert(s):
    """
    Recursively uninverts a string. I.e., injury, abdominal ==> abdominal injury

    Translated directly from uninvert(s,t) in lex.c.

    @param s INPUT: string "s" containing the term to be uninverted.
    @return OUTPUT: string containing the uninverted string.

    """
    if len(s) == 0:
        return s
    sp = s.find(COMMA)
    while sp > 0:
        cp = sp
        cp+=1
        if cp < len(s) and s[cp] == SPACE:
            while cp < len(s) and s[cp] == SPACE:
                cp+=1
            return uninvert(s[cp:]) + " " + s[0:sp].strip()
        else:
            sp+=1
        sp = s.find(COMMA, sp)
    return s.strip();

if __name__ == '__main__':
    if len(sys.argv) > 1:
        print('%s -> %s' % (sys.argv[1], uninvert(sys.argv[1])))
```