АЛГОРИТМ СИНТАКСИЧЕСКОГО АНАЛИЗА, ИСПОЛЬЗУЕМЫЙ В СИСТЕМЕ МОРФО-СИНТАКСИЧЕСКОГО АНАЛИЗА «TREETON»[1]

ALGORITHM OF SYNTAX ANALYSIS EMPLOYED BY THE «TREETON» MORPHO-SYNTACTIC ANALYSIS SYSTEM

Старостин А.С. *(Anatoli.Starostin@avicomp.ru)*
Мальковский М.Г. *(malk@cs.msu.su)*
Московский Государственный Университет им. М.В. Ломоносова

Эта статья является продолжением работы [Мальковский, Старостин 2006]. В ней рассматривается алгоритм анализа, используемый синтаксическим анализатором Treevial. Этот анализатор является частью системы морфо-синтаксического анализа Treeton, разрабатываемой авторами на кафедре алгоритмических языков факультета ВМиК МГУ начиная с декабря 2004 года. Первые три раздела статьи посвящены описанию математической модели, которая стала основой анализатора Treevial. Далее, в оставшихся двух разделах, формально ставится задача синтаксического анализа, приводится алгоритм решения этой задачи и обсуждаются различные свойства этого алгоритма.

Введение

Поскольку система морфо-синтаксического анализа Treeton и анализатор Treevial существует уже достаточно давно и тот формальный аппарат, с которым они имеют дело, все время эволюционирует, авторы приняли решение зафиксировать нынешнее состояние формального аппарата в виде несложной математической модели, состоящей из трех базовых понятий, одной базовой операции над этими понятиями и алгоритма, использующего все вышеупомянутое.

Используемые в рамках модели синтаксические правила позволяют описывать язык в терминах грамматик составляющих (ср. [Хомский, 1962]), в терминах грамматик зависимостей (ср. [Mel'cuk , 1988]), а также с использованием смешанных моделей (в духе [Гладкий, 1985]). Это позволяет сравнивать упомянутые подходы не только умозрительно ([Schneider, 1998]), но и на практике. Такой подход к описанию синтаксиса, как LinkGrammar ([Sleator & Temperley, 1991]) с небольшими оговорками также может быть смоделирован при помощи описываемого формализма.

Создание эффективного алгоритма анализа в рамках предлагаемой обобщенной модели оказывается достаточно сложной задачей. Описываемый в этой статье алгоритм решает эту задачу, хотя и обладает рядом недостатков. Несмотря на это, сам факт существования универсального алгоритма анализа, оперирующего как со структурами зависимостей, так и с системами составляющих, представляется авторам заслуживающим внимания. Использование штрафных функций, влияющих на ход анализа, сближает предлагаемый подход с вероятностными контекстно-свободными грамматиками (PCFG [Johnson, 1998]).

Авторы сознательно не проводят подробного сравнительного анализа предлагаемого алгоритма и других алгоритмов синтаксического анализа, т.к. считают это темой для отдельной работы. Данная статья больше фокусируется на математических понятиях, участвующих в формулировке алгоритма.

Основным элементом формальной модели является *синтаксическая структура* – понятие, очень близкое к понятию тринотации, введенному в [Мальковский, Старостин 2006]. Это не должно смущать читателя. Дело в том, что понятие тринотации было введено прежде всего как понятие из мира программирования (тринотация даже определялись рекурсивно, исходя из структуры данных, используемой авторами), а синтаксическая структура в настоящей статье – это понятие из мира математики. Тем более что некоторые различия между тринотацией и синтаксической структурой все же есть.

Для начала определимся с той абстракцией, которую мы примем для лингвистических описаний. Это необходимо прежде всего для формализации выхода морфологического анализатора.

[1] Работа подготовлена при поддержке ООО "ЯНДЕКС" (www.yandex.ru)

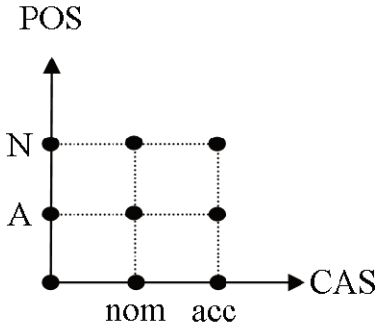
Пространство категорий

Будем считать, что все лингвистические категории, которые необходимо различать, составляют конечное множество *ATTR*, а все их возможные значения заданы в виде инъективного отображения *VAL: ATTR* $\rightarrow$ *{*Y_i| Y_i – конечное или счетное множество*}*.

Обозначим через *C* множество всех наборов вида $\{((f_1, v_1), (f_2, v_2), ..., (f_n, v_n)) \mid 0 \le n, f_i \in ATTR, v_i \in VAL(f_i)$ для всех $1 \le i \le n, f_i \ne f_j$ для i ≠ j}

Будем называть множество *C пространством категорий*.

Пространство категорий содержит в себе все наборы пар вида <*атрибут,значение*>, которые могут появиться в лингвистическом описании (см. рис.1).



ATTR = {POS,CAS}
VAL: POS → {N,A}, CAS → {nom,acc}

C = {
(POS=N,CAS=nom), (POS=N,CAS=acc),
(POS=A,CAS=nom), (POS=A,CAS=acc) ,
(POS=N), (POS=V), (CAS=nom), (CAS=acc), ()
}

Рис.1. Пример
лингвистического описания

Будем считать, что все типы синтаксических связей, которые необходимо различать, принадлежат конечному множеству *RELS.*

Синтаксические структуры, мультиструктуры

В этом разделе описываются два основных понятия, которыми мы будем оперировать при описании алгоритма синтаксического анализа, – синтаксическая структура и мультиструктура.

Синтаксическая структура – это направленное дерево с размеченными узлами и дугами, в котором некоторые дуги могут помечаться специальным символом *w*. С каждым узлом дерева соотносится некоторый элемент пространства категорий *C* и некоторый интервал числовой оси. Интервал для некоторых узлов может быть не определен. Те узлы, для которых интервал определен, соответствуют лингвистическим объектам, обладающим своим местом в тексте, например, словам (точнее, вариантам морфологического анализа слов). Каждое слово где-то начинается и где-то заканчивается. То же самое можно сказать про синтаксические группы. Узлы, для которых интервал не определен, могут потребоваться только в случае добавления в модель синтаксических нулей, т.к. «локализовать» синтаксический нуль не всегда возможно.

Поддеревья, «висящие» на *w*-связях, воспринимаются как вложенные в узел, из которого исходит *w*-связь. Все интервалы, соотнесенные с узлами таких поддеревьев, вложены в интервал, соотнесенный с узлом, из которого исходит *w*-связь. Введение такой связи позволяет с помощью описываемого аппарата работать не только с грамматиками зависимостей, но и с системами непосредственных составляющих и со смешанными системами (системы синтаксических групп А.В. Гладкого [Гладкий, 1985]).

На синтаксическую структуру накладывается еще одно условие: интервалы, соотнесенные с узлами, из которых не выходит ни одна *w*-связь (тривиальными узлами), не могут пересекаться друг с другом (в случае если они определены). Это условие соответствует тому факту, что в контексте некоторой синтаксической структуры одному отрезку текста не могут соответствовать две морфологические интерпретации. Приведем формальное определение синтаксической структуры.

Синтаксической структурой называется набор *(T(N,E),e,f,g), где*

T(N,E) – направленное дерево (*N* – множество вершин, *E* – множество упорядоченных пар *(a,b)*, *a,b* $\in$ *N)*

e – функция атрибутной разметки вершин: *e*: $N \rightarrow C$

f – функция пространственной разметки вершин: *f*: $N \rightarrow \{(s,e) \mid s,e \in R, 0 \le s < e\}$

g – функция разметки дуг: *g*: $E \rightarrow RELS \cup \{w\}$.

Кроме того должны выполняться следующие условия:

1. Для любых $x,z \in N$, таких что в T существует направленный путь вида (x,y,….,z) и $g((x,y)) = w$, верно, что либо оба интервала $f(z)$ и $f(x)$ не определены, либо интервал $f(x)$ определен и интервал $f(z)$ либо не определен, либо вложен в интервал $f(x)$.
2. Для любых $x,y \in N$, таких что $\exists f(x),f(y)$, и что не существует $p \in N$, для которого $(x,p) \in E$ и $g((x,p)) = w$ и не существует $q \in N$, для которого $(y,q) \in E$ и $g((y,q)) = w$, верно, что интервал $f(x)$ не пересекается с интервалом $f(y)$.

В дальнейшем, в случае существования между двумя вершинами x, z направленного пути (x,y,….,z), такого что $g((x,y)) = w$, будем говорить, что z является w-потомком x.

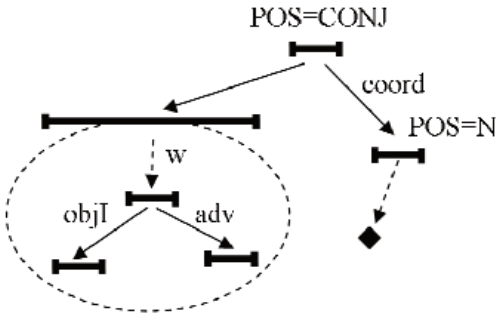


Рис.2. Пример синтаксической структуры. Жирные линии обозначают интервалы, соответствующие узлам. Ромбом обозначен синтаксический ноль.

На рис.2 приводится пример синтаксической структуры.

Будем называть синтаксическую структуру, у которой дерево T состоит из одной вершины, *тривиальной*. Примером тривиальной синтаксической структуры может служить один вариант морфологического анализа.

Для описания алгоритма синтаксического анализа нам потребуется понятие *мультиструктуры*. Мультиструктура – это направленный граф с размеченными узлами и дугами, который допускает разбиение на набор направленных деревьев, каждое из которых является синтаксической структурой. С помощью мультиструктур удобно описывать все промежуточные фазы синтаксического анализа.

Вход синтаксического анализатора является мультиструктурой (т.к. отдельный вариант морфологического анализа – это тривиальная синтаксическая структура). Все промежуточные стадии анализа, в которых какие-то связи проведены, а какие-то нет, какие-то группы собраны, а какие-то нет, также являются мультиструктурами. Как будет показано ниже, алгоритм синтаксического анализа оперирует с мультиструктурами, применяя к ним синтаксические правила и получая тем самым новые мультиструктуры (см. рис.3). Задача синтаксического анализа состоит в том, чтобы из всего многообразия мультиструктур выделить те, в которых присутствуют связные, покрывающие весь исходный текст, синтаксические структуры.

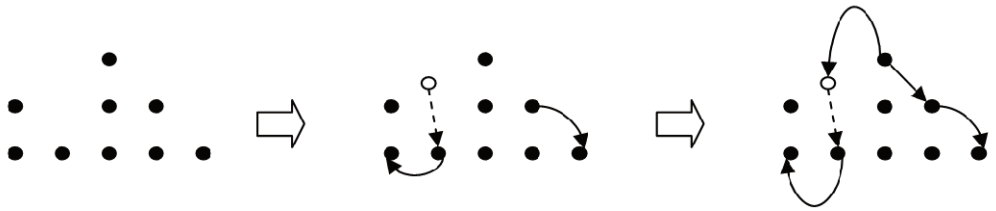


Рис.3. Пример преобразования мультиструктур. Черными кружками обозначены исходные варианты морфологического анализа. Прозрачный кружок обозначает созданную вершину.

Приведем формальное определение мультиструктуры:

Мультиструктурой называется набор $(G(N,E),e,f,g)$, *где*

$G(N,E)$ – направленный граф

e – функция атрибутной разметки вершин: e: $N \rightarrow C$

f – функция пространственной разметки вершин: f: $N \rightarrow \{(s,e)$ | s,e – числа, $0 \le s <$ e$\}$

g – функция разметки дуг g: $E \rightarrow RELS \cup \{w\}$.

При этом граф G обладает следующим свойством:

Множество N представимо в виде $N=N_1 \cup N_2 ... \cup Nm$, а множество E в виде $E=E_1 \cup E_2 ... \cup Em$, и оба разбиения таковы, что $(G'(N_i,E_i),e,f,g)$ является синтаксической структурой для $1 \le i \le m$.

Если указанное свойство выполняется, то можно говорить о синтаксических структурах, образующих разбиение мультиструктуры.

Синтаксические правила

Наиболее важным для дальнейшего изложения понятием является *синтаксическое правило*. Синтаксическое правило применяется к мультиструктуре. В момент применения правила мультиструктура пре-

образуется – меняется ее разбиение на синтаксические структуры. Синтаксическое правило – это дерево, часть узлов и дуг которого объявляются исходными, а часть производными. При применении правила определенные узлы мультиструктуры сопоставляются с исходными узлами правила. Если сопоставление возможно, то производится попытка добавить в мультиструктуру новые узлы и дуги, соответствующие производным узлам и дугам правила. Если после этой процедуры мультиструктура продолжает допускать разбиение на синтаксические структуры, то правило считается применимым к рассматриваемым узлам мультиструктуры.

Следует отметить, что синтаксические правила, описанные в [Мальковский, Старостин 2006], являются частным случаем правил, о которых пойдет речь ниже. Дадим формальное определение синтаксического правила и поясним его.

Синтаксическим правилом называется набор $(T(N_o \cup N_n, E_o \cup E_n), \alpha_o, \alpha_n, \beta, \Psi)$, где

$T(N_o \cup N_n, E_o \cup E_n)$ – направленное дерево ($N_o \cup N_n$ – множество вершин, $E_o \cup E_n$ – множество упорядоченных пар $(a,b), a,b \in N$). N_o обозначает исходные вершины, N_n – производные вершины. E_o обозначает исходные дуги, E_n – производные дуги. $N_o \neq \varnothing$. $E_n \neq \varnothing$. Для $\forall (n_1,n_2) \in E_o$: $n_1 \in N_o$, $n_2 \in N_o$

α_o – функция разметки исходных вершин: α_o: $N_o \rightarrow P(C)$

α_n – функция разметки производных вершин: α_n: $N_n \rightarrow \{f: C^{|No|} \rightarrow C\}$

β – функция разметки дуг: β: $E_o \cup E_n \rightarrow RELS \cup \{w\}$.

Ψ – область многомерных ограничений: $\Psi \in P(C^{|No|} \times \{(s,e) \mid$ s,e – числа, $0 \leq s < e\}^{|No|})$

$P(C)$ – множество всех подмножеств множества C

Функция α_o сопоставляет каждой исходной вершине правила некоторую область пространства категорий. Эту область можно также называть одномерным шаблоном. То, как именно такая область задается, не имеет особенного значения для дальнейшего изложения. Скажем только, что в практическом плане удобны такие способы задания областей, при которых можно эффективно и быстро получать ответ на вопрос: «входит ли данная точка пространства категорий в область?». В [Мальковский, Старостин 2006] описывается язык задания одномерных шаблонов, используемый в системе Treeton. Ограничимся здесь только примером шаблона, описывающего прилагательные и причастия в единственном числе:

(1) (NMB=sg,(POS=V,REPR=part | POS=A))

Функция α_n сопоставляет каждой производной вершине правила некоторую функцию, позволяющую определить элемент пространства категорий по набору из $|No|$ элементов пространства категорий ($|No|$ обозначает количество исходных вершин правила). Эта функция требуется в момент применения правила при добавлении в мультиструктуру нового узла, соответствующего производной вершине, для соотнесения этого узла с некоторым элементом пространства категорий. В этот момент аргументом функции станет набор из $|No|$ элементов пространства категорий, соответствующих узлам мультиструктуры, сопоставленным с исходными вершинами правила. Выбор формального аппарата для задания таких функций не влияет на дальнейшее изложение. В [Мальковский, Старостин 2006] описываются так называемые *векторы присваивания*. В данный момент в системе Treeton используется именно этот способ задания упомянутых функций. Они обладают хорошей выразительной мощностью. Приведем пример вектора присваивания:

(2) (:=A.*, CAS:=B.CAS)*

Этот вектор задает функцию, отображающую два элемента пространства категорий (соотнесенных с A и B) в один элемент в соответствии со следующим принципом: возьми все пары *<атрибут,значение>* из элемента соотнесенного с A, затем добавь к ним пару *<CAS, значение атрибута CAS у элемента, соотнесенного с B>*, причем, если пара с атрибутом CAS уже есть – замени ее.

Функция β сопоставляет как исходным, так и производным дугам правила фиксированные элементы пространства типов связей (здесь и далее будем называть так множество $RELS \cup \{w\}$). Для исходных дуг этот элемент используется в момент применения правила как шаблон для проверки, а для производных дуг, наоборот, как константная функция. Эту часть нашего формального аппарата можно было бы развить по аналогии с шаблонами и функциями для вершин, но в данный момент это кажется немотивированным усложнением.

Область Ψ используется для того, чтобы задавать те ограничения на применение правила, в которых участвует более одного узла (например, согласование нескольких элементов по какой-то категории). Эту область можно также называть многомерным шаблоном. Набор из $|No|$ элементов пространства категорий и $|No|$ числовых интервалов удовлетворяет ограничениям тогда, когда попадает в эту область. При проверке применимости правила такой набор составляется из элементов пространства категорий, соотнесенных с узлами, сопоставленными исходным вершинам правила, и из интервалов, соотнесенных с указанными узлами. Способы задания такой области не влияют на дальнейшее изложение. В системе Treeton в данный момент используется достаточно простой язык логических выражений. Он описывается в [Мальковский, Старостин 2006]. Например, согласование двух элементов, соотнесенных с A и B, по роду, числу и падежу на этом языке выражается следующим образом:

(3) A.CAS == B.CAS && A.NMB == B.NMB && (A.NMB == pl ? true : A.GEND == B.GEND)

Теперь перейдем к понятию применимости правила. Правило может быть применено к набору узлов мультиструктуры тогда, когда эти узлы можно сопоставить исходным вершинам правила так, что элементы пространства категорий, соответствующие узлам мультиструктуры, попадут в области пространства категорий, соответствующие вершинам правила. Кроме того, такое сопоставление определяет для каждой исходной дуги правила упорядоченную пару узлов мультиструктуры (исходные дуги по определению синтаксического правила могут соединять только исходные вершины). Необходимым условием применимости является то, чтобы для каждой такой пары узлов в мультиструктуре существовала дуга, которой бы сопоставлялся тот же элемент пространства типов связей, что и дуге правила. Следующим условием применимости правила является выполнение многомерных ограничений. При упомянутом сопоставлении становится возможным составить вектор, состоящий из набора элементов пространства категорий, соответствующих узлам мультиструктуры, и набора интервалов, соответствующих этим узлам. Правило применимо только тогда, когда построенный таким образом вектор попадает в область Ψ. Последним условием применения правила является то, что после добавления в граф мультиструктуры новых узлов и дуг и соответствующего расширения отображений (в первую очередь отображения *f*), полученный набор должен остаться мультиструктурой, т.е. граф полученной структуры должен допускать разбиение на синтаксические структуры. Это условие означает, что новые дуги не добавляют в граф циклы и ситуации, когда в одну вершину входит более одной дуги. Кроме того, все деревья, полученные после добавления узлов и дуг, удовлетворяют условиям 1 и 2 из определения синтаксической структуры. Процедура применения правила устроена таким образом, что условие 1 выполняется всегда. Отображение *f* всегда достраивается так, чтобы это условие выполнялось, – интервалы меняют свои размеры «автоматически». А вот условие 2 существенно ограничивает возможности применения правил. Без этого условия было бы возможно использование омонимичных вариантов анализа одного и того же слова в рамках одной структуры. См. рис. 4.

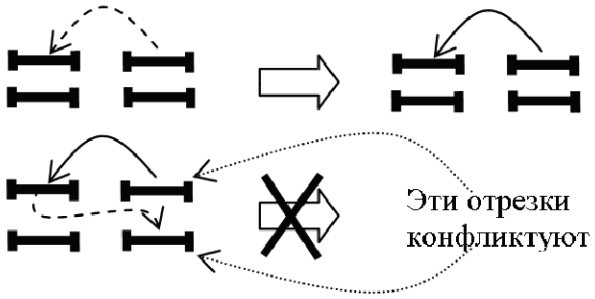


Рис.4. Применимость правил. Ограничение на непересечение отрезков тривиальных узлов делает второй вариант невозможным.

Дадим формальное определение применимости правила и процедуры применения правила и поясним их. Будем говорить, что синтаксическое правило R*($T(N_o \cup N_n, E_o \cup E_n), \alpha_o, \alpha_n, \beta, \Psi$) применимо к мультиструктуре* M(*G(N,E),e,f,g*), если можно задать такое биективное отображение F: $N_o \rightarrow N$, что

1. Для любого $n \in N_o$, $e(F(n)) \in \alpha_o(n)$.
2. Для любого $(r_1,r_2) \in E_o$, $r = (F(r_1), F(r_2)) \in E$ и $g(r) = \beta((r_1,r_2))$.
3. Вектор $(e(F(n_1)), e(F(n_2)),..., e(F(n_{|No|})), f(F(n_1)), f(F(n_2)),..., f(F(n_{|No|}))) \in \Psi$, $n_i \in N_o$, $1 \le i \le |No|$
4. После выполнения п. 1-3 процедуры применения правила M остается мультиструктурой.

Под *процедурой применения правила* понимается следующая последовательность действий:

1. Добавить в граф G новые вершины n_1', n_2',..., $n_{|Nn|}'$
2. Расширить отображение F следующим образом: $F(n_i) = n_i'$, где $n_i \in N_n$.
3. Добавить в граф G новые ребра $(F(r_{1i}), F(r_{2i}))$, где $(r_{1i}, r_{2i}) \in E_n$, $1 \le i \le |En|$
4. Расширить отображение e следующим образом: $e(n_i') = \alpha_n(n_i)(f(F(o_1)), f(F(o_2)),..., f(F(o_{|No|})))$, где $n_i \in N_n$, $o_j \in N_o$
5. Расширить отображение g следующим образом: $g((F(r_{1i}), F(r_{2i}))) = \beta(r_{1i}, r_{2i})$, $1 \le i \le |En|$
6. Для всех путей в графе G вида (x,y,…,z), $g((x,y)) = w$, $\exists f(z)$, в случае если интервал $f(x)$ не определен, принять $f(x)=f(z)$, в противном случае, если интервал $f(z)=(z_s,z_e)$ не вложен в интервал $f(x)=(x_s,x_e)$, принять $f(x) = (\min(x_s,z_s), \max(x_e,z_e))$. Следует отметить, что порядок обхода путей в описанной процедуре не влияет на результат.

В пунктах 4 и 5 вычисляются элементы пространств категорий и пространств типов связей для созданных узлов и дуг соответственно. Пункт 6 обеспечивает выполнение условия 1 в определении синтаксической структуры для всех синтаксических структур результирующей мультиструктуры. Перед началом работы процедуры из п. 6 все интервалы для добавленных вершин не определены. Легко заметить, что интервалы будут соотнесены только с теми новыми узлами мультиструктуры, *w*-потомками которых являются узлы, для которых интервалы определены.

Рассмотрим пример. На рисунках 5-6 приводится пример, иллюстрирующий то, как с помощью описываемого формального аппарата можно смоделировать поведение так называемых присказуемостных прилага-

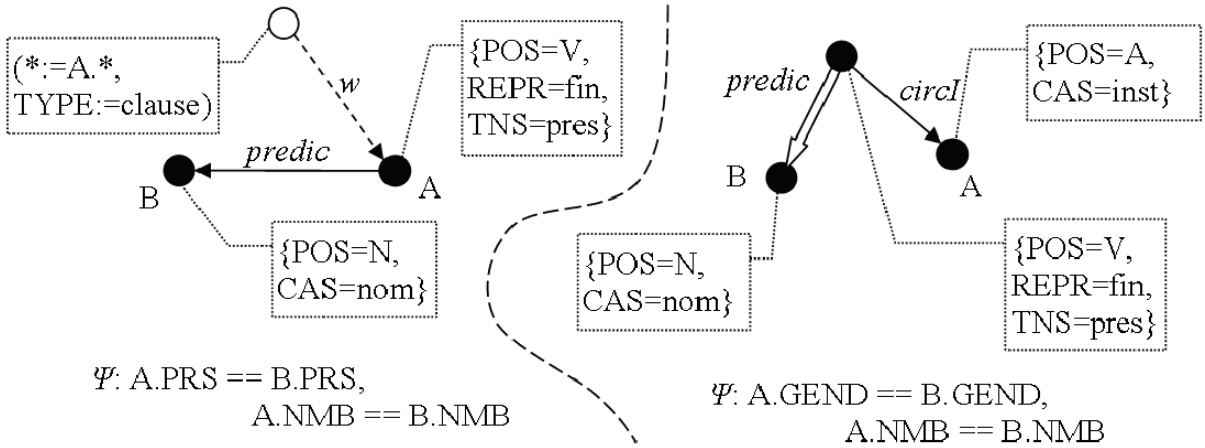


Рис.5. Пример синтаксических правил. Закрашенные окружности обозначают исходные вершины правила, а незакрашенные – производные. Одинарные стрелки обозначают производные дуги, а двойные – исходные.

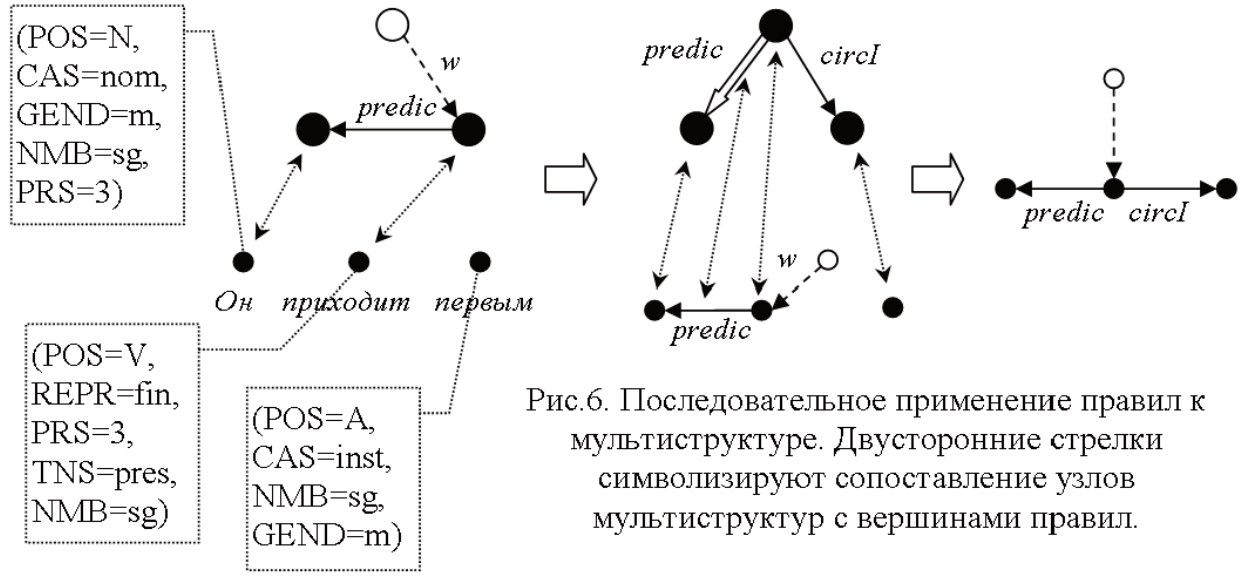


Рис.6. Последовательное применение правил к мультиструктуре. Двусторонние стрелки символизируют сопоставление узлов мультиструктур с вершинами правил.

тельных в русском языке. Эти прилагательные как будто входят в состав сказуемого, но по смыслу относятся к подлежащему или к прямому дополнению и согласуются с ними в числе и роде ([Тестелец, 2001: 48]). На рис.5 приводятся два правила, первое из которых (левое) соединяет подлежащее со сказуемым и помещает их внутрь нового элемента (клаузы), а второе позволяет присоединить прилагательное к сказуемому, но только в том случае, когда к сказуемому уже присоединено подлежащее, с которым это прилагательное согласуется. За согласование отвечает выражение *Ψ*. На рис. 6 показано, как описанные правила применяются к мультиструктуре, соответствующей результату морфологического анализа фразы *Он приходит первым*.

Первое из правил, приведенных на рисунках, типично для грамматик составляющих: два элемента объединяются в один с образованием нового элемента[2]. Второе правило, напротив, характерно для грамматик зависимостей: два элемента связываются друг с другом при выполнении определенных условий (в том числе и контекстных) – новых элементов не образуется.

Задача синтаксического анализа

Теперь, когда мы ввели понятия синтаксической структуры, мультиструктуры и синтаксического правила, мы можем формально поставить *задачу синтаксического анализа*.

[2] Для полного соответствия с грамматиками составляющих следует произвести следующую модификацию первого правила: дополнить выражение *Ψ* требованием примыкания одной исходной вершины к другой, убрать производную связь *predic*, а вместо нее провести производную *w*-связь от производной вершины к вершине, обозначающей существительное.

Будем считать, что анализируемый текст представляет собой набор из n слов $w_1, w_2, ..., w_n$. Будем считать, что слову w_i соответствует интервал числовой оси (s_i, e_i), причем $e_i < s_{i+1}$ для $1 \le i < n$.

Будем также считать, что слову w_i сопоставлен набор элементов пространства категорий (c_{ij}) $1 \le j \le m_i$, соответствующий вариантам морфологического анализа.

Тогда мы можем сконструировать *входную мультиструктуру* $M(G(\{n_{ij}\},\varnothing), e: e(n_{ij})=c_{ij}, f: f(n_{ij})= (s_i, e_i),\varnothing)$, где $\{n_{ij}\}$ – множество узлов, $1 \le i \le n$, $1 \le j \le m_i$.

Обозначим область числовой оси $\cup(s_i, e_i)$ как T. Будем называть ее областью текста.

Будем говорить, что синтаксическая структура покрывает определенную область числовой оси, если объединение интервалов, соотнесенных со всеми тривиальными узлами этой структуры, включает в себя эту область. Напомним, что тривиальными узлами мы называем такие узлы, из которых не исходит ни одна w-связь.

Задача синтаксического анализа ставится следующим образом:

Для заданного множества синтаксических правил Ω, заданной области текста T и заданной входной мультиструктуры M найти все синтаксические структуры, покрывающие область T, которые можно получить в процессе применения правил из Ω к M.

Следует отметить, что можно легко записать систему правил, при которой поставленная задача независимо от входной мультиструктуры имеет бесконечное количество решений. Причем сами эти решения не будут иметь никакого отношения к реальным языковым структурам, наблюдаемым лингвистами. При этом очевидно, что если с помощью предложенного формализма удается промоделировать реальные правила синтаксиса, то решение поставленной задачи для некоторых мультиструктур будет отсутствовать (неграмматичные предложения), а для остальных будет давать конечное (причем небольшое) количество вариантов. Это связано с тем, что в наблюдаемых естественных языках не встречается ситуаций, когда какое-то предложение имеет бесконечное или даже очень большое количество синтаксических интерпретаций. Реальные системы правил, которые авторам пока удалось создать, дают, безусловно, конечное количество решений для любых предложений, но, к сожалению, в силу бедности набора различаемых компьютером категорий, иногда очень большое. В силу того, что возможностей по расширению набора категорий у авторов пока нет (нет словарей сочетаемости единиц, например), было решено смириться с тем, что результатов анализа может быть в общем случае очень много. Авторы попытались добиться того, чтобы алгоритм решения задачи обладал способностью находить более «качественные» синтаксические структуры прежде менее «качественных». Для оценки «качества» структуры авторы используют аппарат *лингвистически содержательных штрафных функций* – функций, соотносящих с любой мультиструктурой некоторое действительное неотрицательное число, называемое штрафом. Они называются лингвистически содержательными потому, что опираются на статистические свойства синтаксических структур, известные в лингвистике. Например, на то свойство, что большинство реальных синтаксических структур проективно [Гладкий,1985]. Методы описания штрафных функций приводятся в [Мальковский, Старостин 2006].

В следующем разделе дается описание переборного алгоритма решения задачи синтаксического анализа, используемого в данный момент в системе Treeton. Этот алгоритм перебирает различные комбинации применения правил «под управлением» штрафной функции. Следует сразу сказать, что помимо чисто лингвистических свойств синтаксических структур, входящих в мультиструктуры, штрафной функцией также учитываются некоторые более простые топологические характеристики мультиструктур. Например, уровень связности – количество нетривиальных синтаксических структур, входящих в мультиструктуру. Точный вид штрафных функций, используемых авторами, в этой статье не приводится, т. к. исследование, касающееся поведения нижеследующего алгоритма в зависимости от вида штрафной функции, пока еще не закончено. Здесь мы ограничимся лишь некоторыми соображениями по этому поводу.

Алгоритм синтаксического анализа

Пусть дано множество синтаксических правил Ω, область текста T, входная мультиструктура M_0 и штрафная функция $p(M)$.

Опишем алгоритм синтаксического анализа в виде последовательности действий, совершаемых с тремя множествами: множеством обработанных мультиструктур *DONE*, множеством мультиструктур-кандидатов *TODO* и результирующим множеством синтаксических структур *RESULT*.

Перед началом работы алгоритма $DONE=\varnothing$, $TODO=\{M_0\}$, $RESULT=\varnothing$

Алгоритм выглядит следующим образом:

пока множество TODO не пусто {
Взять из множества TODO мультиструктуру M, для которой p(M) минимально.
Если M содержит синтаксические структуры, покрывающие T, добавить
их в множество RESULT.

Вычислить множество правил $\Omega' \subseteq \Omega$, применимых к M.
Для каждого $r \in \Omega'$ {
Получить мультиструктуру M' , применив r к M.
Если $M' \notin$ DONE, добавить M' к множеству TODO.
}
Добавить M к множеству DONE.
}

После работы алгоритма множество *RESULT* будет содержать решение задачи синтаксического анализа, т.е. все структуры, покрывающие область *T,* которые можно получить применяя правила из Ω к исходной мультиструктуре.

Скажем несколько слов о том, как запрограммирован приведенный алгоритм. Структуры данных, с которыми работает алгоритм (тринотации и множества тринотаций), описаны в [Мальковский, Старостин 2006]. Для синтаксических правил созданы специальные структуры данных, оптимизированные для быстрого выполнения операции вычисления множества правил, применимых к данному множеству тринотаций (мультиструктуре). При загрузке системы синтаксических правил над всеми одномерными шаблонами, присутствующими в правилах, строится индекс, который впоследствии позволяет быстро отыскивать, какие правила могли бы сработать. Наборы атрибутов входных тринотаций как бы «просеиваются» через этот индекс, и только те правила, во все шаблоны которых попала хотя бы одна тринотация, переходят на следующий уровень, на котором проверяются многомерные ограничения и возможность применения правила с топологической точки зрения (появление циклов, пересечение тривиальных узлов и т.п.).

Для того, чтобы не допускать повторений мультиструктур в ходе перебора, используются хэш-таблицы. Мультиструктуры удается эффективно хэшировать, т.к. каждая мультиструктура однозначно определяется массивом пар вида <*правило + массив ссылок на тринотации, соотнесенные с исходными вершинами правила*>. Хэш-функция строится по этому массиву.

В целях экономии мультиструктуры не хранятся в памяти в виде деревьев. Они хранятся только в виде уже упомянутых массивов пар вида <*правило + массив ссылок на тринотации*>. В тот момент, когда требуется посчитать штраф мультиструктуры или вычислить множество применимых к мультиструктуре правил, мультиструктура конвертируется в набор деревьев, а после указанных операций отведенная память освобождается.

Существенным недостатком нынешней реализации алгоритма является то, что системе часто приходится выполнять одни и те же последовательности применения правил, а следовательно и предварительных вычислений. Например, одну и ту же предложную группу машине приходится собирать десятки раз, применяя одни и те же правила совершенно одинаковым образом. Представляется целесообразным сделать так, чтобы машина помнила какие-то подструктуры, которые удалось построить в определенных ветвях переборного процесса, и могла применять их в других ветвях как единое целое, без дополнительных вычислений. В данный момент авторы работают над такого рода оптимизацией существующего алгоритма.

Вычислительную сложность приведенного алгоритма оценить достаточно сложно, т. к. поведение переборного процесса существенно зависит от вида штрафной функции и вида системы правил. Можно сказать, например, что при моделировании грамматики зависимостей алгоритм показывает себя лучше, чем при моделировании грамматик смешанного типа. При моделировании последних на процесс перебора очень влияет то, насколько сильно штрафуется количество нетривиальных структур внутри одной мультиструктуры. Заметим, что разрешать только одну нетривиальную структуру можно только в случае, когда новые узлы мультиструктуры не порождаются вообще (случай грамматик зависимостей). При работе с грамматиками смешанного типа такой запрет может вообще не позволить отыскать правильный анализ. Например, если считать, что при сочинении двух существительных в единственном числе возникает новая сущность, включающая их, характеризуемая множественным числом, то проанализировать фразу *Вася и Петя любят яблоки и апельсины* упомянутый запрет не позволит. Дело в том, что одну из двух сочинительных групп придется собирать в тот момент, когда другая часть дерева будет уже создана, а привязать группу к глаголу можно будет только на следующем шаге. Значит, существование хотя бы двух нетривиальных структур в одной мультиструктуре допустить необходимо. Можно построить более сложные примеры, для которых это число нужно будет увеличить. Чем грозит сильное увеличение такого числа? Тем, что машине будет позволено перебирать бессвязные структуры и в тех случаях, когда это совершенно не нужно с точки зрения здравого смысла, – количество комбинаций возможностей применения правил будет огромно.

Еще одной проблемой, о которой хотелось бы сказать, является проблема *ранних ошибок*. Речь идет о тех случаях, когда на каком-то шаге применяется некоторое правило, после чего тратится большое количество времени и ресурсов на перебор комбинаций в контексте примененного правила и все ветви этого переборного процесса приводят к тупику или к сильно штрафованному варианту, причем штраф всегда возникает из-за конфлик-

та с исходным правилом. Другими словами, ошибка допускается рано, а обнаруживается поздно. Такие ошибки часто связаны с проективностью: проводится, например, связь, обрамляющая какой-то небольшой отрезок текста (1-2 слова), после чего система перебирает множество комбинаций во внешней по отношению к отрезку области и каждый раз, проводя связь внутрь этого отрезка, штрафует очередной вариант за непроективность. Основная причина описанного явления заключается в том, что штрафная функция не учитывает «перспективность» той или иной мультиструктуры, она лишь оценивает ее нынешнее состояние – по принципу «сейчас проблем нет, значит все в порядке». Создание и описание штрафных функций, способных оценивать перспективность мультиструктур, представляет, на наш взгляд, важную задачу.

Заключение

В этой статье был описан алгоритм синтаксического анализа, используемый в анализаторе Treevial системы морфо-синтаксического анализа Treeton, и были введены формальные понятия, с которыми он оперирует. Обсуждались также некоторые свойства этого алгоритма. В данный момент алгоритм обладает рядом недостатков, которые авторы надеются в скором времени исправить.

Планируется разработка алгоритма, в ходе работы которого на текущем шаге система будет помнить варианты разбора подцепочек, обнаруживаемые на предыдущих шагах (по аналогии с алгоритмом CYK для контекстно-свободных грамматик [Cocke, Schwartz 1970] , [Kasami, 1965], [Younger, 1967]).

В планы авторов входит исследование подходов к распараллеливанию существующего алгоритма. Представляется, что использование активных агентов, параллельно работающих над множеством мультиструктур, может существенно повысить эффективность.

Авторы также не оставляют надежды создать большой свод синтаксических правил для русского языка, что позволит еще более тщательно отладить существующий алгоритм анализа.

Список литературы

1. [Гладкий, 1985] – А.В.Гладкий. Синтаксические структуры естественного языка в автоматизированных системах общения. – М.: Наука, 1985.
2. [Гладкий, Мельчук, 1969] – Гладкий А. В., Мельчук И. А. Элементы математической лингвистики. М.: Наука, 1969.
3. [Мальковский, Старостин, 2006] – М.Г. Мальковский, А.С.Старостин. Модель синтаксиса в системе морфосинтаксического анализа «TREETON» // Труды международной конференции Диалог’2006. М.: изд-во РГГУ, 2006. с. 481-492.
4. [Тестелец, 2001] – Тестелец Я.Г. Введение в общий синтаксис. М.: РГГУ. 2001
5. [Хомский, 1962] – Хомский Н. Синтаксические структуры. – В кн.: «Новое в линвистике», вып. II, 1962, 412-526
5. [Cocke, Schwartz 1970] – John Cocke And Jacob T Schwartz (1970). Programming languages and their compilers: Preliminary notes. Technical report, Courant Institute of Mathematical Sciences, New York University.
6. [Johnson, 1998] – M. Johnson. 1998. PCFG models of linguistic tree representations. Computational Linguistics, 24(4):617-636.
7. [Kasami, 1965] – T. Kasami (1965). An efficient recognition and syntax-analysis algorithm for context-free languages. Scientific report Afcrl-65-758, Air Force Cambridge Research Lab, Bedford, MA.
8. [Mel’cuk , 1988] – Mel’cuk, I.A. Dependency Syntax: Theory and Practice. SUNY Series in Linguistics, Mark Aronoff, series editor. State University of New York Press, Albany, 1988
9. [Pollar, Sag, 1994] – Carl Pollard, Ivan A. Sag (1994): Head-Driven Phrase Structure Grammar. Chicago: University of Chicago Press.
10. [Schneider, 1998] – Garold Schneider, A Linguistic Comparison of Constituency, Dependency and Link Grammar. http://www.ifi.unizh.ch/cl/study/lizarbeiten/lizgerold.pdf // 20.03.07
11. [Sleator & Temperley, 1991] Daniel D. K. Sleator and Davy Temperley, Parsing English with a Link Grammar, Technical Report CMU-CS-91-196, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA, 1991.
12. [Younger, 1967] – Daniel H Younger (1967). Recognition and parsing of context-free languages in time n3. Information and Control 10(2): 189-208.