Pronoun and Anaphor Resolution

6.863 Spring 2018: Final Project

Cordelia Avery
cavery@mit.edu

Gabriel Teixeira
gteixe@mit.edu

Max Vilgalys
vilgalys@mit.edu

May 16, 2018

Abstract

Our final project aims to perform anaphora resolution via a combination of syntactic and semantic analysis. Anaphora resolution refers to the identification of antecedents for non-referential elements in a document. It is an important area of natural language processing, as it enables parsers to extract meaning from sentences that contain pronouns and anaphors. Our approach starts with the fact that anaphor resolution is partially dictated by the syntax, adapting an approach from previous work to add more flexibility and take full advantage of advances in linguistic theory. To supplement this base algorithm we add selectional constraints to restrict the list of possible antecedents and perform syntax correction on incorrect parses to better utilize binding conditions. Thus we are able to not only resolve anaphora, but offer error correction, validated by gold data, on the output from parsing engines.

Contents

1 Introduction

Anaphora resolution, the process of determining which entities are associated with elements such as pronouns and anaphors, is an important area of natural language processing. The ability to resolve these references is valuable from theoretical and practical perspectives. It has widespread applications in the automated interpretation of natural language, as extracting semantic meaning from pronouns and anaphors is crucial to the accurate interpretation of text.

Our interest in building such a system initially stemmed from the cross-linguistic consistency of binding theory, the set of syntactic rules that govern the distribution of pronouns, anaphors, and referent nouns. We wanted to be able to take advantage of this property to produce a system that could potentially be used to resolve anaphors across multiple languages, given the correct parameter settings. As such, the core of our approach stems from a syntactic model, inspired by the one in [Hob78]. After completing the syntactic structure, we use selectional constraints to further narrow the field of possible antecedent, an approach outlined in [Hob78] and [RLR+10]. Because our end goal was to create a framework that was not only accurate but also language-independent, we adapt the algorithms provided to be able to take in language-specific parameters, such as head directionality, to more easily allow for this sort of cross-linguistic experimentation, as discussed in §5.

As anaphora resolution is a fairly well-studied field, our goal was to compare our results not only to previous work, but also to run our final dataset on the CoreNLP deterministic, [LPC+11], statistical, and neural net engines [CM15] for pronoun resolution. Preliminary testing with the deterministic parser showed that while it can correctly resolve references within simple matrix clauses, it fails to correctly index sentences with more complex embedded structure, as we can be seen from the following:

(1) *Mary$_i$ forced her$_i$ to dance.

(2) *Lucy$_j$ was tired, but Mary$_i$ forced her$_i$ to dance.

The deterministic CoreNLP parser incorrectly identifies "Mary" as the antecedent of "her" even when provided with a suitable alternative, as in sentence (2). This is likely due to the fact that the parser interprets "her" as residing in the embedded clause, as opposed to the matrix one, allowing "Mary" and "her" to co-refer without violating binding conditions. The precise syntax of this is explained in §2.

Due to the reliance of our algorithm on syntactic relations between elements in a sentence, the accuracy of our implementation is directly dependent on the accuracy of the parsing engine that is used. CoreNLP's failure to parse the above sentences thus impedes our ability to analyze them correctly. In an effort to resolve anaphora despite these errors, a major component of our algorithm focuses on correcting parses that fail to enforce the requisite binding conditions. This goes beyond simple anaphora resolution, to actually identify and improve upon the output from external parsers, and verify these corrections via improved performance in identifying antecedents.

Consider the following as an additional data point:

(3) Mary forced herself to dance.

At first glance, it would appear that (1) and (3) have the same structure; however, a parse tree that allows "her" and "Mary" to corefer in the former sentence would necessarily mean that binding conditions are not met for "herself" in the latter, implying that the parse must be incorrect. A more in-depth explanation of this behavior and its implications for error correction is given in §3.1. Syntactic constraints can thus correct both pronoun interpretation and assumed sentence structure.

With this in mind, we intend to test our results along three axes: first, the performance of our syntactic and selectional parser on a hand-annotated dataset in comparison to the CoreNLP engine; second, our system's ability to perform parser error correction and its impact on the accuracy of the algorithm; and finally, a cross-linguistic comparison of our parser's performance on parallel datasets in English and German.

We first provide an overview of binding theory in §2, as much of the related literature, presented in §3, makes reference to binding. Due to the heavy reliance on syntax and error correction, the bulk of our implementation actually resides in the implementation of our syntax tree, which we discuss in §4. We then present our algorithm to resolve references based on these tree structures in §5, and discuss results for the comparison to previous work, cross-linguistic flexibility, and error correction in §6.

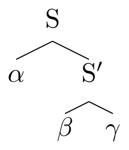
2 Binding Theory

Binding theory, as discussed in [Cho93], defines the set of syntactic relations that dictate how anaphors (reflexives such as "herself", "itself", etc); pronouns ("he", "they", etc); and referential expressions, or R-expressions (roughly, all other nouns, such as "the movies", "Sally", etc.) relate to each other. We give a background in binding theory before discussing previous work, as many of the syntactic-based approaches to anaphora resolution rely on its conditions.

At the center of binding theory is the notion of *c-command*, which plays a central role in the determination of antecedent relations. This structural relation is defined by [LU88] as follows:

C-command: For nodes α and β in a tree, α c-commands β iff every branching node dominating α dominates β and neither α nor β dominates the other.

Consider the following tree:



In order to determine whether α c-commands β, we first look to every branching node dominating α, which in this case is simply the node S, and check that it also dominates β, which it does. Lastly, we observe that neither α nor β dominates the other. Therefore, we say that α c-commands β.

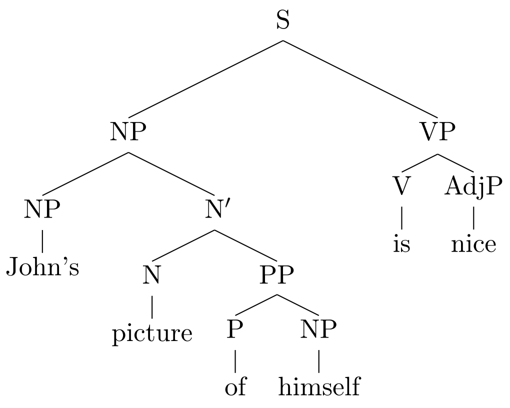
Binding: α binds β iff (i) α c-commands β and (ii) α and β coindexed.

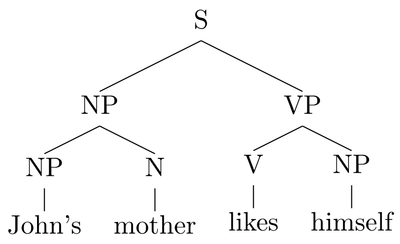
where α and β are said to be coindexed if they refer to the same entity. Consider the following sentences:

(4) John$_i$'s picture of himself$_i$ is nice.

(5) *John$_i$'s mother likes himself$_i$.

In (4), whose syntactic structure is shown below, we see that "John" c-commands "himself", and that the two co-refer. Thus, we say that "John" binds "himself." However, we see that in (5), "John" does not bind "himself" because the c-command relation is not met. We note that the reading of (5) where "John" and "himself" co-refer is ungrammatical, suggesting that this must be due to the failure of "John" to bind "himself".





We now present the core tenets of binding theory, the three conditions that describe the various relations that anaphors, pronouns, and R-expressions are allowed to exhibit.

Condition A: An anaphor (reflexive) must be bound within the smallest TP (a phrase equivalent to IP or S in CoreNLP output) that contains it.

Condition B: A pronoun cannot be bound within the smallest TP that contains it.

Condition C: R-expressions cannot be bound.

In (6), the anaphor "himself" is bound by "John", and they are both within the same TP, therefore Condition A is satisfied and the two may co-refer. However, in (7), although "John" c-commands "himself," the antecedent is outside of the smallest TP that contains the anaphor, so Condition A is violated and the two may not co-refer.

(6) John$_i$ washed himself$_i$.

(7) *John$_i$ asked Mary to wash himself$_i$.

Condition B enforces a requirement on pronouns that is complementary to that enforced by Condition A on anaphors. In (8), "John" binds "him" and is outside of the smallest TP that contains "him." In other words, because "John" is in the matrix clause while "him" is in the embedded clause, and the c-command relation holds, the two may co-refer. Contrastingly, (9) is ungrammatical because while "John" c-commands "him", they are both within the same TP, and therefore the pronoun cannot be bound, in accordance with Condition B.

(8) John$_i$ asked Mary to wash him$_i$.

(9) *John$_i$ washed him$_i$.

In (10) and (11), "John" (an R-expression), cannot be bound. In these examples, "John" is c-commanded by "he", and the antecedent lies inside and outside the smallest TP containing "John," respectively. However, Condition C prohibits the two from co-referring.

(10) *He$_i$ saw John$_i$

(11) *He$_i$ asked Mary to wash John$_i$.

3 Previous Work

There is a wealth of research related to anaphora resolution in natural language processing. These systems tend to fall into one of two groups: those that rely on machine learning to resolve references, and those that use syntactic and semantic interpretations to form the backbone of their systems. While the former approach has its advantages, we chose to build a system that focused on syntactic structure, in line with our original goal of taking advantage of the role of binding theory and syntax in determining coreferences.

In the latter camp, many reference [Hob78], which presents two main algorithms. The first, a straightforward syntactic approach, makes use of binding theory to rule out or require the coreference of entities in certain cases. The algorithm presented assumes the provision of a valid syntax tree, which is then traversed in a specific order to identify potential coreferents, which must be in the correct syntactic position and match the target in both gender and number. This initial algorithm can be improved upon by introducing additional selectional constraints that dictate the validity of actions when applied certain objects. Hobbs provides the following example:

(12) The castle in Camelot remained the residence of the king until 536 when he moved it to London.

The naive algorithm can identify the "he" can refer to "the king", because it is the only element that refers to an individual. However, an additional constraint, namely that large objects (such as castles), dates, and places cannot be moved, is required to ensure that "it" refers to "the residence of the king" as opposed to "the castle". Hobbs found the syntactic approach alone to have an accuracy of 88.3%. Adding selectional constraints raised the accuracy to 91.7%.

Hobb's second algorithm takes a semantic approach, used in conjunction with the syntactic one. Because semantic meaning carries across sentences in a way that syntax does not, this opens the door for multi-sentence anaphora resolution in addition to increasing performance sentence-internally. First, the algorithm defines relations based on real-world axioms, such as the fact that a bank is a building, and that a building has a roof. This allows us to interpret sentences such as the following:

(13) The dog was in the building, and the boy was running on the roof of it.

Under the assumption that buildings have roofs and dogs do not, we can correctly identify that “it” cannot possibly refer to “the dog”. In addition to these semantic classes, Hobbs also defines a set of possible relations between sentences, such as *contrast*, *temporal succession*, and *violated expectation*, among others. These relations require certain truth conditions to hold across sentences, which may be violated given an incorrect resolution of anaphora. The algorithm also introduces the concept of predicate interpretation, which not only defines what sort of arguments a word can take (i.e., “into” must take a region or area as an argument such as “the field”, and the full PP must be the argument of a movement verb such as “ran”), but can also identify elided elements. If someone is “running on a building”, as native speakers we infer that the person is actually running “on the roof a building”. This allows for the correct interpretation of the following:

(14) The dog was in the building, and the boy was running on it.

The reconstruction of this deleted element enables us to derive the same interpretation for (14) as we do for (13), even though we only know that buildings have roofs and dogs do not, and there is no mention of a roof in the latter sentence. The semantic algorithm is able, using these relations, to construct a “chain of inference” that ties “it” to “the building” and not to “the dog”. Unfortunately, the running time of this approach grows exponentially with the size of the lexicon, because all of these relations must be expanded to see which elements can co-refer; however, it is possible to narrow the search by performing it in conjunction with the initial syntactic algorithm, which specifies a subset of the lexicon that ought to be searched.

As alluded to above, many of the syntactic models found in the literature follow a similar model. Notably, [LL94] not only present a similar syntactic approach, but also reference the cross-linguistic properties of the system. They provide results only for English; however, the algorithm laid out is intended to be language-neutral, i.e., any language-specific parameter settings are intentionally excluded from the paper. Additionally, it was published 16 years after [Hob78], which was written prior to many of the more relevant iterations on binding theory[1]. With this in mind, we more closely reference the syntactic constraints laid out in [LL94] as the basis for our syntactic model, while still making use of [Hob78]’s selectional and semantic models. The control flow and overall design of both of these algorithms is consistent. We present our adapted version of the syntactic algorithm in §5 below.

[RLR$^+$10] iterate on the semantic side of this approach, presenting a multi-pass algorithm to assign features moving from high to low precision. This allows them to derive more precise semantic meanings that could further constrain the possible interpretation of the anaphora. They provide the following example

(15) The second attack occurred after some rocket firings aimed, apparently, toward [the israelis], apparently in retaliation. [we]’re checking our facts on that one. [the president is] concerned the strike...does not contribute to the security of [israel].

Most models are unable to correctly resolve “we”, which is usually determined to refer to “the israelis”, because they are both plural. However, determining more precise semantic relations first allows the system to first link “the israelis” to “israel”, and thus mark the former as inanimate. This means that, in a later pass, the system does not mark “the israelis” as a possible antecedent for “we”, which must be animate to take “checking” as a verb.

After implementing the syntactic approach, we implement a more straightforward semantic system that relies on selectional constraints, as presented in [Hob78]; however, our system is sufficiently modular that more complex semantic models, such as that presented in [RLR$^+$10], can be added moving forward.

3.1 CoreNLP Coreference Resolution

Stanford CoreNLP [MSB$^+$14] is a state-of-the-art natural language processing toolkit that is able to relate surface level phonology to its base forms; perform part of speech tagging; generate syntax

[1][Cho93], cited above, was originally published in 1981, and was a significant revision on Chomsky’s previous work on binding. [Hob78] was published before in 1978.

trees on sentences; and resolve noun and general anaphora references. The latter two functions are of particular interest to us. Performing anaphora resolution via syntactic method requires syntax trees for the entire dataset, and thus CoreNLP's parsing ability, which is supported across multiple languages, is invaluable to us. CoreNLP's coreference resolution functionality, `CorefAnnotator`, provides us with an convenient baseline against which to actively assess our system's performance. Importantly, the system is an open-source toolkit as opposed to a closed-source publication, which means that we can directly compare results on a specific dataset, as opposed to drawing conclusions from statistical performance.

CoreNLP's coreference engine is comprised of three systems: a deterministic rule-based system, which is able to achieve 49.5% accuracy on English[2] [LPC+11]; a statistical model, which uses machine learning on dependency parses to achieve an accuracy of 56.2% [CM15]; and a slower, but more accurate, neural net system, which has an accuracy of 60.0% in English [CM16].[3] While these results are certainly a good starting point, the deterministic parser fails to produce the correct indexing in the following:

(16) Mary$_i$ forced her$_j$ to dance.

(17) [$_S$ Mary$_i$ forced [$_S$ her$_i$ to dance]]

[RLR+10], which goes into detail about the deterministic coreference implementation, discusses the use of syntactic structure as part of pronoun resolution. Thus the derived syntax tree from such sentences may help determine why the system fails in these cases. As shown in the generated parse tree in Figure (17), "her" resides only in the embedded clause of (16), as opposed to being present in the matrix clause and controlling a coreferent element[4].

Binding conditions show that this interpretation is incorrect. If "her" was not in fact present in the matrix clause, then Binding Condition B would be irrelevant, as it only applies within the smallest TP . This would mean that the indexing produced in (1) would be valid. As native English speakers, we interpret this to be false; however, our goal is to be able to detect these errors automatically, which is not possible in this case. It is legal for "her" to be present in either the matrix or the embedded clause; its possible antecedents will differ between the two parses, but an incorrect parse in this case will simply lead to an incorrect interpretation without catching the error.

Fortunately, anaphors (the reflexive sisters to pronouns) are slightly more restricted, as in (18):

(18) Mary$_i$ forced herself$_i$ to dance.

(19) * [$_S$ Mary$_i$ forced [$_S$ herself$_i$ to dance]]

Binding Condition A tells us that "herself" in sentence (18) must be bound. Assuming that syntax informs semantics and not the other way around, there is no way to derive different structures for sentences (16) and (18). However, Binding Condition A dictates that "herself" must be bound, and therefore reside in the matrix clause, because there is no element in the embedded clause with which it can corefer. Thus binding theory states that there are *no* legal antecedents for the anaphor "herself" given the parse in (19), from which we must conclude that the parse is incorrect.

Given these conclusions, our goal in iterating on these previous attempts at anaphora resolution is to not only correctly resolve such references, but also to find the parallels between parses such as (17) and (19), and use the fact that anaphors cannot be legally resolved to automatically identify (and potentially improve upon) errors in parsing.

4 Configuration Setup and Tree Implementation

As alluded to above, we intend to use the output from the CoreNLP parser [MSB+14] as the backbone of our anaphora resolution algorithm. While the part-of-speech (POS) tags provided in

[2]They note that the numbers given are lower than in their corresponding published results, because their published code aims to be as general as possible, at the expense of accuracy.

[3]They also provide results for coreference resolution in Chinese. The accuracy of the corresponding systems is slightly lower, but comparable to that of English.

[4]See chapter 9 of [Car13] for a more in-depth discussion on the syntactic subjects of infinitival constructions.

the CoreNLP output distinguish singular and plural nouns, they do not do the same for pronouns. It is therefore necessary to establish an avenue for feature-specification for pronouns. Moreover, there is no mechanism by which NPs themselves are marked as plural in coordinating constructions as in sentence (20).

(20) John and Bill saw each other.

The term "each other" takes a plural antecedent, i.e., it cannot be bound by "John" and "Bill" individually, but must instead select for the complete plural NP, "John and Bill". To make the problem worse, CoreNLP identifies "each other" only in terms of its component parts, as opposed to a single term in and of itself. Neither "each" nor "other" are, by themselves, anaphors, but the phrase "each other" is. This distinction is not drawn in the output from CoreNLP. Thus to enable correct person and number agreement and identify all anaphora present in the input, it is necessary to identify features that are defined syntactically as opposed to simply lexically.

In an effort to address the lack of feature specification for pronouns, we begin our implementation by establishing configuration files for each of the languages that we wish to explore (at present, only English and German),[5]. The initial configs specify gender, person, and number for pronouns and other nouns in the lexicon that may lack a full feature set as provided by CoreNLP. Not only do these configs allow us to enforce agreement in the syntactic algorithm, but the config files alo provide an elegant way to specify selectional constraints.

In order to solve the latter problem of deriving features from the syntax, we designed and implemented our own `Tree` class which is able to generate the correct structure from the string representation of the tree output from CoreNLP, correct for syntax elements that are inconsistent with our algorithm, and extract feature information from certain constructions.

We discuss the format of the configuration files and their use in §4.1 before outlining our `Tree` implementation in §4.2.

4.1 Configuration File Format

The configuration file is first and foremost a lexicon, although we include information about head directionality in each language should it be necessary at a later point. A sample config, including a sample lexicon, can be found in Fig 1. Unlike the first iteration of the config file, this example also specifies the animacy of each term, in addition to the gender, person, number (shown here as "count"), case, and type markings. The first three specifications allow us to enforce basic agreement for words in the lexicon. The case marking is most important in the genitive case, as pronouns used as possessives are subject to slightly different binding conditions than they in other cased positions. The `type` specification distinguishes R-expressions (`R`), pronouns (`P`), and anaphors (`A`).

The values corresponding to these entries are designed to be matched via regular expressions,[6] which simplifies the anaphora resolution algorithm. If we wanted to compare the gender of "itself" and "she", for example, using the Python regex library `re`, calling `re.search("[fm]", "f")` and `re.search("f", "[fm]")` will both result in a match, regardless of whether we are attempting to match the more or less specific expression[7]. This simplifies comparisons, particularly in English, which does not have strong case or gender markings, and can thus result in lexical entries that span multiple gender, number, or case markings.

Note that the lexicon here only contains pronouns (or, more specifically, a pronoun and an anaphor), and animate nouns. While the full lexicon for our implementation is significantly larger, it still does not include the full set of words in our input. For those words that lack specific selectional features such as animacy, feature sets can be derived directly from the CoreNLP output, which provides singular and plural POS tags `NN` and `NNS`, respectively. Noun phrases that are not in the lexicon are assumed to be gender-neutral, third person, inanimate R-expressions. This enables us to expand our lexicon dynamically as our set of input sentences expands.

[5]The configs are included in our repo in `config.json`.
[6]`https://en.wikipedia.org/wiki/Regular_expression`
[7]Find python documentation on the regular expression package at `https://docs.python.org/2/library/re.html`

```
{
    "eng": {
        "I_dir": "left",
        "V_dir": "left",
        "lexicon": {
             "man": {
                "gender": "m",
                "person": "t",
                "count": "s",
                "case": "obj|dat|nom",
                "type": "R",
                "animate": true
            },
            "Mary": {
                "gender": "f",
                "person": "t",
                "count": "s",
                "case": "obj|dat|nom",
                "type": "R",
                "animate": true
            },
            "itself": {
                "gender": "[fm]",
                "person": "t",
                "count": "s",
                "case":"dat|obj",
                "type": "A",
                "animate": false
            },
            "she": {
                "gender": "f",
                "person": "t",
                "count": "s",
                "case": "nom",
                "type": "P",
                "animate": true
            }
        }
    }
}
```

Figure 1: Sample config for English.

In the future, starting configs can easily be expanded to contain feature markings derived from semantic interpretation, or otherwise inferred elsewhere in the narrative. An easy first step here would perhaps be to specify animacy markings for verbs' subjects or object, thus enabling us to update the lexicon when new words interact with such verbs. For example, if the term "rider" was not in the lexicon, but the verb "talk" carried an animacy marking on its subject, the sentence "The rider talked to the boy" would allow us to mark "rider" as animate. Unfortunately, this sort of implementation requires a more expansive set of verb forms, whose compilation we took to be outside the scope of this project. However, we believe our lexicon implementation lends itself naturally to this sort of expansion.

4.2 Tree implementation

We take particular care in discussing our implementation of syntax trees, as their construction bears much of the burden in the execution of the resolution algorithm. The algorithm itself, as seen in §5.1 below, is actually fairly simple, but this is largely because it is able to rely on rich feature settings and structural relations with the parse tree.

While our tree implementation does not strictly conform to X-bar schema, our goal in its design was to allow it to represent schema of this type should the input syntax be consistent. Unlike some traditional tree implementations, where the complete tree object is distinct from the nodes it contains, in our `Tree` class each node in the tree is itself a tree (thus the full parse tree is equivalent to its root node). We chose this implementation because, due to the nature of our algorithm, we need very little information about the tree overall outside of its nodes' relations.

Each `Tree` object contains a `self.__node` field, which corresponds to either a `Bar` or a `Head` object. A `Bar` object contains one or more subtrees as children, while a `Head` object corresponds to a leaf node. All external interactions with `Bar` and `Head` objects must go through their parent `Tree` node. More specific information about this class structure can be found in §4.2.1.

Because our algorithm relies heavily on tree structure, and must be able to enforce agreement at bar nodes in addition to leaves, feature specifications from the lexicon are also added to nodes during construction. As mentioned in the previous section, however, it sometimes is necessary to look further than an single leaf to fully determine the feature set and, by extension, enable anaphora resolution. The mechanism by which features are added to nodes is described in §4.2.2.

We also attempt to correct for bad parser output. While CoreNLP's parser is effective in the larger scheme of things, it does generate parses for certain types of sentences that prevent binding conditions from being correctly applied. We discuss these problems and our solutions in §4.2.3.

4.2.1 Tree-Bar-Head Interaction

As specified above, each node in a tree is itself a tree object. To distinguish between the three as a whole and any one of its component subtrees, we therefore use the term `tree` to refer to the complete tree, and the term `node` to refer to a subtree within this tree. Each node contains information about the *external* view of itself, i.e., its POS and its parent node. Each node also contains a `self.__node` object that corresponds to either a `Head` or `Bar` object, which contains information about the *internal* structure this node, most notably its children and additional structural information useful for debugging and formatting output.

In addition to the `self.__node` variable, the two main variables in a node are its parent, `self.__parent`, a reference to its parent node; its POS tag `self.__tag`, which used in the anaphora resolution algorithm. In its constructor, each node is passed an object containing nested lists of POS tags and words, corresponding to the CoreNLP format of a tree. This format is shown in Fig 2. These tuples are converted to lists of a similar structure, with some small changes.[8] While the CoreNLP structure represents each subtree as a list object of length two or more, containing a POS tag and one or more children, our parser updates the input to always contain a list of two

[8]See the `process_string` function in `tree2.py` for source code.

```
(ROOT
  (S
    (NP (DT The) (NNS men))
    (VP (VBP seem)
      (PP (TO to)
        (NP (DT each) (JJ other)))
      (S
        (VP (TO to)
          (VP (VB be)
            (ADJP (JJ clever))))))
    (. .)))
```

Figure 2: CoreNLP output format for sample sentence.

objects to be passed to the `Tree` constructor, the first of which again corresponds to the node's POS tag. The second object is either a single string, corresponding to a leaf; or list of objects of the parent format, corresponding to child subtrees. For example, on input list `[‘NNS’, ‘men’]`, the `Tree` constructor will set `self.__tag` equal to `‘NNS’`, and `self.__node` equal to a `Head` object corresponding to the string "men". In contrast, on input `[‘NP’, [[‘DT’, ‘The’], [‘NNS’, ‘men’]]]`, `self.__node` will be set to a `Bar` object containing a list of subtrees corresponding to inputs `[‘DT’, ‘The’]` and `[‘NNS’, ‘men’]`.

Note that the output from CoreNLP is not guaranteed to be binary branching. For example, noun phrases containing adjectivals that in turn contain adverbs will store both adjective and its corresponding adverb at the `NP` level, as opposed to storing each one at its own `N’` node. We considered converting the output to be a strictly binary branching format, particularly as an earlier iteration of our algorithm assumed a binary scheme; however, unrelated updates to the algorithm rendered this step unnecessary. We do include the option of merging adverb-adjective constructions into a single `AdjP` node, with no intenral hierarchical structure, but this does not have an effect on the output of the algorithm.

In addition to these bare minimum structural components, each node in the tree also contains two ID variables, a root index and a local index. The root index corresponds to the ID of the tree itself, and is the same for all nodes in the tree, while the local index is unique for each node in a tree. The local indexing is consistent with the order in which nodes are visited in the depth-first-search, left-to-right traversal of the tree. The root index and starting local index can be specified when the root node is constructed[9]; however, neither of these variables are accessible externally. We implement the `__hash__()` function in the `Tree` to return a hash of the tuple `(root_index, local_index)`. This is important to enable, as our implementation of the resolution algorithm relies on the ability to insert nodes into sets.

Leaf nodes also have unique indices, with an indexing scheme equivalent to a 1-indexed, left-to-right traversal of the leaves.[10] This is consistent with the way in which the CoreNLP Coref engine specifies coreferences, allowing us to more easily validate our output. Leaf indices are stored in the corresponding `Head` objects, while `Bar` objects contain the start index, inclusive, and end index, exclusive, of the leaves they contain. A `Tree` has access to its self.__node object's leaf indices.

4.2.2 Adding Feature Specification to Nodes

The config file generally specifies features at the token, or word, level.[11] In X-bar schema, this corresponds to a phrase head, while in our tree implementation, this is equivalent to a leaf, or `Head`, node. However, in the resolution algorithm, we generally want to perform feature matching at the XP level, which captures all relevant features and C-command effects. Take the following as an example:

[9]Our intention in the implementation is that local indices always start at 0, while each tree should be given a unique ID should it be necessary to maintain references to multiple `Tree` objects at once.
[10]This is, in turn, equivalent to a left-to-right traversal of the tokens in a sentence.
[11]Not always. See below.

(21) $[_S\ [_{NP}\ [_{NP}$ John's] father] saw him]

In sentence (21), the head "father" does not C-command "him", in the sense that the closest maximal projection to this head (its parent NP) does not dominate "him"; however, the phrase "John's father" *does* C-command "him" (and, in accordance with binding theory, these two cannot co-refer). Thus we have a disconnect between head-specified features and XP-specified binding properties.

We address this inconsistency by propagating feature markings up to the nearest maximal projection. `Tree` objects also contain a `self.__config` object to store these values. Each time a leaf is added to the tree whose string is in the lexicon, or whose feature markings can be derived from its POS tag, we set its parent's config variable to match its feature specifications. Note that, because the output of the CoreNLP parser is non-binary, feature heads are the direct children of their maximal projections. If we forced our tree to conform to an X-bar schema containing intermediary nodes, we would instead set the config variable at the head's maximal projection.

While propagation of lexical heads provides a good starting point for the syntactic algorithm, generated POS tags for individual tokens do not reflect the fact that these tokens may have a different meaning when part of a constituent group. In order to address this problem, we also perform merge operations on multi-token constructions such as "each other", whose composite meaning is unrelated to their individual meanings; coordinating constructions such as "John and Bill", which as a unit are plural as opposed to singular; and multi-token names such "Mr. Smith", which we want to match to anaphora as a unit.

All three of these merge operations are fairly straightforward:

Multi-token Reflexives In the case of multi-token reflexives, we are merely looking for neighboring nodes, who are both the children of an `NP` node, the combination of whose strings is in the lexicon. For example, the node corresponding to input string `[NP, [[DT, each], [JJ, other], [POS, 's]]]` will not receive any feature markings, because the features of its children individually are, for lack of a better word, nonsense. However, iterating over its output string "each other 's" with a window of size two identifies "each other" as being in the lexicon, and assigns its parent `NP` a third-person plural, gender-neutral reflexive feature set.

Multi-token Names Multi-token names are identified in much the same way. In this case, the algorithm iterates over the node's children searching for neighboring children of type `NNP`, which corresponds to a proper noun. Should the gender of the two names be inconsistent, we naively assume that names in our data set follow the Western name order.[12]. The parent node thus takes the feature set of the first element in the name.

Coordinating Conjunctions Coordinating conjunctions can be found by iterating over a node's children searching for a `CC` node flanked by two `NP` nodes or noun heads. The three children at the node `[NP, [[NNP, John], [CC, and], [NNP, Bill]]]`, for example, fulfill this requirement. Unlike the previous merges, however, coordinating conjunctions do not have an entirely different feature set than their component parts, or borrow it entirely from one component; rather, the feature set of "John and Bill" is a *combination* of its components' features.[13]

4.2.3 Syntax Corrections

While the syntax of the output from CoreNLP is largely correct, the toolkit systematically fails to produce parses that enforce binding conditions under genitive and exceptional case marking constructions. The former refers to possessives such as "John's father", which contains two nested noun phrases. The latter is merely the tip of an iceberg of linguistic theory, which we do not attempt to discuss here; however, the succinct description is that the subjects of embedded clauses receive accusative or dative case as opposed to nominative in certain constructions.

[12] https://en.wikipedia.org/wiki/Personal_name#Name_order

[13] See the `intersect_configs` function in `tree2.py` for an implementation of the merge function.

Genitive Constructions Noun phrases that contain possessives such as "John's " take the same place as determiners within a larger NP, in the specifier of the phrase. Take the following noun phrases, for example:

(22) $[_{NP}$ $[_{NP}$ John's] father]

(23) $[_{NP}$ $[_{NP}$ $[_{NP}$ John's] father's] picture $[_{PP}$ of himself]]

(24) $[_{NP}$ $[_{NP}$ $[_{NP}$ John's] father's] picture $[_{PP}$ of him]]

Note that, under this construction, "John's father" C-commands "himself" in sentence (23), while "John" does not. This is reflected in the fact that "himself" must refer to the father. This syntax is also consistent with the interpretation derived in (24), where "him" can refer to "John", but *not* to his father, again due to binding conditions. However, the CoreNLP inserts an extraneous noun phrase into these constructions, interfering with binding conditions:

(25) $[_{NP}$ $[_{NP}$ $[_{NP}$ $[_{NP}$ $[_{NP}$ John's] father's] portrait] $[_{PP}$ of himself]]]

In this construction, the PP "of himself" is not a direct constituent of "John's father's portrait". This means that "John's father" does not C-command "himself"; in other words, the anaphor is not bound, violating Condition A. In replacing "himself" with "him", the same syntactic conditions holds, incorrectly allowing "him" to co-refer with "John's father".

We refer to this extraneous NP as a *dummy* node, i.e., a node that introduces an additional level in the tree without adding any semantic value. In proper X-bar schema, this node would likely be an N' node that would not interfere in binding conditions. In our implementation, we check for these extraneous NP nodes when we encounter genitives during tree construction. If one of a nodes children is a dummy node, this parent node absorbs its children, effectively eliminating the barrier to the C-command structure.

ECM constructions Exceptional case marking constructions are a fairly complex area of syntactic theory, whose analysis is far outside the scope of this project. [Las98] discuss the theory behind these constructions in greater detail. In short, certain sentence constructions will give the subject of an embedded clause dative or accusative case, as in the following:

(26) John believes himself to be clever.

(27) John believes him to be clever.

(28) John told himself not to talk to him.

This is referred to as "exceptional case marking" because subjects usually receive nominative case in Nominative-accusative languages, of which English and German are both examples.[14] These sorts of constructions pose an interesting theoretical problem: binding theory tells us that "himself" must be bound within the smallest IP; thus in sentences (26) and (28), "himself" must reside in the matrix clause; however, in (28), "himself" and "him" cannot co-refer, implying that "himself" is in the embedded clause. If it were not, binding conditions would not apply to "him".

Linguists attempts to satisfy both requirements, and posit that the element in question resides in both clauses; or rather, there exists a construct which *refers* to this element in both clauses. The NP "himself" receives case in the matrix clause, but controls (i.e., co-refers with) a silent subject in the embedded clause [Las98], [Hae94]. We do not attempt to discuss the validity of various theories on ECM constructions here; however, their conclusions do have an impact on our implementation. CoreNLP interprets these embedded subjects as residing in the embedded clause, causing binding conditions to fail in these cases. Interestingly, the CoreNLP Coref engine also fails to correctly identify antecedents in these cases.

In an attempt to reflect this dichotomy in NP location, we verify that the first argument of each S

[14]See https://en.wikipedia.org/wiki/Nominative-accusative_language for a description of such languages. Essentially, the subjects of nominative-accusative languages always behaves grammatically as a subject; this is contrast to ergative-absolutive languages, whose subjects behave differently depending on whether or not the verb in question is transitive.

node, the equivalent of the IP clause, is an NP with nominative case. If this is not true, then we raise this NP to the parent of the S clause, but leave a Trace object in its place. The Tree class is updated to contain a list of traces; when we run the anaphora resolution on a node, we also check for nodes that are C-commanded by its trace, allowing us to apply binding conditions within two clauses simultaneously. This enables correct anaphora resolution in sentences such as (28)

5 Anaphora Resolution Algorithm

As in previous approaches, we began our implementation with a straightforward syntactic algorithm before introducing selection constraints that can further narrow the field of possible antecedents for each node. Once syntactic and selectional constraints are working sufficiently in English, we expanded our parser cross-linguistically, to handle a corresponding dataset in German.

5.1 Syntactic Algorithm

The original syntactic algorithm laid out in [Hob78], was able to achieve good results; however, this algorithm was developed before many of the modern binding principles were formally introduced in [Cho93]. Although the model was clarified in [LL94], we chose to use these previous implementations largely as guidance in producing our own algorithm as opposed to as strict instructions.

Our iteration on these previous attempts relies more heavily on binding theory, as we divide anaphora into anaphors, pronouns, and R-expressions, subject to the binding conditions A, B, and C, respectively. Given a target node, we compute three sets of nodes:

synset: the set of all other nodes in the sentence that do not directly dominate our target node, and whose features match

c_commanding: the set of all nodes in the sentence that C-command the target node whose features match within the target scope (e.g., if the target node is a pronoun or anaphor, we only check for C-commanding nodes within the smallest IP)

governed: the set of all nodes in the sentence C-commanded by the target such that their coreference would violate binding conditions (i.e., all C-commanded R-expressions, and pronouns C-commanded within the smallest IP).

Binding theory tells us that the anaphors must be bound, while pronouns and R-expressions cannot be bound (all within their target scope). Thus, given a target node, if its type is specified as A, we return c_commanding - governed; it its type is specified as either P or R, we return synset - c_commanding - governed. Code for the algorithm is shown in Fig. 3.

The construction of the c_commanding and governed sets is performed via traversal of the tree

```
"""
node: target node to resolve
NP_nodes: list of all NP nodes in the current sentence

returns: list of possible antecedents
"""
def resolve_anaphor(node, NP_nodes):
    synset = filter (lambda n: match(node, n) and \
                        not dominates(n, node),
                        NP_nodes)
    c_commanding = get_c_commanding_nodes(node)
    governing = get_governing_nodes(node)

    if node.config()['type'] == 'A':
      return c_commanding - governing
    else:
      return (synset - c_commanding) - governing
```

Figure 3: Code for syntactic algorithm.

and basic feature matching. A set of all NP nodes in the Tree is maintained during construction of the tree. This set is passed as an argument to the resolution algorithm, and filtered to produce the synset set. The match function iterates over the specified feature set and performs regex matching or boolean comparison, depending on the feature type, to verify that the features are compatible. Code for get_c_commanding_nodes, get_governing_nodes, and match functions can be found in tree2.py.

5.2 Selectional Constraints

The algorithm in 5.1 returns a list of potential antecedents, ranked by proximity to the pronoun in question. [LL94] lists a few minimal compatability requirements: number, person, and gender, and some very basic semantics that define objects to which certain verbs cannot be applied, as referenced in [Hob78]. [LL94] also discuss a salience weighting system to more intelligently rank candidate anaphors and determine the most likely match.

As alluded to above, selectional constraints are readily included in our implementation via the config file. Implementing an animacy filter, for example, is as simple as specifying an additional animate field in the configuration file for requisite words. We can see the potential results of applying selection constraints in sentence (29).

(29) *[Their pictures]$_i$ of [each other]$_i$ are nice.

Without selectional constraints, our system identifies "their picture" as a possible antecedent of "each other". To humans this is clearly incorrect, as "each other" implies animacy. "their" by itself, on the other hand, is a valid antecedent here.[15]

Unlike the previous lexicon entries, which were matched via regular expressions, these selectional constraints are given boolean values. Because some entries in the lexicon may lack some selectional constraints, these values by default are interpreted to be false.

Our current implementation only applies animacy as a selectional constraint.

5.3 Cross-Linguistic Resolution

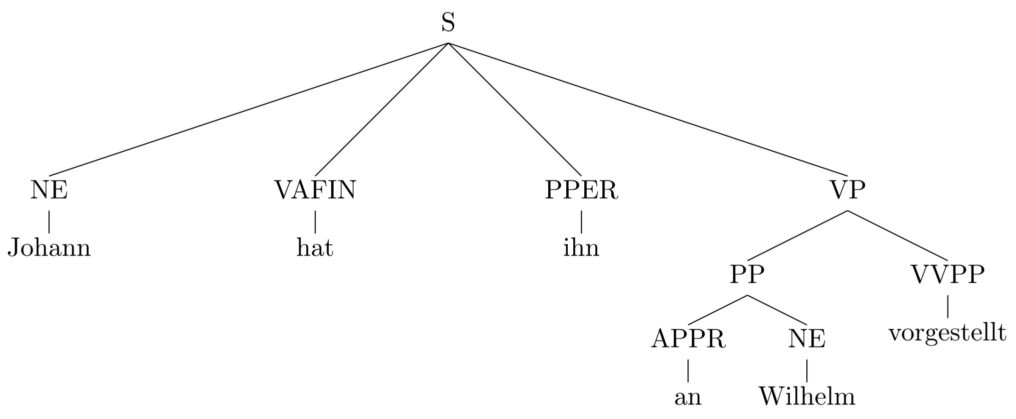
To test the cross-linguistic capability of our algorithm, we began by developing a corresponding dataset. This dataset was constructed by manually translating sentences from the English dataset, adding configuration entries for all nouns and pronouns, and creating a gold data set. We then used the CoreNLP German language package to generate parses. To illustrate this process, we show the stages on sentence (30):

(30) John introduced him to Bill .

We translated this as (31):

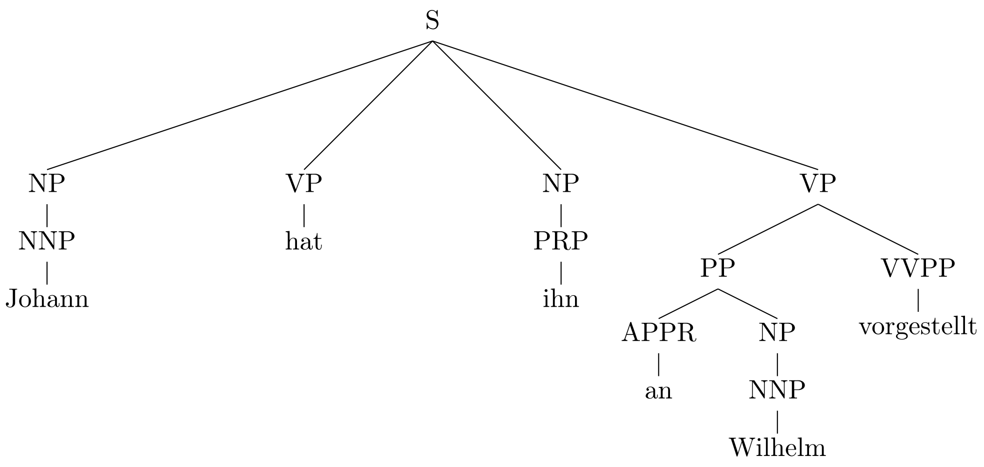
(31) Johann hat ihn an Wilhelm vorgestellt .

The direct output from CoreNLP is displayed in the tree below:



[15]Note that, depending on the parse of this sentence, this binding may or may not be licensed syntactically.

This output does not enable us to run our algorithm as implemented for the English language. The German POS tags differ from the English tags, so in order to recognize proper nouns and anaphors, we added a mapping between certain German POS tags and their English equivalents. The resulting parse tree from applying this mapping is shown below:



With this correction, we were able to use our existing algorithm for the German language data. The results of this analysis are discussed in 6.3.

6 Results

The dataset used to validate the deterministic, statistical, and neural models from CoreNLP as well as the syntactic algorithm consists of 90 sentences. These were drawn from the discussions of binding theory from [Cho93] and [LU88], as well as the examples utilized by [Hob78] to illustrate the capabilities of his algorithm. Sentences created by the experimenters similar to those in 2 were also included in the dataset. The validation data can be found in `gold_data.json`. Although evaluation of the performance of the syntactic algorithm on a larger dataset such as [Ce00] would yield greater insight, this was not done due to the lack of an adequate existing corpus with validated data that is consistent with the format of our system.

6.1 Results of Syntax

In Table 1 we present the accuracies attained by the base syntactic algorithm, the algorithm with each individual constraint applied, and the algorithm with all of the constraints included. For comparison, we also include the performance of each of the three CoreNLP models on the validation dataset discussed above.

Coreference Model	**Accuracy**
CoreNLP deterministic	0.569231
CoreNLP neural	0.600000
CoreNLP statistical	0.569231
Syntactic	0.430769
Syntactic with feature checking	0.476923
Syntactic with multi-token reflexives	0.476923
Syntactic with coordinating conjunctions	0.476923
Syntactic with multi-token names	0.430769
Syntactic without error correction	0.600000

Table 1: Coreference model performance for CoreNLP and Syntactic algorithms.

We observe that the base syntactic algorithm performs over 10% worse than the lowest-scoring CoreNLP model. When adding the selectional constraints and syntax error corrections, each individual constraint improves performance by only a few percentage points. Note that feature checking, multi-token reflexives, and coordinating conjunctions all result in the same accuracy when added individually. This is due to the nature of the dataset, where many of the sentences exhibit interactions between the aforementioned constraints. For instance,

(32) [John and Bill]$_i$ saw [each other]$_i$.

In order for our algorithm to correctly handle (32), we must include both the coordinating conjunctions and the multi-token reflexives. Thus, when evaluating our model with each constraint individually, there are many sentences that cannot be handled by only one of the constraints.

When the full algorithm is evaluated, with all of the constraints added, we achieve an accuracy identical to that of CoreNLP's neural net implementation. We take this to be a significant result, as our algorithm is significantly more efficient and is able to run without a large training dataset.

6.2 Parser-error Correction in CoreNLP

After evaluating the base syntactic algorithm, we then considered its ability to perform error correction on the output of the CoreNLP parser, and the impact of these corrections on the accuracy of anaphora resolution.

Our current algorithm corrects malformed genitive and ECM constructions. On the English dataset, syntax correction was performed on approximately a quarter of the dataset. Corrections for each construction individually, and both combined, are found in Table 2. While the

Syntactic Construction	Portion Corrected	Number Corrected
ECM constructions	0.202128	19
Genitive constructions	0.042553	4
All constructions	0.244680	23

Table 2: Syntax corrections performed for genitive and ECM constructions.

groups of sentences that undergo each type of syntax correction are discrete in this instance, this is not guaranteed to be true for all datasets. The accuracy after these corrections is given in Table 3.

Coreference Model	Accuracy
CoreNLP deterministic	0.569231
CoreNLP neural	0.600000
CoreNLP statistical	0.569231
Syntactic without error correction	0.600000
Syntactic with genitive constructions	0.646154
Syntactic with ECM constructions	0.646154
Syntactic with all constraints	0.692308

Table 3: Coreference model performance for CoreNLP and Syntactic algorithms with parser-error corrections.

While our system identifies many more ECM than genitive constructions, their relative corrections improve the accuracy by the same amount. This is in large part due to the fact that many of the ECM constructions have multiple nested clauses. Thus, while they do correct syntax errors, they do not have an impact on binding conditions. Consider sentence (33):

(33) [$_{S_1}$ John claimed that [$_{S_2}$ the picture of him$_i$ [$_{S_3}$ t_i hanging in the post office] is a fraud]].

The original parse includes "him" in embedded clause S_3. Our correction moves it to clause S_2, and leaves a trace t_i in its original position. However, because its only possible antecedent, John, is not contained in either S_2 or S_3, this does not have an impact on binding conditions. In contrast, all of the sentences that correct for bad genitive construction see changes to binding conditions.

The combination of these two approaches achieves an accuracy of 69.2308% percent, a full 9.2308% higher than CoreNLP's neural net accuracy. This not only implies that our syntax correction is valid, but also provides a compelling argument for the importance of syntax in anaphora resolution.

6.3 Cross-linguistic results

We tested a parallel dataset, using equivalent sentences in German, to evaluate our system's cross-linguistic performance. Our base algorithm achieves an accuracy of 0.307692. The additional improvements we made to the system, including syntax correction, did not improve our accuracy on this dataset.

We believe the lack of improvement from these changes is due to the fact that the German parse trees maintain a relatively flat structure with respect to the English ones. This lowers our ability to correctly apply binding conditions to our dataset. It is also worth noting that our algorithm did not identify any candidates for syntax correction in the dataset, which explains why these additions to the code do not affect its accuracy.

While the performance of our algorithm is not as high in German as it is in English, these results provide a good proof of concept for the cross-linguistic efficacy of our algorithm. Recall that the only change to the algorithm to achieve these results comes from a simple, static mapping of POS tags from German to English. We believe that, given more accurate parses for German sentences, our algorithm could perform comparably on German and English with little to no changes to the core architecture.

7 Conclusion

Our implementation, using a purely syntactic algorithm that applies binding conditions and selectional constraints, achieves the same performance on our given dataset as CoreNLP's neural net model. Unlike CoreNLP's code, which is fairly slow and relies on a training set to compute its results, our code is able to efficiently resolve anaphora with only a simple lexicon. This provides a strong argument for the use of linguistic theory, as opposed to pure machine learning, in performing anaphora resolution and other natural language processing tasks.

Not only are we able to match CoreNLP's performance under the given syntax, but our algorithm can correct for errors in its engine's parses. We can use anaphora resolution to argue for the validity of these corrections, as we see a clear increase in performance when we include syntax correction in our algorithm. Thus our system not outperforms CoreNLP on the given dataset, it is able to adapt to and correct for errors in this system.

Finally, we showed that the order of magnitude of our system's performance is consistent cross-linguistically, allowing us to exapand it to other languages with only trivial additions to a configuration file.

All three of these results support our initial goal, to demonstrate that anaphora resolution can be accurate cross-linguistically on correct syntax trees, and that, given malformed trees, the same algorithm can validate the improvements to these tress' syntax given an improvement in performance.

References

[Car13] Andrew Carnie. *Syntax: A generative introduction.* John Wiley & Sons, 2013.

[Ce00] Eugene Charniak et. al. WSJ corpus, 2000. BLLIP 1987-89.

[Cho93] Noam Chomsky. *Lectures on government and binding: The Pisa lectures.* Number 9. Walter de Gruyter, 1993.

[CM15] Kevin Clark and Christopher D Manning. Entity-centric coreference resolution with model stacking. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1405–1415, 2015.

[CM16] Kevin Clark and Christopher D Manning. Deep reinforcement learning for mention-ranking coreference models. *arXiv preprint arXiv:1609.08667*, 2016.

[Hae94] Liliane Haegeman. *Introduction to government and binding theory.* Wiley-Blackwell, 1994.

[Hir81] Graeme Hirst. Discourse-oriented anaphora resolution in natural language understanding: A review. *Computational Linguistics*, 7(2):85–98, 1981.

[Hob78] Jerry R Hobbs. Resolving pronoun references. *Lingua*, 44(4):311–338, 1978.

[Las98] Howard Lasnik. Exceptional case marking: Perspectives old and new. In *Proceedings of Formal Approaches to Slavic Linguistics*, volume 6, pages 187–211, 1998.

[LCP$^+$13] Heeyoung Lee, Angel Chang, Yves Peirsman, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Comput. Linguist.*, 39(4):885–916, December 2013.

[LL94] Shalom Lappin and Herbert J Leass. An algorithm for pronominal anaphora resolution. *Computational linguistics*, 20(4):535–561, 1994.

[LPC$^+$11] Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Proceedings of the fifteenth conference on computational natural language learning: Shared task*, pages 28–34. Association for Computational Linguistics, 2011.

[LU88] Howard Lasnik and Juan Uriagereka. *A Course in GB Syntax: Lectures on Binding and Empty Categories.* The MIT Press, 1988.

[MSB$^+$14] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[RLR$^+$10] Karthik Raghunathan, Heeyoung Lee, Sudarshan Rangarajan, Nathanael Chambers, Mihai Surdeanu, Dan Jurafsky, and Christopher Manning. A multi-pass sieve for coreference resolution. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 492–501. Association for Computational Linguistics, 2010.