# Автоматический синтаксический анализ

Екатерина Владимировна Еникеева

2 октября 2023

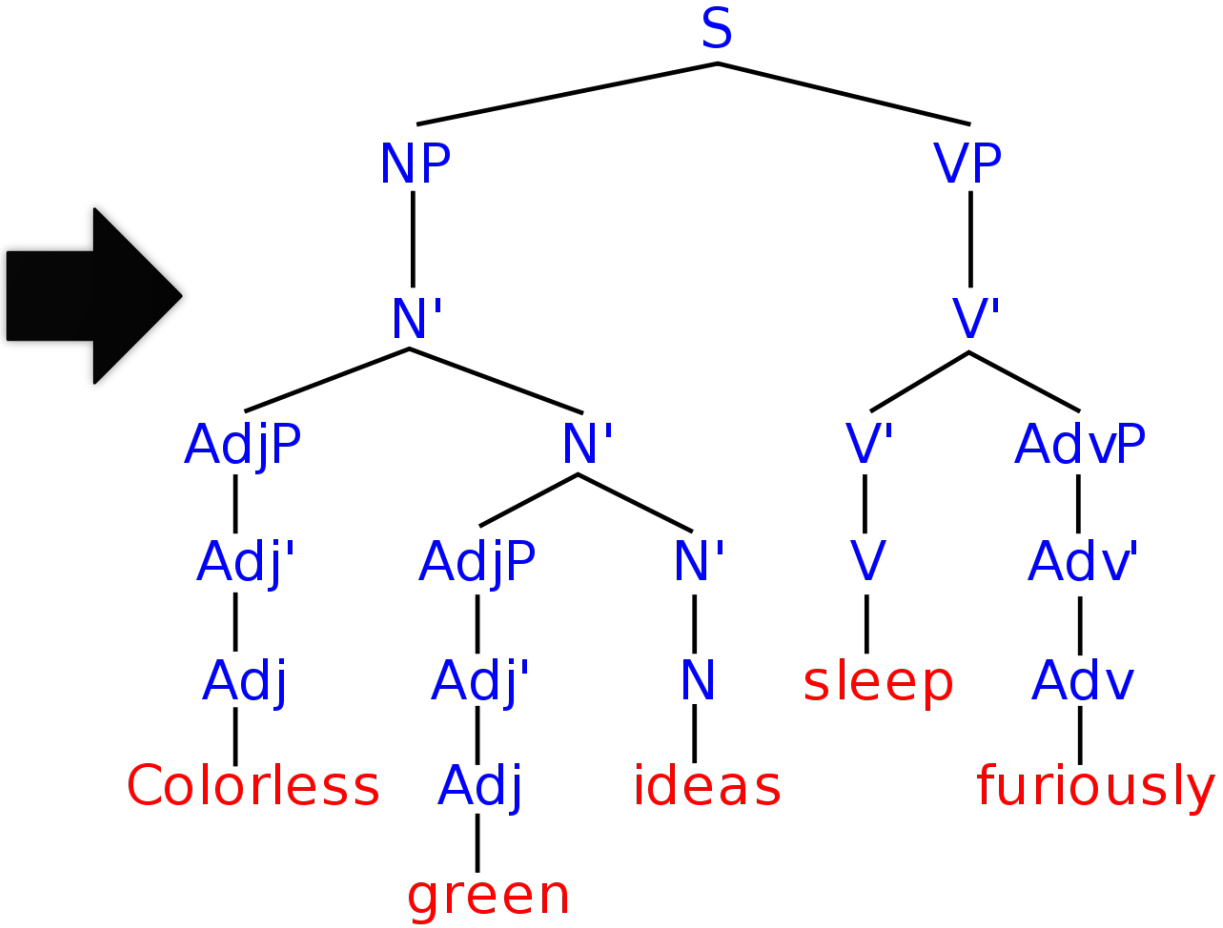
Автоматическая обработка естественного языка, лекция 5

# План

1. Зачем нужен синтаксический анализ?
2. Терминология, представление данных
3. Структура составляющих и КС-грамматики
4. Парсинг составляющих
5. Дерево зависимостей и парсинг зависимостей
6. ML-based подходы

# Основная цель

**Colorless green ideas sleep furiously.**



# Задачи

**Для естественных языков:**

- предварительный этап для семантического анализа
- извлечение фактов / диалоговые фреймы
- разрешение анафоры, кореференции
- сложность текста

**Другое:**

- языки программирования

# Пример узкой задачи

Привести фразу к начальной форме:

**Примеры использования**

все | глобальное потепление | прочие переводы

Global warming!
Глобальное потепление!

They're gonna stop global warming.
Они собираются остановить глобальное потепление.

It is already hit by global warming.
Глобальное потепление уже ударило по ней.

Maybe it's that global warming stuff, she thought.
Может, все дело в глобальном потеплении? подумала Сэмми.

The Polar Orbiting Density Scanner (PODS) was designed to help measure the effects of global warming.
Полярный орбитальный сканер плотности был создан для того, чтобы тщательнее контролировать последствия глобального потепления.

Oh, damn that global warming!
— Ох, чертово глобальное потепление!

In turn, they exacerbate global warming.
В свою очередь, они обостряют процесс глобального потепления.

# Нормализация фразы

- генерация форм по данной форме / лемме
- оценка n-граммной LM
- оптимизация поиска: beam search

Когда нужен синтаксис?

*хороший день* vs *хорошего дня*
*два котенка* vs *двух котят*

# Чанкинг

- не всегда нужно строить целое дерево

Например: выделить только NP

*Adj + Noun*

*(Adj)* + Noun*

*Noun + Noun gent*

*…*

# Терминология

- парсинг / parsing – парсер / parser
  - ЕЯ: преобразование в синтаксическое представление
  - языки разметки
  - ЯП: преобразование кода в представление, которое затем обрабатывает компилятор
- parse tree / синтаксический разбор — результат парсинга

# Терминология 2

Parse tree может выглядеть как

- Структура составляющих – constituent structure
  - объект phrase structure grammars
- Структура / дерево зависимостей – dependency tree / structure / parse
  - объект dependency grammars

# Демо: Stanford Parser

**Stanford Parser**

http://nlp.stanford.edu:8080/parser/index.jsp

Please enter a sentence to be parsed:

My dog also likes eating sausage.

Language: English  Sample Sentence  Parse

**Your query**

*My dog also likes eating sausage.*

**Tagging**

```
My/PRP$  dog/NN  also/RB  likes/VBZ  eating/VBG  sausage/NN  ./.
```

**Parse**

```
(ROOT
  (S
    (NP (PRP$ My) (NN dog))
    (ADVP (RB also))
    (VP (VBZ likes)
      (S
        (VP (VBG eating)
          (NP (NN sausage)))))
    (. .)))
```

**Universal dependencies**

```
nmod:poss(dog-2, My-1)
nsubj(likes-4, dog-2)
advmod(likes-4, also-3)
root(ROOT-0, likes-4)
xcomp(likes-4, eating-5)
obj(eating-5, sausage-6)
```

**Universal dependencies, enhanced**

```
nmod:poss(dog-2, My-1)
nsubj(likes-4, dog-2)
advmod(likes-4, also-3)
root(ROOT-0, likes-4)
xcomp(likes-4, eating-5)
obj(eating-5, sausage-6)
```

# Пример разметки CoNLL-U

```
1	Сумма	сумма	NOUN	_	Animacy=Inan|Case=Nom|Gender=Fem|Number=Sing	4	nsubj	_	_
2	аренды	аренда	NOUN	_	Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing	1	nmod	_	_
3	не	не	PART	_	Polarity=Neg	4	advmod	_	_
4	включает	включать	VERB	_	Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin|Voice=Act
5	в	в	ADP	_	_	6	case	_	_
6	себя	себя	PRON	_	Case=Acc	4	obl	_	_
7	коммунальные	коммунальный	ADJ	_	Animacy=Inan|Case=Acc|Degree=Pos|Number=Plur	8	amod	_	_
8	услуги	услуга	NOUN	_	Animacy=Inan|Case=Acc|Gender=Fem|Number=Plur	4	obj	_	_
9	!	!	PUNCT	_	_	4	punct	_	_
```

# Структура составляющих

**Составляющая—** независимая синтаксическая единица:

- можно перемещать в пределах предложения:
  - *John talked [to the children] [about rules].*
  - *John talked [about rules] [to the children].*
  - **John talked rules to the children about.*
- можно заменять на грамматически похожие:
  - *I sat [on the box / on top of the box / in front of you].*

# Структура составляющих

**NP – Noun Phrase** – именная группа
**VP – Verb Phrase** – глагольная группа
**PP – Prepositional Phrase** – предложная группа
и т.д.

- расположены линейно
- вкладываются друг в друга
- скобочный формат (bracketed notation) / дерево

[ [ [ Colorless [ green ideas ] ] [ sleep furiously ] ]

# Разметка составляющих

Penn Treebank Constituent Tags

Penn Treebank:

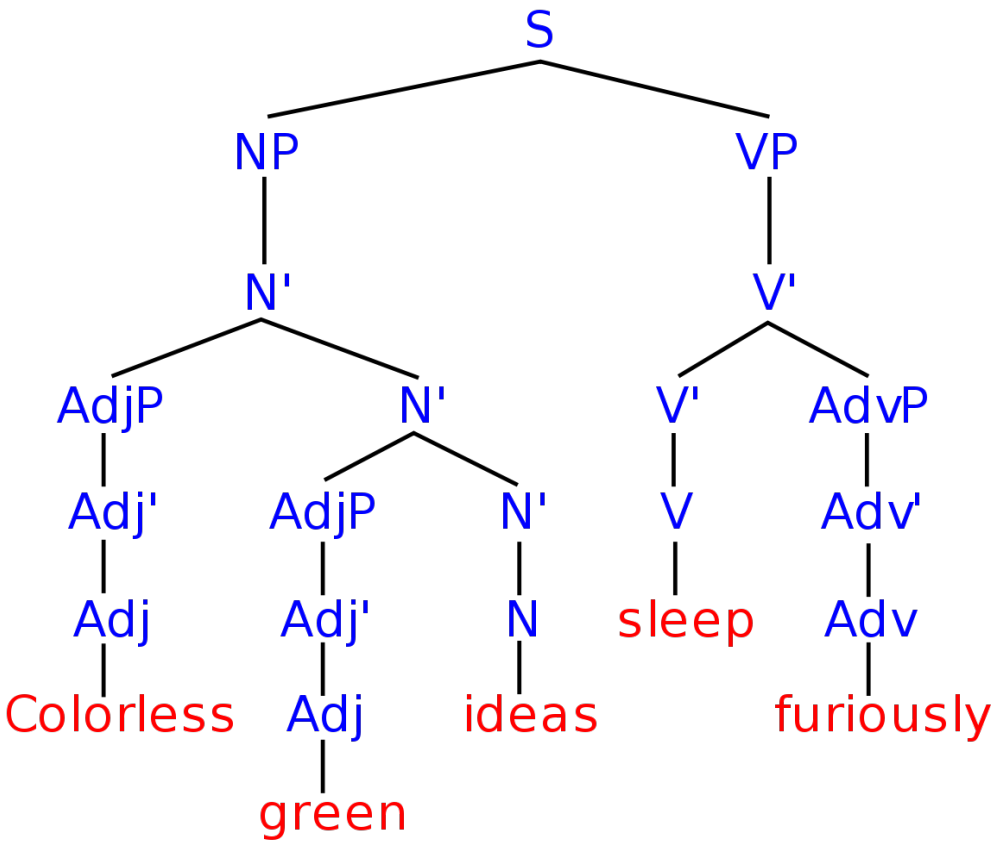
- Brown Corpus
- 1M words from WSJ
- ATIS - Air Traffic Information System

# Формальная грамматика

- Словарь $V$
  - Конечное множество нетерминалов $V_N$ включает *start symbol (S)*
  - Конечное множество терминалов $V_T$ (+ пустой символ ε)
- Грамматика
  - Конечное множество правил (продукций) $P$

➢ Описывает, какие последовательности допустимы в данном языке

# (Не)терминалы



# Пример 1

$$V_N = \{S\}$$

$$V_T = \{a, b, \varepsilon\}$$

$$S \to aSb$$

$$S \to \varepsilon$$

Какие последовательности допустимы в данном языке?

# Пример 1

$$V_N = \{S\}$$

$$V_T = \{a, b, \varepsilon\}$$

$$S \to aSb$$

$$S \to \varepsilon$$

Какие последовательности допустимы в данном языке?

$ab, aabb, aaabbb$ ….

$a^n b^n$

# Иерархия Хомского

Пусть $V^*$ — множество всех строк; $V^+$ — непустых строк.

- тип 0 – неограниченные грамматики

  $\alpha \to \beta$ , где $\boldsymbol{\alpha}$ – любая последовательность, содержащая нетерминал, $\boldsymbol{\beta}$ – любая последовательность
- тип 1 – контекстно-зависимые грамматики

  $\alpha A\beta \to \alpha\gamma\beta : \alpha, \beta \in V^*, A \in V_N \qquad \alpha \to \beta : \ 1 \le |\alpha| \le |\beta|$
- тип 2 – **контекстно-свободные грамматики**

  $A \to \beta : \beta \in V^+ (\beta \in V^*)$
- тип 3 – регулярные грамматики

  $A \to B\gamma \ / \ A \to \gamma : \gamma \in V_T^*, A, B \in V_N$

# Пример 2

$$S \to \langle NP \rangle \langle VP \rangle$$
$$\langle NP \rangle \to A \langle NP \rangle$$
$$\langle NP \rangle \to N$$
$$\langle VP \rangle \to V \langle NP \rangle$$
$$\langle VP \rangle \to V$$
$$N \to ideas \mid linguists$$
$$V \to generate \mid hate \mid eat$$
$$A \to great \mid green$$

*Great linguists generate green ideas.*

*Linguists hate green ideas.*

*Great ideas eat linguists.*

*...*

# Нормальная форма Хомского

грамматика в НФ Хомского (CNF) содержит правила вида:

$$A \to BC$$

$$A \to \alpha$$

$$\left(S \to \varepsilon\right)$$

- является КС-грамматикой
- любую КС-грамматику можно привести к НФ Хомского
- binary branching – строим бинарные деревья

# Упражнение 1

Попробуем привести к CNF следующую КС-грамматику:

$S \rightarrow AB$
$A \rightarrow 0A1 \mid \varepsilon$
$B \rightarrow B1 \mid \varepsilon$

# Упражнение 2

Попробуем построить КС-грамматику, порождающую следующие предложения:

Я ем грушу.
Я ем красивую грушу.
Я ем большую красивую грушу.
Я ем большую красивую грушу и яблоко.
Я ем большую красивую грушу и зелёное яблоко.

# Сложные явления 1

- Модель управления (subcategorization frame)

Verb-with-NP-complement → find | leave | repeat | ...
Verb-with-S-complement → think | believe | say | ...
Verb-with-Inf-VP-complement → want | try | need | ...
VP → Verb-with-NP-comp NP
VP → Verb-with-S-comp S s
...

| Frame | Verb | Example |
|---|---|---|
| ∅ | eat, sleep | I ate |
| *NP* | prefer, find, leave | Find [$_{NP}$ the flight from Pittsburgh to Boston] |
| *NP NP* | show, give | Show [$_{NP}$ me] [$_{NP}$ airlines with flights from Pittsburgh] |
| $PP_{\text{from}}$ $PP_{\text{to}}$ | fly, travel | I would like to fly [$_{PP}$ from Boston] [$_{PP}$ to Philadelphia] |
| *NP* $PP_{\text{with}}$ | help, load | Can you help [$_{NP}$ me] [$_{PP}$ with a flight] |
| *VPto* | prefer, want, need | I would prefer [$_{VPto}$ to go by United Airlines] |
| *S* | mean | Does this mean [$_{S}$ AA has a hub in Boston] |

**Figure 12.6** Subcategorization frames for a set of example verbs.

# Сложные явления 2

- Сочинение (coordination)

NP → NP and NP
VP → VP and VP
…

X → X - metarule

А также: согласование, long-distance dependencies …

# Лексикализованные грамматики

- Combinatory Categorial Grammar  (CCG)
- Lexical-Functional Grammar (LFG)
- Head-Driven Phrase Structure Grammar (HPSG)
- Tree-Adjoining Grammar (TAG)
- …

# Синтаксический анализ

- соотнесение входной строки (предложения) в заданной (или обученной по корпусу) грамматикой:

- распознавание (recognition) : да/нет – однозначный ответ
- собственно анализ (parsing) : дерево разбора / структура составляющих / последовательность продукций – возможны разные варианты

# Оценка качества (составляющие)

*Constituent-level precision / recall / F-score* — аналогично IR

Верный ответ: совпадение индексов начала/конца составляющей и тега нетерминала

$$\textbf{labeled recall:} = \frac{\text{\# of correct constituents in hypothesis parse of } s}{\text{\# of correct constituents in reference parse of } s}$$

$$\textbf{labeled precision:} = \frac{\text{\# of correct constituents in hypothesis parse of } s}{\text{\# of total constituents in hypothesis parse of } s}$$

# Алгоритмы парсинга

- **top-down parsing** – нисходящие алгоритмы разбора (например, *Earley parser*)
- **bottom-up parsing** – восходящие алгоритмы разбора (например, *CYK parser*)

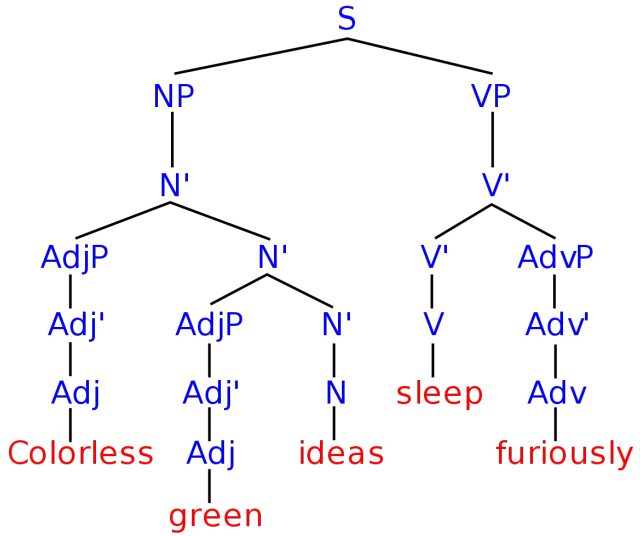
$S \longrightarrow \langle NP \rangle \langle VP \rangle$

$NP \longrightarrow N'$

$VP \longrightarrow V'$

$N' \longrightarrow \langle AdjP \rangle \langle N' \rangle$

…



# CYK парсер

**Алгоритм Кока-Янгера-Касами / CYK / CKY**

| | | | |
|---|---|---|---|
| **S** | | | |
| | | | |
| NP | | VP | |
| Det | N | V | Adv |
| **the** | **cat** | **sleeps** | **quitely** |

$$S \rightarrow \langle NP \rangle \langle VP \rangle$$

$$NP \rightarrow Det \langle NP \rangle$$

$$VP \rightarrow \langle VP \rangle Adv$$

$$NP \rightarrow N$$

$$VP \rightarrow V$$

$$N \rightarrow cat$$

$$Det \rightarrow the$$

$$V \rightarrow sleeps$$

$$Adv \rightarrow quitely$$

# Парсер Эрли

**Earley Parser**

Итеративно «*распознает*» правила, храня таблицу соответствующих *состояний* (*dotted rules*):

$S \rightarrow \cdot\, VP$, [0,0] << 0 позиция в списке входных токенов

$NP \rightarrow Det \;\cdot\; Nominal$, [1,2] << NP начинается с 1 токена, точка в позиции 2

$VP \rightarrow Verb\ NP \cdot$, [0,3] << конец парсинга

# Парсер Эрли

**Процедуры на шаге *k*:**

- *Prediction*: раскрываем нетерминалы справа от точки, добавляя новые правила в таблицу

Из $S \rightarrow \cdot\, VP$, [0,0] добавляем $VP \rightarrow \cdots$

- *Scanning*: сопоставляем POS-нетерминалы справа от точки входным токенам; сдвигаем точку, если нашли совпадение

Если есть $Verb \rightarrow book$, то из $VP \rightarrow \cdot\, Verb\ NP$, [0,0] добавляем $VP \rightarrow Verb \cdot NP$, [0,1]

- *Completion*: если точка оказалась в конце правила, ищем по предыдущим состояниям

$NP \rightarrow Det\ Nominal\ \cdot$, [1,3] + $VP \rightarrow Verb \cdot NP$, [0,1] = успех

# Парсер Эрли

Подробный пример разбора можно найти в учебнике Jurafsky+Martin: глава 13 в изд. 2

# PCFG

**Probabilistic Context Free Grammar**

- каждое правило сопровождается весом (вероятностью)
- сумма всех вероятностей расширений нетерминалов = 1
- консистентная PCFG – сумма вероятностей всех предложений языка = 1

# Вероятностный парсинг

Вероятность разбора $T$, состоящего из $n$ правил вида $LHS \rightarrow RHS$, для предложения $S$:

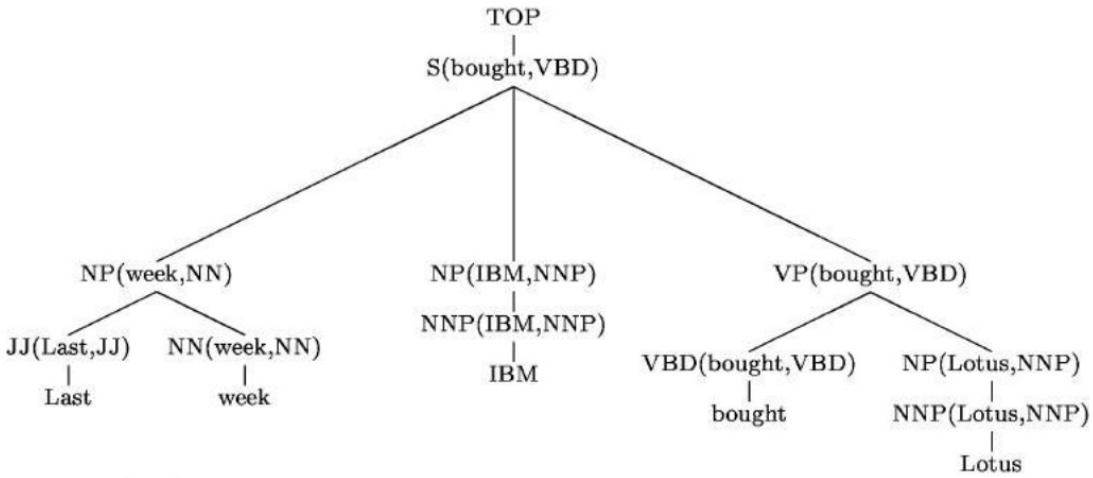
$$P(T,S) = \prod_{i=1}^{n} P(RHS_i \mid LHS_i)$$

Можно использовать вариацию CYK –

**probabilistic CYK**

Оценка вероятностей – по корпусу

# Lexicalized parsers




**Internal Rules:**

| | | | | |
|---|---|---|---|---|
| TOP | → | S(bought,VBD) | | |
| S(bought,VBD) | → | NP(week,NN) | NP(IBM,NNP) | VP(bought,VBD) |
| NP(week,NN) | → | JJ(Last,JJ) | NN(week,NN) | |
| NP(IBM,NNP) | → | NNP(IBM,NNP) | | |
| VP(bought,VBD) | → | VBD(bought,VBD) | NP(Lotus,NNP) | |
| NP(Lotus,NNP) | → | NNP(Lotus,NNP) | | |

**Lexical Rules:**

| | | |
|---|---|---|
| JJ(Last,JJ) | → | Last |
| NN(week,NN) | → | week |
| NNP(IBM,NNP) | → | IBM |
| VBD(bought,VBD) | → | bought |
| NNP(Lotus,NN) | → | Lotus |

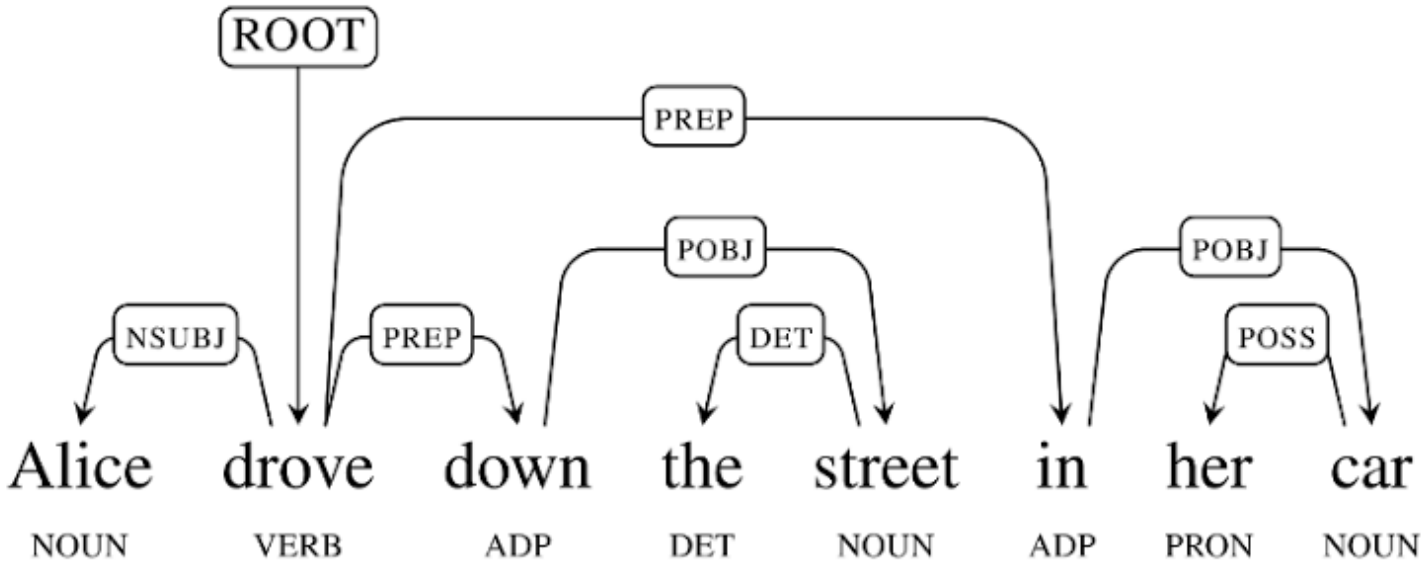
# Вероятностный парсинг

- Collins Parser — *Collins M. (2000). Head-driven Statistical Models for Natural Language Parsing. Computational Linguistics, 29(4).*

http://www.cs.columbia.edu/~mcollins/code.html

- Charniak Parser — *Charniak E. (1997). Statistical Parsing with a Context-Free Grammar and Word Statistics. AAAI-97.*

# Структура зависимостей



# Структура зависимостей

Элементы:

- зависимости (ребро, edge) : типы зависимостей
- узлы / вершины (node / vertex)
  - вершины / главные слова / хозяева
  - зависимые / подчинённые / слуги

> дерево – граф, у которого есть корневой узел (корень)

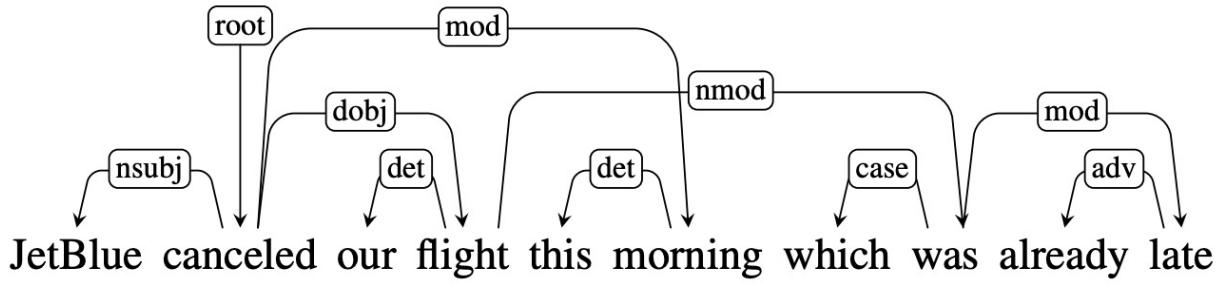
# Требования

- Существует единственный корень
- В каждую вершину входит только одна стрелка
- Для каждой вершины существует единственный путь до неё от корня

# Структура зависимостей

Важное свойство деревьев зависимостей – **проективность**

- никакая пара стрелок не пересекается (принцип непересечения стрелок)
- никакая стрелка не накрывает корневой узел (принцип обрамления стрелок)



# Структура зависимостей

Проблемы:

- принцип единственности вершины
  - *Мы оставили комнату закрытой.*
- сочинительные конструкции
  - *Прошли день и ночь.*
  - *необходимые условия и результаты*
- иерархия синтаксических единиц

# Описание зависимостей

- Могут быть получены из структуры составляющих с использованием алгоритмов head-finding:
  - задаются в КС-грамматике или по правилам
- Упорядоченные пары (head, dep)
  (flight, morning) (<root>, book)
- Пары с указанием направления стрелки:
  rightarc, leftarc
- Тройки (head, dep, relation)

# Оценка качества (зависимости)

- *Unlabelled Attachment Score* (*UAS*) = доля верно приписанных вершин
- *Labelled Attachment Score* (*LAS*) = доля верно приписанных вершин + размеченных отношений (accuracy по паре тегов)
- *Morphology-Aware Labeled Attachment Score* (*MLAS*)
- *Bilexical dependency score* (*BLEX*)

См. CoNLL Evaluation

# Парсер зависимостей

простой подход для языков программирования — shift-reduce parser

- грамматика
- стек (stack)
- список входных токенов

➢ для первых 2 элементов стека находим соответствие в грамматике и т.д.

# Transition-based parser

**Состоит из**

- грамматики (oracle – «чёрный ящик»)
- «конфигурации парсера»
  - стек (stack)
  - список входных токенов
  - список отношений, составляющих дерево зависимостей

# Парсер зависимостей

**Операции переходов (transition operators)**

- LeftArc – первое слово – главное, второе – зависимое, убираем второе из стека
- RightArc – второе слово – главное, первое – зависимое, убираем первое из стека
- Shift – берем очередное слово из входных токенов и кладём в стек

# Arc standard approach

**Ограничения**:

- Анализируются только токены (обычно 2) в верху стека
- Когда у токена находится вершина, он удаляется из стека

# Пример

Разбор предложения «*Book me the morning flight*»

| 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 | |

# Arc eager approach

**Операции переходов (transition operators)**

- LeftArc – входное слово – главное, верх стека – зависимое, убираем верхнее из стека
- RightArc – верх стека – главное, входное слово – зависимое, убираем верхнее из стека
- Shift – берем очередное слово из входных токенов и кладём в стек
- Reduce – удаляем верхнее из стека

# Обучение парсеров

- результат LeftArc / RightArc получается с помощью предсказателя (Oracle) – выбор подходящего отношения зависимости и главного слова
- это как раз и можно обучать!

*Nivre J. (2009). Non-projective Dependency Parsing in Expected Linear Time. ACL IJCNLP 2009.*

# Примеры

https://colab.research.google.com/drive/1GMDA17iP_-SiJq8_MMQTT20VrWPC3blE?usp=sharing

# Как представить данные

- Парсер – конфигурация на каждом шаге
  стек $S$, список отношений $R_c$
- Корпус (treebank) – деревья зависимостей
  множества вершин V и отношений $R_p$

Как получить из parse tree список конфигураций?

- симулируем парсинг при условии готового дерева

  $LEFTARC(r)$: $(S_1\ r\ S_2) \in R_p$

  $RIGHTARC(r)$: $(S_2\ r\ S_1) \in Rp$ и $\forall r', w$ таких что $(S_1\ r' w) \in R_p$
  $(S_1\ r'\ w) \in R_c$

  $SHIFT$: во всех остальных случаях

# Задача

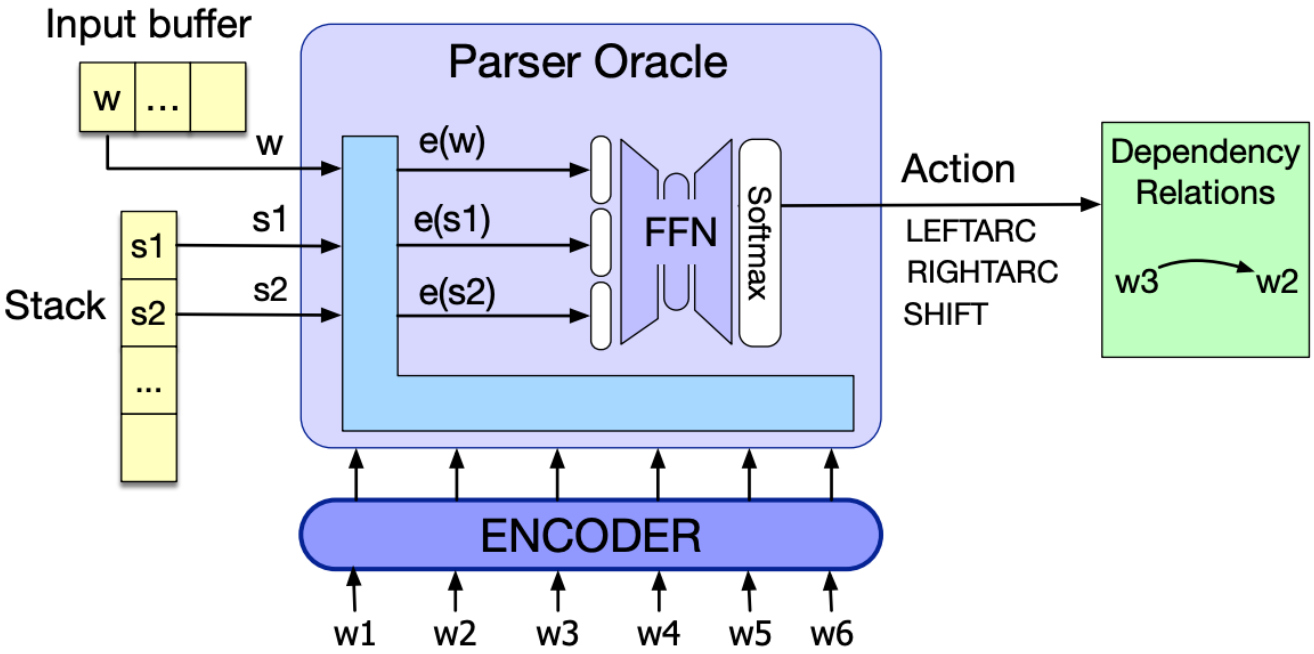
| Stack | Word buffer | Relations |
|---|---|---|
| [root, canceled, flights] | [to Houston] | (canceled → United)<br>(flights → morning)<br>(flights → the) |

*SHIFT*

# Features: classic

- Базовые признаки:
  - токен, лемма, POS-тег …
- Объекты, для которых можем извлечь фичи:
  - слова стека $s_1$, $s\_2$, …
  - слова входного буфера $b_1$ …
- Feature templates:
  - $< s_1.w, op >$
  - $< b_1.t, op >$
  - $< s_1.t + s_2.t, op >$

# Features: embeddings



# Улучшения

- Arc eager approach
  - Добавляет операцию *Reduce*
- Beam search:
  - На каждом шаге разрешаем несколько вариантов
  - Параметр – beam width $N$
  - Кладём лучшие разборы в список – не должен быть длиннее $N$
  - Лучшие по какому признаку?
    для конфигурации $c$ и операции $t$:

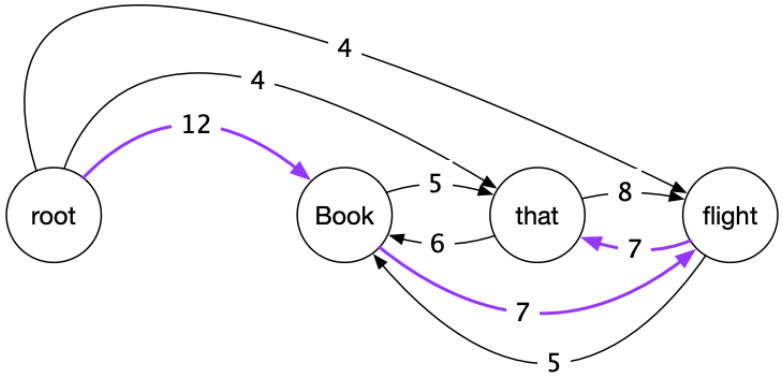
$$ScoreC(c_i) = ScoreC(c_{i-1}) + Score(t_i, c_i)$$

# Graph-based parsing

1. Оценка отношений / edge scoring
   приписываем каждой паре токенов вес, используя вероятностный классификатор

2. (optional) Label scoring

3. Выбор дерева
   алгоритм **maximum spanning tree**
   - покрывает все вершины
   - начинается в root
   - имеет максимальный вес



# Инструменты

- NLTK
- UD Pipe (1 / 2)
- Stanford CoreNLP / stanza
- MaltParser
- spaCy
- DeepPavlov