TRANSLATOR Coreference Resolution

KBC

2023-03-29

This code includes:

1. A representation of annotations in the BioNLP-ST format
2. Semantic rules for coreference resolution
3. Syntactic rules for coreference resolution

BioNLP-ST data: Representing in R

Question: Why use the BioNLP-ST format?

Answer: Because despite putting a bunch of time into the XML and JSON formats, neither of them really worked out very well. Plus, when I looked through what information is in which kind of annotation project, this format gets me to everything that I need, particularly with respect to mapping named entity and coreference annotations to character offsets and mapping between those and the sentence boundaries. Those sentence boundaries are really important for coreference resolution…

Let's try a generalized class for representing a BioNLP-ST "annotation" in R. The goal is for this to work across any type of label, and if all annotations are in the same format, it will.

The format seems to be: one file per article/paper/whatever. Let's say "document." A line seems to have the following things in it:

1. An identifier, numbered sequentially and with a tag that might or might not have some morphology.
2. A tag.
3. A start index.
4. An end index.
5. The covered text (although maybe there isn't any for some types of annotation?)

These five things are in three tab-separated columns:

1. Identifier
2. Space-separated tag, start index, and end index
3. The covered text

Example:

T1 NN 0 7 Complex T2 sentence 0 96 Complex trait analysis of the mouse striatum: independent QTLs modulate volume and neuron number T3 NN 8 13 trait T4 NN 14 22

analysis T5 IN 23 25 of T6 DT 26 29 the T7 NN 30 35 mouse T8 NN 36 44 striatum T9 : 44 45 :

Why does the word *Complex* come before the sentence that contains it, while the rest of the words come after it? I have no idea.

What can we get out of the CRAFT annotations?

We have seen the rationale for using the BioNLP-ST format in general. What can we get from it for the specific purposes of coreference resolution in the CRAFT corpus?

NB that we won't necessarily have the same thing for any other data, but maybe we can tag other data automatically… I guess that if we *can't* build taggers off of the CRAFT annotations, then the project was a loss, so let's operate on the assumption that we *can,* which seems like a reasonable bet.

1. Sentence index is available in the CONLLU format, but I don't see it anywhere else. The sentence index is continuous throughout a paper, i.e. it does not pay attention to paragraph boundaries. (In fact, I don't think it even pays attention to the distinction between a title, a section heading, and a paragraph of text.)
2. I don't see a paragraph index anywhere. Not necessarily a problem, since the sentence indices are continuous.

- *CONLLU: paragraph index, sentence index*. No spans.
- *CONLLX:* no paragraph index, no sentence index, no spans.
- *Treebank:* no paragraph index, no sentence index, no spans.
- *Sections and typography:* no paragraph index, no sentence index. Spans for formatting.
- *Coreference:* no paragraph index, no sentence index. *Spans for text of markables*.
- *Named entities:* no paragraph index, no sentence index. *Spans for text of markables*.

So, it seems like there are some minor hacks for comparing strings and for comparing semantic classes, and that's almost it, within the realm of decency. If we want any of the sentence position stuff, we will have to do some truly filthy hacking to map the coref/named entity annotations, which have character offsets but no sentence indices, to the CONLLU data, which has sentence indices, but no character offsets…

1. We can get noun group and text information out of the coreference annotations. With that, we have boundaries, and we can check for exact string matches and for case-toggled string matches. Maybe for a first step, we count how many within-document sets of each there are?
2. We can map spans in the coreference annotations to spans in the named entity annotations. That way we can see if we have semantic class matches. Maybe for a second step, we count those?

Class BioNLP-ST defined

1. Constructor
2. Method: convert between a list and a dataframe NO

3. Method: convert between a dataframe and a list NO
4. Test driver

TODO: need a way to handle discontinuous spans.

Question: What's the best way to handle semantic class labels? I get leaf nodes from any normalization system, which is wonderful if it matches, but if that matches, it's probably redundant to string match, more often than not… I certainly don't want to toss that level of granular information out, though.

The highest possible node–let's say the equivalent of just knowing what kind of named entity you got labelled with–is much more likely to be widely useful, I suspect. If we want that, we need to trim it off of the concept ID, as far as I know. We should certainly use it…

So, we're resolved that we will keep/use both the named entity class, and the concept ID. The next question: how do we represent them? The choices: as factors, or as strings. Factors seem conceptually cleaner. Strings are simpler from the point of programming logic, though, I suspect… Maybe start with strings, and then refactor (haha) to factors at some later point if it seems clear that it would be helpful?

```
# Concise discussion of object-oriented programming with S4 in R:
# https://www.datamentor.io/r-programming/s4-class/

# example of defining the class with setClass(), from
https://www.datamentor.io/r-programming/s4-class/:
#setClass("student", slots=list(name="character", age="numeric",
GPA="numeric"))
setClass("BioNLPST",
         slots = list(
           filename = "character",
           id = "character",
           label = "factor", # should be a factor, but I need to define that
somehow, so for now make it a character, and convert that with as.factor()
           start = "numeric",
           end = "numeric",
           #discontinuous = "boolean",
           covered.text = "character",
           label.and.indices = "character",
           original.line = "character"))


#bionlp.test.01 <- new(Class = "BioNLPST",
#    filename = "myfilename",
#    id = "T1",
#    label = "GENE",
#    start = 100,
#    end = 150,
#    covered.text = "this length should equal that of start to end offsets",
#    label.and.indices = "GENE 100 150",
#    original.line = "boopity boop")
```

```
# print() and show() are both pretty space-intensive, so let's write
something that would effectively overwrite print()--and if I figure out how
to do that, this can be the implementation!
# args: bionlpst is an object of class BioNLPST
bioprint <- function(bionlpst) {
  # TODO: test that you've been passed the right class
  # TODO: test that mandatory slots are filled (why? not the responsibility
  # of this function...)
  print(paste("Filename:", bionlpst@filename))
  print(paste("ID:", bionlpst@id))
  print(paste("Label:", bionlpst@label))
  print(paste("Covered text:", bionlpst@covered.text))
  print(paste("Start position:", bionlpst@start))
  print(paste("End position:", bionlpst@end))
} # close function definition bioprint()

#bioprint(bionlp.test.01) # was using the old df-based type.
# TODO: rewrite test with the new class BioNLPST

# bng = base noun group, which actually at this point is an object
# of class BioNLPST...
getBioNlpStId <- function(bng) {
  return(bng@id)
} # close function definition getBioNlpStId()
```

Class IdentPair defined

This class represents a pair of anaphor and antecedent *or* candidate antecedent. At some point we should have a representation of a coref chain, but let's start simple for now. We should probably also be able to represent singletons in some similar way.

For a tidy representation, maybe we do this…

1. ID for the pair
2. IDs of its two elements
3. The sieve that got it

This would let you group by pair ID. If pair IDs are sortable, well… you could sort by them. So, these should probably be numeric.

It would also let you group by each anaphor and (potential) antecedent. So, for example, if you group by anaphor, you could have a list of its potential antecedents and their scores. Crucially: you could have more than one potential antecedent. I guess you could also group by (potential) antecedent and see which ones are connected to (potentially) more than one anaphor. I'm not sure how you would use that algorithmically, but from an analytical perspective, it would help you to characterize the complexity/difficulty of the problem for a given dataset.

It would also let you group by sieves, which seems pretty valuable for evaluation.

Assumptions:

1. Not representing singletons (bad)
2. Not representing appositives (bad)

```
library(tidyverse)

setClass("IdentPair",
         slots = list(
           filename = "character",
           pair.id = "numeric",
           anaphor = "BioNLPST",
           candidate = "BioNLPST",
           sieve = "character", # maybe I need a *score*, plus an *array* of
sieves...
           anaphor.id = "character", # see note about antecedent ID
           antecedent.id = "character", # check for consistency with class
                                        # BioNLPST
           #covered.texts = paste(getString(anaphor), "::",
getString(candidate)) # unfortunately, can't do this here, so...
           #covered.texts = paste(anaphor@covered.text, "::",
candidate@covered.text)
           # SHIT--there doesn't seem to be any way to get the covered texts
from within this constructor,
           # so I guess I'll have to have a setter for that...
           # ...but, I still do need to define that
           covered.texts = "character"
           ))

# is there a way to "just" override print()?
# This function prints an IdentPair more prettily than calling structure()
# or something like that.
identPrint <- function(ident.pair) {
  # TODO: validate that you did, indeed, get passed an IdentPair
  print(paste("IdentPair", getPairID(ident.pair)))
  print(paste("Sieve:" , getSieve(ident.pair)))
  print(paste("Anaphor:", getBioNlpStId(anaphor)))
  print(paste("Antecedent:", getBioNlpStId(candidate)))
  print(paste())
  print(paste())
} # close function definition identPrint()
print("NO TESTS IMPLEMENTED FOR IdentPair class")

## [1] "NO TESTS IMPLEMENTED FOR IdentPair class"
```

Operations on IdentPairs

```
library(tidyverse)

DEBUG.CHUNK <- FALSE
```

```
# OK, this makes no sense. You should be getting passed an IdentPair, not its
details!!!
# PICK UP FROM HERE
#printIdentPair <- function(filename, pair.id, sieve, anaphor.id,
antecedent.id) {
#  print(paste(pair.id, sieve, anaphor.id, antecedent.id))
#  print(paste())
#} # close function definition printIdentPair()

printIdentPair <- function(ident.pair) {
  #print("TRYING TO PRINT AN IDENT PAIR...")
  identPair.printing.output <- ""
  identPair.printing.output <- paste(identPair.printing.output,
getPairID(ident.pair))
  identPair.printing.output <- paste(identPair.printing.output,
getCoveredTexts(ident.pair))
  identPair.printing.output <- paste(identPair.printing.output,
paste("Anaphor:", getBioNlpStId(anaphor)))
  identPair.printing.output <- paste(identPair.printing.output,
paste("Antecedent:", getBioNlpStId(candidate)))

  identPair.printing.output <- paste(getPairID(ident.pair), "ANA:",
getBioNlpStId(anaphor), "ANT:", getBioNlpStId(candidate),
getCoveredTexts(ident.pair))

  print(identPair.printing.output)
} # close function definition printIdentPair()

getPairID <- function(ident.pair) {
  if(DEBUG.CHUNK) { print("In function getPairID()") }
  return(ident.pair@pair.id)
}
print("NO TESTS IMPLEMENTED FOR getPairID()")

## [1] "NO TESTS IMPLEMENTED FOR getPairID()"

getSieve <- function(ident.pair) {
  return(ident.pair@sieve)
}

# args: an IdentPair and two bngs (BioNLPST objects)
setCoveredTexts <- function(ident.pair, anaphor, candidate) {
  ident.pair@covered.texts <- paste(getString(anaphor), "::",
getString(candidate))
} # close function definition setCoveredTexts()

# args: an IdentPair
# returns: string
getCoveredTexts <- function(ident.pair) {
```

```
  return(ident.pair@covered.texts)
}

print("NO TESTS IMPLEMENTED FOR getSieve()")

## [1] "NO TESTS IMPLEMENTED FOR getSieve()"
```

Convert a line of BioNLP-ST data to a BioNLPST object

This will replace the function to turn a line of BioNLP-ST data to a BioNLPST object.

```
DEBUG <- FALSE
# Given a line from a file in BioNLP-ST format, parse it, make a new
# BioNLPST object, and return that
# args: a line from a file in BioNLP-ST format (see above)
lineToBioNLPST <- function(input.line) {
  input.line.contents <- unlist(strsplit(input.line, "\u0009"))
# \u0009 is hex for tab. Might not be necessary.
  if (DEBUG) { print(paste("Input line splits to a vector of length",
length(input.line.contents))) }
  if (DEBUG) { print(paste("Input line contents split to:",
input.line.contents))}
  if (DEBUG) {
    print(paste("Element 1:", input.line.contents[1]))
    print(paste("Element 2:", input.line.contents[2]))
    print(paste("Element 3:", input.line.contents[3]))
  }
  id <- input.line.contents[1]
  label.and.indices <- input.line.contents[2]
  covered.text <- input.line.contents[3]

  # separate the label and indices into the label, the start index,
  # and the end index ("index" means character offset, sorry)
   label.and.indices.split <- unlist(strsplit(label.and.indices, "\\s"))
   label <- label.and.indices.split[1]
   start <- label.and.indices.split[2]
   end <- label.and.indices.split[3]
   label <- as.factor(label)
   start <- as.numeric(start) # has to be numeric for later processing
   end <- as.numeric(end) # has to be numeric for later processing
bionlp.new <- new(Class = "BioNLPST",
    filename = "myfilename",
    id = id,
    label = label,
    start = start,
    end = end,
    covered.text = covered.text,
    label.and.indices = label.and.indices,
    original.line = input.line)
```

```
    return(bionlp.new)
} # close function definition lineToBioNLPST()
```

Line of BioNLP-ST data in, dataframe out

This will be deprecated after replacement with code to read in a line of BioNLP-ST data and output a BioNLPST object.

This function does the basic parsing of a line of data in the BioNLP-ST format into its components.

The fact that this makes a dataframe is an artifact of an earlier conception of the approach. A separate function allows you to turn it into a list() (see above).

Note that this code chunk contains some useful examples of how to utilize the stuff returned from various string-manipulating functions.

```
library(tidyverse)
# nice post on futzing with columns: https://www.marsja.se/how-to-remove-a-column-in-r-using-dplyr-by-name-and-index/

# args: a line from a .bionlp file
# returns: a tibble with the following columns:
# id: the identifier in the .bionlp file
# tag: could be POS, "sentence," whatever
# start: beginning character  offset
# end: ending character offset
# covered.text: the original text covered by the span start -> end
parse.bionlp.st <- function(input.line) {
  FUNCTION.DEBUG <- FALSE
  if (FUNCTION.DEBUG) { print("In function as.bionlp.st().")}
  if (FUNCTION.DEBUG) { print(input.line) }
  # see here: https://stackoverflow.com/questions/46518228/return-value-of-strsplit
  input.line.contents <- unlist(strsplit(input.line, "\u0009"))
  #input.line.contents <- str_split(string = input.line, pattern = "\u0009")
# \u0009 is hex for tab. Might not be necessary--I think my bug might be related to not understanding what str_split() returns... Is it a list??
  if (FUNCTION.DEBUG) { print(paste("Input line splits to a vector of length", length(input.line.contents))) }
  if (FUNCTION.DEBUG) { print(paste("Input line contents split to:", input.line.contents))}
  if (FUNCTION.DEBUG) {
    print(paste("Element 1:", input.line.contents[1]))
    print(paste("Element 2:", input.line.contents[2]))
    print(paste("Element 3:", input.line.contents[3]))
  }
  #if (TRUE) { print("Type of return from str_split:", typeof(input.line.contents))}
  input.line.df <- data.frame(id = input.line.contents[1],
```

```
                                tag.and.offsets = input.line.contents[2],
                                covered.text = input.line.contents[3],
                                input.line = input.line)
  #colnames(input.line.df) <- c("input.line")
  if (TRUE) { head(input.line.df) }
  # convert the data frame into a tibble:
  input.line.df <- as_tibble(input.line.df)

  # now you should have a 3-column tibble.
  # next you need to split the column containing the tag and character
offsets into three separate columns.
  #input.line.df <- tidyr::separate(data = input.line.df,
  #                                 col = tag.and.offsets,
  #                                 into = c("tag", "start", "end"),
  #                                 sep = " ")
  if (TRUE) { print(head(input.line.df)) }
  # break up the tag and offsets into separate columns:
  input.line.df <- tidyr::separate(data = input.line.df,
                col = tag.and.offsets,
                into = c("tag", "start", "end"),
                sep = " ",
                remove = FALSE)
  if (TRUE) { print("After splitting tag.and.offsets into separate columns:")
}
  if (TRUE) { head(input.line.df) }
  # now you can remove the columns that contain (1) the complete input line
from
  # the file, and (2) the combination of tag and offsets that you just split
  # into separate columns.
  #input.line.df <- select(input.line.df, -c(input.line, tag.and.offsets))
  #input.line.df <- select(input.line.df, -c(tag.and.offsets)) but, maybe
this gets blown away by the call to separate that I do above?
  # THOUGHT: I actually wouldn't mind keeping it, if for no other reason than
for validating tags and offsets in later processing steps...
  if (TRUE) { print("Variable input.line.df:") }
  if (TRUE) { head(input.line.df) }

  return(input.line.df)


} # close function definition as.bionlp.st()

if (TRUE) {
  lines <- readLines("/Users/kevincohen/Dropbox/N-Z/translator-relation-
extraction/code/testData/11319941.bionlp")
  print(paste("Number of lines read in:", length(lines)))
  print("First 5 lines:")
  print(lines[1:5])
  #print("Last 5 lines:")
```

```
  # clunky, but: should print the last 5 lines
  #...but, I think it prints the whole fucking array ;-)
  #print(lines[length(lines)-4:length(lines)])
  #print("") # I just want a little space in the output
  #before the next stuff starts happening...

  # set this as you like. use it to cut down on how much output you get if
you're debugging and don't want/need to see hundreds of lines. For the
contents of an entire file, set the variable to the number of lines in the
file (length(lines)). To limit the amount of output that you have to wade
through, set it to a smaller number than that!
  #number.of.lines.to.print <- length(lines)
  number.of.lines.to.print <- 2

  if (TRUE) { print("START ITERATING OVER LINES") }
  for (i in 1:number.of.lines.to.print) {
    line <- lines[i]
    print(paste("Read in line", i, ":", line))
    line.bionlp.st <- parse.bionlp.st(line)
    print("RETURNED FROM parse.bionlp.st():")
    print(line.bionlp.st)
  } # close for-loop through file

}
```

Getting sentence offsets from the BioNLP-ST data

To figure out sentence positions (same sentence, preceding sentence, preceding sentence left edge/right edge), we want character offsets. We will get those from the CRAFT files in BioNLP-ST format, which contain lines that give character offsets for sentences, like this:

T1 NN 0 7 Complex T2 sentence 0 96 Complex trait analysis of the mouse striatum: independent QTLs modulate volume and neuron number T3 NN 8 13 trait T4 NN 14 22 analysis T5 IN 23 25 of

```
library(tidyverse) # TODO: Can I suppress the warnings about build versions?

DEBUG <- FALSE
# for now, we'll just do one file at a time. See here for a loop to open
everything in a directory: https://rpubs.com/LMunyan/363306
#sentences.directory <- "/Users/kevincohen/Dropbox/a-m/Corpora/craft-
sentences-bionlp/*.bionlp"
#sentences.df <- read.table(sentences.directory, sep = "\t", header = FALSE)
sentences.df <- read.table("/Users/kevincohen/Dropbox/a-m/Corpora/craft-
sentences-bionlp/11319941.bionlp", sep = "\t", header = FALSE)
if (DEBUG) { head(sentences.df) }
colnames(sentences.df) <- c("token.id", "tag.and.offsets", "text")
if (DEBUG) { head(sentences.df) }
sentences.tibble <- as_tibble(sentences.df)
sentences.df <- "" # just free up the memory...
```

```
# drop the columns you don't want--we can always change our minds about this
later.
# nice post on futzing with columns: https://www.marsja.se/how-to-remove-a-
column-in-r-using-dplyr-by-name-and-index/
sentences.tibble <- select(sentences.tibble, -c(token.id, text))
if (DEBUG) { head(sentences.tibble) }
sentences.tibble <- sentences.tibble %>% filter(str_detect(tag.and.offsets,
"sentence"))
if (DEBUG) { head(sentences.tibble) }
#str_split()
sentences.tibble <- tidyr::separate(data = sentences.tibble,
                col = tag.and.offsets,
                into = c("tag", "start", "end"),
                sep = " ")
# we can just drop the "sentence" tag column...
if (DEBUG) { head(sentences.tibble) }
sentences.tibble <- select(sentences.tibble, -c(tag))
if (TRUE) { head(sentences.tibble) }
```

```
## # A tibble: 6 × 2
##   start end
##   <chr> <chr>
## 1 0     96
## 2 98    106
## 3 108   118
## 4 120   213
## 5 214   399
## 6 401   408
```

Overlapping annotations

Sometimes you need to know whether or not two separate annotations cover the same text. Sometimes you need to know whether or not the text covered by one annotation is within the range of text covered by another. These functions provide that functionality.

```
# I've got a coreference annotation covering character offsets XX to YY. Is
there a named entity annotation exactly matching/within/containing that
range?

# document ID
# span start
# span end

# range() returns a vector containing the minimum and maximum of all of the
given arguments. Not quite what we want... It just gives you the minimum and
the maximum, which if you gave it a span start and span end, would be the
span start and the span end. No value added. I think what we want is, a
function that given the start and the end, will give you a vector of all of
the integers between them. Then you can ask whether or not two OTHER span
positions (start/end) are in that vector.
```

```
generateFullRange <- function(start, end) {
  fullRange <- c(start:end)
  return(fullRange)
}
testFullRange <- generateFullRange(start = 345, end = 500)
#print("345 to 500:")
#print(testFullRange)
if (400 %in% testFullRange) { print(".") }
```

```
## [1] "."
```

```
if (1000 %in% testFullRange) { print("FAIL") } else { print(".") }
```

```
## [1] "."
```

```
# OK, that works alright.
```

Span match/within/containing logic

This builds on the generateFullRange() function that is defined in the previous code chunk. Cases:

1. No overlap at all between the two.
2. Markable span matches named entity span.
3. Markable span within named entity span.
4. Markable span contains named entity span.
5. Spans overlap, markable to left of named entity.
6. Spans overlap, markable to right of named entity.

There might not be a meaningful difference between (5) and (6), in which case I guess we just call it (partial) overlap or something.

```
# question we're trying to answer: given a markable, do we know its semantic
class, and if so: what is it?

# if there is an exact span match between the markable and some named entity,
then yes, we do know its semantic class, and its semantic class is the
semantic class of the named entity.

# if the overlap is not complete (see 3-6 above), then it's a bit more
complicated, but we can use a heuristic. Let's say maybe like this: if there
is overlap between the markable and one and only one named entity, then we
assign the named entity's semantic class to the markable. If there is overlap
between the markable and more than one named entity, then we assign the
semantic class of the named entity that covers the largest span. Or the
smallest! ;-)
```

Implementation of some semantic rules

If the semantic classes of the two NGs match, then they're more likely to be coreferential than if they don't.

Limitations of this implementation:

1. Both NGs have to be labelled with a semantic class.
2. I don't have an easy *and effective* way to take shared ancestors (or lack thereof) between the leaf nodes and the root node into account.

TODO: This will need to be rewritten to take a list as input, rather than a dataframe. Otherwise the elements of the base noun group will get returned as arrays, rather than as singletons.

```
# bng is a base noun group
# ...technically, an object of class BioNLPST

# bng is a base noun group represented as a BioNLPST object
# returns: a string, or a factor? A factor, now.
getSemanticClassLabel <- function(bng) {
  FUNCTION.DEBUG <- FALSE
  if (FUNCTION.DEBUG) {
    print("In function getSemanticClassLabel().")
  } # close if-debug
  semanticClassLabel <- bng@label
  return(semanticClassLabel)
} # close function definition getSemanticClassLabel()

# returns: TRUE/FALSE
hasSemanticClassLabel <- function(bng) {
  # this is essentially a hack to attempt to force R to make a copy of the
object
  # TODO: see if I can take out this hack, now that I have fixed the previous
problem with bngs being a list versus a vector
    bng@filename <- filename

  if (is.na(getSemanticClassLabel(bng))) {
    return(FALSE)
  } else { return (TRUE) }
} # close function definition hasSemanticClassLabel()


# anaphor and candidate are base noun groups
# returns: TRUE/FALSE
semanticClassesMatch <- function(anaphor, candidate) {
  anaphorSemanticClass <- getSemanticClassLabel(anaphor)
  print(paste("anaphor semantic class:", anaphorSemanticClass))
  candidateSemanticClass <- getSemanticClassLabel(candidate)
  print(paste("candidate semantic class:", candidateSemanticClass))
```

```
  if (anaphorSemanticClass == candidateSemanticClass) {
    return(TRUE)
  } else {
    return(FALSE)
  }
} # close function definition semanticClassesMatch()

testSemanticMethods <- function() {
  print("NO SEMANTIC METHOD TESTS RUNNING YET.")
} # close function definition testSemanticMethods()

testSemanticMethods()

## [1] "NO SEMANTIC METHOD TESTS RUNNING YET."
```

Non-semantic functions on base noun groups

1. Getters for strings and for paragraph and sentence indices
2. Comparators for strings and for paragraph and sentence indices

TODO: as above, these functions need to be rewritten to take a list as the input argument, rather than a dataframe...

```
DEBUG <- FALSE

# TODO: change name of this function from the not-very-informative
getString()
# to the more informative getCoveredText(), or something similar
# bng is a base noun group
# returns: char
getString <- function(bng) {
  return(bng@covered.text)
} # close function definition

# args: bng is a base noun group/BioNLPST object
# return: integer (TODO: be sure to make these be integers!)
getSentenceIndex <- function(bng) {
  return(bng@sentence.index) # not sure this actually exists!
} # close function definition getSentenceIndex()

# args: bng is a base noun group
# return: integer (TODO: be sure to make these be integers!)
#getSentenceIndex <- function(bng) {
#  return(bng$sentenceIndex)
#} # close function definition getSentenceIndex()

getParagraphIndex <- function(bng) {
  return(bng@paragraph.index) # not sure this actually exists!
} # close function definition getParagraphIndex()
```

```
stringsMatch <- function(anaphor, candidate) {
  DEBUG.FUNCTION <- FALSE
  # TODO validate: both of these should be BioNLPST objects
  anaphor.covered.text <- anaphor@covered.text
  candidate.covered.text <- candidate@covered.text
  if (DEBUG.FUNCTION) {print(paste("In function stringsMatch(). Anaphor:",
anaphor.covered.text, "Candidate:", candidate.covered.text))}
  # is this the right comparator for strings??
  if (anaphor.covered.text == candidate.covered.text) {
    print("Match in function stringsMatch()!")
    return(TRUE)
  } else {
    if (DEBUG.FUNCTION) { print("No match in function stringsMatch()...") }
    return(FALSE)
  }
} # close function definition stringsMatch()

print("NO TESTS IMPLEMENETED FOR stringsMatch() FUNCTION.")

## [1] "NO TESTS IMPLEMENETED FOR stringsMatch() FUNCTION."

# note that you should only be calling this if you have already found that
they *don't* match *without* toggling the case.
# args: two base noun groups
# returns: T/F
stringsMatchIfCaseToggled <- function(anaphor, candidate) {

  DEBUG.FUNCTION <- FALSE
  # TODO validate: both of these should be BioNLPST objects
  if (DEBUG.FUNCTION) {
    print("In function stringsMatchIfCaseToggled()")
    print("Anaphor:")
    class(anaphor)
    mode(anaphor)
    str(anaphor)
    print("Candidate:")
    class(candidate)
    mode(candidate)
    str(candidate)
  }
  anaphor.lowercase <- tolower(anaphor@covered.text)
  candidate.lowercase <- tolower(candidate@covered.text)
  # is this the right comparator for strings??
  if (anaphor.lowercase == candidate.lowercase) {
    return(TRUE)
  } else {
    return(FALSE)
  }
} # close function definition stringsMatchIfCaseToggled()
```

```
# TODO: convert from list to BioNLPST object
# anaphor and candidate are base noun groups
# returns: TRUE/FALSE
# OOH, BIG BUG (that I have fixed): I have assumed that you ALREADY checked
that the PARAGRAPH indices match!!! So, I was just checking the sentence
indices. So, the function would return TRUE even if the sentences were in
different paragraphs. For example, the first sentence of paragraph 1 and the
first sentence of paragraph 2 would return TRUE. Issue here is that what I
*really* want to know is not whether or not they have the same sentence index
number, but whether or not they are in THE SAME SENTENCE... So, I have now
added a check of the paragraph indices into the sentence-indices-checking
function. All is now well.
sentenceIndicesMatch <- function(anaphor, candidate) {
  # if they're not in the same paragraph, then they're not in the same
sentence...
  if ( ! (paragraphIndicesMatch(anaphor, candidate)) ){
    return(FALSE)
  }
  # now that we've verified that they're in the same paragraph, we can go on
to check whether or not they're in the same sentence within that paragraph.
  if (anaphor$sentenceIndex == candidate$sentenceIndex) {
    return(TRUE)
  } else {
    return(FALSE)
  }
} # close function definition sentenceIndicesMatch()

# anaphor and candidate are base noun groups
# returns: TRUE/FALSE
paragraphIndicesMatch <- function(anaphor, candidate) {
  if (anaphor$paragraphIndex == candidate$paragraphIndex) {
    return(TRUE)
  } else {
    return(FALSE)
  }
} # close function definition paragraphIndicesMatch()

testNonSemanticRules <- function() {
  print("NON-SEMANTIC RULES NOT IMPLEMENTED.")
  print("BE SURE TO TEST FOR MISMATCH BETWEEN ZERO-BASED AND 1-BASED
INDICES.")
} # close function definition testNonSemanticRules()

testNonSemanticRules()
```

```
## [1] "NON-SEMANTIC RULES NOT IMPLEMENTED."
## [1] "BE SURE TO TEST FOR MISMATCH BETWEEN ZERO-BASED AND 1-BASED INDICES."
```

How to compare relative sentence positions?

What do you need to do for a Hobbs-like approach to coreference resolution? You need to know things like whether or not the candidate is at the beginning of the preceding sentence, end of the preceding sentence, further away than the preceding sentence… What would you have to do to represent that?

Let's suppose you wanted a function that returned true if the candidate is in the preceding sentence, and false otherwise… (We'll leave aside the "beginning" versus "end" issue for the moment.)

1. First and second sentences of the same paragraph should return TRUE.
2. Last sentence of one paragraph and first sentence of the next paragraph should return TRUE.

What would the sentence index and paragraph index relationships be in those two cases?

1. Paragraph indices are identical. Sentence index of anaphor is > sentence index of candidate.
2. Paragraph index of anaphor is 1 larger than paragraph index of candidate; sentence index of anaphor is 1 (or zero in a language other than R); sentence index of candidate equals the maximum value of sentence index for its paragraph.

```
# NB: not great from a programming point of view--there are two exit
points...
# @args: two base noun groups
# returns: boolean
inSameSentence <- function(anaphor, candidate) {
  if (paragraphIndicesMatch(anaphor = anaphor, candidate = candidate)) {
    if (sentenceIndicesMatch(anaphor = anaphor, candidate = candidate)) {
return(TRUE) } else { return(FALSE) }
  } else {
    return(FALSE)
  }
} # close function definition inSameSentence()

#inAdjacentSentences <- function(anaphor, candidate) {
#  if (paragraphIndicesMatch(anaphor = anaphor, candidate = candidate)) {
#    if (anaphor$sentenceIndex == (candidate$sentenceIndex + 1)) {
#      return(TRUE) # leaves out possibility of cataphoric reference
#    }
#  } elsif (anaphor$sentenceIndex == min(#oh, this is bad--I need the max
sentence index for a paragraph...)) {}
#} # close function definition inAdjacentSentences()

# given two sentences: is this one the immediate precedent of that one?

# TODO: convert from list to BioNLPST object
# given one sentence: is a given entity/markable at the left edge, at the
right edge, or neither?
```

```r
# args: sentence beginning/end, entity beginning/end
# returns: boolean TRUE if at beginning of sentence, FALSE if not
atLeftEdge <- function(sentence.offsets, entity.offsets) {
  sentence.start <- sentence.offsets[1]
  sentence.end <- sentence.offsets[2]
  entity.start <- entity.offsets[1]
  entity.end <- entity.offsets[2]
  if (sentence.start == entity.start) {
    return(TRUE)
  } else {
    return(FALSE)
  }
} # close function definition atLeftEdge()

# test
if (atLeftEdge(c(1, 100), c(1, 10))) {  print(".")} # should return TRUE
```

```
## [1] "."
```

```r
if ( ! atLeftEdge(c(1, 100), c(90, 100))) { print(".") } # should return
FALSE
```

```
## [1] "."
```

```r
# TODO: convert from list to BioNLPST object
# args:
# returns: boolean TRUE if entity at end of sentence, FALSE otherwise
atRightEdge <- function(sentence.offsets, entity.offsets) {
  sentence.start <- sentence.offsets[1]
  sentence.end <- sentence.offsets[2]
  entity.start <- entity.offsets[1]
  entity.end <- entity.offsets[2]
  if (sentence.end == entity.end) {
    return(TRUE)
  } else {
    return(FALSE)
  }
} # close function definition atRightEdge()

# test
if (! atRightEdge(c(1, 100), c(1, 10))) {  print(".")} # should return FALSE
```

```
## [1] "."
```

```r
if (atRightEdge(c(1, 100), c(90, 100))) { print(".") } # should return TRUE
```

```
## [1] "."
```

```r
# TODO: convert from list to BioNLPST object
# ASSUMPTION: TWO-CHARACTER DIFFERENCE BETWEEN SENTENCES!
# args: start/end offsets for two separate sentences.
```

```
# assumption: the one on the left really is on the left
# returns: boolean TRUE if the one on the left immediately precedes the one
on the right
# assumption/limitation: cataphoric reference is not covered
sentencesAdjacent <- function(left.sentence.offsets, right.sentence.offsets)
{
  left.sentence.start <- left.sentence.offsets[1]
  left.sentence.end <- left.sentence.offsets[2]
  right.sentence.start <- right.sentence.offsets[1]
  right.sentence.end <- right.sentence.offsets[2]
  if (left.sentence.end + 2 == right.sentence.start) {
    return(TRUE)
  } else {
    return(FALSE)
  }
} # close function definition sentencesAdjacent()

if (sentencesAdjacent(c(1, 100), c(102, 202))) { print(".") }

## [1] "."

if ( ! sentencesAdjacent(c(1, 100), c(200, 300))) { print(".") }

## [1] "."

# What would happen if I looped through the sentence offsets for an entire
file? Every pair should return TRUE--if not, I'll have found some special
cases that need to be taken care of...
```

Process a file

This part of the code reads in the contents of a file and stores all potential anaphora/candidates in a vector of base noun groups (loosely defined). Subsequent parts of the code will then operate on that vector.

Assumption: there will not be enough stuff in that vector to exceed available memory.

```
#DEBUG <- TRUE
DEBUG <- FALSE

# store the full set of annotations in this object...
bngs <- c() # bng = base noun group. We will make the (incorrect) assumption
# that every annotation is of a base noun group. Later we can filter for
character overlaps...
# can I make absolutely sure that bngs is and remains forever a vector, #
versus a list?

#covered.texts.anis <- c()

filename <- "/Users/kevincohen/Dropbox/N-Z/translator-relation-
extraction/code/testData/11319941.bionlp"
```

```
#lines <- readLines("/Users/kevincohen/Dropbox/N-Z/translator-relation-
extraction/code/testData/11319941.bionlp")
lines <- readLines(filename)
number.of.lines <- length(lines)

if (DEBUG) { print(paste("Number of lines read in:", number.of.lines)) }
if (DEBUG) { print("First 5 lines:") }
if (DEBUG) { print(lines[1:5]) }
if (DEBUG) { print(paste("")) }
#print("Last 5 lines:")
# clunky, but: should print the last 5 lines
#...but, I think it prints the whole fucking array ;-)
#print(lines[length(lines)-4:length(lines)])
#print("") # I just want a little space in the output
#before the next stuff starts happening...

# set this as you like. use it to cut down on how much output you get if
you're debugging and don't want/need to see hundreds of lines. For the
contents of an entire file, set the variable to the number of lines in the
file (length(lines)). To limit the amount of output that you have to wade
through, set it to a smaller number than that!
#number.of.lines.to.print <- 3
number.of.lines.to.print <- number.of.lines

if (DEBUG) { print("START ITERATING OVER LINES") }
  for (i in 1:number.of.lines.to.print) {
    line <- lines[i]
    if (DEBUG) { print(paste("Read in line", i, ":", line)) }
    new.bionlpst <- lineToBioNLPST(line)
    if (DEBUG) { print("RETURNED FROM lineToBioNLPST():") }
    if (DEBUG) { bioprint(new.bionlpst) }
    if (DEBUG) { print(paste("Type of object returned by new:",
typeof(new.bionlpst))) }
    if (DEBUG) { print(paste("...which is a", is(new.bionlpst))) }
    #print(new.bionlpst[[slots]])
    if (DEBUG) { print(str(new.bionlpst)) } # this is printing NULL--why?? #
especially
    # puzzling in light of the fact that the next line does print out
    # the covered text...
    if (DEBUG) { print(paste("Covered text:", new.bionlpst@covered.text)) }

    bngs <- c(bngs, new.bionlpst)
    if (DEBUG) { print("Just added this object to the list bngs:") }
    if (DEBUG) { print(new.bionlpst) }
    if (DEBUG) { print("Call some method with it:") }
    if (DEBUG) { hasSemanticClassLabel(new.bionlpst) }
    if (DEBUG) { print("Great, it didn't crash! Pass it and a copy of it to
some method:") }
```

```
    if (DEBUG) { getSemanticClassLabel(new.bionlpst) }
    if (DEBUG) { print("Great, it didn't crash when I called
hasSemanticClassLabel()!") }
    if (DEBUG) { copy.new.bionlpst <- new.bionlpst }
    if (DEBUG) { stringsMatchIfCaseToggled(new.bionlpst, copy.new.bionlpst) }
    if (DEBUG) { print("Great, it still hasn't crashed!") }
    if (DEBUG) { stringsMatch(new.bionlpst, copy.new.bionlpst) }


  } # close for-loop through file
```

```
## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion

## Warning in lineToBioNLPST(line): NAs introduced by coercion
```

```
## Warning in lineToBioNLPST(line): NAs introduced by coercion

if (DEBUG) { print("FINISHED ITERATING THROUGH FILE.")}
if (DEBUG) { print("")}

# now let's see what's in the vector bngs, since that's what's giving
# me trouble later...
if (DEBUG) {
  print(paste("Length of vector bngs:", length(bngs)))
  print(paste("First three elements of bngs:"))
  print(bngs[1:3])
} # close if-debug
```

Try the string match functions

```
DEBUG.CHUNK = FALSE

library(tidyverse)
# store IdentPairs here. Must be a tibble to take full advantage of it...
# ident.pairs <- as_tibble(filenames = c(),
#                          pair.ids = c(),
#                          anaphor.ids = c(),
#                          antecedent.ids = c(),
#                          sieves = c()) # wait, no--fuck... my
representation of IdentPairs is as a class!!! OK, let's rethink this...


ident.pairs.in.list <- c()

# At this point in the game, you have all of the annotations in the file
(which contains one per line) stored in bngs.
# The problem is, you mean for bngs to be a vector, but for some reason, it's
a fucking list:
# So: you must go through it as a list.


# override the value of this variable if you don't want to see the entire
# file contents
number.of.lines <- 3
number.of.lines <- length(bngs)

print(paste("LOOP THROUGH THE list, first", number.of.lines))

## [1] "LOOP THROUGH THE list, first 542"

for (i in 1:number.of.lines) {
  # what happens if you try this with the first one??
  # at that point, you don't have a preceding one to check against...
  if (1 == i) { next; }
  # variable bngs is supposed to be a vector of BioNLPST objects,
```

```
  # but I'm getting a complaint about type when I call the string-
  # matching function, so let's check:
  if (DEBUG.CHUNK) { print(paste("Type of current bngs element:",
is(bngs[[i]]))) }

  # Can't build the SeuratObject library--tripping over some C++ stuff, and
  # I don't want to go down that rabbit hole...
  #SeuratObject::ListToS4()
  anaphor <- bngs[[i]]
  candidate <- bngs[[i-1]]

  if (stringsMatch(anaphor = anaphor, candidate = candidate)) {
    print(paste("STRINGS MATCH!", getString(anaphor), getString(candidate)))
    if (DEBUG.CHUNK) {
      print("Anaphor:")
      bioprint(anaphor)
      print("Candidate:")
      bioprint(candidate)
    }
    # add pair to *something* here
    current.pair.id <- current.pair.id + 1 # need to increment the numeric
                                           # ID for pairs every time that you
                                           # match any sieve
                                           # ...so I should have a function
for
                                           # adding new pairs to wherever it
                                           # is that I'm storing them...
    current.pair <- new (Class = "IdentPair",
                         filename = filename,
                         pair.id = current.pair.id,
                         anaphor.id = getBioNlpStId(anaphor),
                         antecedent.id = getBioNlpStId(candidate),
                         sieve = "STRING.MATCH.EXACT")
    #identPrint(current.pair)
    printIdentPair(current.pair)
    ident.pairs.in.list <- c(ident.pairs.in.list, current.pair)

  } # close if-exact-string-match
  #if (stringsMatchIfCaseToggled(bngs[[i]], bngs[[i - 1]])) {
  if (stringsMatchIfCaseToggled(anaphor = anaphor, candidate = candidate)) {
    #print(paste("Strings match if case-toggled:", bngs[[i]]@covered.text),
bngs[[i-1]]@covered.text)
    print(paste("STRINGS MATCHED! (case-toggled)", getString(anaphor),
getString(candidate)))
    current.pair.id <- current.pair.id + 1 # need to increment the numeric
                                           # ID for pairs every time that you
                                           # match any sieve
                                           # ...so I should have a function
for
                                           # adding new pairs to wherever it
```

```
                                          # is that I'm storing them...
    current.pair <- new (Class = "IdentPair",
                         filename = filename,
                         pair.id = current.pair.id,
                         anaphor.id = getBioNlpStId(anaphor),
                         antecedent.id = getBioNlpStId(candidate),
                         sieve = "STRING.MATCH.CASE.TOGGLED")
    ident.pairs.in.list <- c(ident.pairs.in.list, current.pair)

    if (DEBUG.CHUNK) {
      print("Anaphor:")
      bioprint(anaphor)
      print("Candidate:")
      bioprint(candidate)
    }
  } # close if-case-toggled-string-match
} # close for-loop through file
```

```
## [1] "STRINGS MATCHED! (case-toggled) brain Brain"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! neuron neuron"
## [1] "2 ANA: T28 ANT: T27 "
## [1] "STRINGS MATCHED! (case-toggled) neuron neuron"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! neuron neuron"
## [1] "4 ANA: T29 ANT: T28 "
## [1] "STRINGS MATCHED! (case-toggled) neuron neuron"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! striatal striatal"
## [1] "6 ANA: T33 ANT: T32 "
## [1] "STRINGS MATCHED! (case-toggled) striatal striatal"
## [1] "STRINGS MATCHED! (case-toggled) Brain brain"
## [1] "STRINGS MATCHED! (case-toggled) brain Brain"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! mice mice"
## [1] "10 ANA: T123 ANT: T122 "
## [1] "STRINGS MATCHED! (case-toggled) mice mice"
## [1] "STRINGS MATCHED! (case-toggled) striatal Striatal"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! brain brain"
## [1] "13 ANA: T128 ANT: T127 "
## [1] "STRINGS MATCHED! (case-toggled) brain brain"
## [1] "STRINGS MATCHED! (case-toggled) Neuron neuron"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! neurons neurons"
## [1] "16 ANA: T145 ANT: T144 "
## [1] "STRINGS MATCHED! (case-toggled) neurons neurons"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! neurons neurons"
## [1] "18 ANA: T146 ANT: T145 "
```

```
## [1] "STRINGS MATCHED! (case-toggled) neurons neurons"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! neurons neurons"
## [1] "20 ANA: T147 ANT: T146 "
## [1] "STRINGS MATCHED! (case-toggled) neurons neurons"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! animals animals"
## [1] "22 ANA: T159 ANT: T158 "
## [1] "STRINGS MATCHED! (case-toggled) animals animals"
## [1] "STRINGS MATCHED! (case-toggled) Striatal striatal"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! striatal striatal"
## [1] "25 ANA: T173 ANT: T172 "
## [1] "STRINGS MATCHED! (case-toggled) striatal striatal"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! striatal striatal"
## [1] "27 ANA: T190 ANT: T189 "
## [1] "STRINGS MATCHED! (case-toggled) striatal striatal"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! allele allele"
## [1] "29 ANA: T214 ANT: T213 "
## [1] "STRINGS MATCHED! (case-toggled) allele allele"
## [1] "STRINGS MATCHED! (case-toggled) Striatal striatal"
## [1] "STRINGS MATCHED! (case-toggled) striatal Striatal"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! brain brain"
## [1] "33 ANA: T220 ANT: T219 "
## [1] "STRINGS MATCHED! (case-toggled) brain brain"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! striatal striatal"
## [1] "35 ANA: T229 ANT: T228 "
## [1] "STRINGS MATCHED! (case-toggled) striatal striatal"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! neurons neurons"
## [1] "37 ANA: T240 ANT: T239 "
## [1] "STRINGS MATCHED! (case-toggled) neurons neurons"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! neuron neuron"
## [1] "39 ANA: T252 ANT: T251 "
## [1] "STRINGS MATCHED! (case-toggled) neuron neuron"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! mice mice"
## [1] "41 ANA: T291 ANT: T290 "
## [1] "STRINGS MATCHED! (case-toggled) mice mice"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! neuron neuron"
## [1] "43 ANA: T305 ANT: T304 "
## [1] "STRINGS MATCHED! (case-toggled) neuron neuron"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! Vax1 Vax1"
```

```
## [1] "45 ANA: T403 ANT: T402 "
## [1] "STRINGS MATCHED! (case-toggled) Vax1 Vax1"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! brain brain"
## [1] "47 ANA: T430 ANT: T429 "
## [1] "STRINGS MATCHED! (case-toggled) brain brain"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! neuron neuron"
## [1] "49 ANA: T437 ANT: T436 "
## [1] "STRINGS MATCHED! (case-toggled) neuron neuron"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! brain brain"
## [1] "51 ANA: T455 ANT: T454 "
## [1] "STRINGS MATCHED! (case-toggled) brain brain"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! brain brain"
## [1] "53 ANA: T462 ANT: T461 "
## [1] "STRINGS MATCHED! (case-toggled) brain brain"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! brain brain"
## [1] "55 ANA: T465 ANT: T464 "
## [1] "STRINGS MATCHED! (case-toggled) brain brain"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! mice mice"
## [1] "57 ANA: T474 ANT: T473 "
## [1] "STRINGS MATCHED! (case-toggled) mice mice"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! mice mice"
## [1] "59 ANA: T475 ANT: T474 "
## [1] "STRINGS MATCHED! (case-toggled) mice mice"
## [1] "STRINGS MATCHED! (case-toggled) Brains brains"
## [1] "STRINGS MATCHED! (case-toggled) brain Brain"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! brain brain"
## [1] "63 ANA: T497 ANT: T496 "
## [1] "STRINGS MATCHED! (case-toggled) brain brain"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! brain brain"
## [1] "65 ANA: T498 ANT: T497 "
## [1] "STRINGS MATCHED! (case-toggled) brain brain"
## [1] "STRINGS MATCHED! (case-toggled) Brain brain"
## [1] "Match in function stringsMatch()!"
## [1] "STRINGS MATCH! brain brain"
## [1] "68 ANA: T540 ANT: T539 "
## [1] "STRINGS MATCHED! (case-toggled) brain brain"
```

```
  # whoops, I meant to do string-matching!

  print("FINISHED LOOPING THROUGH INPUT FILE.")
```

```
## [1] "FINISHED LOOPING THROUGH INPUT FILE."
```

```
  print(paste("Found", length(ident.pairs.in.list), "candidate pairs."))
```

```
## [1] "Found 69 candidate pairs."
```

Try the semantic functions

TODO: Add a function that just checks whether or not the annotations come from the same ontology. I could do this by looking at the beginning strings of the two bngs, but the cleaner way would probably be to make this an additional element of the class definition, and figure it out when I call new()...

TODO: When you find a match, output the full details of each of the two bngs with bioprint().

TODO: When you find a match, store it away by the two annotation IDs.

```
DEBUG.CHUNK = FALSE

# TODO: convert from list to BioNLPST object
# why am I doing this off of bngs, rather than passing in two objects? Oh, I
guess I am passing in two objects--but, doing so via bngs only makes sense if
I'm doing a whole file at a time. Actually, that sorta makes sense *if* I add
to each individual annotation/object a list of the other objects that have a
matched semantic class. Wouldn't it make more sense conceptually to do that
via a discourse model, though?

# here's a sort of test across an entire file...
print("")
```

```
## [1] ""
```

```
print("RUNNING semanticClassesMatch() on entire file...")
```

```
## [1] "RUNNING semanticClassesMatch() on entire file..."
```

```
print("")
```

```
## [1] ""
```

```
print(paste("Current value of pair ID:", current.pair.id))
```

```
## [1] "Current value of pair ID: 69"
```

```
# set to length(bngs) to process entire file,
# or to smaller number for devtesting
number.of.lines <- length(bngs)
#number.of.lines <- 3

if (TRUE) {
```

```
  # start at 2 because you can't do this for the first one alone
  for (i in 2:number.of.lines) {
    if (DEBUG) { print(paste("Index to bngs:", i)) }
    anaphor <- bngs[[i]]
    candidate <- bngs[[i-1]]

    if (DEBUG.CHUNK) {
      print("Anaphor:")
      bioprint(anaphor)
      print("Candidate:")
      bioprint(candidate)
    }
    if (semanticClassesMatch(anaphor, candidate)) {
      print(paste("<<<SEMANTIC MATCH!>>>", getSemanticClassLabel(anaphor),
getSemanticClassLabel(candidate)))
      if (DEBUG.CHUNK) {
        print("Anaphor:")
        #print(anaphor)
        bioprint(anaphor)
        print("Candidate:")
        #print(candidate)
        bioprint(candidate)
      }
      if (DEBUG.CHUNK) { print(paste("CURRENT PAIR ID TO TROUBLESHOOT BUG:",
current.pair.id)) }
      current.pair <- new (Class = "IdentPair",
                           filename = filename,
                           pair.id = current.pair.id,
                           anaphor.id = getBioNlpStId(anaphor),
                           antecedent.id = getBioNlpStId(candidate),
                           sieve = "SEMANTIC.LEAF.NODE.MATCH")
      if (TRUE) {
        print("New IdentPair:")
        printIdentPair(current.pair)
      }
      ident.pairs.in.list <- c(ident.pairs.in.list, current.pair)
    } # CLOSE IF-SEMANTIC-CLASSES-MATCH
  } # close for-loop through all annotations

  print("")
  print("FINISHED RUNNING SEMANTIC CHECKS ON FULL FILE")
  print("")
  # TODO: add a filter for noncontinuous annotations until such a time as I
can handle them!
} # close run across entire file

## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: UBERON:0002435"
```

```
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: GO:0050890"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: GO:0050890"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: UBERON:0010011"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0010011"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0000955 UBERON:0000955"
## [1] "New IdentPair:"
## [1] "69 ANA: T16 ANT: T15 "
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0000540"
## [1] "<<<SEMANTIC MATCH!>>> CL:0000540 CL:0000540"
## [1] "New IdentPair:"
## [1] "69 ANA: T21 ANT: T20 "
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: NCBITaxon:33208"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: NCBITaxon:33208"
```

```
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0000540"
## [1] "<<<SEMANTIC MATCH!>>> CL:0000540 CL:0000540"
## [1] "New IdentPair:"
## [1] "69 ANA: T28 ANT: T27 "
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0000540"
## [1] "<<<SEMANTIC MATCH!>>> CL:0000540 CL:0000540"
## [1] "New IdentPair:"
## [1] "69 ANA: T29 ANT: T28 "
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: SO:0000771"
## [1] "<<<SEMANTIC MATCH!>>> SO:0000771 SO:0000771"
## [1] "New IdentPair:"
## [1] "69 ANA: T31 ANT: T30 "
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0002435 UBERON:0002435"
## [1] "New IdentPair:"
## [1] "69 ANA: T33 ANT: T32 "
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: NCBITaxon:33208"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: NCBITaxon:33208"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: UBERON:0001062"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: UBERON:0005382"
## [1] "candidate semantic class: UBERON:0001062"
## [1] "anaphor semantic class: UBERON:0000125"
## [1] "candidate semantic class: UBERON:0005382"
## [1] "anaphor semantic class: UBERON:0002743"
## [1] "candidate semantic class: UBERON:0000125"
```

```
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: UBERON:0002743"
## [1] "anaphor semantic class: GO:0050890"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: GO:0050890"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: NCBITaxon:9606"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: GO:0042995"
## [1] "candidate semantic class: NCBITaxon:9606"
## [1] "anaphor semantic class: CL:1001474"
## [1] "candidate semantic class: GO:0042995"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:1001474"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: GO:0010467"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: GO:0098793"
## [1] "candidate semantic class: GO:0010467"
## [1] "anaphor semantic class: GO:0098794"
## [1] "candidate semantic class: GO:0098793"
## [1] "anaphor semantic class: CL:1001474"
## [1] "candidate semantic class: GO:0098794"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:1001474"
## [1] "anaphor semantic class: CL:0000099"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: GO:0050890"
## [1] "candidate semantic class: CL:0000099"
## [1] "anaphor semantic class: http://purl.obolibrary.org/obo/MONDO_0007739"
## [1] "candidate semantic class: GO:0050890"
## [1] "anaphor semantic class: CL:1001474"
## [1] "candidate semantic class:
http://purl.obolibrary.org/obo/MONDO_0007739"
## [1] "anaphor semantic class: http://purl.obolibrary.org/obo/MONDO_0005395"
## [1] "candidate semantic class: CL:1001474"
## [1] "anaphor semantic class: NCBITaxon:9606"
## [1] "candidate semantic class:
http://purl.obolibrary.org/obo/MONDO_0005395"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: NCBITaxon:9606"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: NCBITaxon:10088"
```

```
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: GO:0008283"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: GO:0008283"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: GO:0051867"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: GO:0051867"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: SO:0000704"
## [1] "<<<SEMANTIC MATCH!>>> SO:0000704 SO:0000704"
## [1] "New IdentPair:"
## [1] "69 ANA: T75 ANT: T74 "
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: GO:0050767"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: GO:0050767"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: SO:0000704"
## [1] "<<<SEMANTIC MATCH!>>> SO:0000704 SO:0000704"
## [1] "New IdentPair:"
## [1] "69 ANA: T82 ANT: T81 "
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: NCBITaxon:40674"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0001017"
## [1] "candidate semantic class: NCBITaxon:40674"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0001017"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: UBERON:0005382"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: UBERON:0005382"
```

```
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: SO:0000771"
## [1] "<<<SEMANTIC MATCH!>>> SO:0000771 SO:0000771"
## [1] "New IdentPair:"
## [1] "69 ANA: T91 ANT: T90 "
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: UBERON:0001893"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: GO:0021537"
## [1] "candidate semantic class: UBERON:0001893"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: GO:0021537"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: GO:0008283"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: GO:0008283"
## [1] "anaphor semantic class: UBERON:0001890"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: UBERON:0001890"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: UBERON:0000955"
```

```
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0000955 UBERON:0000955"
## [1] "New IdentPair:"
## [1] "69 ANA: T117 ANT: T116 "
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0000955 UBERON:0000955"
## [1] "New IdentPair:"
## [1] "69 ANA: T118 ANT: T117 "
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "<<<SEMANTIC MATCH!>>> NCBITaxon:10088 NCBITaxon:10088"
## [1] "New IdentPair:"
## [1] "69 ANA: T123 ANT: T122 "
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0002435 UBERON:0002435"
## [1] "New IdentPair:"
## [1] "69 ANA: T126 ANT: T125 "
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0000955 UBERON:0000955"
## [1] "New IdentPair:"
## [1] "69 ANA: T128 ANT: T127 "
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0000955"
```

```
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0000540"
## [1] "<<<SEMANTIC MATCH!>>> CL:0000540 CL:0000540"
## [1] "New IdentPair:"
## [1] "69 ANA: T136 ANT: T135 "
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0002435 UBERON:0002435"
## [1] "New IdentPair:"
## [1] "69 ANA: T138 ANT: T137 "
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0000540"
## [1] "<<<SEMANTIC MATCH!>>> CL:0000540 CL:0000540"
## [1] "New IdentPair:"
## [1] "69 ANA: T145 ANT: T144 "
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0000540"
## [1] "<<<SEMANTIC MATCH!>>> CL:0000540 CL:0000540"
## [1] "New IdentPair:"
## [1] "69 ANA: T146 ANT: T145 "
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0000540"
## [1] "<<<SEMANTIC MATCH!>>> CL:0000540 CL:0000540"
## [1] "New IdentPair:"
## [1] "69 ANA: T147 ANT: T146 "
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: UBERON:0002435"
```

```
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: NCBITaxon:33208"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: NCBITaxon:33208"
## [1] "candidate semantic class: NCBITaxon:33208"
## [1] "<<<SEMANTIC MATCH!>>> NCBITaxon:33208 NCBITaxon:33208"
## [1] "New IdentPair:"
## [1] "69 ANA: T159 ANT: T158 "
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: NCBITaxon:33208"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0002435 UBERON:0002435"
## [1] "New IdentPair:"
## [1] "69 ANA: T162 ANT: T161 "
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: NCBITaxon:33208"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: UBERON:0000955"
```

```
## [1] "candidate semantic class: NCBITaxon:33208"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0002435 UBERON:0002435"
## [1] "New IdentPair:"
## [1] "69 ANA: T173 ANT: T172 "
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0002435 UBERON:0002435"
## [1] "New IdentPair:"
## [1] "69 ANA: T190 ANT: T189 "
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: SO:0000771"
```

```
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0001023"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: SO:0001023"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: SO:0001026"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: SO:0001026"
## [1] "anaphor semantic class: SO:0001026"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: SO:0001026"
## [1] "anaphor semantic class: SO:0001023"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: SO:0001023"
## [1] "anaphor semantic class: SO:0001023"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0001023"
## [1] "candidate semantic class: SO:0001023"
## [1] "<<<SEMANTIC MATCH!>>> SO:0001023 SO:0001023"
## [1] "New IdentPair:"
## [1] "69 ANA: T213 ANT: T212 "
## [1] "anaphor semantic class: SO:0001023"
## [1] "candidate semantic class: SO:0001023"
## [1] "<<<SEMANTIC MATCH!>>> SO:0001023 SO:0001023"
## [1] "New IdentPair:"
## [1] "69 ANA: T214 ANT: T213 "
## [1] "anaphor semantic class: UBERON:0002435"
```

```
## [1] "candidate semantic class: SO:0001023"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0002435 UBERON:0002435"
## [1] "New IdentPair:"
## [1] "69 ANA: T216 ANT: T215 "
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0002435 UBERON:0002435"
## [1] "New IdentPair:"
## [1] "69 ANA: T217 ANT: T216 "
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0000955 UBERON:0000955"
## [1] "New IdentPair:"
## [1] "69 ANA: T220 ANT: T219 "
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: SO:0001023"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: SO:0001023"
## [1] "anaphor semantic class: UBERON:0001017"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0001017"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0002435 UBERON:0002435"
## [1] "New IdentPair:"
## [1] "69 ANA: T229 ANT: T228 "
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0001023"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: SO:0001023"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: CL:0002613"
```

```
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0000540"
## [1] "<<<SEMANTIC MATCH!>>> CL:0000540 CL:0000540"
## [1] "New IdentPair:"
## [1] "69 ANA: T236 ANT: T235 "
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: SO:0001023"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: SO:0001023"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0000540"
## [1] "<<<SEMANTIC MATCH!>>> CL:0000540 CL:0000540"
## [1] "New IdentPair:"
## [1] "69 ANA: T240 ANT: T239 "
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0001026"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: SO:0001026"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0001023"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: SO:0001023"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0000540"
## [1] "<<<SEMANTIC MATCH!>>> CL:0000540 CL:0000540"
## [1] "New IdentPair:"
## [1] "69 ANA: T252 ANT: T251 "
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
```

```
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0005382"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0005382"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: UBERON:0001017"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0001017"
## [1] "anaphor semantic class: UBERON:0001017"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0001017"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: CHEBI:15355"
## [1] "candidate semantic class: CL:0002613"
```

```
## [1] "anaphor semantic class: CL:0000108"
## [1] "candidate semantic class: CHEBI:15355"
## [1] "anaphor semantic class: CHEBI:5613"
## [1] "candidate semantic class: CL:0000108"
## [1] "anaphor semantic class: CL:0000120"
## [1] "candidate semantic class: CHEBI:5613"
## [1] "anaphor semantic class: UBERON:0005381"
## [1] "candidate semantic class: CL:0000120"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0005381"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: CL:0000598"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: UBERON:0002313"
## [1] "candidate semantic class: CL:0000598"
## [1] "anaphor semantic class: UBERON:0002304"
## [1] "candidate semantic class: UBERON:0002313"
## [1] "anaphor semantic class: CL:0000120"
## [1] "candidate semantic class: UBERON:0002304"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: CL:0000120"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "<<<SEMANTIC MATCH!>>> NCBITaxon:10088 NCBITaxon:10088"
## [1] "New IdentPair:"
## [1] "69 ANA: T291 ANT: T290 "
## [1] "anaphor semantic class: UBERON:0001016"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: UBERON:0009050"
## [1] "candidate semantic class: UBERON:0001016"
## [1] "anaphor semantic class: UBERON:0018545"
## [1] "candidate semantic class: UBERON:0009050"
## [1] "anaphor semantic class: UBERON:0001792"
## [1] "candidate semantic class: UBERON:0018545"
## [1] "anaphor semantic class: CL:0000740"
## [1] "candidate semantic class: UBERON:0001792"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0000740"
## [1] "anaphor semantic class: UBERON:0001017"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0001017"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: CL:0002613"
```

```
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0000540"
## [1] "<<<SEMANTIC MATCH!>>> CL:0000540 CL:0000540"
## [1] "New IdentPair:"
## [1] "69 ANA: T305 ANT: T304 "
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0000540"
## [1] "<<<SEMANTIC MATCH!>>> CL:0000540 CL:0000540"
## [1] "New IdentPair:"
## [1] "69 ANA: T306 ANT: T305 "
## [1] "anaphor semantic class: UBERON:0002304"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0002304"
## [1] "anaphor semantic class: CL:0000598"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: UBERON:0000119"
## [1] "candidate semantic class: CL:0000598"
## [1] "anaphor semantic class: CL:0000120"
## [1] "candidate semantic class: UBERON:0000119"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: CL:0000120"
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0000540"
## [1] "<<<SEMANTIC MATCH!>>> CL:0000540 CL:0000540"
## [1] "New IdentPair:"
## [1] "69 ANA: T315 ANT: T314 "
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: UBERON:0001017"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0001017"
## [1] "anaphor semantic class: CL:1001474"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: CL:1001474"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0000540"
## [1] "<<<SEMANTIC MATCH!>>> CL:0000540 CL:0000540"
## [1] "New IdentPair:"
```

```
## [1] "69 ANA: T322 ANT: T321 "
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0001017"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: UBERON:0001017"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: SO:0000704"
## [1] "<<<SEMANTIC MATCH!>>> SO:0000704 SO:0000704"
## [1] "New IdentPair:"
## [1] "69 ANA: T333 ANT: T332 "
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: SO:0001026"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: SO:0001026"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: NCBITaxon:10088"
```

```
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: SO:0001023"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: SO:0001023"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: NCBITaxon:9606"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: NCBITaxon:9606"
## [1] "anaphor semantic class: GO:0010467"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: GO:0010467"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: SO:0000771"
## [1] "<<<SEMANTIC MATCH!>>> SO:0000771 SO:0000771"
## [1] "New IdentPair:"
## [1] "69 ANA: T359 ANT: T358 "
## [1] "anaphor semantic class: SO:0000018"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: NCBITaxon:33208"
## [1] "candidate semantic class: SO:0000018"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: NCBITaxon:33208"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: GO:0007420"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: PR:000008241"
## [1] "candidate semantic class: GO:0007420"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: PR:000008241"
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: http://purl.obolibrary.org/obo/MONDO_0007739"
```

```
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class:
http://purl.obolibrary.org/obo/MONDO_0007739"
## [1] "anaphor semantic class: GO:0045202"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: GO:0045202"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: PR:000010184"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: PR:000010184"
## [1] "anaphor semantic class: MOP:0000124"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: PR:000010184"
## [1] "candidate semantic class: MOP:0000124"
## [1] "anaphor semantic class: PR:000010184"
## [1] "candidate semantic class: PR:000010184"
## [1] "<<<SEMANTIC MATCH!>>> PR:000010184 PR:000010184"
## [1] "New IdentPair:"
## [1] "69 ANA: T378 ANT: T377 "
## [1] "anaphor semantic class: CHEBI:36357"
## [1] "candidate semantic class: PR:000010184"
## [1] "anaphor semantic class: UBERON:0001893"
## [1] "candidate semantic class: CHEBI:36357"
## [1] "anaphor semantic class: GO:0021537"
## [1] "candidate semantic class: UBERON:0001893"
## [1] "anaphor semantic class: PR:000010184"
## [1] "candidate semantic class: GO:0021537"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: PR:000010184"
## [1] "anaphor semantic class: http://purl.obolibrary.org/obo/MONDO_0009022"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: UBERON:0002336"
## [1] "candidate semantic class:
http://purl.obolibrary.org/obo/MONDO_0009022"
## [1] "anaphor semantic class: UBERON:0001890"
## [1] "candidate semantic class: UBERON:0002336"
## [1] "anaphor semantic class: UBERON:0001950"
## [1] "candidate semantic class: UBERON:0001890"
## [1] "anaphor semantic class: PR:000010185"
## [1] "candidate semantic class: UBERON:0001950"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: PR:000010185"
## [1] "anaphor semantic class: GO:0010467"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: GO:0010467"
```

```
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: GO:0007420"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: NCBITaxon:10114"
## [1] "candidate semantic class: GO:0007420"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: NCBITaxon:10114"
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: UBERON:0001893"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: GO:0021537"
## [1] "candidate semantic class: UBERON:0001893"
## [1] "anaphor semantic class: PR:000017268"
## [1] "candidate semantic class: GO:0021537"
## [1] "anaphor semantic class: PR:000017268"
## [1] "candidate semantic class: PR:000017268"
## [1] "<<<SEMANTIC MATCH!>>> PR:000017268 PR:000017268"
## [1] "New IdentPair:"
## [1] "69 ANA: T403 ANT: T402 "
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: PR:000017268"
## [1] "anaphor semantic class: PR:000011336"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: PR:000011336"
## [1] "anaphor semantic class: PR:000017268"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: UBERON:0001890"
## [1] "candidate semantic class: PR:000017268"
## [1] "anaphor semantic class: GO:0010467"
## [1] "candidate semantic class: UBERON:0001890"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: GO:0010467"
## [1] "anaphor semantic class: GO:0009790"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: CHEBI:36357"
## [1] "candidate semantic class: GO:0009790"
## [1] "anaphor semantic class: GO:0030424"
## [1] "candidate semantic class: CHEBI:36357"
## [1] "anaphor semantic class: GO:0007411"
## [1] "candidate semantic class: GO:0030424"
## [1] "anaphor semantic class: UBERON:0002336"
```

```
## [1] "candidate semantic class: GO:0007411"
## [1] "anaphor semantic class: UBERON:0000959"
## [1] "candidate semantic class: UBERON:0002336"
## [1] "anaphor semantic class: PR:000017268"
## [1] "candidate semantic class: UBERON:0000959"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: PR:000017268"
## [1] "anaphor semantic class: PR:000017268"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: CHEBI:36357"
## [1] "candidate semantic class: PR:000017268"
## [1] "anaphor semantic class: PR:000014841"
## [1] "candidate semantic class: CHEBI:36357"
## [1] "anaphor semantic class: PR:000012315"
## [1] "candidate semantic class: PR:000014841"
## [1] "anaphor semantic class: PR:000012318"
## [1] "candidate semantic class: PR:000012315"
## [1] "anaphor semantic class: PR:000013771"
## [1] "candidate semantic class: PR:000012318"
## [1] "anaphor semantic class: GO:0030900"
## [1] "candidate semantic class: PR:000013771"
## [1] "anaphor semantic class: UBERON:0002743"
## [1] "candidate semantic class: GO:0030900"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002743"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0000955 UBERON:0000955"
## [1] "New IdentPair:"
## [1] "69 ANA: T430 ANT: T429 "
## [1] "anaphor semantic class: UBERON:0001017"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0001017"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: NCBITaxon:9606"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: NCBITaxon:9606"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0000540"
## [1] "<<<SEMANTIC MATCH!>>> CL:0000540 CL:0000540"
## [1] "New IdentPair:"
```

```
## [1] "69 ANA: T437 ANT: T436 "
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0000125"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000125"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0002435 UBERON:0002435"
## [1] "New IdentPair:"
## [1] "69 ANA: T444 ANT: T443 "
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0000704"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: SO:0000704"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: UBERON:0001893"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0001893"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0000955 UBERON:0000955"
## [1] "New IdentPair:"
## [1] "69 ANA: T455 ANT: T454 "
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: GO:0065007"
```

```
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: UBERON:0001017"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0001017"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0000955 UBERON:0000955"
## [1] "New IdentPair:"
## [1] "69 ANA: T462 ANT: T461 "
## [1] "anaphor semantic class: NCBITaxon:33208"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: NCBITaxon:33208"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0000955 UBERON:0000955"
## [1] "New IdentPair:"
## [1] "69 ANA: T465 ANT: T464 "
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: GO:0065007"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: GO:0065007"
## [1] "anaphor semantic class: UBERON:0002037"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0002037"
## [1] "anaphor semantic class: SO:0001026"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: SO:0001023"
## [1] "candidate semantic class: SO:0001026"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: SO:0001023"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "<<<SEMANTIC MATCH!>>> NCBITaxon:10088 NCBITaxon:10088"
## [1] "New IdentPair:"
## [1] "69 ANA: T474 ANT: T473 "
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "<<<SEMANTIC MATCH!>>> NCBITaxon:10088 NCBITaxon:10088"
## [1] "New IdentPair:"
## [1] "69 ANA: T475 ANT: T474 "
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0000955"
```

```
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: CHEBI:30879"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: CHEBI:15377"
## [1] "candidate semantic class: CHEBI:30879"
## [1] "anaphor semantic class: UBERON:0002084"
## [1] "candidate semantic class: CHEBI:15377"
## [1] "anaphor semantic class: CHEBI:37586"
## [1] "candidate semantic class: UBERON:0002084"
## [1] "anaphor semantic class: CHEBI:64276"
## [1] "candidate semantic class: CHEBI:37586"
## [1] "anaphor semantic class: CHEBI:50913"
## [1] "candidate semantic class: CHEBI:64276"
## [1] "anaphor semantic class: CHEBI:64276"
## [1] "candidate semantic class: CHEBI:50913"
## [1] "anaphor semantic class: UBERON:0000033"
## [1] "candidate semantic class: CHEBI:64276"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0000033"
## [1] "anaphor semantic class: CHEBI:50913"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: CHEBI:50913"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0000955 UBERON:0000955"
## [1] "New IdentPair:"
## [1] "69 ANA: T491 ANT: T490 "
## [1] "anaphor semantic class: CHEBI:16842"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: CHEBI:16842"
## [1] "anaphor semantic class: CHEBI:52815"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: CHEBI:52815"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0000955 UBERON:0000955"
## [1] "New IdentPair:"
## [1] "69 ANA: T496 ANT: T495 "
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0000955 UBERON:0000955"
## [1] "New IdentPair:"
## [1] "69 ANA: T497 ANT: T496 "
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0000955"
```

```
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0000955 UBERON:0000955"
## [1] "New IdentPair:"
## [1] "69 ANA: T498 ANT: T497 "
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0000955 UBERON:0000955"
## [1] "New IdentPair:"
## [1] "69 ANA: T499 ANT: T498 "
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: NCBITaxon:10088"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0002435 UBERON:0002435"
## [1] "New IdentPair:"
## [1] "69 ANA: T505 ANT: T504 "
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0002435 UBERON:0002435"
## [1] "New IdentPair:"
## [1] "69 ANA: T506 ANT: T505 "
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: CL:0002613"
## [1] "candidate semantic class: CL:0002613"
## [1] "<<<SEMANTIC MATCH!>>> CL:0002613 CL:0002613"
## [1] "New IdentPair:"
## [1] "69 ANA: T508 ANT: T507 "
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0002613"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: GO:0005730"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: GO:0005730"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: CL:0000540"
```

```
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0000125"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000125"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: CL:0000540"
## [1] "<<<SEMANTIC MATCH!>>> CL:0000540 CL:0000540"
## [1] "New IdentPair:"
## [1] "69 ANA: T522 ANT: T521 "
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: CL:0000540"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: CL:0000540"
## [1] "anaphor semantic class: SO:0001026"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: UBERON:0002106"
## [1] "candidate semantic class: SO:0001026"
## [1] "anaphor semantic class: NCBITaxon:33208"
## [1] "candidate semantic class: UBERON:0002106"
## [1] "anaphor semantic class: CHEBI:24866"
## [1] "candidate semantic class: NCBITaxon:33208"
## [1] "anaphor semantic class: GO:0030849"
## [1] "candidate semantic class: CHEBI:24866"
## [1] "anaphor semantic class: GO:0000805"
## [1] "candidate semantic class: GO:0030849"
## [1] "anaphor semantic class: NCBITaxon:33208"
## [1] "candidate semantic class: GO:0000805"
## [1] "anaphor semantic class: SO:0001026"
## [1] "candidate semantic class: NCBITaxon:33208"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: SO:0001026"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
## [1] "anaphor semantic class: SO:0000771"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: UBERON:0002435"
## [1] "candidate semantic class: SO:0000771"
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0002435"
```

```
## [1] "anaphor semantic class: UBERON:0000955"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "<<<SEMANTIC MATCH!>>> UBERON:0000955 UBERON:0000955"
## [1] "New IdentPair:"
## [1] "69 ANA: T540 ANT: T539 "
## [1] "anaphor semantic class: UBERON:0001062"
## [1] "candidate semantic class: UBERON:0000955"
## [1] "anaphor semantic class: NCBITaxon:10088"
## [1] "candidate semantic class: UBERON:0001062"
## [1] ""
## [1] "FINISHED RUNNING SEMANTIC CHECKS ON FULL FILE"
## [1] ""
```

Produce output from the semantic rules

```
print("IDENT PAIRS FROM ALL RULES:")
```

```
## [1] "IDENT PAIRS FROM ALL RULES:"
```

```
# Just printing the list sucks--way too much empty space! Let's iterate over
# the list and call the pretty-print function for each pair; if I don't have
# one yet, let's write one.
#print(ident.pairs.in.list)

for (i in 1:length(ident.pairs.in.list)) {
  #identPrint(ident.pair = ident.pairs.in.list[[i]])
  printIdentPair(ident.pair = ident.pairs.in.list[[i]])
} # close for-loop through list of IdentPairs
```

```
## [1] "1 ANA: T542 ANT: T541 "
## [1] "2 ANA: T542 ANT: T541 "
## [1] "3 ANA: T542 ANT: T541 "
## [1] "4 ANA: T542 ANT: T541 "
## [1] "5 ANA: T542 ANT: T541 "
## [1] "6 ANA: T542 ANT: T541 "
## [1] "7 ANA: T542 ANT: T541 "
## [1] "8 ANA: T542 ANT: T541 "
## [1] "9 ANA: T542 ANT: T541 "
## [1] "10 ANA: T542 ANT: T541 "
## [1] "11 ANA: T542 ANT: T541 "
## [1] "12 ANA: T542 ANT: T541 "
## [1] "13 ANA: T542 ANT: T541 "
## [1] "14 ANA: T542 ANT: T541 "
## [1] "15 ANA: T542 ANT: T541 "
## [1] "16 ANA: T542 ANT: T541 "
## [1] "17 ANA: T542 ANT: T541 "
## [1] "18 ANA: T542 ANT: T541 "
## [1] "19 ANA: T542 ANT: T541 "
## [1] "20 ANA: T542 ANT: T541 "
## [1] "21 ANA: T542 ANT: T541 "
## [1] "22 ANA: T542 ANT: T541 "
## [1] "23 ANA: T542 ANT: T541 "
## [1] "24 ANA: T542 ANT: T541 "
## [1] "25 ANA: T542 ANT: T541 "
## [1] "26 ANA: T542 ANT: T541 "
## [1] "27 ANA: T542 ANT: T541 "
## [1] "28 ANA: T542 ANT: T541 "
## [1] "29 ANA: T542 ANT: T541 "
## [1] "30 ANA: T542 ANT: T541 "
## [1] "31 ANA: T542 ANT: T541 "
## [1] "32 ANA: T542 ANT: T541 "
## [1] "33 ANA: T542 ANT: T541 "
## [1] "34 ANA: T542 ANT: T541 "
## [1] "35 ANA: T542 ANT: T541 "
```

```
## [1] "36 ANA: T542 ANT: T541 "
## [1] "37 ANA: T542 ANT: T541 "
## [1] "38 ANA: T542 ANT: T541 "
## [1] "39 ANA: T542 ANT: T541 "
## [1] "40 ANA: T542 ANT: T541 "
## [1] "41 ANA: T542 ANT: T541 "
## [1] "42 ANA: T542 ANT: T541 "
## [1] "43 ANA: T542 ANT: T541 "
## [1] "44 ANA: T542 ANT: T541 "
## [1] "45 ANA: T542 ANT: T541 "
## [1] "46 ANA: T542 ANT: T541 "
## [1] "47 ANA: T542 ANT: T541 "
## [1] "48 ANA: T542 ANT: T541 "
## [1] "49 ANA: T542 ANT: T541 "
## [1] "50 ANA: T542 ANT: T541 "
## [1] "51 ANA: T542 ANT: T541 "
## [1] "52 ANA: T542 ANT: T541 "
## [1] "53 ANA: T542 ANT: T541 "
## [1] "54 ANA: T542 ANT: T541 "
## [1] "55 ANA: T542 ANT: T541 "
## [1] "56 ANA: T542 ANT: T541 "
## [1] "57 ANA: T542 ANT: T541 "
## [1] "58 ANA: T542 ANT: T541 "
## [1] "59 ANA: T542 ANT: T541 "
## [1] "60 ANA: T542 ANT: T541 "
## [1] "61 ANA: T542 ANT: T541 "
## [1] "62 ANA: T542 ANT: T541 "
## [1] "63 ANA: T542 ANT: T541 "
## [1] "64 ANA: T542 ANT: T541 "
## [1] "65 ANA: T542 ANT: T541 "
## [1] "66 ANA: T542 ANT: T541 "
## [1] "67 ANA: T542 ANT: T541 "
## [1] "68 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
```

```
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
## [1] "69 ANA: T542 ANT: T541 "
```

```r
print(paste("Found", length(ident.pairs.in.list), "candidate pairs."))
```

```
## [1] "Found 128 candidate pairs."
```

Reproducibility/Repeatability

TODO: after calling this, give the option of printing out the entire file contents for debugging/traceability purposes.

```
sessionInfo()
```

```
## R version 4.0.1 (2020-06-06)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS  10.16
##
## Matrix products: default
## BLAS:
/Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK:
/Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] forcats_0.5.1   stringr_1.4.0   dplyr_1.0.8     purrr_0.3.4
## [5] readr_2.1.2     tidyr_1.2.0     tibble_3.1.6    ggplot2_3.3.5
## [9] tidyverse_1.3.1
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.1.1 xfun_0.29        haven_2.4.3      colorspace_2.0-2
##  [5] vctrs_0.3.8      generics_0.1.2   htmltools_0.5.2  yaml_2.2.2
##  [9] utf8_1.2.2       rlang_1.0.1      pillar_1.7.0     withr_2.4.3
## [13] glue_1.6.1       DBI_1.1.2        dbplyr_2.1.1     modelr_0.1.8
## [17] readxl_1.3.1     lifecycle_1.0.1  munsell_0.5.0    gtable_0.3.0
## [21] cellranger_1.1.0 rvest_1.0.2      evaluate_0.14    knitr_1.37
## [25] tzdb_0.2.0       fastmap_1.1.0    fansi_1.0.2      broom_0.7.12
## [29] Rcpp_1.0.8       backports_1.4.1  scales_1.1.1     jsonlite_1.7.3
## [33] fs_1.5.2         hms_1.1.1        digest_0.6.29    stringi_1.7.6
## [37] grid_4.0.1       cli_3.2.0        tools_4.0.1      magrittr_2.0.2
## [41] crayon_1.5.0     pkgconfig_2.0.3  ellipsis_0.3.2   xml2_1.3.3
## [45] reprex_2.0.1     lubridate_1.8.0  assertthat_0.2.1 rmarkdown_2.11
## [49] httr_1.4.2       rstudioapi_0.13  R6_2.5.1         compiler_4.0.1
```