Lab 1

**Create your own XML DTD, XSD, and RNC or RNG**

Consider these examples of XML documents for clause sets consisting of zero or more facts f (Prolog: f.) and zero or more 'backward' rules c <- p (Prolog: c :- p.), in any order ("*myurl*" will be replaced as explained below):

<?xml version="1.0" standalone="no"?>

<!DOCTYPE clauses SYSTEM "*myurl*">

 <clauses>

 <fact> f </fact>

 <rule> <conc> c </conc> <prem> p </prem> </rule>

 </clauses>

<?xml version="1.0" standalone="no"?>

<!DOCTYPE clauses SYSTEM "*myurl*">

<clauses>

 <rule> <conc> c1 </conc> <prem> p1 </prem> </rule>

 <fact> f1 </fact>

 <rule> <conc> c2 </conc> <prem> p2 </prem> </rule>

 <fact> f2 </fact>

 <fact> f3 </fact>

</clauses>

Inductively complete this XML DTD (overwrite the "..." lines) for such clause sets:

<!ELEMENT clauses (................)>

<!ELEMENT rule (................)>

<!ELEMENT fact (#PCDATA)>

<!ELEMENT ........ (................)>

<!ELEMENT ........ (................)>

Upload a file containing (just) your completed DTD somewhere in your personal Web space under a URL of your choice, here called "*myurl*".

Validate the above document examples and other ones online:

> Go to

<http://validator.w3.org/#validate_by_input>

> Paste in and edit the examples at “Enter the Markup to validate:”

> like here for *myurl* = http://www.cs.unb.ca/~boley/cs6795swt/country-address.dtd

> (already uploaded − on the course site − for convenience):

<?xml version="1.0" standalone="no"?>

<!DOCTYPE address SYSTEM "http://www.cs.unb.ca/~boley/cs6795swt/country-address.dtd">

<address>

<name>Xaver M. Linde</name>

<street>Wikingerufer 7</street>

<town>10555 Berlin</town>

<country>Germany</country>

</address>

> Hit the 'Check' button

> You should basically get:

**This document was successfully checked … .**

You can now experiment with variations of the original XML DTD and document examples.

 **XSD**

If you reformulate the original XML DTD into an XML Schema Definition (XSD), W3C's XSV Validator for XML Schema at <http://www.w3.org/2001/03/webdata/xsv> can be employed instead.

You can do this DTD-XSD reformulation in analogy to the reformulation of the Country Address example in the Lecture Notes. For getting familiar with XSV, you can experiment with the example’s uploaded instance and XSD here (in some browsers, to see XML details, you have to enter control-U or select Tools | Web Developer | Page Source):

<http://www.cs.unb.ca/~boley/cs6795swt/address.xml>
<http://www.cs.unb.ca/~boley/cs6795swt/country-address.xsd>

Then copy & edit these for your personal Web space, working your way from the <address> example to the <clauses> example.

 **RNC and RNG**

If you reformulate the original EBNF grammar / XML DTD into Relax NG Compact Syntax (RNC) or reformulate the above XML XSD into Relax NG XML Syntax (RNG), Henri Sivonen’s validation service at [http://validator.nu](http://validator.nu/) can be employed instead.

You can do these reformulations in analogy to the reformulations of the Country Address example in the Lecture Notes. For getting familiar with Validator.nu, you can experiment with the example’s uploaded instance and RNC or RNG here (at ‘Address’, copy the instance URL; at ‘Schemas’, copy the RNC or RNG URL; select “Be lax about HTTP Content-Type” and possibly “Show Source”):

<http://www.cs.unb.ca/~boley/cs6795swt/address.xml>
<http://www.cs.unb.ca/~boley/cs6795swt/country-address.rnc>
<http://www.cs.unb.ca/~boley/cs6795swt/country-address.rng>

Then copy & edit these for your personal Web space, working your way from the <address> example to the <clauses> example.

Lab 2

**Generate an RDF graph**

Go to the W3C RDF Validation Service:[**http://www.w3.org/RDF/Validator/**](http://www.w3.org/RDF/Validator/)

Copy and edit an RDF serialization of your choice over the example in the text field or just use the example itself, at “**Display Result Options**:”select “Triples and Graph”, and

Hit the 'Parse RDF:' button.

If there are “Error Messages” fix your RDF serialization until an RDF graph shows you were successful. Try to explain this “Graph of the data model” generated from the RDF serialization. Can you also explain the correspondence between it and the generated “Triples of the Data Model”?

What are the main differences between this RDF ‘Validation’ Service and an XML Validation Service such as the ones used in Lab 1?

Lab 3

**Protégé: OWL Ontology Development & Reasoning and SWRL Rules**

1. Download Protégé from <http://protege.stanford.edu/download/download.html> and install/test Protégé on your computer (suggested as of Sep 2011: Protégé 4.1 Release). Proceed **as described** at <http://protegewiki.stanford.edu/wiki/Protege4GettingStarted> (except that Protege 4.0 now should be 4.1), and load, navigate, and modify the Pizza ontology.
2. Perform reasoning **as described** with the ‘built-in’ reasoner HermiT.
3. Develop (a version of) one of the ontologies discussed in class or in an assignment, and again perform reasoning.

Alternatively to step 2. you can use the ‘plugin’ reasoner Pellet **as described**, or:

2’ Download Pellet from <http://clarkparsia.com/pellet/download>, and unzip it. (Suggested as of Oct 2009: [pellet-1.5.2](http://clarkparsia.com/pellet/download/pellet-1.5.2)).

 Perform some reasoning tasks in Pellet. This can be done in the following three ways:

* 1. Use the plug-in in Protégé.
	2. Access Pellet’s basic functionality through the command line interface.

Running “java -jar lib/pellet.jar” in the command line, you will get help on all the parameters supported by Pellet.

* 1. Use the Pellet reasoner in a Java program through the OWL API. A good place to get familiar with the OWL API is <http://owlapi.sourceforge.net/documentation.html>. Download and try the examples listed there.

Optionally, try the Semantic Web Rule Language (SWRL) Tab (SWRLTab) if available in your version of Protégé. See link from <http://protege.cim3.net/cgi-bin/wiki.pl?SWRLLanguageFAQ>.

Lab 4

**Transform RuleML to a normal form with XSLT**

As shown in <http://ruleml.org/0.89/#XSLT-Based%20Normalizer>, an XSLT stylesheet has been developed by David Hirtle for normalizing the syntax used in a ‘stripe-skipped’ RuleML instance, which has been extended for RuleML 1.0 (<http://ruleml.org/1.0/xslt/normalizer/100_normalizer.xslt>) by Derek Smith. It reconstructs all skipped role tags to be in a fully-expanded, normal form. For example, the compact version of the 'own' example (<http://ruleml.org/1.0/xslt/normalizer/own_compact.ruleml>),

 **<Assert mapClosure="universal">**

 **<Implies>**

 **<And>**

 **<Atom>**

 **<Rel>buy</Rel>**

 **<Var>person</Var>**

 **<Var>merchant</Var>**

 **<Var>object</Var>**

 **</Atom>**

 **<Atom>**

 **<Rel>keep</Rel>**

 **<Var>person</Var>**

 **<Var>object</Var>**

 **</Atom>**

 **</And>**

 **<Atom>**

 **<Rel>own</Rel>**

 **<Var>person</Var>**

 **<Var>object</Var>**

 **</Atom>**

 **</Implies>**

 **...**

 **<Atom>**

 **<Rel>keep</Rel>**

 **<Ind>Mary</Ind>**

 **<Ind>XMLBible</Ind>**

 **</Atom>**

 **</Assert>**

is normalized to the expanded version (<http://ruleml.org/1.0/xslt/normalizer/own_expanded.ruleml>):

 **<Assert innerclose="universal">**

 **<formula>**

 **<Implies>**

 **<if>**

 **<And>**

 **<Atom>**

 **<op><Rel>buy</Rel></op>**

 **<arg index="1"><Var>person</Var></arg>**

 **<arg index="2"><Var>merchant</Var></arg>**

 **<arg index="3"><Var>object</Var></arg>**

 **</Atom>**

 **<Atom>**

 **<op><Rel>keep</Rel></op>**

 **<arg index="1"><Var>person</Var></arg>**

 **<arg index="2"><Var>object</Var></arg>**

 **</Atom>**

 **</And>**

 **</if>**

 **<then>**

 **<Atom>**

 **<op><Rel>own</Rel></op>**

 **<arg index="1"><Var>person</Var></arg>**

 **<arg index="2"><Var>object</Var></arg>**

 **</Atom>**

 **</then>**

 **</Implies>**

 **</formula>**

 **...**

 **<formula>**

 **<Atom>**

 **<op><Rel>keep</Rel></op>**

 **<arg index="1"><Ind>Mary</Ind></arg>**

 **<arg index="2"><Ind>XMLBible</Ind></arg>**

 **</Atom>**

 **</formula>**

 **</Assert>**

(See a [comparison of these two files](http://ruleml.org/1.0/xslt/normalizer/own_compare.htm) generated using [HTML Diff](http://www.componentsoftware.com/products/htmldiff/).)

Additional examples are located in the [Normalizer](http://ruleml.org/1.0/xslt/normalizer/) directory.

1. Work yourself into the new XSLT features of [100\_normalizer.xslt](http://www.ruleml.org/1.0/xslt/normalizer/100_normalizer.xslt) using the XML Bible, 3rd Edition, Chapter 15: <http://www.cafeconleche.org/books/bible3/chapters/ch15.html> .
2. Go to the W3C XSLT Service:
<http://www.w3.org/2005/08/online_xslt/>

and give it the above URIs.

To see the result, in some browsers you need to do "View | Page Source".
3. How could the inverse direction be specified in XSLT? Is it easier?

Lab 5

**Prolog and POSL examples running in jDREW and OO jDREW**

If you are interested in the Prolog examples running in a Java implementation, use the original jDREW as described below.

If you are interested in an extended (GUI-based) functionality, start the newer OO jDREW from <http://www.jdrew.org/oojdrew/demo.html> or download its Java implementation from <http://www.jdrew.org/oojdrew/download.html>, and use POSL versions of the examples (POSL variables are “?”-prefixed).

1. This lab assumes you have installed Java on your computer. There are several options to achieve this as there are a number of other IDE’s available. The most popular now, while rather large, is Eclipse <http://www.eclipse.org/downloads/>. Another popular alternative is the tiny Emacs-based JDEE <http://jdee.sunsite.dk/> and see <http://www.ibm.com/developerworks/java/library/j-emacs/> . See <http://www.cs.unb.ca/courses/gen> for some instructions from FCS for an older installation of JDEE that still appears to work well. Any FCS lab computer probably has Java installed. You may use any Java environment with which you are already familiar.
2. Create a new project directory and populate it with the contents of this zip file: [j-DREW.zip](http://www.cs.unb.ca/~boley/cs6795swt/j-DREW.zip). Note that this jDREW version is a simplified version – no purpose or warranty is stated or implied. Compile the file PrologTopLevel.java. Depending on your system this means invoking something like javac PrologTopLevel.java which should invoke java’s compiler on all Java files in the directory. Note that you may need to explicitly refer to jar files on the classpath, so try
javac -classpath .;jdom.jar;xerces.jar PrologTopLevel.java.
3. Try running the class PrologTopLevel (something like java PrologTopLevel, perhaps needing the jar files on the classpath). Note that one of the lines in the PrologTopLevel.java file is dt.parseDCFile("examples.dc"); This means that the file examples.dc, which contains the examples discussed in class during the Prolog Tutorial, will be read into the discrimination tree data structure. To find out who is holding a party, your interaction should look something like this, where your input is in red:
Enter a query
**hold\_party(Who).**$top:-hold\_party(helen).
more? (y/n) [y] **y**
No (more) answers
4. Answer these questions – you may need to be creative in your use of j-DREW, and may need to recompile various parts.
	1. Prolog assumes the order of clauses is important. Does j-DREW?
	2. Does j-DREW assume that the order of goals in a clause is important? How can you tell?
	3. Recompile PrologTopLevel.java with SHOW\_TREE set to TRUE. What effect does that have? Try out the on\_route example with the query on\_route(Where).
	4. Change the on\_route example so that the second clause reads
	on\_route(Place):-
	 on\_route(NewPlace),
	 move(Place,Method,NewPlace).
	Since “,” means “and” and since A and B is the same as B and A, it should have no effect, right? Check it out with the query on\_route(Where).
	5. Change the dfsolver from a depthFirstSolutionIterator to an iterativeDepthFirstSolutionIterator. What effect does that have on the modified on\_route example using the same query? Revert the literals in the second clause back to their original order. What is the order of solutions now? What criteria does the iterativeDepthFirstSolutionIterator use to order its solutions?
	6. Look into the DCTree.java and find the INTERNAL\_TRACE variable, and set it to TRUE. Recompile and run a small query and also the on\_route(Where) query. Can you make sense of some of the voluminous information? It is probably easier to make sense of the output from the depthFirstIterator than it is from the iterativeDepthFirstIterator.

Ref Web Site: <http://www.cs.unb.ca/~boley/cs6795swt/labs.html>