

Computational Syntax

Денис Кирьянов, Сбербанк,

denkirjanov@gmail.com,

21/03/2019

О лекторе

- Филфак СПбГУ, ФГН ВШЭ
(магистратура, комплингвистика)
- *“Мое дело”* — кастомный поисковик
- *Double Data* — поиск по людям в соцсетях и скоринг профилей
- *Angry Analytics* — мониторинг отзывов, сентимент, антиспам и др.
- *Сбербанк* — goal-oriented chat-bot

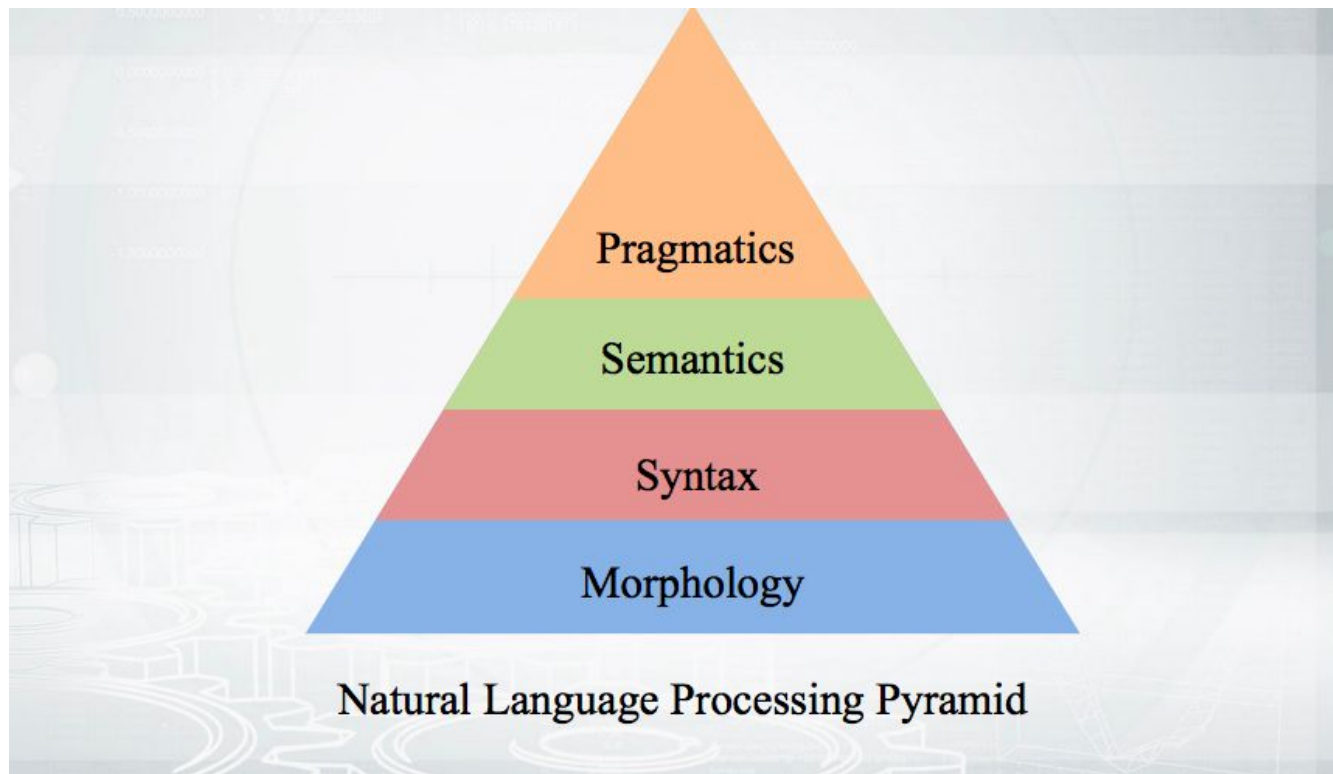
Disclaimers

- Практическое применение vs **теоретическое устройство парсеров**
- Ходите по [ссылкам!](#)
- Вопросы погромче :)

Содержание

1. **Что такое синтаксис и зачем он нужен**
2. Теоретические фреймворки
3. Dependency parsing
4. Метрики и соревнования
5. Инструменты
6. Varia

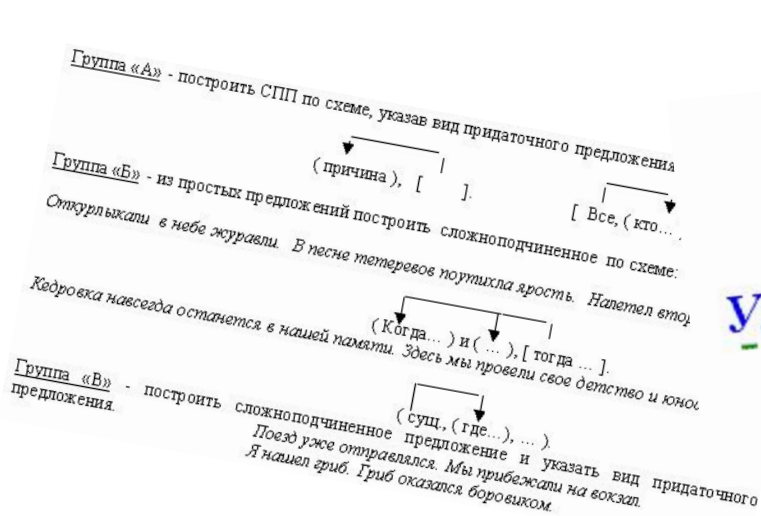
Где мы вообще?



Где мы вообще?

машинный анализ структуры текста, в т.ч. структуры предложения

Мы все это делали в школе, и машине это зачастую тоже под силу



Зачем это нужно?

- «Банкомат съел карту» vs «карта съела банкомат»;
- Определение правильности грамматики фразы (при порождении речи);
- Question answering;
- Машинный перевод;
- Information extraction (*напомни мне сделать икс*);
- Синтаксическая роль токена как метрика его важности (подлежащее важнее определения), использование весов в классификаторе.

Содержание

1. Что такое синтаксис и зачем он нужен
- 2. Теоретические фреймворки**
3. Dependency parsing
4. Метрики и соревнования
5. Инструменты
6. Varia

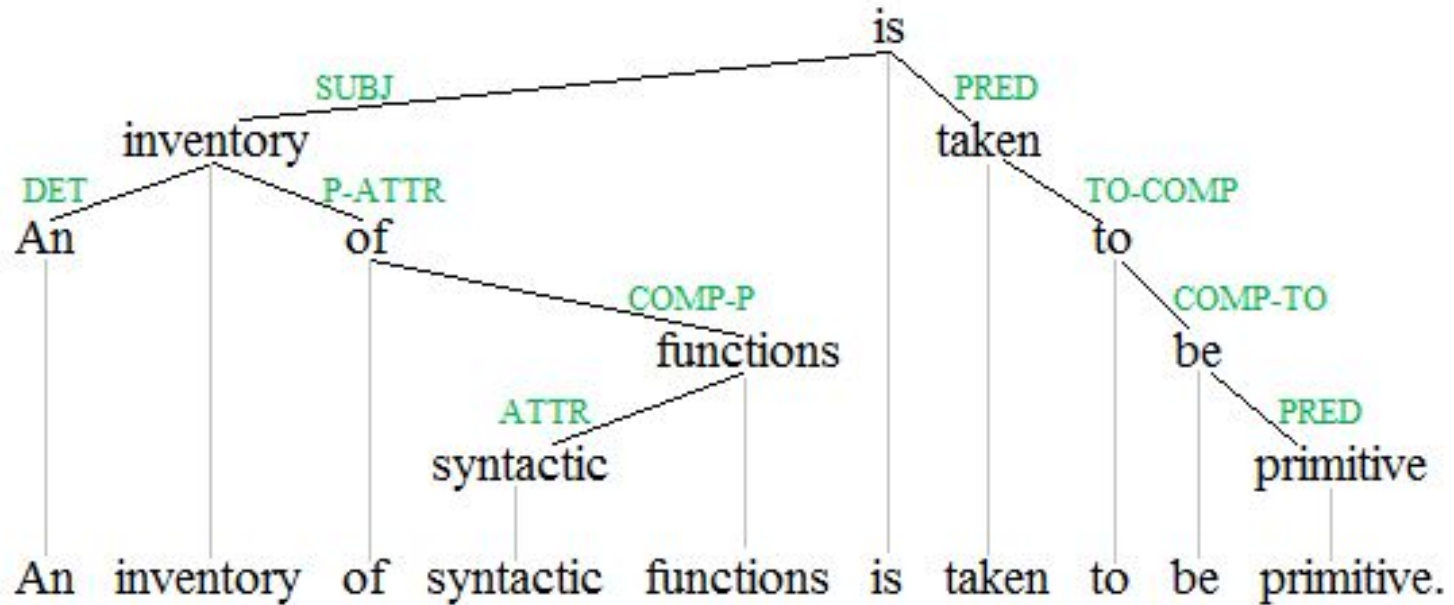
Грамматика непосредственно составляющих (constituency) и грамматика зависимостей (dependency)

●  Моя мама мыла грязную раму

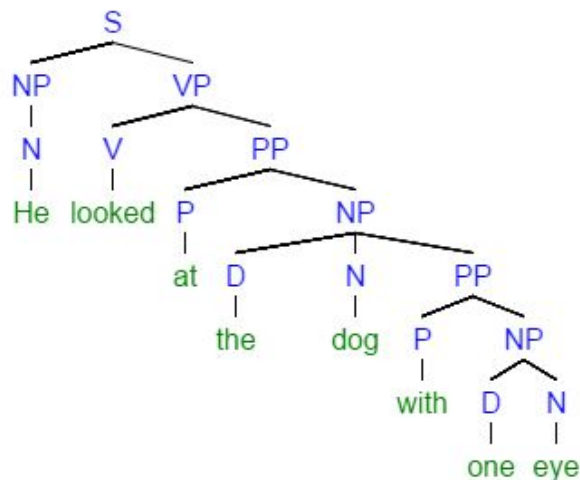
● [[моя мама] [мыла] [грязную раму]]

- В автоматическом парсинге для РЯ нет phrase-based парсеров (проблемы со “свободным” порядком слов);
- У остальных языков [получше](#).

Пример разбора: зависимости



Пример разбора: непосредственно составляющие



<https://i.imgur.com/ShMtNEy.png>

VP - глагольная группа, verb phrase

(грубо: состоит из глагола и зависимых;
но не подлежащее)

NP - именная группа, noun phrase

(грубо: вершина — существительное)

PP - предложная группа, prepositional phrase

AP - группа прилагательного, adjective phrase

D (Det) - детерминативы: артикли, указ., притяж.,
определятельные местоимения, квантификаторы,
числительные, вопросительные слова

...

Phrases vs Dependencies

- Хорошо разработанные в лингвистике теории;
- Отчасти (!) формально (!) взаимозаменяемые — грубо говоря, обе основаны на правилах взаимодействия частей речи;
- Нет главной и нет вторичной (хотя ГЗ слегка устарела);
- Как водится, много проблем на периферии у обеих, см. ([Тестелец 2001](#)).

А в чем проблемы?

- Эллипсис (aka пропуски);
- Синтаксическая омонимия

Он увидел их семью своими глазами

Он сам увидел их семью

Он увидел их при помощи своих семи глаз

ср. хрестоматийное “Эти типы стали есть в цехе”, “подпись руководителя группы или командированного лица”

- Непроективность

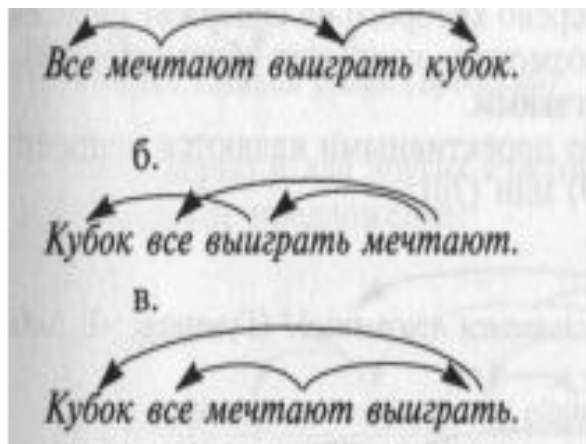
(Не)проективность (формализуя “нехорошее”)

Предложение называется проективным, если <...>:

а) Ни одна из стрелок не пересекает другую стрелку;

б) Никакая стрелка не накрывает корневую (~ сказуемое -> подлежащее)

[[Тестелец 2001](#): 95]



Проективное

НЕпроективное — нарушен принцип пересечения

НЕпроективное — нарушен принцип обрамления

Грамматика зависимостей

- Активное развитие в computational сфере;
- Лучше применима к парсингу русского языка;
- Всё не успеть за одну пару;
- Субъективный выбор лектора;
- Constituency parsing освещен в литературе, особенно см. [три главы учебника Журафского и Мартина](#).

Содержание

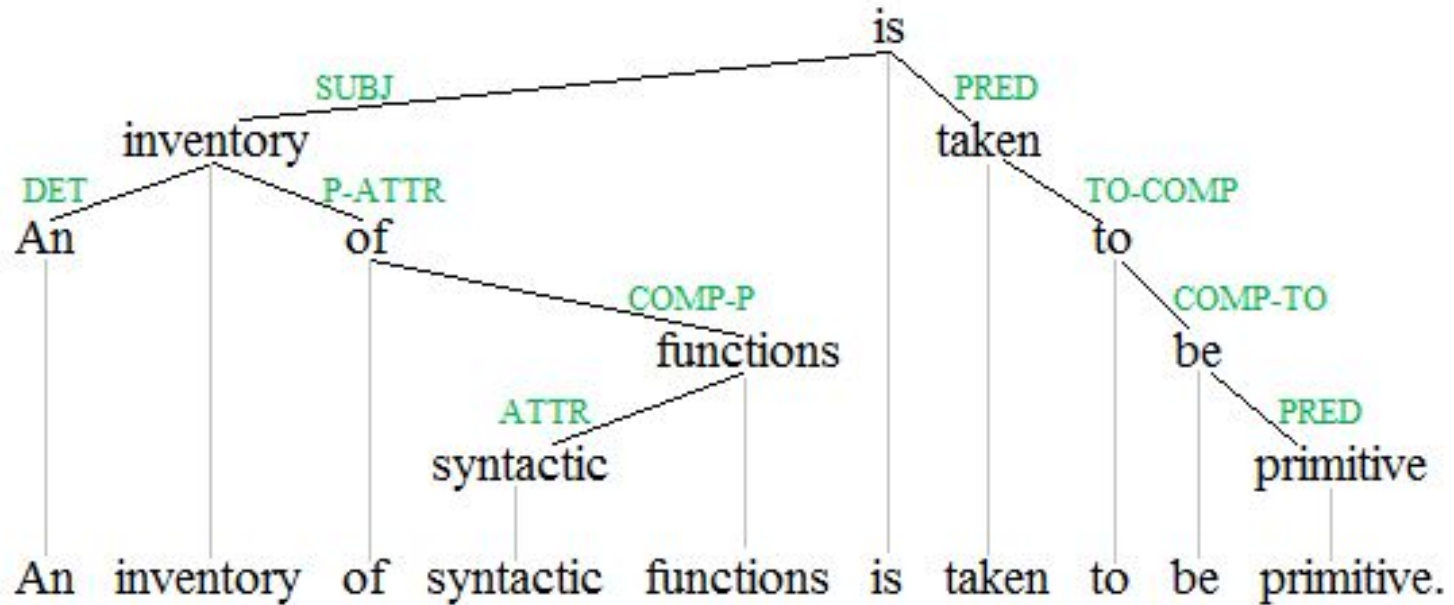
1. Что такое синтаксис и зачем он нужен
2. Теоретические фреймворки
3. **Dependency parsing**
4. Метрики и соревнования
5. Инструменты
6. Varia

Дерево зависимостей [[Jurafsky & Martin 2017](#)]

“Dependency tree is a directed graph that satisfies the following constraints:

- There is a single designated root node that has no incoming arcs.
- With the exception of the root node, each vertex has exactly one incoming arc.
- There is a unique path from the root node to each vertex in V .”

Пример разбора: зависимости



Dependency parsing: алгоритмы

построение дерева зависимостей по предложению

Два основных подхода

- **transition-based**

жадно набираем дерево (см. далее)

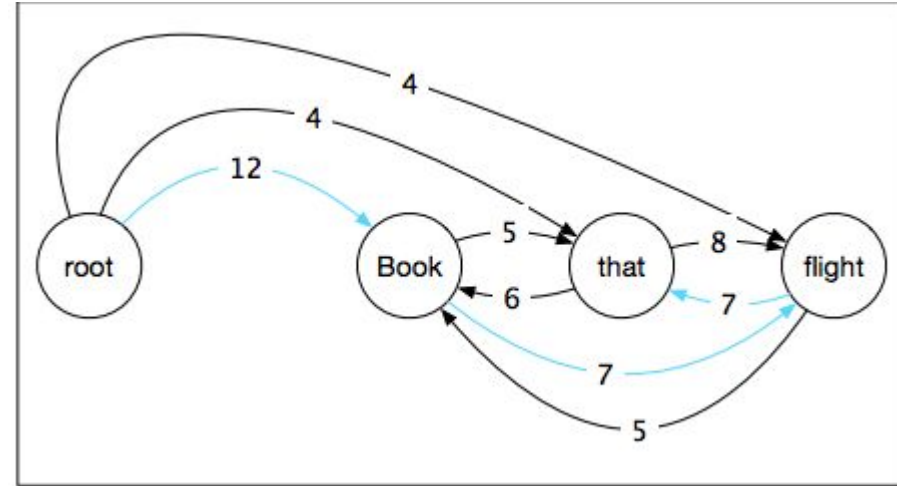
- **graph-based** (minimum spanning tree — минимальное остовное дерево)

ищем минимальное остовное дерево в полном графе всех возможных связей (быстрее работает при непроективности + обычно лучше работает с длинными предложениями)

$$\hat{T}(S) = \operatorname{argmax}_{t \in \mathcal{G}_S} \operatorname{score}(t, S)$$

Graph-based dependency parsing

- Изначально имеем полный орграф;
- Все ребра и все типы связей;
- При обучении учимся скорить связи;
- Фичи те же, что у transition-based;
- + могут быть фичи про порядок слов;
- Постпроцессинг: фильтр на циклы;



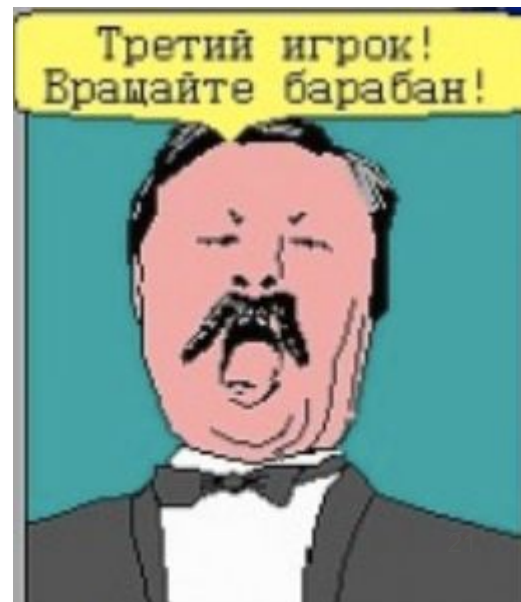
Проще справляться с непроективностью: без этого ограничения нам нужно меньше постпроцессинга, просто берем топ-кандидата, не отбирая именно топ-проективного

Transition-Based DP: интуиция

Представим, что нам Леонид Якубович открывает по одному слову, а мы строим разбор предложения на лету

Book...

Окей, что-то про книгу, книга может быть и подлежащим

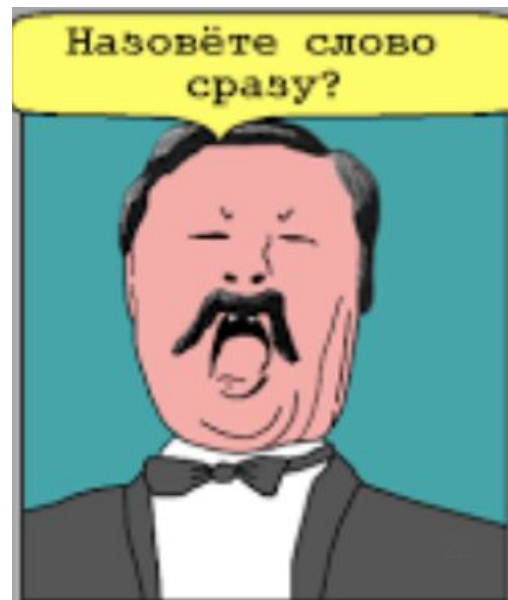


Transition-Based DP: интуиция

Представим, что нам Леонид Якубович открывает по одному слову, а мы строим разбор предложения на лету

Book me...

ан нет! “забронируй меня” или “забронируй мне”,
но **me** явно зависимое



Transition-Based DP: интуиция

Представим, что нам Леонид Якубович открывает по одному слову, а мы строим разбор предложения на лету

Book me the...

пока непонятно, но всё-таки это просьба
забронировать **что-то**



Transition-Based DP: интуиция

Представим, что нам Леонид Якубович открывает по одному слову, а мы строим разбор предложения на лету

Book me the morning...

“забронируй мне утро?” странновато, конечно (парсеру зависимостей это м.б. и не важно), но — **morning** может зависеть от **book**:
“забронируй мне утро у стоматолога”



Transition-based dependency parsing: интуиция

Представим, что нам Леонид Якубович открывает
по одному слову, а мы строим разбор предложения на лету

Book me the morning flight.

А вот теперь всё ясно!



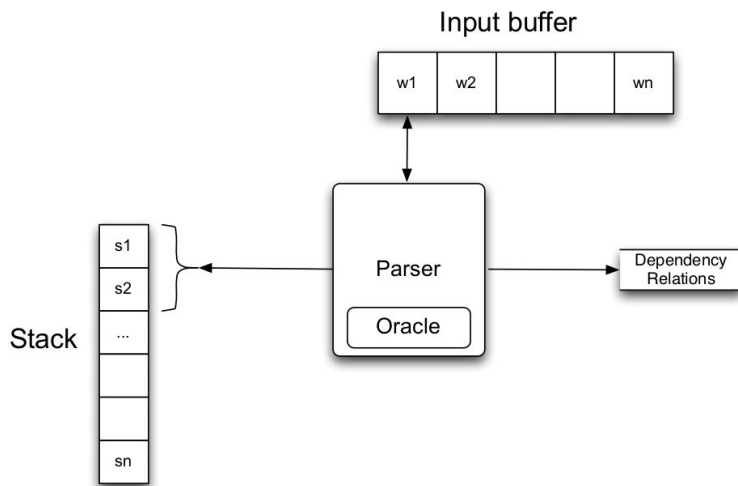
Transition-based (arc-standard) dependency parsing

Есть список токенов, стек (изначально содержит только root) и конфигурация (изначально пустая). Три дефолтных способа изменить конфигурацию:

- **LeftArc** [применим, если второй элемент стека не ROOT]
проводим зависимость между токеном на верхушке стека и вторым + выкидываем второй из стека
- **RightArc**
то же, но зависимость в другую сторону, и выкидываем верхушку стека
- **Shift**
переносим очередное слово из буфера в стек
- + **Swap** вернуть второй элемент стека в буфер, см. [[Nivre 2009](#)]

Transition-based dependency parsing

Ключевое понятие: “**конфигурация**” = **состояние** процесса разбора:
входящие токены, верхушка стека и набор уже построенных отношений
(то, что мы “держали в уме”, когда играли; да, аналогия не вполне точна)



Потому и transition-based -- мы сейчас будем **переходить** из состояния в состояние системы по правилам

Псевдокод

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

state \leftarrow { [root], [*words*], [] } ; initial configuration

while *state* **not final**

t \leftarrow ORACLE(*state*) ; choose a transition operator to apply

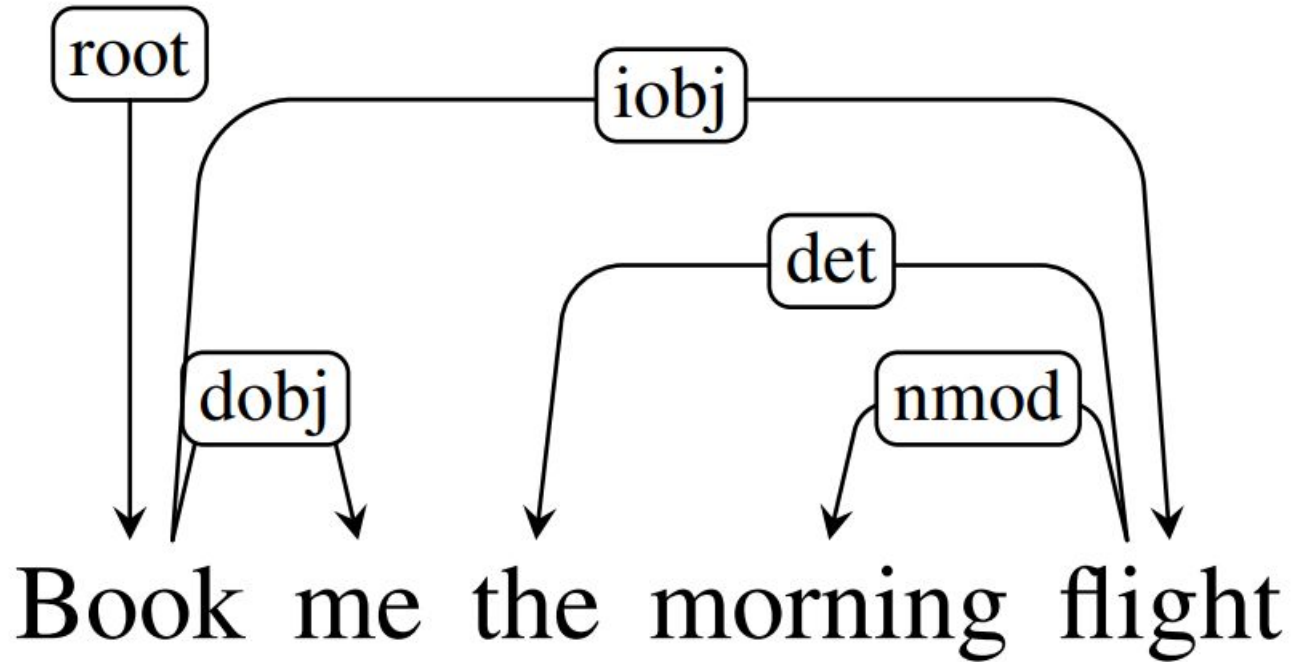
 state \leftarrow APPLY(*t*, *state*) ; apply it, creating a new state

return *state*

Пример работы

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

Пример работы [[Jurafsky & Martin 2017](#)]



Transition-based parsing

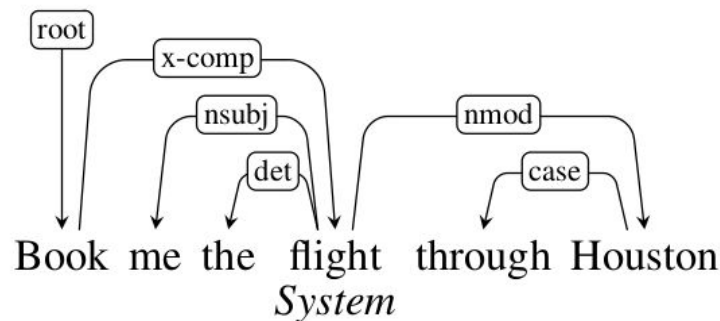
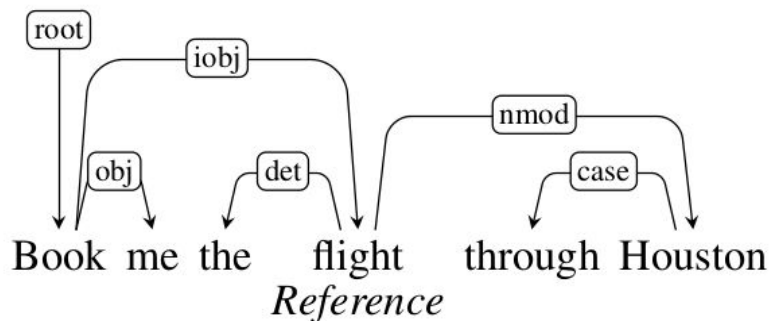
Есть разные модификации:

- arc-eager parsing (проводим ребра сразу)
- non-adjacent arc transitions (можно проводить ребра между несмежными элементами)
- еще бывает beam search для того, чтобы не быть настолько жадными (иногда это тоже вызывает проблемы)
- etc...

Содержание

1. Что такое синтаксис и зачем он нужен
2. Теоретические фреймворки
3. Dependency parsing
4. **Метрики и соревнования**
5. Инструменты
6. Varia

Оценка качества



Unlabeled Attachment Score (UAS) = 5/6

(правильно приписана вершина)

Labeled Attachment Score (LAS) = 4/6

(правильно приписана вершина И тип метки)

+ macro-averaged vs micro-averaged (по предложениям vs независимо от предложений)

И что, берем accuracy?

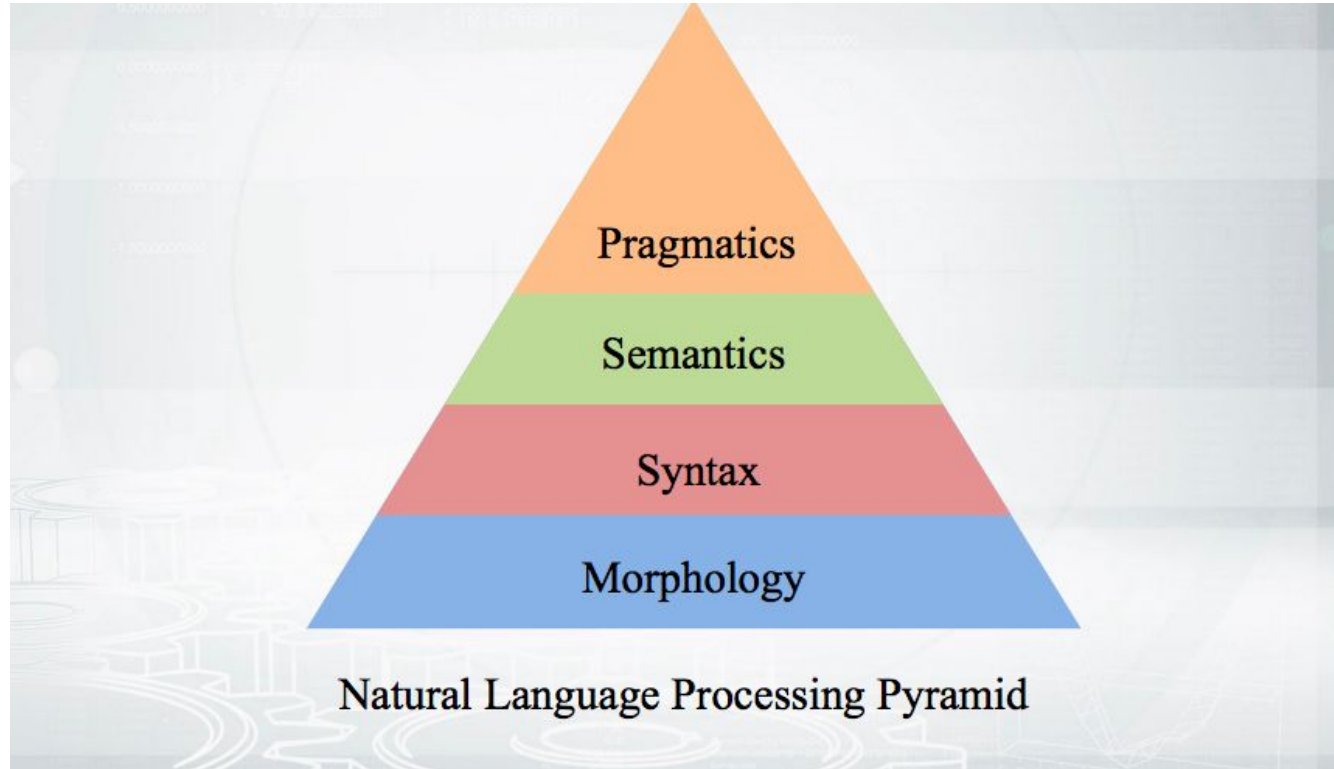
Проект Universal dependencies

- Лингвистическая проблема: несоответствие терминов и правил из грамматик зависимостей разных языков;
- Computational challenge: обучить синтаксический парсер для многих языков, включая low-resource languages;
=> <http://universaldependencies.org/>
- > 100 версионированных трибанков (размеченных корпусов) для 60 языков, теги зависимостей [унифицированы](#).

Соревнование пайплайнов

- [Conll 2017 Shared Task](#) “from raw text to dependencies”
- 81 трибанк, 49 языков;
- Парсинг сырого текста **vs** брать бейзлайн токенизацию и/или морфологию;
- Абсолютно лучший результат среди всех команд и всех трибанков показан на корпусе РЯ: 94% UAS и 92,6% LAS.

From raw text to dependencies



Соревнование пайплайнов: к дискуссии о метриках

- [Conll 2017 Shared Task](#)
- F-мера!
- Точность = количество точных попаданий/количество предсказаний;
- Полнота = количество точных попаданий/количество связей в размеченных данных;
- От чего зависит полнота?

Соревнование пайплайнов: к дискуссии о метриках

- При идеальной токенизации точность совпадает с полнотой, а значит:

$$F = 2PR/(P+R) = 2*x*x/(x+x) = x \text{ (=точность=полнота=accuracy)}$$

- При неидеальной токенизации значение F-меры меняется (точность != полнота);
- “Синтаксическая” метрика зависит от токенизации.

Соревнование пайплайнов: к дискуссии о метриках











- Не очень честное сравнение пайплайнов (кто-то мог не делать свою токенизацию);
- Отсутствие метрики, позволяющей чисто теоретически выделить идеальный сферический парсер в вакууме;
- Отсутствие единой метрики, позволяющей сравнить весь пайплайн целиком (from raw text to dependencies)
- Но как делать иначе — не очень понятно...

Соревнование пайплайнов: [Conll 2018 Shared Task](#)

- **LAS** (labeled attachment score) will be computed the same way as in the 2017 task so that results of the two tasks can be compared.
- **MLAS** (morphology-aware labeled attachment score) is inspired by the CLAS metric computed in 2017, and extended with evaluation of POS tags and morphological features.
- **BLEX** (bi-lexical dependency score) combines content-word relations with lemmatization (but not with tags and features).

Данные: русский язык

Russian treebanks

▶	SynTagRus	1,107K	(L)(F)(D)	 	
▶	PUD	19K	(F)	 W	
▶	GSD	99K	(L)(F)	W	
▶	Taiga	20K	(L)(F)	  	

See [here](#) for comparative statistics of Russian treebanks.

=> Практически все эксперименты проводятся на синтагматическом

(корпуса для всех языков [лежат на гите](#))

Содержание

1. Что такое синтаксис и зачем он нужен
2. Теоретические фреймворки
3. Dependency parsing
4. Метрики и соревнования
- 5. Инструменты**
6. Varia

Инструменты

- См. результаты дорожек [Conll-17](#) и [Conll-18](#);
- Осторожно: есть академические и закрытые разработки;
- UDPipe;
- Syntaxnet (оба — transition-based, но не обманывайтесь: например, в дорожке-17 на русском языке их обошел [graph-based](#) парсер, после чего UDPipe и некоторые другие поменяли архитектуру).

UDPipe vs Syntaxnet

	UDPipe (2.0)	Syntaxnet (parseysaurus-17)
UAS (russian, syntagrus)	92.96%	92.67%
LAS (russian, syntagrus)	91.46%	88.68%
Время парсинга одного предложения	~ 3 ms	~ 100 ms
Возможность “распилить” пайплайн	+	-
Запуск напрямую без докера и др.	+	-

UDPipe

- UDPipe — пайплайн, обучаемый токенизации, лемматизации, морфологическому тэггингу и парсингу, основанному на грамматике зависимостей;
- [Статья об архитектуре](#), [репозиторий с кодом обучения](#), [мануал](#);
- Есть готовые [модели](#) (в том числе и для РЯ);
- Подобранные для каждого корпуса параметры обучения [зарелизены](#).

UDPipe: архитектура

- **Совместное деление на слова и предложения:**
однослойная двухсторонняя GRU, для каждого символа предсказывающая, последний ли он в предложении и/или токене.
- **Теггер:** по последним четырем символам каждого слова генерируем триплеты (UPOS, XPOS, FEATS), при помощи перцептрона выбираем лучшего кандидата.

UDPipe: архитектура

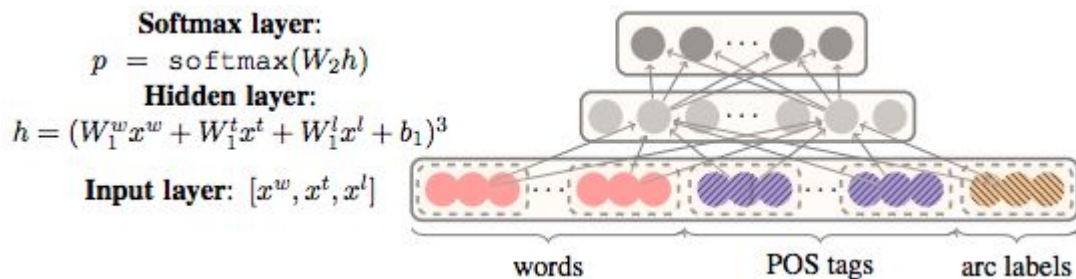
- **Лемматизатор:** Генерируем пары (lemma rule, UPOS), лемму предсказываем, отрезая префиксы и суффиксы и генерируя новые на их место. Перцептрон выбирает наилучшего кандидата.
- **Раздельное предсказание тегов и лемм** (2 модели, но можно соединить в одну);
- + можно подключить свой список лемм.

UDPipe: архитектура

- Dependency parsing ([[Straka et al. 2015](#)]): transition-based arc-standard dependency parser;
- Один скрытый слой, нет рекуррентности, см. картинку;
- Mini-batched SGD при обучении;
- До 18 источников фич на вход: 3 элемента на вершине стека, 3 элемента на вершине буфера, первый и второй левый и правый потомки 2 элементов на вершине стека, и самый левый и самый правый потомок 2 элементов на вершине стека;
- Можно загрузить pre-trained семантические эмбединги форм или лемм (см. *мышь ест стол*).

UDPipe: фичи для парсера

- Each node is represented using distributed representations of its form, its POS tag and its arc label; the latter only if it has already been assigned;
- word2vec-like training
- Network like in [\[Chen 2014\]](#):



Содержание

1. Что такое синтаксис и зачем он нужен
2. Теоретические фреймворки
3. Dependency parsing
4. Метрики и соревнования
5. Инструменты
6. **Varia**

Varia

- А что делать, если у языка нет обучающего корпуса?
 - Взять корпуса родственных языков, взболтать, перемешать, обучить [delexicalized model](#)
- А другая морфология (кроме POS-тегов) влияет?
 - *Глокая куздра штеко будланула бокра и курдячит бокрѣнка.*
 - Непохоже, см. [мою статью на хабре](#)
 - Но то, что влияет, — влияет на целевые примеры
 - Автор UDPipe [утверждает](#), что влияет :)

Varia

- А если у меня будет идеальная морфология, я смогу сделать идеальный парсер при помощи распространенного инструмента?
 - Попробуйте! Но вообще не в случае с русским, я писал на хабре.
- А что делать с веб-текстами, в них же все по-другому, включая пунктуацию?
 - А черт его знает... Нужно собирать корпус...

Полезные ссылки

- [Об архитектуре парсера в Spacy + библиография;](#)
- [Программа воркшопа на EMNLP-18;](#)
- [Материалы курса на ESSLI-18;](#)
- [J. Nivre's workshop at EACL-2014;](#)
- [SyntaxRuEval-2012;](#)
- [Дорожка по гэппингу на Диалоге-19.](#)

Заключение

- **Теоретические фреймворки:** ГЗ vs ГНС;
- **Dependency parsing:** minimum spanning tree (aka graph-based) vs transition-based подходы;
- **Метрики:** UAS, LAS + более сложные для from raw text to dependencies;
- **Соревнования:** SyntaxRuEval-12, CoNLL-17, CoNLL-18, dialogue-2019;
- **Данные:** корпуса с дорожек CoNLL (в формате conllu);
- **Инструменты:** Syntaxnet, UDPipe.

СПАСИБО ЗА ВНИМАНИЕ!

Денис Кирьянов, Сбербанк, denkirjanov@gmail.com

telegram: @kirdin

Автор благодарит за помощь в подготовке презентации
А. М. Алексеева, М. Ю. Князева и Е. Л. Артёмову.