Синтаксический анализ и его применение

Попов Артём

Математические методы анализа текстов
осень 2020

Введение

Идеально ли наше представление текста?

Архитектуры, которые мы рассматривали, работают с текстом как с последовательностью или набором токенов.

Почему нам может быть этого недостаточно?

1. Разные последовательности задают один и тот же смысл:

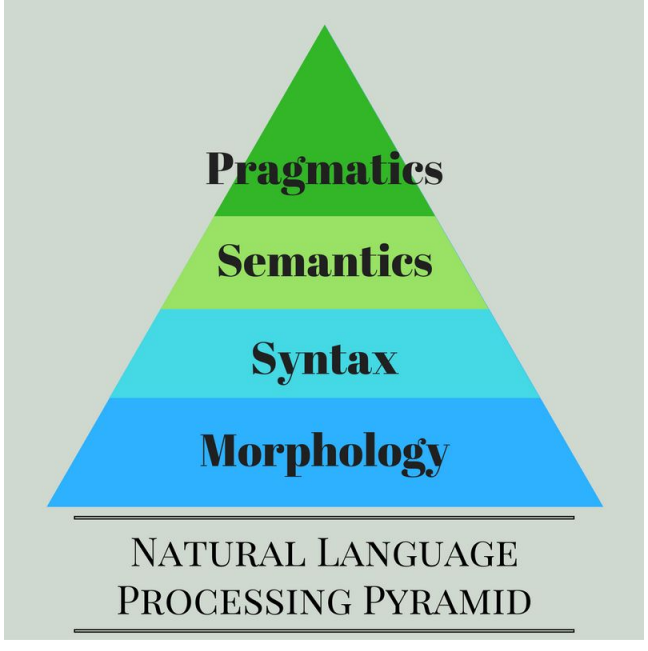
   мама мыла раму; мама раму мыла; раму мама мыла

2. Одинаковые (после лемматизации) последовательности токенов задают разный смысл:

   карта заблокирована; карту заблокируйте

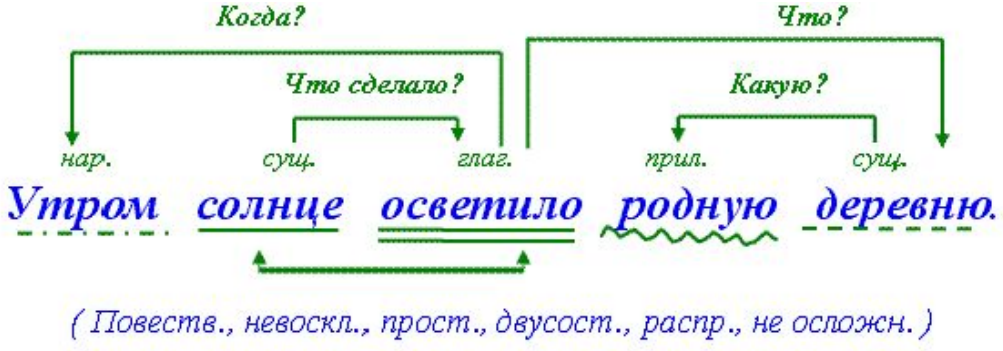
Вспомним пирамиду NLP

Прагматика	Тексты	Суммаризация
Семантика	Предложения + контекст	NER
Синтаксис	Предложения / словосочетания	Определение частей речи
Морфология	Слова	Нормализация слова



Синтаксический разбор предложения

Синтаксический разбор — анализ структуры предложения:



1. Находим не только характеристики отдельных слов, но и зависимости между словами.
2. Строим по тексту некоторый “полезный” граф.

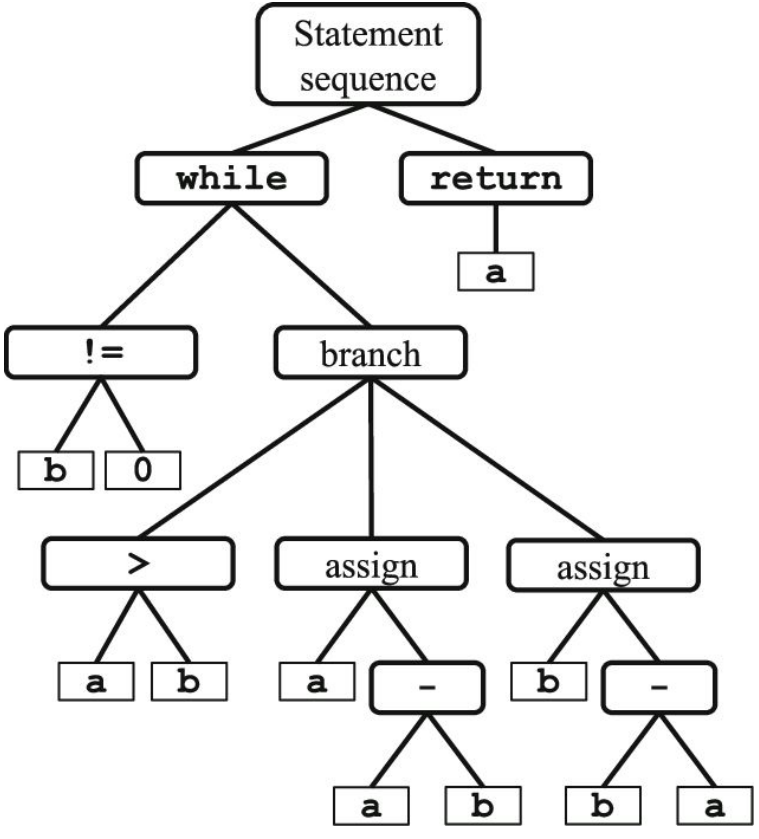
Абстрактное синтаксическое дерево (AST)

AST — представление кода.

- внутренниы вершины — операторы
- листья — операнды

Код для дерева справа:

```
while b ≠ 0
    if (a > b) a := a − b
    else b := b − a
return a
```



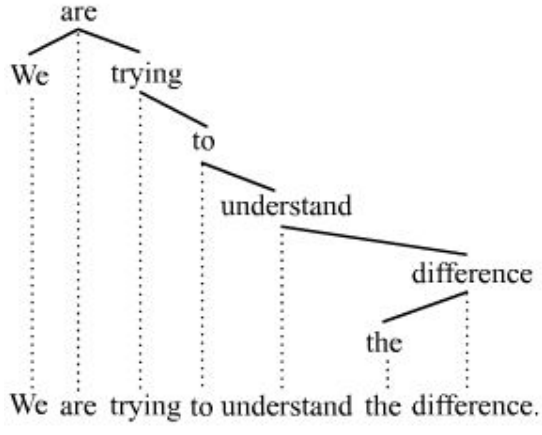
Что может дать синтаксис?

1. Разным последовательностям *мама мыла раму; мама раму мыла; раму мама мыла* соответствуют одинаковые синтаксические деревья?
2. Можно ли различить последовательности *карта заблокирована; карту заблокируйте* после лемматизации, используя синтаксические деревья?

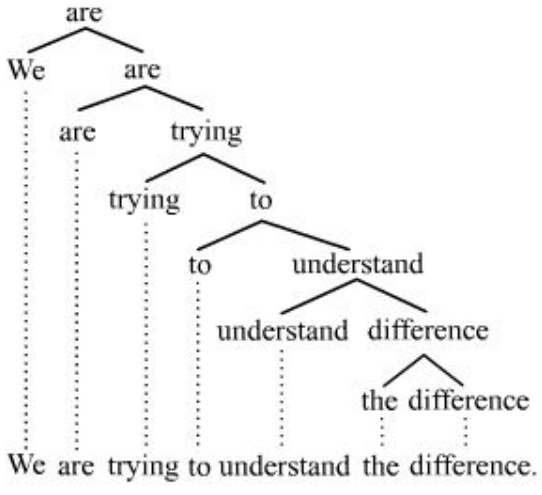
Модели построения разбора

Модели построения разбора

1. Грамматика составляющих (constituency, phrases)
2. Грамматика зависимостей (dependency)



Dependency



Constituency (BPS)

Формальное определение: составляющие

S — линейно упорядоченное множество слов.

Система составляющих на **S** — множество **C** отрезков **S**.

C содержит **S** и каждое слово, входящее в **S**.

Любые два отрезка, входящие в **C**, либо не пересекаются, либо один из них содержится в другом.

Элементы множества **C** — составляющие.

Пример разбора: составляющие

S — исходное предложение

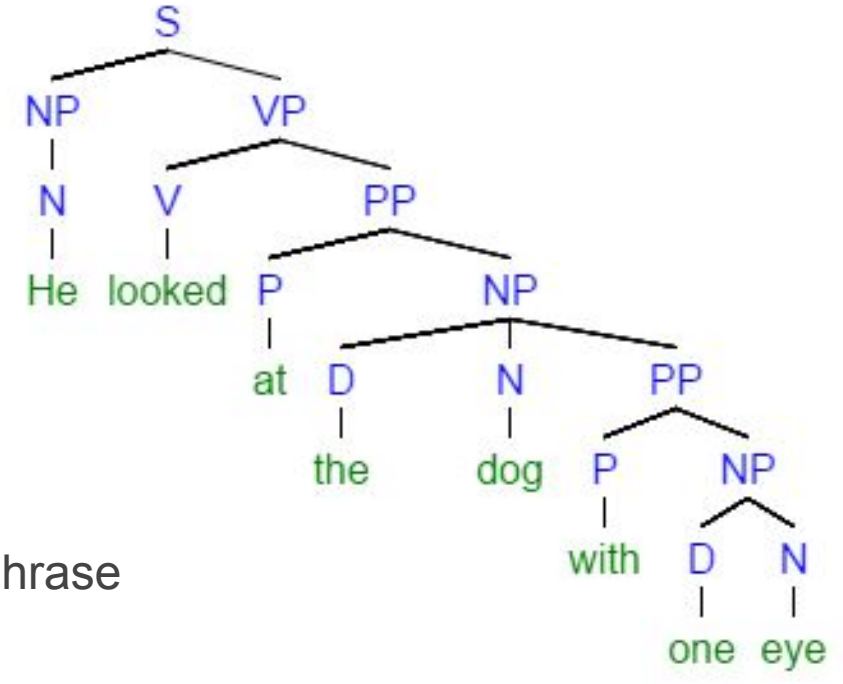
VP — глагольная группа, verb phrase (глагол + зависимые)

NP — именная группа, noun phrase (существительное)

PP — предложная группа, prepositional phrase

AP — группа прилагательного, adjective phrase

D (Det) — детерминативы (артикли и т.п.)



Что дают составляющие на практике

- Составляющие можно перемещать в рамках предложений:

  John talked [to the children] [about rules].

  John talked [about rules] [to the children].

  **John talked rules to the children about.*

- Составляющие можно заменять на похожие:

  I sat [on the box / on top of the box / in front of you].

Резюме: составляющие

- популярен в лингвистике
- лучше описан в “учебной” литературе
- плохо применим для языков, в которых может быть произвольный порядок слов (например, для русского языка)
- часто описывается контекстно-свободными языками
- для построения разбора может использоваться алгоритм CYK (Cocke-Younger-Kasami)

Jurafsky and Martin, Speech and Language Processing (три главы)

Формальное определение: зависимости

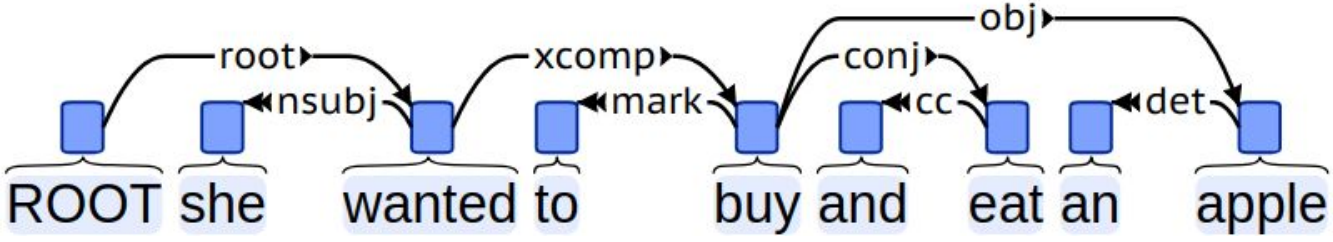
Дерево зависимостей — направленный граф, такой что:

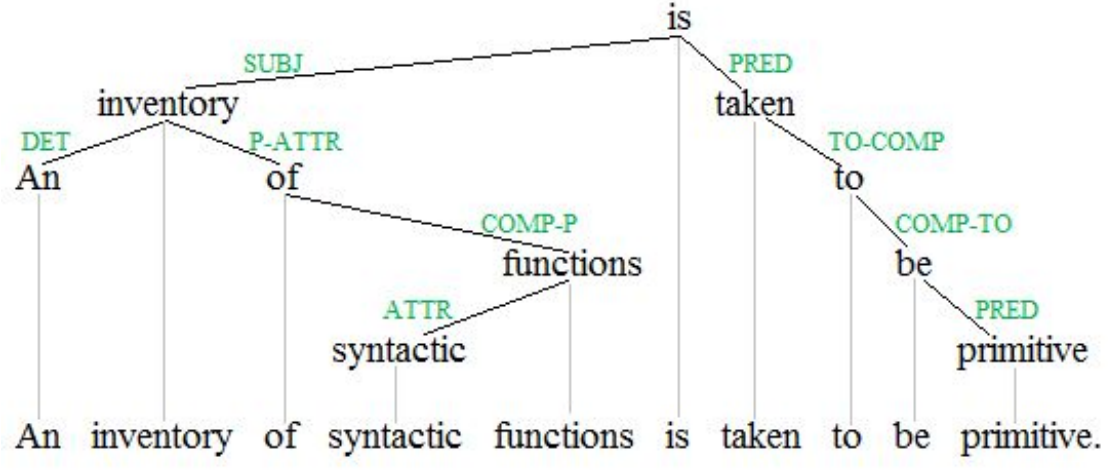
- граф является деревом
- вершины графа — слова и [ROOT]
- в каждую вершину, кроме [ROOT], входит одно ребро
- в [ROOT] не входит ни одно ребро

Рёбра дерева описывают зависимость одного слова от другого.

Рёбра могут иметь “тип” связи.

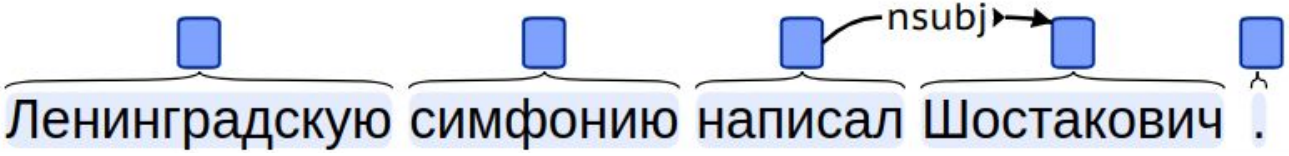
Пример разбора: зависимости

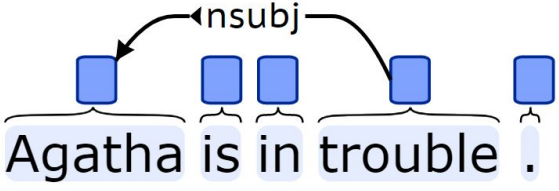




Пример связей: nsubj

Кто совершает действие? Кто субъект действия?

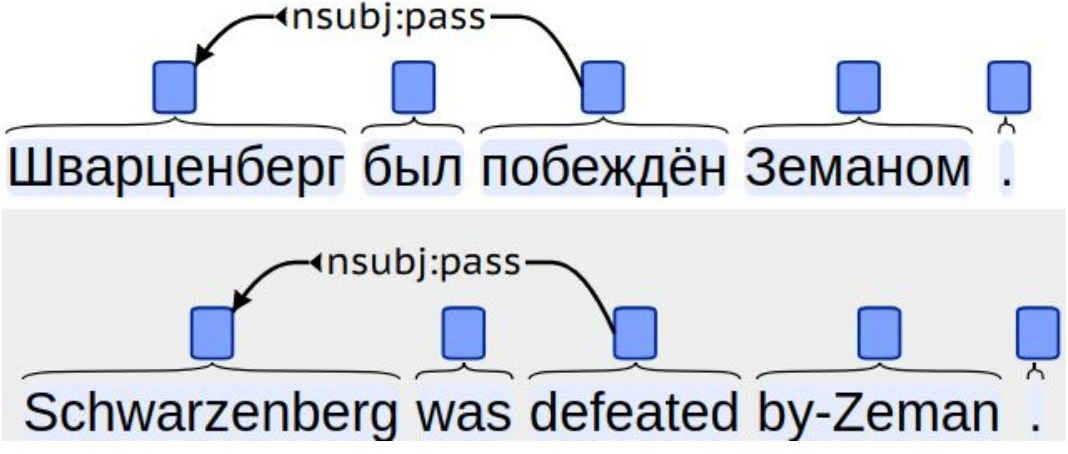




Пример связей: nsubj:pass

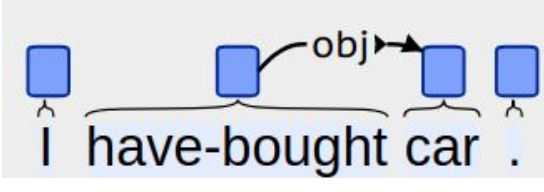
Кто совершает действие? Кто субъект действия?

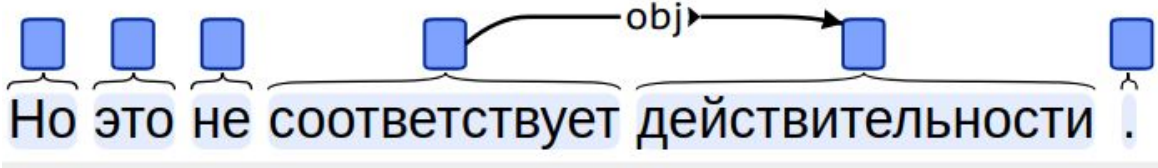
+ пассивный залог

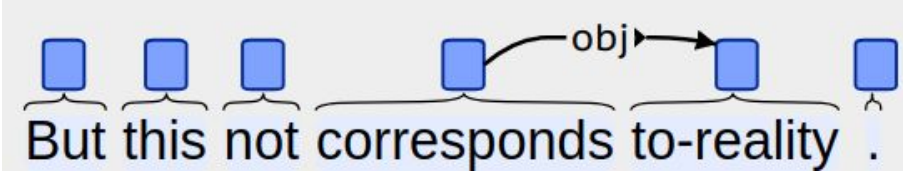


Пример связей: obj

Над кем совершают действие? Кто объект действия?

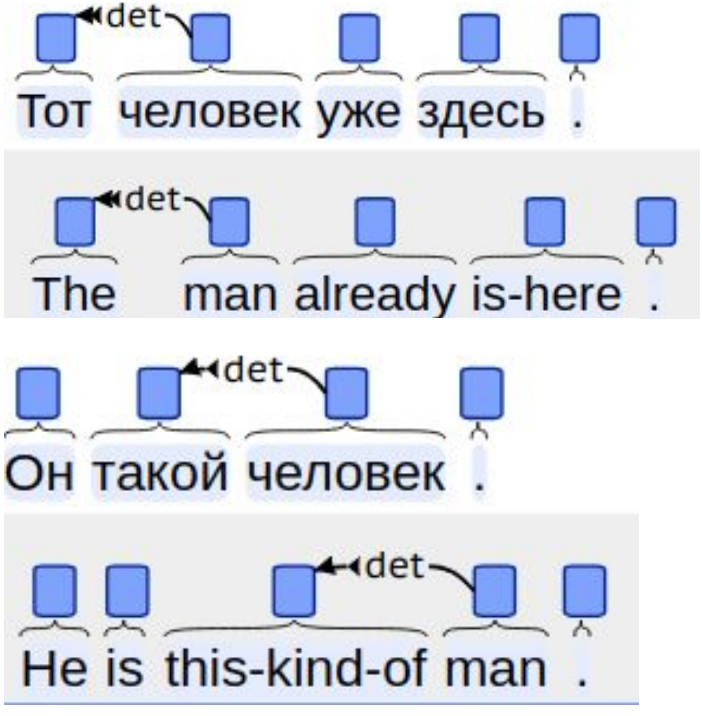






Пример связи: det

Связь объекта и указывающего местоимения



Свойство проективности

Предложение называется проективным, если:

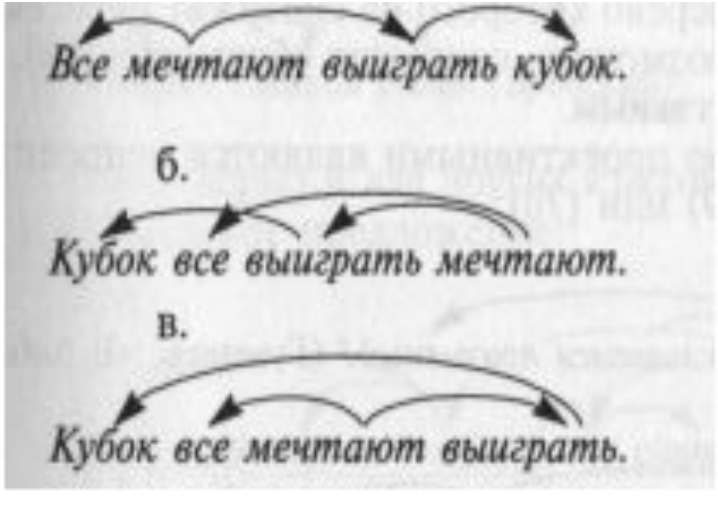
1. ни одна из стрелок не пересекает другую стрелку

2. нет стрелок накрывающих корневую

а. проективное

б. непроективное — нарушено 1

в. непроективное — нарушено 2



Зачем нужна проективность?

- Слова близкие друг к другу синтаксически, обычно близки по положению в тексте.
- Непроективность иногда несёт особую информацию, которая не будет содержаться в соответствующем проективном аналоге:

  Кубок все выиграть мечтают.

  Очень они хорошие были люди.

Составляющие vs зависимости

- хорошо разработанные в лингвистике теории
- можно считать взаимозаменяемыми
- нет главной и нет вторичной + есть другие теории
- есть неоднозначности у обеих:

  Он сам увидел их семью

  Он увидел их при помощи своих семи глаз

  Эти типы стали есть в цехе

  Эти люди решили начали есть в цехе

Построение дерева зависимостей

Данные для обучения

Хотим обучить алгоритм генерирующий по предложению его дерево зависимостей.

Для того, чтобы обучить алгоритм, нам нужна размеченная выборка: предложения и их разбор.

Удивительно, но для большого числа языков такие выборки (treebanks) можно найти: treebanks для разных языков.

Проект Universal dependencies

Лингвистическая проблема: несоответствие терминов и правил из грамматик зависимостей разных языков.

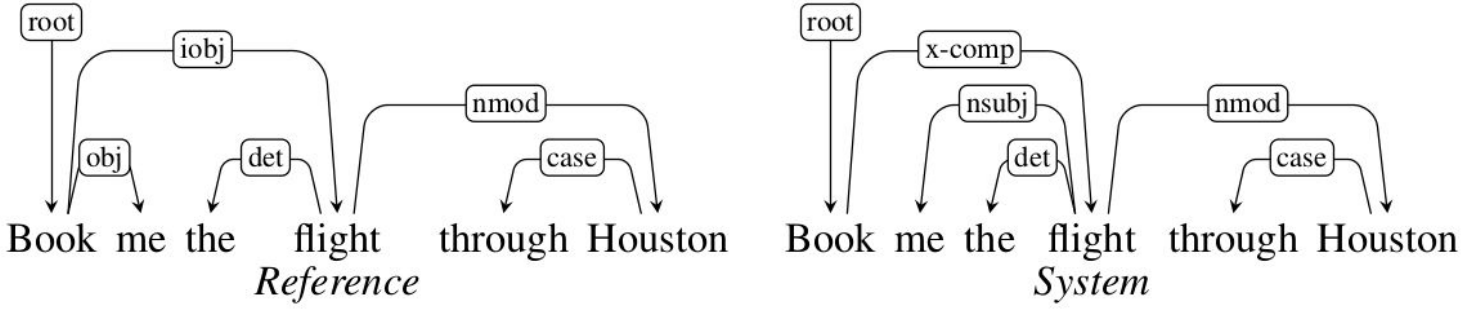
Data Science проблема: построить модель парсера, применимую для разных языков.

Решение: http://universaldependencies.org/

- 100 корпусов для 60 языков, все теги зависимостей унифицированы.

Метрики качества для построенного дерева

- Доля правильных разборов
- Unlabeled Attachment Score (UAS) — доля правильно угаданных рёбер
- Labeled Attachment Score (LAS) — доля правильно угаданных рёбер с правильным типом метки



Подходы построения дерева зависимостей

1. Transition-based — жадный способ построения дерева
2. Graph-based — полный поиск по всем возможным деревьям

Основная проблема: построить дерево

Предсказать метки по построенному дереву проще

(классификатор на каждую пару вершин по их признакам)

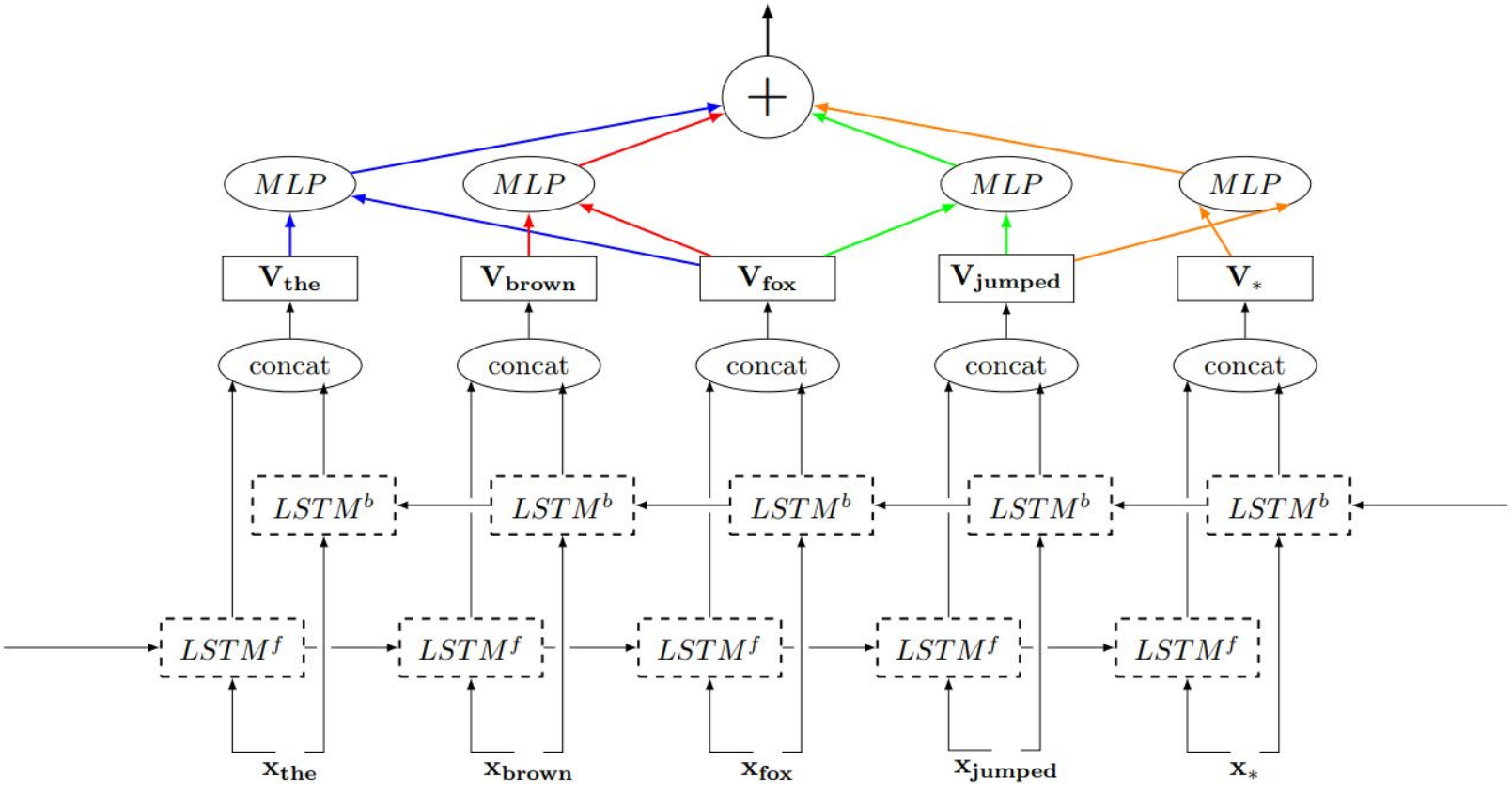
Graph-based подход

Для каждой пары слов **h, m** в предложении **s** оцениваем, нужно ли их добавить в дерево. Например так:

$$v = BiLSTM(s)$$
$$score_{hw} = MLP(v_h \circ v_w)$$

Функция потерь для **s** и правильного дерева **y**:

$$\max\left(0, 1 - \max_{y' \neq y} \sum_{(h,m) \in y'} MLP(v_h \circ v_m) + \sum_{(h,m) \in y} MLP(v_h \circ v_m)\right)$$



Kiperwasser et al (2016); Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations

Применение graph-based подхода

Алгоритм применения:

1. Подсчитать оценки всех пар вершин
2. Выбрать максимальное остовное дерево
3. Если учитывать проективность, то (2) сложнее…

Особенности:

- хорошее качество (особенно на длинных предложениях)
- невысокая скорость (квадратичная сложность)

Transition-based подход: сущности

Пусть у нас есть:

- список токенов (изначально — всё предложение)
- стек (изначально — [ROOT])
- конфигурация — итоговый набор зависимостей (изначально — пустая)
- набор действий, которые могут менять три сущности

Хотим обучить систему выбирать последовательность действий, приводящую к правильной конфигурации.

Nivre (ACL 2003): An Efficient Algorithm for Projective Dependency Parsing

Transition-based подход: действия

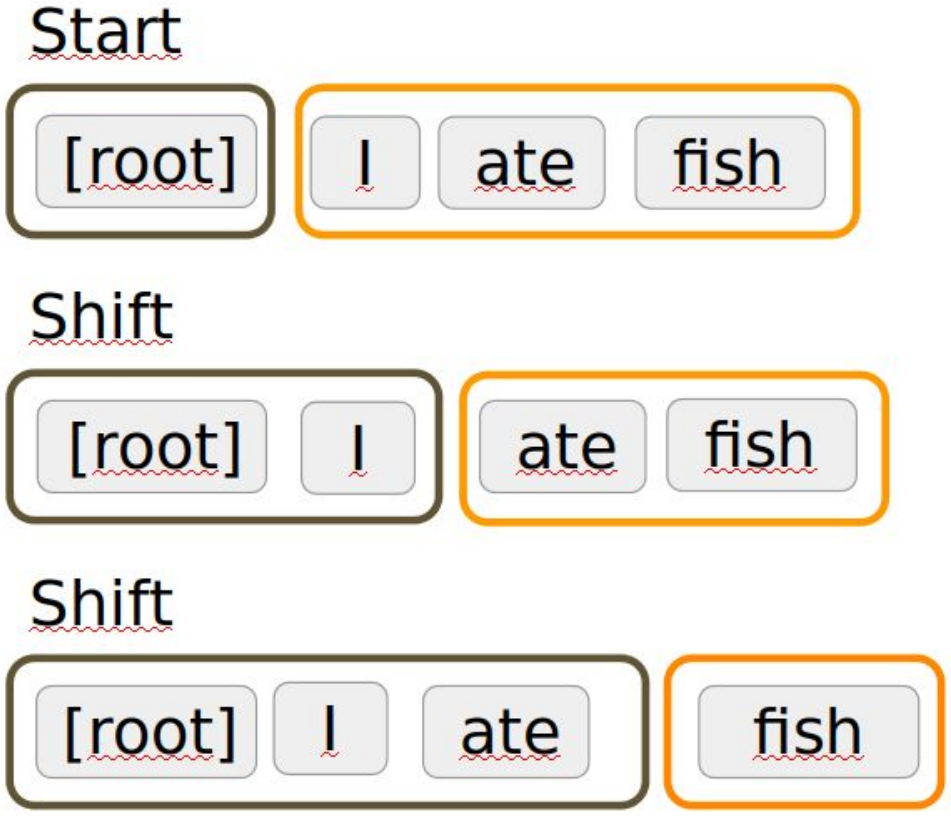
- **LeftArc** (если второй элемент стека не ROOT) — проводим зависимость от первого токена на верхушке стека к второму, и выкидываем второй из стека
- **RightArc** — проводим зависимость от второго токена на верхушке стека к первому, и выкидываем первый из стека
- **Shift** — переносим очередное слово из буфера в стек

В некоторых системах есть четвёртое действие:

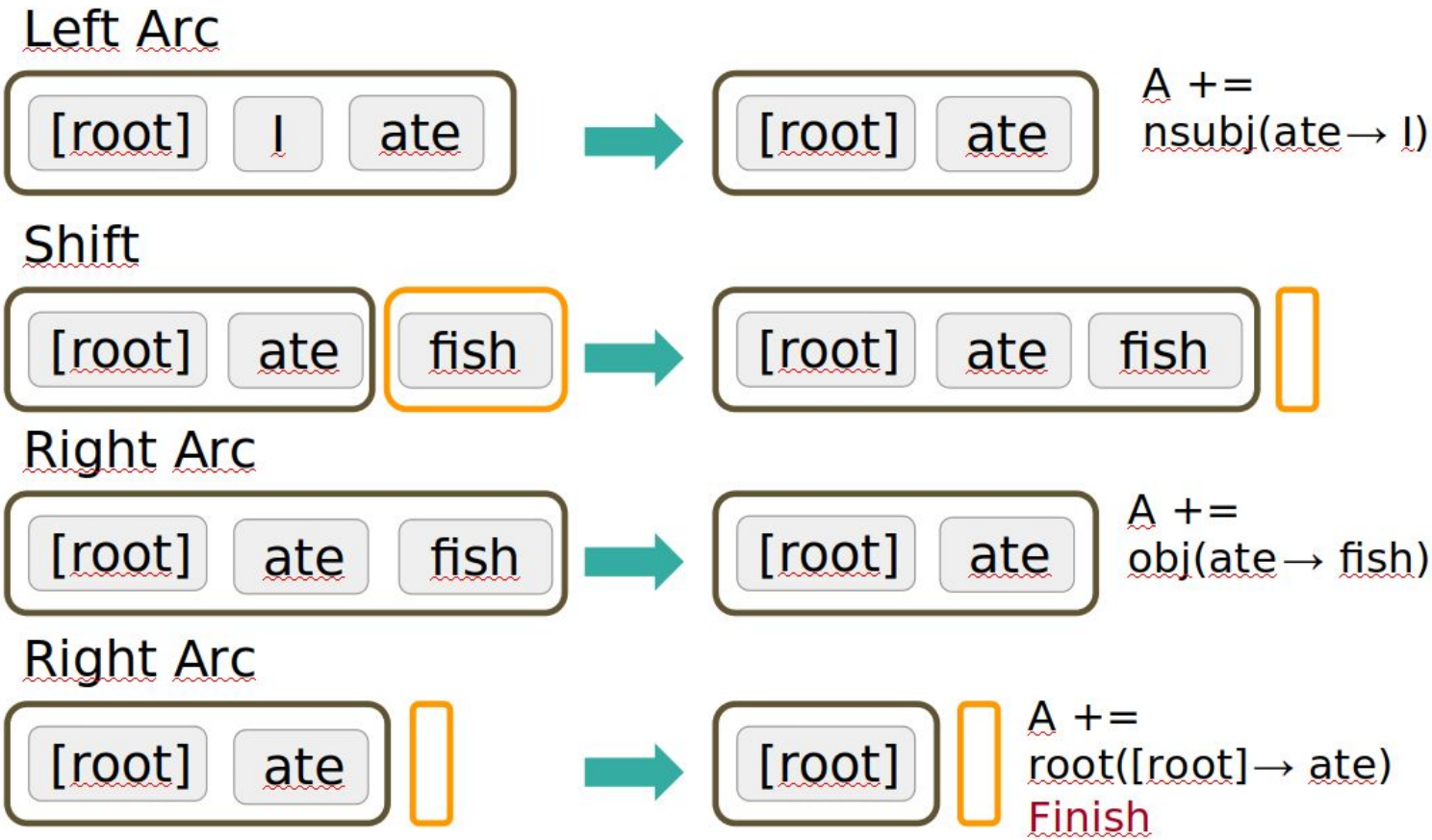
- **Swap** — вернуть второй элемент стека в буфер

Nivre et al (ACL 2009); An Improved Oracle for Dependency Parsing with Online Reordering

Пример работы на предложении “I ate fish”



Пример работы на предложении “I ate fish”



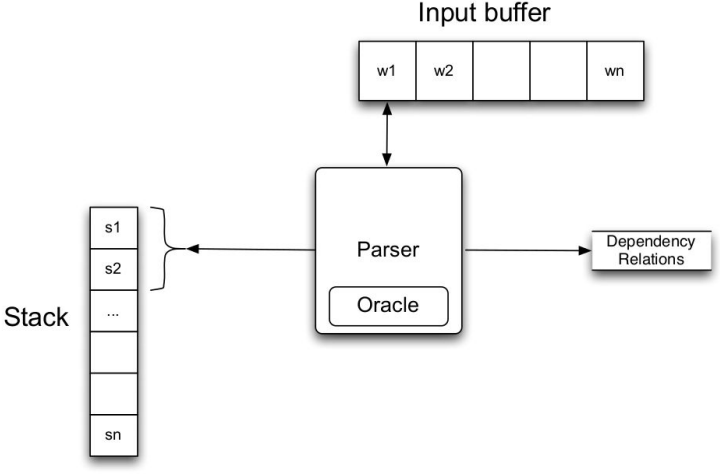
Пример работы на предложении “Book me the morning flight”

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	

Алгоритм применения модели на тесте

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*



Модификации transition-based подхода

Проблема: зависимые удаляются из стека сразу после того, как мы смогли приписать им вершину.

Но при этом у них могут быть свои зависимые…

Хорошая новость: алгоритм будет учится выкидывать их из стека в последнюю очередь.

Но можно модифицировать алгоритм!

Transition-based arc-eager parsing

- **LeftArc** (если второй элемент стека не ROOT) — проводим зависимость от токена на верхушке буфера к токену на верхушке стека, выкидываем верхушку стека
- **RightArc** — проводим зависимость от токена на верхушке стека к токену на верхушке буфера, добавляем в стек верхушку буфера
- **Shift** — добавляем в стек верхушку буфера
- **Reduce** (если уже есть связь, ведущая в вершину) — выкидываем верхушку стека

Пример работы arc-eager парсера

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

Что обучаем?

Классификатор действий

Признаки: стек, буфер, конфигурация

Первая система: 10^6-10^7 индикаторных признаков

Пример:

$$s1.w = \text{good} \wedge s1.t = \text{JJ}$$
$$s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

Классический нейросетевой парсер

Каждому слову соответствует вектор размерности 3d (вектора для слов, pos-тегов, dependency меток).

Входной вектор составляется по 18 словам:

- 3 верхних слова в буфере
- 3 верхних слова в стеке
- 2 ближайших ребёнка слева и справа двух слов стека
- 1 ближайший ребёнок слева и справа для первых детей слева и справа двух слов стека

Chen et al (ACL 2014): A Fast and Accurate Dependency Parser using Neural Networks

Архитектура сети

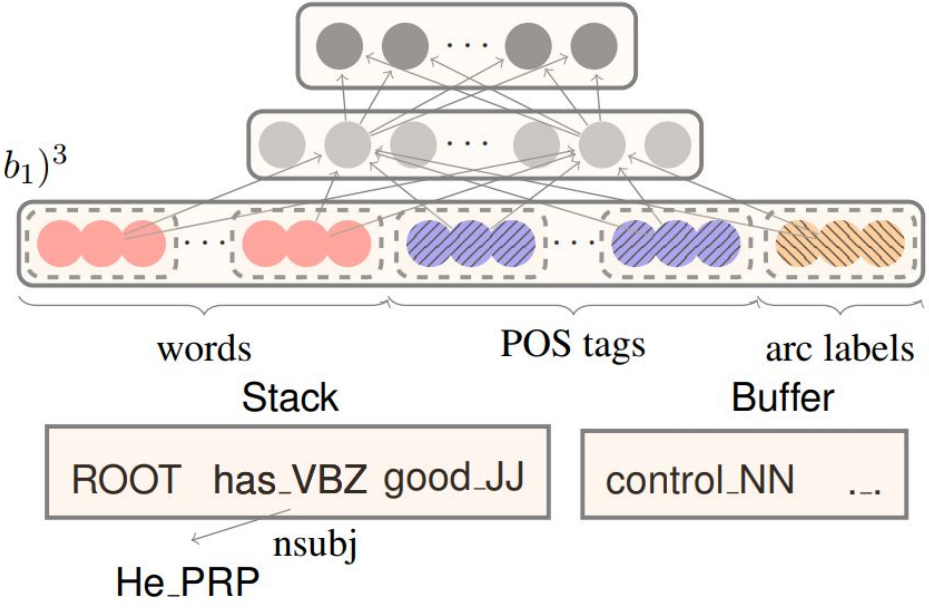
Softmax layer:

$$p = \texttt{softmax}(W_2 h)$$

Hidden layer:

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3$$

Input layer: $[x^w, x^t, x^l]$



Особенности обучения и применения

Функционал обучения: кросс-энтропия по действиям

Применение: жадная генерация действий + beamsearch

При обучении мы не учитываем способ применения:

- можно адаптировать разные трюки из предыдущих лекций: генерация действий при обучении, CRF и т.п.

Что использовать на практике: UDPipe

UDPipe — пайплайн, обучаемый токенизации, лемматизации, морфологическому тэггингу и парсингу, основанному на грамматике зависимостей.

Есть готовые модели (в том числе и для русского языка).

Для синтаксиса — парсер похожий на рассмотренный.

+ помните, что если подавать на вход не сырой текст, а обработанный другими теггерами/лемматизаторами, могут быть проблемы

Использование синтаксического анализа на практике

Анализ тональности для сущности

Тональность отзыва и отдельной сущности может различаться.
Хотим определять тональность конкретных сущностей.

1) Сопоставим сущности из списка покупок со словами.
2) Построим синтаксическое дерево для всех предложений.
   Выделим группу, в которую входит нужное нам слово.
3) Классифицируем только фразу с этим словом без привязки
   к остальному тексту.

Пример: отзывы из приложения доставки

Ценник выше среднего, а так вполне неплохо, правда рыба на филе оставляет желать лучшего.

Даю 2 звезды за то, что рис в роллах сварен правильно, качество сашими на высоте. Суп с морепродуктами это вода, абсолютно безвкусный и естественно холодный.

Заказывала горячие роллы, но привезли холодные. В салате цезарь не было помидоров, порции маленькие. Вообще роллы мне понравились, но больше заказывать не буду.

Information extraction

Information extraction (IE) — автоматическое извлечение структурированной информации из неструктурированного текстового источника.

Relation extraction (RE) — IE, где структура извлечённых данных задаётся триплетом (сущность 1, сущность 2, связь).

Relations	Types	Examples
Physical-Located	PER-GPE	**He** was in **Tennessee**
Part-Whole-Subsidiary	ORG-ORG	**XYZ**, the parent company of **ABC**
Person-Social-Family	PER-PER	**Yoko**'s husband **John**
Org-AFF-Founder	PER-ORG	**Steve Jobs**, co-founder of **Apple**...

Глава в Jurafsky, Martin про relation extraction

RE: модель разметки + шаблоны

1. Для каждого типа отношения задаются шаблоны, включающие сущности и слова.
2. Каждое предложение пропускается через алгоритм разметки (POS, NER) и сопоставляется с шаблонами

PER, POSITION of ORG:
George Marshall, Secretary of State of the United States

PER (named|appointed|chose|etc.) PER Prep? POSITION
Truman appointed Marshall Secretary of State

PER [be]? (named|appointed|etc.) Prep? ORG POSITION
George Marshall was named US Secretary of State

RE: модель разметки + классификатор

1. Применяем к предложению модель разметки.
2. Для каждой “связанной” пары определяем тип связи.

```
function FINDRELATIONS(words) returns relations

    relations ← nil
    entities ← FINDENTITIES(words)
    forall entity pairs ⟨e1, e2⟩ in entities do
        if RELATED?(e1, e2)
            relations ← relations+CLASSIFYRELATION(e1, e2)
```

Один из способов определения “связанности” — анализ синтаксического дерева.

Майнинг данных для обучения классификатора

Где взять такие специфичные данные для классификатора?

1. Пусть имеется множество размеченных триплетов для конкретного типа отношения.
2. Найдём все предложения с парами сущностей из триплетов
3. Каждое предложение преобразуем в шаблон.
4. Используя популярные шаблоны, найдём новые триплеты.
5. Повторяем шаги 2-4.

Преобраование в шаблон — самая сложная часть алгоритма.

Distant supervision для майнинга данных

Пусть у нас есть большая база триплетов (например, DBPedia).

Ищем все предложения, в которых сущности одного триплета связаны. Все такие предложения будем считать положительными примерами.

Где в RE может использоваться синтаксис?

1. задание сложных шаблонов
2. определение связности сущностей
3. unsupervised relation extraction без шаблонов
4. для признаков в классификаторе

Чем меньше обучающих данных, тем больше может быть полезен dependency parser.

Другие применения синтаксиса на практике

- Определение парафразов:

  «Карта заблокирована» vs «заблокируйте карту»

- Проверка качества при генерации текста.
- Аугментация данных (перестановка/удаление слов).
- Использование для задания весов слов (например, чем ближе к корню, тем больший вес).
- К полученному дереву можно применять графовые модели.

А какие проблемы?

- Разбор предложения — очень долгая и дорогая операция.
- Идеальные деревья получаются только на чистых текстах.
- Много нюансов при предобработке данных.
- Более простые методы часто не уступают в качестве.

Основной вывод: синтаксический анализ — не первое, что вы должны пробовать при решении задачи. Но при правильном применении, можно повысить качество решения.

Полезные ссылки

- лекция Дениса Кирьянова в ВШЭ
- лекция Маннига в Стэнфорде
- Глава в Juravsky, Martin про dependency parsing
- Глава в Jurafsky, Martin про relation extraction
- Об архитектуре парсера в Spacy + библиография;
- Программа воркшопа на EMNLP-18;
- Материалы курса на ESSLLI-18;
- J. Nivre’s workshop at EACL-2014;
- SyntaxRuEval-2012.