Thea
A Web Ontology Language - OWL Library for [SWI] Prolog.

Vangelis Vassiliadis
SemanticWeb.gr
Greece
E-mail:vangelis@semanticweb.gr

Release Notes.

Thea is a Prolog library for generating and manipulating OWL (Web Ontology Language) content. **Thea** version 0.5.5 consists of:

- Thea OWL parser,
- Thea OWL generator,
- Thea SQL to OWL converter and
- Thea OWL reasoner.

Thea OWL parser uses SWI-Prolog's Semantic Web library for parsing RDF/XML serialisations of OWL documents into RDF triples and then it builds a representation of the OWL ontology as it is defined in the OWL Web Ontology Language Semantics and Abstract Syntax part of the OWL specification. The OWL ontology abstract syntax is implemented as Prolog terms.
Thea has been tested extensively against the OWL test cases and for almost all cases it generates the correct syntactic constructs.

Thea OWL generator is used to the OWL abstract syntax constructs from Prolog terms into RDF triples and saving the resulting RDF model into an RDF/XML file. Thea OWL generator is also using SWI-Prolog's Semantic Web library for saving RDF models into RDF/XML files.

Thea SQL to OWL converter is used to generate OWL facts from records in a relational database. **SQL2OWL** uses SWI-Prolog's ODBC package to access the RDBMS. The conversion is guided by a mapping between Relational entities (Tables and Columns) and OWL constructs (Classes and Properties). The mapping is defined in a declarative form by means of Prolog terms.

Thea OWL reasoner is the newest module of Thea. It contains two sub modules:
a) A OWL abstract terms to Prolog converter based on the concept of DLP[1] (Description Logic Programs) and
b) A Prolog wrapper to the DIG[2] interface so that a Thea ontology can communicate with a DIG-enabled reasoning engine.

Current version of **Thea** OWL Prolog library (v 0.5.5) released April 4, 2007. **Thea** is developed by Vangelis Vassiliadis and is available under the GNU/GPL license.

Downloads

You can download **Thea** from its home page at **www.semanticweb.gr/TheaOWLLib**

Version 0.5.5 Changes

- OWL reasoner module added.
- Bugs fixed in owl_parser module (correct handling of differentIndividuals sets).

Version 0.5 Changes

- SQL to OWL converter was added to Thea OWL library.

Known issues and limitations:

- **Thea** parses all 'versions' of OWL ontologies (Full/DL/Lite) but does it is not currently a 'species' validator.
- The value of an individual is not tested for structure sharing.
- The SQL to OWL converter is based on a simple declarative mapping.

Abstract

Thea is a Prolog library for manipulating OWL (Web Ontology Language) content. **Thea** version 0.5.5 consists of:

- Thea OWL parser,
- Thea OWL generator and
- Thea SQL to OWL converter and
- Thea OWL reasoner.

Thea OWL parser uses SWI-Prolog's Semantic Web library for parsing RDF/XML serialisations of OWL documents into RDF triples and then it builds a representation of the OWL ontology as it is defined in the OWL Web Ontology Language Semantics and Abstract Syntax part of the OWL specification. The OWL ontology abstract syntax is implemented as Prolog terms.

Thea OWL generator is used to the OWL abstract syntax constructs from Prolog terms into RDF triples and saving the resulting RDF model into an RDF/XML file. Thea OWL generator is also using SWI-Prolog's Semantic Web library for saving RDF models into RDF/XML files.

Thea SQL to OWL converter is used to generate OWL facts from records in a relational database. **SQL2OWL** uses SWI-Prolog's ODBC package to access the RDBMS. The conversion is guided by a mapping between Relational entities (Tables and Columns) and OWL constructs (Classes and Properties). The mapping is defined in a declarative form by means of Prolog terms.

Thea OWL reasoner consists of two sub modules:

a) A OWL abstract terms to Prolog converter based on the concept of DLP (Description Logic Programs) and

b) A Prolog wrapper to the DIG interface so that a Thea ontology can communicate with a DIG-enabled reasoning engine.

Table of Contents

1. Introduction

Thea is a Prolog library for manipulating OWL (Web Ontology Language) content. **Thea** version 0.5 consists of:

- Thea OWL parser,
- Thea OWL generator and
- Thea SQL to OWL converter.

Thea OWL parser uses SWI-Prolog's Semantic Web library for parsing RDF/XML serialisations of OWL documents into RDF triples and then it builds a representation of the OWL ontology as it is defined in the OWL Web Ontology Language Semantics and Abstract Syntax part of the OWL specification. The OWL ontology abstract syntax is implemented as Prolog terms.

Thea OWL generator is used to the OWL abstract syntax constructs from Prolog terms into RDF triples and saving the resulting RDF model into an RDF/XML file. Thea OWL generator is also using SWI-Prolog's Semantic Web library for saving RDF models into RDF/XML files.

Thea SQL to OWL converter is used to generate OWL facts from records in a relational database. **SQL2OWL** uses SWI-Prolog's ODBC package to access the RDBMS. The conversion is guided by a mapping between Relational entities (Tables and Columns) and OWL constructs (Classes and Properties). The mapping is defined in a declarative form by means of Prolog terms.

Thea OWL reasoner consists of two sub modules:
a) A OWL abstract terms to Prolog converter based on the concept of DLP (Description Logic Programs) and
b) A Prolog wrapper to the DIG interface so that a Thea ontology can communicate with a DIG-enabled reasoning engine.

Current version of **Thea** OWL Prolog library (v 0.5.5) released April 4, 2007. **Thea** is semanticweb.gr project developed by Vangelis Vassiliadis and is available under the GNU/GPL license.

2. The Abstract Syntax representation

The result of parsing is an OWL ontology abstract syntax representation in the form of Prolog terms as defined below:

```
- ontology(OntologyID,AnnotationList).

- class(ClassID, Deprecated, Modality,
            AnnotationList, DescriptionList).
```

```
-  subclassOf(Description1, Description2).

-  disjointSet(DescriptionList).
-  equivalentSet(DescriptionList).

-  property(PropertyID,Deprecated,
             AnnotationList,SuperPropertyList,
             PropertyTypeList, DomainList, RangeList).

-  annotationProperty(PropertyID).

-  individual(IndividualID,AnnotationList,
             TypeList,PropertyValueList).

-  differentIndividuals(IndividualList).

-  sameIndividuals(IndividualList).
```

Where Description can be any of

```
ClassID
Restriction
intersectionOf(DescriptionList)
unionOf(DescriptionList)
complementOf(Description)
oneOf(IndividualList)
```

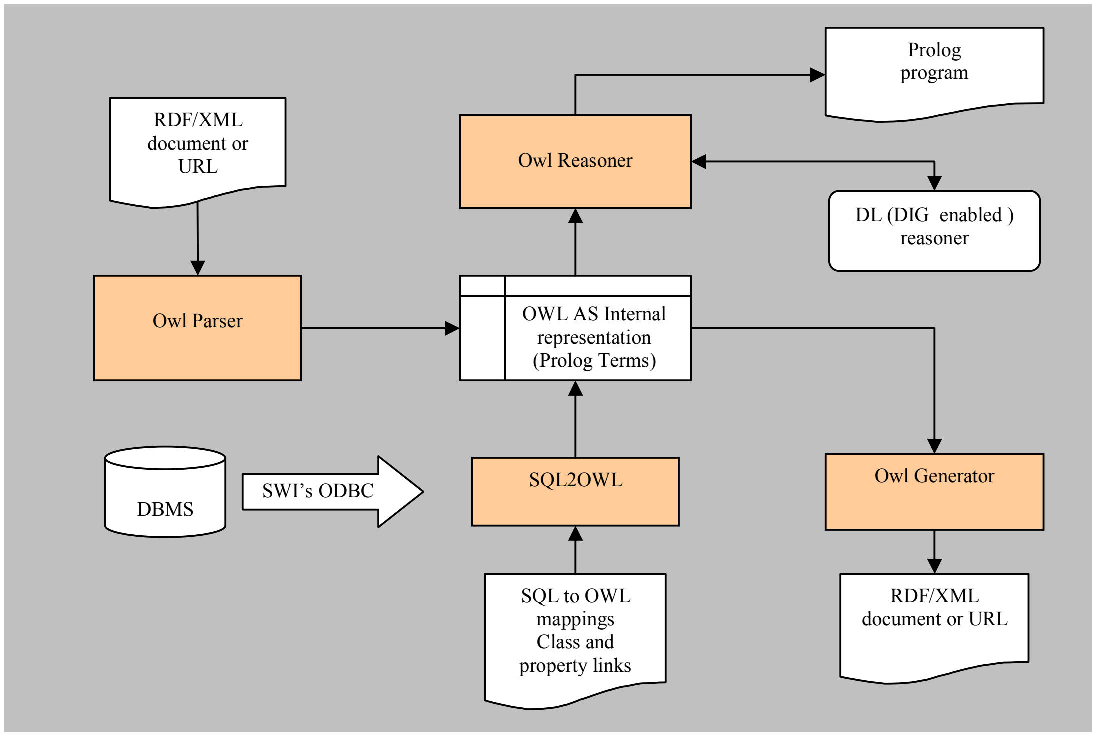
and Restriction can be any of

```
restriction(PropertyID,allValuesFrom(Description))
restriction(PropertyID,someValuesFrom(Description))
restriction(PropertyID,cardinality(C))
restriction(PropertyID,maxcardinality(C))
restriction(PropertyID,mincardinality(C))
restriction(PropertyID,value(V))
```

The semantics of the above representation is straightforward as it matches the OWL abstract syntax as defined in the OWL Web Ontology Language Semantics and Abstract Syntax.

3. Architecture

The overall architecture of the library is shown in the following figure:



3.1 Parser architecture

The **Thea** OWL parser follows a parsing strategy similar to the one described in OWL Web Ontology Language Parsing OWL in RDF/XML.[3] Initially all rdf (S, P, O) triples are copied into the owl(S, P, O, not_used) terms. The parser works by searching for `owl/4` terms that can be used to construct OWL ontology abstract facts and axioms based on the transformation rules defined in OWL Web Ontology Language Semantics and Abstract Syntax.

Every `owl/4` term used in as transformation is marked as 'used' (i.e. retracted from the Prolog database and asserted again as owl(S,P,O,used). The parser terminates when no other constructs can be build from the remaining triples. If any owl triples remain unused this is an indication of an 'external' parsing error, i.e. there are missing RDF triples that prevent the parser from using all triples to create constructs. Examples:

- If an owl:Restriction property is missing, any associated owl:onProperty triples will remain unused.
- Whenever the blank nodes in a class description in OWL DL form a directed cycle. (See OWL test at http://www.w3.org/2002/03owlt/I5.26/consistent006)

The parser tries to create constructs in the following sequence (in parenthesis the Prolog library predicates):

```
-  Classes and class descriptions (owl_parse_named_classes)
-  Class axioms
      Subclasses (owl_parse_subclasses)
      DisjointWith (owl_parse_disjoint_classes)
      EquivalentClasses (owl_parse_equivalent_classes)
-  Properties (owl_parse_property)
-  Annotation Properties (owl_parse_annotationProperty)
-  Individual axioms    (owl_parse_individual_axioms)
      SameAs
      DifferentFrom
-  Ontology definitions (owl_parse_ontology)
-  Names Individuals          (owl_parse_named_individuals)
-  Unamed Classes       (owl_parse_unnamed_classes)
-  Unamned Individuals (owl_parse_unnamed_individuals)
```

In addition to the Prolog terms that construct the OWL abstract syntax, **Thea** parser defines the following as dynamic predicates:

- owl/4: For storing and tracking the use of individual triples).
- blanknode/3: For storing and tracking the use of blank nodes, and thus be able to detect any structure sharing).
- owl_parser_log/2: For logging OWL parser's activity.

3.2 SQL to OWL converter[4].

3.2.1 The mapping declarations (links).

The data conversion from Relational database to OWL facts is guided by a set of links that define the relationship between RDBMS concepts (Tables and Columns) on the one hand and OWL Ontology concept on the other (Class and Properties). Two types of links are defined:

1. **Class link**: A class link defines a mapping between a Table in the RDBMS with a class in the OWL ontology. The syntax of a class link is:

   ```
   class_link(Class, Table, Column_PK).
   ```

   Examples:

   - `class_link('Person', 'swc_researchers', rid).`
     Defines a link between a Class `Person` and a table `swc_researchers`. The column `rid` is the primary/unique key for this table. The converter will generate one OWL Individual fact for each row in the `swc_researchers` table. The ID of the Individual fact will be the value of the PK column.

   - `class_link('Organisation', 'swc_organisations', oid).`

Similarly a link between the Organisation class and the swc_organisations table.

- `class_link('http://owl.org/swc_ontology#Project','swc_projects',pid).`

This example shows that the class can be a fully namespace URL.

2. **<u>Property link</u>**: A property link defines a mapping between 2 related columns in the RDBMs, and a binary property in the OWL ontology. The syntax of a property link is:

```
property_link(+ClassOrSubject, +Property, +ClassOrObject,
Options)
```

Where

`ClassOrSubject` can be either a Class defined in one class_link or any column in the RDBMS.

`Property` is any literal or URL.

`ClassOrObject` can be either a Class defined in one class_link or any column in the RDBMS.

`Options` is a (Prolog) list containing any combination of the following terms (each term can appear 0 or 1 times) :

`sf(SF).`
`op(OP).`
`tpf(PFList)`, where PFList is a 'Prolog' list of PrimaryKey-ForeignKey terms

Each property_link is processed and produces a Subject, Property, Object relationship using the following algorithm:

The Subject is derived from the ClassOrSubject argument. If ClassOrSubject is a Class in a class_link then the Subject equals ColumnPK argument of that class_link. Otherwise ClassOrSubject is the RDBMs column itself.

- The Property is the Property argument
- The Object is derived from the ClassOrObject argument in the same way as the Subject above.
- The Options formulate the part of the SQL query that joins the Subject with the Object tables.

Examples:

- ```
  property_link('Person', 'works_for',
  'swc_organisations.title',
  [op('swc_organisations.oid'),
  tpf(['swc_researchers_organisations.rid'-
  'swc_researchers_organisations.oid'])]).
  ```
  ```

Given the `class_links` above this property link maps to the following SQL query:

```
Select concat('Person','-',swc_researchers.rid) as _IID ,
       swc_organisations.title as works_for
From
       swc_researchers_organisations,
       swc_researchers,swc_organisations
Where  swc_researchers.rid = swc_researchers_organisations.rid
and    swc_researchers_organisations.oid = swc_organisations.oid
```

- `property_link('Person', 'works_for_2', 'Organisation', [tpf(['swc_researchers_organisations.rid'-'swc_researchers_organisations.oid'])]).`

SQL Query:

```
Select concat('Person','-',swc_researchers.rid) as _IID ,
       concat('Organisation','-',swc_organisations.oid) as
       works_for_2
From
       swc_researchers_organisations,
       swc_researchers,swc_organisations
Where  swc_researchers.rid = swc_researchers_organisations.rid
  and  swc_researchers_organisations.oid = swc_organisations.oid
```

- `property_link('Person','''example:Name''','swc_researchers.name',[]).`

SQL Query:

```
Select concat('Person','-',swc_researchers.rid) as _IID ,
       swc_researchers.name as 'example:Name'
From   swc_researchers
```

- `property_link('swc_researchers.rid','Name','name',[]).`

SQL Query:

```
Select swc_researchers.rid as _IID , name as Name
From   swc_researchers
```

- `property_link('Person','Name2','name',[]).`

SQL Query:

```
Select concat('Person','-',swc_researchers.rid) as _IID ,
       name as Name2
From   swc_researchers
```

3.2.2 The converter

As shown in the examples above the class and property links are in effect a guide to an SQL pre-processor. The actual query and conversion of the RDBMS data to OWL facts is done via the following two Prolog predicates:

`populate_class(DBConnection, Class).`
This predicate formulates an SQL query quided by the class_link defined by the `Class` argument, and **all the property_links having this class as a subject.** The

query is executed against the ODBC connection `DBConnection` and the result set is asserted as OWL 'Individual' facts in the Prolog database.

`populate_property(DBConnection, Property).`
In a similar fashion the query corresponding to the property_link of the `Property is` executed and the result set is asserted as OWL 'Individual' facts in the Prolog database.

See examples in section 5.2.2

3.3 OWL Reasoner.

The OWL reasoner module of Thea consists of two sub modules,

a) the OWL to Prolog converter that implements the concept of DLP and converts an OWL ontology from Prolog abstract syntax terms into Prolog code (predicates), and

b) the OWL to DIG interface is a Prolog wrapper around the DIG specification that enables Prolog programs to call a DIG enabled reasoner and get its result back also as Prolog constructs.

3.3.1 OWL to Prolog code[5].

This sub-module is a Prolog implementation of the Description Logic Programs (DLP) concept presented in [1] where also the mapping between DL axioms and logic programs is introduced.

Usage

Top level predicate is the `owl_as2prolog( +OwlAsTerm, +Options).` It converts the Prolog OWL abstract syntax term (as parsed by the OWL parser) into Prolog logic code. **The Prolog code is written into the current output stream, so redirecting the output stream into a file before calling this predicate is suggested in order to capture the generated code.** Options are generic options to modify the behaviour of the code generation. Currently only the `no_base(Namespace)` is supported. This option tells the code generator not to prefix the Prolog predicates with the namespace prefix.

Example

Suppose we have parsed (using OWL parser) the example 'Wine' ontology. Class definitions for `WhiteWine` and `WhiteTableWine` would be (among others) in the Prolog terms:

```
class('http://www.w3.org/2002/03owlt/miscellaneous/consistent001#WhiteWine',
        false,
        complete,
        [],
        [intersectionOf(['http://www.w3.org/2002/03owlt/miscellaneous/consistent001#Wine',
        restriction('http://www.w3.org/2002/03owlt/miscellaneous/consistent001#hasColor',
        value('http://www.w3.org/2002/03owlt/miscellaneous/consistent001#White'))])]).
```

```
class('http://www.w3.org/2002/03owlt/miscellaneous/consistent001#WhiteTableWine',
        false,
        complete,
        [],
        [intersectionOf(['http://www.w3.org/2002/03owlt/miscellaneous/consistent001#TableW
        ine',
        restriction('http://www.w3.org/2002/03owlt/miscellaneous/consistent001#hasColor',
        value('http://www.w3.org/2002/03owlt/miscellaneous/consistent001#White'))])]).
```

To generate the Prolog code we call the following predicate:

```
:- C='http://www.w3.org/2002/03owlt/miscellaneous/consistent001#WhiteWine',
        class(C,A2,A3,A4,A5),
        owl_as2prolog(class(C,A2,A3,A4,A5),[no_base('wine')]).
```

The following is the result of the conversion.

```
'Wine'(X):-
     'WhiteWine'(X).
hasColor(X,'wine:White'):-
     'WhiteWine'(X).
'WhiteWine'(X):-
     'Wine'(X),hasColor(X,'wine:White').
```

Note the `no_base` option that instructs the Prolog code generator not to use the namespace part of the generated predicate terms. Quotes are automatically generated when needed (using SWI's `writeq` predicate). Calling the same predicate without the `no_base` option would result in:

```
wine_Wine(X):-
     wine_WhiteWine(X).
wine_hasColor(X,'wine:White'):-
     wine_WhiteWine(X).
wine_WhiteWine(X):-
     wine_Wine(X),wine_hasColor(X,'wine:White').
```

I.e. all generated predicates are prefixed with the namespace plus _ (instead of :).

Also the URIs are generated using the short namespace format `ns:Term`. SWI semweb package's `rdf_db:ns/2` dynamic predicate is used to store the declared namespaces. Prior to generating the Prolog code, the user should ensure all needed namespace declarations are stored in the `rdf_db:ns/2`.

The following table summarizes the mappings that are implemented in the current version.

OWL AS axiom and fact	**Prolog code generated**
Class C complete declaration with single description D	C and D equivalent → subclassOf(C,D) AND subclassOf(D,C).
Class C complete declaration with multiple descriptions DL	subclassOf(C, Map(intersectionOf(DL))).
Class C partial declaration with multiple descriptions DL	subclassOf(C,D) for each D in DL.
subclassOf(C,D)	Map(D)(X) :- Map(C)(X).
intersectionOf(DL) (only if intersection in head or body of a rule).	Map(D1),Map(D2)…Map(Dn)
unionOf(DL) (only as body of a rule or as facts)	Map(D1) ; Map (D2) ; . . . ; Map(Dn)

oneOf(IL) (only in body of rules)	member(X,IL)
Restriction(property,Value)	property(X,V)
Restriction(property,allValuesForm(D)) (head)	Map(D)(Y) :- property(X,Y).
Restriction(property,allValuesForm(D)) (fact)	Map(D)(X) :- property(ID,X)
Restriction(property,someValuesForm(D)) (body)	Map(D)(Y) , property(X,Y).
C (class URI)	classURI(X)
S is super property of P	S(X,Y) :- P(X,Y)
C in the domain of P	Map(C)(X) :- p(X,Y)
C in the range of P	Map(C)(Y) :- p(X,Y)
P is functional property	sameIndividuals(X,Y) :- p(Z,X),p(Z,Y).
P is inverse functional property	sameIndividuals(X,Y) :- p(X,Z),p(Y,Z).
P is a transitive property	p(X,Z) :- p(X,Y), p(Y,Z).
P is a symmetric property	p(X,Y) :- p(Y,X)
P is the inverse of Q	p(X,Y) :- q(Y,X). q(X,Y) :- p(Y,X)
individual(IID,_,DescriptionList,ValueList)	Map(D)(IID) for each D in DescriptionList p(IID,V) for each V in ValueList

More conversion examples are given in section 5.3

3.3.2 OWL DIG Wrapper.

The OWL to DIG interface is a Prolog wrapper around the DIG specification that enables Prolog programs to call a DIG enabled reasoner and get its result back also as Prolog constructs.

There are the following 6 top-level predicates that implement the respective DIG commands:

1. `dig_reasoner_id(+ReasonerURL, -Response)`
To request the Reasoner's Identification. Response is the XML response by the reasoner.

2. `dig_new_kb(+ReasonerURL, +NewKB, -Result)`
Request to create a new KB named NewKB. The Result is the XML response by the reasoner. The name and the ID of the KB assigned by the reasoner are related by the dynamic dig_kb/2 predicate

3. `dig_release_kb(+ReasonerURL, +KBName, -Result)`
Request to release a KB.

4. `dig_tell(+ReasonerURL,+KBName,+Tells,-Response)`
Sends a TELL request to the reasoner containing all the requests in the Tells list. The Response is the XML response by the reasoner.

5. `dig_ask(+ReasonerURL,+KBName,+Query,-Result)`
Sends an ASK request to the reasoner about the KBName knowledge base. The Query contains the request in DIG's ASK language. The result is translated in Prolog terms according to the predicate list presented in section 4.4 below.

Example
(Assuming the food ontology is already parsed using Thea OWL parser).

`:-dig_new_kb('http://localhost:8081',food,X).`

```
X = [element(response, [xmlns='http://dl.kr.org/dig/2003/02/lang',
'xmlns:xsi'='http://www.w3.org/2001/XMLSchema-instance',
'xsi:schemaLocation'='http://dl.kr.org/dig/2003/02/lang http://dl-
web.man.ac.uk/dig/2003/02/dig.xsd'], [element(kb, [uri='urn:dig:pellet:kb-
4eb1341:1119d250bef:-8000'], [])])]
```

`:- dig_tell_all('http://localhost:8081',food,X).`

```
X = [element(response, [xmlns='http://dl.kr.org/dig/2003/02/lang',
'xmlns:xsi'='http://www.w3.org/2001/XMLSchema-instance',
'xsi:schemaLocation'='http://dl.kr.org/dig/2003/02/lang http://dl-
web.man.ac.uk/dig/2003/02/dig.xsd'], [element(ok, [], [])])]
```

Where **dig_tell_all** is defined as

```
dig_tell_all(ReasonerURL,KBName,X) :-
        findall(Rc,(class(C1,C2,C3,C4,C5), owl_as2dig(class(C1,C2,C3,C4,C5),Rc)),LRC),
        findall(Rsc,(subclassOf(SC1,SC2), owl_as2dig(subclassOf(SC1,SC2),Rsc)),LRSC),
        findall(Ri,(individual(A,B,C,D),owl_as2dig(individual(A,B,C,D),Ri)),LRI),
        findall(Rp,(property(A1,A2,A3,A4,A5,A6,A7),owl_as2dig(property(A1,A2,A3,A4,A5,A6,A
7),Rp)),LRP),
        flatten([LRC,LRSC,LRI,LRP],RF8),
        dig_tell(ReasonerURL,KBName,RF8,X).
```

`:- dig_ask('http://localhost:8081',allIndividuals,X).`

```
X = ['http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#White',
'http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#Pork',
'http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#FraDiavolo',
'http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#OffDry',
'http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#Lobster',
'http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#Scrod',
'http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#Steak',
'http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#Strong',
'http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#MixedFruit'|...]
```

4. Library Predicates

4.1 Parser

Top Level Predicates

owl_parse(+URL, +RDF_Load_Mode, +OWL_Parse_Mode, +Imports).
owl_parse(+OWL_Parse_Mode).

owl_pack_ontology.
owl_report.

UTILITY Predicates

owl_parser_log(+Log):
Log is a list; together with a timestamp it is asserted as an owl_parser_log/2 term.

owl_clear_as.
Clears the prolog terms that store the Abstract Syntax implementation of the OWL ontology.

convert(T,V,typed_value(T,V)).

rdf_2_owl.
Converts RDF triples to OWL/4 triples so that their use can tracked by the OWL parser.

rdf_load_stream(+URL, +ImportedList)
This predicate calls the rdf parser to parse the RDF/XML URL into RDF triples. URL can be a local file or a URL. The predicate recursively calls itself for all URLs that need to be imported, ie. are objects to an owl:imports predicate. The ImportedList argument contains the imported so far URLs, to avoid re-visiting the same URLs. (Empty List in 1st call).

fix_no(+A,-B).
This is used to correct an RDF parser error: To remove duplicate ## from a URL.
Obsolete with version 5.5.x of SWI's RDF parser

owl_count(?U).
Returns/Checks the number of unused OWL triples.

use_owl(?S,?P,?O).
Marks an OWL triple as used. Expands the S,P,O.

use_owl(?S,?P,?O,named).
Same as use_owl/3, but marks only if S is Named URI (i.e. non blank node).

expand_ns(+NS_URL, ?Full_URL).

Expands a 'namespaced' URI of the form ns:fragment to a full URI substituting the full expansion for ns from the ns/2 facts

collapse_ns(+Full_URL, ?NS_URL).

Collapses a full URI of the form Path#fragment to a Namespaced URI NS:fragment substituting the full expansion for ns from the ns/2 facts

uri_split(+URI,-Namespace,-Term,+Split_Char).

Splits a URI into the Namespace and the Term parts separated by the Split_Char character.
It supposes URI = concat(Namespace, Split_Char, Term)

owl_collect_linked_nodes(+Node,+Predicate, +InList,-OutList).

Appends Node to the InList, and recursively, all other Nodes that are linked with the Predicate to the Node. The result is returned to OutList.

OWL Parser implementation predicates

owl_deprecated_class(+CID,-Deprecated).

Deprecated is set to true if Class CID is defined as deprecated. false otherwise.

owl_deprecated_property(+PID,-Deprecated).

Deprecated is set to true if Property PID is defined as deprecated; false otherwise.

owl_get_bnode(+Node,+Description).

if Node is a blank (not named) node, then it is asserted in the database as a blanknode(Node,Description,used) term. The purpose is to record when a blank node has been used, so subsequent uses of it will result in structure sharing.

owl_optional_type(+D).

It simply consumes any optional owl:Class or rdfs:Class type triples for description D

owl_description_list(+Node, -List).

If +Node is defined as rdf:type rdf:List, then List returns a prolog list of descriptions for this Node.

owl_individual_list(+Node, -List).

If +Node is defined as rdf:type rdf:List, then List returns a prolog list of individuals for this Node.

owl_restriction(+Element,-Restriction).

If Element is defined as a owl:Restriction on property P then Restriction binds to a restriction(Property,Type) term, according to OWL Abstract syntax specification.

owl_description(+Node,-Description).

It implements OWL AS production rules for Descriptions.
I.e. a Description can be any of

- a Class ID
- an existing blank node (in which case we have structure

sharing),
- a unionOf(DescriptionList) term.
- a intersectionOf(DescriptionList) term.
- a complementOf(Description) term.
- a oneOf(IndividualList) term.

During the construction of the Description any blank node is recorded for later structure sharing checks.

Classes, SubClasses and Class axioms

owl_parse_named_classes.

Any named node defined as an owl:Class or rdfs:Class is asserted int the database as a class/5 term with all Descriptions and annotations defined for this Class ID Note that the construction of a class term cannot be done incrementally, i.e. we cannot add descriptions or annotations to an existing class.

owl_parse_unnamed_classes

Same as above for unnamed Classes. (Not in OWL DL)

owl_parse_subclasses.

Asserts a subclassOf(DescriptionX,DescriptionY) term for each X rdfs:subClassOf Y triple.

owl_parse_equivalent_classes.

Asserts an equivalentSet(DescriptionList) term for each set of connected with owl:equivalentClass Nodes. DescriptionList is a list of Descriptions for these Nodes.

owl_parse_disjoint_classes.

Constructs Disjoint Sets for nodes that are connected with owl:disjointWith links.

owl_disjoint_arcs(+ArcList).

ArcList contains a list of A-B elements (arcs) where A and B are owl:disjointWith Classes/descriptions. Predicate constructs disjointSet(DescriptionList) of Node descriptions from these arcs such that the sets are the largest possible sets of mutually disjoint Nodes. Blank nodes are used only once, named nodes can be re-used.

owl_set_descriptions(+NodeList,-DescriptionList).

Get the description for each node in the NodeList. If no descrition exists, return the Node it self....

owl_disjoint_nodes(+InNodeList,-OutNodeList, +InList,-OutList).

Calls owl_disjoint_node (see below) for each Node in InNodeList.

owl_disjoint_node(?Node,-NodeList, +InList,-OutList).

InList contains a list of A-B elements where A and B are owl:disjointWith Classes/descriptions. The predicate appends Node into NodeList if there is a X in NodeList such that Node-X (or X-Node) exists in InList.In such a case Node-X (or X-Node) are removed from InList (only if either X or Node are blank noded) and the resulted list is OutList.

owl_remove(+Element,+InList,-OutList).

Outlist is InList with all occurences of Element removed.

owl_remove_sym(A-B,+InList,-OutList).

Element is expected to be in the form A-B where A and B are owl:disjointWith Classes/descriptions. The predicate removes from InList all occurences of A-B and B-A in case either A or B are blank (unamed) nodes. The resulted list is OutList.

Properties

owl_parse_property.

Any named node defined as having rdf:type any of the OWL defined property types (e.g. Object, Datatype, Functional, etc) is asserted into the database as a property/7 term with all super properties, annotations, range and domain information defined for this Property ID Note that the construction of a property term cannot be done incrementally, i.e. we cannot add ranges, domains or annotations to an existing property.

```
property(PropertyId,
      Deprecated(true/false),
      AnnotationsList
      SuperPropertyList
      PropertyTypeList
      DomainList  (DescriptionList)
      RangeList   (DescriptionList))
```

The second clause collects the equivalentProperty axioms.

owl_parse_property_type(-PID,+[-OT,-F,-IF,-T,-S,iof(-Inv)]).

Returns a PropertyID and the correct property type as a list of atoms describing the property type.

owl_annotation(+C,annotation(-APID,-Value).

For a given name id (C) it returns an annotation construct. APID is either an existing annotation Property, or it is a new one. Predefined annotation properties are rdfs:comment, rdfs:label, rdfs:seeAlso.

owl_parse_annotationPropery.

It creates an annotationProperty term for each occurence of an owl:AnnotationProperty typed ID. Range properies for annotation are not processed yet.

Ontology

get_ontology.

Parses owl:Ontology types and creates ontology/2 terms as part of abstract syntax. of core owl_parser, parses the Ontology properties and annotations.

get_ontology_annotation(+P,annotation(-P,-Value)).

Parses OntologyProperties and ontology-specific annotation properties: owl:versionInfo, rdfs:isDefineBy, owl:imports, owl:backwardCompatibleWith, owl:priorVersion.

Individuals & Individual Axioms (AllDifferent, differentFrom, sameAs)

owl_parse_named_individuals

Any named node not defined as an individual is sserted into the database as a individula/5 term with all types, properties and annotations defined with this named individual as a subject. Note that the construction of an individual term cannot be done incrementally, i.e. we cannot add types, properties or annotations to an existing individual.

owl_parse_unnamed_individuals.

Same as above for unnamed individuals.

owl_parse_individual_axioms.

Handles the owl:AllDiferent axiom by asserting a differentIndividuals(List) prolog term where List is a list of different individuals. The second clause of this term handles the differentFrom construct.

owl_parse_individual_axioms_2.

Handles the owl:sameAs axiom by asserting a sameIndividuals(List) prolog term where List is a list of sam individuals.

owl_different_arcs(+ArcList).

ArcList contains a list of A-B elements (arcs) where A and B are owl:differentFrom Individuals. Predicate constructs disjointIndividuals(IndividualList) of Nodes from these arcs such that the sets are the largest possible sets of mutually different Nodes. Blank nodes are used only once, named nodes can be re-used.

owl_different_nodes(+InNodeList,-OutNodeList, +InList,-OutList).

Calls owl_different_node (see below) for each Node in InNodeList.

owl_different_node(?Node,-NodeList, +InList,-OutList).

InList contains a list of A-B elements where A and B are owl:differentFrom Individuals. The predicate appends Node into NodeList if there is a X in NodeList such that Node-X (or X-Node) exists in InList. In such a case Node-X (or X-Node) are removed from InList (only if either X or Node are blank noded) and the resulted list is OutList.

4.2 RDF Generator

owl_generate_rdf(+FileName, +RDF_Load_Mode)

Top level predicate to generate an RDF/XML FileName from the existing OWLAS predicates of the Thea OWL library. If the +RDF_Load_Mode is 'complete' then all existing RDF triples are first removed.

owl_rdf2n3

Prints out the RDF triples in N3 notation.

owl_as2rdf_class.

Generates RDF triples for Class constructs.

owl_as2rdf_class_2(+Class, +CP, +DescriptionList).
owl_as2rdf_subclass.

Generates RDF triples for SsubClass constructs.

owl_as2rdf_equivalentSet.
Generates RDF triples for equivalentSet constructs.

owl_as2rdf_equivalentSet_2.
Generates RDF triples for equivalentClass and equivalentProperty constructs.

owl_as2rdf_disjointSet.
Generates RDF triples for disjointSet constructs.

owl_as2rdf_differentIndividuals
Generates RDF triples for differentIndividuals constructs.

owl_as2rdf_sameIndividuals
Generates RDF triples for sameIndividuals constructs.

owl_as2rdf_set2pairs(+Set,+Predicate).
Given a list (Set) it generates (X Predicate Y) rdf triples for all X, Y elements in the list

owl_as2rdf_set2pairs(+X, +Set,+Predicate).
It generates (X Predicate Y) rdf triples for all Y elements in the list Set. X and Y are converted to RDF Nodes first.

owl_as2rdf_property.
Generates RDF triples for Property constructs.

owl_as2rdf_property_2(+PId, +Deprecated, +PropertyTypeSet)
Generates RDF triples for property PID based on the type of property as defined in the PropertTypeSet options list

owl_as2rdf_individual.
Generates RDF triples for individual constructs.

owl_as2rdf_ontology.
Generates RDF triples for ontology constructs.

owl_as2rdf_triple_list(+ID,+Predicate,+List).
Generates RDF triples of the form (ID, Predicate, Y) where Y is each element in List. Nodes are generated for each element Y and specific cases for elements Y are handled. Eg Y=type(T) or Y = value(P,V) or Y = annotation(P,V).

owl_as2rdf(+Construct, -Node).
Generates RDF triples for the Construct based on AS transformation rules. If not existing a blankNode is also generated to represent the construct. A Construct can be any Description (incl Restrictions), URL, blanknode or literal.

owl_as2rdf_list(+List, -Node).
Generates RDF triples for the List of construct based on Abstract Syntax list transformation rules. Node represents the List in the resulting RDF graph

owl_rdf_assert(+S,+P,+O).
Expands the NS the S, P, O terms and asserts into the RDF database

owl_as2rdf_bnode(+X,-Node).

It generates a bnode Node for construct X in case it does not exist already as a blanknode/3 clause.

4.3 SQL to OWL converter

populate_class(+DBConnection, +Class)

It constructs and executes an SQL query (against a SWIs ODBC-package DBConnection), based on the class_link and any property_links for this class. For each row, it creates one Individual of class Class. Assigns Property-Value pairs to this Individual as defined by the property_links having Class as SubjectClass.
NOTE. All records of the Table linked to Class will be populated. No filter is possible in this version. There is always to workaround linking the class with a View/filter iso a Table.

populate_property(+DBConnection, +Property)

It constructs and executes an SQL query (against a SWIs ODBC-package DBConnection), based on the property_link for the Property. It creates one Individual of class as defined in the Subject property_link for each row returned. Assigns a Property-Value pair to this Individual as defined by the property_link.
NOTE: Use only for properties having link with a 'Class' Subject

process_class_link(+Class, -CTW)

Processes all property_links of a Class and returns a CTW term, containing the SELECT, FROM and WHERE elements of an SQL query, to be used for populating Individuals with this class link.

execute_sql(+DBConnection, +Class, +SQL_Query)

Executes SQL_Query against SWI-Prolog's ODBC package DBConnection.
For each row returned, it asserts an 'Individual' fact.

make_individual_from_row(+ColumnList,-ID,-PVList)

Converts a column(_,Column,Value) list (the results of the odbc_query) into a Property-Value pair list PVList.
It treats the Value of the Column named _IID as the individual's identifier (ID) and not as a property value.

process_property_link(+ClassOrSubject, +PropertyName, +ClassOrObject, +Options, -CTW)

Processes the elenents of a property_link (Subject, Property, Objectm Options) and returns a CTW term, containing the SELECT, FROM and WHERE elements of an SQL query, to be used for populating Individuals with this property link.

process_where_list/4, process_where_list/3, build_from_list/5

Utility predicates building the FROM and WHERE elements of a CTW term. Called by process property link.

ctw_to_sql(+CTW,-SQL_String)

Converts a CTW term containing the SELECT, FROM and WHERE elements of an SQL query, to an SQL Query string ready for execution by ODBC package.

expand_sql_list(+List,-CSList).

Converts a List into a comma separated list. Used by build_sql_class to create the list of tables in the FROM clause of the query

expand_sql_list(+XYList,-ListCS, +Operator, +Separator).

Converts a List with X-Y elements into a list with elements X Operator X separated by Separator. Used by build_sql_class to create the SELECT and the WHERE clauses of the query

merge_ctw(+CTW_List, -Merged_CTW_List)

The CTW_list is a list with ctw(C,T,W) elements.
The result is a one-element ctw(Cm,Tm,Wm) list where Cm,Tm and Wm are the merged lists of all C, T and W respectively.
Used by build_class_sql to merge the SELECT, FROM and WHERE clauses of the individual property_link SQLs.

list_sql(+List, -String).

Utility predicate to concat a list to its string representation

4.4 OWL Reasoner

dig_reasoner_id(+ReasonerURL, ?Response)

Identification Request from the reasoner in ReasonerURL
The Response is the XML response returned by the reasoner.

dig_new_kb(+ReasonerURL,+NewKB,?Result)

Requests a new knowledge base from the reasoner.
If successfull the URI is stored as a dig_kb(NewKB,URI) predicate.
If the NewKB already exists in dig_kb then no request is made to the reasoner.
Result is the response from the reasoner.

dig_release_kb(+ReasonerURL, +KBName, ?Result)

Requests reasoner to release existing KBName.
If successfull the KB is also removed from the

dig_kb(NewKB,URI) facts.

dig_tell(+ReasonerURL,+KBName,+Tells,?Response)

Sends a list of tell requests to the DIG reasoner.
Response is the unparsed XML response from reasoner.
KBName must be instantiated and in the dig_kb.

dig_ask(+ReasonerURL,+KBName,+Query,?Result)

Sends a DIG ask Query, expressed in DIG Ask language to the reasoner.
The query is transformed to the XML representation by the dig_ask/2 predicate.
Reasoner's response is processed by the dig_ask_response/3 and the Result is a List representation of the DIG response lagnuage.

dig_request(+ReasonerURL, +Request, ?Response)

Lower level predicate. Sends a DIG Request to the reasoner and get's its Response. It is using SWI Prolog's HTTP and SGML packages.

dig_ask(+ASKQuery, ?XMLRepresentation)

Lower level predicate. It converts a query in the DIG Ask language to the XML representation required by the reasoner.

Conversion Table

DIG ASK term query	XML representation	Result Set
allConceptNames	<allConceptNames id=ID/>	<conceptSet> C1,C2,Cn </conceptSet>
allRoleNames	<allRoleNames id=ID/>	<roleSet> C1,C2,Cn </roleSet>
allIndividuals	<allIndividuals id=ID/>	<individualSet> C1,C2,Cn </individualSet>
satisfiable(C)	<satisfiable>D</satisfiable> where D = description(C)	True/false
subsumes(C1,C2)	<subsumes> D1 D2 </subsumes> where Di = description(Ci)	True/false
disjoint(C1,C2)	<disjoint> D1 D2 </disjoint> where Di = description(Ci)	True/false
parents(C)	<parents> D </parents>	<conceptSet> C1,C2,Cn </conceptSet>
children(C)	<children> D </children>	<conceptSet> C1,C2,Cn </conceptSet>
ancestors(C)	<ancestors> D </ancestors>	<conceptSet> C1,C2,Cn </conceptSet>
descendants(C)	<descendants> D </descendants>	<conceptSet> C1,C2,Cn </conceptSet>
equivalents(C)	<equivalents> D </equivalents>	<conceptSet> C1,C2,Cn </conceptSet>
instances(C)	<instances> D </instances>	<individualSet> C1,C2,Cn </individualSet>
types(I)	<types> I </types>	
instance(I,C)	<instance> <individual name=I> D </instance>	True/false
roleFillers(I,R)	<roleFillers> <individual name=I> <ratom name=R> </roleFillers>	<individualSet> C1,C2,Cn </individualSet>

relatedIndividuals(R)	<relatedIndividuals> <individual name=I> </relatedIndividuals>	<individualPairSet> C1,C2,Cn </individualPairSet>
toldValues(I,R)	<toldValues> <individual name=I> <attribute name=R> </toldValues>	

dig_ask_response(+ASKQuery, +ReasonerResult, ?Result)

Lower level predicate. It converts the XML representation (ReasonerResult) of the responses to an ASK query to a list representation of the Results based on DIGs response language.

owl_as2dig(+OwlAsTerm,?TellElement)

Predicate to convert a Thea prolog OWL abstract term into a DIG Tell element ready to be submitted to the DIG reasoner via a tell request.

OWL AS axiom and fact	DIG Tell element
Class C with no description	<defconcept name=C/>
Class C complete declaration with single description D	<defconcept name=C/> C and D equivalent → MAP(subclassOf(C,D)) AND MAP(subclassOf(D,C)).
Class C complete declaration with multiple descriptions DL	<defconcept name=C/> MAP(subclassOf(C, Map(intersectionOf(DL)))).
Class C partial declaration with multiple descriptions DL	<defconcept name=C/> MAP(subclassOf(C,D)) for each D in DL.
subclassOf(C,D)	<impliesc> Map(D) Map(C)</impliesc>
intersectionOf(DL)	<and> Map(D1) Map(D2)…Map(Dn) </and>
unionOf(DL)	<or> Map(D1) Map(D2) . . . Map(Dn)</or>
complementOf(C)	<not> Map(C) </not>
oneOf(IL)	<iset> Map(IL) </iset>
Restriction(property,Value)	<some> <ratom name=property> <iset> <individual name=Value/> </iset> </some>
Restriction(property,allValuesForm(D))	<all> <ratom name=property> Map(D) </all>
Restriction(property,someValuesForm(D)) (body)	<some> <ratom name=property> Map(D) </some>
Cardinalities	<atmost num=c>, [<atleast num=C>] <ratom name=property/> <top/> </atmost>
S is super property of P	<impliesr> <ratom name=P> <ratom name=S> </impliesr>
C in the domain of P	<domain> <ratom name=P> D </domain>
C in the range of P	<rangeint> / <rangestring> <attribute name=P/> </rangeint> <range> <ratom name=P/> D

	</range>
P is functional property	<functional> <ratom name=P/> </functional>
P is inverse functional property	<inverse> <ratom name=P/> </inverse>
P is a transitive property	<transitive> <ratom name=P/> </transitive>
P is a symmetric property	<equalr> <ratom name=P/> <inverse> <ratom name=P/> </inverse> </equalr>
P is the inverse of Q	<equalr> <ratom name=P/> <inverse> <ratom name=Q/> </inverse> </equalr>
individual(IID,_,DescriptionList,ValueList)	<instanceof> <individual name=IID> D </instanceof> (for each D in Description List) <value> <individual name=IID/> <attribute name=Pi/> <ival/sval name=Vi/> </value> (for datatype properties Pi,Vi in ValueList) <value> <individual name=IID/> <ratom name=Pi/> <individual name=Vi/> </value> (for object properties Pi, Vi in ValueList)

owl_as2dig(property(PID,_Deprecated,_AnnotationList,PID_SuperList,PTList ,PID_DomainList,PID_RangeList),Tells)

Property translation. Translates to a set of subproperties, domain, range and property attribute tells that are handled through a set of mapping functions.

owl_as2dig(individual(IID,_,TypeList,PropertyList),L)

Individual translation. Translates to a set of instancof (type), property and role tells that are handled through apprpropiate mapping functions.

process_pt_dig(PID, [Type,F,IF,T,S,iof(Inv)],[Typet, Ft,IFt,Tt,St,INVt])

Translate property attributes. Define attribute or role. Create functional, inverse, transitive, symmetric and inverse property tells.

map_property_dig(IID, value(P,V), Tells)

Mapping of an instance property values. For a datatypeproperty value(I, P, V) for an objectproperty related(I, R, I).

```
/*
            OWL_AS 2 Prolog submodule
           -------------------------------

*/
```

owl_as2prolog(+OwlAsTerm,+Options)

Converts the prolog OWL abstract syntax term (as parsed by OWl parser) into prolog logic code, based on the mapping proposed by [Grosof] in the context of DLP. The prolog code is written into the current output stream, so redirecting the output stream into a file is suggested in order to capture the generated code. Options are generic options to modify the behaviour of the code generation. Currently only the no_base(Namespace) is supported. This option tells the code generator not to prefix the prolog predicates with the namespace prefix.

owl_write_prolog_code(+Term,+Options)

Term is an intermediate format generated from the owl_as2prolog/3 predicate. This predicate handles the prolog code generation from this intermediate format into prolog code.
For Options see the owl_as2prolog/2 predicate.

owl_as2prolog(+OwlAsTerm, -ResultTerm, ?Mode)

Predicate to convert a Thea prolog OWL abstract term into the intermediate term used for prolog (logic) code generation. The Mode is used to differentiate the conversion depending on wether the OWL construct appears in the head or in a body of a prolog rule. It cna be on of head, body and fact.

process_pt_list(PID, [_,F,IF,T,S,iof(Inv)],[Ft,IFt,Tt,St,INVt])

Mappings generated from the attributes of a property.
a. Functional and inverse functionals generate a
 sameIndividuals(X,Y) :- p(Z,X), P(Z,Y)
 Transitive: p(X,Z) :- p(X,Y), p(Y,Z).
 Symmetric: p(X,Y) :- p(Y,X).
 Inverse : p(X,Y) :- inv(Y,X) and inv(X,Y) :- p(Y,X).

5. Examples

Thea has been tested extensively against the OWL test cases[6] and for almost all cases it generates the correct syntactic constructs.

In the following we discuss the results of running Thea against few specific OWL test cases.

5.1 OWL Parser Examples

5.1 Test case <Restriction/consistent003>

This test case demonstrates the parsing of *owl:Restriction* constructs. The resulting Ontology contains the 4 classes below:

Class Name	Description
'http://www.w3.org/2002/03owlt/Restriction/consistent003#C'	[intersectionOf(['http://www.w3.org/2002/03owlt/Restriction/consistent003#superC', restriction('http://www.w3.org/2002/03owlt/Restriction/consistent003#dp', someValuesFrom('http://www.w3.org/2001/XMLSchema#byte'))])]
'http://www.w3.org/2002/03owlt/Restriction/consistent003#superC'	
'http://www.w3.org/2002/03owlt/Restriction/consistent003#D'	[intersectionOf(['http://www.w3.org/2002/03owlt/Restriction/consistent003#superD', restriction('http://www.w3.org/2002/03owlt/Restriction/consistent003#dp', someValuesFrom('http://www.w3.org/2001/XMLSchema#byte'))])]
'http://www.w3.org/2002/03owlt/Restriction/consistent003#superD'	

The Ontology is not OWL DL because structure sharing occurred:

```
20 ?- blanknode(A,B,C).

A =
'__file:c:/sw/supportmaterial/owl/approved/restriction/consistent003.rdf#__Node1'
B = restriction('http://www.w3.org/2002/03owlt/Restriction/consistent003#dp',
someValuesFrom('http://www.w3.org/2001/XMLSchema#byte'))
C = shared ;
```

5.2 Test case <Restriction/consistent004>

The above example Ontology in OWL Lite: Same resulting Ontologies but no shared blank nodes:

```
25 ?- blanknode(A,B,C).

A =
'__file:c:/sw/supportmaterial/owl/approved/restriction/consistent004.rdf#__Description1'
B = restriction('http://www.w3.org/2002/03owlt/Restriction/consistent004#dp',
```

```
someValuesFrom('http://www.w3.org/2001/XMLSchema#byte'))
C = used ;

A =
'__file:c:/sw/supportmaterial/owl/approved/restriction/consistent004.rdf#__Description2'
B = restriction('http://www.w3.org/2002/03owlt/Restriction/consistent004#dp',
someValuesFrom('http://www.w3.org/2001/XMLSchema#byte'))
C = used ;
```

5.3 Test case <disjointWith/consistent008>

The following four disjoint sets were identified correctly. The four nodes (A,B,C,D) do not form a single disjoint set because structure sharing has occurred in the blank nodes intersection(B) and intersection(C).

```
29 ?- disjointSet(X).

X =
[intersectionOf(['http://www.w3.org/2002/03owlt/disjointWith/consistent008#B']),
'http://www.w3.org/2002/03owlt/disjointWith/consistent008#A'] ;

X =
[intersectionOf(['http://www.w3.org/2002/03owlt/disjointWith/consistent008#C']),
'http://www.w3.org/2002/03owlt/disjointWith/consistent008#A'] ;

X = ['http://www.w3.org/2002/03owlt/disjointWith/consistent008#D',
intersectionOf(['http://www.w3.org/2002/03owlt/disjointWith/consistent008#B'])] ;

X = ['http://www.w3.org/2002/03owlt/disjointWith/consistent008#D',
intersectionOf(['http://www.w3.org/2002/03owlt/disjointWith/consistent008#C'])] ;

No
30 ?- blanknode(A,B,C).

A =
'__file:c:/sw/supportmaterial/owl/approved/disjointwith/consistent008.rdf#__Node1'
B = intersectionOf(['http://www.w3.org/2002/03owlt/disjointWith/consistent008#B'])
C = shared ;

A =
'__file:c:/sw/supportmaterial/owl/approved/disjointwith/consistent008.rdf#__Node2'
B = intersectionOf(['http://www.w3.org/2002/03owlt/disjointWith/consistent008#C'])
C = shared ;
```

5.4 Test case <disjointWith/consistent009>

Two disjoint sets were identified: [D,intersection(B),A] and [D,intersection(C),A]. Only named classes (nodes) have been shared so it is in OWL DL.

```
34 ?- disjointSet(X).

X = ['http://www.w3.org/2002/03owlt/disjointWith/consistent009#D',
intersectionOf(['http://www.w3.org/2002/03owlt/disjointWith/consistent009#B']),
'http://www.w3.org/2002/03owlt/disjointWith/consistent009#A'] ;

X = ['http://www.w3.org/2002/03owlt/disjointWith/consistent009#D',
intersectionOf(['http://www.w3.org/2002/03owlt/disjointWith/consistent009#C']),
'http://www.w3.org/2002/03owlt/disjointWith/consistent009#A'] ;

No
```

```
35 ?- blanknode(A,B,C).

A =
'__file:c:/sw/supportmaterial/owl/approved/disjointwith/consistent009.rdf#__Node1'
B = intersectionOf(['http://www.w3.org/2002/03owlt/disjointWith/consistent009#B'])
C = used ;

A =
'__file:c:/sw/supportmaterial/owl/approved/disjointwith/consistent009.rdf#__Node2'
B = intersectionOf(['http://www.w3.org/2002/03owlt/disjointWith/consistent009#C'])
C = used ;
```

5.5: Test case <maxCardinality/inconsistent001>

The correct individual term is constructed (type and property values). Still no consistency checking is done.

```
40 ?- individual(A,B,C,D).

A = 'http://www.w3.org/2002/03owlt/maxCardinality/inconsistent001#sb1'
B = []
C =
[restriction('http://www.w3.org/2002/03owlt/maxCardinality/inconsistent001#prop',
maxCardinality(literal('2')))]
D = [value('http://www.w3.org/2002/03owlt/maxCardinality/inconsistent001#prop',
'http://www.w3.org/2002/03owlt/maxCardinality/inconsistent001#ob1'),
value('http://www.w3.org/2002/03owlt/maxCardinality/inconsistent001#prop',
'http://www.w3.org/2002/03owlt/maxCardinality/inconsistent001#ob2'),
value('http://www.w3.org/2002/03owlt/maxCardinality/inconsistent001#prop',
'http://www.w3.org/2002/03owlt/maxCardinality/inconsistent001#ob3')] ;

No
```

5.6 Test case <maxCardinality/inconsistent002>

An example of an external error due to missing triples: In this case the definition of the `otherprop` as a property is missing thus the following triples remain unused. The parser does not infer that since `otherprop` is a `subPropertyOf` of `prop` is itself a property.

```
44 ?- owl(A,B,C,not_used).

A = 'http://www.w3.org/2002/03owlt/maxCardinality/inconsistent002#sb1'
B = 'http://www.w3.org/2002/03owlt/maxCardinality/inconsistent002#otherprop'
C = 'http://www.w3.org/2002/03owlt/maxCardinality/inconsistent002#ob3' ;

A = 'http://www.w3.org/2002/03owlt/maxCardinality/inconsistent002#otherprop'
B = 'http://www.w3.org/2000/01/rdf-schema#subPropertyOf'
C = 'http://www.w3.org/2002/03owlt/maxCardinality/inconsistent002#prop' ;
```

5.7 Test case <miscellaneous/Manifest001#test>

This is the Wine Ontolog used in the OWL guide. **Thea** fully parses this DL ontology.

```
46 ?-
owl_parse('c:/sw/supportmaterial/owl/approved/miscellaneous/consistent001.rdf',complete,complete).
Re-hash ...ok
% Parsed "consistent001.rdf" in 0.23 sec; added 2,264 triples
Re-hash ...ok

Yes
48 ?- owl_parser_log(X,Y), print(X), print(':'), print(Y), nl, fail.
"Tue Mar 22 12:30:38 2005":['Removing existing owl triples']
"Tue Mar 22 12:30:38 2005":'Copying RDF triples to OWL triples'
"Tue Mar 22 12:30:38 2005":['Number of owl triples copied: ', 2165]
"Tue Mar 22 12:30:38 2005":['Getting named classes...', 804, ' triples used']
"Tue Mar 22 12:30:38 2005":['Getting subclasses...', 654, ' triples used']
"Tue Mar 22 12:30:38 2005":['Getting disjoint sets of classes...', 1, ' triples used']
"Tue Mar 22 12:30:38 2005":['Getting equivalent sets of classes...', 0, ' triples used']
"Tue Mar 22 12:30:38 2005":['Getting properties...', 45, ' triples unused']
"Tue Mar 22 12:30:38 2005":['Getting Annotation properties...', 0, ' triples used']
"Tue Mar 22 12:30:38 2005":['Getting Individual axioms...', 172, 'triples used']
"Tue Mar 22 12:30:38 2005":['Getting Ontology...', 7, ' triples used ']
"Tue Mar 22 12:30:38 2005":['Getting named Individuals...', 482, ' triples used']
"Tue Mar 22 12:30:38 2005":['Getting unamed classes...', 0, ' triples used']
"Tue Mar 22 12:30:38 2005":['Getting unnamed Individuals...', 0, 'triples used']
"Tue Mar 22 12:30:38 2005":['Number of unused remain triples ', 0]

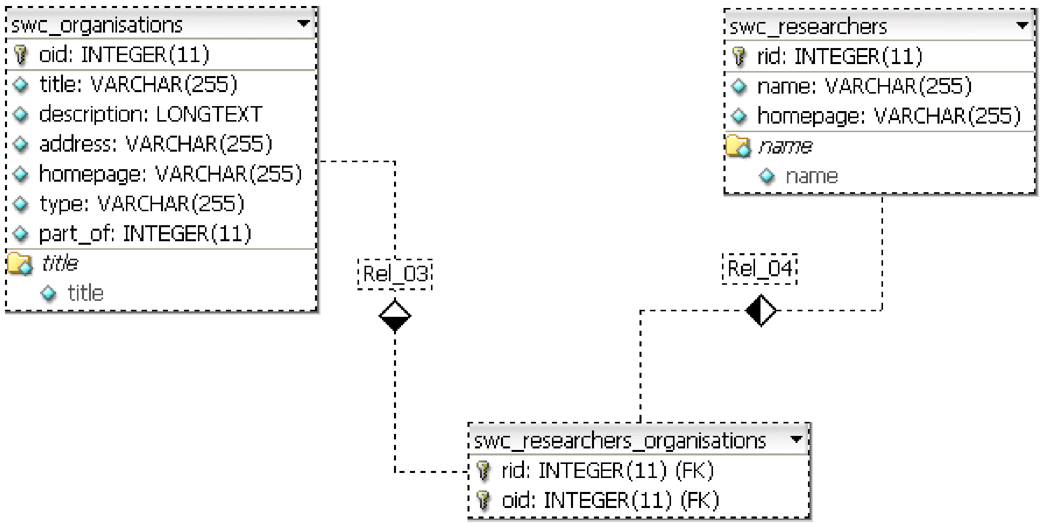
No
```

and it generates:

74	class terms.
126	subclassOf terms.
13	property terms
1	disjointSet
161	individuals
5	differentIndividual Sets
2	ontology terms.

5.2 SQL2OWL converter examples.

The examples below were produced using the following DBMS Schema




```
20 ?- populate_class(mysql,'Person').

Yes

22 ?- individual(X,Y,Z,A), writeq(individual(X,Y,Z,A)), nl,fail.
individual('Person-2', [], ['Person'], [value(works_for_2, 'Organisation-2'),
value('Name2', 'Mike Dean'), value(works_for, 'BBN Technologies / Verizon'),
value('example:Name', 'Mike Dean')])
individual('Person-3', [], ['Person'], [value(works_for_2, 'Organisation-2'),
value('Name2', 'Kelly Barber'), value(works_for, 'BBN Technologies / Verizon'),
value('example:Name', 'Kelly Barber')])
individual('Person-4', [], ['Person'], [value(works_for_2, 'Organisation-6'),
value('Name2', 'John Punin'), value(works_for, 'Rensselaer Polytechnic Institute'),
value('example:Name', 'John Punin')])
individual('Person-5', [], ['Person'], [value(works_for_2, 'Organisation-8'),
value('Name2', 'Chris Waterson'), value(works_for, 'Netscape'), value('example:Name',
'Chris Waterson')])
individual('Person-6', [], ['Person'], [value(works_for_2, 'Organisation-8'),
value('Name2', 'David Hyatt'), value(works_for, 'Netscape'), value('example:Name', 'David
Hyatt')])
individual('Person-7', [], ['Person'], [value(works_for_2, 'Organisation-8'),
value('Name2', 'Robert Churchill'), value(works_for, 'Netscape'), value('example:Name',
'Robert Churchill')])
individual('Person-9', [], ['Person'], [value(works_for_2, 'Organisation-11'),
value('Name2', 'Grigoris Antoniou'), value(works_for, 'Institute of Computer
Science,FORTH'), value('example:Name', 'Grigoris Antoniou')])
individual('Person-10', [], ['Person'], [value(works_for_2, 'Organisation-12'),
value('Name2', 'Siegfried Handschuh'), value(works_for, 'Institute AIFB University of
Karlsruhe (TH)'), value('example:Name', 'Siegfried Handschuh')])
individual('Person-12', [], ['Person'], [value(works_for_2, 'Organisation-12'),
value('Name2', 'York Sure'), value(works_for, 'Institute AIFB University of Karlsruhe
(TH)'), value('example:Name', 'York Sure')])
individual('Person-13', [], ['Person'], [value(works_for_2, 'Organisation-5'),
value('Name2', 'Brian McBride'), value(works_for, 'HP Labs'), value('example:Name', 'Brian
McBride')])
```

```
individual('Person-14', [], ['Person'], [value(works_for_2, 'Organisation-2'),
value('Name2', 'John Flynn'), value(works_for, 'BBN Technologies / Verizon'),
value('example:Name', 'John Flynn')])
individual('Person-15', [], ['Person'], [value(works_for_2, 'Organisation-24'),
value('Name2', 'Chris Bussler'), value(works_for, 'National University of Ireland Galway -
Digital Enterprise Research Institute (DERI),'), value('example:Name', 'Chris Bussler')])
….
```

5.3 OWL reasoner examples.

You can run more OWL reasoner examples by visiting **Thea**'s on line demo page at **www.semanticweb.gr/TheaOWLLib**

---

[1] Description Logic Programs. http://www2003.org/cdrom/papers/refereed/p117/p117-grosof.html
[2] DL Implementation Group DIG Interface specification. http://dl.kr.org/dig/interface.html
[3] Sean Bechhofer (seanb@cs.man.ac.uk), University of Manchester: *OWL Web Ontology Language Parsing OWL in RDF/XML. (http://www.w3.org/TR/owl-parsing/)*
[4] Similar work in mapping between Relational Databases and RDF is carried out in D2RMAP
[5] Similar work in converting OWL to Prolog is done in dlpconvert
[6] http://www.w3.org/TR/owl-test/