RDF processing for Bioconductor: Redland

©2005 VJ Carey <stvjc@channing.harvard.edu>

October 3, 2006

Contents

1 Introduction 1
2 Illustration 2
   2.1 Simple manipulations with a fragment of GO 2
   2.2 BioPAX Level 1 5
   2.3 BioPAX level 2 8
   2.4 HumanCyc 9
3 Future work 12

1 Introduction

Resource Description Framework (RDF) is a graphical model for information. RDF
statements are ordered triples of the form (subject, predicate, object). Subjects and
objects are viewed as nodes in a directed graph, and predicates are viewed as arcs
in the graph. RDF is a key component of current developments towards a semantic
web, with considerable work completed on web resource metadata representation and
exchange using RDF. A richer metadata model is provided by OWL (Web Ontology
Language), but most OWL models are serialized using XML/RDF. Thus, as we will
illustrate, various public OWL resources can be processed by this package.

Redland is the name of an open source software project downloadable from librdf.org. Redland is a C language library with bindings provided to a variety of other
languages. Redland is highly modular, and allows developers to drop in components to
substitute for base functionalities. Because metadata resources can be very voluminous,
such flexibility is important. A solution to the problem of persistent storage of indexed
metadata is provided through the use of BerkeleyDB serializations of Redland models.

Redland is an R package that provides interfaces to facilities of Redland. Configu-
ration support is currently limited. You will be able to use Rredland if you do a stock
installation of librdf and BerkeleyDB. If you have these resources in nonstandard locations, you can set the Makevars variables in src to reflect your configuration. You may need to set LD_LIBRARY_PATH.

2 Illustration

2.1 Simple manipulations with a fragment of GO

Eric Jain of ISB-CH has provided an RDF serialization of the UniProt database and associated annotation resources, including an RDF serialization of GO. A fragment of this serialization is distributed with the Rredland package.

> library(Rredland)
A redland RDF world has been created in package:Rredland as .GredlWorld.

> gofrag <- system.file("RDF/gopart.rdf", package = "Rredland")

Here we dump the first 10 lines of this document as text:

> readLines(gofrag, n = 10)

[1] "<?xml version='1.0' encoding='UTF-8'?>"
[6] "<rdfs:comment>The distribution of mitochondria, including the mitochondrial genome, from one cell to another during cell division.</rdfs:comment>"
[9] "</rdf:Description>"

This could be processed as an XML document, but let’s use Redlands modeling facilities. First we need to set up a URI object for the model source document.

> gouri <- makeRedlURI(paste("file:" gofrag, sep = ""))

Now we read from this document. We will set the useCore option to use in-memory storage.

> gof <- readRDF(gouri)
> gof
redlModel object, status=open.

We are handed back an S4 object of class redlModel.

> getClass("redlModel")

Slots:

Name: ref storagetype stateEnv world
Class: externalptr character environment redlWorld

We need to use the model accessor to get to the model reference.

We can easily compute the number of statements (also computed with show()):

> size(gof)

[1] 69

We can also transform to a data frame:

> godf <- as(gof, "data.frame")
> godf[1:4, ]

<table>
<thead>
<tr>
<th>subject</th>
<th>predicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 urn:lsid:uniprot.org:go:0000001</td>
<td>1 <a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a></td>
</tr>
<tr>
<td>2 urn:lsid:uniprot.org:go:0000001</td>
<td>2 <a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a></td>
</tr>
<tr>
<td>3 urn:lsid:uniprot.org:go:0000001</td>
<td>3 <a href="http://www.w3.org/2000/01/rdf-schema#comment">http://www.w3.org/2000/01/rdf-schema#comment</a></td>
</tr>
<tr>
<td>4 urn:lsid:uniprot.org:go:0000001</td>
<td>4 <a href="http://www.w3.org/2000/01/rdf-schema#subClassOf">http://www.w3.org/2000/01/rdf-schema#subClassOf</a></td>
</tr>
</tbody>
</table>

1

2

3 "The distribution of mitochondria, including the mitochondrial genome, into daughter 

4

We see that long text strings can cause a problem for rendering.

> as.character(godf[1:4, 3])

[1] "urn:lsid:uniprot.org:ontology:Concept"
[2] ""mitochondrion inheritance""
[3] "The distribution of mitochondria, including the mitochondrial genome, into daughter 

The data frame representation is useful for splitting up the statement set.

```r
> bypred <- split(godf, as.character(godf$predicate))
> names(bypred)
[1] "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
[2] "http://www.w3.org/2000/01/rdf-schema#comment"
[4] "http://www.w3.org/2000/01/rdf-schema#subClassOf"

> sapply(bypred, nrow)
http://www.w3.org/1999/02/22-rdf-syntax-ns#type       16
http://www.w3.org/2000/01/rdf-schema#comment        15
http://www.w3.org/2000/01/rdf-schema#label           19
http://www.w3.org/2000/01/rdf-schema#subClassOf       17
urn:lsid:uniprot.org:ontology:obsolete             2

The `subClassOf` predicate helps determine the DAG structure:

```r
classof <- bypred$"http://www.w3.org/2000/01/rdf-schema#subClassOf"
> classof[, -2]

<table>
<thead>
<tr>
<th>subject</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>urn:lsid:uniprot.org:go:0000012</td>
<td>urn:lsid:uniprot.org:go:0006281</td>
</tr>
</tbody>
</table>
2.2 BioPAX Level 1

The BioPAX pathway ontologies are available.

```r
> bp1 <- makeRedlURI(paste("file:", system.file("RDF/biopax-level1.owl", + package = "Rredland"), sep = ""))
> bp1m <- readRDF(bp1)
> size(bp1m)

[1] 630
```

This is a manageable object, so we convert to data frame:

```r
> bp1df <- as(bp1m, "data.frame")
> sapply(bp1df[1:5, ], substring, 1, 70)

subject
[1,] "http://www.biopax.org/release/biopax-level1.owl"
[2,] "http://www.biopax.org/release/biopax-level1.owl"
[3,] "http://www.biopax.org/release/biopax-level1.owl#physicalEntityParticip"
[4,] "http://www.biopax.org/release/biopax-level1.owl#chemicalStructure"
[5,] "http://www.biopax.org/release/biopax-level1.owl#physicalEntityParticip"

predicate
[1,] "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
[2,] "http://www.w3.org/2000/01/rdf-schema#comment"
[3,] "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
[4,] "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
[5,] "http://www.w3.org/2002/07/owl#disjointWith"

object
[1,] "http://www.w3.org/2002/07/owl#Ontology"
[2,] ""This is version 1.4 of the BioPAX Level 1 ontology. The goal of the"
[3,] "http://www.w3.org/2002/07/owl#Class"
[4,] "http://www.w3.org/2002/07/owl#Class"
[5,] "http://www.biopax.org/release/biopax-level1.owl#chemicalStructure"
```

The namespace qualifications make the strings difficult to render. A simple approach uses substitution up to the pound sign, preceded by eliminating any XSD postfix information.

```r
> strip2pound <- function(x) gsub(".*#", ",", cleanXSDT(as.character(x))
> sapply(bp1df[1:5, ], strip2pound)

subject                     predicate
[1,] "http://www.biopax.org/release/biopax-level1.owl" "type"
[2,] "http://www.biopax.org/release/biopax-level1.owl" "comment"
```
Working with a data frame, it is easy to filter statements of interest. Suppose we wish to determine all the instances of \texttt{owl#Class} in the model.

```r
> isTypeOwlClass <- grep("owl#Class", as.character(bp1df[, 3]))
> strip2pound(bp1df[isTypeOwlClass, 1])
```

We see a number of decipherable terms, and some tokens of the form (rnnn...). The latter are called blank nodes. These are created to define classes that have no names, but that are implicitly defined in the model. For example, a class that is the union of entity and physicalEntity is a blank node in this model.

To get the detailed commentary on a class definition, the following function can be used:

```r
> getClassComment <- function(term, df, nsPref = "http://www.biopax.org/release/biopax-level1.owl#", +   commPred = "http://www.w3.org/2000/01/rdf-schema#comment",
```

```r
[3,] "physicalEntityParticipant"      "type"
[4,] "chemicalStructure"             "type"
[5,] "physicalEntityParticipant"      "disjointWith"
[1,] "Ontology"
[2,] "\"This is version 1.4 of the BioPAX Level 1 ontology. The goal of the BioPAX group is to...
[3,] "Class"
[4,] "Class"
[5,] "chemicalStructure"

> isTypeOwlClass <- grep("owl#Class", as.character(bp1df[, 3]))
> strip2pound(bp1df[isTypeOwlClass, 1])
```
+ doChop = TRUE, nword = 12) {
+ ind <- which(as.character(df[, 1]) == paste(nsPref, term,
+ sep = " ") & as.character(df[, 2]) == commPred)
+ chopLong(cleanXSDT(as.character(bp1df[ind, 3])), nword = nword)
+ }

> cat(getClassComment("chemicalStructure", bp1df))

"A utility class that defines a small molecule structure. An instance of this class can also define additional information about a small molecule, such as its chemical formula, names, and synonyms. This information is stored in the slot STRUCTURE-DATA, in one of two formats: the CML format (see URL www.xml-cml.org) or the SMILES format (see URL www.daylight.com/dayhtml/smiles/). The STRUCTURE-FORMAT slot specifies which format is used. An example is the following SMILES string, which describes the structure of glucose-6-phosphate:

'C(OP(=O)(O)O)CH1(CH(O)CH(O)CH(O)CH(O)O1)'."

> cat(getClassComment("biochemicalReaction", bp1df))

"A conversion interaction in which one or more entities (substrates) undergo covalent changes to become one or more other entities (products). The substrates of biochemical reactions are defined in terms of sums of species. This is what is typically done in biochemistry, and, in principle, all of the EC reactions should be biochemical reactions.

Example: ATP + H2O = ADP + Pi.

In this reaction, ATP is considered to be an equilibrium mixture of several species, namely ATP4-, HATP3-, H2ATP2-, MgATP2-, MgHATP-, and Mg2ATP. Additional species may also need to be considered if other ions (e.g. Ca2+) that bind ATP are present. Similar considerations apply to ADP and to inorganic phosphate (Pi). When writing biochemical reactions, it is important not to attach charges to the biochemical reactants and not to include ions such as H+ and Mg2+ in the equation. The reaction is written in the direction specified by the EC nomenclature system, if applicable, regardless of the physiological direction(s) in which the reaction proceeds. (This definition from EcoCyc)

NOTE: Polymerization reactions involving large polymers whose structure is not explicitly captured should generally be represented as unbalanced reactions in which the monomer is consumed but the polymer remains unchanged, e.g. glycogen + glucose = glycogen."
2.3 BioPAX level 2

Here we check the classes available in BioPAX level 2.

```r
> bp2 <- makeRedlURI(paste("file:", system.file("RDF/biopax-level2.owl", + package = "Rredland"), sep = ""))
> bp2m <- readRDF(bp2)
> size(bp2m)
[1] 910

> bp2df <- as(bp2m, "data.frame")
> isTypeOwlClass <- grep("owl#Class", as.character(bp2df[, 3]))
> strip2pound(bp2df[isTypeOwlClass, 1])

[1] "dataSource"               "openControlledVocabulary"
[3] "xref"                    "bioSource"
[5] "externalReferenceUtilityClass" "dnaParticipant"
[7] "rnaParticipant"          "dna"
[9] "physicalEntityParticipant" "proteinParticipant"
[11] "complexParticipant"      "smallMoleculeParticipant"
[13] "transportWithBiochemicalReaction" "biochemicalReaction"
[15] "transport"               "complexAssembly"
[17] "conversion"              "physicalEntity"
[19] "interaction"             "entity"
[21] "pathway"                 "unificationXref"
[23] "relationshipXref"        "publicationXref"
[25] "physicalInteraction"     "smallMolecule"
[27] "protein"                 "rna"
[29] "complex"                 "sequenceLocation"
[31] "confidence"              "evidence"
[33] "chemicalStructure"       "utilityClass"
[35] "pathwayStep"             "sequenceInterval"
[37] "sequenceSite"            "sequenceFeature"
[39] "modulation"              "catalysis"
[41] "control"                 "experimentalForm"
[43] "(r1159910716r141)"       "(r1159910716r156)"
[45] "(r1159910716r159)"       "(r1159910716r166)"
[47] "(r1159910716r170)"       "(r1159910716r173)"
[49] "(r1159910716r176)"       "(r1159910716r182)"
[51] "(r1159910716r186)"       "(r1159910716r189)"
[53] "(r1159910716r201)"       "(r1159910716r204)"
[55] "(r1159910716r207)"       "(r1159910716r211)"
```
2.4 HumanCyc

The BioCyc project (www.biocyc.org) is a collection of pathway/genome databases in a variety of structures. The data resources are available to academic researchers, and a registration/download process must be completed for access. We illustrate use of Rredland to work with the BioPAX encoding of HumanCyc. This is 19MB of RDF and an in-core storage model is not likely to be satisfactory. We will use the default BerkeleyDB storage approach.

> humu <- makeRedlURI(paste("file:", "humancyc.owl", sep = ")))  
> humm <- readRDF(humu, storageType = "bdb", storageName = "hucyc")

Note that the vignette cannot assume that you have this OWL file. After the above commands, we have

```
-rw-r--r-- 1 stvjc stvjc 59723776 Jul 28 13:09 test-sp2o.db  
-rw-r--r-- 1 stvjc stvjc 39538688 Jul 28 13:07 test-po2s.db  
-rw-r--r-- 1 stvjc stvjc 57499648 Jul 28 13:07 test-so2p.db
```

These are the BerkeleyDB hashes representing aspects of the graph.

It is not too difficult to transform into a data frame.

> hudf <- as(humm, "data.frame")  
> husubs <- as.character(hudf[, 1])  
> hupreds <- as.character(hudf[, 2])  
> huobs <- as.character(hudf[, 3])  
> table(hupreds)

```
hupreds

http://www.biopax.org/release/biopax-level1.owl#AUTHORS
  31432
http://www.biopax.org/release/biopax-level1.owl#CELLULAR-LOCATION
  2800
http://www.biopax.org/release/biopax-level1.owl#COFACTOR
  11
http://www.biopax.org/release/biopax-level1.owl#COMMENT
  1231
http://www.biopax.org/release/biopax-level1.owl#COMPONENTS
  36
http://www.biopax.org/release/biopax-level1.owl#CONTROL-TYPE
  36
http://www.biopax.org/release/biopax-level1.owl#CONTROLLED
  2216
http://www.biopax.org/release/biopax-level1.owl#CONTROLLER
```
To find the named pathways,

```r
> isPw <- grep("pathway", husubs)
> isNa <- grep("NAME", hupreds)
> isnp <- intersect(isPw, isNa)
> cleanXSDT(huobs[isnp][1:10])
```

[1] ""biosynthesis of aspartate and asparagine; interconversion of aspartate and asparagus"
[2] ""serine and glycine biosynthesis"
[3] ""alanine biosynthesis II"
[4] ""alanine biosynthesis I"
[5] ""alanine biosynthesis III"
[6] ""superpathway of alanine biosynthesis"
[7] ""arginine biosynthesis III"
[8] ""citrulline biosynthesis"
[9] ""asparagine biosynthesis I"
[10] ""aspartate biosynthesis and degradation"

So we see in the predicate set what kinds of relationships are described, and we get a glimpse of the pathway names addressed in this resource.

Note that there is no need to parse the data once the Berkeley DB hashes are made available. The BDBSexists option on readRedlModel can be used to revive a model-hash association.
3 Future work

We will need to take unions of RDF models and C code will be required for that. We need R interfaces to Redland approaches to model filtering. Some graph/set-theoretic activities can be introduced to bring some RDF/RDFS inferencing in.