# Natural Language Processing

## 6: Syntax. Trees and Dependencies. Part 1

# Objectives

**to be able to answer questions**

(1) What is syntax, grammar?

(2) What are most common Syntactic Representations?

(3) How grammars work?

(4) How phrase structure grammar work?

(5) What are the Phrasal categories, why are they useful?

# Basic Background: Syntax, Grammar


SYNTAX?
I ALREADY PAY THE GOVERNMENT ENOUGH!
imgflip.com

# Grammaticality

(1) We recognise some word strings are well-formed or grammatical sentences, and others not, e.g.

(a) The dog bit the cat.
(b) *The the bit cat dog.
* means ungrammatical

(2)

A sentence may be grammatical but **not acceptable**.

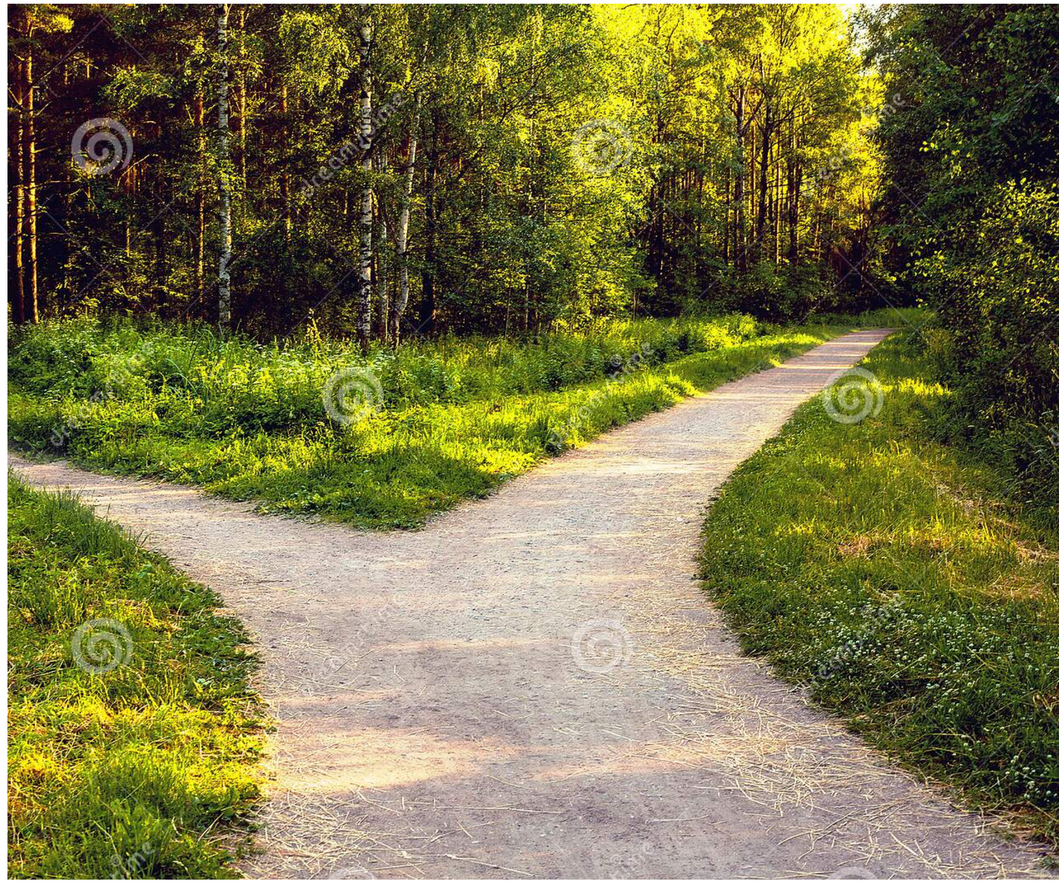
# Grammaticality

## Examples

(1) acceptability affected by performance factors

  (1) e.g. multiple centre embeddings:
  **The malt (the rat (the cat chased) ate) stank**

  (2) e.g. so-called garden path sentences:
  **The horse raced past the barn fell**



# Grammaticality

**Grammaticality is independent of 'semantic appropriateness'**

(1) a famous example: sentence grammatical, even though it is nonsensical:

(a) Colourless green ideas sleep furiously.

(2) contrasts with ungrammatical (b):

(b) *Furiously sleep ideas green colourless.

(3) even following minor variants ungrammatical:

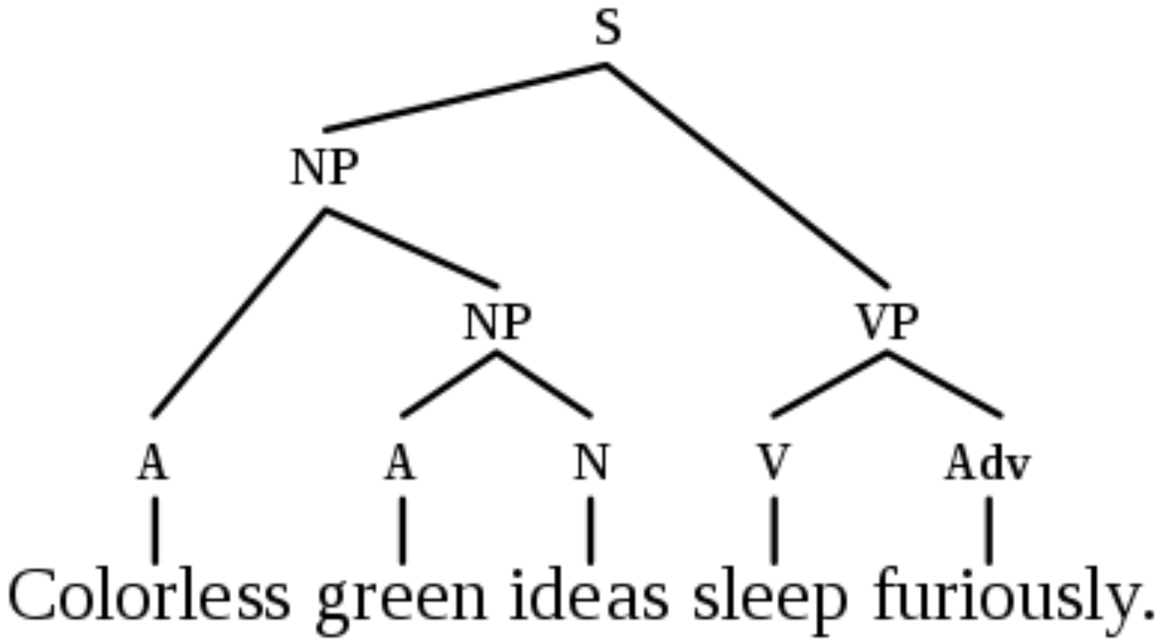
(c) *Colourless green idea sleep furiously.

(d) *Colourless green ideas sleeps furiously.

# Syntactic Theory and Natural Language

(1) The task of explaining why some strings are grammatical and others not is addressed by **syntactic theory**

(2) Syntactic theory is the study of the principles governing how words are combined to form sentences



# A formal grammar or generative grammar

(1) In the generative paradigm of syntactic research, this goal pursued via the specification of **formal grammars**

(2) A formal grammar or generative grammar is . . .
an abstract system of **rules and principles** that formally characterise the properties of the well-formed sentences of a language, distinguishing them from ungrammatical strings

(3) Such a grammar is said to **generate the language**

# Why Should Natural Language Have Syntax?

(1) Syntax plays a vital role in making language useful for communication

(2) Syntax limits possible sentence forms:
(a) John found Mary.
(b) Mary found John.
(c) *Found John Mary

(3) But ALSO limits the possible sentence forms for a given meaning
e.g. the meanings of (a) and (b) distinct

(4) Word order in English plays crucial role in allowing us to determine the precise relation between word meanings (in other languages?)

# Why Should Natural Language Have Syntax?

(1) This information — the precise relations between the meanings of words — can be signalled in other ways (but some mechanism is required)

(2) Japanese, for example, has greater freedom of word order

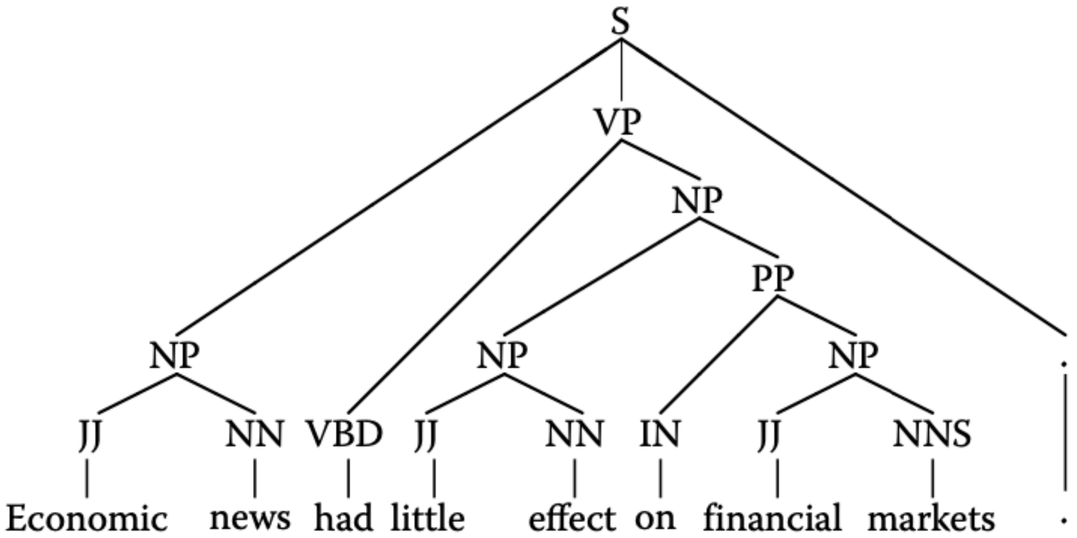
(1) roles are instead signalled by case-marking

i.e. by the morphological form of words, especially the affixes that appear

**(1) John-ga Mary-o mituketa (= John found Mary)**

**(2) Mary-o John-ga mituketa (= John found Mary)**

# Phrase Structure Grammar



# Definition of PSG

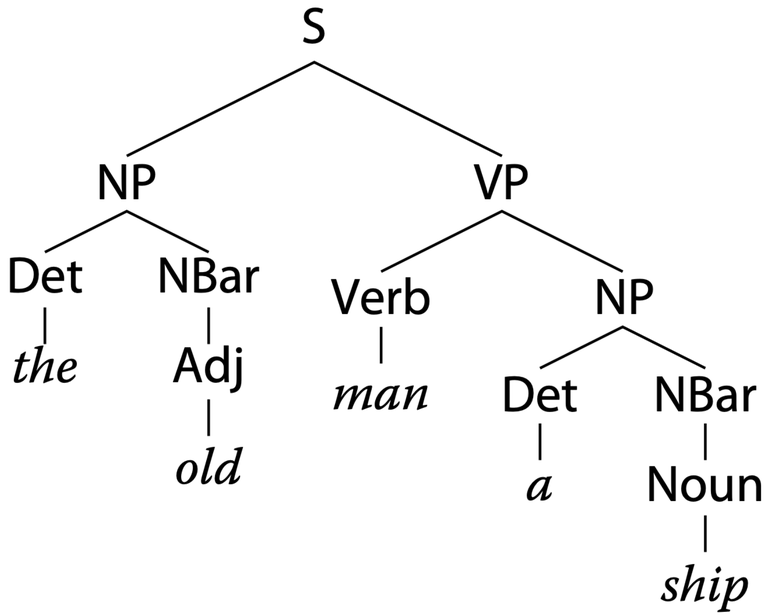
**just in case if a proffessor will ask you about it...**

(1) N a set of **non-terminal symbols**

(2) Σ a set of **terminal symbols** (disjoint from N)

(3) R a **set of productions or rules** of the form A→β, where A is a non-terminal and β is a string of symbols from (Σ∪ N)*

(4) S, a designated non-terminal called the **start symbol**




[S [NP [Det the][NBar [Adj old]]][VP [Verb man][NP [Det a][NBar [Noun ship]]]]]

# Phrase Structure Grammar

## Basics

(1) A Phrase Structure Grammar (PSG) has two key components:
a **lexicon** and **phrase structure rules**

(2) **Lexicon**
Words are categorised into word classes
a.k.a. parts of speech, form classes, lexical categories

(3) A lexicon (dictionary) is supplied as part of a PSG

(1) provides an assignment of lexical categories to words

(2) note that many words may fall into more than one class

# Lexicon

## Examples

(1) Some examples (with abbreviation in brackets):

(1) Determiner (Det) e.g. a, the

(2) Noun (N) e.g. man, fish

(3) Proper Name (PN) e.g. John, Mary

(4) Verb (V) e.g. run, eat, sit, see

(5) Preposition (P) e.g. in, to, by

(6) Adjective (Adj) e.g. red, small, angry

# Phrase structure rules

(1) words are grouped into larger units called **phrases or constituents**
phrases in turn may group into larger phrases

(2) each phrase is marked with a phrasal category

(3) the ways that words/phrases may group into larger phrases is stated by a set of rules — known as **phrase structure rules** (a.k.a. rewrite rules, context-free rules)

(4) Example — the rule: NP → Det N
• says Det followed by N may be grouped to give Noun Phrase (NP)

# Phrase structure rules

## Some other noun phrases rules

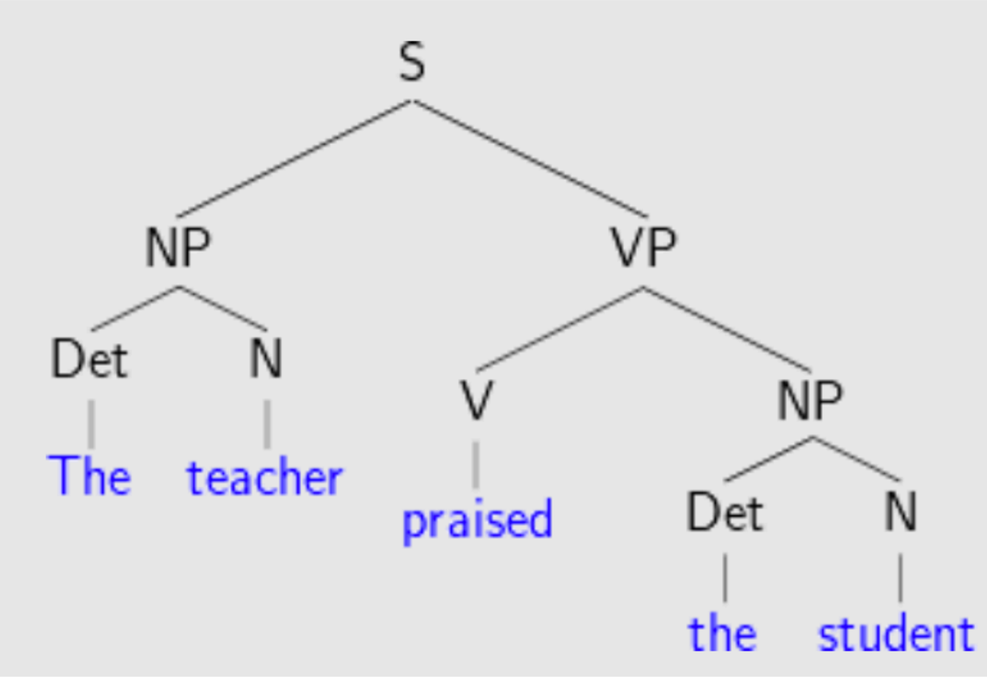
(1) NP → PN

(2) NP → Det Adj N

(3) NP → NP and NP

# PSG

(1) A sentence structure is frequently displayed in the form of **a phrase structure tree:**




(1) Alternatively, a labelled bracketting may be given, e.g.
[S [NP [Det the] [N teacher]] [VP [V praised] [NP [Det the] [N student]]]]

# What is the benefit of recognising phrases?

(1) Consider sentences:
The boy ran
John ran
The young boy ran
The young girl and John ran

(2) Might express these possibilities by a series of rules, e.g.

(1) S → Det N V

(2) S → PN V

(3) S → Det Adj N and PN V

(4) S → Det N and Det Adj N V

# How are phrases recognised?

(1) Phrases bearing the same category exhibit **similar distributional behaviour**
i.e. may appear in roughly the **same contexts**

(2) Distributional similarity is tested by substitution:

! substituting one for other in range of contexts should preserve grammaticality !

i.e. so strings before and after substitution should either both be grammatical or both be ungrammatical

# Phrase recognition

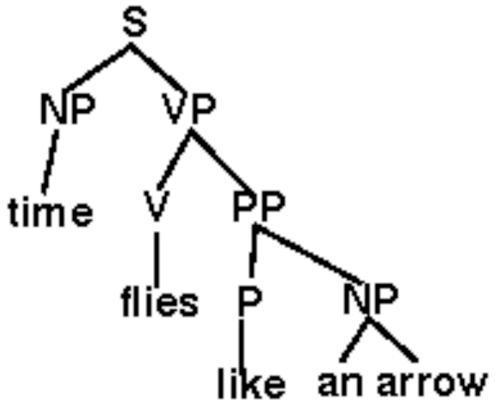
## Example

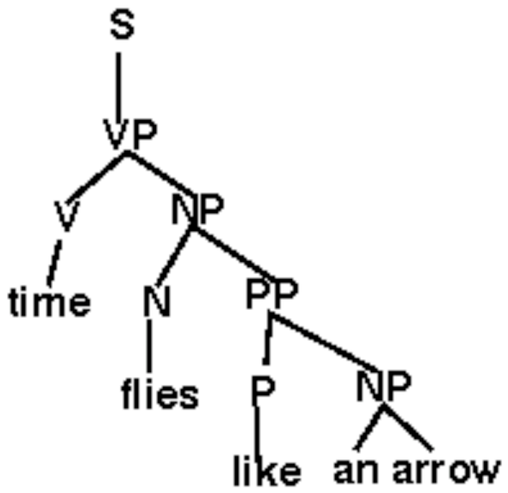
the strings **a boy** and **the fish** are both noun phrases
so, we expect following subsitution to preserve grammaticality

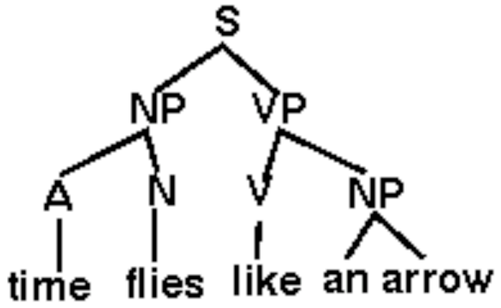
Mary admires **a boy** ⇔ Mary admires **the fish**

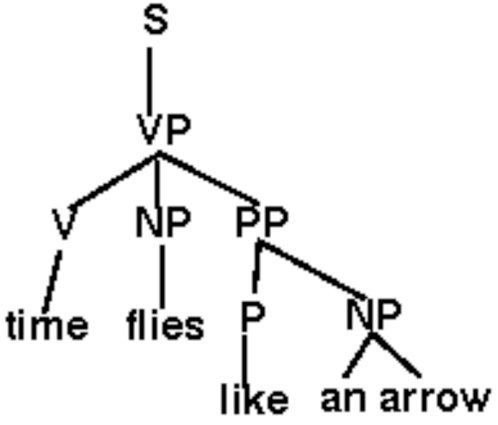
# time flies like an arrow

## how would you call this phenomenon?



# Syntax Parsing

**CKY = Cocke-Kasami-Younger algorithm**

(1) CKY Recogniser detects if a sentence is in the language defined by a grammar

(2) CKY Parser actually gives the derivation (a tree) for a sentence

(3) Parser may give many possible parses for a sentence, so we need to rank them

(4) Typical approach is **neural constituency parser** using an encoder (e.g. BERT)

(5) Detalis in Chapter 17.7 of the book

# Break

# Parsing assignment

**“garden paths”**

(1) Parse sentence and draw a parse tree

    (1) Example 1: "The old man the boat."

    (2) Example 2: “The complex houses married and single soldiers and their families.”

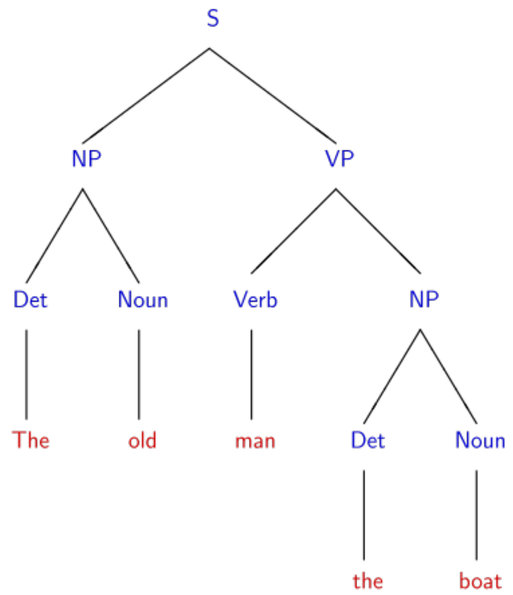
(2) Discuss your results

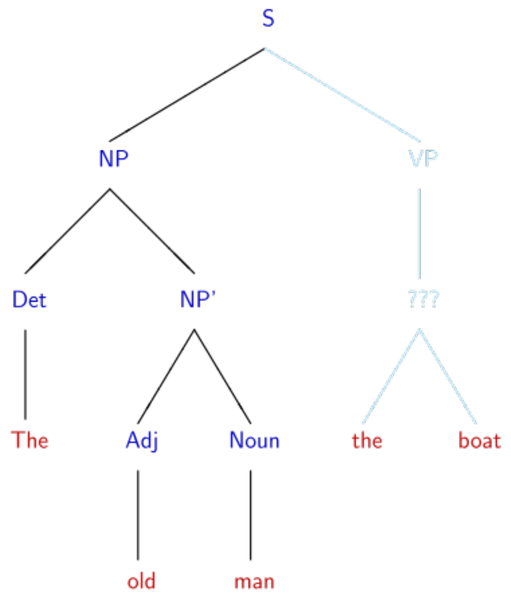
# Parsing assignment

"garden paths"

(1) Example 1: "The old man the boat."

(2) Discuss your results



# PSG

## some terminology

(1) Phrases bearing the same category exhibit **similar distributional behaviour**
i.e. may appear in roughly the same contexts

(2) Distributional similarity is tested by substitution:

  (1) substituting one for other in range of contexts should preserve grammaticality

  (2) i.e. so strings before and after substitution should either both be grammatical or both be ungrammatical

(3) Example: the strings **a boy** and **the fish** are both **noun phrases**
so, expect following subsitution to preserve grammaticality
**Mary admires a boy ⇔ Mary admires the fish**

# PSG. Terminology

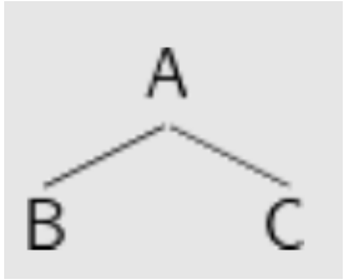
**Some standard terminology for talking about trees**

(1) **dominates:**
node X dominates node Y in tree iff there a path down from X to Y.

(2) **immediately dominates:**
node X immediately dominates node Y in tree iff X dominates Y and there are no intervening nodes on the path between

(3) **local tree:**
a local tree is a tree or subtree licensed by a single PS rule

(4) mother / daughter / sisters



# PSG Terminology: Heads

(1) Various phrasal category names make reference to a lexical category
e.g. noun phrase, verb phrase

(2) This element is the head of the phrase

(3) The head **principally determines the character** of the phrase, and
thereby its distribution

(4) For a given phrase type, phrase and head have **similar semantic** 'feels'

(5) For example, all the NP rules have a noun (or proper name) subconstituent, which is the NP's head.
e.g. the noun boy is head of NPs:
**the boy**
**the young boy**
**the young boy who lives next door**

# PSG Terminology: Complements

(1) A head may require other phrases to appear alongside it: called its **complements**

  (1) e.g. **admires** requires (or 'takes') an object NP:
    (a) John admires Mary
    (b) *John admires

  (2) e.g. **died** takes no complements:
    (c) John died
    (d) *John died Mary

  (3) e.g. verb **put** requires an object NP and PP complements:
    (e) John put the book on the table
    (f) *John put on the table

(2) What complements a head requires is related to **its meaning**
  e.g. for **admires**: meaning involves relationship between an admirer (subject NP) and an admiree (object NP)

# PSG Terminology: Complements (ctd)

(1) Complements are often obligatory, but some are **optional**
e.g. compare the following
(a) John ate some cheese
(b) John ate
— in (b), the existence of something eaten is still strongly implied

(2) Heads may be further classified in terms of the **patterns of complements** which they allow

this classification is known as **subcategorisation**

# PSG Terminology: Complements (ctd)

**subcategorisation**

(1) A head is said to **subcategorise** for its complements
e.g. a transitive verb **subcategorises for** a single object NP

(2) It is not only verbs that take complements

(1) e.g. nouns: **the chance of a lifetime** ... **the fact that he had won**

(2) e.g. adjectives: he was **eager** to please

# PSG Terminology: Adjuncts

(1) Adjuncts are phrases that supply additional information about events or entities, etc.

(2) Their presence is **always optional**

(3) An adjunct is said to **modify** the phrase with which it is associated

(4) Some examples:

(5) (a) John left **yesterday**
(b) John saw Mary **on Monday**
(c) John parked his car **in the carpark**
(d) John wrote a book **about flower arranging**

# PSG Terminology: Adjuncts (ctd)

(1) Can usually have multiple adjuncts:

(e) John left **in a hurry**, **yesterday**

(f) I saw the key **on the table**, **beside the book**

(g) John parked his car **in the carpark**, **near the entrance**

(2) Can allow for optionality/multiple occurrence characteristic of adjuncts using self recursive rules

N → N P P

VP → VP PP

S → S PP

# CFG vs. PSG

(1) **Context-free grammars** are the same as phrase structure grammar

(2) Backus–Naur form (BNF) is a metasyntax notation for context-free grammars, often used to describe the syntax of languages used in computing, protocols, ...

```
<postal-address> ::= <name-part> <street-address> <zip-part>

     <name-part> ::= <personal-part> <last-name> <opt-suffix-part> <EOL> | <personal-part> <name-part>

 <personal-part> ::= <initial> "." | <first-name>

<street-address> ::= <house-num> <street-name> <opt-apt-num> <EOL>

      <zip-part> ::= <town-name> "," <state-code> <ZIP-code> <EOL>

<opt-suffix-part> ::= "Sr." | "Jr." | <roman-numeral> | ""
   <opt-apt-num> ::= <apt-num> | ""
```



https://en.wikipedia.org/wiki/Backus–Naur_form

# Trees and Dependencies

# Chomsky's hierarchy

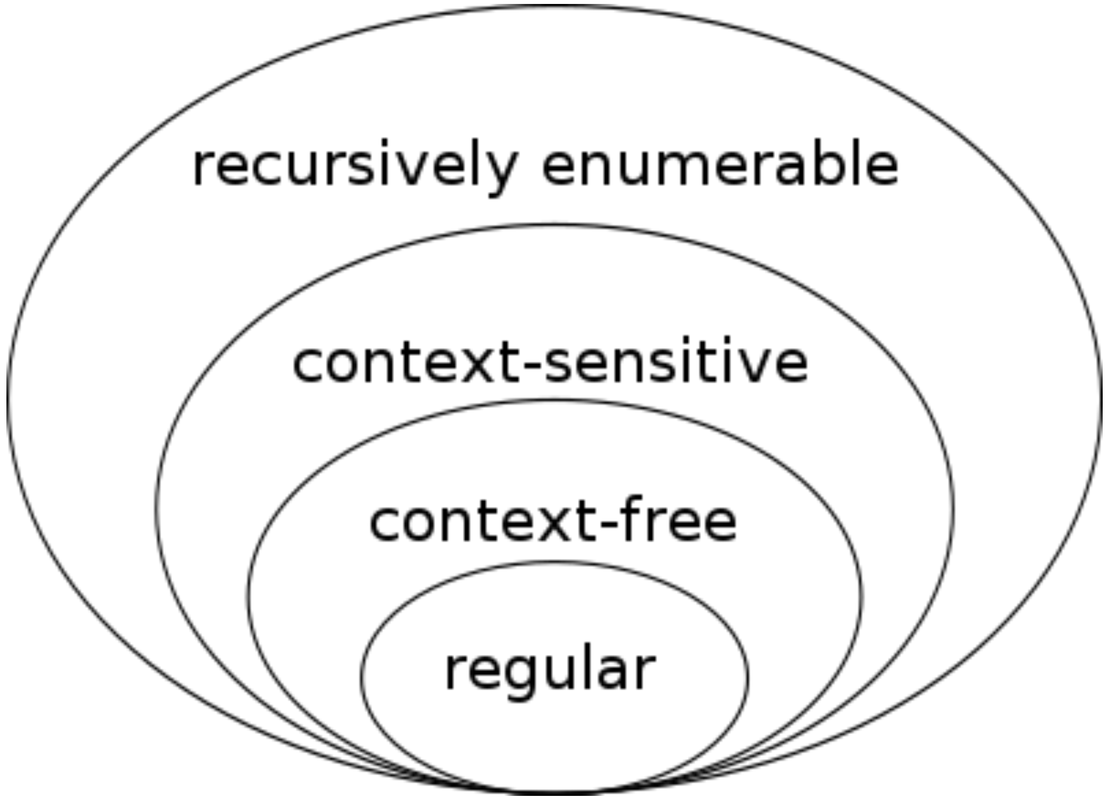
# Some history

(1) Context free grammar (CFG) was introduced by Noam Chomsky in 1956

(2) Most grammar formalisms are derived from or can somehow be related to CFG

(3) Context-sensitive is the next step (1959)

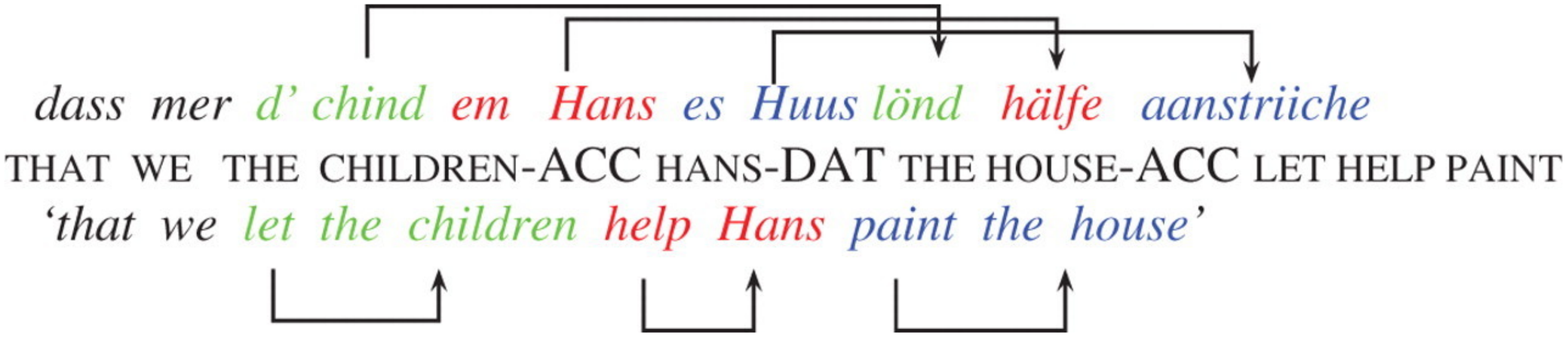
(4) **Recursively enumerable** languages are exactly all languages that can be recognized by a Turing machine.



# Where are natural languages located?

(1) For sure, natural languages are not regular

(2) Question whether they are Context-free was answered only in the mid-1980s
Context-free grammars can only handle an unbounded number of interlocked dependencies **if they are nested.**
Here, In Swiss-German number of crossing dependencies between objects and verbs are **unbounded:**



# Mildly context-sensitive languages

**Suggested by Joshi (1985)**

(1) Captures the precise formal power needed for defining natural languages

(2) Conditions:

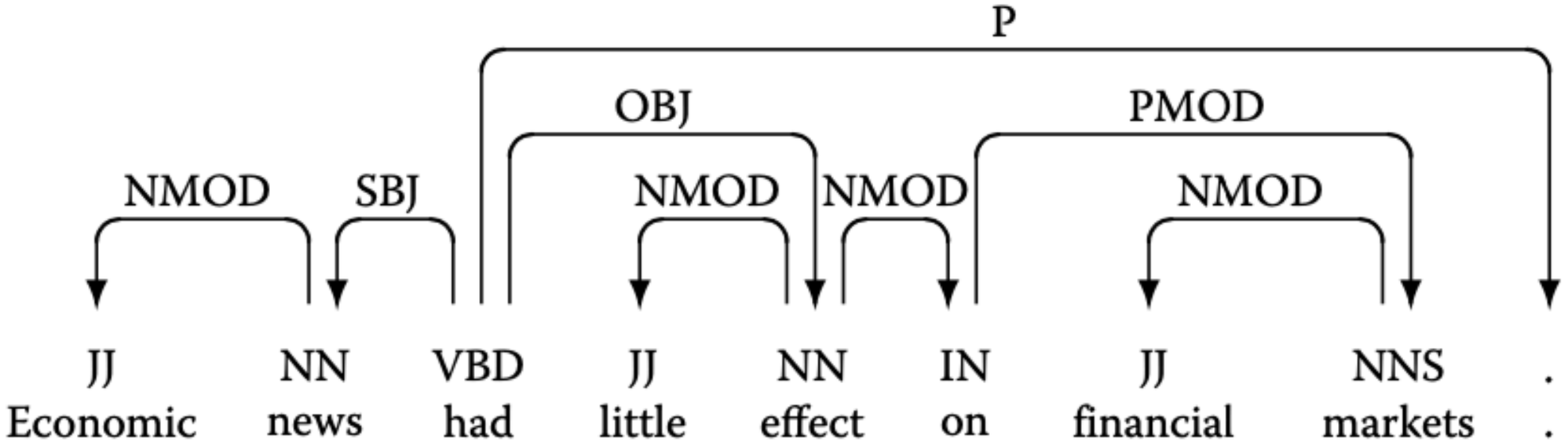
  (1) It contains all Context Freee Languages

  (2) It can describe a limited number of types of cross-serial dependencies

  (3) Its membership problem has polynomial complexity (it is **false** of Context sensitive)

  (4) All languages in it have constant growth property

# Dependency Grammar



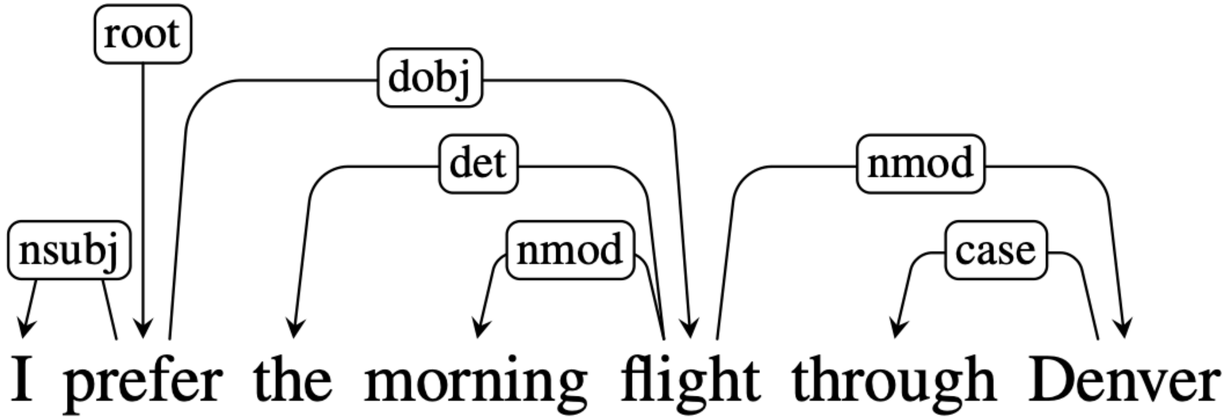
# Dependency Grammars

**DAG == directed acyclic graph**

(1) dependency grammar lacks phrasal nodes;

(2) instead the structure consists of lexical elements linked by **binary dependency relations**

(3) It is a DAG between the words in the surface sentence edges are labeled with syntactic functions (such as SUBJ, OBJ, MOD, etc.).






https://web.stanford.edu/~jurafsky/slp3/14.pdf

# Dependency Grammars

(1) The dependency grammar tradition constitutes a diverse family of different formalisms

    (1) different constraints on the dependency relation (such as allowing or disallowing crossing edges),

    (2) incorporate different extensions (such as feature terms).

(2) A major advantage of dependency grammars is their ability to deal with languages that are morphologically rich and have a **relatively free word order** (e.g. Russian)

# Dependency Tree

## Definition

(1) A dependency tree is a directed graph G=(V,E) that satisfies the following constraints:

  (1) There is a single designated **root node that has no incoming arcs**

  (2) With the exception of the root node, **each vertex has exactly one incoming arc**

  (3) There is a unique path from the root node to each vertex in V

(2) which means that:

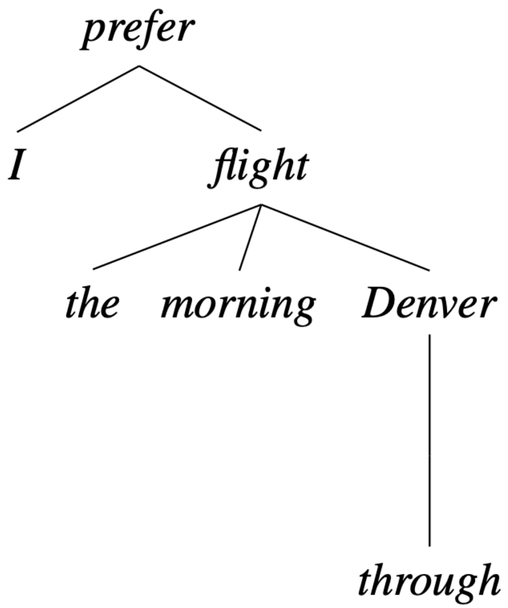
  (1) each word has a single head,

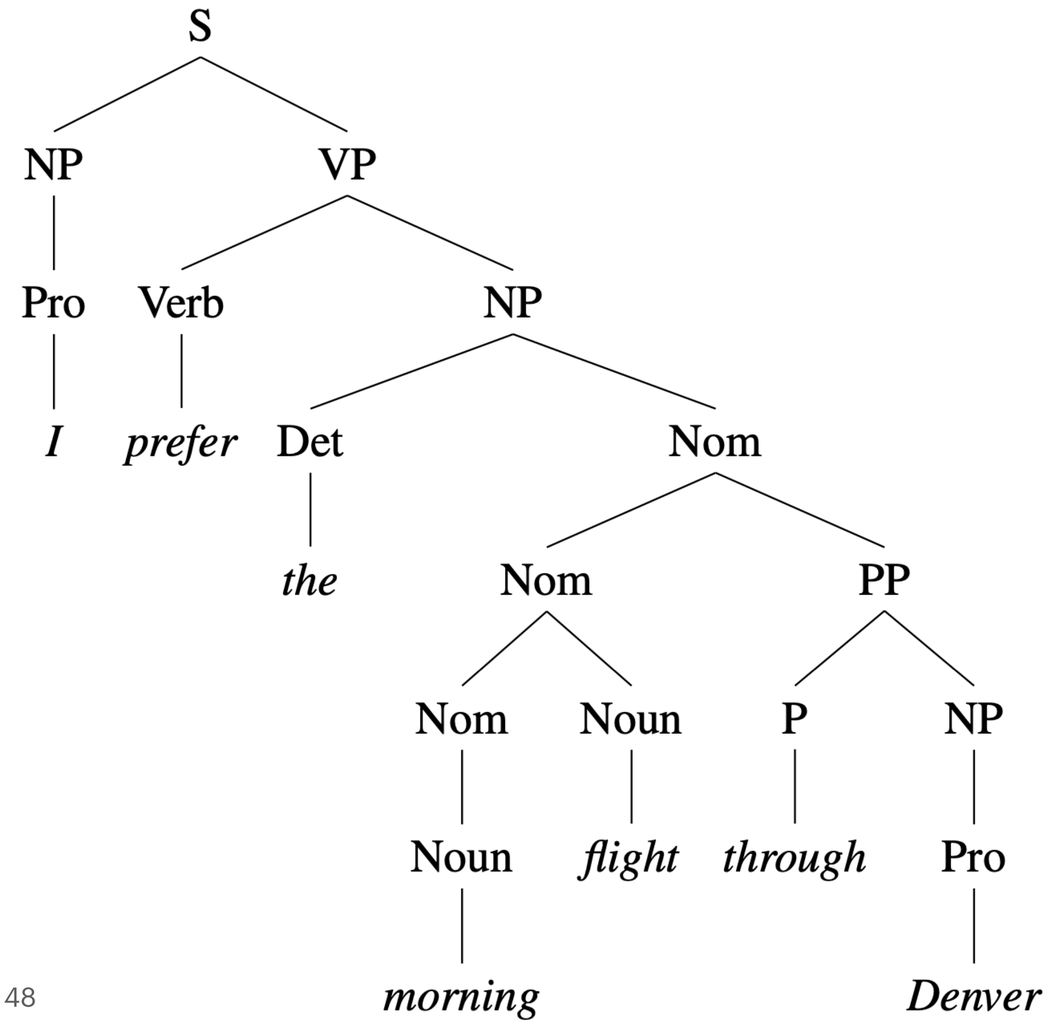
  (2) the dependency structure is connected,

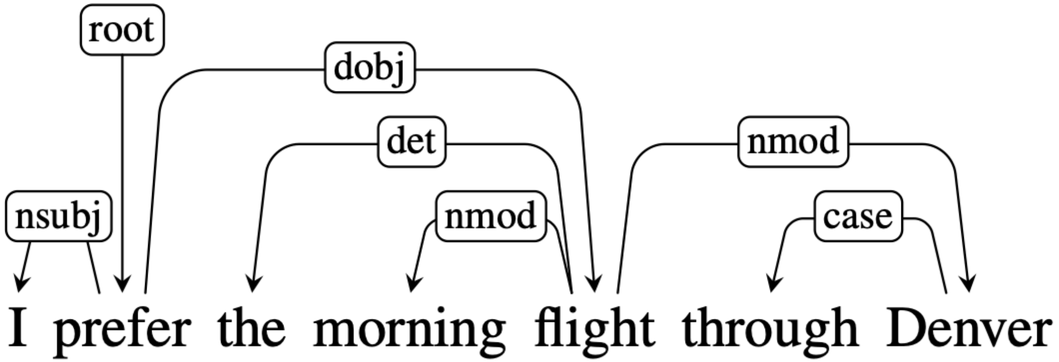
  (3) there is a single root node from which one can follow a unique directed path to each of the words in a sentence

# Example. PSG vs. Dependencies

I prefer the morning flight through Denver.



# Dependency Relations

core Universal Dependency relations

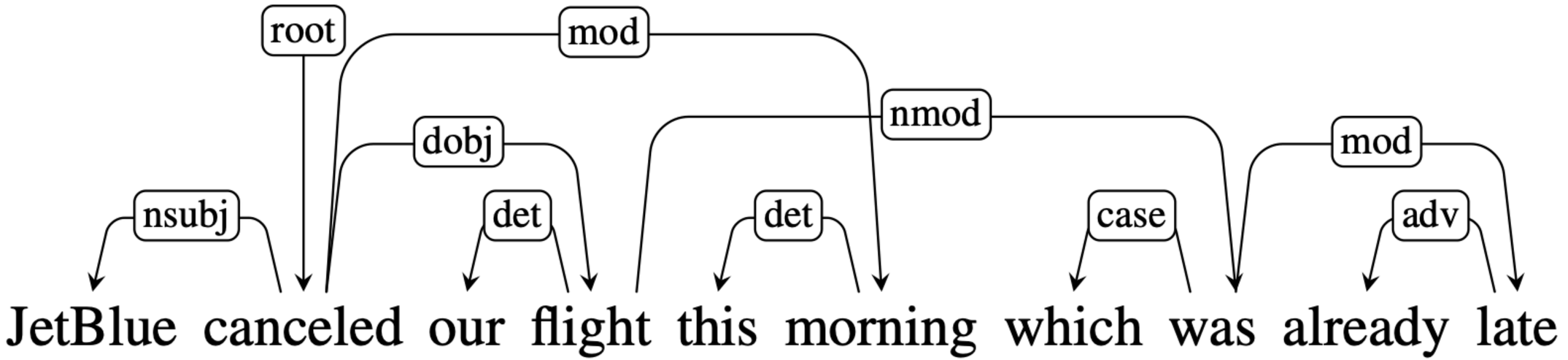
| Relation | Examples with *head* and **dependent** |
|---|---|
| NSUBJ | **United** *canceled* the flight. |
| DOBJ | United *diverted* the **flight** to Reno. |
| | We *booked* her the first **flight** to Miami. |
| IOBJ | We *booked* **her** the flight to Miami. |
| NMOD | We took the **morning** *flight*. |
| AMOD | Book the **cheapest** *flight*. |
| NUMMOD | Before the storm JetBlue canceled **1000** *flights*. |
| APPOS | *United*, a **unit** of UAL, matched the fares. |
| DET | **The** *flight* was canceled. |
| | **Which** *flight* was delayed? |
| CONJ | We *flew* to Denver and **drove** to Steamboat. |
| CC | We flew to Denver **and** *drove* to Steamboat. |
| CASE | Book the flight **through** *Houston*. |

# Projectivity

## an additional constraint

(1) Example of a non-projective arc: flight -> was

(2) non-projective means have crossing arcs (which cannot be avoided)



# Transition-Based Dependency Parsing

**Shift-Reduce algorithm**

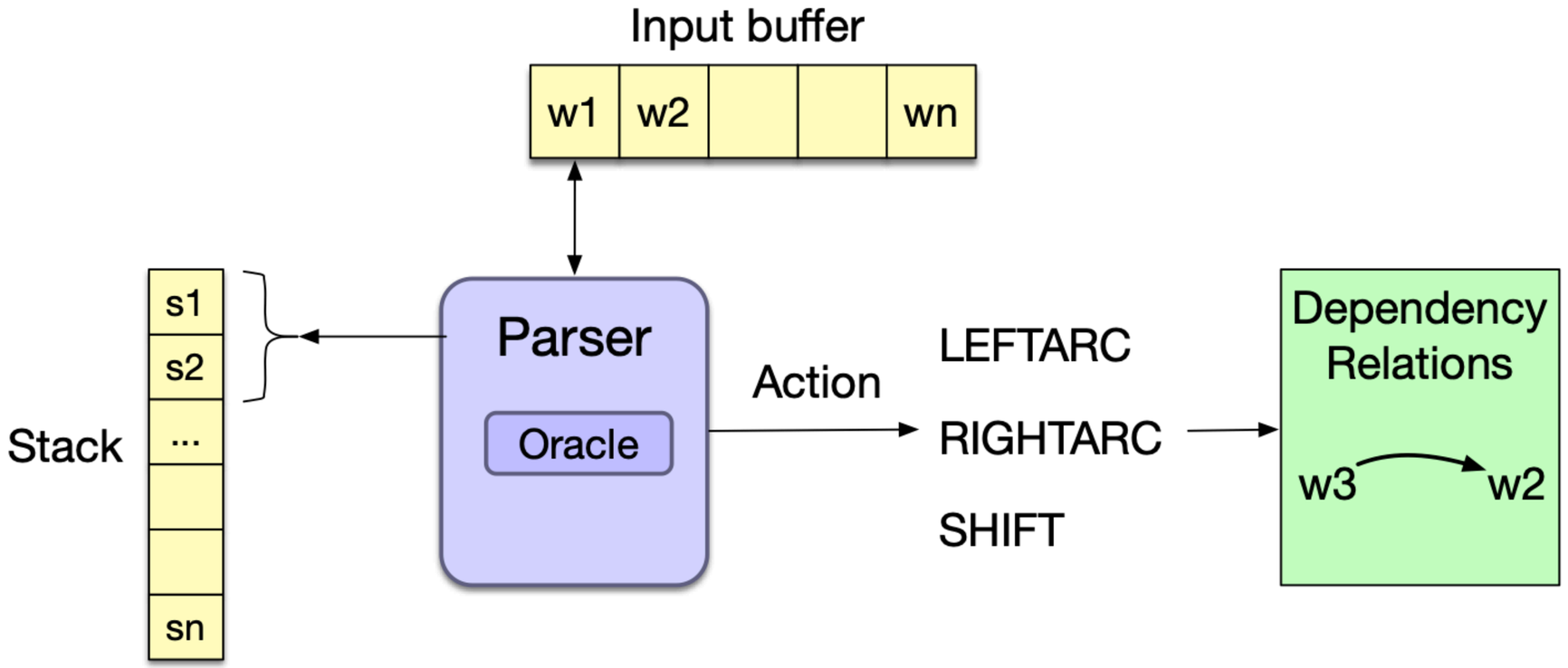
(1) Components:

  (1) Stack

  (2) Buffer

  (3) Parser

(2) 3 Actions



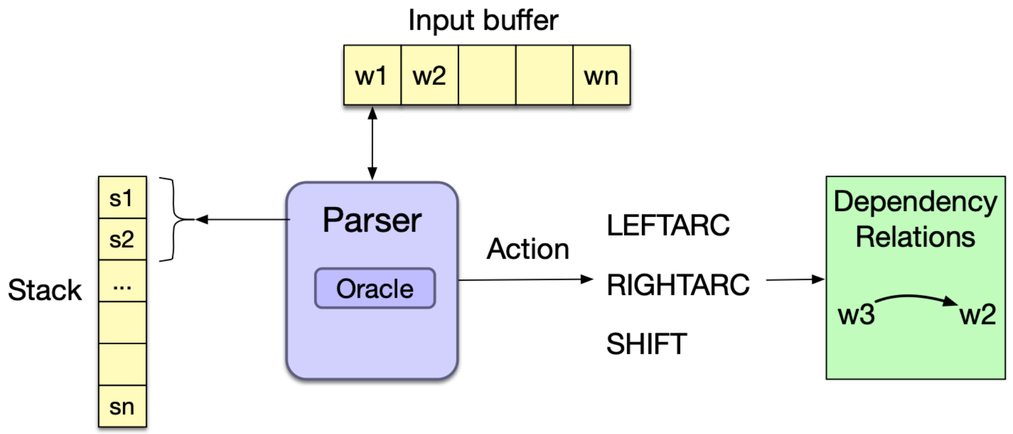
# Transitions (aka Actions)

(1) following three transition operators that will act on the **top two elements** of the stack:

(1) **LEFTARC**: Assert a head-dependent relation between the word at the top of the stack and the second word; remove the second word from the stack.

(2) **RIGHTARC**: Assert a head-dependent relation between the second word on the stack and the word at the top; remove the top word from the stack;

(3) **SHIFT**: Remove the word from the front of the input buffer and push it onto the stack.

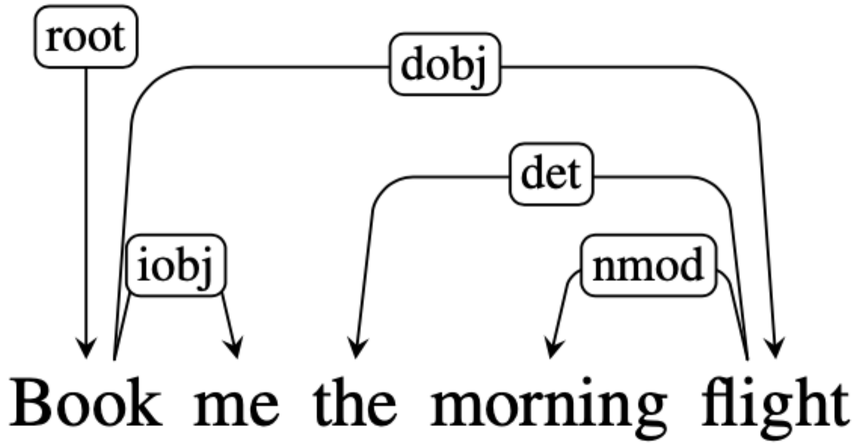


# Configuration

(1) the stack,

(2) an input buffer of words or tokens, and

(3) a set of relations representing a dependency tree.




| Stack | Word List | Relations |
|---|---|---|
| [root, book, me] | [the, morning, flight] | |

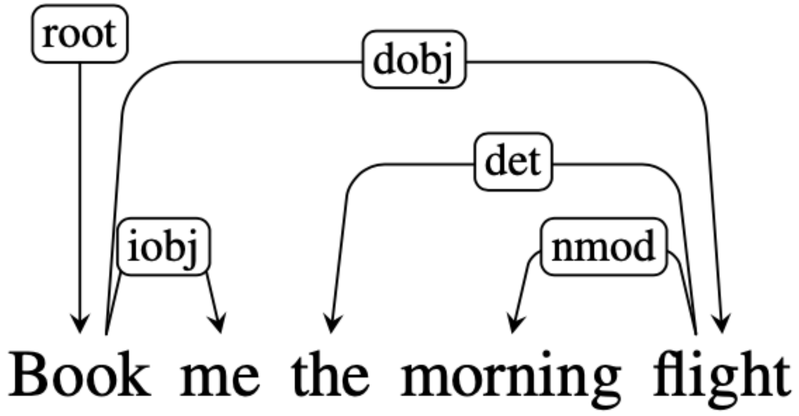
# Dependency parser

## A generic transition-based dependency parser

**function** DEPENDENCYPARSE(*words*) **returns** dependency tree

```
state ← {[root], [words], [] }  ; initial configuration
while state not final
    t ← ORACLE(state)           ; choose a transition operator to apply
    state ← APPLY(t, state)     ; apply it, creating a new state
return state
```

# Example




| Step | Stack | Word List | Action | Relation Added |
|---|---|---|---|---|
| 0 | [root] | [book, me, the, morning, flight] | SHIFT | |
| 1 | [root, book] | [me, the, morning, flight] | SHIFT | |
| 2 | [root, book, me] | [the, morning, flight] | RIGHTARC | (book → me) |
| 3 | [root, book] | [the, morning, flight] | SHIFT | |
| 4 | [root, book, the] | [morning, flight] | SHIFT | |
| 5 | [root, book, the, morning] | [flight] | SHIFT | |
| 6 | [root, book, the, morning, flight] | [] | LEFTARC | (morning ← flight) |
| 7 | [root, book, the, flight] | [] | LEFTARC | (the ← flight) |
| 8 | [root, book, flight] | [] | RIGHTARC | (book → flight) |
| 9 | [root, book] | [] | RIGHTARC | (root → book) |
| 10 | [root] | [] | Done | |

# Training the Oracle

(1) Reference parses come from TreeBanks (syntactically annotated sentences).

(2) Given a reference parse and a configuration, the training oracle proceeds as follows:

    (1) Choose **LEFTARC** if it produces a correct head-dependent relation given the reference parse and the current configuration,

    (2) Otherwise, choose **RIGHTARC** if

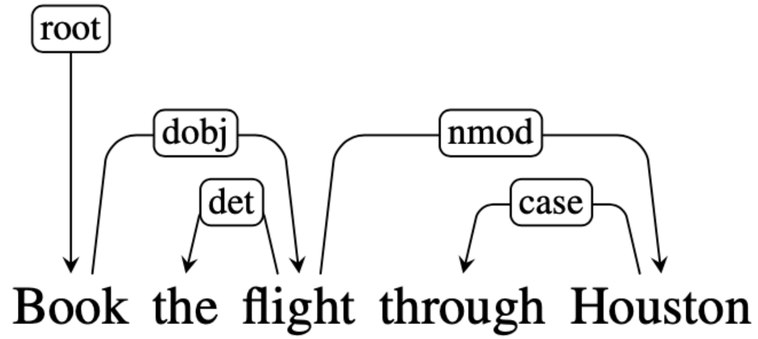
        (1) it produces a correct head-dependent relation given the reference parse and

        (2) all of the dependents of the word at the top of the stack have already been assigned,

    (3) Otherwise, choose **SHIFT**.

# Example

## Reference




How to use it in training of a feature-based classifier ?

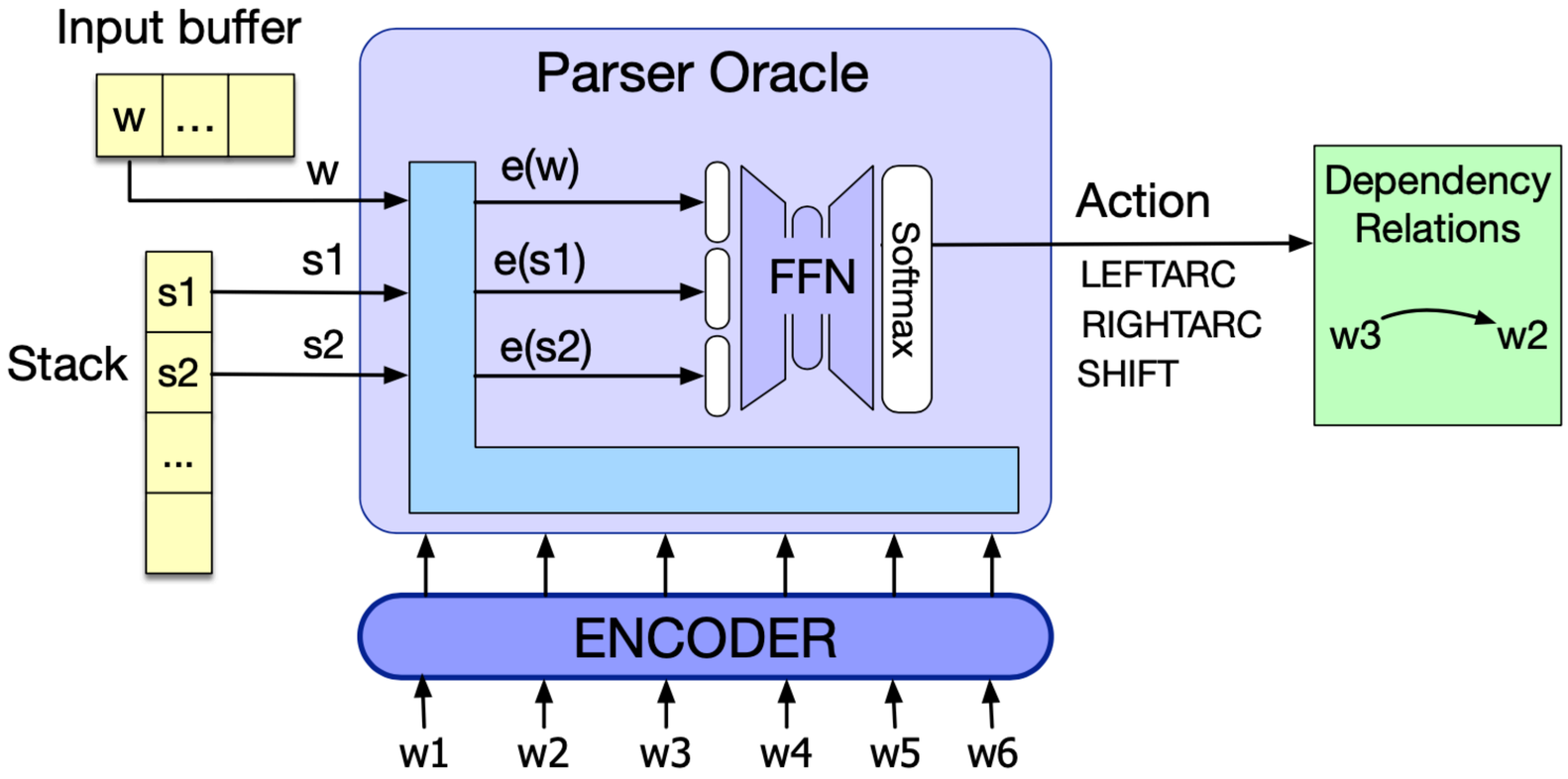
| Step | Stack | Word List | Predicted Action |
|---|---|---|---|
| 0 | [root] | [book, the, flight, through, houston] | SHIFT |
| 1 | [root, book] | [the, flight, through, houston] | SHIFT |
| 2 | [root, book, the] | [flight, through, houston] | SHIFT |
| 3 | [root, book, the, flight] | [through, houston] | LEFTARC |
| 4 | [root, book, flight] | [through, houston] | SHIFT |
| 5 | [root, book, flight, through] | [houston] | SHIFT |
| 6 | [root, book, flight, through, houston] | [] | LEFTARC |
| 7 | [root, book, flight, houston ] | [] | RIGHTARC |
| 8 | [root, book, flight] | [] | RIGHTARC |
| 9 | [root, book] | [] | RIGHTARC |
| 10 | [root] | [] | Done |

# A neural classifier

(1) The parser takes the top 2 words on the stack and the first word of the buffer,

(2) Represents them by their encodings (from running the whole sentence through the encoder),

(3) concatenates the embeddings and passes through a softmax to choose a parser action (transition)

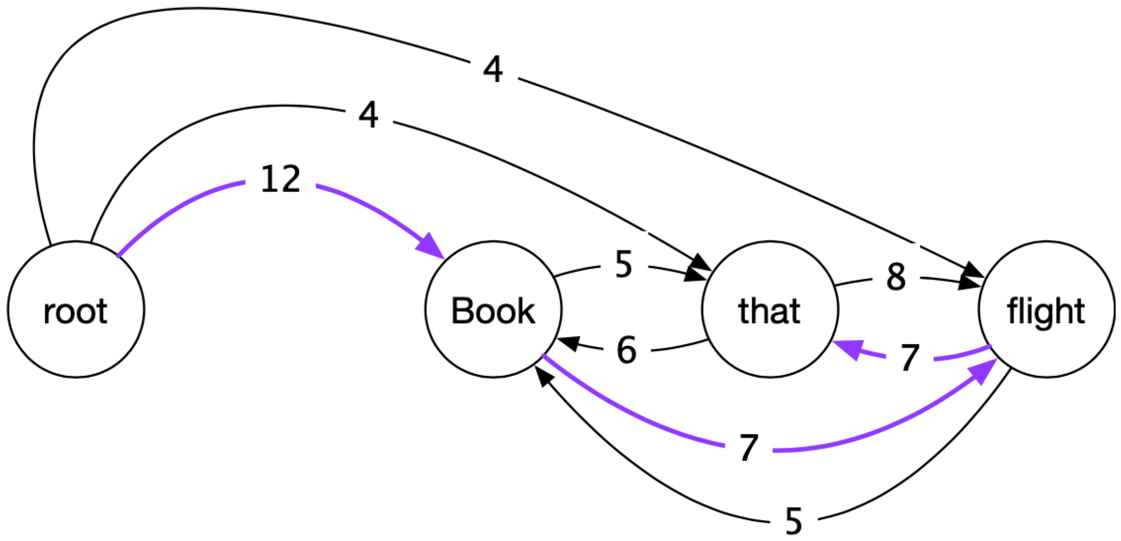


# Graph-based parsers

(1) Transition-based parsing systems employ a greedy stack-based algorithm to create dependency structures

(2) **Graph-based methods** for creating dependency structures are based on the use of **maximum spanning tree** methods from graph theory

(3) Both transition-based and graph-based approaches are developed using supervised machine learning techniques



# Summary