



Tests and Benchmarks

v.2.8.2, 3 Mar 2006 (DRAFT)

1	Introduction	2
2	Configuring Sesame	2
3	Running Entailment Test Cases	3
3.1	JUnit Entailment Tests	3
3.2	W3C Entailment Tests	6
4	Running Lehigh University Benchmark	7

1 Introduction

OWLIM is a semantic repository, packaged as a Storage and Inference Layer (SAIL) for Sesame RDF database. More information on OWLIM can be found at <http://www.ontotext.com/owlim>.

This document describes the various tests included in the OWLIM distribution and gives detailed information on how to setup and run them. For the user's convenience, the distribution includes several batch files (Windows only) that can be used to run the different tests. In order to run them on the user's PC, an initial modification/adaptation is required.

Requirements for all included tests:

- Java v1.4 or higher;
- Sesame v. 1.2.1 to 1.2.4;

The user also needs:

- JUnit 3.8.1 – to run the JUnit-related test cases;
- W3C owl testcases - to run the respective entailment tests;

To run Lehigh University Benchmark, it is required that the test fileset be generated beforehand. To do so, the script named `lubm-generate.cmd` (part of the distribution) can be used. The distribution includes a prebuilt library of the benchmark's source code, available for download at <http://swat.cse.lehigh.edu/projects/lubm/index.htm>. The prebuilt `lubm.jar` library is located in the `lib` distribution folder and also includes the wrapper classes that the benchmarks codebase require in order to run over Sesame repository, configured with OWLIM.

2 Configuring Sesame

To run properly the test cases, the test Sesame SAIL must be configured beforehand. The name of the config file is `system.conf`. The file must be located in the distribution `test/test` directory. The config file's content is divided in two parts: system configuration (contains user names and passwords); and repository list (contains information of each repository - file names and the namespaces of the imported ontologies as well as other parameters related to the repository). A sample config file can be found in the "test" dir.

The config file contains a repository with the `id="test"`, defined as follows:

```
<repository id="test">
  <title>Test Entailment repository </title>
  <sailstack>
    <sail class="org.openrdf.sesame.sailimpl.OWLIMSchemaRepository">
      <param name="file" value="./kb/test.nt"/>
      <param name="dataFormat" value="ntriples"/>
      <param name="compressFile" value="no"/>
      <param name="ruleset" value="owl-max"/>
      <param name="partialRDFS" value="true"/>
      <param name="indexSize" value="4000000"/>
      <param name="new-triples-file"
        value="./kb/new-temp-triples.nt"/>
      <param name="auto-write-time-minutes" value="15"/>
      <param name="base-URL"
        value="http://example.org/entailment/test#"/>
    </sail>
  </sailstack>
</repository>

<!-- semicolon should be used as delimiter -->
```

```

        <param name="imports" value=
            "/kb/owl/owl.rdfs;
            /kb/test/testSchema.rdf;
            /kb/test/testOntology.rdf;"/>
        <param name="defaultNS" value=
            "http://www.w3.org/2002/07/owl#;
            http://www.w3.org/2000/10/rdf-
            tests/rdfcore/testSchema#;
            http://www.w3.org/2002/03owl-t/testOntology#;"/>
    </sail>
</sailstack>

<!--ACLlist can contain zero or more 'user' elements-->
<acl worldReadable="true" worldWritable="true">
    <user login="admin" readAccess="true" writeAccess="true"/>
</acl>
</repository>
    
```

This repository uses `org.openrdf.sesame.sailimpl.OWLIMSchemaRepository` as a SAIL. It stores the data in N-Triple format in the file `test/kb/test.nt`. The temporary statements are stored in the file `test/kb/new-temp-triples.nt`. Its base URL is `http://example.org/entailment/test#` and imports `owl.rdfs`, `testSchema.rdf` and `testOntology.rdf` with default namespaces `http://www.w3.org/2002/07/owl#`, `http://www.w3.org/2000/10/rdf-tests/rdfcore/testSchema#` and `http://www.w3.org/2002/03owl-t/testOntology#` respectively.

3 Running Entailment Test Cases

There are two different kinds of entailment tests for OWLIM – `TestOWLIMRepository` and `EntailmentTest`. The first type uses a set of OWL files; the set contains few facts and checks whether or not a given set of statements is present in the inferred closure. To do so, the hard-coded logic of the test is employed. The test itself does not rely on any ontology different from OWL (loaded by default).

3.1 JUnit Entailment Tests

`TestOWLIMRepository` uses JUnit library. Each document is used in a separate test case. There are 14 test cases as follows:

- `owl_test1a` – a symmetric property test: if a property `symmetricTestProp` is an `owl:SymmetricProperty` and there is a statement `John symmetricTestProp Mary`, then the statement `Mary symmetricTestProp John` must be present in the repository; similarly, if the fact (the statement) `symmetricTestProp rdf:type owl:SymmetricProperty` is removed from the repository, then a statement as `Mary symmetricTestProp John` must not be present;
- `owl_test2a` – a transitive property test: if a `transitiveTestProp` is an `owl:TransitiveProperty` and the repository contains the statements `Mary transitiveTestProp Harry` and `John transitiveTestProp Mary`, then it must also contain the statement `John transitiveTestProp Harry`. Similarly, if one of these statements is removed, then the last statement must be removed as well and if the removed statement is added anew, the last statement must again be added;
- `owl_test1a2a` – defines `transitiveAndSymmetricTestProp`, which is both `owl:SymmetricProperty` and `owl:TransitiveProperty`. `Mary` and `Harry`, as

well as John and Mary, are connected by means of this relation. The test checks if the full symmetric and transitive closure is inferred.

- *owl_test3a* – defines **inverseOfTransitive** property which is `owl:inverseOf owl:TransitiveProperty`. John, Mary and Harry are **owl:transitiveTestProp** in this order as it was the case with *owl_test2a*. The test checks if they are **inverseOfTransitive** in this order: Harry, Mary, John. It also checks if any of them is **inverseOfTransitive** of itself (because the inferencer applies the transitivity **a trans b and b trans c → a trans c** only when the three variables are distinct).
- *owl_test10a* – same as *owl_test2a*, except that it defines **eqTestProp** which is `owl:equivalentProperty` to **transitiveTestProp**. `owl:equivalentProperty` means that **eqTestProp** and **transitiveTestProp** might be interchangeably used. In the inferred closure all subjects and objects linked with **transitiveTestProp** must also be linked with **eqTestProp** and that is what the test checks.
- *owl_test10b* – test focused on `rdfs:domain-` and `rdfs:range`. If John, Mary and Harry are **transitiveTestProp** in this order. **inverseOfTransitive** takes its subject from the class **DomainName**, while **eqTestProp** (`owl:equivalentProperty` to **inverseOfTransitive**) takes its object from the class **RangeName**. It follows that, Harry and Mary must belong to **DomainName**; Mary and John must belong to **RangeName**. Moreover, Harry must not be a member of **RangeName** and John must not be a member of **DomainName**, while Mary must belong to both classes.
- *owl_test12a* – similar to *owl_test10b* with a slight difference, namely that **eqTestProp** is `owl:sameAs` **inverseOfTransitive** (stronger than `owl:equivalentProperty`). In fact, it will copy all the statements containing **inverseOfTransitive** and will replace it with **eqTestProp** no matter if the former appears somewhere as a subject, predicate or object). The results must be the same as that of *owl_test10b*.
- *owl_test12b* – tests the `owl:sameAs` inference. If John, Mary and Harry are **transitiveTestProp** as in the example above and at the same time, **MaryAlias** is `owl:sameAs` **Mary**, then **MaryAlias** and **Mary** must be used interchangeably.
- *owl_test15a*, *owl_test15b* – tests the `owl:FunctionalProperty` inference. According to the rule `id:owl_FunctProp` if a property **p** is `owl:FunctionalProperty` and at the same time **x p y** and **x p z**, it follows that **y owl:sameAs z** (i.e. the “value” of **p** for a given argument is unique, since **p** represents a function). To illustrate: if John’s mother is Mary and at the same time John’s mother is Carmen, then Mary and Carmen must be one and the same person as long as **hasMother** is `owl:FunctionalProperty`. But if in a given moment **hasMother** ceases to be `owl:FunctionalProperty`, then the test asserts that Mary and Carmen are different persons. The property **motherOf** is `owl:inverseFunctionalProperty` of **hasMother** (i.e. if **p** is `owl:inverseFunctionalProperty` and **x p z** and **y p z** are asserted, then **x** must be `owl:sameAs z`). With the above conditions we have Mary **motherOf** John (the inverse of John **hasMother** Mary) and Carmen **motherOf** John (the inverse of John **hasMother** Carmen). And again, if in a given moment **motherOf** ceases to be `owl:inverseFunctionalProperty`, then Mary and Carmen must be different persons (and note: if this is so, **hasMother** will cease to be `owl:FunctionalProperty`).

- *owl_test16a, owl_test16b* – tests of `owl:FunctionalProperty` and `owl:InverseFunctionalProperty`. The property **ownSSNumber** (Social Security Number) is defined as `owl:InverseFunctionalProperty`. If two persons have the same Social Security Number, it follows that they must be `owl:sameAs`. In *owl_test16b*, **ownSSNumber** is defined as `owl:inverseOf ownedBy`, which in its turn is `owl:FunctionalProperty`. And again, if two people have the same Social Security Number, it follows that they must be one and the same person (the two definitions of **ownSSNumber** are equivalent).
- *owl_testgos1a* – tests a number of transitive properties – **partOf**, **subRegionOf** and **livesIn**. Europe and Asia are **partOf** the Earth; Germany and Japan are respectively **subRegion**-s of Europe and of Asia; Berlin is **subRegionOf** Germany. **partOf** is `owl:TransitiveProperty`, **subRegionOf** is `rdfs:subPropertyOf partOf` (but it is NOT `owl:TransitiveProperty partOf`), and **livesIn** is `protons:transitiveOver subRegionOf`. Then, if John **livesIn** Berlin, it follows that he **livesIn** Europe, but it is not true that he **livesIn** Earth (because **livesIn** is `protons:transitiveOver subRegionOf`, but **subRegionOf** itself is not `owl:TransitiveProperty` although it is `rdfs:subPropertyOf partOf`).
- *owl_test_oneOf* – tests wheter an instance, specified within the declaration of an enumerated class **tst:TShirt** is made a member of this class, **tst:TShirt** – checked for the **tst:small** instance.
- *owl_minCardinality (1)* – the test class **tst:PersonWithEmail** is declared to be equivalent to a restriction over property **tst:email** with `owl:minCardinality` with value of "1"^^`xsd:nonNegativeInteger`, then some instance, **tst:Pesho** that is related to a literal with the property **tst:email** is checked to be a member of **tst:PersonWithEmail** class.
- *Owl_TBOX_SomeValFrom* – this test checks for class subsumption based on `someValuesFrom` restrictions over sub-properties to values of classes that are sub-classes. The test case has two named class definitions **tst:Employee** and **tst:ResearchAssistant** equivalent to such restrictions over **tst:worksFor** property and relating values of **tst:Organisation** and its subclass **tst:ResearchGroup**. The test ceack wheter an instance **tst:Naso**, which is related with the property **tst:workFor** to some instance of class **tst:ReasearchGroup** is a member of the **tst:Employee**.
- *Owl_TBOX_AllValFrom* – a test similar to the above but covering relationships between restrictins of type `allValuesFrom`. Consult the test case OWL document – `owl_TBO_AVF.owl` for more details of the derived subsumtion.
- *Owl_TBOX_hasValue* – a test similar to the two above, but covering relationships between two `owl:hasValue` restrictions.
- *Owl_TBOX_HasValSomeValuesFrom* - is a test that examine the relations between `someValuesFrom` and `hasValues` restrictions to derive subsumption.
- *Owl_TBOX_SomeValues_minCardinality1* – similar as the above ones, but now in the scope of the test is the relation between restioctions of type `someValuesFrom` and `minCardinality(1)`.

Note: TBOX and cadinality tests will pass only if the chosen ruleset is "owl-max"!

In order to use the `junit-test.cmd` script, use your favorite textpad to open the script and modify:

JAVA_HOME to point to the installation folder of Java

JUNIT_HOME to point to the folder containing the junit library

SESAME_LIB to point to the location of the related Sesame libraries

and then run the script.

3.2 W3C Entailment Tests

The second kind of test depends on the, so called, approved tests that W3C provides. Each approved test is a set of premises and conclusions given as OWL documents. Depending on the kind of the test employed (described in the corresponding Manifest OWL document), the check determines whether or not the conclusions are present in the repository.

The following types of tests are described in `testSchema.rdf` and `testOntology.rdf`:

- **Positive entailment tests** – check if the inference engine infers the expected results;
- **Negative entailment tests** – check if the inference engine does infer a/the statement(s) listed in a separate file created especially for the test;
- **True test** – a test that checks if the repository contains a number of statements that are either axioms or inferred from axioms (and thus must be present in every repository, including in the “empty” ones);
- **OWL for OWL test** – test that illustrates how OWL is used to describe/for the description of OWL Full.
- **Import level tests** – serve to indicate if there is an interaction between `owl:imports` and the sublanguage elements in the main document (that imports certain ontologies);
- **Not OWL feature test** – a kind of negative test; checks if the inference engine supports features of DAML+OIL, which OWL does not support anymore;
- **Import entailment test** – checks if the premise document, and its “imports” closure, entails the conclusion document;
- **Inconsistency test** – a kind of positive entailment test that indicates if there are inconsistencies in the inferred closure of an ontology (e.g. if two entities are `owl:sameAs` while at the same time they are `owl:differentFrom` one another; if an entity is of type `owl:Nothing` or it is `rdfs:subClassOf owl:Nothing` and at the same time it is `owl:differentFrom owl:Nothing`. Statements of this kind are inconsistent with the semantics of OWL);
- **Consistency test** – a kind of negative entailment test; checks if the OWL document does entail a falsehood.

For now, the OWLIM entailment test supports the Positive and Negative Entailment tests only. Inconsistency tests are not supported yet. They could be used to determine what part of OWL OWLIM inference engine is able to handle.

The test OWL files are available at <http://www.w3.org/TR/owl-test/> (as part of the file `approved.zip`). On the same web-site one can find more information on the features that each test checks.

After the test cases have been retrieved and unpacked in a specified folder, modify the `entailment-test.cmd`, so that:

`APPROVED_DIR` points to the folder on the user's PC, where the testcases reside;

`JAVA_HOME` points to the installation folder of Java;

`SESAME_LIB` points to the location of the related Sesame libraries;

After that, run the script. It then automatically passes over the contents of the subfolders and parses the ManifestXXX files found there. Then, it proceeds with a particular testcase and delivers the results.

If the name of an `APPROVED_DIR`'s subfolder is specified as an optional script argument, then only the testcases contained in that particular subfolder will be performed. For example:

```
C:\owlim\test>entailment-test.cmd AllDifferent
```

has the following output:

```
Processing folder AllDifferent
    positive test:(#Full): AllDifferent/Manifest001#test    passed.
```

4 Running Lehigh University Benchmark

First you need to obtain the benchmark test's fileset. Look the `lubm-generate.cmd` script file and modify `JAVA_HOME`, so that it points to the installation folder of Java. The script accepts a single numeric argument as the target number of the universities and creates a subfolder 'univer#' where it places the generated OWL files. Once the fileset is generated, take a look at the `config.kb.example.owlim` file and comment/uncomment the appropriate benchmark configuration section you'd like to be executed. By default, a single University dataset is configured and its section looks like this:

```
# 1-university
[OWLIM_1]
class=owlim.OwlWrapper
data=./univer1
database=jdbc:ignore
ontology=http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl
```

Simply use '#' at the beginning of the line to comment it or remove it to leave uncommented.

Modify `lubm-benchmark.cmd`, so that:

`JAVA_HOME` points to the installation folder of Java;

`SESAME_LIB` points to the location of the related Sesame libraries;

then execute it. To find more about this benchmark, visit its web site at: <http://swat.cse.lehigh.edu/projects/lubm/index.htm>