SITA3012 Natural Language Processing

Unit 2

Syntactic Processing

Grammar for natural language - Toward efficient parsing - Ambiguity resolution - Statistical Methods, Feature Structure

1.1 Grammar for natural language

Grammar is defined as the rules for forming well-structured sentences. Grammar also plays an essential role in describing the syntactic structure of well-formed programs, like denoting the syntactical rules used for conversation in natural languages.

What is Syntactic Processing?

Syntactic processing is the process of analyzing the grammatical structure of a sentence to understand its meaning. This involves identifying the different parts of speech in a sentence, such as **nouns, verbs, adjectives,** and **adverbs,** and how they relate to each other in order to give proper meaning to the sentence. Let’s start with an example to understand **Syntactic Processing**:

- **New York is the capital of the United States of America.**
- **Is the United States of America the of New York capital.**

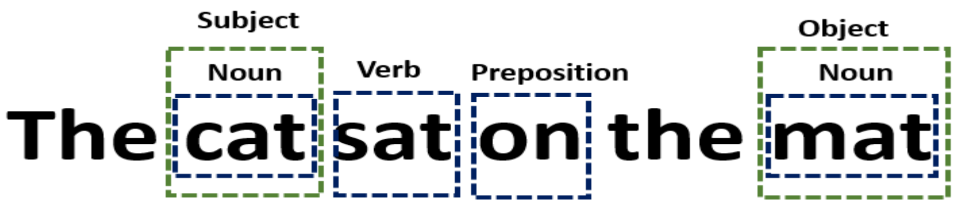
If we observe closely, both sentences have the same set of words, but only the first one is grammatically correct and which have proper meaning. If we approach both sentences with lexical processing techniques, we can’t tell the difference between the two sentences. Here, comes the role of syntactic processing techniques which can help to understand the relationship between individual words in the sentence.

Difference between Lexical Processing and Syntactic Processing

Lexical processing aims at data cleaning and feature extraction, by using techniques such as **lemmatization**, **removing stop words**, **correcting misspelled words**, etc. However, in **syntactic processing**, our aim is to understand the roles played by each of the words in the sentence, and the relationship among words and to parse the grammatical structure of sentences to understand the proper meaning of the sentence.

How Does Syntactic Processing Work?

To understand the working of syntactic processing, lets again start with an example. For example, consider the sentence "**The cat sat on the mat.**" Syntactic processing would involve identifying important components in the sentence such as "cat" as a **noun**, "sat" as a **verb**, "on" as a **preposition**, and "mat" as a **noun**. It would also involve understanding that "cat" is the **subject** of the sentence and "mat" is the **object**.



Syntactic processing involves a series of steps, including tokenization, part-of-speech tagging, parsing, and semantic analysis. **Tokenization** is the process of breaking up a sentence into individual words or tokens. **Part-of-speech (PoS) tagging** involves identifying the part of speech of each token. **Parsing** is the process of analyzing the grammatical structure of a sentence, including identifying the subject, verb, and object. The semantic analysis involves understanding the meaning of the sentence in context.

There are several different techniques used in syntactic processing, including rule-based methods, statistical methods, and machine learning algorithms. Each technique has its own strengths and weaknesses, and the choice of technique depends on the specific task and the available data.

Why is Syntactic Processing Important in NLP?

Syntactic processing is a crucial component of many NLP tasks, including **machine translation, sentiment analysis,** and **question-answering**. Without accurate syntactic processing, it is difficult for computers to understand the underlying meaning of human language.Syntactic processing also plays an important role in **text generation**, such as in chatbots or automated content creation.

Syntactic Structure and its Components:

Syntactic structure refers to the arrangement of words or phrases to form grammatically correct sentences in a language. It involves several components that organize and govern the way words come together to convey meaning. The fundamental components of syntactic structure include:

- Phrases: Phrases are groups of words functioning as a single unit within a sentence. They can be noun phrases (NP), verb phrases (VP), prepositional phrases (PP), etc.

- Words/Word Classes: Words are the basic building blocks of language. Different word classes (parts of speech) include nouns, verbs, adjectives, adverbs, prepositions, conjunctions, determiners, etc. Each word class has its own role and function within a sentence.

- Constituents: Constituents are smaller units within a sentence that form larger structures. For instance, in the sentence "The cat chased the mouse," "the cat" and "chased the mouse" are constituents that make up the larger sentence.

- Syntax Rules: These are the rules or principles that dictate the acceptable arrangement of words to form grammatically correct sentences in a language. They govern how words can combine to create phrases and sentences.

- Syntax Trees/Parse Trees: These graphical representations illustrate the hierarchical structure of a sentence. They show how different constituents (phrases) are nested within one another to form a complete sentence, with the main elements branching out into smaller units.

Syntax refers to the set of rules, principles, processes that govern the structure of sentences in a natural language. One basic description of *syntax* is how different words such as Subject, Verbs, Nouns, Noun Phrases, etc. are sequenced in a sentence. Some of the syntactic categories of a natural language are as follows:

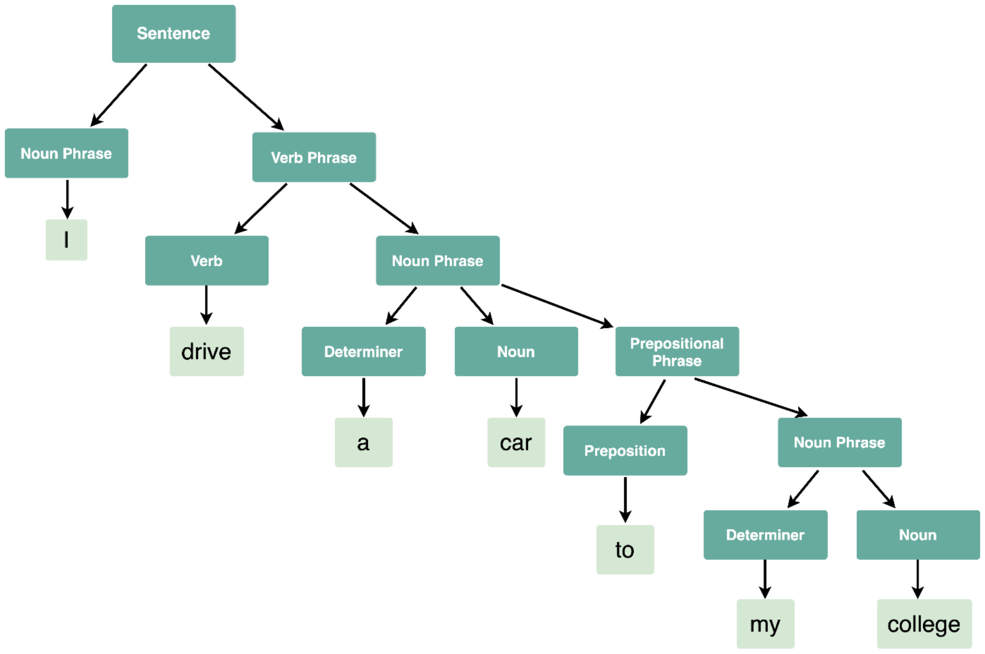
- Sentence(S)
- Noun Phrase(NP)
- Determiner(Det)
- Verb Phrase(VP)
- Prepositional Phrase(PP)
- Verb(V)
- Noun(N)

Syntax Tree:

A Syntax tree or a parse tree is a tree representation of different syntactic categories of a sentence. It helps us to understand the syntactical structure of a sentence.

Example: The syntax tree for the sentence given below is as follows:

I drive a car to my college.



Clauses: Clauses are units that contain a subject and a predicate. They can be independent (complete sentences) or dependent (incomplete sentences that rely on an independent clause to make complete sense).

2.2 Toward efficient parsing

According to Chomsky's grammatical hierarchy, encompasses different types of grammars, but the most relevant type for natural language processing (NLP) is the Context-Free Grammar (CFG), which falls under Type 2 in Chomsky's classification.

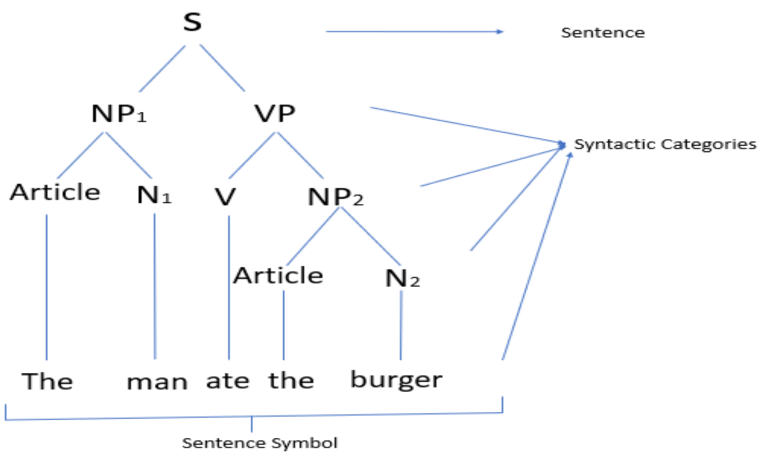
Context-Free Grammars (CFGs) have significant relevance in NLP for modeling the syntax or structure of natural languages. Here's how CFGs relate to natural language:

1. Syntax Modeling: CFGs are used to describe the syntax of natural languages by defining rules that specify how valid sentences can be formed from constituents like nouns, verbs, adjectives, etc. These grammars help capture the hierarchical structure of sentences in a language.
2. Phrase Structure: CFGs define the phrase structure of sentences, breaking them down into constituents such as noun phrases (NP), verb phrases (VP), prepositional phrases (PP), etc. These constituents are formed by recursive rules defined in the grammar.
3. Parsing: CFGs are crucial in parsing natural language sentences. Parsing involves analyzing the syntactic structure of sentences according to the rules specified in the grammar. Techniques like top-down and bottom-up parsing algorithms use CFGs to generate parse trees for sentences.
4. Formal Representation: CFGs formalize the rules governing the arrangement of words in a sentence. These rules dictate how words and phrases can be combined to form grammatically correct sentences.

Example to generate parse tree using grammars:

Sentence: The man ate the burger

Parse Tree:



The Sentence-to-sentence symbol is called as Top-down Parsing.

<s>	**→**	**<Np1> <VP>**
<NP1>	**→**	**<A><N1>**
<A>	**→**	**the**
<N1>	**→**	**man**
<VP>	**→**	**<V> <NP2>**
<V>	**→**	**ate**
<NP2>	**→**	**<A><N2>**
<A>	**→**	**the**
<N2>	**→**	**fruit**

The above rule is called as production rule. In natural language processing (NLP), a production rule, also known as a rewrite rule or a grammar rule, describes how symbols in a formal grammar can be replaced by other symbols or sequences of symbols. These rules define the structure and syntax of a language, providing guidelines for generating valid sentences or phrases.

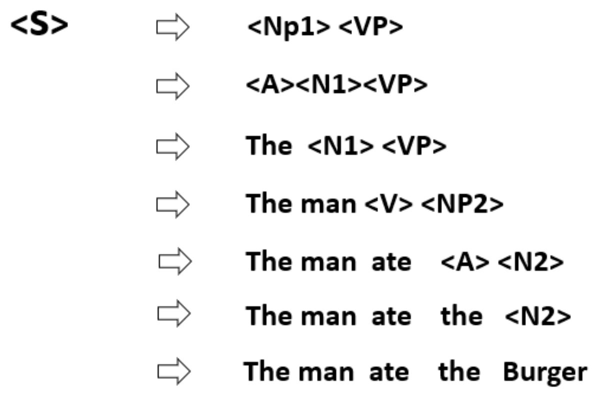
Formally, a production rule consists of two parts:

1. Left-hand side (LHS): This is the symbol or non-terminal on the left side of the rule. It represents a syntactic category or a symbol that can be expanded or replaced according to the rule.

2. Right-hand side (RHS): This is the sequence of symbols on the right side of the rule. It consists of terminals (actual words or symbols representing the basic units of the language) and/or non-terminals (symbols that can be further expanded by other production rules).

Bottom-up parsing

The essence of bottom-up parsing lies in starting with individual words or tokens and gradually constructing larger syntactic units by applying grammar rules until the entire input is successfully parsed.



```
<S>  ⇨  <Np1> <VP>
     ⇨  <A><N1><VP>
     ⇨  The  <N1> <VP>
     ⇨  The man <V> <NP2>
     ⇨  The man  ate   <A> <N2>
     ⇨  The man  ate   the   <N2>
     ⇨  The man  ate   the   Burger
```

Fig: Bottom up parsing

Efficient parsing in Natural Language Processing (NLP) is crucial for various language understanding tasks. Here are ways to achieve efficiency in parsing within NLP.

- Optimized Algorithms: Employ parsing algorithms tailored for NLP tasks. Techniques like transition-based parsing (e.g., Shift-Reduce parsers) or chart-based parsing (e.g., CYK) can efficiently parse sentences.
- Dependency Parsing: Utilize dependency parsers that focus on relationships between words rather than phrase structure. Dependency parsing often leads to faster and simpler parsing.
- Neural Network Models: Leverage neural network architectures for parsing, such as graph-based parsers or transformer models (e.g., BERT, GPT) that excel in handling contextual information and have shown efficiency in various parsing tasks.

- Incremental Parsing Models: Use models that allow for incremental parsing, enabling real-time analysis and faster processing of incoming language input.
- Domain-Specific Parsers: Develop parsers specifically tailored for certain domains or types of text. These parsers can focus on the specific linguistic patterns prevalent in those domains, leading to faster and more accurate parsing.
- Parallel Processing: Employ parallel computing techniques to process multiple sentences concurrently, speeding up parsing, especially in large-scale NLP tasks.
- Feature Engineering and Selection: Optimize feature sets used in parsing models to reduce computational overhead. Feature selection and dimensionality reduction techniques can streamline parsing without sacrificing accuracy.
- Language-Specific Optimizations: Implement language-specific optimizations that leverage the characteristics and structures inherent in certain languages. This includes language-specific rules or techniques that can expedite parsing.
- Incremental Model Updates: For applications where the parsing model needs to adapt to new data continuously, incremental learning techniques can be employed to update the model efficiently without retraining from scratch.
- Hybrid Approaches: Combine different parsing techniques or models to take advantage of the strengths of each, creating hybrid systems that are both efficient and accurate.

2.3 Ambiguity resolution

Ambiguity resolution in natural language processing (NLP) refers to the process of disambiguating or resolving multiple possible meanings or interpretations within a given context. Ambiguity is inherent in language due to the richness and complexity of human communication. Resolving ambiguity is crucial for NLP tasks like machine translation, sentiment analysis, and speech recognition to ensure accurate understanding and processing of text.

Here are some common types of ambiguity in NLP and methods used for resolution:

1. Lexical Ambiguity: Words with multiple meanings (homonyms, polysemes) can lead to ambiguity. Contextual information is often used to disambiguate. Techniques like part-of-speech tagging, word sense disambiguation (WSD), and using contextual embeddings (like BERT, GPT) help in understanding the intended meaning.

2. Syntactic Ambiguity: Ambiguity arising from the structure of sentences. For instance, "I saw the man with the telescope." (Did the man have the telescope or was the speaker using a telescope?) Syntax parsing, tree structures, and probabilistic models aid in resolving such ambiguities.
3. Referential Ambiguity: Occurs when pronouns or references lack clarity regarding what they refer to. Co-reference resolution is used to link pronouns or noun phrases to their correct antecedents within the text.
4. Semantic Ambiguity: Arises due to multiple interpretations of the overall meaning of a sentence or phrase. Knowledge graphs, semantic role labeling, and deep learning models that consider broader contexts help disambiguate such cases.
5. Anaphoric Ambiguity: Similar to referential ambiguity, involving interpretations of anaphors (expressions that refer back to another word). Resolving this type of ambiguity involves understanding the relationships between various linguistic elements.

NLP models leverage various techniques, from rule-based approaches to statistical methods and deep learning models, to resolve ambiguity. Contextual information, semantic understanding, syntactic analysis, and large pre-trained language models have significantly improved ambiguity resolution in NLP tasks.

2.3.1 Ambiguity resolution – Statistical Model

Statistical methods play a significant role in solving ambiguity in natural language processing (NLP). They use probabilities and patterns derived from large datasets to disambiguate various linguistic ambiguities. Here are some statistical approaches commonly used:

1. Word Sense Disambiguation (WSD): Statistical methods utilize corpus-based frequencies to determine the most probable sense of a word in a given context. Techniques like Lesk algorithm, supervised learning models (Naive Bayes, Support Vector Machines), and more recently, neural network-based approaches leverage contextual information to disambiguate word senses.

Consider the word "bank," which can have multiple meanings: a financial institution or the side of a river.

Example Sentence: "I went to the bank to deposit my paycheck."

In this sentence, without context, it's unclear if "bank" refers to a financial institution or the edge of a river. Statistical methods for WSD analyze the surrounding words and their frequencies in a large corpus to determine the most probable sense. For instance, if the word "money," "ATM," or "tell" frequently co-occurs with "bank" in a specific context in the training data, the statistical model might lean toward the financial institution sense.

LESK algorithm

The LESK algorithm is a computational approach used in natural language processing and information retrieval to disambiguate the meaning of words in context. It was developed by Michael E. Lesk in 1986. The primary goal of the LESK algorithm is to determine the most appropriate sense of a word by comparing the context in which it appears with the definitions of the word in a lexical database.

Overview of how the LESK algorithm works:

i. **Gather Context:**

   a. Collect the surrounding words (context) of the target word that needs to be disambiguated. The context usually consists of words within a certain window around the target word.

ii. **Retrieve Definitions:**

   a. Retrieve the definitions of the ambiguous word from a lexical database, such as WordNet. WordNet is a widely used lexical database that organizes words into sets of synonyms, called synsets, and provides definitions for each synset.

iii. **Overlap Calculation:**

   a. Calculate the overlap between the words in the context and the words in each definition of the ambiguous word. The idea is that the sense of the word with the highest overlap with the context is likely to be the correct sense.

iv. **Sense Selection:**

   - Select the sense with the highest overlap as the disambiguated sense for the ambiguous word.

2. Probabilistic Parsing: Statistical parsing techniques assign probabilities to different parse trees based on training data. This helps in resolving syntactic ambiguity by selecting the most probable parse for a given sentence. Probabilistic context-free grammars (PCFGs) and statistical dependency parsers are examples of such approaches.

Example Sentence: "Time flies like an arrow."

This sentence has multiple possible interpretations in terms of syntax. Probabilistic parsing assigns probabilities to various ways of parsing the sentence based on training data. It might assign a higher probability to a tree where "like an arrow" is a modifier for "flies" rather than for "time," based on the statistical frequency of such structures in the training corpus.

3. N-gram Models: These models calculate the probability of a word given its preceding context of 'n' words. They are used in language modeling and can help in disambiguating based on the likelihood of certain word sequences occurring together.

Example Text:

"Mary had a little lamb. Its fleece was white as snow."

Creating N-Grams:

Unigrams (1-gram):

Unigrams consist of single words in the text.

- Unigrams: [Mary, had, a, little, lamb, Its, fleece, was, white, as, snow]

Bigrams (2-gram):

Bigrams are pairs of consecutive words.

- Bigrams: [(Mary, had), (had, a), (a, little), (little, lamb), (lamb, Its), (Its, fleece), (fleece, was), (was, white), (white, as), (as, snow)]

Trigrams (3-gram):

Trigrams are sequences of three consecutive words.

- Trigrams: [(Mary, had, a), (had, a, little), (a, little, lamb), (little, lamb, Its), (lamb, Its, fleece), (Its, fleece, was), (fleece, was, white), (was, white, as), (white, as, snow)]

Using N-Gram Models:

Suppose we're using a bigram model to predict the next word in the sequence "Its fleece ____."

- From the bigrams in the text, we see that "Its fleece" is followed by "was" and "was" is followed by "white."
- Using the bigram probabilities, we predict that the most likely next word after "Its fleece" is "was."

Calculating Probabilities:

N-gram models calculate probabilities based on the frequency of occurrences of these sequences in a given text. For instance, in a bigram model:

- P(w | w_1) = Count(w_1, w) / Count(w_1)

Where:

- P(w | w_1) is the probability of word w given the preceding word w_1.
- Count(w_1, w) is the count of the bigram (w_1, w).
- Count(w_1) is the count of occurrences of the word w_1.

4. Statistical Machine Translation (SMT): Ambiguity often arises in translation tasks. Statistical approaches in machine translation use large bilingual corpora to determine the most likely translation by analyzing word and phrase probabilities within the given context.

In translation, words and phrases can have multiple possible translations depending on context.
Example: Translating "I saw her duck" from English to Spanish.
The word "duck" can be a verb or a noun. Statistical machine translation models, trained on bilingual corpora, use context and probabilities to determine the most likely translation. If the statistical model sees that "duck" as a noun is often translated to a Spanish word for a bird, while as a verb, it's translated to a word for avoiding something, it will choose the translation based on the statistical likelihood within the context of the sentence.

5. Statistical Semantic Analysis: Methods like distributional semantics use statistical patterns derived from large text corpora to represent word meanings and contexts. This aids in disambiguating semantic ambiguities by identifying words' meanings based on their usage patterns in the corpus.

Example : Consider a corpus where the sentences "The cat sat on the mat" and "The dog lay on the rug" are present.

- Statistical semantic analysis would analyze the co-occurrence patterns of words. It might observe that "cat" and "dog" occur in similar contexts (with words like "on" and "the"), suggesting a degree of semantic similarity.
- Using word embeddings, words like "cat" and "dog" would have vector representations that are closer together in the vector space compared to less related words like "table" or "computer."

6. Named Entity Recognition (NER): Statistical models are employed to identify and classify named entities (such as names of people, organizations, locations) in text. These models use statistical patterns to recognize entities based on context and linguistic features.

Example: "Apple Inc. was founded by Steve Jobs, Steve Wozniak, and Ronald Wayne on April 1, 1976, in Cupertino, California."
In this sentence, NER would identify and classify different types of entities:

- Organization: "Apple Inc."
- Persons: "Steve Jobs," "Steve Wozniak," and "Ronald Wayne"

- Date: "April 1, 1976"
- Location: "Cupertino, California"

7. Co-reference Resolution: Statistical methods analyze patterns in text to determine relationships between pronouns and their antecedents, resolving referential ambiguity by identifying which nouns or phrases pronouns refer to.

Example:
"Sarah went to the park. She enjoyed the sunshine. The girl had a picnic with her friends."
In this text, coreference resolution aims to link the pronouns ('she', 'The girl', 'her') to their antecedent ('Sarah') to understand that they all refer to the same person.

Statistical approaches in NLP often rely on annotated or labeled data for training models. These methods leverage the statistical properties of language use in corpora to infer the most likely interpretation or meaning of ambiguous elements within a given context. While statistical techniques have been foundational in NLP, recent advancements in deep learning have also made significant strides in addressing ambiguity through neural network-based models that can capture complex linguistic patterns and nuances more effectively.

2.4 Feature Structure

feature structure is defined as a mapping from features to values that defines the relevant properties of the constituent. A feature structure for a constituent ART1 that represents a particular use of the word a might be written as follows:

ART1: (CAT ART **ROOT** a **NUMBER** s**)**

This says it is a constituent in the category ART that has as its root the word a and is singular. Usually an abbreviation is used that gives the CAT value more prominence and provides an intuitive tie back to simple context-free grammars. In this abbreviated form, constituent ART1 would be written as:

ART1: (ART **ROOT** a **NUMBER** s**)**

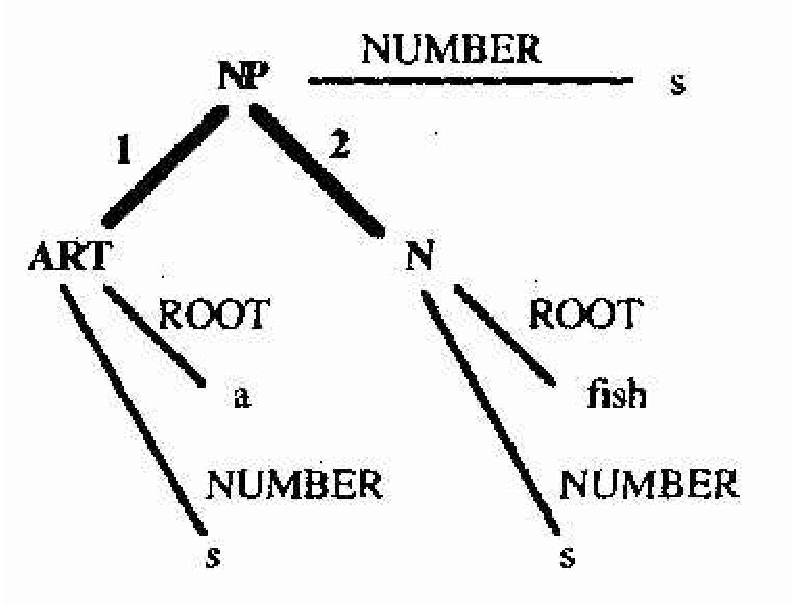
Feature structures can be used to represent larger constituents as well. To do this, feature structures themselves can occur as values. Special features based on the integers - 1, 2, 3, and so on - will stand for the first subconstituent, second subconstituent, and so on, as needed. With this, the representation of the NP constituent for the phrase "*a fish*" could be

NP1: (NP NUMBERs 1 (ART **ROOT** a **NUMBER** s**) 2 (N ROOT** fish **NUMBER** s**))**

The rules in an augmented grammar are stated in terms of feature structures rather than simple categories. Variables are allowed as feature values so that a rule can apply to a wide range of situations.
For example, a rule for simple noun phrases would be as follows:

(NP **NUMBER** ?n**) - (**ART **NUMBER** ?n**) (**N **NUMBER** ?n**)**

This says that an NP constituent can consist of two subconstituents, the first being an ART and the second being an N, in which the NUMBER feature in all three constituents is identical. According to this rule, constituent NP1 given previously is a legal constituent.



Viewing a feature structure as an extended parse tree

The constituent (NP **1** (ART **NUMBER** s) **2** (N **NUMBER** s))
is not allowed by this rule because there is no NUMBER feature in the NP, and the constituent

*(NP **NUMBER** s 1 (ART **NUMBER** s) 2 (N **NUMBER** p))
is not allowed because the NUMBER feature of the N constituent is not identical to the other two NUMBER features.

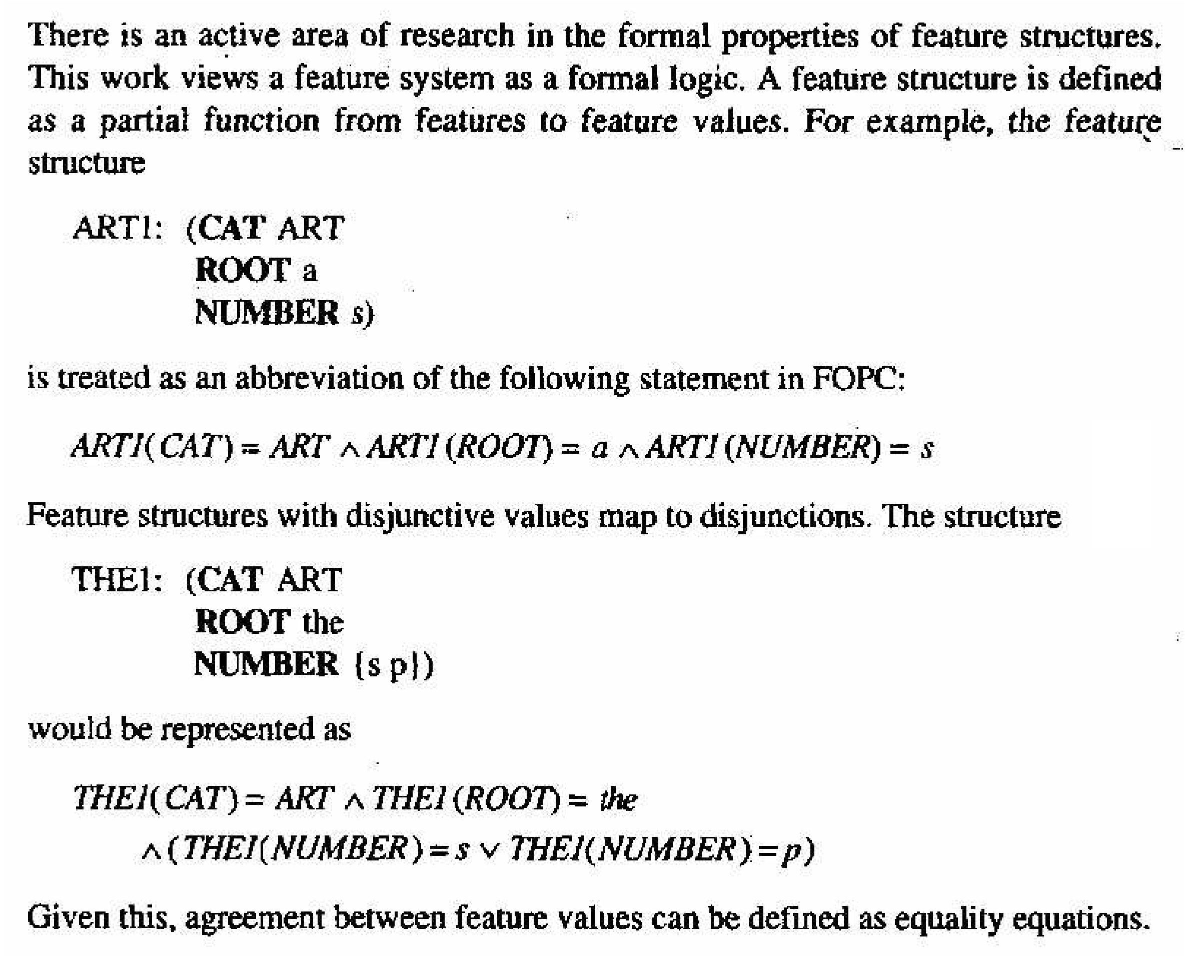
Variables are also useful in specifying ambiguity in a constituent. For instance, the word fish is ambiguous between a singular and a plural reading. Thus the word might have two entries in the lexicon that differ only by the value of the NUMBER feature. Alternatively, we could define a single entry that uses a variable as the value of the NUMBER feature, that is,

(N **ROOT** fish **NUMBER** ?n)

This works because any value of the NUMBER feature is allowed for the word fish. In many cases, however, not just any value would work, but a range of values is possible. To handle these cases, we introduce constrained variables, which are variables that can only take a value out of a specified list. For example, the variable ?n{s p} would be a variable that can take the value s or the value p. The word fish might be represented by the constituent

(N **ROOT** fish **NUMBER** ?n{sp})
or more simply as
(N **ROOT** fish **NUMBER** {s p})

<u>Formalizing Feature Structure</u>



There is an active area of research in the formal properties of feature structures. This work views a feature system as a formal logic. A feature structure is defined as a partial function from features to feature values. For example, the feature structure

```
ART1: (CAT ART
       ROOT a
       NUMBER s)
```

is treated as an abbreviation of the following statement in FOPC:

$$ART1(CAT) = ART \land ART1(ROOT) = a \land ART1(NUMBER) = s$$

Feature structures with disjunctive values map to disjunctions. The structure

```
THE1: (CAT ART
       ROOT the
       NUMBER {s p})
```

would be represented as

$$THE1(CAT) = ART \land THE1(ROOT) = the \land (THE1(NUMBER) = s \lor THE1(NUMBER) = p)$$

Given this, agreement between feature values can be defined as equality equations.

If the set of feature values is finite, then it would always be possible to create new constituent categories for every combination of features. Thus it is expressively equivalent to a context-free grammar. If the set of feature values is unconstrained, however, then such grammars have arbitrary computational power. In practice, even when the set of values is not explicitly restricted, this power is not used, and the standard parsing algorithms can be used on grammars that include features.

Questions to Revise:

1. Ambiguity resolution methods
2. Parsing - top down parsing and bottom up parsing – create a parse tree
3. Syntactic processing - components and applications – linguistic ambiguity
4. Syntactic processing - linguistic ambiguity – feature structures
5. Statistical Models – to solve ambiguity