Issues in Anaphora Resolution

Imran Q. Sayed
iqsayed@stanford.edu

I. Background

Anaphora resolution (AR) which most commonly appears as pronoun resolution is the problem of resolving references to earlier or later items in the discourse. These items are usually noun phrases representing objects in the real world called referents but can also be verb phrases, whole sentences or paragraphs. AR is classically recognized as a very difficult problem in NLP (see [Mitkov99], [Denber98]). A very brief introduction to the problem is given here; the reader is referred to [Mitkov99] for a more detailed survey on problems and techniques.

There are primarily three types of anaphora:

- Pronominal: This is the most common type where a referent is referred by a pronoun.
Example: "John found the love of his life" where 'his' refers to 'John'.

- Definite noun phrase: The antecedent is referred by a phrase of the form "<the> <noun phrase>".
Continued example: "The relationship did not last long", where 'The relationship' refers to 'the love' in the preceding sentence.

- Quantifier/Ordinal: The anphor is a quantifier such as 'one' or an ordinal such as 'first'.
Continued Example: "He started a new one" where 'one' refers to 'The relationship' (effectively meaning 'a relationship').

The traditional resolution techniques include:

- Eliminative Constraints: An anaphor and a referent must agree in certain attributes to generate a match. These include gender (male/female/neutral), number (singular/plural), and semantic consistency (e.g. ‘a disk’ is ‘copied’ and ‘a computer’ ‘disconnected’, not vice versa).

- Weighting Preferences: These factors are used to assign likelihood of match to the competing referents. They include proximity (of an antecedent phrase to the anaphor in the text), centering (which determines the center of attention object) and syntactic/semantic (role) parallelism. As examples of the latter, consider the form "<noun-1> <verb> <noun-2> however <noun-3> <verb> <anaphor>" where <anaphor> is likely to match with <noun-2>.

There are important applications of anphora resolution in information extraction such as "comprehending" a discourse in order to summarize it or answer questions from it.

II. Objective

The aim of this project is to explore the following issues or types of anaphora resolution that the current literature on the topic seems to have paid not much attention to:

- Recognizing pleonastic anaphora 'it' and miscategorization of 'it' as pleonastic when there infact is a referent
- Multi-sentential anaphora resolution, specially in the context of very large texts, through a process of 'chaining' and keeping a history list
- Non-noun-phrase referents; in particular, verb-phrases as referents
- Extracting and using semantic information about attributes; this information can be used in other IE tasks in general

III. Formal definition of Anaphora

A grammatical definition of anaphora is not common in literature on the problem. One is proposed here, which is for anaphora and cataphora combined, call them 'phora' if you will:

> Head of a Noun Phrase that is not a noun and is one of the words in Appendix A (only a partial list), or a noun phrase of the form 'the <ADJ>* <NN|NNS>'[1].

The definition can be further tested and refined.

IV. Design of Techniques

This section describes techniques that can be used to solve the problems alluded to above.

Multi-Sentential Resolution

A history list technique is proposed here[2]:

> A list of referents[3] is maintained as new referents are encountered in the sequential parsing of the input text. The number of referents can grow very large and prohibit efficient and sensible search for resolution of ambiguous anaphora. So the list is to be periodically trimmed every N words/M[4] sentences. Only those referents are discarded which are not re-referred for this long a stretch; the re-reference can either be through an anaphor or a detailed phrase (possibly a re-introduction of the referent). To facilitate this, a variable is kept to track the most recent reference to an antecedent entity.

As anaphora are resolved during parsing, they contribute to this book-keeping just as noun phrases do. This phenomenon of anaphora referring back to antecedent anaphora can be called 'anaphora chaining'.

[1] Determiner 'the' indicates a concrete object that has either been mentioned elsewhere in the discourse or is implied; the implied case is called associative anaphora (see [MeyerDale02])

[2] Some authors describe more complicated techniques such as [StrubHahn95] which uses multiple lists to keep the search scope limited

[3] See Appendix D for a programming language representation of a referent

[4] Say 20 sentences since there is evidence suggesting that antecendents can be upto nearly this far from the anaphora referring them; in particular, [Mitkov99] gives the figure of 17

Attributes as Semantic clues

Consider the following example:

There were dresses of several different colors and styles.
They were all pretty, labeled with price tags.
Sally chose a blue one. Mary chose a skimpy one.

Did they choose 'dresses' or 'price tags'? Since both dresses and price tags in general (like any other physical object) have colors, both answers would be equally right without knowing further about the real world (such as women's need and desire for pretty clothes compared to price tags). However, the context describes color and style as attributes of dresses, and not tags. That should in principle increase the likelihood of 'blue' and 'skimpy' being applied to dresses rather than tags. The proposition then is to determine and record what attributes a certain referent has; then check if one of those attributes is applied as an adjective to the pronoun ('one' here), i.e. the adjective is a value of the attribute (‘blue’ a value of ‘color’).

When there are no competing alternatives ('price tags' above), then it is relatively straight forward to resolve the anaphora ('one' here) even in the absence of attribute information:

They stumbled upon several pretty dresses.
Sally chose a blue one. Mary chose a skimpy one.

When other alternatives are present but attribute information is not, then the resolution has to take higher level world knowledge into account which is beyond the scope here. (That is, information which is at a much higher semantic level than extracting the attributes of nouns/objects or verbs/actions from the text.)

Misclassifying ‘it’ as Pleonastic

The term “pleonastic” refers to an occurrence of ‘it’ that does not refer to any entity and usually occurs in a sentence about state. [Denber98] calls them ‘state references’ and classifies them into ‘meteorological’ and ‘temporal’ pleonasms. An example of the former is “It is raining” and that of the latter is “It’s three o’clock”. [Denber98] however does not recognize other state reference forms that don’t classify as either; consider “It is <pleasant | fine | okay | refreshing | tense | a tense situation>” which need not have referents in the discourse.

Both [Denber98] and [Mitkov99] classify ‘it’ as pleonastic in sentences like ‘it seems that …’, ‘it should be obvious that …’, ‘it is important to note that …’, etc. However, strictly speaking, these occurrences of ‘it’ are not pleonastic as they refer to facts usually followed by ‘that’ in the sentence. Apart from theoretical correctness, this distinction is important in order to answer questions such as “What is unlikely?” given the facts “It is unlikely that the algorithm resolve all the anaphora”, “It is unlikely that Australia win the 7th ODI”, etc. It is therefore desirable to consider the phrase following ‘that’ as referent in these cases.

It can also be noted that the statements of the form “It <is | would be | will be> < fine | acceptable | a disaster > if <NP> <VP>” and similar expressions are instances of pleonastic ‘it’ but are not recognized as such in the literature.

Consider the example:

Devising an ideal social and political system is next to impossible.
It would be wonderful if we could do that.

It is clear that ‘It’ here is not referring to ‘Devising’ or ‘system’ but actually is a state reference.

Verb Phrases as Referents

A class of definite noun phrase anaphora was considered where the noun is a so called action-noun and refers to a verb phrase representing an action, process, event or happening.[5] This example illustrates the use: "John tried to convince Mary of his love. The attempt was not successful." ‘The attempt’ here refers to the verb phrase in the preceding sentence, i.e. 'tried to convince Mary of his love'. The easy cases like the given example can be solved by simply extracting the antecedent action. More complicated cases involve several competing alternatives and other semantic/syntactic information. Two techniques are proposed here, in particular for examples of the following form:

Example:
The cop found the kid but went after the kidnapper.
The decision he would regret forever.

Example:
John tried to convince Mary of his love and brought flowers for her.
The attempt was not successful.

(syntactic technique) Find the dominant antecedent action:

What does "decision" in the first example refer to: 'found the kid' or 'went after the kidnapper'? Apart from the semantic knowledge that ‘going for something’ is normally a deliberate action or decision but ‘finding’ is usually not one, we can notice that the 'dominant' antecedent action is "went after the kidnapper" implied by the presence of "but". Other such indicators could be "despite", "however", etc.

(semantic technique) Use synonymity between action noun and the dominant verb in the antecedent verb phrases:

What does "The attempt" refer to in the second example above: "tried to convince Mary of his love" or "brought flowers for her"? Since "attempt" is synonymous with "try" but not with "bring", the balance is tilted in favor of the former.

V. Implementation and Results

[5] See Appendix C for examples of action nouns

An anaphora resolution software (ARS) `resolve` was written to implement some of the techniques described in the preceding sections.

A parse tree of the input sentences is first obtained using the Stanford Lexicalized Parser. The tree is then examined to obtain verb-phrase and noun referents as well as anaphora in the discourse. The program then proceeds with resolution.

The classical eliminative techniques of gender and number agreement were implemented to provide a basic resolver, so that other advanced techniques can reasonably be tested. The lists of proper nouns in Appendix B were used in addition to the gender knowledge about typical words such as "man" and "woman". In a similar eliminative fashion, action-nouns are compared against verb-phrases only. Proximity of an anaphor with the referent phrase is used as preference in the absence of other information.

The semantic technique of using synonymity to resolve action-noun anaphora is implemented. The action-noun is first mapped to the corresponding verb (e.g. demonstration->demonstrate) using the first list in Appendix C. The WordNet lexical database is then searched to find the synonyms of this verb. Then the algorithm checks for each verb-phrase referent whether its verb head is one of these synonyms. (Verb head is first reduced to its basic form using the second list in Appendix C). A certain (currently arbitrarily chosen amount of) preference/score is given if this anaphoric verb and the referent verb are found to be synonymous.

Java code for this application is given in Appendix H.

A separate utility called `attributify` was written to extract attribute information from prepositional phrases modifying noun referents. The script is given in Appendix G. Some results are given in Appendix F. This utility is not integrated into ARS as yet.

The results of ARS are impressive. However, it is not straight forward to test the effectiveness of a limited scope anaphora resolver. A generic corpus can not be use since the resolver is not comprehensive. Besides, the correctness would need to be checked with manually tagged data.

Some hand picked sentences were given to the system and the results are given in Appendix E. Note that the action-noun anaphora are neatly resolved even in the presence of competing referents (see the last example : 'The attempt'). Similarly, anaphora referring back to entities at a multi-sentence distance are also resolved (see the second occurrence of 'He'). Also, cataphora whose referents occur in the same sentence are also correctly resolved if the correct referent is the closest one among gender and number agreeing alternatives (see the 'post office' example). However, all the noun-phrase anaphora whose correct referents occur prior to other constraint matching referents would resolve incorrectly.

VI. Miscellaneous Philosophical Issues

This section describes a couple of philosophical questions brought forth by the anaphora resolution analysis.

Making linguistics:

1) language independent

It is desirable to be able to perform as much of the linguistic analysis and NLP systems design as possible, without referring to a specific language such as English, French, etc. This vision can be pursued by modeling a natural language after real world concepts, e.g. nouns and verbs abstracted as entities and actions, proper nouns as concrete entities (e.g. John) compared to abstract entities (e.g. man), and so on. The next step would be to devise a high level grammar that captures expressions about these objects and actions, and their attributes and relationships. This grammar would be invariant with respect to the order of phrases; it would only describe their mutual relationships. For example in English, simple adjectives normally come before the nouns they are modifying but in other languages such as Urdu and Persian, they can come before as well as after the noun (with the noun or adjective being in a slightly different morphology). With this so called universal linguistic device, it would be possible to at least design and describe, if not implement, the algorithms and systems such as those for anphora resolution without being specific about any language.

2) time-direction independent

Text usually follows chronological order of events but it does not always strictly hold, especially in literature, where earlier incidents can be described latter in the narration. Besides, sometimes order is irrelevant, e.g. in lists of facts.
However, a temporal order is normally followed at least locally in the discourse and any technique of AR has to take that into account; however, statistical approaches can potentially ignore direction, after assigning a lopsided probability distribution to the referents.

VII. Future Directions

- Design an evaluation strategy for assessing the performance of the system within its scope of anaphora types [6]

- Compare performance and complexity of history list method for multi-sentential resolution with other techniques such as [StrubHahn95]

- Consider associative action-noun anaphora, the verb-phrase antecedent counterpart of the regular noun-phrase case given in [MeyerDale02]; find where they could arise and how to resolve them

- Implement the syntactic technique of finding dominant antecedent verb phrase

- Incorporate attribute extracting module into the resolution program

- Determine statistically optimal scoring system for likelihood assignment based on preferences (i.e. how much to reward a referent for proximity or synonymity)

- Use adjective phrases, subjects of "have" verbs and possibly other structures to obtain further attribute information

[6] [BarbuMitkov01] can be helpful here

VIII. References

[Denber98] Michel Denber, Automatic Resolution of Anaphora in English, Technical report, Eastman Kodak Co., 1998; http://www.wlv.ac.uk/~le1825/anaphora_resolution_papers/denber.ps

[Mitkov99] Ruslan Mitkov, Anaphora Resolution: The State of the Art, 1999; Paper based on the COLING'98/ACL'98 tutorial on anaphora resolution; University of Wolverhampton

[StrubHahn95] Michael Strube and Udo Hahn, ParseTalk about Sentence- and Text-Level Anaphora, 1995; Computational Linguistics Research Group, Freiburg University, Germany; http://acl.ldc.upenn.edu/E/E95/E95-1033.pdf

[MeyerDale02] Josef Meyer and Robert Dale, Mining a Corpus to Support Associative Anaphora Resolution, 2002; Centre for Language Technology Macquarie University, Sydney, Australia

[BarbuMitkov01] Catalina Barbu and Ruslan Mitkov, Evaluation tool for rule-based anaphora resolution methods, 2001; School of Humanities, Languages and Social Sciences, University of Wolverhampton

IX. Appendices

Appendix A
A partial list of noun-heads serving as anaphora

it its it's he she his her they them they're himself herself themselves one none everyone all any many one <cardinal> <ordinal> this these that those

Appendix B
Demo lists of male and female proper nouns

males.names:	females.names:
----------------	------------------
John	Mary
Smith	Sally
Chris	Kathy
Rajat	Tina
Imran	Kristina
	Julie

Appendix C
Demo lists of action-nouns and word-bases

action.nouns:
----

demonstration	demonstrate
persuation	persuade
action	act
play	play
attempt	attempt
struggle	struggle
endeavor	endeavor

word.bases:
----

demonstrated	demonstrate
protested	protest
colors	color
colored	color
attempted	attempt
saw	see
tried	try

Appendix D
A programming language representation of referent (real world objects or phrases) -- class definition in Java

```
class Referent {
   // (ideally) the most comprehensive phrase describing the referent in the text
   String phrase;

   // the noun of verb head representing the referent
   String head;

   // 'M', 'F', 'N' for neutral, 'U' for unknown; detault is 'N'
   // 8 bits allow for more genders in the future -;)
   char gender;

   char number; // 's' for single, 'p' for multiple; default is 's'

   // 'e' for entity, 'a' for action/verb; default is 'e'
   char type;

   // gender, number and type are constraint variables; they need to strictly
   //  match between the referent and anaphora

   // the index in the text where it was last referred to; apart from helping
   //  in determining that it is too old to keep in the database/records, it
   //  can also be used to determine proximity to the anaphora
   // not used in the current system
   int last_touch_index;

   // for future use; the index in text where the referent is first mentioned
   int first_touch_index;

   Vector attributes; // vector of strings, attribute names; e.g. 'color',
                      // 'size', 'park' (as value of the location attrib.)
                      // etc. not used in the current system
}
```

Appendix E
Output of a test run of ARS (Anaphora Resolution System developed in the project)

```
The discourse:
--------------
Mary saw a fat bald man in the park on her way.
John is walking down the street.
He is lonely.
He attempted to convince Sally of his intelligence.
The endeavor was more interesting than the outcome.
Because she was going to the post office, Julie was asked to post
a small parcel.
Many famous scientists are coming to the conference.
The participants are going to present their useless research.
John tried to convince Mary of his love and brought flowers for
her.
The attempt was not successful.


The anaphora used in the discourse, in the order of appearance,
and the actions/entities they refer to:
-----------------------------------------
anaphora -> referent
--------------------
 the park -> way
 her -> Mary
 the street -> way
 He -> John
 He -> John
 his -> John
 The endeavor -> attempted to convince Sally of his intelligence
 the outcome -> intelligence
 she -> Julie
 the post -> office
 the conference -> a small parcel
 The participants -> scientists
 their -> scientists
 his -> John
 her -> Mary
 The attempt -> tried to convince Mary of his love

The referents (entities and actions) mentioned
in the discourse, in the order of appearance:
---------------------------------------------
 female entity: Mary
 male entity: a fat bald man
 entity: way
 action: saw a fat bald man in the park on her way
 male entity: John
 action: walking down the street
 female entity: Sally
 entity: intelligence
 action: attempted to convince Sally of his intelligence
 entity: office
 female entity: Julie
 entity: a small parcel
 action: going to the post office
 action: asked to post a small parcel
 plural entity: scientists
 action: coming to the conference
 entity: research
 action: going to present their useless research
 male entity: John
 female entity: Mary
 entity: love
 plural entity: flowers
 action: tried to convince Mary of his love
 action: brought flowers for her
```

Appendix F
Results of attribute extracting utility 'attributify'

```
Sentences:
----------
Mary saw a fat bald man in the park on her way.
There were dresses of serveral different colors and styles.
Julie placed the books in their corresponding shelves.

Results (may be none):
<entity> <attribute name or value, e.g. location or house>
----------------------------------------------------------
man    park
dresses    colors
books    shelves
```

Appendix G
attributify: the utility to extract attribute information about entities from prepositional phrases

attributify

```
#!/bin/csh

touch test.tree.raw test.tree object_atrrib.pairs
rm -f test.tree.raw test.tree object_atrrib.pairs

setenv PWF $cwd # PWF = Path of the Working Folder
cd ~iqsayed/cs224n/lexparser-2003-03-25
lexparser.csh $PWF/attrib.sentences > $PWF/test.tree.raw
cd $PWF
```

```
# delete messages from the output
sed -e '/^Parsing/d' test.tree.raw > test.tree

setenv PATH /afs/ir/data/linguistic-data/bin/sun4x_57:$PATH
tprep test.tree
setenv TGREP_CORPUS ./test.tree.corpus
tgrep  '(NP  <  `NN|NNS)  $  (PP  <<  `NN|NNS)'  |  extract_attribs.pl  >
object_atrrib.pairs

echo; echo 'Results (may be none):';
echo '<entity> <attribute name or value, e.g. location or house>'
echo '-----------------------------------------------------------'
cat object_atrrib.pairs
echo;
```

extract_attribs.pl

```
#!/usr/local/bin/perl

while ( $line = <STDIN> ) {
    chop($line);
    $input .= $line;
}

@output = $input =~ /([A-Za-z0-9]+)\)/g;

$count = @output;
while ($count > 1) {
    print shift(@output), " ", shift(@output), "\n";
    $count = @output;
}
```

Appendix H
Shell script and Java code for Anaphora Resolver developed in the project

resolve

```
#!/bin/csh
touch test.tree.raw test.tree
rm -f test.tree.raw test.tree
setenv PWF $cwd
cd ~iqsayed/cs224n/lexparser-2003-03-25
lexparser.csh $PWF/resolve.sentences > $PWF/test.tree.raw
cd $PWF
sed -e '/^Parsing/d' test.tree.raw > test.tree # delete the messages from
output
setenv PATH /afs/ir/data/linguistic-data/bin/sun4x_57:$PATH
setenv PATH ~iqsayed/cs224n/WordNet:$PATH
make
java Resolve
```

Resolve.java

```
import java.io.*;
import java.util.*;

// ----------------------------------------------------------------------
// This same structure is used to represent both referents as well as anphora

class Referent extends Object {
    // the most comprehensive (ideally) phrase describing the referent in the
    //  text
    String phrase;
    // the noun of verb head representing the referent
    String head;

    // 'M', 'F', 'N' for neutral, 'U' for unknown;
    // 8 bits allow for more genders in the future -;)
    char gender;

    char number; // 's' for single, 'p' for multiple

    // 'e' for entity, 'a' for action/verb
    char type;

    // gender, number and type are constraint variables; they need to strictly
    //  match between the referent and anaphora

    // the position in the text where it was last referred to; apart from
    // helping in determining that it is too old to keep in the
    // database/records, it can also be used to determine proximity to the
    // anaphora not used in this version
    // Referents have integer values; anaphora have values of the form n+1/2
    float ltp;

    // for future use
    int first_touch_index;


    Vector attributes; // vector of strings, attribute names; e.g. 'color',
                      // 'size', 'park' (as value of the location attrib.)
                      // etc. not used in the code for now.

    public Referent(String p) {
       // default values
       phrase = p;
       gender = 'N';
       number = 's';
       type = 'e';
    }

    public void print() {
       if (gender=='M') System.out.print(" male");
       else if (gender=='F')System.out.print(" female");

       if (number=='p') System.out.print(" plural");

       if (type=='a') System.out.print(" action: ");
       else System.out.print(" entity: ");

       System.out.println( phrase );
    }
}
// ----------------------------------------------------------------------
class Num extends Object {
    private float val_;
    public Num( float v ) {
       val_ = v;
    }
    public void incr() { val_++; }
    public void decr() { val_--; }
    public void add(float v) { val_ += v; }
    public void value(float v) { val_ = v; }
    public float value() { return val_; }
}
// ----------------------------------------------------------------------
class Resolver {
    HashSet maleNames, femaleNames;
    Hashtable actionNouns, wordBases;
    static String synonyms;

    // referent entities/actions that are not refered to for very long in the
    //  text (say 15-20 sentences) are removed from consideration in anaphora
    // resolution; not used for now though.
    int purge_referents_every_this_many_words;
    // ----------------------------------------------------------------------
    public void loadProperNouns() throws IOException, FileNotFoundException {
       maleNames = new HashSet();
       femaleNames = new HashSet();
       String line, token;

       // File names of lists are hard coded to avoid unnecessary flexibility
       // Files are assumed to contain one name per line and possibly white
       //  space but nothing else

       File file = new File("males.names");
       BufferedReader in = new BufferedReader( new FileReader(file) );
       while ( (line=in.readLine()) != null ) {
           token = line.trim();
           maleNames.add( token );
       }
       maleNames.add("man"); maleNames.add("boy");
       maleNames.add("male");

       file = new File("females.names");
       in = new BufferedReader( new FileReader(file) );
       while ( (line=in.readLine()) != null ) {
           token = line.trim();
           femaleNames.add( token );
       }
       femaleNames.add("woman"); femaleNames.add("girl");
       femaleNames.add("female");
    }
    // ----------------------------------------------------------------------
    public void loadActionNounsAndWordBases()
       throws IOException, FileNotFoundException {
       actionNouns = new Hashtable();

       String line, key, value;

       // File names of lists are hard coded to avoid unnecessary flexibility
       // Files are assumed to contain one noun-verb pair per line and possibly
       //  white space but nothing else

       File file = new File("action.nouns");
       BufferedReader in = new BufferedReader( new FileReader(file) );
       while ( (line=in.readLine()) != null ) {
           StringTokenizer st = new StringTokenizer(line);
           if ( st.countTokens() == 2 ) {
```

```
          key = st.nextToken();
          value = st.nextToken();
          actionNouns.put( key, value );
       }
   }

   wordBases = new Hashtable();

   file = new File("word.bases");
   in = new BufferedReader( new FileReader(file) );
   while ( (line=in.readLine()) != null ) {
       StringTokenizer st = new StringTokenizer(line);
       if ( st.countTokens() == 2 ) {
          key = st.nextToken();
          value = st.nextToken();
          wordBases.put( key, value );
       }
   }
}
// ---------------------------------------------------------------------
public void printTestFile() throws IOException, FileNotFoundException {
   String line;
   // File name for the sentence file is hard coded to 'resolve.sentences'

   System.out.println( "\nThe discourse:");
   System.out.println( "--------------");

   File file = new File("resolve.sentences");
   BufferedReader in = new BufferedReader( new FileReader(file) );
   while ( (line=in.readLine()) != null ) {
       System.out.println( line );
   }
   System.out.println( "" );
}
// ---------------------------------------------------------------------
public void printLexicalData() {
   String token;
   Iterator it;

   System.out.println("Repository of male names:");
   it = femaleNames.iterator();
   while ( it.hasNext() ) {
       token = (String)it.next();
       System.out.println(token);
   }

   System.out.println("Repository of female names:");
   it = femaleNames.iterator();
   while ( it.hasNext() ) {
       token = (String)it.next();
       System.out.println(token);
   }

   String verb, noun;
   Enumeration e = actionNouns.keys();
   while ( e.hasMoreElements() ) {
       noun = (String)e.nextElement();
       verb = (String)actionNouns.get( noun );
       System.out.println( verb+"<->"+noun );
   }
}
// ---------------------------------------------------------------------
int essenceSynonymity(String word) throws IOException {
   String w = (String)wordBases.get(word); // e.g. tried -> try
   if (w==null) w=word; // essence is the same as the word
   int index = synonyms.indexOf(w);
   return index;
}
// ------------------------------------------------------------------------
void readVerbSynonymsOfActionNoun(String word)
   throws IOException, InterruptedException {
   String w = (String)actionNouns.get(word);
   if (w==null) return;
   Runtime rt = Runtime.getRuntime();

   Process p = rt.exec("wn "+ w +" -synsv");
   p.waitFor();

   InputStream in = p.getInputStream();
   byte[] byte_arr = new byte[in.available()];
   in.read( byte_arr );
   String process_output = new String(byte_arr);
   synonyms = process_output; // for use in essenceSynonymity()
}
// ------------------------------------------------------------------------
void readReferentsResolveAnaphora()
   throws IOException, FileNotFoundException, InterruptedException {
   // Each sentence in the original file is supposed to be terminated by
   //  a period i.e. '.'
   // Assumes that NN occurs as '(NN <string>)'; similar for NNS, NNP
   // Allows expressions of the form "DT [JJ]* [NN|NNS|NNP]"

   try {
       printTestFile();
   }
   catch (Exception e) {
       System.err.println("Could not read file\n"+e);
       System.exit(0);
   }

   Vector referents = new Vector();
   Vector anaphora = new Vector();

   String line, token, sentence, referent, next;
   String det = "";
   sentence = new String(""); referent = new String("");
   token = new String("");
   File file = new File("test.tree");
   BufferedReader in = new BufferedReader( new FileReader(file) );
   char mode = 0;

   System.out.println("\nThe anaphora used in the discourse, in "
          + "the order of appearance,\nand the actions/entities "
          + "they refer to:");
   System.out.println("----------------------------------------");
   System.out.println("anaphora -> referent");
   System.out.println("--------------------");

   while ( (line=in.readLine()) != null ) {
       sentence += line.trim();
       if ( line.indexOf('.') > -1 ) { // sentence terminates
          anaphora.clear();
          // Collect all the referents in this sentence in two parses of
            //  the tree description
          // Collect all the anaphora in the first of these parses
```

```
String tag="";
// 'true' here indicates that delimiters are to be returned
StringTokenizer st = new StringTokenizer(sentence," )(",true);
boolean dont_read = false; char num='s';
while ( st.hasMoreTokens() || dont_read ) {
    if (!dont_read) token = st.nextToken();
    dont_read = false;

    while (token.equals("(")) {
       // get the next token until a pos tag
       token = st.nextToken();
       tag = token;
    }

    if (token.equals("DT")) {
       // will collect until the next "NN|NNS"
       mode = 'd';
    }
    // option 2:
    else if ( token.equals("NN") || token.equals("NNS")
             || token.equals("NNP") ) {
       num = 's';
       if ( token.equals("NNS") ) num = 'p';
       token = st.nextToken(); // space ' '
       token = st.nextToken();
       if ( mode == 'd' ) {
           if (referent.length()>0) referent += " ";
           referent += token;
           mode=0;
           if ( det.equals("the") || det.equals("The") ) {
              Referent a = new Referent(referent);
              a.head = token;
              a.number = num;
              a.ltp = 0.5f+referents.size();

              if ( actionNouns.get(token) != null ) {
                  a.type = 'a';
              }

              anaphora.add( a );
              // considered an anaphora, not referent,
              //  if determiner 'the' is used
              referent = "";
           }
       }
       else referent = token;
    }
    // keep it last option
    else if ( !token.equals(")") && !token.equals(" ")
             && st.hasMoreTokens() ) {
       next = st.nextToken();
       if ( next.equals(")") ) {
           // token is a word in the sentence
           if ( mode == 'd' ) {
              if (referent.length()>0) referent += " ";
              referent += token;
           }

           if ( tag.equals("DT") ) {
              det = token;
           }
           else if ( tag.equals("PRP")
                    || tag.equals("PRP$") ) {
```

```
              Referent a = new Referent(token);
              a.head = token;
              a.ltp = 0.5f+referents.size();
              if ( token.equals("his") || token.equals("he")
                   || token.equals("His")
                   || token.equals("He") ) {
                  a.gender = 'M';
              }
              else if ( token.equals("her")
                       || token.equals("she")
                       || token.equals("Her")
                       || token.equals("She") ) {
                  a.gender = 'F';
              }
              else if ( token.equals("they")
                       || token.equals("they're")
                       || token.equals("them")
                       || token.equals("their")) {
                  a.number = 'p';
              }

              anaphora.add( a );
           } // if(PRP)
       }
       else {
           token = next;
           dont_read = true;
       }
    }

    if ( mode==0 && referent.length()>0 ) {

       Referent r = new Referent(referent);
       r.head = token; // coming from option 2
       r.number = num;
       r.ltp = referents.size();
       if ( maleNames.contains(token) ) r.gender = 'M';
       if ( femaleNames.contains(token) ) {
           if (r.gender == 'M') r.gender = 'U';
           else r.gender = 'F';
       }

       referents.add( r );
       referent = "";
    }

}
// second pass through the sentence; in search of verb phrases
st = new StringTokenizer(sentence," )(",true);
dont_read = false; mode = 0; referent = "";
int count=0; tag=""; String head="";
while ( st.hasMoreTokens() || dont_read ) {
    if (!dont_read) token = st.nextToken();
    dont_read = false;


    while (token.equals("(")) {
       // get the next token until a pos tag
       if (mode=='v') count++;
       token = st.nextToken();
       tag = token;
    }
    if (token.equals("VP")) {
       // will collect until VP is completed
       mode = 'v';
       count = 1;
    }
    else if ( mode == 'v' && token.equals(")") ) {
       count--;
    }
    // keep it last option
    else if ( mode == 'v' && !token.equals(")")
             && !token.equals(" ") && st.hasMoreTokens() ) {
       next = st.nextToken();
       if ( next.equals(")") ) {
           if (referent.length()>0) referent += " ";
           referent += token;
           count--;
           // more verb forms can be added
           if ( tag.equals("VB") || tag.equals("VBD")
               || tag.equals("VBZ") || tag.equals("VBP") ) {
              if ( token.equals("is") || token.equals("was")
                   || token.equals("are")
                   || token.equals("were") ) {
                  referent = "";
                  mode = 0;
              }
              else if ( head.length()==0 ) { // head of the
                  // verb phrase head of the verb phrase is
                  // assumed to occur before all other verbs
                  // in the phrase
                  head = token;
              }
           }
       }
       else {
           token = next;
           dont_read = true;
       }
    }
    if ( count==0 && referent.length()>0 ) {
       Referent r = new Referent(referent);
       r.head = head;
       r.type = 'a';
       r.ltp = referents.size();
       referents.add( r );
       referent = "";
       mode = 0;
       head="";
    }
}
sentence = "";

// Now resolve all the anaphora encountered in this sentence
// Necessary data is in the vector anaphora and referents
for (int i=0; i<anaphora.size(); i++) {
    Referent an = (Referent)anaphora.elementAt(i);

    if ( an.type == 'a' ) {
       readVerbSynonymsOfActionNoun(an.head);
    }

    float best_prior = 100;
    Referent best_r = null;
    for (int j=referents.size()-1; j>=0; j--) {
```

```
                    Referent rf = (Referent)referents.elementAt(j);

                    float prior = Math.abs(rf.ltp-an.ltp);

                    if ( rf.type == an.type && rf.gender == an.gender
                      && rf.number == an.number ) {

                        if ( rf.type == 'a' ) {
                           // reward for synonymity
                           if (essenceSynonymity(rf.head)>-1) {
                               // 10 is an arbitrary amount
                               prior -= 10;
                           }
                        }
                        // proximity as a preference; an antecedent is given
                        //  priority over a post-cedent when tie occurs
                        if ( prior<=best_prior ) {
                           best_r = rf;
                           best_prior = prior;
                        }
                    }
                 }
                 if ( best_r == null ) {
                    System.out.println( " " + an.phrase + " -> unknown!" );
                 }
                 else {
                    System.out.println( " " + an.phrase
                                     + " -> " + best_r.phrase );
                 }
             }
          } // if (sentence_terminated)
       } // while(line)

       System.out.println("\nThe referents (entities and actions) mentioned");
       System.out.println("in the discourse, in the order of appearance:");
       System.out.println("--------------------------------------------");
       for (int j=0; j<referents.size(); j++) {
           Referent rf = (Referent)referents.elementAt(j);
           rf.print();
       }
       System.out.println("");

    } // readReferentsResolveAnaphora()
} // class Resolver
// ---------------------------------------------------------------------
class Resolve {

    public static void main(String args[]) {
       Resolver resolver;
       resolver = new Resolver();
       try {
           resolver.loadProperNouns();
           resolver.loadActionNounsAndWordBases();
           resolver.readReferentsResolveAnaphora();
       }
       catch (Exception e) {
           System.err.println("Encountered problem -\n"+e);
           System.exit(0);
       }
    }
}
// ---------------------------------------------------------------------
```