ЛАБОРАТОРНАЯ РАБОТА №5

Функционал смысла: построение семантических схем предложений

Цель. Освоить вычислительное представление смысла фрагмента текста, познакомиться с основами вычислительной теории семантической интерпретации и опробовать их на примерах. Научиться по дереву синтаксических зависимостей строить обратную польскую запись (ОПЗ) функционала смысла и соответствующую ей семантическую схему предложения. Освоить практику разметки текста с помощью библиотеки Natasha и нормализации слов с использованием pymorphy2.

5.1. Теоретические сведения

В рамках вычислительной теории семантической интерпретации смысл текстового фрагмента определяется через функционал над смысловыми значениями входящих в него слов, а определение смысла всего фрагмента сводится к построению и выполнению вычислительной процедуры над смыслами отдельных слов.

Основные обозначения:

1) $S(\alpha)$ – множество смысловых значений слова α;

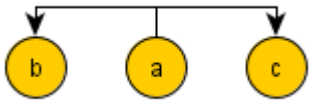
2) $S(\alpha)=\Phi\left(S(a_1),\ S(a_2),\ \ldots,\ S(a_n)\right)$, где $\alpha=a_1a_2\ldots a_n$ – текстовый фрагмент, Φ – функционал смысла.

Операции:

1. $\vec{\cap}$ – операция контекстного уточнения смысла. Так, запись $S(a)\vec{\cap}S(b)$ означает, что слово b контекстно уточняет смысл слова a.

2. $\cap$ – операция пересечения смыслов.

Пример текстового фрагмента:



И его функционала смысла:

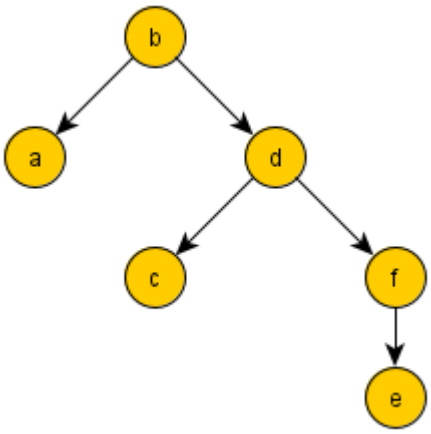
$$\left(S(a)\vec{\cap}S(b)\right)\cap\left(S(a)\vec{\cap}S(c)\right).$$

При вычислениях удобно использовать обратную польскую запись функционала смысла. ОПЗ не содержит скобок, и ее вычисления проводится за один проход слева направо. Операции выполняются по мере их встречаемости в процессе прохода выражения. Поэтому структура ОПЗ однозначно определяет структуру вычислительной процедуры. Графическое представление ОПЗ называется семантической схемой.

Пусть дан фрагмент:

$$q=\text{"}\underbrace{\text{международное}}_{a}\ \underbrace{\text{признание}}_{b}\ \underbrace{\text{образовательных}}_{c}\ \underbrace{\text{программ}}_{d}\ \underbrace{\text{российских}}_{e}\ \underbrace{\text{вузов}}_{f}\text{"}.$$

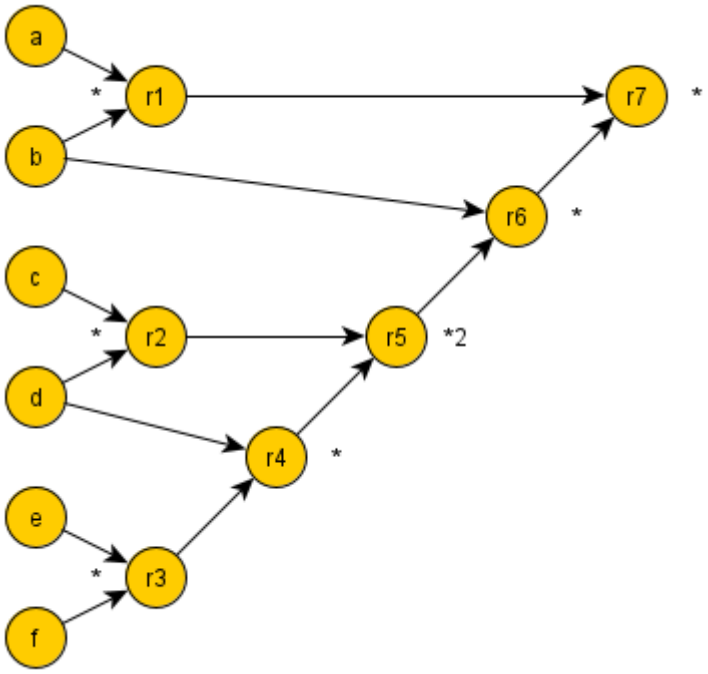
Обозначим буквами $a, b, \ldots, f$ слова этого фрагмента в том порядке, в каком они следуют в фрагменте. Тогда дерево зависимостей выглядит следующим образом:



ОПЗ этого фрагмента:

$$S(a)S(b)\overleftarrow{\frown}S(c)S(d)\overleftarrow{\frown}S(e)S(f)\overleftarrow{\frown}S(d)\overleftarrow{\frown}\frown S(b)\overleftarrow{\frown}\frown$$

Семантическая схема:

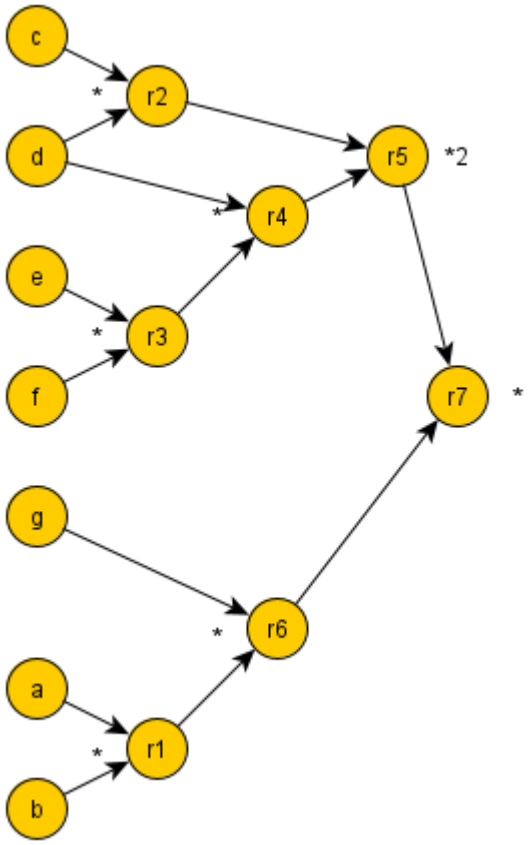


При этом операция контекстного уточнения смысла на семантической схеме обозначается *, а операция пересечения смыслов – $*n$, где n – количество смыслов, пересечение которых необходимо найти. $r_1, r_2, \ldots, r_7$ в данном случае – это фрагменты, на основе которых осуществляется сравнение на семантическую близость.

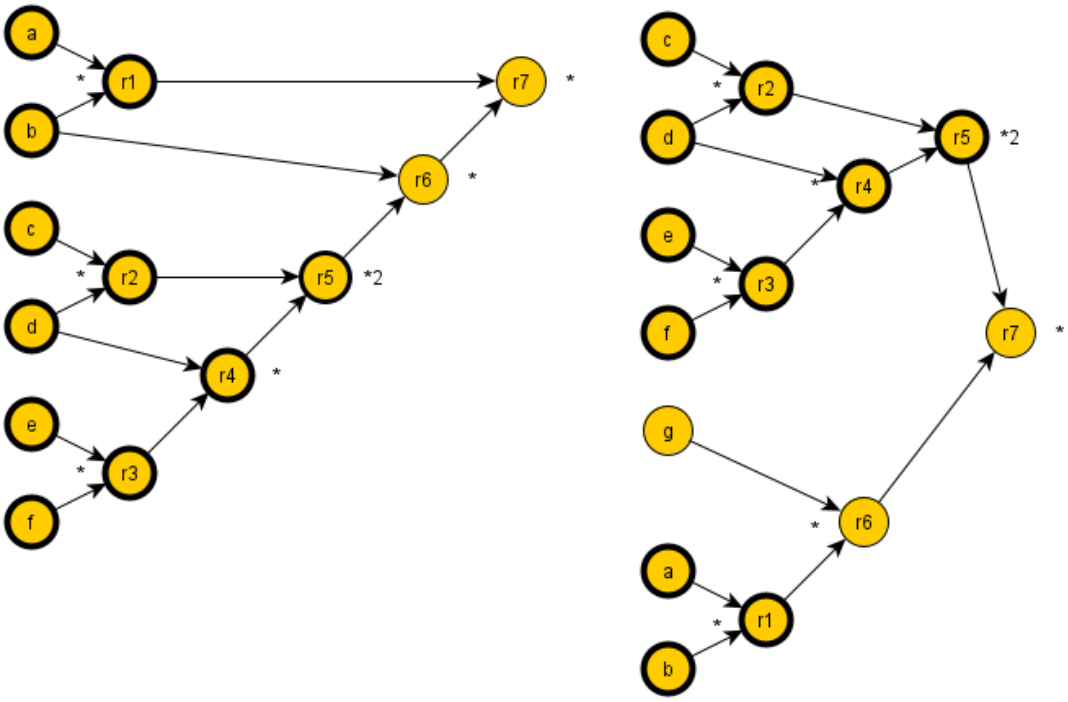
Пример сравнения:

Пусть дан фрагмент t_2 = «образовательные программы российских вузов нуждаются в международном признании».

Семантическая схема этого фрагмента выглядит следующим образом:



Общие фрагменты выделены ниже:



Как это следует из рисунка, первая семантическая схема содержит 7 узлов r_1, r_2, …, r_7, пять из которых присутствуют во второй семантической схеме, образуя в обеих схемах одинаковые поддеревья. Поэтому семантическая близость в данном случае равна $\frac{5}{7}$.

По дереву зависимостей строится формула функционала смысла. Затем:

1. Формула переводится в **обратную польскую запись (ОПЗ)** – последовательность

операндов и операций без скобок.

2. По ОПЗ строится **семантическая схема** – ориентированный граф, где:

– жёлтые круги соответствуют словам (или их леммам),

– узлы вида $r_1, r_2, \ldots, r_k$ соответствуют промежуточным результатам вычисления функционала смысла,

– дуги помечены операциями (* – контекстное уточнение, *n – пересечение n смыслов).

Переход от дерева зависимостей к ОПЗ функционала смысла. Общая идея алгоритма:

1. Для каждого токена предложения создаётся вершина дерева. Вершина-корень соответствует главному слову (обычно сказуемое или опорное существительное).

2. Выполняется обход дерева (например, в глубину).

3. Для каждого перехода «родитель → потомок» в ОПЗ добавляется операция контекстного уточнения (*).

4. Если у вершины более одного потомка, после их обработки в ОПЗ добавляется операция пересечения (*n), где n – число потомков.

5. В ОПЗ используются **леммы** слов, чтобы одинаковые словоформы сводились к одной форме.

Построение семантической схемы по ОПЗ. Семантическая схема строится как граф:

1. Каждый исходный операнд (лемма) – отдельный узел.

2. Проходите по последовательности ОПЗ слева направо:

– при встрече операции контекстного уточнения объединяйте два последних операнда/фрагмента в новый узел r_i;

– при встрече *n объединяйте последние n фрагментов в новый узел;

– каждый новый узел помечайте номером $r_1, r_2, \ldots$ и добавляйте дуги от исходных элементов к нему.

5.2. Выполнение работы

Установка Graphviz и проверка

1. **Python-пакет:** pip install graphviz

2. **Системная утилита** dot:

– macOS: brew install graphviz

– Ubuntu/Debian: sudo apt-get install graphviz

– Windows: установить Graphviz с сайта проекта и добавить ...\Graphviz\bin в PATH.

Проверка: выполните в терминале dot -V. Если команда не находится, система не видит dot (путь не добавлен). Если import graphviz проходит, но при рендеринге ошибка «executables not found», это та же причина – поставьте системный Graphviz и/или добавьте его bin в PATH.

```
from natasha import Segmenter, NewsEmbedding, NewsMorphTagger,
NewsSyntaxParser, Doc
import pymorphy2 as pm
from graphviz import Digraph


def nid(t):
    _, x = map(int, t.split('_'))
    return x


def s2d(s):
    w = {}
```

```
    for tok in s.tokens:
        if tok.pos != 'PUNCT':
            i = nid(tok.id)
            w[i] = tok.text
            if tok.rel == 'root':
                r = i
    tr = {k: [] for k in w.keys()}
    for t in s.tokens:
        if t.pos != 'PUNCT':
            i = nid(t.id)
            h = nid(t.head_id)
            if h != 0:
                tr[h].append(i)
    return w, r, tr


ma = pm.MorphAnalyzer()


def lem(x):
    return ma.parse(x)[0].normal_form


def d2r(w, r, tr):
    if len(w) <= 1:
        return list(w.values())
    rp = []
    st = [(0, r)]
    col = {k: 0 for k in w.keys()}

    while st:
        p, c = st[-1]
        if col[c] == 0:
            col[c] = 1
        elif col[c] == 1:
            if len(tr[c]) > 1:
                rp.extend(['*', len(tr[c])])
            if c != r:
                if len(tr[c]) == 0:
                    rp.extend([w[c], w[p], '*'])
                else:
                    rp.extend([w[p], '*'])
            st.pop()
        for nx in reversed(tr[c]):
            if col[nx] == 0:
                st.append((c, nx))
    return rp


def r2ss(rp):
    if len(rp) <= 1:
        return rp
    ss = []
    i = 0
    while i < len(rp):
        if rp[i] == '*':
            if i + 1 < len(rp) and isinstance(rp[i + 1], int):
                n = rp[i + 1]
                m = 2
                rp[i + 1] = str(rp[i + 1])
            else:
                n = 2
                m = 1
            r = ' '.join(rp[i - n:i + m])
```

```
                rp[i - n:i + m] = [r]
                ss.append(r)
                i -= n
            i += 1
        return ss


def s2ss(s):
    w, r, tr = s2d(s)
    for k in w.keys():
        w[k] = lem(w[k])
    rp = d2r(w, r, tr)
    ss = r2ss(rp)
    return ss


def viz(s, k='dep', fn='tree', dr='TB', lm=True, rn=True):
    d = Digraph(comment=f'{k} tree', format='png')
    d.attr(rankdir=dr, splines='polyline', nodesep='0.35',
ranksep='0.6')
    d.attr('node', shape='circle', fontname='Arial', fontsize='12')
    d.attr('edge', fontname='Arial', fontsize='10')

    if k == 'dep':
        w, r, tr = s2d(s)
        if lm:
            for kx in list(w.keys()):
                w[kx] = lem(w[kx])
        for kx, wx in w.items():
            d.node(str(kx), wx)
        root_added = False
        for p, chs in tr.items():
            for ch in chs:
                if p == 0:
                    if not root_added:
                        d.node('ROOT', 'ROOT', shape='plaintext',
fontsize='10')
                        root_added = True
                    d.edge('ROOT', str(ch))
                else:
                    d.edge(str(p), str(ch))
    elif k == 'ss':
        w, r, tr = s2d(s)
        if lm:
            for kx in list(w.keys()):
                w[kx] = lem(w[kx])
        rp = d2r(w, r, tr)
        lbl = {}
        for kx in sorted(w.keys()):
            txt = w[kx]
            nid2 = f'w{kx}'
            lbl.setdefault(txt, []).append(nid2)
            d.node(nid2, txt)
        mg = {}
        wrk = list(rp)
        i = 0
        rc = 0
        fc = 0
        while i < len(wrk):
            if wrk[i] == '*':
                if i + 1 < len(wrk) and isinstance(wrk[i + 1], int):
                    n = wrk[i + 1]
                    m = 2
                    lab = f'*{n}'
```

```
                    wrk[i + 1] = str(wrk[i + 1])
                else:
                    n = 2
                    m = 1
                    lab = '*'
                ops = wrk[i - n:i]
                chs = []
                for op in ops:
                    if op in mg:
                        chs.append(mg[op])
                    else:
                        if op in lbl and len(lbl[op]) > 0:
                            nid3 = lbl[op][0]
                        else:
                            fc += 1
                            nid3 = f'leaf_{fc}'
                            d.node(nid3, op)
                        chs.append(nid3)
                rc += 1
                rid = f'r{rc}'
                rexpr = ' '.join(wrk[i - n:i + m])
                mg[rexpr] = rid
                d.node(rid, rid, xlabel=lab)
                for ch in chs:
                    d.edge(ch, rid)
                wrk[i - n:i + m] = [rexpr]
                i -= n
            i += 1
    else:
        raise ValueError("k должно быть \"dep\" или \"ss\"")
    if rn:
        d.render(fn, cleanup=True)
    return d


seg = Segmenter()
emb = NewsEmbedding()
mt = NewsMorphTagger(emb)
sp = NewsSyntaxParser(emb)

q = 'международное признание образовательных программ российских вузов'
dq = Doc(q)
dq.segment(seg)
dq.tag_morph(mt)
dq.parse_syntax(sp)
s1 = dq.sents[0]

t = 'международное признание образовательных программ поднимает
репутационный рейтинг российских вузов'
dt = Doc(t)
dt.segment(seg)
dt.tag_morph(mt)
dt.parse_syntax(sp)
s2 = dt.sents[0]

for label, sent_obj, text in [('q', s1, q), ('t', s2, t)]:
    w, r, tr = s2d(sent_obj)
    for k in w.keys():
        w[k] = lem(w[k])
    rp = d2r(w, r, tr)
    ss = r2ss(list(rp))

    print('{:s} = {:s}'.format(label, text))
    print('ОПЗ функционала смысла:')
```

```
            print(' '.join(map(str, rp)))
            print('Фрагменты функционала смысла:')
            for x in ss:
                print('  ', x)
            print()

        viz(s1, k='dep', fn='qd', dr='TB')
        viz(s1, k='ss', fn='qs', dr='LR')
        viz(s2, k='dep', fn='td', dr='TB')
        viz(s2, k='ss', fn='ts', dr='LR')
```

Вывод программы:

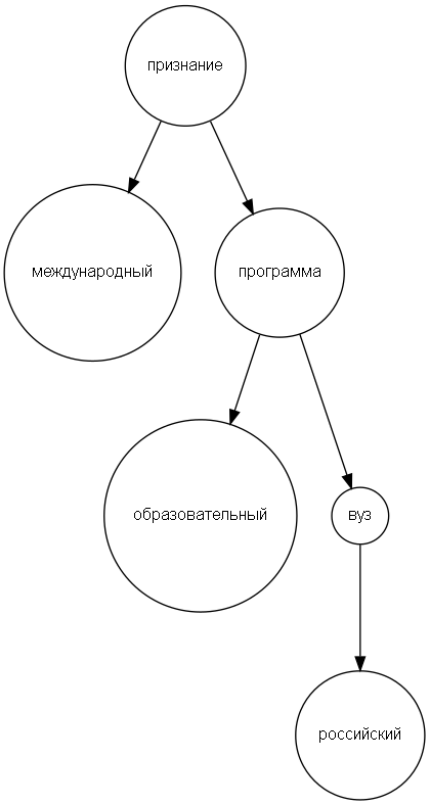
```
        q = международное признание образовательных программ российских вузов
        ОПЗ функционала смысла:
        международный признание * образовательный программа * российский вуз *
программа * * 2 признание * * 2
        Фрагменты функционала смысла:
           международный признание *
           образовательный программа *
           российский вуз *
           российский вуз * программа *
           образовательный программа * российский вуз * программа * * 2
           образовательный программа * российский вуз * программа * * 2
признание *
           международный признание * образовательный программа * российский вуз
* программа * * 2 признание * * 2

        t = международное признание образовательных программ поднимает
репутационный рейтинг российских вузов
        ОПЗ функционала смысла:
        международный признание * образовательный программа * признание * * 2
поднимать * репутационный рейтинг * российский вуз * рейтинг * * 2 поднимать
* * 2
        Фрагменты функционала смысла:
           международный признание *
           образовательный программа *
           образовательный программа * признание *
           международный признание * образовательный программа * признание * * 2
           международный признание * образовательный программа * признание * * 2
поднимать *
           репутационный рейтинг *
           российский вуз *
           российский вуз * рейтинг *
           репутационный рейтинг * российский вуз * рейтинг * * 2
           репутационный рейтинг * российский вуз * рейтинг * * 2 поднимать *
           международный признание * образовательный программа * признание * * 2
поднимать * репутационный рейтинг * российский вуз * рейтинг * * 2 поднимать
* * 2
```

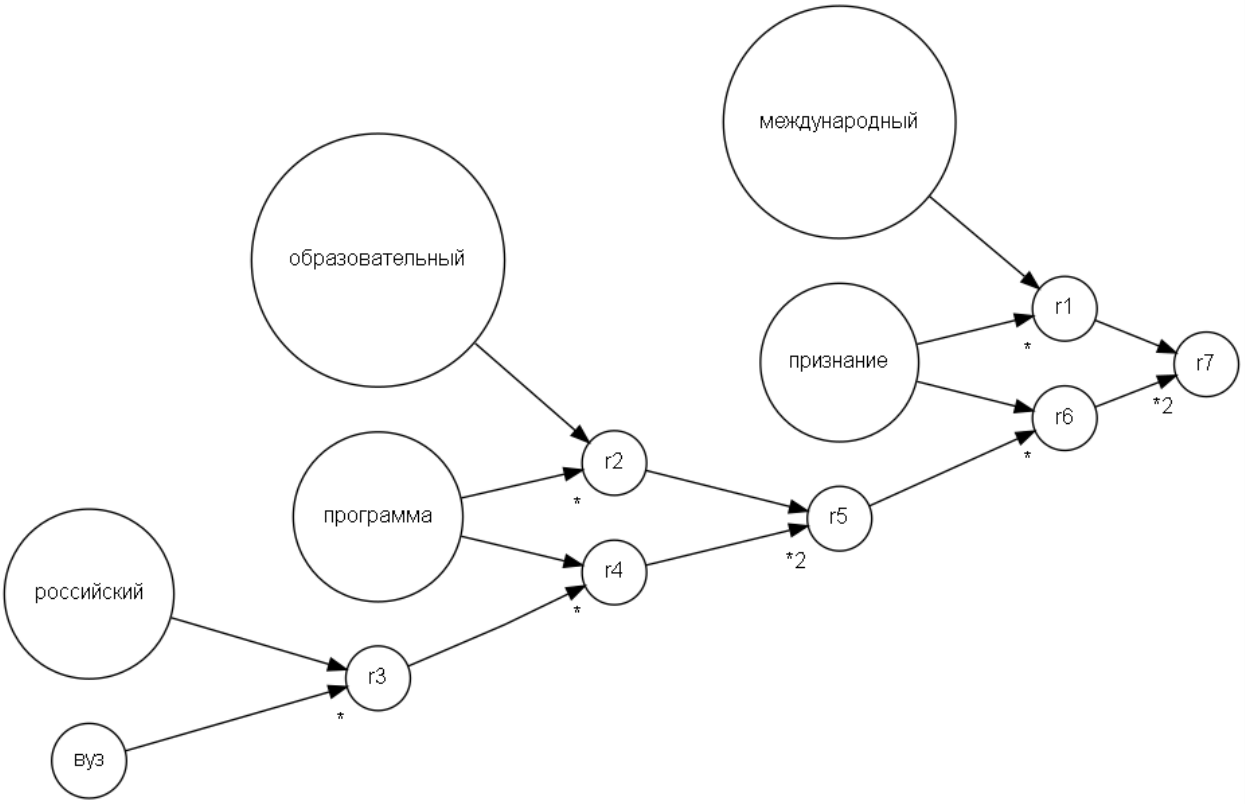
В папке со скриптом появятся 4 файла:

1. qd.png – дерево зависимостей запроса.
2. qs.png – семантическая схема запроса.
3. td.png – дерево зависимостей текста.
4. ts.png – семантическая схема текста.

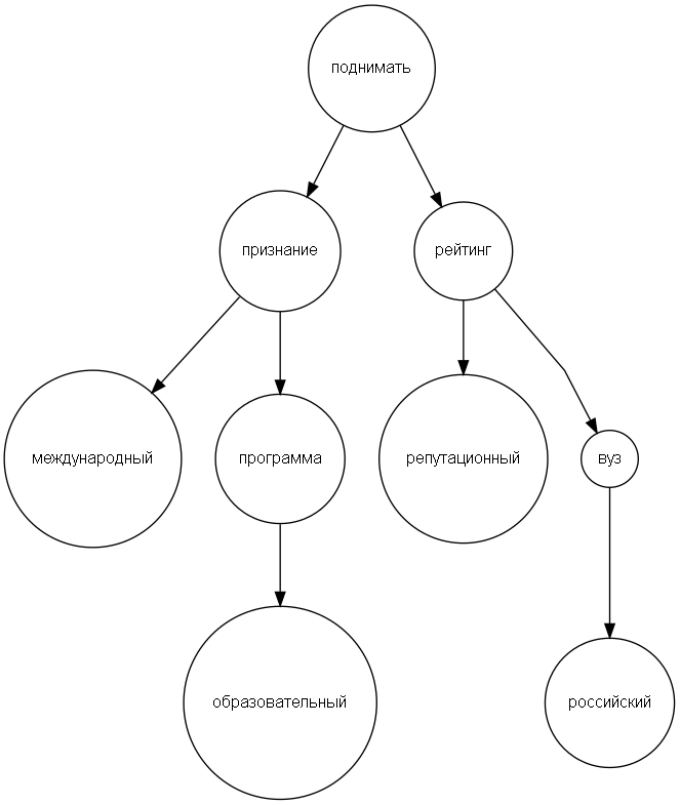
Дерево зависимостей запроса:



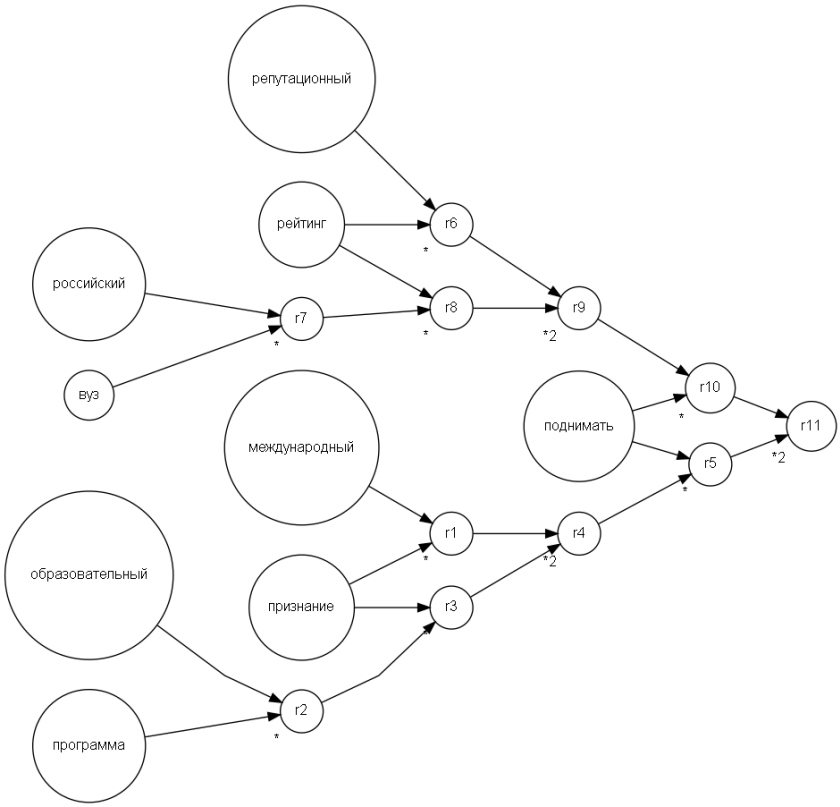
Семантическая схема запроса:



Дерево зависимостей текста:



Семантическая схема текста:



5.3. Задания к лабораторной работе №5

1. Выберите **5 предложений** из личного датасета (согласованного ранее с преподавателем).

2. Для каждого предложения:

– получите дерево синтаксических зависимостей с помощью Natasha;

– постройте ОПЗ функционала смысла (последовательность лемм и операций);

– постройте семантическую схему в виде ориентированного графа (Graphviz или аналогичный инструмент).

3. Для **одного** предложения приведите подробный разбор:

– исходное предложение;

– фрагмент дерева зависимостей (скриншот);

– детальное объяснение, как по дереву получена ОПЗ;

– семантическая схема (рисунок) с пояснениями того, что означают узлы r_i.

4. Для остальных четырёх предложений представьте краткий формат: текст, ОПЗ и изображение семантической схемы без подробных комментариев.

Требования к отчету к ЛР №5

1. Краткая теоретическая часть:

– определение функционала смысла;

– описание операций контекстного уточнения и пересечения;

– связь между деревом зависимостей, ОПЗ и семантической схемой.

2. Программная часть:

– листинги ключевых функций, реализующих преобразование «дерево зависимостей → ОПЗ → семантическая схема»;

– краткие комментарии к коду.

3. Результаты:

– для каждого из 5 предложений: текст, дерево зависимостей (картинка), ОПЗ, семантическая схема (картинка);

– для одного предложения – подробный разбор.

4. Выводы:

– какие конструкции в вашем датасете чаще всего порождали операции пересечения с большой арностью;

– в каких случаях возникали неоднозначности в дереве зависимостей и как они влияли на схему;

– удобен ли выбранный способ визуализации.

5.4. Используемые источники

1. Вишняков Ю.М., Вишняков Р.Ю. Вычислительная теория семантической интерпретации текстов научно-технического стиля естественного языка // Сборник трудов Международной научно-технической конференции «Актуальные проблемы прикладной математики, информатики и механики», Воронеж, 18–20 декабря 2017 г. – Воронеж: Издательство «Научно-исследовательские публикации», 2017. С. 219–226.