Lecture 6

Dependency parsing

By HKT & Stable Chat and Krystaleah Ramkissoon

Contents

Dependency parsing is an NLP technique. We will cover the uses of parsing, followed by a discussion on parse trees, shallow parsing, dependency structure, and the motivation for dependency parsing. Next, we will delve into dependency grammar, transition-based dependency parsing, and evaluation of dependency parsers.

Uses of Parsing

There are two kinds of parsing in NLP: dependency and constituency parsing. *Dependency parsing* focuses on understanding the grammatical structure of a sentence and is commonly used in machine translation, sentiment analysis, and information extraction. In constituency parsing, sentences are represented as parse trees, where each node corresponds to a phrase, and edges represent parent-child relationships. *Constituency parsing* is used in syntactic analysis, grammar checking, and text generation. Both dependency parsing and constituency parsing are applied in co-reference resolution, which identifies expressions, such as pronouns, that refer to the same entity.

Parsing is also used in sentiment analysis to analyze the syntactic structure of a sentence, identifying the sentiment-bearing parts and improving aspect-based sentiment analysis. In question answering, another NLP application, parsing aids in recognizing relevant entities and relationships needed to retrieve the correct answer. Finally, in information extraction, parsing can extract structured information from unstructured text, including subject-verb-object relationships that are crucial when extracting events or facts.

When To Use Dependency Parsing?

When it comes to dependency parsing, it is primarily utilized for enhancing the performance of downstream tasks. It serves as a supportive component rather than a standalone application. Prior to 2015, parsing was commonly incorporated into NLP pipelines. However, with the advent of end-to-end models, dependency parsing is now more commonly used to enrich text representation, for example, by tagging a text before vectorization or as an additional feature vector.

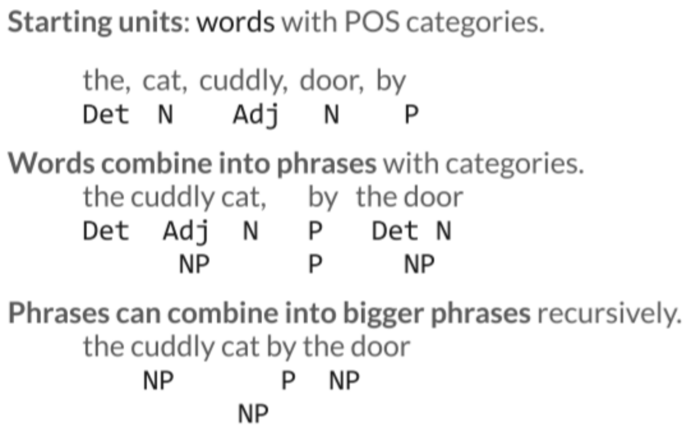
An effective dependency parsing model necessitates extensive training on large datasets. Consequently, a custom model trained on a smaller dataset can benefit from leveraging the valuable information gained from larger datasets. This can be seen as transferring knowledge from large datasets to smaller models, thus resembling transfer learning (although technically distinct).

Parse Trees

Language Is Recursive

Consider a set of words, such as "the", "cat", "cuddly", "door", and "by". Each word has a corresponding POS: "the" is a determiner, "cat" is a noun, "cuddly" is an adjective, "door" is another noun, and "by" is a preposition. These words can be combined to form phrases with specific categories.

For instance, we can create the phrase "the cuddly cat" by combining the determinant "the", the adjective "cuddly," and the noun "cat" to form a noun phrase. Similarly, we can create another phrase, "by the door," by combining the preposition "by" with the noun "door." These two phrases can be combined further to create a larger noun phrase, such as "the cuddly cat by the door." This demonstrates the recursive nature of language as we can continue building larger and larger phrases using the same rule repeatedly.



Starting units: **words** with POS categories.

```
the, cat, cuddly, door, by
Det  N    Adj     N     P
```

Words combine into phrases with categories.

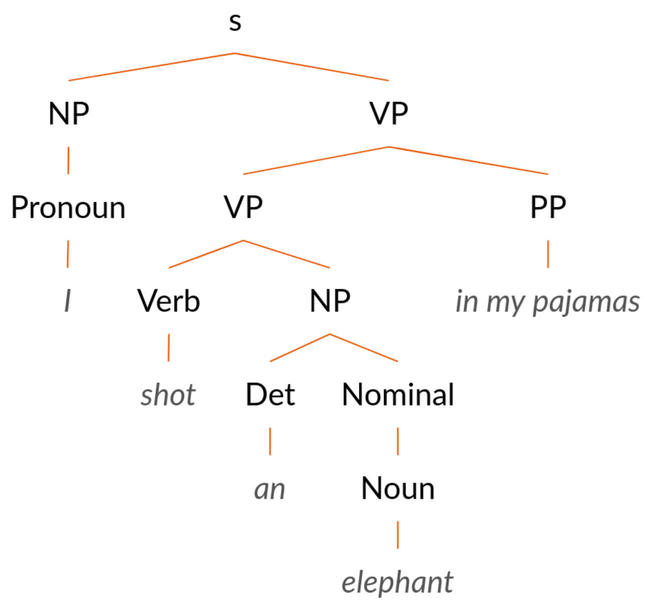
```
the cuddly cat,   by  the door
Det  Adj  N       P   Det N
     NP           P    NP
```

Phrases can combine into bigger phrases recursively.

```
the cuddly cat by the door
    NP         P  NP
          NP
```

Parse Trees

Parse trees are a way to represent the grammatical structure of sentences by breaking them down into their constituent phrases. Consider the sentence, "I shot an elephant in my pajamas." At the first level, individual words have their respective parts of speech: "I" is a pronoun, "shot" is a verb, and "an" is a determiner. "Elephant" is a noun, and "in my pajamas" is a preposition followed by a noun phrase where "my" is a pronoun, and "pajamas" is a noun.



From Fig 13.2 in Speech & Language Processing (2020)

By combining these constituents, we can construct a parse tree: "I" creates a noun phrase, "shot an elephant" is a verb phrase consisting of a verb and the noun phrase "an elephant." This verb phrase can be combined with the preposition phrase "in my pajamas" to form another verb phrase, "shot an elephant in my pajamas." Finally, the noun phrase "I" can be combined with the verb phrase to create a complete sentence, "I shot an elephant in my pajamas." In this manner, we have parsed the sentence into a parse tree, representing its grammatical structure.

Shallow Parsing

Shallow parsing, also known as chunking, involves grouping adjacent tokens into phrases based on their POS tags. Well-known chunks are noun, verb and prepositional phrases.

Noun phrases consist of at least two words with a noun, pronoun, or determiner as the head and may include modifiers. They function as a noun in a sentence. Examples of the form **modifier** noun:

> **the** man
> **a** girl
> **the** doggy **in the window**.

arguments verb:

> She **can** smell the pizza
> He **has** appeared on screen as an actor.

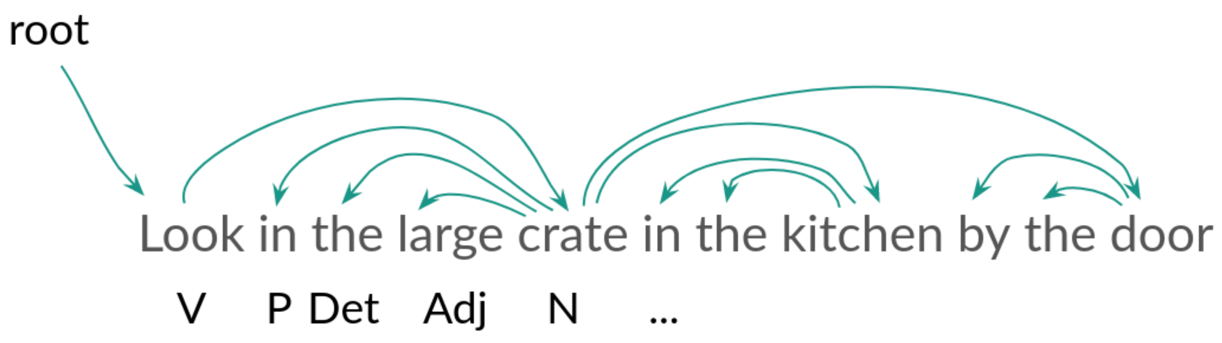
Verb phrases, on the other hand, consist of at least two words with a verb as the head and arguments that further illustrate the verb's tense, action, or tone. Examples of verb phrases include "can smell" in the sentence "She can smell the pizza" and "has appeared" in the sentence "He has appeared on screen as an actor."

Shallow parsing allows us to break down sentences into noun and verb phrases to better understand their grammatical structure.

Dependency Parsing

Dependency Structure

Dependency structure is a representation of how words in a sentence are related to one another, specifically, which words modify or act as arguments for other words. For example, in the sentence, "Look in the large crate in the kitchen by the door," we can identify a root word (e.g., "look"), words that depend on the root (e.g., "crate"), words in turn, that depend on those words (e.g. “door”) and so on.



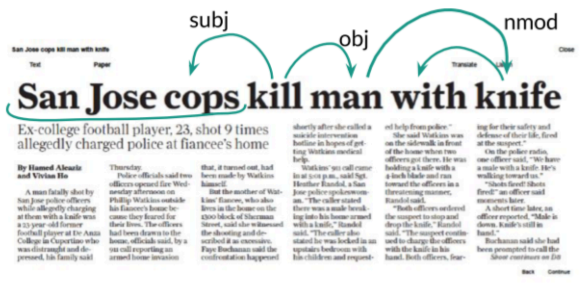
This dependency structure highlights the relationships between words in a sentence and helps in understanding the sentence's grammatical structure.

Why Do We Need Dependency Parsing?

Dependency parsing is essential in NLP as it helps to understand the structure of sentences and the relationships between words. Sentence structure plays a crucial role in conveying complex meanings through language and, therefore, is vital for many NLP tasks. Knowing how words are connected and function in relation to one another is the core concept of dependency parsing.

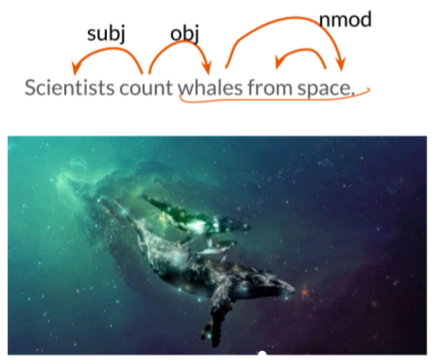
2 alternative meanings.

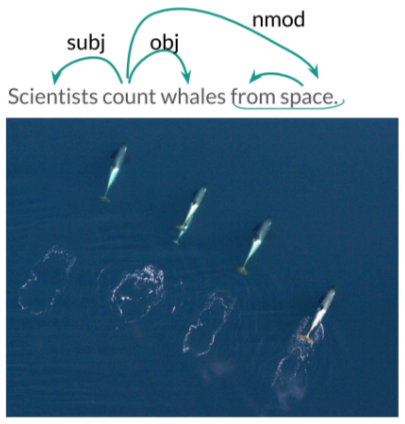
- The man who was killed had a knife.
- Cops lethally stab a man.


subj
obj
nmod
San Jose cops kill man with knife
Ex-college football player, 23, shot 9 times allegedly charged police at fiancee's home

subj	- subject
obj	- object
nmod	- nominal modifier

Understanding sentence structure is important for interpreting language accurately and avoiding ambiguity. Correct parsing can sometimes prevent misinterpretation, as shown in the example headline, "San Jose Cops kill man with knife." This headline is ambiguous and can be interpreted in two ways: the man who was killed had a knife or the cops used a knife to lethally stab a man. Another example is "scientists count whales from space," which can be misinterpreted as whales coming from space or scientists counting them from space using satellite images.





Dependency Grammar

In the field of NLP, dependency grammar is a core concept that focuses on the relationships between words in a sentence, often represented as binary asymmetric connections called dependencies (aka arrows). These relationships are typically labelled by grammatical roles like subject, determiner, prepositional object, and others. Dependency grammar has a long history, with roots tracing back to Pāṇini, a logician, Sanskrit philologist, grammarian, and revered scholar in ancient India. He lived sometime between the 6th and 4th century BCE and wrote his texts on birch bark.

The earliest known NLP dependency parser was developed in 1962 by David Hays.

Universal dependency relations are defined on resources such as the Universal Dependencies website and the Stanford Typed Dependencies manual. Some common grammatical relations in dependency parsing are shown in the table below. For complete lists consult Universal Dependencies or the Stanford Typed Dependencies manual.

Causal argument relation	**Description**
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object

CCOMP	Clausal complement
Nominal modifier relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions & other case markers
Other notable relations	**Description**
CONJ	Conjunction
CC	Coordinating conjunction

Relations are represented by abbreviations for easy identification and understanding.

Building a Dependency Parser

To build a dependency parser, we data annotated for supervised learning, for example, from existing treebanks. One valuable resource for dependency parsing is the Universal Dependencies database, which contains sentences labelled with dependencies and their types across more than 100 languages.

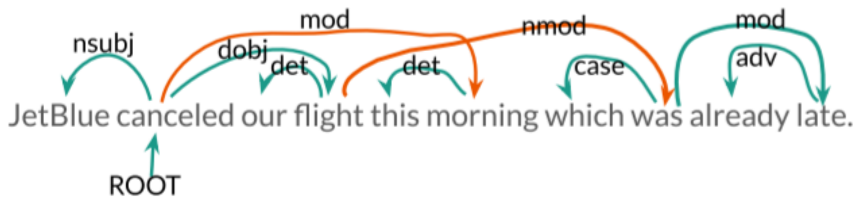
Using a pre-existing data set like Universal Dependencies provides several advantages. First, it saves us the effort of labelling the data ourselves. Second, it can be used for multiple parsers, allowing for reusability of labour. Third, it provides broad and systematic coverage of real-world language use, making it useful for statistical analysis and linguistic research. Lastly, it serves as a basis for evaluating parsers by benchmarking performance on a standardized data set.

By using the Universal Dependencies database or similar resources, we can effectively train our dependency parsers and measure their accuracy compared to other models. This aids in continuous improvement and ensures our parsers are performing optimally on real-world language data.

The Basics of a Dependency Parser

Dependency parsing involves analyzing sentences and determining the relationships between words, often represented as a tree, aka a directed acyclic graph (DAG). To create a dependency structure, we decide for each word which other words it is dependent on, following certain constraints.

- Only one word can be independent of the root, ensuring a clear hierarchical structure.
- There must be no cycles within the dependency graph, meaning that if B is dependent on A, A cannot be dependent on B. If a dependency structure meets these constraints, it forms a tree without any cycles, i.e. a DAG.
- A more optional constraint is that arrows cannot cross. In these cases, the dependency tree is referred to as non-projective.


nsubj
mod
dobj
det
det
nmod
case
mod
adv
JetBlue canceled our flight this morning which was already late.
ROOT

Types of dependency parsers

There are various methods for dependency parsing, which can be broadly categorized into four groups.

1. **Dynamic programming**: Eisner (1996) gives a clever algorithm with complexity $O(n^3)$, by producing parse items with heads at the ends rather than in the middle.
2. **Graph algorithms**: In this category, minimum spanning trees are built for sentences. An example of this method is McDonald and colleagues (2005) MSTParser which scores dependencies independently using an ML classifier.
3. **Constraint satisfaction**: Here, edges that don't satisfy hard constraints are eliminated. An example of such a dependency parser is one developed by Karlsson in 1990.
4. **Transition-based parsing**: This type of parser selects dependencies greedily guided by ML classifiers. For example, the MaltParser by Nivre and others (2008) is a popular and effective transition-based parser.

Now, let's focus on transition-based parsing, an efficient method for building dependency parsers in NLP.

Transition-based Parsing

Transition-based parsing, introduced by Jochem Nivre in 2008, is a simple and greedy discriminative dependency parser. This parsing method involves a sequence of actions performed by the parser to analyze the input sequence. The parser consists of four components:

1. A stack initially containing only the root node.
2. A buffer initially containing the input sequence.

3. A set of dependency arcs (A) which is initially empty.
4. A set of actions available to the parser.

The Arc-standard Parser

In this simplified model, the parser has three available actions:

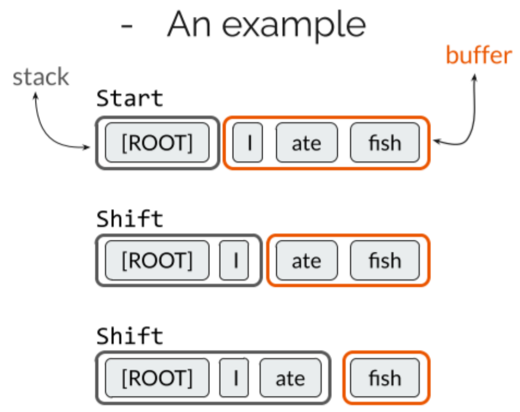
1. `Left_arc`: Asserts a head-dependent relationship between the word at the top of the stack and the word directly beneath it, removing the lower word from the stack.
2. `Right_arc`: Asserts a head-dependent relationship between the second word of the stack and the word at the top, removing the word at the top.
3. `Shift`: Removes the word from the front of the input buffer and puts it onto the stack.

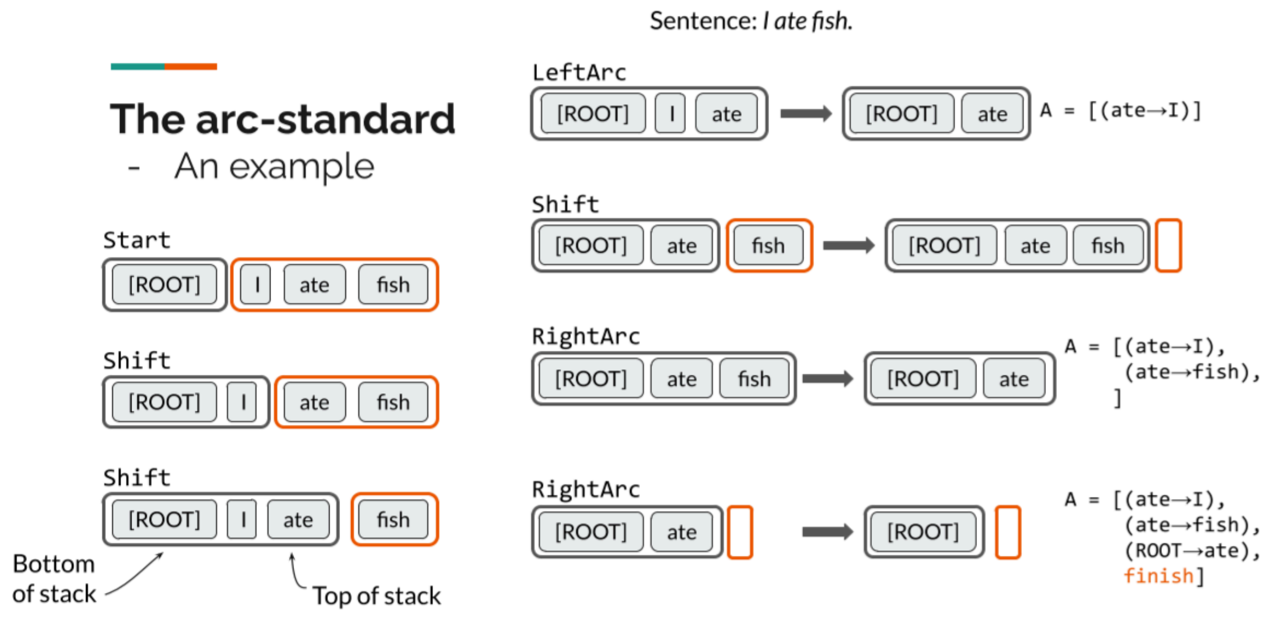
How to Select The Action

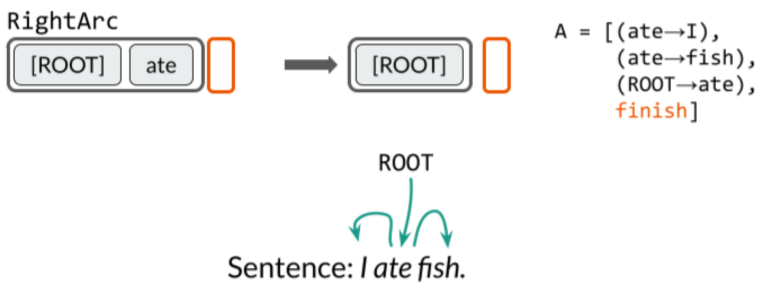
In a transition-based dependency parser, the correct action is chosen using an "oracle." The oracle is often an ML classifier trained to predict the actions needed in a particular scenario. In the arc-standard, the three actions are `left_arc`, `right_arc`, and `shift`.

Lycurgus Consulting the oracle in Delphi (*Pythia*). Image credits: By Eugène Delacroix, Public Domain, https://commons.wikimedia.org/w/index.php?curid=29190439.

The inputs to the oracle are features derived from the sentence being parsed. The features are often created through feature engineering or, more recently, with deep learning methods and pre-trained language models such as BERT or RoBERTa. Earlier approaches involved more manually created features, while modern parsers take advantage of powerful neural networks and embeddings to represent linguistic phenomena. For instance, the early MaltParser employed sparse, hand-coded features, while its successor, Chen and Manning's parser from 2014, relied on deep learning-based embeddings as features. Nowadays, state-of-the-art parsers use advanced deep learning techniques, such as transformer networks, to achieve superior performance in parsing and understanding natural language.

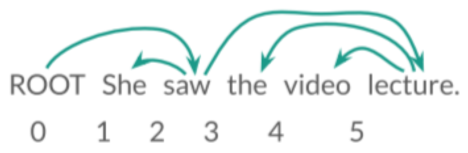






Evaluating a Dependency Parser

Dependency parsers are evaluated using two scores, the unlabelled attachment score (UAS) and the labeled attachment score (LAS). Both rely on a labelled dataset for their calculation.



Unlabelled Attachment Score

Evaluating a dependency parser involves the use of the UAS. In this case, the focus is on the structure of the dependency tree and not the labels associated with the arrows. The evaluation process starts by having ground truth data, which consists of a sentence with its corresponding dependency arrows. Next, we have the dependency parser generate a predicted set of arrows for the same sentence.

Gold (i.e. ground truth)

to	from	word	type
1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	obj

Parsed (i.e. predicted)

to	from	word	type
1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

$$Acc = \frac{\#true\ dependencies}{\#dependencies}$$

$$UAS = \frac{4}{5} = 80\%$$

To calculate the unlabeled attachment score, we count the number of correctly predicted dependencies and divide it by the total number of dependencies in the ground truth data. The unlabeled attachment score approach bears similarities to normal classification accuracy in evaluating the performance of the dependency parser.

Labelled attachment score

Evaluating a dependency parser using the LAS involves not only considering the dependency arrows but also the specific type of dependency present, such as a subject, root, determiner, or causal complement. To assess the performance of the parser, we start by having ground truth data that includes both the dependency arrows and the corresponding types.

Gold (i.e. ground truth)

to	from	word	type
1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	obj

Parsed (i.e. predicted)

to	from	word	type
1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

$$Acc = \frac{\#true\ dependencies}{\#dependencies}$$

$$UAS = \frac{4}{5} = 80\%$$

$$LAS = \frac{2}{5} = 40\%$$

We then compare this ground truth to the dependency parser's predictions, counting the number of correctly identified dependencies – both in terms of the arrow direction and the type of dependency. Finally, we divide this number by the total dependencies in the ground truth data to obtain the labelled attachment score. This metric provides a more comprehensive assessment of the parser's accuracy in predicting both the dependencies and their respective types.

State of The Art

State-of-the-art dependency parsers have achieved impressive results in recent years, as evidenced by the Penn Treebank task on the NLP Progress website. As of 2023, the best dependency parser attained a UAS of 97.42% and a LAS of 96.26%, approaching the ceiling performance.

These top-performing parsers typically feature large, deep neural networks and have undergone significant hyperparameter tuning to optimize their performance. For example, the current SOTA (2023-10-23) uses XLNet a transformer-based neural network architecture. In addition, instead of relying on greedy search for action selection, these advanced parsers leverage beam search. Furthermore, many of them integrate self-attention mechanisms within their networks, a technique that will be discussed in further detail in upcoming lectures.

Terminology

- Parsing
- Constituency parsing
- Dependency parsing
- Parse tree
- Shallow parsing
- Chunking
- Noun phrase
- Verb phrase

- Dependency grammar
- Dependencies
- Grammatical role
- Dependency parser
- Transition-based parser
- The arc-standard parser
- Oracle
- LAS
- UAS