ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № 3(17), 2013, с.??—??

УДК 004.042, 004.822

С. М. Салибекян

(Объектно-атрибутный подход к обработке графовых структур данных)[1]

Аннотация. Статья посвящена описанию возможностей, которые предоставляет объектно-атрибутный (ОА) подход к организации вычислительного процесса, для работы с графовыми (сетевыми) структурами данных. Внимание уделено формальному аппарату (ОА-грамматика), позволяющему описывать трансформацию графовых структур посредством применения правил преобразования графа. ОА-грамматика обладает достаточно широкими возможностями и может быть использована как для синтеза графа, так и для его модификации. В статье приводится пример использования ОА-подхода для семантического анализа естественного языка. Также в статье рассказывается о программной реализации блока, осуществляющего обработку ОА-графа.

Ключевые слова и фразы: Граф-трансформирующая система, объектно-атрибутная архитектура, вычислительная система с управлением потоком данных, сложно структурированные данные.

Введение

Граф-транформирующая система (graph rewriting /transformation system или графовая грамматика (graph grammar) – это система, преобразующая графовую (сетевую) структуру данных. Преобразование графовых структур необходимо во многих областях: семантическое распознание текста на естественном языке, моделирование сложных динамических систем, геоинформационные системы, техническое зрение, компьютерная графика и т.д. Работы в данном направлении ведутся на теоретическом и практическом уровнях: созданы формальные модели граф-трансформации (Regular tree grammar, head driven phrase structure grammar

[1] (Рекомендована к публикации.... Поддержана...!)

(HPSG) [1]), а также произведено программное обеспечение для решения подобной задачи (GraphSynth, GrGen.NET, GReAT, VIATRA, Fujaba). Однако существующие формальные модели имеют большие ограничения, например, они могут только генерировать граф типа «дерево». А программное обеспечение не отличается простотой эксплуатации или универсальностью и не стоится на основе формальной формали.

Мы же предлагаем новое решение проблемы – формальное описание граф-трансформирующей системы на основе объектно-атрибутного (ОА) подхода к организации вычислений и структур данных [2]. ОА-подход обеспечивает предельную гибкость при работе с графовыми структурами данных: динамический синтез и модификация ОА-графа (граф, построенный по ОА-принципам) без опасности нарушения целостности данных. Такая гибкость обеспечивается благодаря тому, ОА-граф похож на конструктор, элементы которого могут быть добавлены, удалены или модифицированы непосредственно во время вычислительного процесса.

В результате проделанной исследовательской работы нами были созданы две формальные модели. Одна – для описания преобразований графа через последовательность элементарных операций (ОА-грамматика), способная описать преобразования без привязки к какой-либо архитектуре вычислительной системы, реализующей трансформацию графа. Другая – для описания вычислительного процесса в ОА-системе для преобразования сложно структурированных данных (ОА-автоматная сеть). Созданные формальные модели были нами использована в разработке системы семантического анализа естественного языка, представляющей собой граф-трансформирующую систему, преобразующую ОА-граф специального формата, содержащий описания слов исходного текста, в семантическую структуру, отражающую смысл текста. Поэтому в статье также будет приведено описание программного модуля, осуществляющего трансформацию ОА-графа в данной системе на примере семантического анализа естественного языка.

1. Формализм для описания трансформации графа

Основными понятиями ОА-подхода являются информационная пара (ИП), информационная капсула (ИК) и ОА-граф. ИК представляет собой множество ИП, ИП же является двойкой <a,l>, где a – атрибут, l – нагрузка ИП (l∈{$\Re$∪nil∪S∪Ω} (L – множество данных в нагрузке, $\Re$ – множество рациональных чисел, nil – обозначение пустой нагрузки, Ω - множество индексов ИК (каждая ИК имеет свой уникальный индекс); S ∈ $\sum^*$ – множество цепочек символов, принадлежащих алфавиту $\sum$). ОА-подход организации графа (сети) заключается в том, что вершина графа ассоциируется с информационными капсулами (ИК), дугами же являются ссылки на ИК, расположенные в нагрузках информационных пар (ИП) (рис. 1). Таким образом, все ИП по виду нагрузки можно разделить на два класса: константные (содержат в нагрузке константу) и ссылочные (содержат в нагрузке индекс ИК). Первый вид необходим для хранения данных, второй – для описания топологии графа.

Для оперирования ОА-графом был разработан формализм, позволяющий производить его синтез и модификацию – ОА-грамматика. Грамматикой формализм называется по аналогии с формальными грамматиками, применяемыми для анализа языка, и в том числе способных произвести синтез графовых структур (атрибутная грамматика, regular tree grammar HPSG и т.п.). По сути, ОА-грамматика является граф-трансформирующей системой (англ. graph rewriting/transformation system). Она описывает только последовательность трансформации графа, и никак не привязывается к архитектуре той вычислительной системы, на которой данная трансформация будет осуществляться. Формально ее можно определить как четверку (1):

OAG={A,L,P,I,G}, где (1)

A – множество атрибутов ИП (A⊆N);

L – множество нагрузок (L={const∪Ω}, где const – множество констант), Ω - множество индексов ИК (каждая ИК, в

том числе синтезированная во время трансформации графа, получает свой уникальный индекс);

G – исходный граф, который будет подвергаться трансформации;

I – множество индексов правил ОА-грамматики ($I \subseteq N$).

P – совокупность правил преобразования ОА-графа.

Правила ОА-грамматика, описывающие трансформацию ОА-графа, записываются с помощью специальной нотации. Каждое правило состоит из трех частей – левой, где указывается шаблон искомого подграфа, и правой, где описывается трансформация данного подграфа, а также центральной, хранящей ограничения на значения констант, хранящихся в искомом подграфе. Центральная часть является необязательной. Шаблон в левой части задаются с помощью множеств ИП (множество ИП – это ИК) различных типов: упорядоченное – неупорядоченное, необязательные и недопустимые элементы и т.д. Если в преобразуемом графе встречается подграф, совпадающий с шаблоном, описанном в левой части правила, то правило называется разрешенным (т.е. оно может быть выполнено), иначе оно называется запрещенным. Каждое правило ОА-грамматики имеет свой номер (индекс); если в один момент в ОА-грамматике находится сразу несколько разрешенных правил, то выполняется то из них, у которого наименьший номер. Если в правиле имеются ограничения на значения констант (центральная часть), то правило считается разрешенным только в том случае, когда все ограничения выполняются. Преобразования ОА-графа, которые необходимо произвести, если правило разрешенное, задаются с помощью нескольких операций: конкатенация (соединение) множеств, удаление первого или последнего элемента множества.

Теперь перейдем к описанию нотации правила. Итак, левая и правая части правила в ОА-грамматике отделяется от правой знаком «→», а левая от центральной части, если она присутствует в правиле, отделяется с помощью знака «:». Для удобства изложения материала, разделим все обозначения в нотации правила ОА-

грамматики на четыре группы: общие, а также используемые в левой, правой и центральной частях.

Общие обозначения используются во всех частях правила, к ним относятся:

1. Атрибуты ИП обозначаются с помощью мнемоник.
2. Константы и указатели (индексы ИК из множества Ω) также обозначаются с помощью мнемоник.
3. = - обозначение ИП (слева от знака «=» располагается символьное обозначение атрибута, справа – обозначение нагрузки (нагрузка может быть константой или ссылкой)).
4. Name{} – обозначение ИК (обозначение одной или нескольких ИП, входящих в ИК, помещаются внутри скобок; Name – идентификатор ИК; для удобства восприятия человеком каждый индекс ИК заменяется мнемоническим выражением).
5. ...=Name{} – ссылка на ИК в нагрузке ИП (обозначение ссылки на ИК Name может отсуствовать);

В левой части правила используются обозначения:

1. ( ) – упорядоченное множество ИП (в множество могут входить ИП, ИК, или фрагменты ОА-графа).
6. { } – неупорядоченное множество.
7. [ ] – множество необязательных элементов подграфа (этот элемент(ы) может присутствовать или отсутствовать в ОА-графе).
8. $< >$ - недопустимое множество элементов

4. $\oplus$ - множество типа исключающее ИЛИ (т.е. в ОА-графе должен присутствовать только один элемент из множества). Данное обозначение стоит между обозначений элементов, входящих в ИК.

5. || - множество типа ИЛИ (один или несколько элементов). Данное обозначение стоит между обозначений элементов, входящих в ИК.

В правой части трансформация фрагмента ОА-графа задается с помощью нотации:

1. * - конкатенация (соединение) двух ИК (все ИП, из двух ИК объединяются в одну ИК).
2. Atr=*{...} – Конкатенация к ИК по адресу из нагрузки ИП.
3. #Point – копирование ИК с индексом Point, указанным в левой части правила.
4. Atr= #Point Указатель на копированную ИК помещается в нагрузку ИП
5. ^{F} Point - первая ИП в капсуле по адресу Point.
6. ^{L} Point - последняя ИП в капсуле по адресу Point.
7. Variable(...) – Значение переменной по умолчанию. Если в левой части правила задан альтернативный элемент ОА-графа с заданной переменной, и этот элемент не встречается в ОА-графе, то при вычислениях используется значение по умолчанию, которое указывается в скобках.
8. +,-,*,/ и т.д. – арифметические операции, для получения константы в нагрузке ИП.

   Центральная часть используется для записи условий, накладываемых на значения, хранящиеся в нагрузках ИП, формирующих ОА-граф. В данной части используются Здесь используются следующие обозначения:

1. =, <>, <= и т.д. – Логические операции для проверки условий разрешенности правила ОА-грамматики.
2. ( ) – Операция группировки логических операций между условиями.
3. ; - Логическая операция «ИЛИ» между условиями.
4. « » (пробел) или «,» - Логическая операция «И» между условиями.

Условия могут присутствовать и в левой части правила – они записываются непосредственно в нагрузке ИП и выделяются круглыми скобками. Та переменная, которая изначально хранится в нагрузке, выделяется жирным шрифтом. Например, записи {Atr=(0≤**Value**≤ToopVal)}; аналогична запись {Atr=Value}: 0≤Value≤ ToopVal →...

Используемые в ОА-грамматике идентификаторы (ссылки) записываются в виде текстовой строки (мнемоники). Каждый идентификатор обозначает индекс ИК (каждая ИК имеет свой уникальный индекс). Так, если мнемоника идентификатора записана перед записью, обозначающую ИК, то он обозначает идентификатор данной ИК. Если идентификатор записан после знака «=», то подразумевается, что в нагрузке данной ИП хранится индекс ИК, обозначаемый данным идентификатором. Правило (2) обозначает, что в результате его выполнения ИК с индексом temp будет помещено в нагрузку ИК с атрибутом SemProp.

temp{Atr=0} → {SemProp=temp} (2)

Следует еще рассказать об операции конкатенации нагрузки ИП. Иногда требуется конкатенация ИК, индекс которой хранится в нагрузке ИП. В этом случае применяется следующая запись: ...=*temp , где temp – идентификатор ИК. Пусть изначально ОА-граф был {SemProp={Atr=0 Atr=5}}, то после операции (3) получится {SemProp={Atr=0 Atr=5 Atr=10 Atr=15}}.

{SemProp} → { SemProp=*{ Atr=10 Atr=15}} (3)

В качестве примера приведем описание модификации ОА-графа, используемой при анализе естественного языка. Анализ текста в ОА-системе начинается с формирования списка толкований слов естественного языка (рис. 1 а), в который помещаются ИК с описаниями толкований слов. Список представляет собой ОА-граф специального формата, который с помощью последовательности операций ОА-грамматики будет преобразовываться в семантическую структуру, описывающую смысл текста. Перечень элементов списка представляет собой ИК, каждая ИП которой содержит ссылку на элемент этого списка (т.е. каждому квадратику на рис. 2 соответствует ИП с индексом ИК, являющейся элементом списка).

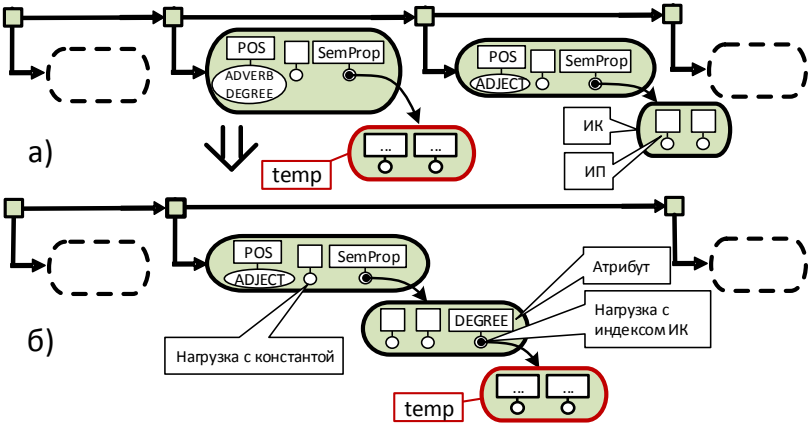

Рис. 1 Трансформация ОА-графа при склейке описаний наречия степени с прилагательным

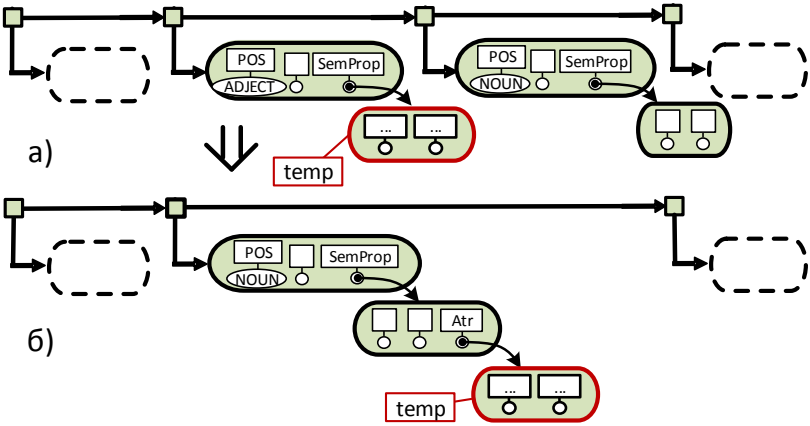

Рис. 2. Трансформация ОА-графа при склейке описаний прилагательного с существительным

При семантическом анализе языка в качестве модификации ОА-графа в основном применяется операция склейки, когда описание семантических свойств зависимого слова «приклеивается» к описанию семантических свойств главного слова, а описание зави-

симого слова удаляется из списка. Рассмотрим операцию склейки наречия степени (данная часть речи указывает интенсивность свойства: например, «очень», «слишком») и прилагательного (рис. 1). На рисунках 1,2 используются следующие атрибуты: POS (от Part Of Speach) – часть речи, SemProp (от semantic properties) – семантические свойства слова, Atr – атрибут (свойство) объекта, заданного словом, ADVERB_DEGREE – наречие степени, NOUN – существительное; ADJECT (от adjective) – прилагательное. Результат склейки представлен на рис. 1 б: семантические свойства наречия степени прикрепляются с семантическим свойствам существительного (атрибут DEGREE - степень). Аналогично осуществляется склейка существительного и прилагательного (рис. 2), только здесь описание семантических свойств «приклеивается» с атрибутом Atr, обозначающим свойство объекта. Запишем данные операции с помощью правила ОА-грамматики (4,5). Метка temp служит идентификатором (индексом) ИК с описанием семантических свойств, заданных наречием степени. Elem – атрибут элемента списка.

{Elem={POS=ADVERB_DEGREE SemProp=temp} Elem={POS= ADJECT}} → {Elem={POS=ADJECT SemProp=*{Atr=temp}}}(4)

{Elem={POS=ADJECT SemProp=temp} Elem={POS=NOUN}} → {Elem={POS= NOUN SemProp=*{Atr=temp}}} (5)

Идентификатор temp хранит индекс ИК с описанием степени в (4) и семантических свойств прилагательного в (5). Во время анализа текста данные правила должны быть выполнены в строгом порядке (сначала (4) потом (5)), иначе разбор текста будет произведен некорректно.

С помощью ОА-грамматики можно работать не только с графами типа «дерево», но и любой топологии. Допустим, нам необходимо в левой части правила описать шаблон графа с тремя вершинами, образующими цикл. Это делается так (6). Т.е. в первой ИК есть ссылка на вторую ИК, во второй на третью, в третья ссылается на ИК с индексом temp (т.е. на первую ИК). Аналогично, и

в правой части правила можно описать синтез ОА-графа любой топологии.

temp{Atr={Atr={Atr=temp}}} (6)

О применении правил ОА-грамматики в работе системы семантического анализа текста написано в [3].

2. Реализация трансформации графа в вычислительной системе

С помощью разработанного аппарата ОА-грамматики появилась возможность описания последовательности операций преобразования ОА-графа. Однако остался вопрос практической реализации преобразования графа на вычислительной системе. Можно было бы пойти по пути создания устройства (программного или аппаратного), которое непосредственно реализовывало бы правила ОА-грамматики. Однако такое решение, во-первых, усложнило бы устройство и, во-вторых, сделало бы поиск шаблона подграфа из левой части правила неоптимальным. Поэтому было принято решение сделать устройство достаточно «глупым», чтобы пользователь мог более детально (и оптимально) описывать алгоритм поиска шаблона подграфа; а процесс создания граф-трансформирующей системы, таким образом, разделился на две фазы: первая – описание трансформации ОА-графа с помощью правил ОА-грамматики, второй – создание программы для устройства модификации графа, исходя из правил ОА-грамматики.

Т.к. устройство входит в состав вычислительной системы ОА-архитектуры, то оно реализуется в виде функционального устройства (ФУ), управляемого милликомандами (атрибутированными данными). Для описания работы такого ФУ можно, например, воспользоваться формализмом (ОА-автоматная сеть), предложенным в [4]. Далее приведем пример реализации ФУ модификации ОА-графа, выступающее в роли лингвистического процессора. Его задачей является преобразование ОА-графа, содержащего список толкований слов исходного текста в семантическую структуру, хранящую смыл текста. Назовем такое устройство GraphModifier.

Следует отметить, что GraphModifier, предназначенный для семантического разбора естественного языка, уже реализован программно.

Итак, GraphModifier работает следующим образом. Сначала устройство производит поиск ИК, которая входит в шаблон искомого подграфа (ИК задается программной). После того, как шаблон ИК будет найден в ОА-графе, для дальнейшего поиска используется так называемый «бегунок» - указатель на анализируемую ИК ОА-графа. Для поиска подграфа GraphModifier обрабатывает набор милликоманд (ИП для управления ФУ), с помощью которых он перемещает «бегунок» по ОА-графу, а также набор милликоманд для сравнения ИК из ОА-графа, на которых находится «бегунок», с определенными шаблонами ИК. С помощью таких милликоманд и производится обход подграфа «бегунком» и его сравнение с шаблоном. Последовательность такого обхода задается пользователем в виде ОА-программы (последовательности милликоманд, поступающих на ФУ GraphModifier). Такие милликоманды, могут оперировать с ИК, на которых находится «бегунок»: выдача адреса ИК, на которой находится «бегунок», исключение ИК из ОА-графа; удаление ИК из ОА-графа. Также имеются милликоманды, работающие с отдельными ИП: добавление ИП в ИК, удаление ИП из ИК, изменение атрибута и нагрузки ИК.

Для реализации заданного функционала GraphModifier должен иметь определенный контекст – совокупность внутренних регистров (переменных), определяющих состояние ФУ. Итак, в контекст ФУ входят:

- Указатель (индекс ИК) на ОА-граф, подвергающийся модификации.

- «Бегунок» (traveler) - идентификатор текущей ИК, где ищется совпадение ИК.

- «Бегунок» следующего элемента ОА-списка (NextTraveler) – в этот регистр автоматически помещается индекс следующего элемента (ИК) списка.

- Стек индексов ИК, пройденных бегунком (TravelStack). Стек необходим для того, чтобы бегунок мог возвращаться обратно по пройденному пути.

- Идентификатор ИК с миллипрограммой, активизирующейся в том случае, когда будет найдена ИК, совпадающая с шаблоном (SerchProg). Идентификатор выполняемой милликоманды (миллипрограмма представляет собой последовательность милликоманд, оформленную в виде ИК; форматы милликоманды и ИК совпадают (только они по-разному интерпретируются устройством: милликоманда – как управляющая команда, а информационная ИП – как данные для обработки). После того, как милликоманда выполнена, происходит переход к следующей милликоманде из ИК.

- Идентификатор ИК (NoSerchProg) с миллипрограммой, активизирующейся в том случае, если не будет найдена ИК, совпадающая с шаблоном.

- Идентификатор ИК (TravelProg) с миллипрограммой, активизирующейся в том случае, когда ИК из нагрузки милликоманды совпадает с ИК, на которую указывает бегунок.

- Идентификатор ИК (NoTravelProg) с миллипрограммой, активизирующейся в том случае, когда ИК из нагрузки милликоманды не совпадает с ИК, на которую указывает бегунок.

Замечание: считается, что ИК шаблона совпадает с ИК ОА-графа в том случае, когда для каждой ИП из ИК шаблона находится идентичная ИП в ИК ОА-графа.

Далее необходимо перечислить милликоманды для управления GraphModifier. Начнем с милликоманд, предназначенных для поиска подграфа:

- GraphSet – установить идентификатор одной из ИК ОА-графа, где будет происходить поиск подграфа.

- SerchProgSet – установить идентификатор миллипрограммы, активизирующейся в случае нахождения первой ИК из шаблона.

- NoSerchProgSet – установить миллипрограмму, активизирующейся, если не найдена первая ИК из шаблона.

- TravelProgSet – установить TravelProg.

- NoTravelProgSet – установить NoTravelProg.

- Cmp – сравнить ИК, на которую указывает бегунок с ИК из нагрузки милликоманды.

NextCmp – то же самое для следующего элемента списка.

- YesCmpProgSet – установить YesCmpProg.

- NoCmpProgSet – установить NoCmpProg.

- NextTravelProgSet – установить миллипрограмму, запускаемую при совпадении ИК под «бегунком» следующего элемента ИК.

Перечислим милликоманды управления «бегунком»:

- Search – проверка совпадения ИК из нагрузки милликоманды с ИК, на которую указывается бегунок (в случае совпадения запускается миллипрограмма по указателю SerchProg, иначе NoSerchProgSet).

- Go – перейти по ссылке: в нагрузке милликоманды указывается индекс атрибута ИП, в которой находится индекс ИК, куда следует перейти «бегунку».

- NextGo – то же самое для следующего элемента списка.

- Back – вернуть бегунок на одну ИК назад по пройденному в ОА-графе маршруту;

- BackNext – то же самое для бегунка следующего элемента списка.

- GoBack – вернуться на одну капсулу по пройденном бегунком маршруту, и выполнить милликоманду Down (т.е. вернуться, а затем перейти по другому идентификатору.

- NextGoBack – то же самое для бегунка следующего элемента списка.

NextCmp – сравнение следующего элемента списка с ИК из нагрузки милликоманды.

Приведем несколько милликоманд для модификации ОА-графа:

LineOut, NextOut, PrevOut – выдать ссылку на ИК, на которую указывает «бегунок», «бегунок» следующего и предыдущего элементов списка.

LinePop, NextPop, PrevPop – выдать ссылку на ИК, на которую указывает «бегунок», «бегунок» следующего и предыдущего элементов списка и исключить ИК из списка (милликоманды необходимы для того, чтобы переместить ИК в другое место ОА-графа).

Concat – конкатенация ИК, адрес которой указан в нагрузке милликоманды, к ИК, на которую указывает бегунок.

NextConcat – то же самой для бегунка следующего элемента списка.

Теперь опишем правила ОА-грамматики (4,5) на ОА-языке (рис. 1,3). Пусть устройство называется GModifier, мнемоника милликоманд для него будет указывается после названия устройства через знак точки. Комментарии в программе обозначаются «//». Для более компактной записи будем опускать название GModifier и ставить перед мнемоникой милликоманды только знак «.». Следует отметить следующую тонкость: милликоманда GModifier.Pop=temp записывает индекс ИК в ИП {DEGREE=temp(nil)!}, т.е. индекс ИК с семантическими свойствами записывается в нагрузку ИП с атрибутом DEGREE и затем эта ИП конкатенируется с ИК семантических свойств главного слова.

```
GModifier.GraphSet=Graph // Graph – идентификатор графа

// Склейка наречия степени с прилагательным
GModifier.SerchProgSet={//Подпрограмма поиска прилагат-го
  .NextTravelProgSet={
    .Go=SemProp // Перевести бегунок на нагрузку ИП с атрибутом SemProp
    .Pop=temp//Выдать индекс ИК с семантич. свойствами наречия степени
    .NextGo=SemProp // перевести бегунок следующей строки
    .NextConcat={DEGREE=temp(nil)!} // конкатенация
    .Del // Удалить текущий элемент списка
  }
  .NextCmp={POS=ADVERB} // Поиск прилагательного
}
GModifier.Search={POS=ADVERB_DEGREE}
```

```
// Склейка прилагательного с существительным
GModifier.SerchProgSet={//Подпрограмма поиска существительного
  .NextProgSet={.Go=SemProp
    .Pop=temp .NextGo=SemProp
    .NextConcat={Atr=temp!} .Del}
  .NextCmp={POS=NOUN}
 }
GModifier.Search={POS=ADJECTIVE}
```

Рис. 3. Миллипрограмма склейки наречия степени с прилагательным и прилагательного с существительным

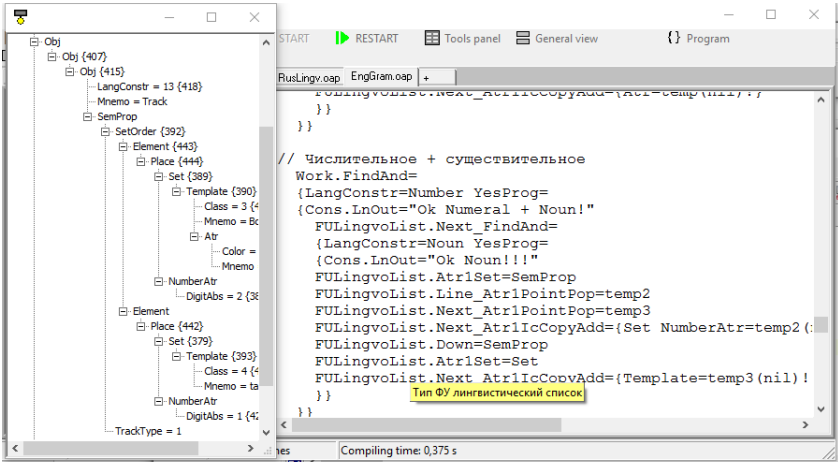

Рис. 4. Результат работы ФУ GraphModifier

Устройство GraphModifier уже реализовано программно и встроено в системы объектно-атрибутного программирования и моделирования. Результаты его работы можно видеть на рис. 4, где проиллюстрирован фрагмент ОА-графа, описывающего смысл текста.

Заключение

Результатом проделанной работы является формализм для описания динамического преобразования графа (ОА-грамматика),

а также практическая (программная) реализация преобразователя графа. Ближайшими аналогами ОА-грамматики являются HPSG, Regular tree grammar. Однако приведенные аналоги применяются исключительно для обработки языка, когда как ОА-грамматика может найти более широкое применение, т.к. она способна преобразовывать графы любой топологии.

Создание ФУ GraphModifier позволила реализовать преобразователь графа на ЭВМ. Следует отметить, что ФУ работает по принципам ОА-архитектуры, относящейся к классу dataflow (управление вычислениями с помощью потока данных) [5]. Dataflow принцип обеспечивает максимальное распараллеливание вычислений и их реализацию на распределенных вычислительных системах. ФУ позволяет производить обработку данных практически без опасности нарушить их целостность. Это обеспечивается благодаря единообразию построения ОА-графа, который состоит из ИП и ИК, объединенных между собой ссылками.

Список литературы

[1] Levine, Robert D.; W. Detmar Meurers Head-Driven Phrase Structure Grammar Linguistic Approach, Formal Foundations, and Computational Realization // To appear in Keith Brown (Ed.): Encyclopedia of Language and Linguistics (second edition). Ed. Keith Brown. Oxford: Elsevier. 2006, URL: http://www.sfs.uni-tuebingen.de/~dm/papers/ell2-hpsg.pdf

[2] Салибекян С.М., Панфилов П.Б. Объектно-атрибутная архитектура – новый подход к созданию объектных систем // Информационные технологии. 2012, №2 стр. 8-14

[3] Салибекян С.М., Халькина С.Б., Тиновицкий К.Д. Объектно-атрибутный подход для семантического анализа естественного языка. // Объектные системы - 2014: материал VIII Международной научно-практической конференции (Ростов-на-Дону, 10-12 мая 2014 г.) / Под общ. ред. П.П. Олейника. - Ростов-на-Дону: ШИ ЮРГТУ (НПИ), 2014. - С. 80-86 URL: http://objectsystems.ru/files/2012/Object_Systems_2014_Proceedings.pdf

[4] Салибекян С. М., Панфилов П. Б. Вопросы автоматно-сетевого моделирования вычислительных систем с управлением потоком данных // Информационные технологии и вычислительные системы. 2015. № 1. С. 3-9.

[5] Jurij Silk, Borut Robic and Theo Ungerer «Asynchrony in parallel computing: From dataflow to multithreading» Institut Jozef Stefan, Technical Report CDS-97-4, September 1997.

Об авторах:

Салибекян Сергей Михайлович

Национальный исследовательский университет «Высшая школа экономики», Московский институт электроники и математики, к.т.н., доцент.

e-mail: salibek@yandex.ru

Образец ссылки на публикацию:

(И. О. Фамилия!). (Название статьи!) // Программные системы: теория и приложения: электрон. научн. журн. 2013. Т. 4, № 3(17), с. ??–??.
URL: http://psta.psiras.ru/read/???

(Инициалы, фамилия автора латинскими буквами!). (Название статьи по-английски!).

Abstract. (Перевод аннотации на английский язык!).

Key Words and Phrases: (Перевод ключевых слов на английский язык!).