ПРИМЕНЕНИЕ ТЕХНИКИ УПРАВЛЕНИЯ СОБЫТИЯМИ ДЛЯ АНАЛИЗА ТЕКСТА В СИСТЕМЕ АБРИАЛЬ

А.И. Пацкин
РосНИИ Искусственного интеллекта, Москва
alpackin@mtu-net.ru

Введение

Каждому программисту, пробовавшему разрабатывать обработчики текста на естественном языке (**ЕЯ),** пользуясь "классическими" **императивными языками программирования**, вроде Си или Паскаля, известно, какая это неблагодарная задача. В императивной вычислительной модели текст обрабатывается знак за знаком, и функция, считавшая очередной знак из входного потока с помощью сложной конструкции ЕСЛИ/ТО/ИНАЧЕ должна принять решение, как с этим знаком поступить. Это в высшей степени неудобно, поскольку один и тот же знак, в разных контекстах может играть совершенно различные роли. Например, точка нужна для первичной структуризации - разбиения текста на предложения. Но та же точка может входить в состав многоточий, сокращений, дат, дробных чисел, электронных адресов, и т.д. и т.п.... (Кстати, предыдущее предложение кончается четырьмя точками). Очевидно, что решить в одном месте программы, что значит очередной знак, для реальных текстов - практически неподъемная задача. И это только пунктуация, а что уж говорить про сами слова, и их смысл - там те же проблемы вырастают тысячекратно.

Для преодоления этой трудности в 60-х годах была изобретена **продукционная модель вычислений**, в которой на текст (или на другие сложные структуры данных) накладываются **образцы** (шаблоны) и в случае согласования фрагмента текста (данных) с образцом запускается некоторое **правило**, в котором сосредоточены обрабатывающие **действия**. Правила имеют сравнительно простую и интуитивную структуру, и в значительной степени могут разрабатываться и изменяться независимо. Эти идеи были в разной степени воплощены в таких системах, как Рефал, Пролог, Плэннер, СНОБОЛ, OPS5 и др. Однако на практике, все системы с образцами при увеличении длины текста или числа правил быстро захлебывались в вычислениях из-за банальных комбинаторных проблем. Т.е. правила и образцы при всём своем внешнем удобстве, оказывались практически неприменимыми к ЕЯ-анализу из-за неэффективности.

Решение указанной дилеммы - между неудобством императивных вычислений и неэффективностью процедурных - искали, и ищут до сих пор. В частности в 70-е годы были предложены так называемые **RETE-алгоритмы** [3], в которых весь массив образцов в правилах предварительно транслировался в некоторый граф - дерево решений. Это дерево применялось затем к данным в качестве единой аналитической супер-программы. Другая идея состояла в **разбиении правил на группы**, и в применении единовременно только правил из одной группы (например, blackboard model). Эти и другие методы несколько снижали остроту проблемы, и позволяли решать задачи с десятками правил в каком-нибудь искусственном мирке. Но до работы с настоящими живыми текстами современным продукционным системам по-прежнему далеко, поскольку для реальных ЕЯ-текстов нужны массивы правил размером в сотни тысяч, или миллионы, т.к. эти количества должны быть примерно сопоставимы с размерами ЕЯ-словарей.

Учитывая вышесказанное, трудно переоценить важность нахождения **эффективного продукционного механизма,** позволяющего, например, так применить порядка полумиллиона правил к ЕЯ-тексту размером с "Войну и мир" Толстого, чтобы за несколько секунд прошла полная обработка этого текста на современном компьютере. Причем правила эти не должны быть достаточно просты, чтобы не быть плодом изощренной программистской работы, а скорее - продуктом рутинной деятельности большого числа специалистов в различных областях языка.

Настоящая работа представляет приложение к анализу ЕЯ-текстов продукционного механизма, входящего в состав ядра системы представления знаний "Абриаль" [1] [6]. Принцип работы этого механизма таков, что поставленная

цель - быстрой работы сотен тысяч правил на больших текстах - представляется вполне достижимой. Во всяком случае, принципиальных теоретических препятствий сейчас не видно, а практическая реализация уже достаточно далеко продвинута.

Общая постановка задачи анализа текста

Задача анализа текста (на естественном языке, в первую очередь) в общем виде может иметь два варианта: понимания текста, или извлечения смыслов.

- Задача **понимания текста** означает *полное* и *однозначное* сопоставление (обычно небольшого) фрагмента текста некоторой формальной структуре, описывающей его смысл. Это на практике проецируется на перевод или на диалог с пользователем.
- Задача **извлечения смыслов** ставит целью выборку из текста всех элементов понимания, полных и частичных, при этом допускается противоречивость элементов между собой. Эта задача направлена на обработку больших массивов текстов, в частности на поиск, фильтрацию, статистическую обработку.

В рамках данной работы нас интересуют оба варианта постановки задачи. При этом стоит отметить только, как специфика этих вариантов сказывается на природе реализующих алгоритмов.

Для первой задачи, где нас интересует ровно одно полное понимание фрагмента, (т.е. переводимой фразы, реплики пользователя, запроса, делового письма и т.д.), различные варианты прочтения многозначных элементов текста анализируются последовательно, и первое адекватное прочтение всего фрагмента прекращает анализ. При этом если частичное понимание фрагмента не складывается в общую структуру, оно само по себе бесполезно, и требуется удалить это частичное понимание, чтобы гарантировать его неучастие в анализе альтернативных вариантов прочтения. Таким образом, nolens volens, первая задача приводит к применению поиска с возвратами (бэктрекинга).

Второй вариант позволяет не использовать бэктрекинг, поскольку нас интересуют все смыслы, полные и частичные, извлекаемые из текста, и в том числе альтернативные прочтения одного и того же. Но и здесь нужно позаботиться о том, чтобы альтернативные варианты прочтения не смешивались на верхних уровнях анализа. Если не ограничиваться совсем простыми случаями, вообще говоря, это задача нетривиальная.

Структура сети объектов (базы данных). События.

База данных (она же **Сеть)** в системе "Абриаль" состоит из множества **объектов**, соединенных между собой многоместными **связями**, т.е. связями унарными, бинарными, тернарными и большей размерности. Одну связь можно записать в виде некоторого **факта** (в стиле языка пролог):

Ri(O1,O2,...,On)

Где **Ri** есть имя отношения, а **O1...On** - имена (или обозначения) конкретных объектов. Например: **Родитель(Иван,Василий)** или **Пол(Иван,Мужской)** - пример записи внешнего представления двух бинарных (двухместных) связей.

Обратим внимание, что унарные (одноместные) связи, например **Грозный(Иван),** связаны только с одним объектом, и по существу являются двоичными признаками объектов, а к понятию "связь" относятся лишь формально.

Отношения и соответственно связи, могут быть:

- ***реальными***, т.е. фактически присутствующими в базе данных;
- ***виртуальными***, т.е. рассчитываемыми динамически. Виртуальные отношения будем называть ***ассоциациями,*** и разделять на ***примитивные***, которые вычисляются ядром системы, т.е. виртуальной машиной, и ***сложные*** (составные), которые рассчитываются на основе других отношений и ассоциаций;
- ***временными***, т.е. такими же как реальные, но существующими только в пределах одной *транзакции* (см. ниже). Временные связи - это как раз и есть те самые ***события***, о которых мы здесь ведем речь.

Таким образом, что принципиально важно для понимания нашего механизма: **события это просто временные связи**, существующие только в пределах транзакции. **Транзакция** - это минимальная неделимая операция изменения базы данных, которая может окончиться либо удачей, и тогда множество согласованных изменений фиксируется, либо полной неудачей, и тогда происходит откат, т.е. полная отмена всех изменений. События не отличаются от обычных реальных связей ничем, кроме ограниченного времени жизни.

Представление текста в Базе Данных

Пусть текст **T** представляет линейную последовательность знаков с номерами от 1 до **N**. В Базе Данных всем знакам текста соответствует объекты класса **Знак**: **C1,C2,.... CN**. Все эти объекты связаны в цепочку двуместным отношением **Next** ("следует") которое распадается на N-1 связей:

Next(C1,C2) Next(C2,C3) ... Next(C...,CN)

Каждая такая связь, например **Next(C2,C3)** отражает тот факт, что за знаком **C2** следует знак **C3**.

Кроме того двуместным отношением **In** ("входит в") объекты **Ci** связаны со текстом:

In(T,C1) In(T,C2) .... In(T,CN)

Итак, изначально текст в Базе Данных представлен в виде примитивного дерева с корнем в **T** и с листьями - объектами **Ci**, причем листья дерева связаны в последовательную цепочку.

Общая схема процесса анализа

Процесс начинается с последовательной **активизации** всех объектов **Ci,** представляющих знаки.

Подробнее, всё начинается с некоторого стартового события **E0(T)**, аргументом которого является весь текст **T**, и смысл которого: "активизировать текст **T**". На событие **E0** реагирует некоторое правило **R0,** которое порождает множество событий **E1(C1),E2(C2),...EN(CN)**. Каждое из этих "вторичных" событий может также благодаря правилам, хранимым в БД порождать свои события-следствия. Таким образом, на данном уровне рассмотрения мы имеем дело с неким **деревом событий**. Корнем этого дерева является событие-первопричина **E0**. По мере срабатывания правил дерево событий растет из своего корня, и постепенно из событий-листьев этого дерева вырастают ветви последствий.

Чем заканчивается этот рост? Во-первых, могут появляться события без последствий. И дерево продолжает расти, пока листьями его не станут только события без последствий. На этом процесс анализа закончится. Общий результат ожидается трех видов:

- во-первых, результатом процесса будет некоторая новая построенная в базе данных **семантическая сеть S**, соответствующая анализируемому тексту; подробнее о строении **S** см. ниже.
- во-вторых, некоторые события могут порождать побочный **внешний эффект**, в виде, например, вывода на экран или в файлы.
- в-третьих, результатом всего процесса будет его **удачный/неудачный исход**, т.е. успех/неуспех события E0. Практически можно строить правила так, чтобы этот результат всегда был успешным, то есть, чтобы текст всегда распознавался хотя бы примитивно, если не удается распознавание на высоком уровне.

Итак, основным результатом распознавания текста должна стать некая подструктура **S** в базе данных, соответствующая семантической сети данного текста. Эта сеть, как и вся остальная база данных, состоит из объектов и связей. Новые объекты, связи и события порождаются правилами. Новые связи до конца транзакции имеют статус события. Потенциально они могут быть удалены при откате. Можно сказать, что связи - это те события, которые остаются в базе данных после окончания транзакции. Таким образом, в целом, процесс представляет собой транзакцию, начинающуюся со стартового события **E0,** которое порождает **N** событий - активизации каждого знака текста. Из них далее вырастает дерево событий; после окончания транзакции часть дерева события удаляется за ненадобностью, а часть - остается в базе данных в виде семантической сети, представляющей структуру распознанной в тексте информации.

Принцип управления событиями

В приведенной схеме процесса пока мало внимания уделялось самим правилам, хотя весь процесс продвигается вперед за счет срабатывания правил. Отметим коренное отличие данной системы от тех продукционных систем, где активизация идет от правил, и где весь массив правил или одно правило необходимо *применять* к тексту. В описываемой системе источниками активности являются не правила, а события. Поэтому нулевое время тратится на поиск правил, применимых к событию, или наоборот - событий соответствующих правилу. Каждое событие уже в момент своего рождения получает готовый список применимых к нему правил. Это обусловлено структурой памяти в Абриале, где "всё со всем связано".

Фактически события как бы "вызывают" правила, подобно тому, как в императивном языке программирования операторы вызывают соответствующие функции/процедуры. Работа механизма управления событиями, по сути,

сходна тому, как вела бы себя обычная программа на императивном языке, но с очень большим количеством сложных, вложенных конструкций IF/THEN/ELSE.

По существу вся хитрость данного механизма состоит в том, что он позволяет ясные и внешне прозрачные правила заставить работать как "классические" запутанные, но эффективные процедуры. Это удается сделать, потому что обратимые двусторонние связи в Абриале позволяют "выворачивать наизнанку" правила.

Активизация правил. Хиты.

В описанном механизме роста дерева событий был для простоты опущен ряд интересных моментов: каким образом события активизируют и запускают в работу правила. Ведь на самом деле не просто одно событие генерирует другое, но каждое событие может активизировать целый каскад правил, каждое из которых может сработать или не сработать, и только сработавшие правила генерируют события - следствия. Рассмотрим это подробнее.

Правило состоит из левой и правой частей:

- Левая часть, **образец,** состоит из нескольких **условий**.
- Правая, активная часть, состоит из одной или нескольких цепочек **действий**, эти цепочки мы будем называть **альтернативами**.

Соответственно исполнение правила четко делится на два этапа: на первом этапе проверяется образец, в случае удачного завершения первого этапа, происходит совпадение с образцом, то есть **хит правила**. И на втором этапе, т.е. после *хита* происходит исполнение правой части.

Если неудача происходит на первом этапе, т.е. до хита, то ничего не происходит, т.е. данное правило не срабатывает, и пропускается, как если бы его и вовсе не было. Если же неудача происходит после хита, т.е. на выполнении некоторой альтернативы, то сначала пробуется очередная альтернатива, но если её нет или все альтернативы вернули неудачу, то всё правило возвращает неудачу. При неудаче правила происходит откат (бэктрекинг) до предыдущего хита правила с альтернативами, имеющего неиспользованные альтернативы, или если такого хита не найдется, то до самого первого, стартового события, которое в этом случае возвращает неудачу.

При удачах процесс продвигается следующим образом:

1. события и новые связи активизируют условия в образцах правил;
2. когда активизируется некоторое (потенциально - любое) условие из образца правила, оно становится стартовым, и проверяются смежные условия (понятие смежности будет подробно рассмотрено ниже);
3. когда для некоторого стартового условия в образце, сначала смежные, а через них и все остальные условия данного образца к этому моменту (или изначально) выполнены, то происходит хит данного правила;
4. хит запускает поочередно альтернативы (цепочки действий) правой части правила;
5. каждое действие порождает событие или новую связь;
6. далее см. пункт 1.

Здесь ещё раз стоит обратить внимание на центральный отличительный момент для данного механизма:

Правила т.е их образцы не применяются к фрагментам данных целиком, напротив: образцы активизируются через свои условия и потенциально любое условие может оказаться стартовым. Какое условие окажется стартовым определяется динамически. Так же динамически определяется порядок обхода соседних условий. Существенно то, что соседние условия не проверяются на истинность, вместо этого просто обходятся все данные, на которых эти условия истинны.

Чтобы понять, как конкретно это происходит, нужно подробнее разобрать структуру правил.

Строение правила, образца, условий и действий. Переменные.

Мы знаем к настоящему моменту, что правило состоит из образца и одной или нескольких альтернатив, образец состоит из условий, а альтернативы - из действий. Рассмотрим, как устроены условия и действия.

Условия и действия устроены принципиально одинаковым образом: логически это те же самые связи, в записи которых хотя бы один конкретный объект заменен абстрактной переменной. Например **Родитель (Иван,Y)** В этом тексте мы будем для имен переменных использовать большие латинские буквы.

С каждым правилом ассоциирован некоторый набор переменных. Т.е. одно имя переменной, например **Y**, означает одну и ту же переменную в пределах одного правила (и для условий и для действий - один и тот же набор переменных).

Чтобы образец правила считался корректным, множество его условий на множестве переменных правила должно образовывать связный граф (это необходимо, но не достаточно для корректности). Некорректный образец никогда не образует хит, поэтому наличие правил с некорректными образцами может иметь последствием только бесполезную трату времени процессора.

Процесс сопоставления образца

Когда некоторое событие совпадает с одним из условий образца (в левой части правила), условие активизирует процесс сопоставления образца и становится в этом процессе стартующим условием. При этом переменные в этом условии конкретизируются значениями - в них попадают (ссылки на) соответствующие объекты. Те условия из образца, которые содержат внутри себя эти вновь конкретизированные переменные, становятся "смежными" т.е. их можно сопоставить со связями сопоставившихся объектов. Каждая такая сопоставившаяся связь конкретизирует объектными значениями другие переменные и так далее, до тех пор, пока все переменные, входящие в образец, не получат значения, и все условия образца не будут сопоставлены (выполнены) Так, в общих чертах происходит сопоставление образца: от активизации стартующего условия до хита.

В результате образец сопоставляется фрагменту сети (включая временную её часть, состоящую из событий), и переменные образца конкретизируются значениями нескольких взаимосвязанных объектов.

Обратим внимание на то, что **одной активизации правила** через стартовое условие обычно **соответствует несколько хитов**, т.е. некое множество фрагментов сети, соответствующих образцу данного правила. Точнее образуется **дерево хитов**, в котором стартующее условие становится корнем, хиты - листьями, а каждое очередное проверяемое условие добавляет "ярус" ветвей. Порядок проверки условий в случае их конкуренции выбирается динамически - отдельно для каждой активации правила. При этом используется общий принцип оптимизации времени обхода условий: приоритет имеют условия с меньшим ожидаемым числом проверок. Поэтому дерево хитов строится таким образом, что наиболее "кустистыми" оказываются последние ярусы. При этом отрицательные результаты проверок "обрубают" большие ветки дерева хитов как можно ближе "к стволу", т.е. на более ранних стадиях, так что на бесполезные проверки тратится минимум времени.

Вышеописанный метод сопоставления образцов является основной отличительной особенностью продукционного механизма системы Абриаль. Он может быть практически реализован, видимо, только на специфической модели памяти, в которой нет односторонних ссылок и все связи одинаково быстро проходимы в обе стороны.

Динамический принцип выбора порядка проверок условий, и по большому счету именно "управление событиями" отличает данный метод от наиболее продвинутых продукционных методик, в частности от так называемых RETE-алгоритмов, где ***статическое*** "дерево решений", строится по массиву правил один раз, заранее, перед началом расчета [3].

Несколько примеров правил для анализа текста

Данный набор правил предназначен для извлечения из русского текста всех фраз типа "мама мыла раму" или "толстые тети любят длинные макароны". Он незначительно упрощен для наглядности и несколько не полон по сравнению с рабочим вариантом, но основная идея методики в этом примере должна быть отражена. Правила представлены в формате:

***Имя-правила*:RULE(*условие1 условие 2... => действие1 действие2...*)**

Альтернативы, отрицания и бэктрекинг здесь не используются, т.к. нас интересуют все варианты анализа, полные и неполные. Русским текстом перед скобками записаны имена отношений, русские слова в скобках - имена конкретных объектов. Большими латинскими буквами в скобках обозначены имена переменных.

Правила, собирающие слова из знаков, не отражены. Грамматические формы слов определяются первым правилом. Оно занимает всего несколько строк, т.к. предполагает использование гиперсловаря, аналогичного представленному на Диалоге 2002 гиперсловарю "Ариадна" [7].

Грамматический_Анализ_Слова:RULE(

Разбиение_Слова(WORD,BASE,TAIL) Лексема_Основа_Профиль(LEXEME,BASE,PROFILE)

Профиль_Форма_Хвост(PROFILE,FORM,TAIL)

=> Генерация_Смысла(MEANING,WORD)

```
 Смысл_и_форма_слова(MEANING,LEXEME,FORM)
)

Существительное_есть_именная_группа:RULE(
 Смысл_и_форма_слова(MEAN,LEX,FRM) Часть_речи_лексемы(LEX,Существительное)
 => Именная_группа(MEAN)
)
Прилагательное_есть_атрибутная_группа:RULE(
 Смысл_и_форма_слова(MEAN,LEX,FRM) Часть_речи_лексемы(LEX,Прилагательное)
 => Атрибутная_группа(MEAN)
)

Переходные_глаголы:RULE(
 Смысл_и_форма_слова(MEAN,LEX,FRM) Граматический_признак_лексемы(LEX,Переходный_глагол)
 => Переходный_глагол(MEAN)
)
Личные_формы_глагола:RULE(
 Смысл_и_форма_слова(MEAN,LEX,FRM) Личная_форма(FRM) => Личная_форма_глагола(MEAN)
)

Именные_группы:RULE(
 Непосред_следует_за(M1,M2) Именная_группа(M2) Атрибутная_группа(M1)
 Род_число_падеж(M1,GNC) Род_число_падеж(M2,GNC)
 =>
 Генерация_над_смысла(M3,M2) Атрибуция(M3,M1) Именная_группа(M3)
)
Переходные_группы:RULE(
 Непосред_следует_за(M1,M2) Переходный_глагол(M1) Личная_форма_глагола(M1)
 Именная_группа(M2) Род_число_падеж(M2,GNC) Падеж_РЧП(GNC,Винительный)
 =>
 Генерация_над_смысла(M3,M1) Управление_глаголом(M3,M2) Переходная_группа(M3)
)
Описание_действия:RULE(
 Непосред_следует_за(M1,M2) Именная_группа(M1) Переходная_группа(M2)
 Род_число_падеж(M1,GNC) Падеж_РЧП(GNC,Именительный)
 =>
 Генерация_над_смысла(M3,M2) Субъект_действия(M3,M1) Действие(M3)
)
```

Состояние реализации

Система Абриаль реализована в качестве программы в среде Win32, и доступна на сайтах РОСНИИ или автора [1]. Представленный продукционный механизм является частью ядра системы Абриаль, законченной весной 2001-го года и с тех пор почти не развивавшейся. За прошедший период в 2001-м году были разработаны две лексикографические базы данных [7] в среде Абриаля: то есть гиперсловарь синонимов английского языка, по Тезаурусу Роже [2], и гиперсловарь (Ариадна) русского языка по материалам грамматического словаря А.А. Зализняка [4] и словаря морфем А.И. Кузнецовой [5]. В 2002-м году в рамках 2-й версии системы была создано средство построения приложений в виде гипертекстовых оболочек в веб-стиле вокруг баз данных.

В момент написания данного текста все эти составные части в основном готовы к интеграции в единый комплекс. На нем можно будет проверить достижимость поставленной цели, а именно: быстрой работы данного механизма на больших ЕЯ-текстах, и практическую независимость скорости обработки от количества правил.

Литература

1. http://www.artint.ru/packin/abrial/ http://packin.narod.ru/pro/
2. Roget's Thesaurus, Crowell company - 1911 (Electronic version by MICRA, Inc. - 1991).
3. Forgy C.L. RETE: A fast algorithm for many pattern/many object pattern match problem. Artif. Intell. 19, pp 17-37 (1982)
4. Зализняк А.А. , Грамматический словарь русского языка, М., Русский язык - 1977
5. Кузнецова А.И., Ефремова Т.Ф. Словарь морфем русского языка. М., Русский язык – 1986
6. Пацкин А.И. Программа ABRIAL - конструктор баз знаний в системе ИНФО-Т. Труды 7-й национально конференции по искусственному интеллекту КИИ-2000. Переславль-Залесский 2000.
7. Пацкин А.И. Гиперсловари на базе системы Абриаль. ДИАЛОГ'2002, Труды межд. семинара. М., 2002.