ESTRELLA

IST-2004-027655

European project for Standardized Transparent Representations in order to Extend LegaL Accessibility
Specific Targeted Research or Innovation Project

Specific Targeted Research Project
Information Society Technologies

Deliverable N°: 4.2

Refined translators to and from LKIF for each of the knowledge formats of the participating vendors

Version:	FINAL 1.0 Universiteit van Amsterdam
Due date of Deliverable:	September 30, 2008
Actual submission date:	September 30, 2008
Start date of Project:	1 January 2006
Duration:	30 months
Project Coordinator:	Universiteit van Amsterdam (NL)
Lead contractor deliverable:	Corvinus University Budapest
Participating contractors:	Alma Mater Studiorum - Universita di Bologna (IT), University of Liverpool (UK), Fraunhofer Gesellschaft zur foerderung der angewandten forschung e.v. (DE), RuleWise b.v. (NL), RuleBurst (EUROPE) Ltd. (UK), knowledgeTools International GmbH (DE), Interaction Design Ltd. (UK), SOGEI - Societa Generale d'Informatica S.P.A. (IT), Ministro per le Riforme e le Innivazioni nella Publica Amministrazione (IT), Hungarian Tax and Financial Control Administration (HU), Budapesti Corvinus Egyetem (HU), Ministero dell'Economia e delle Finanze (IT), Consorzio Pisa Ricerche SCARL (IT)

Project funded by the European Community under the 6th Framework Programme

	Dissemination Level	
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

0.1 Introduction

This doument specifies how to use the translator developed at the UvA, which converts documents containing OWL axioms (via LKIF-rules in RDF) to s-expressions. The translator only provides a partial translation as far as the OWL file is concerned. Just classes, properties and both the class hierarchy and property hierarchy are translated. With hierarchy we mean the rdfs:subclassOf and rdfs:subPropertyOf properties:

- The property rdfs:subClassOf is an instance of rdf:Property that is used to state that all the instances of one class are instances of another.
- The property rdfs:subPropertyOf is an instance of rdf:Property that is used to state that all resources related by one property are also related by another.

For more technical information on how the conversion from OWL axioms to LKIF Rules works, we refer the reader to a report about this [Boer, A., 2008].

0.2 Translating OWL to s-expressions

The translator is written in Java and has two tasks:

- Given an input in the form of an OWL document the translator transforms classes, properties and class and property hierarchy to LKIF rules in RDF.
- Given the output from the previous step the translator converts the LKIF rules to s-expressions. This output is a format which can for example be digested by Carneades.

To be able to run the translator, a user should have a working copy of the Eclipse framework available. Eclipse is an open IDE for developing Java programs and can be freely downloaded from the following location: http://www.eclipse.org/

Subsequently the project that contains all the necessary class files should be imported as a project into the Eclipse environment. The relevant folder can be found at the Estrella UvA SVN repository[1].

The next step is to set the input and output parameters correctly in the main class of the translator project: TestMain.java. Three different parameters are present in this class file:

- owlRaw: which denotes the input in the form of an OWL file
- output: which denotes the output of the conversion of OWL into LKIF rules, but also represents the input for the second task: translating the LKIF rules into s-expressions.

[1] http://svn.leibnizcenter.org/svn/EstrellaUvA/Translator/TS

- output2: the name of the output file that will contain the result of both the translation steps in the form of s-expressions.

To run the translator use TestMain.java as entry point. The code of the translator is listed in the appendix.

0.3 Appendix

```
package translator;

import java.io.FileOutputStream;
import java.io.IOException;

import com.hp.hpl.jena.ontology.OntModel;


public class TestMain {

        String owlRaw,owlModified;
        String eu2="file:ontologies/
           EU_directive_LKIF_rules_1.0/directive-31990L0434.
           owl";
        String template = "file:ontologies/out/template.owl"
           ;
        String output="ontologies/out/out.owl";
        String output2="ontologies/out/out.txt";

        public TestMain() throws IOException{

                owlRaw=eu2;
//              owlModified="file:ontologies/out/
    subclass2LkifOut.owl";
                testO2L(owlRaw,output);

                testL2S("file:"+output,output2);
                testL2S(eu2,output2);

        }

        public void testO2L(String inFile,String outFile)
           throws IOException{

                Owl2Lkif sl=new Owl2Lkif(inFile,template);
                OntModel mOut = sl.run();
```

```
                FileOutputStream fo=new FileOutputStream(
                    outFile);
                mOut.write(fo);
                fo.close();

        }

        public void testL2S(String inFile,String outFile)
            throws IOException{

                Lkif2Sexp ls=new Lkif2Sexp(inFile);
                String s=ls.run();

                FileOutputStream fo=new FileOutputStream(
                    outFile);
                fo.write(s.getBytes());
                fo.close();
        }

        public static void main(String[] Args) throws
            Exception{

                new TestMain();
        }


}


package translator;

import java.util.Iterator;

import com.hp.hpl.jena.ontology.Individual;
import com.hp.hpl.jena.ontology.OntClass;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntModelSpec;
import com.hp.hpl.jena.ontology.OntProperty;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.Property;
import com.hp.hpl.jena.rdf.model.RDFNode;
import com.hp.hpl.jena.rdf.model.Statement;

public class Owl2Lkif {

        OntModel mIn;
```

```
OntModel mOut;
OntClass PositiveStatement;
OntClass Variable;

Property object;
Property subject;
Property predicate;
Property subClassOf;
Property subPropertyOf;
Property type;

String rdfs="http://www.w3.org/2000/01/rdf-schema#";
String owl="http://www.w3.org/2002/07/owl#";
String rdf="http://www.w3.org/1999/02/22-rdf-syntax-
    ns#";
String variable="file:ontologies/
    EU_directive_LKIF_rules_1.0/variable.owl#";
String lkif="file:ontologies/
    EU_directive_LKIF_rules_1.0/LKIFv4.1.owl#";

int VarCounter=0;

public Owl2Lkif(String onto, String template){

        mIn = ModelFactory.createOntologyModel(
            OntModelSpec.OWL_MEM, null
        );
        mOut = ModelFactory.createOntologyModel(
            OntModelSpec.OWL_MEM, null
        );

        // read the source document
        mIn.read( onto );
        mOut.read(onto);

        subClassOf = mOut.createProperty(rdfs+"
            subClassOf");
        subPropertyOf = mOut.createProperty(rdfs+"
            subPropertyOf");
        object=mOut.getProperty(lkif+"object");
        subject=mOut.getProperty(lkif+"subject");
        predicate=mOut.getProperty(lkif+"predicate")
            ;
        type=mOut.getProperty(rdf+"type");
        PositiveStatement=mOut.getOntClass(lkif+"
            PositiveStatement");
```

```
                Variable=mOut.getOntClass(variable+"Variable
                    ");

        }

        public OntModel run(){

//              System.out.println(VarCounter);
                subClass();
                subProperty();
                facts();
                return mOut;

        }

        public void subClass(){

                for (Iterator i=mIn.listClasses();i.hasNext
                    ();){

                        OntClass oC=(OntClass)i.next();
                        String ns=oC.getNameSpace();
                        if(!(oC.isAnon()||ns.equals(owl)||ns
                            .equals(rdfs))){

                        Individual iv1=PositiveStatement.
                            createIndividual();
                        iv1.addProperty(predicate, type);
                        iv1.addProperty(object, oC);
                        Individual ivc=Variable.
                            createIndividual();
                        iv1.addProperty(subject, ivc);
                        VarCounter++;

                        for(Iterator j=oC.listSubClasses();j
                            .hasNext();){
                        OntClass oSub=(OntClass)j.next();
                        if(!oSub.isAnon()){
//                              System.out.println(oSub.
    getLocalName()+
//                              " is subclass of "+oC.
    getLocalName());

                        Individual iv2=PositiveStatement.
                            createIndividual();
```

```
                    iv2.addProperty(predicate,
                        subClassOf);
                    iv2.addProperty(subject, oSub);
                    iv2.addProperty(object, oC);

                    }
                    }
                    }

            }
    }

    public void facts(){

            //instances
            for (Iterator i=mIn.listClasses();i.hasNext
                ();){

                    OntClass oC=(OntClass)i.next();
                    String ns=oC.getNameSpace();
                    if(!(oC.isAnon()||ns.equals(owl)||ns
                        .equals(rdfs))){

                        for(Iterator j=oC.
                            listInstances();j.hasNext
                            ();){

                                Individual oI=(
                                    Individual)j.next
                                    ();

                                Individual iv1=
                                    PositiveStatement
                                    .createIndividual
                                    ();
                                iv1.addProperty(
                                    predicate, type);
                                iv1.addProperty(
                                    object, oC);
                                iv1.addProperty(
                                    subject, oI);

                        }
                    }
            }
```

```
                //properties
                for (Iterator i=mIn.listStatements();i.
                    hasNext();){

                        Statement oS=(Statement)i.next();

                        RDFNode obj=oS.getObject();
                        RDFNode sub=oS.getSubject();
                        Property pro=oS.getPredicate();

                        String ns=pro.getNameSpace();

                        if(!(ns.equals(rdf)||ns.equals(owl)
                            ||ns.equals(lkif)||ns.equals(rdfs
                            ))){
                                Individual iv1=
                                    PositiveStatement.
                                    createIndividual();
                                iv1.addProperty(predicate,
                                    pro);
                                iv1.addProperty(object, obj)
                                    ;
                                iv1.addProperty(subject, sub
                                    );
                        }
                }
        }


public void subProperty(){
        for (Iterator i=mIn.listOntProperties();i.hasNext()
            ;){

                OntProperty oP=(OntProperty)i.next();
                String ns=oP.getNameSpace();
                if(!(oP.isAnon()||ns.equals(owl)||ns.equals(
                    rdfs))){

//                      Individual iv1=PositiveStatement.
    createIndividual();
//                      iv1.addProperty(predicate, type);
//                      iv1.addProperty(object, oP);
//                      Individual ivc=Variable.
    createIndividual();
//                      iv1.addProperty(subject, ivc);
//                      VarCounter++;
```

```
                        for(Iterator j=oP.listSubProperties
                            ();j.hasNext();){
                                OntProperty pSub=(
                                    OntProperty)j.next();
                                if(!pSub.isAnon()){
//                                      System.out.println(
    pSub.getLocalName()+
//                                      " is subProperty of
    "+oP.getLocalName());

                                      Individual iv2=
                                          PositiveStatement
                                          .createIndividual
                                          ();
                                      iv2.addProperty(
                                          predicate,
                                          subPropertyOf);
                                      iv2.addProperty(
                                          subject, pSub);
                                      iv2.addProperty(
                                          object, oP);

                                }
                        }
                    }

        }
}

}

package translator;

import java.util.ArrayList;
import java.util.Formatter;
import java.util.Iterator;
import java.util.List;

import com.hp.hpl.jena.ontology.OntClass;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntModelSpec;
import com.hp.hpl.jena.ontology.UnionClass;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.Property;
import com.hp.hpl.jena.rdf.model.RDFNode;
```

```
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.rdf.model.Statement;

public class Lkif2Sexp {

        OntModel mIn;
        OntClass Rule;
        OntClass PositiveStatement;
        OntClass NegativeStatement;
        OntClass Statement;

        Property predicate;
        Property object;
        Property subject;
        Property antecedent;
        Property consequent;
        Property unless;
        Property type;

        String rdfs="http://www.w3.org/2000/01/rdf-schema#";
        String owl="http://www.w3.org/2002/07/owl#";
        String rdf="http://www.w3.org/1999/02/22-rdf-syntax-
           ns#";
        String lkif="file:ontologies/
           EU_directive_LKIF_rules_1.0/LKIFv4.1.owl#";
        String variable="file:ontologies/
           EU_directive_LKIF_rules_1.0/variable.owl#";

        List<unionCache> unionList;
        int VarCounter=0;

        public Lkif2Sexp(String inFile){

                //model
                mIn = ModelFactory.createOntologyModel(
                   OntModelSpec.OWL_MEM, null
                );
                mIn.read(inFile);
                mIn.loadImports();

                //classes
                Rule = mIn.getOntClass( lkif+"Rule" );
                PositiveStatement = mIn.getOntClass(lkif+"
                   PositiveStatement" );
                NegativeStatement = mIn.getOntClass(lkif+"
                   NegativeStatement" );
```

```
                Statement = mIn.getOntClass(lkif+"Statement"
                    );

                //properties
                predicate = mIn.getProperty(lkif+"predicate"
                    );
                object = mIn.getProperty(lkif+"object");
                subject = mIn.getProperty(lkif+"subject");
                antecedent = mIn.getProperty(lkif+"
                    antecedent");
                consequent = mIn.getProperty(lkif+"
                    consequent");
                unless =mIn.getProperty(lkif+"unless");
                type=mIn.getProperty(rdf+"type");

                unionList=new ArrayList<unionCache>();

        }

        public String run(){

                StringBuilder sb= new StringBuilder();
                sb=ruleQuery(sb);
                sb=stQuery(sb);
//              sb=factQuery(sb);
                sb=writeUnion(sb);

                System.out.println(sb);
                return sb.toString();
        }

        public StringBuilder ruleQuery(StringBuilder sb){

                Formatter formatter = new Formatter(sb);
                for(Iterator i=Rule.listInstances();i.
                    hasNext();){
                        Resource iRule=(Resource)i.next();
                        formatter.format("(rule %1s\n\t(if(
                            and", iRule.getLocalName());
                        for(Iterator j=iRule.listProperties(
                            antecedent);j.hasNext();){
                                Resource iStatement=((
                                    Statement)j.next()).
                                    getResource();
                                formatter.format("\n\t\t");
```

```
                        writeSt(formatter,iStatement
                            );
                }
                formatter.format(")\n");

                for(Iterator j=iRule.listProperties(
                    unless);j.hasNext();){
                        Resource iStatement=((
                            Statement)j.next()).
                            getResource();
                        formatter.format("\n unless:
                            %1s", iStatement);
                        writeSt(formatter,iStatement
                            );
                }

                for(Iterator j=iRule.listProperties(
                    consequent);j.hasNext();){
                        Resource iStatement=((
                            Statement)j.next()).
                            getResource();
                        formatter.format("\t");
                        writeSt(formatter,iStatement
                            );
                        formatter.format("))\n\n");
                }
        }

        formatter.format("\n\n");
        return sb;
}

public StringBuilder stQuery(StringBuilder sb){

        Formatter formatter = new Formatter(sb);
        int counter1=0,counter2=0;
        for(Iterator i=PositiveStatement.
            listInstances();i.hasNext();){
                Resource iSt=(Resource)i.next();
                if(iSt.hasProperty(predicate)){
                        Resource pro=iSt.getProperty
                            (predicate).getResource()
                            ;

                        if(pro.toString().equals(
```

```
"http://www.w3.org/2000/01/
    rdf-schema#subClassOf")){

        Resource stObject=
            iSt.getProperty(
            object).
            getResource();
        Resource stSubject=
            iSt.getProperty(
            subject).
            getResource();

        formatter.format("(
            rule*_subClass_%1
            s\n\t", counter1
            ++);
        formatter.format("(
            if\t(%1s_?x)\n",
            stSubject.
            getLocalName());
        formatter.format("\t
            \t(%1s_?x)))\n\n"
            ,stObject.
            getLocalName());
}else if(pro.toString().
    equals(
"http://www.w3.org/2000/01/
    rdf-schema#subPropertyOf"
    )){

        Resource stObject=
            iSt.getProperty(
            object).
            getResource();
        Resource stSubject=
            iSt.getProperty(
            subject).
            getResource();

        formatter.format("(
            rule*_
            subProperty_%1s\n
            \t", counter2++);
        formatter.format("(
            if\t(%1s_?x_?y)\n
            ",stSubject.
```

```
                                        getLocalName());
                                    formatter.format("\t
                                        \t(%1s ?x ?y)))\n
                                        \n",stObject.
                                        getLocalName());
                                }
                        }
                }

                return sb;
        }

        public StringBuilder factQuery(StringBuilder sb){

                Formatter formatter = new Formatter(sb);

                formatter.format("(rule* facts");

                for(Iterator i=PositiveStatement.
                    listInstances();i.hasNext();){
                        Resource iSt=(Resource)i.next();
                        Resource pro=iSt.getProperty(
                            predicate).getResource();
                        if(pro.toString().equals(
                        "http://www.w3.org/1999/02/22-rdf-
                            syntax-ns#type")){
                                Resource stObject=iSt.
                                    getProperty(object).
                                    getResource();
                                Resource stSubject=iSt.
                                    getProperty(subject).
                                    getResource();

                                formatter.format("\n\t(%1s \
                                    t%2s)", stObject.
                                    getLocalName(),stSubject.
                                    getLocalName());
                        }else if(!(pro.toString().equals("
                            http://www.w3.org/2000/01/rdf-
                            schema#subClassOf")
                                        ||pro.toString().
                                            equals("http://
                                            www.w3.org
                                            /2000/01/rdf-
                                            schema#
                                            subPropertyOf")))
```

```
                                {
                            Resource stObject=iSt.
                                getProperty(object).
                                getResource();
                            Resource stSubject=iSt.
                                getProperty(subject).
                                getResource();
                            Resource stPredicate=iSt.
                                getRequiredProperty(
                                predicate).getResource();
                            if(!(stObject.isAnon()||
                                stSubject.isAnon())){
                                    formatter.format("\n
                                        \t(%1s \t%2s \t%3
                                        s)", stPredicate.
                                        getLocalName()
                                                    ,
                                                        stObject
                                                        .
                                                        getLocalName
                                                        ()
                                                        ,
                                                        stSubject
                                                        .
                                                        getLocalName
                                                        ()
                                                        )
                                                        ;
                            }
                        }
                    }
                    formatter.format(")");
                    return sb;
            }

            private void writeSt(Formatter formatter,Resource
                stat){

                    String sObject="",sSubject="",prefix="";

                    // find which class this 'stat' instance
                        belongs to


                    String st=stat.getProperty(type).getObject()
                        .toString();
```

```
if(st.equals(PositiveStatement.toString())){
        if(stat.hasProperty(object)){
                RDFNode ob=stat.getProperty(
                    object).getResource();

                if(ob.canAs(OntClass.class))
                    {

                        OntClass obC=((
                            OntClass)ob.as(
                            OntClass.class));

                        if(obC.isUnionClass
                            ()){
                                //detect
                                    union
                                    class
                                UnionClass
                                    ucs=obC.
                                    asUnionClass
                                    ();
                                //put union
                                    class
                                    info into
                                     union
                                    cache
                                unionCache
                                    uch=new
                                    unionCache
                                    ("union"+
                                    this.
                                    VarCounter
                                    ++);
                                sObject=uch.
                                    getName()
                                    ;
                                for(Iterator
                                     it=ucs.
                                    listOperands
                                    ();it.
                                    hasNext()
                                    ;){
                                        OntClass

                                            uc
```

```
                                        =(
                                        OntClass
                                        )
                                        it
                                        .
                                        next
                                        ()
                                        ;
                                    uch.
                                        unions
                                        .
                                        add
                                        (
                                        uc
                                        .
                                        getLocalName
                                        ()
                                        )
                                        ;

                                }
                                unionList.
                                    add(uch);

                            }else{
                                sObject=stat
                                    .
                                    getProperty
                                    (object).
                                    getResource
                                    ().
                                    getLocalName
                                    ();

                                String va=
                                    stat.
                                    getProperty
                                    (object).
                                    getResource
                                    ().
                                    getProperty
                                    (type).
                                    getObject
                                    ().
                                    toString
                                    ();
```

```
                                        if(va.equals
                                            (variable
                                            +"
                                            Variable"
                                            )){
                                                sObject
                                                    =
                                                    "
                                                    ?
                                                    "
                                                    +
                                                    sObject
                                                    ;
                                        }
                                    }
                                }else{
                                    sObject=stat.
                                        getProperty(
                                        object).
                                        getResource().
                                        getLocalName();

                                    String va=stat.
                                        getProperty(
                                        object).
                                        getResource().
                                        getProperty(type)
                                        .getObject().
                                        toString();
                                    if(va.equals(
                                        variable+"
                                        Variable")){
                                            sObject="?"+
                                                sObject;
                                    }
                                }
                            }else{
                                return;
                            }
                        }else if (st.equals(NegativeStatement.
                            toString())){
                            if(stat.hasProperty(object)){
                                sObject=stat.getProperty(
                                    object).getResource().
                                    getLocalName();
```

```
                    String va=stat.getProperty(
                        object).getResource().
                        getProperty(type).
                        getObject().toString();
                    if(va.equals(variable+"
                        Variable")){
                            sObject="?"+sObject;
                    }
                    prefix="not ";
            }
    }

    //subject
    if(stat.hasProperty(subject)){

            sSubject=stat.getProperty(subject).
                getResource().getLocalName();

            String va=stat.getProperty(subject).
                getResource().getProperty(type).
                getObject().toString();
            if(va.equals(variable+"Variable")){
                    sSubject="?"+sSubject;
            }

    }else{
            return;
    }

    Resource stPredicate;
    if(stat.hasProperty(predicate)){
            stPredicate=stat.getProperty(
                predicate).getResource();
    }else{
            return;
    }

    if(stPredicate.toString().equals(rdf+"type")
        ){
            formatter.format(prefix+"( %s %s)" ,
                sObject, sSubject);
    }else{
            formatter.format(prefix+"( %s %s %s)
                " , stPredicate.getLocalName(),
                            sObject, sSubject);
    }
```

```
        }

        public StringBuilder writeUnion(StringBuilder sb){

                Formatter formatter = new Formatter(sb);

                for(unionCache uch:unionList){
                        formatter.format(uch.toString());
                }

                return sb;
        }

        private class unionCache{

                private String name;
                public List<String> unions;

                public unionCache(String name){
                        this.setName(name);
                        unions=new ArrayList<String>();
                }
                public void setName(String name) {
                        this.name = name;
                }
                public String getName() {
                        return name;
                }

                public String toString(){
                        String out="\n(rule*␣unions\n\t(if(
                            or";

                        for(String u: unions){
                                out+="\n\t\t("+u+"␣?x)";
                        }
                        out+="\n\t("+name+"␣?x)))";
                        return out;
                }
        }
}
```

Bibliography

[Boer, A., 2008] Boer, A. (2008). Generating LKIF Rules from DL Axioms. Amsterdam.

Generating LKIF Rules from DL Axioms

Alexander Boer
Leibniz Center for Law
Faculty of Law
University of Amsterdam
The Netherlands

Leibniz Center for Law
Faculty of Law
University of Amsterdam
PO Box 1030
1000BA Amsterdam
The Netherlands

tel: +31 20 525 3485
fax: +31 20 525 3495
http://www.leibnizcenter.org

Corresponding author:
Alexander Boer
tel: (+31)20 525 3499
A.W.F.Boer@uva.nl

1 Introduction

This is the specification of the OWL axioms into LKIF rules translator. Note that the OWL axioms are obtained by interrogating a non-reasoning OWL model. We are translating the given axioms: not everything entailed by them.

The translator has a number of different translation modes:

1. Translation of LKIF rules in RDF to s-expression syntax.
2. Translation of RDF statements into LKIF rules in RDF;
3. Translation of OWL axioms into LKIF rules in RDF;

The three translators translate all objects of the appropiate type in a Jena RDF model. The responsibility of selecting the appropriate content is left to the user.

2 RDF Syntax for LKIF Rules

Translating the LKIF Rules RDF syntax into LKIF Rules S-Expression syntax is trivial.

For LKIF rules we assume for now the following simplified syntax: A rule is of the form `(if R (`$S_1 S_2 .. S_n$`) (`$S_1 S_2 .. S_n$`))`, where S is a sentence. A sentence S is of the form `(CN ?x)`, `(not (CN ?x))`, `(P ?x ?y)`, or `(not (P ?x ?y))`, where `?x` is a variable, `CN` a OWL concept name, and `P` an OWL property name. `R` is the rule name, which is not given in the case of OWL axioms. Unless clauses obviously do not occur in LKIF rules translated from OWL axioms.

Note that a two-stage translation process is used: translate OWL axioms into LKIF Rules in RDF, and translate LKIF Rules in RDF into s-expressions.

3 RDF Statements

RDF statements are assertions (about individual i_1 .. i_n). These are easy to translate:

1. `(type` i `C)` or `(C` i`)` becomes `(if () (C` i`))`
2. `(P` $i_1 i_2$`)` becomes `(if () (P` $i_1 i_2$`))`

Assertions that have an OWL class as argument where an individual is expected are not translated. Since we need to check whether something is a **rdfs:Class** we need entailment and therefore a reasoning model to make these checks.

4 OWL Class Axioms

The OWL Class axioms are harder. For ease of reading the description logic syntax is sometimes given where no ambiguity on the syntactic structure of the corresponding

OWL statement exists, while in most cases the OWL syntactic structure is given. This of course has no significance for the translation process.

First, the OWL axioms are rewritten into the right form: the description is rewritten to a conjunction of positive or negative literals, and axioms are rewritten to the `(subClassOf BODY HEAD)` form. We eliminate inconvenient negations and disjunctions. Disjunctions (`unionOf`) in class descriptions for instance require a newly generated concept name CN_{bnode} (blank node), if they occur on the `HEAD` of a class axiom. Easy to solve are rules with disjunctions on the `BODY` side. Also `someValuesFrom` restrictions should only occur in the `BODY`, and `allValuesFrom` restrictions only in the `HEAD`, before translation to rules takes place, because variables in rules should be read as universally quantified. The required existential variables are not obtained by rewriting: the best thing we can do is approximate.

These rules are applied as long as there remain candidates for normalization, with the axiom rewrite rules taking precedence. Let C be a class description, CN a class name, and CN_{bnode} an automatically generated blank node name in the following lists.

The following are axiom rewrite rules:

1. `(equivalentTo` $C_1 C_2$`))` rewritten as `(subClassOf` $C_1 C_2$`) (subClassOf` $C_2 C_1$`)`
2. `(disjointWith` C_1 C_2`)` $[C_1 \sqsubseteq \neg C_2]$ rewritten as `(subClassOf` C_1 `(complementOf` C_2`))`
3. `(subClassOf` C `(unionOf` $C_1 C_2$`))` rewritten as `(disjointWith` C CN_{bnode}`)` `(equivalentTo` CN_{bnode} `(intersectionOf (complementOf` C_1`)` `(complementOf` C_2`)))`
4. `(subClassOf (unionOf` $C_1 C_2$`)` C`)` rewritten as `(subClassOf` C_1 C`)` `(subClassOf` C_2 C`)`
5. **(unsound)** `(subClassOf (restriction (onProperty` P`)(allValuesFrom` CN`))` C`)` $[\forall P.CN \sqsubseteq C]$ rewrites to `(subClassOf` CN_{bnode} C`)` `(subClassOf` CN_{bnode} `(restriction (onProperty` P`)(allValuesFrom` CN`)))`, or is replaced by nothing
6. **(unsound)** `(subClassOf` C `(restriction (onProperty` P`)(someValuesFrom` CN`)))` $[C \sqsubseteq \exists P.CN]$ rewrites to `(subClassOf` C CN_{bnode}`)` `(subClassOf (restriction (onProperty` P`)(someValuesFrom` CN`))` CN_{bnode}`)`, or is replaced by nothing

A note on the unsound rules: both the rewrite and the removal are unsound. They make some sense in relation to the LKIF rules specification which states that a rule with more than one conclusion (a conjunction of conclusions), can be read as an argument for each of these conclusions. In OWL this is not the case. The option of dropping leaves us with a dangling CN_{bnode}, but it could be augmented with a label that states "all P are CN" and still play a useful role.

The following rules rewrite descriptions C:

1. `(complementOf (complementOf` C`))` $[\neg\neg C]$ rewritten as C

2. `(complementOf (unionOf` C_1C_2`))` $[\neg(C_1 \sqcup C_2)]$ rewritten as $(\neg C_1 \sqcap \neg C_2)$
3. `(complementOf (intersectionOf` C_1C_2`))` rewritten as $(\neg C_1 \sqcup \neg C_2)$
4. `(unionOf` C_1C_2`))` $[C_1 \sqcup C_2]$ rewritten as CN_{bnode}, assert `(equivalentTo` CN_{bnode} `(unionOf` C_1C_2`))`
5. `(restriction (onProperty` P`)(someValuesFrom` C`))` $(\exists P.C)$ rewritten as `(restriction (onProperty` P`)(someValuesFrom` CN_{bnode}`))`, assert `(equivalentTo` $CN_{bnode}C$`)`
6. `(restriction (onProperty` P`)(allValuesFrom` C`))` $(\forall P.C)$ rewritten as `(restriction (onProperty` P`)(allValuesFrom` CN_{bnode}`))`, assert `(equivalentTo` $CN_{bnode}C$`)`
7. Optionally only in the `HEAD` to reduce the number of rules: `(restriction (onProperty` P`)(someValuesFrom` CN`))` rewritten as CN_{bnode}, assert `(equivalentTo` CN_{bnode} `(restriction (onProperty` P`)(someValuesFrom` CN`)))`
8. Optionally only in the `BODY` to reduce the number of rules: `(restriction (onProperty` P`)(allValuesFrom` CN`))` rewritten as CN_{bnode}, assert `(equivalentTo` CN_{bnode} `(restriction (onProperty` P`)(allValuesFrom` CN`)))`

Rewrite any spurious nested conjunctions into a single conjunction `(intersectionOf` $C_1C_2C_3$ `..)`. Note that in all rules above we have assumed that any list `(intersectionOf` $C_1C_2C_3$..) can be read as `(intersectionOf` C_1 `(intersectionOf` C_2 `(intersectionOf` C_3 `..)))`. Same goes for `unionOf` lists. For an implementation in an RDF store this should be insignificant, since the order of conjuncts (disjuncts) is not significant to the API.

Next the translation of the remaining axioms, which is simple after the normalization has been completed. Let C be a class description, and CN a class name:

1. `(subClassOf` C_1C_2`)` $(C_1 \sqsubseteq C_2)$ rewritten to `(if (`C_1`)(`C_2`))`
2. `(subClassOf` CN_1 `((restriction (onProperty` P`)(allValuesFrom` CN_2`))` C`))` rewritten to `(if ((`CN_1 `x)(P x y)) (`CN_2 `y))`, where `?y` is a newly introduced variable, i.e. does not already occur in the rule
3. Only if description rewrite rule 8 is not applied: `(subClassOf` C_1 `(intersectionOf ((restriction (onProperty` P`)(allValuesFrom` CN_2`))) (`C_2`))` rewritten to `(if ((`C_1 `x)(P x y)) ((`CN_2 `y) (`C_2`))`, where `?y` is a newly introduced variable, i.e. does not already occur in the rule

And the remaining class descriptions C:

1. `(intersectionOf` C_1C_2`)` $(C_1 \sqcap C_2)$ rewritten to C_1C_2
2. `(complementOf` CN`)` $(\neg CN)$ rewritten to `(not` CN`)`
3. CN rewritten to `(CN ?x)`

4. `((restriction (onProperty` P`)(someValuesFrom` CN`)))` $[\exists P.CN]$ (occuring only in the body) rewritten to `(P ?x ?y) (CN ?y)`, where `?y` is a newly introduced variable, i.e. does not already occur in the rule

Cardinality restrictions cannot be done, except for the "at-least 1" constraint which is formally equivalent to "some".

5 OWL Property Axioms

For the property axioms we make no distinction between description and name:

1. `(subPropertyOf` P_1P_2`)` $(P_1 \sqsubseteq P_2)$ becomes `(if ((`P_1 `?x ?y))((`P_2 `?x ?y)))`
2. `(equivalentProperty` P_1P_2`)` $(P_1 \equiv P_2)$ becomes `(if ((`P_1 `?x ?y))((`P_2 `?x ?y))) (if ((`P_2 `?x ?y))((`P_1 `?x ?y)))`
3. `(inverseOf` P_1P_2`)` becomes `(if ((`P_1 `?x ?y))((`P_2 `?y ?x))) (if ((`P_2 `?x ?y))((`P_1 `?y ?x)))`

The other ones we'll skip for now. These seem easy to reconstruct, given the presentation on class axioms above, and the fact that property restrictions have only very limited variety in syntax.

6 Remarks on Expressiveness of LKIF Rules

We miss `skolem functions`. For instance we can rewrite the following:
`(subclassOf` CN_1 `(restriction (onProperty` P`)(someValuesFrom` CN_2`)))`
to:
`(if (`CN_1 `?x) ((P ?x f(?x))(`CN_2`(f(?x))))).`

`f(?x)` is here a skolem function, which (given a proper interpretation function) is mapped to some object in the domain.

Instead of skolem functions, we can introduce constants, and an auxiliary functional predicate which would play a similar role to a skolem function. So, given some value of `?x`, such that (CN_1 ?x), we can conclude that there is an object called *sk* such that (P ?x sk)(C sk). Of course the incorrectness is that for any value of `?x` we obtain the same `sk`, which eventually will be mapped to a single object in the domain, which doesn't have to be true. For many purposes it's sufficient to know that there's one such object in the domain, since in DL it is quite irrelevant anyway whether objects involved in existential restrictions are the same or different. Each rule containing existentially quantified variables should use a unique constant name.

There is some unclarity on the `sameAs` and `differentFrom` predicates, and the absence of equality and inequality in LKIF Rules, or put differently, I wonder whether `(ancestor Alexander Alexander)` is indeed a valid model of of Tom's family support example, which defines only the following, or not:

```
<rule id="r2" strict="yes">
    <body><s>ancestor ?x ?z</s>
        <s>ancestor ?z ?y</s></body>
    <head><s>ancestor ?x ?y</s></head>
</rule>
```

Skolem functions and (in)equality together would help also for to alleviate problems for cardinality restrictions, although I imagine "a space shuttle has at least 270,000 moving parts" will remain problematic, for Carneades, for the translator, and for human readability.

Another point of unclarity, relating to the LKIF rules specification (D1.1, p. 92) which appears to state that a rule with more than one conclusion, can be read as an argument for each of these conclusions: Is it indeed the case that in LKIF Rules, given `(if ((P ?x) (assuming (Q ?x))) ((R ?x)(S ?x))` and `(P ?x)` and `(not (S ?x))`, `(R i)` is a valid argument conclusion, backed by `((P i) (assuming (Q ?x)))`?

LKIF Rules and LKIF Ontology

Alexander Boer
Leibniz Center for Law
Faculty of Law
University of Amsterdam
The Netherlands

Abstract. In this note I discuss two approaches to translate an OWL-DL ontology in LKIF-Rules. The goal of this translation is to enable Carneades to use also terminological knowledge cast in OWL-DL. The first one aims at translating the ontology via a KRSS syntax and translator into a DLP subset of LKIF-Rules. A second approach is to start from the XML/RDF syntax from OWL and translate these in two steps into LKIF-Rules in s-expression syntax. Although the latter solution is somewhat more complex it has the advantage that less OWL constructs get lost in translation. Therefore, I propose the latter solution for the UvA translators.

Leibniz Center for Law
Faculty of Law
University of Amsterdam
PO Box 1030
1000BA Amsterdam
The Netherlands

tel: +31 20 525 3485
fax: +31 20 525 3495
http://www.leibnizcenter.org

Corresponding author:
Alexander Boer
tel: (+31)20 525 3499
A.W.F.Boer@uva.nl

1 Introduction

The LKIF OWL ontology is specified in OWL DL, a relatively expressive description logic (cf. generally [Horrocks and Patel-Schneider, 2004, Horrocks et al., 2003, Baader and Sattler, 2001]). The LKIF Rules are inspired by Datalog, extended with a number of LKIF-specific operators described in the deliverable. The combination of OWL DL and Datalog is known to be undecidable (cf. [ter Horst, 2005, Donini et al., 1998]). The LKIF Rules are however based on another kind of semantics based on argumentation theory. The interaction between LKIF Rules and OWL DL was not fixed in the specification (cf. [Boer et al., 2007]), and is not immediately obvious given that OWL and LKIF Rules use very different styles of semantics specifications.

In the white paper [Gordon, 2007] an architecture for WP4 is proposed, based on a strategy for generating proofs, similar to the one used by RuleBurst, and one method for establishing truth of a proposition is to ask the user. If our rules knowledge base for instance contains just the rule `(if (single-mother ?x) (housing-benefit ?x))`, and we inquire about `?-(housing-benefit a)`, then the resolution function will select rules with `(housing-benefit ?x)` as head, and unify, and replace the head literal with the conjunction of literals which form the body of the rule, in this case just `?-(single-mother a)`. A rule for `(single-mother ?x)` is unavailable. A known fact `(single-mother a)` is treated like a rule with an empty body, and a method for obtaining them is to ask the user for `(single-mother a)`. Note that asking `(housing-benefit ?x)` of the user is strictly speaking also a valid move, but not very sensible for a dialog pragmatics point of view since it is what the user asked.

2 Example Problem

The following is an example, which is representative of the problem the interaction between OWL DL and LKIF Rules may cause:

1. `housing-benefit` $\sqsubseteq$ `social-benefit`
2. `(if (and (single-mother ?x) ..) (housing-benefit ?x))`
3. `(if (and (social-benefit ?x) ..) (foo ?x))`

The problem occurs if you try to inquire about `(foo a)`: the system will ask the user about `(social-benefit a)`, but not about `(single-mother a)`. The dialog strategy sketched in the architecture proposal leads to incompleteness.

3 Proposed Solution

The proposed solution is to treat OWL DL axioms as LKIF Rules. The easiest method for doing so is to *translate* OWL DL axioms into LKIF Rules. There are several possible approaches to doing this, oulined in the last section.

Whichever approach is selected, it has no bearing on the implementation of the architecture, of proof of concept applications, or on modeling. The developers may be assured that the architecture only deals with rules. The modelers should keep in mind that modeling an ontology should as a matter of principle not be guided by application-specific concerns.

The translator(s) from OWL DL to LKIF Rules will be made by the University of Amsterdam, as outlined in the WP4 workplan.

3.1 Caveat Emptor

The solution proposed here also does not definitively solve the more general problem of the semantics of OWL DL + LKIF Rules: it merely constrains it. It is a practical solution for this application scenario, and does not address what to do with a `KB` that contains for instance `(if (single-mother ?x) (housing-benefit ?x))`, ¬`(housing-benefit a)`, and `(single-mother a)`.

The University of Amsterdam proposes alternative methods of combining LKIF Rules with OWL DL axioms.

3.2 About LKIF Semantics and Modeling in LKIF

To proceed with modeling in LKIF we need an easy-to-understand manual. The question of precise semantics is also not of concern here. We are perfectly capable of describing the meaning of an LKIF Rule or an OWL axiom without recourse to a formal specification (which would anyway only be comprehensible by logicians).

4 About LKIF Syntax

Both LKIF Rules and LKIF OWL DL axioms will be displayed for human readers and modelers in the form of s-expressions. For OWL DL we use the existing KRSS syntax (used by the Racer OWL DL/KRSS theorem prover). For LKIF Rules [Boer et al., 2007] already contains a proposal. Translating XML/OWL to s-expressions is trivial. The other way around takes a little more work.

For editing OWL axioms either an OWL editor (Protege, TopBraid) or a text editor for s-expressions is used. For the LKIF Rules s-expressions will be used.

For OWL DL obviously OWL syntax is used as XML syntax. A new XML syntax for LKIF Rules is proposed. I strongly prefer an OWL syntax for LKIF Rules. Whatever the choice, the modeler will typically not use an OWL or XML editor to write rules, so legibility is not a concern. I am of the opinion that processing OWL will be easier and less resource-intensive than processing a new XML language. XML is what is exchanged by the architecture.

5 Approaches to translating OWL DL into LKIF Rules

In [Gordon, 2007] it is proposed to use an OWL DLP subset of an OWL DL ontology, which *is* translatable to LKIF Rules.

I object to a restriction to DLP. Although it may be true that "80% of OWL ontology axioms are within the DLP fragment"[1], there are certain constructs – for instance disjointness of concepts, or partitioning of a concept into subconcepts – that are central to taxonomies, certainly at the top level, but cannot be expressed in DLP. Also equality and inequality cannot be expressed in DLP, so we cannot for instance check whether a relation is functional. See also section 3.4 of [Boer et al., 2007] for a general summary of description axioms that cannot be rephrased as a rule.

It is also not practical to extract a DLP ontology from an OWL DL ontology, because a lot of useful material is thrown away. Take for instance a terminological axiom $C_1 \equiv \neg C_2$. Depending on context this can be translated into LKIF rules

1. `(if (not (`C_1 `x)) (`C_2 `x))`,
2. `(if (`C_2 `x) (not (`C_1 `x)))`,
3. `(if (not (`C_2 `x)) (`C_1 `x))`,
4. `(if (`C_1 `x) (not (`C_2 `x)))`, or
5. any subset of these.

These reformulations do not entirely respect the semantics of the $\neg$ operator in OWL DL, and they are not covered by OWL DLP, but they are all consistent with the original axiom and typically would be desirable as useful arguments pro or con a certain proposition. Restriction to the DLP fragment forces us to ignore the axiom, which is unwise.

Direct translation to an LKIF rule is superior to a detour via DLP. Note by the way that this may force us to make an informed *choice* between the possible translations, as one would do in a Prolog program. This is the case for a technical reason if the LKIF rules processor would not check for non-termination of proofs, and in other cases dialog considerations may prompt such a choice. Take for instance the axiom (`left` $\equiv \neg$ `right`): if `(if (left x) (not (right x)))` makes sense for dialog purposes, and `(if (not (right x)) (left x))` does not.

We propose the following:

1. There are two kinds of queries to an LKIF knowledge base: LKIF queries (`KB + rules` $\models \phi$), where ϕ is a positive or negative literal (fact-form), and generic terminological queries (`KB` $\models \phi$) that can be posed through any generic OWL API.
2. LKIF queries optionally initiate a dialog with the user, depending on the environment used. The dialog with the user is restricted to queries about fact-form propositions that *unify with a premise in the body of a rule* in the `KB`. If a user however proactively asserts a proposition into the `KB`, the full expressiveness of OWL DL is available.
3. An LKIF query is a fact-form proposition that *unifies with the head of a rule* in the `KB`.

[1] Provided you count triples as axioms.

4. The remaining problem is to *optionally* generate additional LKIF rules for sets of fact-form propositions from pairs of different rules that have terminological connections. There are different possible options for generating this set S:

   (a) For any pair of fact-form propositions `(`p_1 `?x)` and `(`p_2 `?x)` occurring in two different rules, regardless of whether they occur in the head or the body, if `KB` $\models p_1 \sqsubseteq p_2$ or `KB` $\models p_2 \sqsubseteq p_1$, then the corresponding bridging rule `(if (`p_1 `?x) (`p_2 `?x))` (`(if (`p_2 `?x) (`p_1 `?x)))` is in S.

   (b) For any pair of sets s_1, s_2 of propositions in the power set of the set of (fact-form) propositions in the rules, if the conjunction of members in s_1 is c_1 and the conjunction of members in s_2 is c_2, then, if `KB` $\models c_1 \sqsubseteq c_2$ or `KB` $\models c_2 \sqsubseteq c_1$, then the corresponding bridging rule is in S'. In $S \subset S'$ there are no rules r for which there is a rule r' in S such that $r' \models r$[2]. It goes without saying that this is very inefficient, but has to be done only once offline.

About point 3: Besides being technically necessary, this principle is also plausible from a functional point of view. If you consult a legal expert system, you are most likely inquiring about your legal position. Legal qualifications of facts are typically found at the head of a constitutive rule. The system is not likely to be competent wrt. to propositions which do not occur at the head of a rule.

Referenties

[Baader and Sattler, 2001] Baader, F. and Sattler, U. (2001). An Overview of Tableau Algorithms for Description Logics. *Studia Logica*, 69:5–40.

[Boer et al., 2007] Boer, A., Gordon, T. F., van den Berg, K., Di Bello, M., Förhécz, A., and Vas, R. (2007). Specification of the legal knowledge interchange format. Deliverable 1.1, Estrella.

[Donini et al., 1998] Donini, F. M., Lenzerini, M., Nardi, D., and Schaerf, A. (1998). AL-log: Integrating Datalog and Description Logics. *Journal of Intelligent Information Systems*, 10(3):227–252.

[Gordon, 2007] Gordon, T. (2007). The ESTRELLA open platform for legal knowledge systems: white paper. Internal note for estrella consortium, Fraunhofer FOKUS.

[Horrocks and Patel-Schneider, 2004] Horrocks, I. and Patel-Schneider, P. (2004). Reducing OWL entailment to description logic satisfiability. *J. of Web Semantics*, 1(4):345–357.

[Horrocks et al., 2003] Horrocks, I., Patel-Schneider, P., and van Harmelen, F. (2003). From shiq and rdf to owl: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26.

[2] I leave this unspecified: please apply common sense here.

[ter Horst, 2005] ter Horst, H. J. (2005). Combining RDF and part of OWL with rules: Semantics, decidability, complexity. In *International Semantic Web Conference*, pages 668–684.