Dependency grammar and dependency parsing

Francis M. Tyers

ftyers@hse.ru
https://www.hse.ru/org/persons/209454856
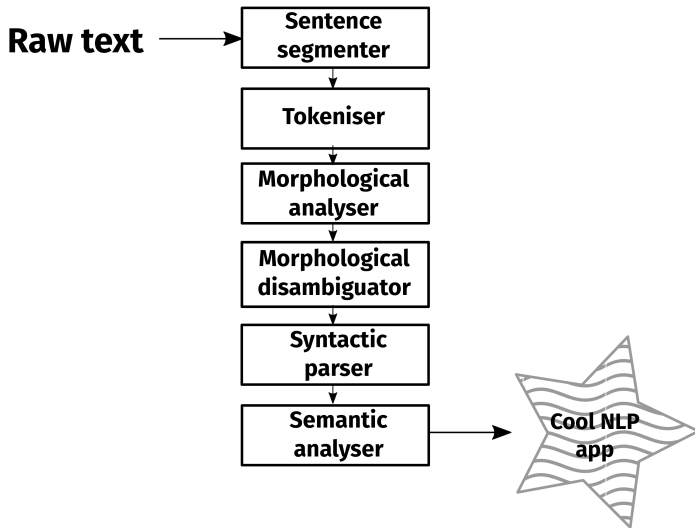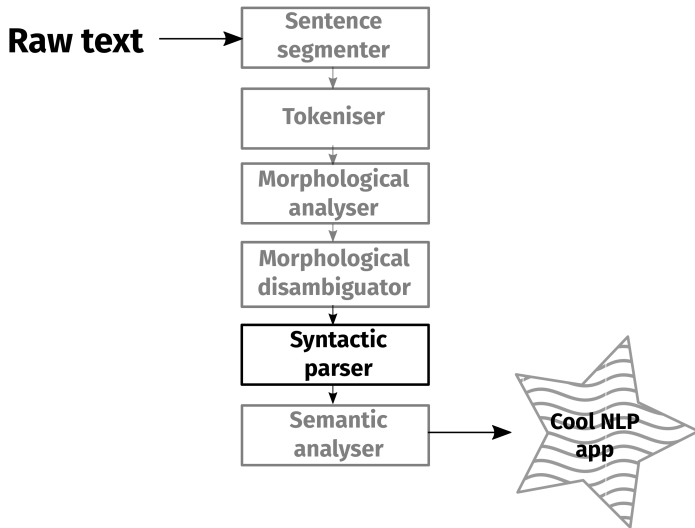
Национальный исследовательский университет
«Высшая школа экономики» (Москва)
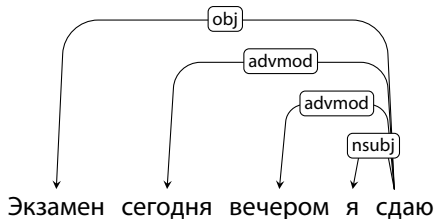
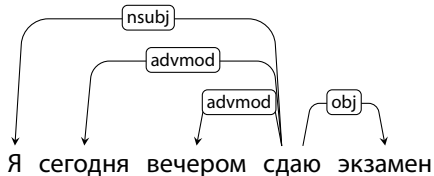27 марта 2018 г.

Motivating example

Сегодня я сдаю экзамен вечером
Я сегодня вечером сдаю экзамен
Экзамен сегодня вечером я сдаю

Bigrams
я сдаю, сдаю экзамен
вечером сдаю, сдаю экзамен
я сдаю, сдаю EOS

- Generalise over linear order
- Generalise long-distance

Motivating example

Bigrams
я сдаю, сдаю экзамен
Я сдаю, сдаю экзамен
я сдаю, сдаю экзамен

Dependency syntax
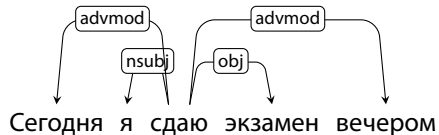
- Word based
- No non-terminals
- Words are linked by one-way binary relations
- Relations may be typed or untyped

Dependency structure

Они состоят из водяного льда с включениями органики .
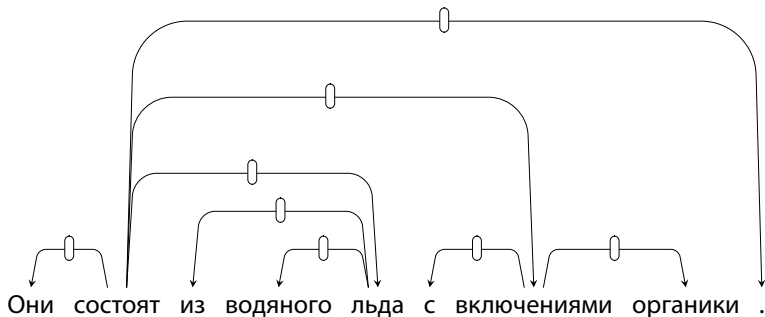
Dependency structure

Они состоят из водяного льда с включениями органики .

Dependency structure
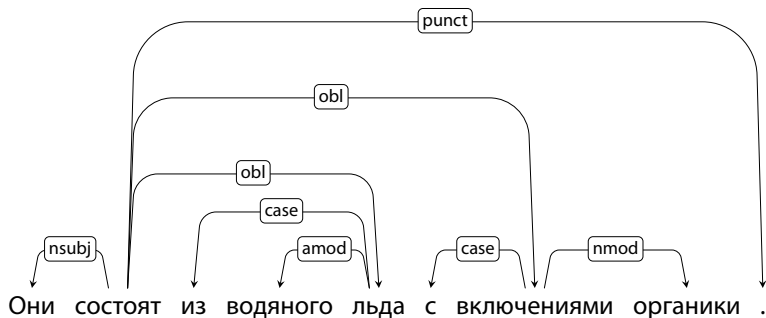
- Labels describe functional relations

Terminology

Superior	Inferior
Head	Dependent
Governor	Modifier
Regent	Subordinate
Mother	Daughter
Parent	Child

Notational variants
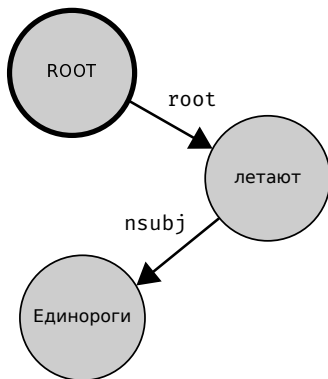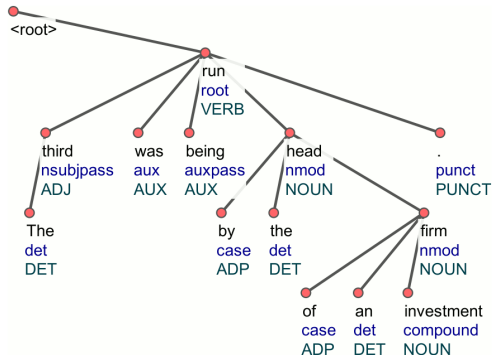
- Prague style
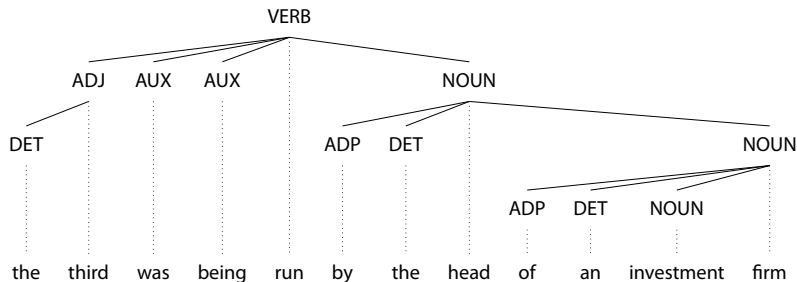
- «I wish I were a phrase-structure parse» style

- Most familiar style

Phrase structure

Comparison
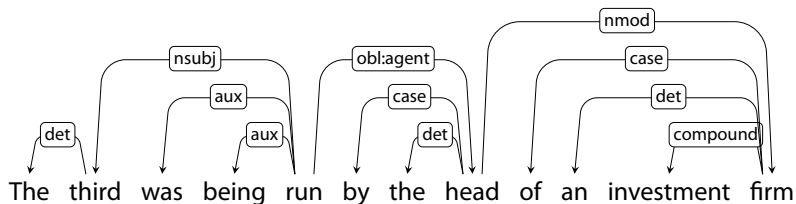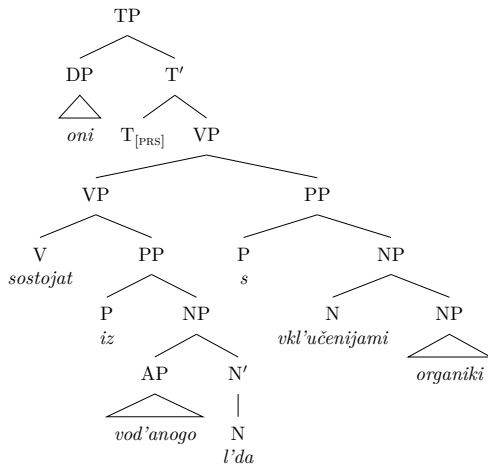
Dependency structures explicitly represent:

- head–dependent relations (directed arcs)
- functional categories (arc labels)

Phrase structures explicitly represent:

- phrases (non-terminal nodes)
- structural categories (non-terminal labels)

Heads and dependents

- Criteria for a syntactic relation between a head *H* and a dependent *D* in a construction *C* (Zwicky, 1985)[1]

  1. *H* determines the syntactic category of *C*; *H* can replace *C*
  2. *H* determines the semantic category of *C*; *D* specifies *H*
  3. *H* is obligatory, *D* may be optional
  4. *H* selects *D* and determines optionality of *D*
  5. The form of *D* depends on *H* (agreement or government)
  6. Linear position of *D* is specified with reference to *H*

- An issue:
  - Syntactic (and morphological) versus semantic criteria

---

[1] Zwicky, A. (1985) "Heads" *Journal of Linguistics*, 21:1–29

Some fuzzy cases

- **Complex verb groups** (auxiliary–main verb)
- Subordinate clauses (complementiser–verb)
- Coordination (coordinator–conjuncts)
- Adpositional phrases (adposition–nominal)
- Punctuation

(a) Умеют ли единороги летать ?

(b) Умеют ли единороги летать ?

Some fuzzy cases

- Complex verb groups (auxiliary–main verb)
- **Subordinate clauses** (complementiser–verb)
- Coordination (coordinator–conjuncts)
- Adpositional phrases (adposition–nominal)
- Punctuation

(a) Я знаю что ты придёшь — xcomp, mark

(b) Я знаю что ты придёшь — ccomp, sbar

- Complex verb groups (auxiliary–main verb)
- Subordinate clauses (complementiser–verb)
- **Coordination** (coordinator–conjuncts)
- Adpositional phrases (adposition–nominal)
- Punctuation

(a) Любишь чай и кошек ? (b) Любишь чай и кошек ?

- Complex verb groups (auxiliary–main verb)
- Subordinate clauses (complementiser–verb)
- Coordination (coordinator–conjuncts)
- **Adpositional phrases** (adposition–nominal)
- Punctuation

(a) Проходил я по лесам

(b) Проходил я по лесам

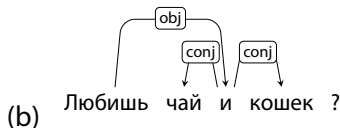- Complex verb groups (auxiliary–main verb)
- Subordinate clauses (complementiser–verb)
- Coordination (coordinator–conjuncts)
- Adpositional phrases (adposition–nominal)
- **Punctuation**

A dependency graph, *G*

- a set of *V* nodes, $V = \{0, 1, 2, 3, 4\}$
- a set of *A* arcs, $A = \{0 \rightarrow 2, 2 \rightarrow 1, 2 \rightarrow 4, 4 \rightarrow 3\}$
- a linear precedence order $<$ on *V*

Labelled graphs:

- Nodes in *V* are labelled with word forms (and annotation)
- Arcs in *A* are labelled with dependency types

Conditions

- **Connectedness**: The syntactic structure must be complete, every word must be covered by the structure
- **Acyclicity**: The structure must be hierarchical, no cycles,
- **Single-headedness**: Each word must have at most one head

Conditions

- **Connectedness**: The syntactic structure must be complete, every word must be covered by the structure
- **Acyclicity**: The structure must be hierarchical, no cycles,
- **Single-headedness**: Each word must have at most one head

Conditions

- **Connectedness**: The syntactic structure must be complete, every word must be covered by the structure
- **Acyclicity**: The structure must be hierarchical, no cycles,
- **Single-headedness**: Each word must have at most one head

Projectivity

- Projectivity = no crossing arcs
- Non-projective → more complex to parse

Graph-based

Transition-based

...

Transition-based

Components
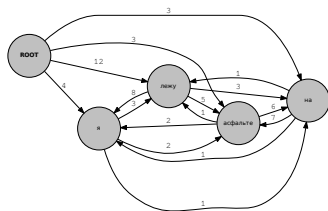
Data structures:
- Stack:
  - Starts as containing only the ROOT
- Buffer
  - Starts as containing the full sentence
- Arcs
  - Starts as empty

Operations:
- SHIFT: Take the word on top of the buffer and put it on the stack
- LEFT-ARC: Make the word at the top of the stack the head of the word below it
  - Then remove the word at the top
- RIGHT-ARC: Make the word second from top the head of the word above it
  - Then remove the second from top word

ROOT   Мы   пошли   домой

Stack   **Buffer**
ROOT   Мы пошли домой

SHIFT

ROOT Мы пошли домой

Stack   **Buffer**
ROOT Мы   пошли домой

SHIFT

ROOT Мы пошли домой

Stack	**Buffer**
ROOT Мы пошли	домой

LEFT-ARC
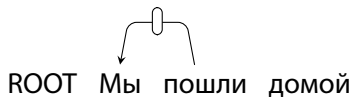
ROOT   Мы   пошли   домой

Stack   **Buffer**
ROOT пошли   домой

SHIFT

ROOT Мы пошли домой

Stack   **Buffer**
ROOT пошли домой

RIGHT-ARC

ROOT   Мы   пошли   домой
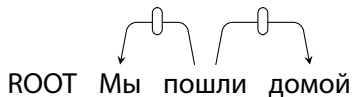
Stack   **Buffer**
ROOT пошли

RIGHT-ARC

ROOT Мы пошли домой
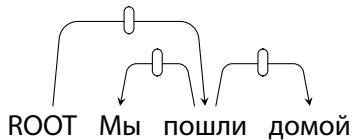
Stack   Buffer
ROOT

Configurations

A **configuration** is a snapshot of the state of the parser at a given time.

- A stack: Representing the word(s) currently being processed
- A buffer: Representing the remaining words
- A set of arcs representing a (partial) tree

We can conceive parsing as transitioning from one configuration to another via an operation.

How do we get the sequence of operations?

Deterministic algorithm:

- LEFT-ARC: Configuration has arc from the top of stack to the word below
- RIGHT-ARC: Configuration has arc from the of the stack to the first word in the input buffer
  - In addition: The dependent must have no dependents of its own
- SHIFT: All other cases

How do we get the sequence of operations?

Deterministic algorithm:

- LEFT-ARC: Configuration has arc from the top of stack to the word below
- RIGHT-ARC: Configuration has arc from the of the stack to the **first word in the input buffer**
  - In addition: The **dependent** must have no dependents of its own
- SHIFT: All other cases

ROOT Мы пошли домой

| **Stack** | **Buffer** |
| ROOT | Мы пошли домой |
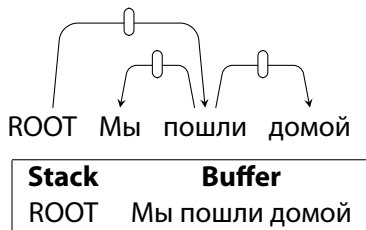
- Is there an arc from the first word in the buffer to the top of the stack?
  - (Мы, ROOT)
- Is there an arc from the top of the stack to the first word in the buffer?
  - (ROOT, Мы)
- $\rightarrow$ then SHIFT

Stack	Buffer
ROOT Мы	пошли домой
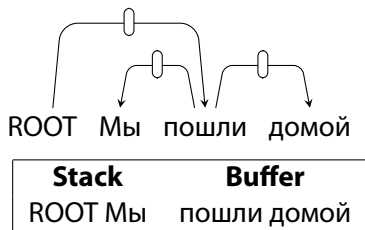
- Is there an arc from the first word in the buffer to the top of the stack?
    - (пошли, Мы) — YES, LEFT-ARC
- Is there an arc from the top of the stack to the first word in the buffer?
    - (Мы, пошли)

ROOT Мы пошли домой

Stack	Buffer
ROOT	пошли домой
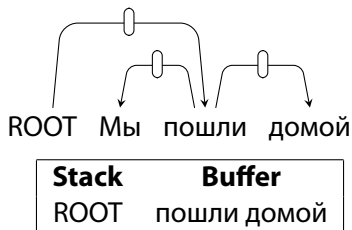
- Is there an arc from the first word in the buffer to the top of the stack?
    - (пошли, ROOT)
- Is there an arc from the top of the stack to the first word in the buffer?
    - (ROOT, пошли) – YES, but *пошли* still has dependents
- $\rightarrow$ then SHIFT
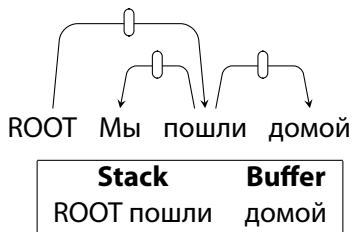
- Is there an arc from the first word in the buffer to the top of the stack?
    - (домой, пошли)
- Is there an arc from the top of the stack to the first word in the buffer?
    - (пошли, домой) – YES, and *домой* has no dependents
    - → RIGHT-ARC

The "only" training data required is a treebank.

- Collection of sentences annotated for dependency structure
- Universal dependencies: 67 languages, 100s of treebanks

Data trains a classifier to predict a transition from a configuration.

http://universaldependencies.org

|		Language		Size	
▸	🇿🇦	Afrikaans	1	49K	
▸		Ancient Greek	2	414K	
▸		Arabic	3	1,042K	
▸		Bambara	1	<1K	
▸		Basque	1	121K	
▸		Belarusian	1	8K	
▸		Bulgarian	1	156K	
▸		Buryat	1	10K	
▸		Cantonese	1	<1K	
▸		Catalan	1	531K	
▸		Chinese	4	153K	
▸		Coptic	1	11K	
▸		Croatian	1	197K	
▸		Czech	5	2,222K	
▸		Danish	1	100K	
▸		Dutch	2	310K	
▸		English	6	576K	
▸		Erzya	1	<1K	
▸		Estonian	1	106K	
▸		Finnish	3	377K	
▸		French	6	1,099K	
▸		Galician	2	164K	
▸		German	2	313K	
▸		Gothic	1	55K	
▸		Greek	1	63K	
▸		Hebrew	1	161K	
▸		Hindi	2	375K	
▸		Hungarian	1	42K	
▸		Indonesian	2	147K	
▸		Irish	1	23K	
▸		Italian	4	436K	
▸		Japanese	3	402K	

		Language		Size	
▸		Kazakh	1	11K	
▸		Korean	5	97K	
▸		Kurmanji	1	10K	
▸		Latin	3	491K	
▸		Latvian	1	90K	
▸		Lithuanian	2	5K	
▸		Maltese	1	2K	
▸		Marathi	1	3K	
▸		Naija	1	12K	
▸		North Sami	1	26K	
▸		Norwegian	3	625K	
▸		Old Church Slavonic	1	57K	
▸		Persian	1	152K	
▸		Polish	2	214K	
▸		Portuguese	3	570K	
▸		Romanian	2	239K	
▸		Russian	3	1,226K	
▸		Sanskrit	1	1K	
▸		Serbian	1	86K	
▸		Slovak	1	106K	
▸		Slovenian	2	170K	
▸		Spanish	3	1,004K	
▸		Swedish	3	195K	
▸		Swedish Sign Language	1	1K	
▸		Tagalog	1	<1K	
▸		Tamil	1	9K	
▸		Telugu	1	6K	
▸		Thai	1	23K	
▸		Turkish	2	74K	
▸		Ukrainian	1	100K	
▸		Upper Sorbian	1	10K	
▸		Urdu	1	138K	
▸		Uyghur	1	15K	
▸		Vietnamese	1	43K	

	Stack	**Buffer**
	ROOT пошли	домой
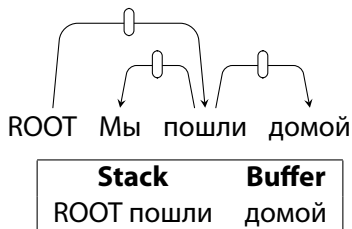
Features indexed by address (in stack or buffer) and attribute name.

Traditional:

- (Stack[0], Form) = пошли
- (Buffer[0], Form) = домой
- (Stack[0], UPOS) = VERB
- (Stack[1], Form) = ROOT

Can return NULL.

Indicator:

- Combinations of such features, e.g.
    - (Stack[0], Form) = пошли & (Buffer[0], UPOS) = ADP

Instead of all of those features, what do people do nowadays?

- Use embeddings
  - For words, POS tags, characters, features etc.

Why? Defining features is easy enough, but defining all those indicator features is tiresome.

- **Labelled parsing**: Instead of having three transitions, the LEFT-ARC and RIGHT-ARC transitions are expanded for the number of labels,
  - e.g. LEFT-ARC$_{nsubj}$
- **Non-projective parsing**: Add an extra transition which "swaps" adjacent nodes

Graph-based

Basic model
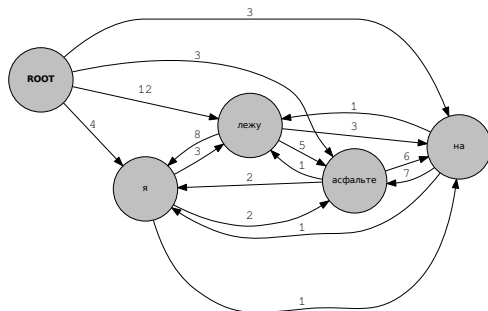
- Represent sentence as directed graph
- Score every edge
- Running the spanning tree algorithm

The old intuitions:

- transition based: greedy decisions
  - $\rightarrow$ worse on long distance deps
- graph based: global optimisation, but local indep assumptions

With neural methods, these apply less.

Where do the scores come from ?

- The score of a tree is sum of all arcs in the tree
- The score of an arc is the linear combination of features and weights

How about the features ?

- Similar to those used in transition-based parsing:
  - e.g. word form, lemma, POS of head and dependent
  - the dependency relation, direction of relation, etc.

Maximum spanning tree

- Higher score
- Contains all nodes
- Each node has at most one incoming edge
- Originates from a single, predefined root

Dense graph
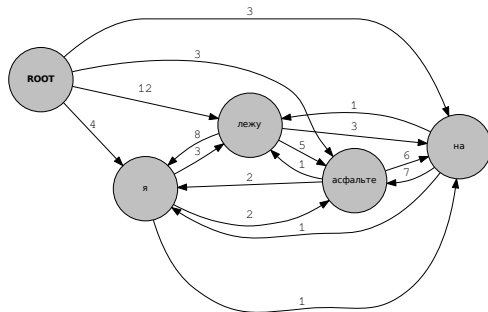
- Links between all nodes
- Except the root

- For each node in the graph
  - pick the incoming arc with the highest weight.
  - if this makes a tree → it's the MST
- Otherwise:
  - For each cycle
    - contract the cycle
    - find the incoming arc with highest weight
    - remove incompatible arcs in the cycle
  - repeat

function MaxSpanningTree($G=(V,E), root, score$) **returns** *spanning tree*

$F \leftarrow []$
$T' \leftarrow []$
$score' \leftarrow []$
for each $v \in V$ **do**
    $bestInEdge \leftarrow \text{argmax}_{e=(u,v) \in E} \, score[e]$
    $F \leftarrow F \cup bestInEdge$
    for each $e=(u,v) \in E$ **do**
        $score'[e] \leftarrow score[e] - score[bestInEdge]$

    if $T=(V,F)$ is a spanning tree **then return** it
    else
        $C \leftarrow$ a cycle in F
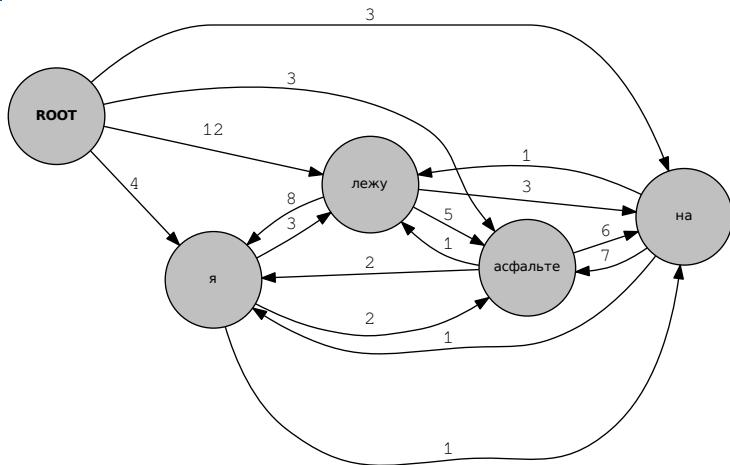 $G' \leftarrow \text{Contract}(G, C)$
        $T' \leftarrow \text{MaxSpanningTree}(G', root, score')$
        $T \leftarrow \text{Expand}(T', C)$
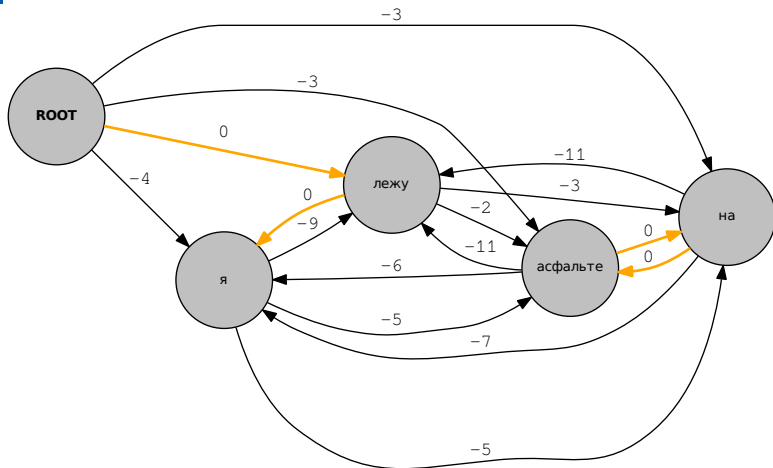   **return** T

function Contract(G, C) **returns** *contracted graph*

function Expand(T, C) **returns** *expanded graph*

Example

Example
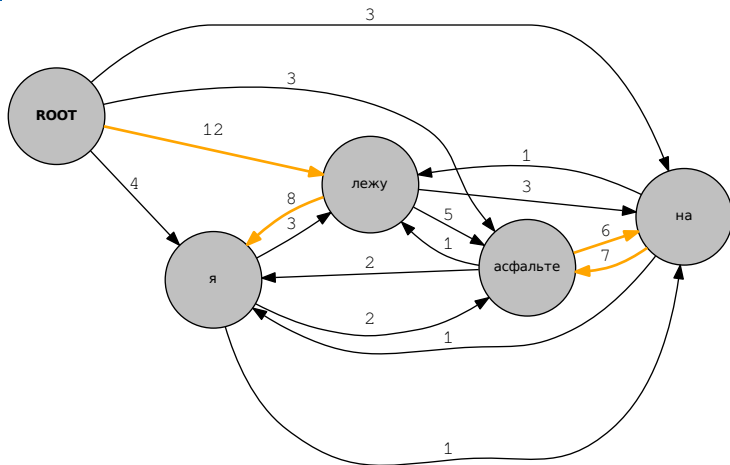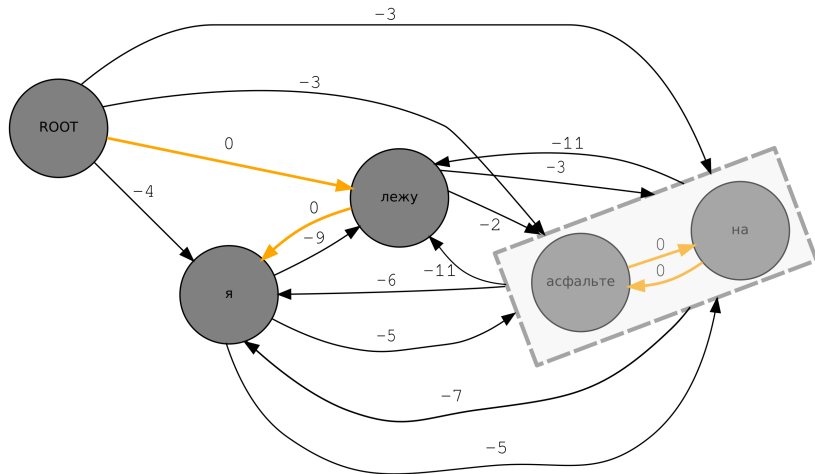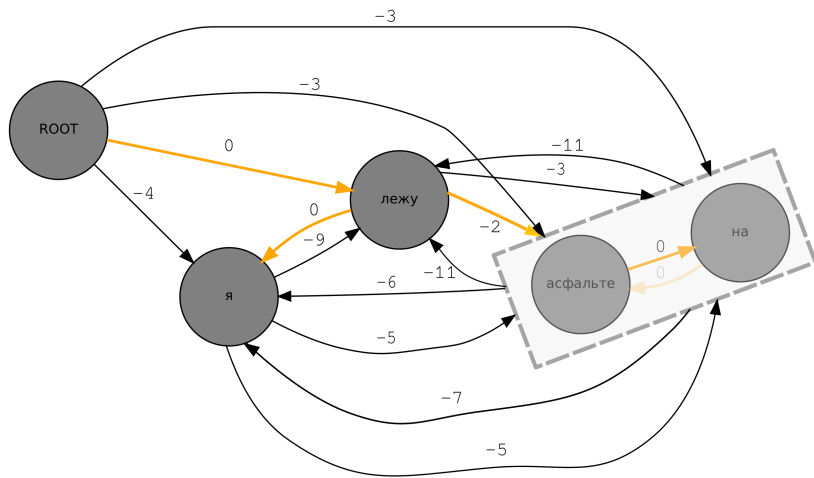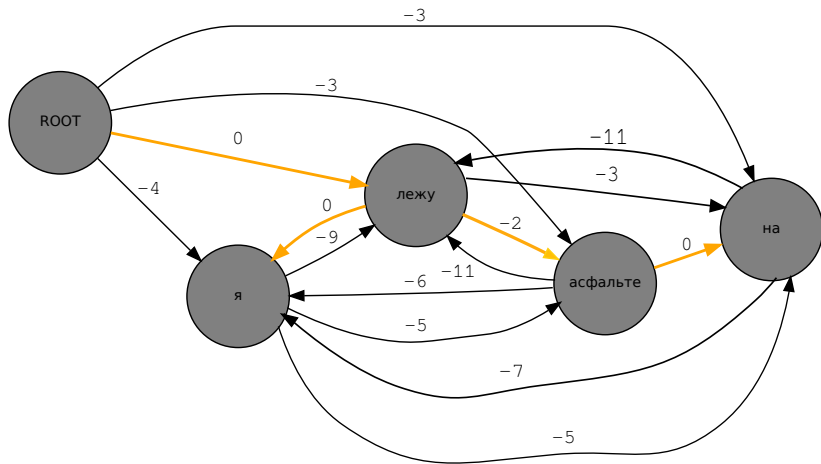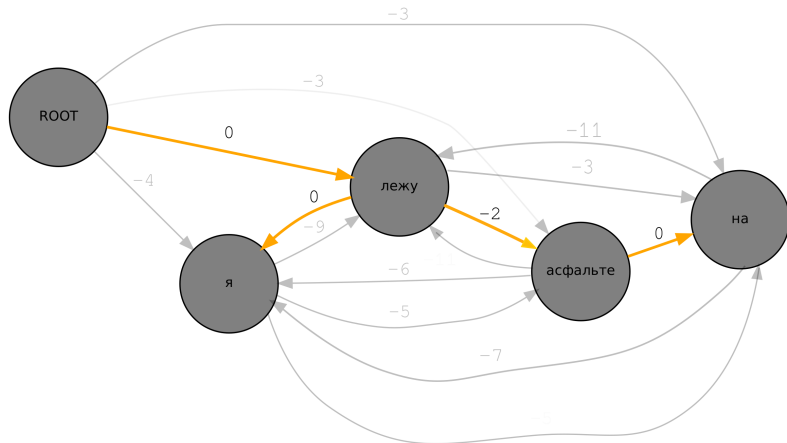
Training is online, like perceptron algorithm:

- Set some random initial weights
- Parse a sentence
- If the parse matches, do nothing,
- otherwise: find the features in the incorrect parse that aren't in the reference and lower the weights

Parsers

Modern:

- UDPipe `http://github.com/ufal/udpipe`
- SyntaxNet `https://github.com/tensorflow/models/tree/master/research/syntaxnet`
- BiST `https://github.com/elikip/bist-parser`
  - Both MST and transition variants
- Stanford Parser `https://nlp.stanford.edu/software/nndep.html`

Historical:

- MaltParser
- MSTParser
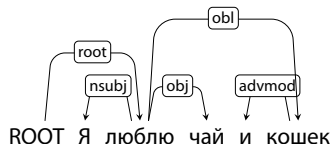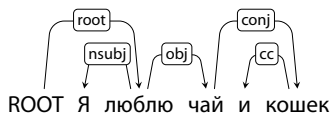
Simple evaluation:

- Unlabelled attachment score, **UAS**: correct heads/total heads
- Labelled attachment score, **LAS**: (correct heads+labels)/total heads
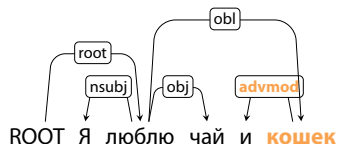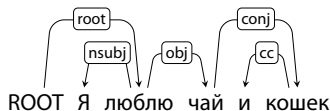
Simple evaluation:

- Unlabelled attachment score, **UAS**: correct heads/total heads
- Labelled attachment score, **LAS**: (correct heads+labels)/total heads

LAS 3/5 = 60%

UAS 4/5 = 80%

Shared tasks

- CoNLL 2006 (13 langs: ar, cs, bg, da, de, es, ja, nl, pt, sl, sv, tr, zh)
- CoNLL 2007 (10 langs: ar, ca, cs, el, en, eu, hu, it, tr, zh)
- CoNLL 2008: + semantic dependencies (English)
- CoNLL 2009: + semantic dependencies (ca, cs, de, en, es, ja, zh)
- ICON 2009 (Hindi, Bangla, Telugu)
- ICON 2010 (Hindi, Bangla, Telugu)
- SPMRL 2013 (9 languages: ar, de, eu, fr, he, hu, ko, pl, sv)
- SPMRL 2014 (9 languages: ar, de, eu, fr, he, hu, ko, pl, sv)
- VarDial 2017 (cross-lingual: cs-sk, sl-hr, da/sv-no)
- CoNLL 2017 (45 languages + surprise + end-to-end parsing)
- CoNLL 2018 (?? languages, end-to-end parsing)

Shared tasks

- CoNLL 2006 (13 langs: ar, cs, bg, da, de, es, ja, nl, pt, sl, sv, tr, zh)
- CoNLL 2007 (10 langs: ar, ca, cs, el, en, eu, hu, it, tr, zh)
- CoNLL 2008: + semantic dependencies (English)
- CoNLL 2009: + semantic dependencies (ca, cs, de, en, es, ja, zh)
- ICON 2009 (Hindi, Bangla, Telugu)
- ICON 2010 (Hindi, Bangla, Telugu)
- SPMRL 2013 (9 languages: ar, de, eu, fr, he, hu, ko, pl, sv)
- SPMRL 2014 (9 languages: ar, de, eu, fr, he, hu, ko, pl, sv)
- VarDial 2017 (cross-lingual: cs-sk, sl-hr, da/sv-no)
- **CoNLL 2017 (45 languages + surprise + end-to-end parsing)**
- **CoNLL 2018 (?? languages, end-to-end parsing)**

```
 1. Stanford (Stanford)                76.30 ± 0.12
 2. C2L2 (Ithaca)                       75.00 ± 0.12
 3. IMS (Stuttgart)                     74.42 ± 0.13
 4. HIT-SCIR (Harbin)                   72.11 ± 0.14
 5. LATTICE (Paris)                     70.93 ± 0.13
 6. NAIST SATO (Nara)                   70.14 ± 0.13
 7. Koç University (İstanbul)           69.76 ± 0.13
 8. ÚFAL — UDPipe 1.2 (Praha)           69.52 ± 0.13
 9. UParse (Edinburgh)                  68.87 ± 0.14
10. Orange — Deskiñ (Lannion)           68.61 ± 0.13
11. TurkuNLP (Turku)                    68.59 ± 0.14
12. darc (Tübingen)                     68.41 ± 0.13
13. BASELINE UDPipe 1.1 (Praha)         68.35 ± 0.14
14. MQuni (Sydney)                      68.05 ± 0.13
15. fbaml (Palo Alto)                   67.87 ± 0.13
16. LyS-FASTPARSE (A Coruña)            67.81 ± 0.13
17. LIMSI-LIPN (Paris)                  67.72 ± 0.14
18. RACAI (Bucureşti)                   67.71 ± 0.13
19. IIT Kharagpur (Kharagpur)           67.61 ± 0.14
20. naistCL (Nara)                      67.59 ± 0.15
21. Wanghao-ftd-SJTU (Shanghai)         66.53 ± 0.14
22. UALING (Tucson)                     65.24 ± 0.13
23. Uppsala (Uppsala)                   65.11 ± 0.13
24. METU (Ankara)                       61.98 ± 0.14
25. CLCL (Genève)                       61.82 ± 0.13
26. Mengest (Shanghai)                  61.33 ± 0.14
27. ParisNLP (Paris)                    60.02 ± 0.14
```

ru_syntagrus

```
 1. Stanford (Stanford)              software1      92.60
 2. C2L2 (Ithaca)                    software5      90.03
 3. IMS (Stuttgart)                  software2      89.80
 4. HIT-SCIR (Harbin)                software4      89.77
 5. NAIST SATO (Nara)                software1      89.31
 6. UParse (Edinburgh)               software1      89.18
 7. Uppsala (Uppsala)                software1      88.04
 8. LATTICE (Paris)                  software7      87.90
 9. LyS-FASTPARSE (A Coruña)         software5      87.55
10. Orange — Deskiñ (Lannion)        software1      87.10
11. fbaml (Palo Alto)                software1      86.83
12. ÚFAL — UDPipe 1.2 (Praha)        software1      86.80
```

en

```
 1. Stanford (Stanford)              software1      82.23
 2. HIT-SCIR (Harbin)                software4      79.94
 3. C2L2 (Ithaca)                    software5      79.64
 4. LATTICE (Paris)                  software7      78.91
 5. IMS (Stuttgart)                  software2      78.71
 6. NAIST SATO (Nara)                software1      77.93
 7. fbaml (Palo Alto)                software1      77.57
 8. Orange — Deskiñ (Lannion)        software1      77.51
 9. ÚFAL — UDPipe 1.2 (Praha)        software1      77.25
10. MQuni (Sydney)                   software2      76.81
11. TurkuNLP (Turku)                 software1      76.68
12. UParse (Edinburgh)               software1      76.42
```

15th April	Training and development data will be released.
30th April	Baseline parsing models will be released.
30th April	Test data available in TIRA. Test phase starts.
26th June	Test phase ends.
28th June	Results will be announced.
10th July	Submission of system description papers.

There is a Vyshka team!

Ace the exam!