Язык правил для построения синтаксического дерева

К.К. Боярский, Е.А. Каневский

Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики,
Санкт-Петербургский экономико-математический институт РАН
Boyarin9@yandex.ru, Kanev@emi.nw.ru

Аннотация

Описан язык продукционных правил, применяемый для морфологического и синтаксического анализа предложений русского языка. Правила образуют двухуровневую иерархию, позволяющую вести просмотр текста вправо и влево от рабочей точки в пределах предложения.

Введение

Для решения многих задач, связанных с автоматическим анализом текста на естественном языке необходимо выполнить два предварительных условия: снять морфологическую и грамматическую неоднозначность и построить синтаксическое дерево, т. е. выявить связи между словами. Этим целям служит разрабатываемый в настоящее время семантико-синтаксический анализатор SemSin. После морфологического анализа каждому токену (слову, буквенно-цифровому комплексу, знаку препинания) сопоставляется максимально полная морфологическая, синтаксическая (актанты), и семантическая (класс по классификатору [1]) информация. Эта информация хранится в линейной структуре PipeLine в порядке, соответствующем порядку токенов в предложении. Затем цепочка токенов обрабатывается с помощью системы продукционных правил с целью снижения неоднозначности и перехода от линейной схемы связей к синтаксическому дереву. Система делает попытку применения каждого из правил последовательно ко всем токенам и при выполнении указанных в правиле условий совершает определенные действия.

К сожалению, информации о структуре языка для описания правил автоматического анализа русскоязычного текста, в современной литературе явно недостаточно. Некоторые сведения можно почерпнуть из [2–5]. Авторы учли также свой опыт создания морфолого-лексического анализатора [6]. Принятый за основу язык уточнялся в процессе разработки системы, сейчас написано более 200 правил. Опишем подробнее разработанный нами язык правил, который, по нашему мнению, достаточно адекватен как решаемой задаче, так и условию простоты отладки.

Труды XIV Всероссийской объединенной конференции «Интернет и современное общество» (IMS-2011), Санкт-Петербург, Россия, 2011.

Структура правил

Каждое правило состоит из нескольких частей.

Имя правила

Имя правила – уникальный идентификатор, позволяющий обращаться к данному правилу из других правил. Принято соглашение, что правила, имена которых начинаются с «SR» (SlaveRule) сами по себе не исполняются, они могут только быть вызваны из других правил (MasterRule). Если имя начинается с символов «RToL», то данное правило применяется к токенам справа налево. Остальные правила применяется к токенам в обычном порядке (слева направо).

Задание переменных

Каждая переменная принимает значения, соответствующие позиции токена в цепочке PipeLine, т. е. позицию точки относительно начала абзаца. Имена переменных начинаются с символа #. Первая переменная (в данной работе будем ее обозначать #W1) всегда указывает на обрабатываемый в данный момент токен, имеющий адрес gCurPos. В правилах типа Slave значение первой переменной задается в точке вызова. Для задания остальных переменных используются команды типа:

– #W2 = #W1 ± 1 – переменная #W2 указывает на токен, следующий или предшествующий #W1.
– #W2=gCurPos – #W2 ставится на позицию gCurPos. Используется в правилах типа Slave.
– #W2 = Next (#W1) или #W2 = Prev (#W1) – если #W1 находится внутри именной группы (ИГ), то #W2 ставится на первый элемент справа (слева), не принадлежащий этой ИГ; иначе #W2 = #W1 ± 1, причем знак препинания пропускается.
– #W2 = SegNext (#W1) или #W2 = SegPrev (#W1): #W2 – ставится на первый элемент за границей сегмента.

- #W2 = CentrSeg (#W1) – ставит переменную на центр текущего сегмента.
- #W2 = RightBorder(#W1) – ставит переменную на правую границу сегмента.
- #W2 = InLink (#W1) – записывает в #W2 адрес источника связи с приемником #W1.
- #W2 = OutLink (#W1,"Субъект") – записывает в #W2 адрес приемника связи с источником #W1 и именем «Субъект».

В связи с тем, что по мере срабатывания правил однородность структуры PipeLine нарушается за счет появления блоков (сегменты, именные и предложные группы и т. д.), нам пришлось ввести три типа переменных. Если имя переменной начинается с #W, то при попадании внутрь ИГ переменная сдвигается на центр ИГ. Например, если в процессе работы образовалась ИГ типа существительное–прилагательное и команда выставила переменную #W2 на прилагательное, она автоматически переместится на существительное. Во избежание зацикливания принято дополнительное соглашение: если имя правила заканчивается знаком «+», то просмотр токенов осуществляется вправо от исходной точки и переменная типа #W может сдвигаться только тоже вправо, если же имя правила заканчивается знаком «-» – то влево.

Переменные типа #Y игнорируют любые блоки и принимают именно то значение, которое указано в команде. Для всех остальных переменных ИГ игнорируются, но команды сдвига типа ± 1 не могут вывести переменную за границу сегмента (кроме команд SegNext, SegPrev).

Проверка условий

Условная часть правил строится по обычной для языков программирования схеме If…Then…ElseIf…Then…Else…EndIf. Внутри каждого блока могут использоваться операторы конъюнкции & и дизъюнкции OR. Разрешено использовать вложенные операторы If. Мы сознательно ограничились только одним уровнем вложения, поскольку иначе очень сложно производить отладку правил. Если первое условие не выполнено, остальные не проверяются. Блок OR всегда заключается в скобки. Если какое-то условие в блоке выполнено, остальные не проверяются.

Большая часть операторов имеет вид «ОПЕРАТОР (ПЕРЕМЕННАЯ) = ЗНАЧЕНИЕ». В зависимости от типа оператора параметр ЗНАЧЕНИЕ может быть целочисленный или строковый. Для численных значений сравнение может иметь вид типа =, !=, >, >=, <, <=; для строковых – только = и !=. Строковые условия могут иметь вид ОПЕРАТОР = "абв" или ОПЕРАТОР = "*абв", или ОПЕРАТОР = "абв*", или ОПЕРАТОР = "{абв,где,жзи}", где * заменяет произвольную группу символов.

Первая группа операторов определяет позицию токена в абзаце или предложении.

- CurPos (#W1) != #W2 – позиция точки, указанной в переменной #W1 относительно начала абзаца, не равна значению переменной #W2. Этот оператор полезен для исключения срабатывания правил при совпадении значений различных переменных из-за автоматической переадресации внутри ИГ.
- RelPos (#W1) > 2 – позиция точки, указанной в данной переменной относительно начала предложения, больше двух, т. е. это не первое слово предложения. Используется при разборе слов, написанных с прописной буквы.
- FinSentence (#W1) = 1 – конец предложения. Необходима при разбивке абзаца на предложения, поскольку не каждая точка – граница предложения.

Вторая группа операторов проверяет тип токена:

- Punct (#W1) = 1 – знак пунктуации;
- Cipher (#W1) = 1 – здесь цифра;
- Registr (#W1) = 1 – первая буква прописная (…= 2 – все буквы прописные, используется при анализе аббревиатур);
- Len (#W1) <= 2 – длина словоформы не превосходит 2 (используется при разборе инициалов).

Следующая группа анализирует морфологические характеристики слова. Поскольку одному токену могут соответствовать несколько разных лексем, а каждой лексеме – различные грамматические параметры (падеж, число, время и др.), операторы, названия которых начинаются с префикса «Is» возвращают значение TRUE, если условие выполняется хотя бы для одной лексемы. К таким операторам относятся, например, следующие:

- IsPos (#W1) = "СУЩ" – хоть одна лексема имеет такую часть речи,
- IsCase (#W1) = "Род." – хоть у одной лексемы имеется такой падеж,
- IsCaseSg (#W1) = "Род." – хоть у одной лексемы имеется такой падеж ед. числа,
- IsGender (#W1) = "Муж." – хоть одна лексема имеет такой род,
- IsLemma (#W1) = "НАМЕРЕН" – хоть одна лемма имеет такое значение, и т. д.

К этой же группе можно отнести операторы проверки словоформы sWord (#W1) = "." и проверки числа различных лексем для данного токена iLex (#W1) = 2. Следует также упомянуть оператор OnlyPos, который, в отличие от оператора IsPos, проверяет, что все лексемы данного токена относятся к одной и той же части речи. Это важно, например, если словоформа может представлять глагол совершенного/ несовершенного вида или одушевленное/ неодушевленное существительное.

- Include ({Abbr2},#W1) = 1 – проверяет наличие леммы токена #W1 в именованном наборе Abbr2.

Такие наборы слов хранятся в отдельном файле. В них включаются, прежде всего, сокращения разных типов. Например, сокращения тов. (*товарищ*) или им. (*имени*) не могут стоять в конце предложения, сокращения рис. (*рисунок*) или д. (*дом*) предполагают последующие цифры и т. д. Отдельный на-

бор включает всевозможные титулы (*величество*, *высокопреосвященство*, *светлость*), которые согласуются с последующими словами по особым правилам. Имеется также набор подчинительных союзных слов типа *где*, *где-нибудь*, *какой*, *который*, *кто* и др.

Четвертая группа условий с именами, начинающимися с префикса IsConcord, проверяет полное согласование слов по роду, числу и падежу (IsConcordGNC(#W1,#W2)) или частичное согласование (IsConcordNC, IsConcordGN, IsConcordC). В частности, проверяется согласование прилагательных (причастий и порядковых числительных) с существительными и с прилагательными (причастиями и порядковыми числительными). Согласование подлежащего со сказуемым проверяют операторы IsConcordSP и IsConcordSP1, последний разрешает глаголу быть во множественном числе, а существительному – в единственном. Особо следует упомянуть оператор IsConcordArgCase(#W1,#W2), который проверяет согласование аргументов слова из #W1 с падежами и классами слова из #W2

Для подавляющего большинства лексем в используемом нами словаре указаны актанты (падежи или наименования семантических связей, а иногда и допустимые классы) тех слов, которые могут присоединяться к этим лексемам. Например, глагол ОБЛИВАТЬСЯ имеет такие актанты: !Им (*мальчик*), !Тв (*водой*) и !Откуда (*из шланга*), причем к актанту !Тв. могут подсоединяться только слова, которые имеют классы «Жизнь/Выделения» или «Физ.Объекты/Неодуш./Вещества/Жидкости». Поскольку все слова в словаре имеют свои классы, то это позволяет в целом ряде случаев осуществлять однозначный выбор присоединяемых слов, как это будет показано ниже.

Наконец последняя группа анализирует фрагменты уже созданного синтаксического дерева:

– Link(#W1) < 0 – на данную точку уже установлена какая-то связь слева (> 0 справа); Link(#W1) = 0 проверяет отсутствие связи вообще.
– InLinkName (#W1) = "Какой" – проверка типа входной связи.
– IsCentrIG (#W2) != "Название" – не центр ИГ, имеющей тип «Название».
– TypeSeg (#W1) = "SEG_PRIDAT" – проверка типа сегмента.

Всего на данный момент в условной части правил использовано 48 типов операторов проверки.

Исполнительная часть правил

Если все проверявшиеся условия удовлетворены, начинают выполняться команды исполнительной части. Это, во-первых, операторы, уменьшающие неоднозначность разбора: KeepLex и RemoveLex. Первый из них оставляет, а второй удаляет морфологические варианты и целые лексемы, удовлетворяющие определенным условиям. Например, команда KeepLex(Case (#W1) = "Им.") оставляет только ту лексему, которая содержит Им. падеж единственного или множественного числа, причем все остальные падежи в вариантах разбора удаляются. Команда KeepLex (Argum(#W1) = "$105/1,!Тв") оставляет только ту лексему, которая в списке своих аргументов содержит творительный падеж с классом $105/1 «Жизнь/Выделения». Команда же RemoveLex (Pos(#W1) = "СУЩ") удаляет те лексемы, которые являются существительными.

Команды с префиксом Concord действуют аналогично соответствующим операторам в условиях правил с той разницей, что они оставляют только согласованные варианты токенов.

Несколько команд служат для сегментации предложения. К ним относятся:

– SelectSeg (#W1,"SEG_GER") – выделяет сегмент типа SEG_GER (деепричастный оборот) с центром #W1 с границами по ближайшим Punct = 1.
– SetSegCentr (#W1) – для всех точек выделенного ранее сегмента устанавливает центр в позиции #W1. Центром сегмента является сказуемое (глагол, краткое прилагательное или причастие, предикат) в главных или придаточных предложениях, деепричастие или причастие в соответствующих оборотах.
– JoinSeg (#W1,#W2) – объединяет сегменты, содержащие указанные элементы. Объединение сегментов производится по правила близким к описанным в [7].

Команда, выделяющая ИГ и устанавливающая связи между словами:

– SetImGr (#W2,#W1,"ИмяСвязи","ИмяГруппы") – установить связь и ИГ для слова #W2 с центром #W1. Если "ИмяГруппы" = "", то только ставится связь.

При исполнении этой команды проверяется, что #W2 и #W1 не были связаны между собой ранее (хотя бы косвенно, через другие слова). Это позволяет избежать возникновения циклов.

– Команда CallRule(A,#W1,"Имя_Правила") вызывает правило с указанным именем с позиции #W1.

Параметр «A» может принимать значения U,N,Y. Правила с U запускаются всегда, с N, если предыдущее правило SlaveRule не сработало (проверочная часть дала отрицательный результат), с Y, если предыдущее правило сработало.

– Пустой оператор Break() служит для прерывания цепочки рекуррентных вызовов.

Всего в исполнительной части использовано 45 команд.

Примеры

Для примера приведем группу правил, использующуюся для дифференциации существительного и наречия по словоформе «ПОТОМ». Двойной слэш здесь отделяет комментарии.

Правила

<ПОТОМ>
// имя правила для дифференциации п'отом - пот'ом

```
    #W1 = #W1; #W2 = #W1-1; #W3 = #W1+1;
    If sWord(#W1) = "потом" Then
// условие запуска вспомогательных правил
      CallRule(U,#W2,"SR:ПОТОМ1a-");
// поиск влево, выполняется всегда
      CallRule(N,#W2,"SR:ПОТОМ1-");
// выполняется, если предыдущее правило не сработало
      CallRule(N,#W3,"SR:ПОТОМ2+");
// поиск вправо, если предыдущее правило не сработало
      CallRule(N,#W1,"SR:ПОТОМ3");
// только если поиски влево и вправо не дали результата
    EndIf
  </>

  <SR:ПОТОМ1a-> // служебное правило, слева предлог – оставляем существительное
    #W1=#W1; #W2=gCurPos;
    If sWord(#W1)="с" Then
      KeepLex(Pos(#W2)="СУЩ");
    ElseIf (Punct(#W1) = 1 OR IsPos(#W1)="ГЛАГ" OR IsPos(#W1)="ДЕЕПР" OR IsPos(#W1)="ИНФ") Then
      Break();
// знак препинания или глагольная форма, прерываем исполнение – правило не выполнилось
    Else
      CallRule(U,#W1-1,"SR:ПОТОМ1a-");
// рекуррентный вызов со смещением влево
    EndIf
  </>

  <SR:ПОТОМ1-> // слева подходящий глагол – оставляем сущ., связываем с глаголом
    #W1=#W1; #W2=#W1-1; #W3=gCurPos;
    If (IsPos(#W1)="ГЛАГ" OR IsPos(#W1)="ПРИЧ" OR IsPos(#W1)="ДЕЕПР" OR IsPos(#W1)="ИНФ") & IsArgum(#W1)="$105/1,!Тв" Then
      KeepLex(Pos(#W3)="СУЩ"); // оставляем сущ.
      KeepLex(Argum(#W1)="$105/1,!Тв");
// оставляем только тот глагол, который удовлетворяет условию
      SetImGr(#W3,#W1,"Тв","");
// устанавливаем связь
    Else
      CallRule(U,#W2,"SR:ПОТОМ1-");
// рекуррентный вызов со смещением влево
    EndIf
  </>

  <SR:ПОТОМ2+> // то же, что в предыдущем правиле, но подходящий глагол ищем справа
    #W1=#W1; #W2=#W1+1; #W3=gCurPos;
    If IsPos(#W1)="ГЛАГ" & IsArgum(#W1) = "$105/1,!Тв" Then
      KeepLex(Pos(#W3)="СУЩ");
      KeepLex(Argum(#W1)="$105/1,!Тв");
      SetImGr(#W3,#W1,"Тв","");
    Else
      CallRule(U,#W2,"SR:ПОТОМ2+");
    EndIf
  </>

  <SR:ПОТОМ3>
// нет глагола или предлога – оставляем наречие
    #W1=#W1;
    If sWord(#W1) = "потом" Then
      RemoveLex(Pos(#W1)="СУЩ";
    EndIf
  </>
```

Применение правил

Рассмотрим ряд предложений, взятых из НКРЯ и содержащих слово *потом*.

1. *Кусок земного металла, жаркий слиток земных надежд, продукция мозга и мышц, смешанная с нашим потом и с кровью тех, которые этого уже не услышат.*

Правило <ПОТОМ> обнаруживает соответствующую словоформу, вызывает подчиненное правило <SR:ПОТОМ1a->, которое рекуррентным образом находит слева предлог и принимает решение оставить существительное *пот*.

2. *Зачем мы обливаемся потом и падаем на каждом шагу от усталости.*

Правило <ПОТОМ> находит словоформу *потом*, вызывает подчиненное правило <SR:ПОТОМ1a->, которое предлога не обнаруживает. Затем главное правило вызывает подчиненное правило <SR:ПОТОМ1->, которое находит слева глагол *обливаться*, способный подсоединить существительное нужного класса. Оставляет существительное *пот*.

3. *Милия Алексеевича едва потом не прошибло.*

Правила <SR:ПОТОМ1a-> и <SR:ПОТОМ1-> не выполняются, так что главное правило вызывает подчиненное правило <SR:ПОТОМ2+>. Это правило находит справа подходящий глагол (*прошибить*). Оставляет существительное *пот*.

4. *Сдадите ли потом квартиру или просто комнату*.

Правила <SR:ПОТОМ1a->,<SR:ПОТОМ1-> и <SR:ПОТОМ2+> не выполняются (глагол *сдавать* не имеет подходящего актанта), поэтому вызывается правило <SR:ПОТОМ3>, которое оставляет наречие *потом*.

Заключение

Команды, использованные в нашей системе правил, можно разделить на три большие группы.

К первой группе относятся команды, с помощью которых определяются границы предложений, проводится сегментация и анализ аббревиатур. Здесь большую роль играют такие параметры, как расположение токена в предложении, длина токена, регистр и другие графематические характеристики.

Команды второй группы используют обычную морфологическую информацию: часть речи, падеж и т. д. Эти команды обеспечивают связь между прила-

гательными и существительными, снятие омонимии типа наречие – краткое прилагательное и др.

Команды третьей группы используют синтаксическую (актанты) и семантическую (классы) информацию. Эти команды обеспечивают подключение существительных к хозяину, связки предложных групп и их подключение к хозяину.

Литература

[1] Тузов, В.А. Компьютерная семантика русского языка. СПб.: Изд-во С.-Петерб. ун-та, 2004.

[2] Сокирко, А.В. Семантические словари в автоматической обработке текста (по материалам системы ДИАЛИНГ) // Диссертация на соискание ученой степени кандидата технических наук. – М. 2001.

[3] Ножов, И.М. Морфологическая и синтаксическая обработка текста (модели и программы) // Диссертация на соискание ученой степени кандидата технических наук. – М. 2003.

[4] Урюпина, О. Автоматическое разбиение текста на предложения для русского языка // Компьютерная лингвистика и интеллектуальные технологии: По материалам ежегодной Международной конференции «Диалог» (Бекасово, 4–8 июня 2008 г.). Вып. 7 (14). – М.: РГГУ, 2008, с. 539–544.

[5] Боярский, К.К. О предварительном преобразовании синтаксического дерева предложения / Боярский К.К., Каневский Е.А., Лезин Г.В. // Компьютерная лингвистика и развитие семантического поиска в Интернете: Труды научного семинара XIII Всероссийской объединенной конференции «Интернет и современное общество». – СПб.: ООО «МультиПроджектСистем-Сервис», 2010, с. 3–8.

[6] Каневский, Е.А. Морфолого-лексический анализатор и классификация текста / Каневский Е.А., Боярский К.К. // Прикладная лингвистика в науке и образовании. Материалы V международной научно-практической конференции 25–26 марта 2010. – СПб.: «Лема», 2010, с. 157–163.

[7] Кобзарева, Т.Ю. Принципы сегментационного анализа русского предложения // Московский лингвистический журнал, 2004. т. 8, №1, с. 31–80.

Rule language for construction of a syntactic tree

K. Boyarsky, E. Kanevsky

The production rules language, applied for morphological and syntactical parsing of Russian sentences is described. Rules form the two-level hierarchy, allowing to conduct text viewing to the right and to the left from a working point within the sentence.