Трансляция автоматной грамматики (формата SRGS) в решётку слов (формат SLF)

Матвеев И.А.,
кандидат физико-математических наук

Рассмотрены два известных способа задания последовательности произносимых слов входного языка в системе распознавания речи: описание возможной структуры предложений и их частей при помощи автоматных грамматик и непосредственное задание графа состояний и переходов между ними, соответствующих произнесению определённых слов. Предложен алгоритм трансляции представления первого вида (удобного для человека) в представление второго вида (непосредственно применяемое в автоматических системах распознавания, основанных на марковских моделях). Алгоритм состоит из двух основных шагов: разложение входной грамматики до набора элементарных операций («и», «исключающее или», «опция») и построения графа состояний путём последовательного пополнения тривиального начального графа подграфами элементарных операций. Предложен метод сохранения семантической информации в графе состояний.

Введение

Система распознавания речи может быть представлена как агент, имеющий следующие входы и выходы:

- Вход 1: звуковой поток.
- Вход 2: набор акустико-фонетических моделей (например, скрытых марковских моделей звуков).
- Вход 3: грамматика, определяющая слова и последовательности слов (фразы), которые могут возникнуть во входном звуковом потоке и должны быть распознаны.
- Выход: текст, состоящий из языковых единиц, удовлетворяющий грамматике.

Основная цель применения грамматики при распознавании речи — описание тех слов (и фраз), которые распознающий агент может регистрировать в каждый момент времени. Грамматику для распознавания речи (проблемно-ориентированного языка, используемого в какой-либо специфической деятельности) удобно задавать в виде шаблонов предложений, описывающих порядок следования слов (членов предложения) во фразе и возможные варианты этих слов. Один из вариантов задания такого рода грамматики — так называемая грамматика для распознавания речи (Speech Recognition Grammar Specification, SRGS) [1], которая основана на расширенной форме Бэкуса-Наура (Augmented Backus–Naur Form, ABNF) [2, 3]. В этой форме фразы представляются в виде наборов графов — деревьев И/ИЛИ, одно из которых является начальным (корневым), а остальные представляют собой последовательное раскрытие узлов этого дерева вплоть до терминальных узлов (листьев или *токенов* в терминологии SRGS), соответствующих самим произносимым словам. Такое представление удобно для задания грамматики языка разработчиком системы распознавания, однако неудобно для непосредственной работы распознающего агента, представляющего собой модель марковского процесса, который в каждый момент времени должен решать, в какое состояние ему следует перейти, т.е. в данном случае — какое из слов языка сейчас может быть произнесено. Для обеспечения работы такого распознающего агента лучше всего построить граф состояний марковского процесса. Этот граф может быть задан в формате стандартной решётки слов (Standard Lattice Format, SLF) [4].

Таким образом, возникает необходимость построения транслятора из грамматики, заданной в формате SRGS (удобном для человека), в формат SLF (удобный для непосредственной работы марковской модели).

Входной формат (SRGS/ABNF)

Используется подмножество SRGS [1], которое определяется следующим образом: грамматика описывается в единственном файле и применяются только локальные правила, явно определённые в данном файле. Веса и идентификаторы языка не используются. Предполагается отсутствие циклов.

Поддерживаемые разделители:

- доллар ('$') — обозначает ссылки на правила;
- двойные кавычки — выделение токена из нескольких слов;
- скобки (' (' и ')') — определение порядка расширения правил;
- вертикальная черта ('|') — разделитель альтернатив;
- квадратные скобки (' [' и']') — выделяют необязательное правило (далее этот оператор называется «опция»);

- фигурные скобки ('{' и'}') — выделяют тэги;
- двойная косая ('//') — комментарий.

Специальные символы — правила $NULL, $VOID, $GARBAGE.

Пример грамматики, заданной в указанном подмножестве языка SRGS:

```
   root $basicCmd;
   $basicCmd =             $startPolite {start}
                           $command {first command}
                           [$endPolite {end}];   // rule1
   $command {command} =    $action {action}
                           $object {object};   // rule2
   $object =               [the | a] (window | file | menu) ;
//rule3
   $action =               open |   // rule4
                           close |
                                  delete {!!}|
                                   move ;
   $startPolite =          please |   // rule 5
                           "could you" |
                                  "oh   mighty   computer"   {so
long?};
   $endPolite =            please | "thank you"; // rule 6
```

Эта грамматика содержит шесть правил. Оператор «root» выделяет корневое правило грамматики.

Представление грамматики SRGS в виде дерева

Достаточно удобно для иллюстрации рассуждений представление правил (и грамматики в целом) в виде деревьев «И/ИЛИ», а более точно в виде деревьев «И/ИСКЛЮЧАЮЩЕЕ ИЛИ». Первое и последнее правила приведённого примера выглядят так:

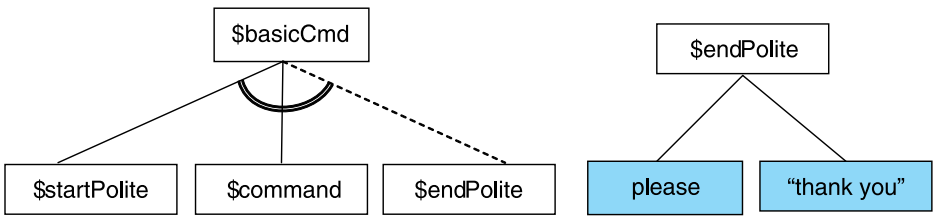

Рис. 1. Деревья «И/ИСКЛЮЧАЮЩЕЕ ИЛИ» первого и последнего правил примера

Верхний элемент правила — родительский узел, нижние — узлы-потомки. Двойная дуга, соединяющая потомков, обозначает операцию «И», т. е. в грамматике необходимы все соединённые ей элементы, причём в порядке слева направо. Потомки, не соединённые двойной дугой, — операция «ИСКЛЮЧАЮЩЕЕ ИЛИ», т. е. в грамматике необходим один и только один из потомков. Штриховая линия обозначает необязательный элемент (оператор «опция»). Элементы с именами, начинающимися со знака $ — правила, требующие дальнейшего раскрытия, элементы с именами без знака $ (выделяемые здесь и далее тёмным фоном) — токены, представляющие собой непосредственно произносимые слова. Таким образом, первое правило может давать два варианта предложений грамматики: «$startPolite $command $endPolite» и «$startPolite $command», а последнее правило задаёт два возможных варианта произнесения правила $endPolite: «please» и «thank you». Раскрыв в первом правиле элемент $endPolite при помощи последнего правила, получим три варианта предложений, соответствующих грамматике: «$startPolite $command», «$startPolite $command please» и «$startPolite $command “thank you”». Представление в виде дерева:

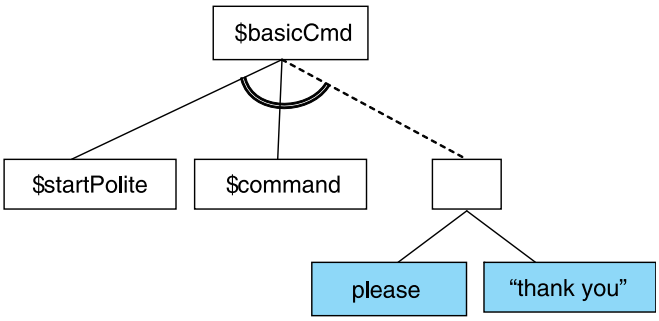

Рис. 2. Объединение первого и последнего правил

Элементы $startPolite и $command, в свою очередь, также должны быть раскрыты до токенов (терминальных элементов).

Можно заметить, что после объединения правил больше нет необходимости помнить название правила $endPolite. Это символическое имя необходимо лишь для связи правил. Продолжив раскрытие правил согласно грамматике примера (и опуская при этом имена правил), можно построить *терминальное дерево*, в котором все правила раскрыты до токенов (**рис. 3**). Грамматике удовлетворяют предложения: «Please open window» или «Oh mighty computer delete the file thank you» и т. п. Ошибки в задании грамматики, выявляемые при построении терминального дерева: противоречивость, неполнота и наличие циклов. Противоречивость — присутствие нескольких правил с одинаковым именем. Неполнота — невозможность раскрыть очередной нетерминальный элемент грамматики (отсутствие правила с таким именем). Наличие циклов — присутствие циклических зависимостей (правило A_1 расширяется через правило A_2, правило A_{n-1} расширяется через правило A_n, правило A_n расширяется через правило A_1). Все эти ошибки приводят к невозможности построить конечное терминальное дерево, а значит, и конечную сеть состояний марковской модели распознающей системы.

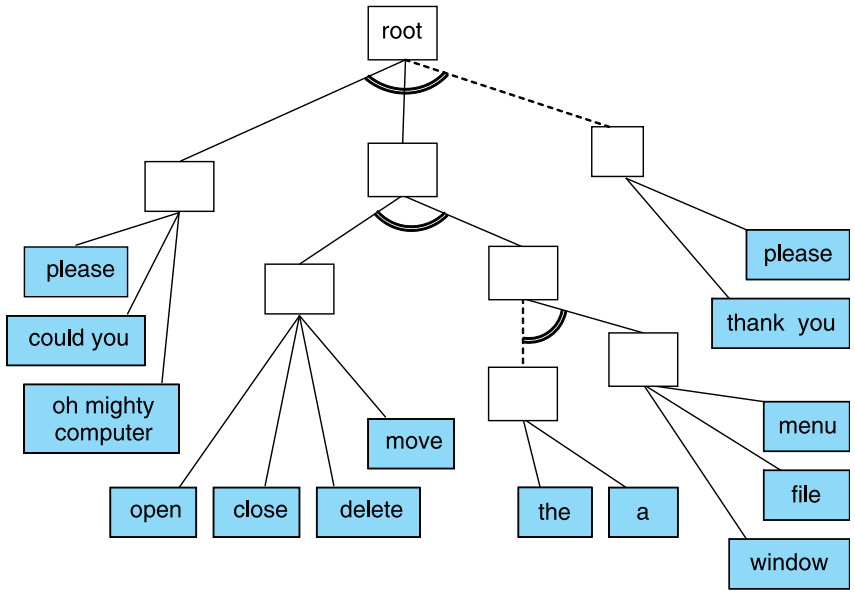

Рис. 3. Терминальное дерево «И/ИСКЛЮЧАЮЩЕЕ ИЛИ»

Выходной формат

Формат SLF представляет собой описание дерева состояний марковского процесса. Вершинами дерева являются состояния, к рёбрам приписаны токены (терминальные символы). Некоторые рёбра могут не иметь приписанных токенов, что эквивалентно приписыванию специального токена $NULL. Дерево имеет определённые числа вершин и дуг, которые указываются в первой строке выходного файла. Далее идёт перечисление дуг с указанием последовательно номера дуги, номера начальной вершины, номера конечной вершины и приписанного токена.

Пример, приведённый выше, транслируется в следующий текст формата SLF:

```
N=6       L=16
J=0        S=2        E=1        W=$NULL
J=1        S=2        E=1        W=please
J=2        S=2        E=1        W="thank you"
J=3        S=0        E=3        W="oh mighty computer"
J=4        S=0        E=3        W=please
J=5        S=0        E=3        W="could you"
J=6        S=3        E=4        W=move
J=7        S=3        E=4        W=delete
J=8        S=4        E=5        W=$NULL
J=9        S=4        E=5        W=the
J=10       S=4        E=5        W=a
J=11       S=5        E=2        W=menu
J=12       S=3        E=4        W=open
J=13       S=3        E=4        W=close
J=14       S=5        E=2        W=window
J=15       S=5        E=2        W=file
```

Число вершин равно 6 (нумеруются от нуля), число рёбер равно 16. Графическое представление описанного дерева:

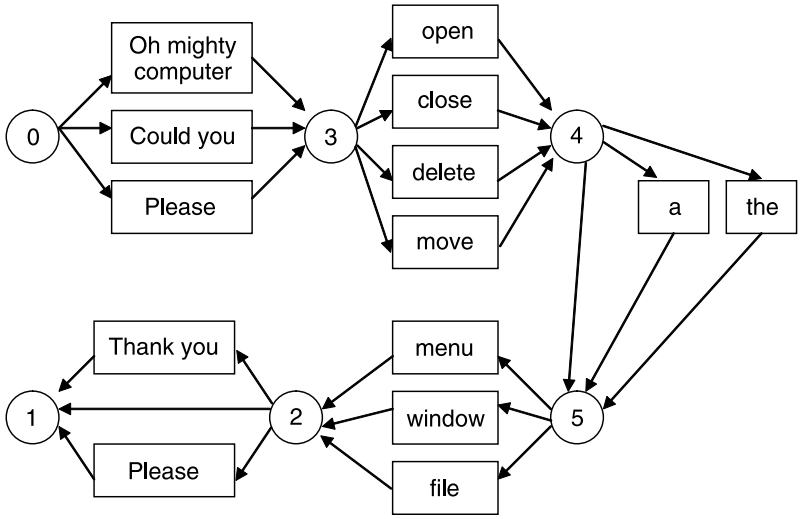

Рис. 4. Графическое представление дерева грамматики

Алгоритм трансляции

Алгоритм состоит из двух основных шагов: разложение входной грамматики до набора элементарных операций и построения графа переходов из тривиального начального графа последовательным пополнением соответствующими подграфами элементарных операций.

Трансляция разбита на несколько этапов:

- преобразование текста формата SRGS в обратную польскую запись;
- преобразование польской записи в набор элементарных (бинарных) операций;
- проверка существования терминального дерева грамматики;
- трансляция набора бинарных операций в марковскую сеть, представленную в формате SLF.

Преобразование в обратную польскую запись

Преобразование в постфиксную нотацию обратной польской записи (Postfix Reverse Polish Notation, RPN) [5] выполняется алгоритмом Дийкстры [6]. На этом шаге производятся следующие действия:

- удаляются комментарии, переносы строк, табуляция и иные элементы текста, не несущие синтаксической нагрузки. Также возможно исключение тэгов. Выходная запись форматируется (элементы разделяются ровно одним пробелом, правила разделяются ровно одним переносом строк и т.п.), что облегчает последующую обработку;
- из записи исключаются круглые и квадратные скобки (…) и […], задающие порядок операций;
- осуществляется лексический и синтаксический контроль.

В польской записи добавляются два символа операций: & — как обозначение бинарного оператора «И» (который в исходном формате SLF применялся по умолчанию при последовательном перечислении токенов) и ~ как обозначение унарного оператора «опция» (в исходном формате SLF этот оператор обозначался квадратными скобками). Приведённый выше пример грамматики SLF транслируется в следующую польскую запись:

```
$basicCmd = $startPolite $command & $endPolite ~ &
$object = the a | ~ window file | menu | &
$command = $action $object &
$action = open close | delete | move |
$startPolite = please "could you" | "oh mighty computer" |
$endPolite = please "thank you" |
```

Преобразование в набор бинарных операций

Форматированная польская запись позволяет простым образом прочитать каждое отдельное правило грамматики как набор бинарных операций. Описание каждой такой бинарной операции содержит родительский узел, два узла-потомка, вид операции, а также флаги необязательной операции, которые могут быть приписаны и к родительскому узлу, и к потомкам. Операции являются бинарными, но деревья «И/ИСКЛЮЧАЮЩЕЕ ИЛИ» на **рис. 1–3** — нет (т.е. у узлов на таких деревьях может быть более одного потомка). Дерево, соответствующее, например, первому правилу грамматики, и содержащее только вершины с двумя потомками, можно нарисовать в двух эквивалентных формах:

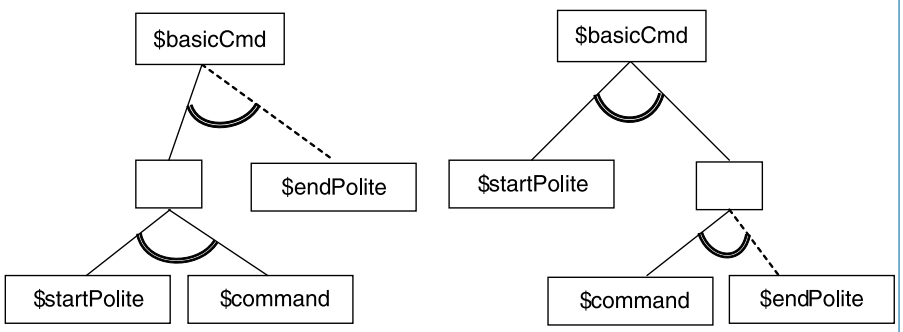

Рис. 5. Формы первого правила грамматики из примера

Приведённая выше польская запись первого правила соответствует левому дереву на **рис. 5**. Какая из форм будет выбрана, зависит от настроек алгоритма Дийкстры. Одним из узлов-потомков корневого узла является безымянный узел, не соответствующий непосредственно никакому правилу или токену. Такие узлы получаются при трансляции правила, содержащего более одной бинарной операции. Как следует из построения, безымянные узлы не могут быть терминальными узлами дерева правила. Поскольку безымянных узлов может быть несколько, им (а также всем нетерминальным узлам) присваиваются уникальные номера. После такой обработки любое составное правило

может быть разложено на *тривиальные деревья*, каждое из которых представляет собой бинарную операцию и состоит из вершины и двух листьев. Например, первое правило грамматики представляется в виде двух тривиальных деревьев:

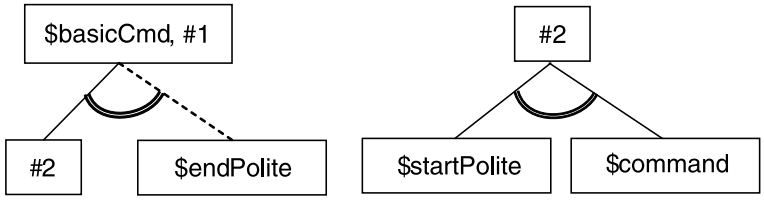

Рис. 6. Представление первого правила тривиальными деревьями

Родительским узлом тривиального дерева может быть корневой узел всей грамматики (выделенный оператором 'root', в рассматриваемом примере это узел $basicCmd), корневой узел какого-либо правила ($object, $command, и т. д.) или безымянный узел. Узел-потомок может быть безымянным, содержать имя какого-либо правила (за исключением корневого) или токен, т. е. терминальный элемент (лист) дерева грамматики.

Набор тривиальных деревьев (соответственно, бинарных операций), получаемых для грамматики примера, дан в таблице:

Номер	Имя	Операция	Первый лист	Второй лист
1	$basicCmd (root)	&	#2	~ $endPolite
2		&	$startPolite	$command
3	$object	&	~ #4	#5
4		|	**the**	**a**
5		|	#6	**menu**
6		|	file	**window**
7	$command	&	$action	$object
8	$action	|	#9	**move**
9		|	#10	**delete**
10		|	**open**	**close**
11	$startPolite	|	#12	**oh mighty computer**
12		|	**could you**	**please**
13	$endPolite	|	**please**	**thank you**

Трансляция в марковскую сеть в формате SLF

Марковская сеть для системы распознавания представляет собой ориентированный граф без циклов с единственной начальной и единственной терминальной вершинами. К каждому ребру графа приписан один и только один токен (соответствующий определённому

слову или словосочетанию языка) или специальный символ ($NULL, $VOID, $GARBAGE). Если для грамматики существует терминальное дерево, то существует и конечная марковская сеть. Эта сеть строится из набора бинарных операций следующим алгоритмом, последовательно раскрывающим элементы, начиная с корневого:

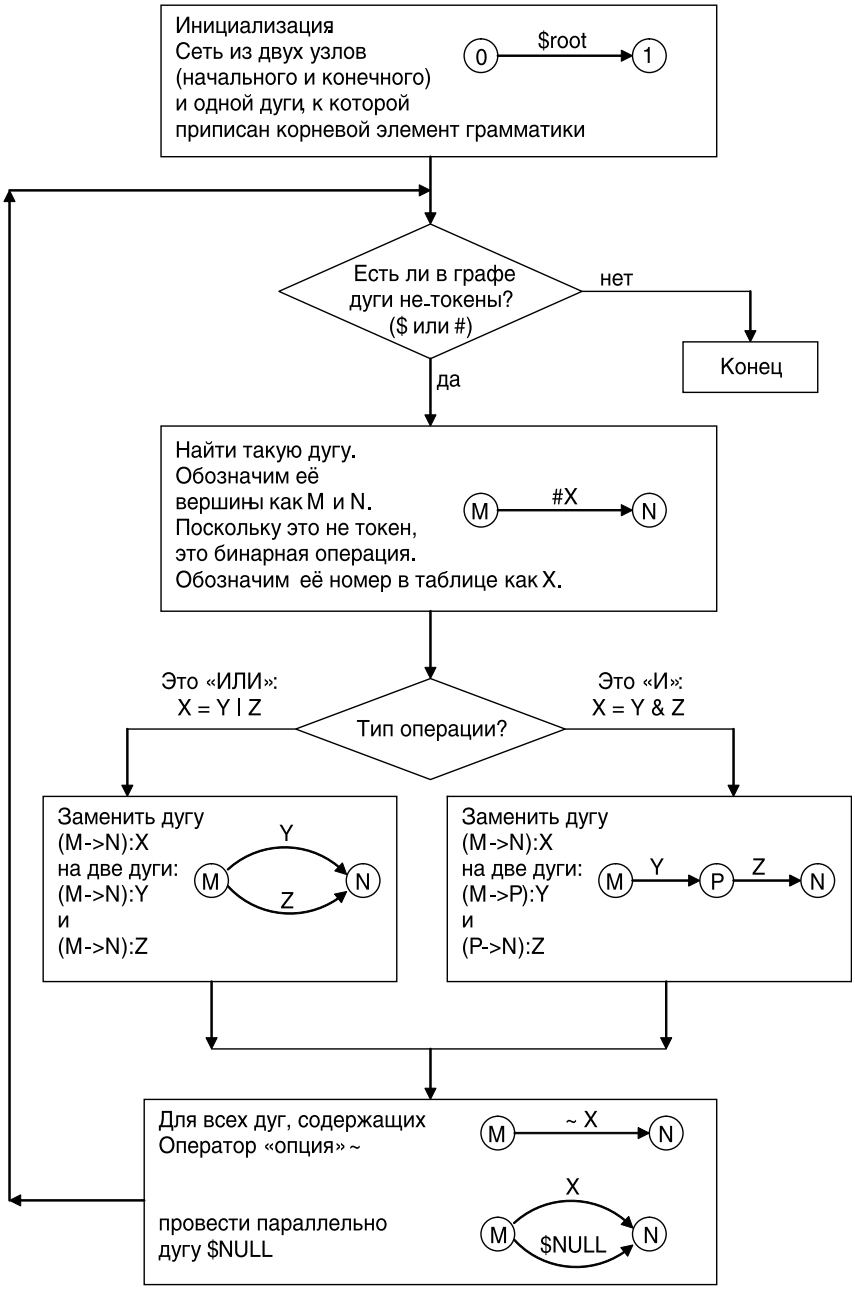

Рис. 7. Алгоритм трансляции набора бинарных операций в марковскую сеть

Несколько последовательных шагов алгоритма при обработке примера:

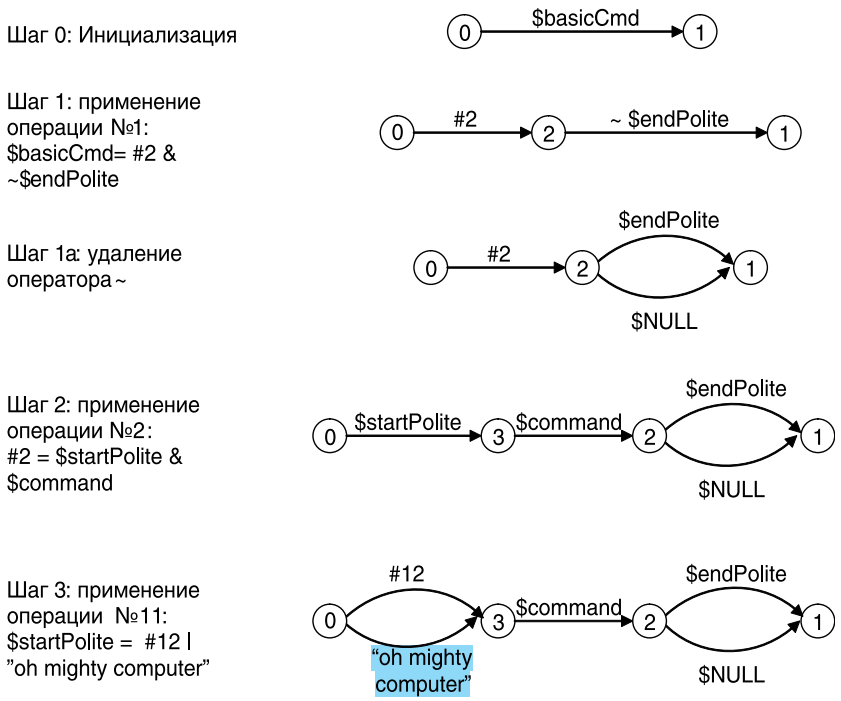

Рис. 8. Шаги алгоритма трансляции в SLF

Ориентированный граф, полученный этим алгоритмом, обладает следующими свойствами:

- отсутствуют циклы;
- единственной начальной вершиной является вершина с номером 0;
- единственной конечной вершиной является вершина с номером 1.

Представление семантической информации

Любая полная грамматика, не содержащая циклов и противоречий (двух и более разных правил с одним именем), может быть представлена в виде конечного терминального дерева и транслирована в сеть состояний вышеописанным алгоритмом. Однако при таком преобразовании утрачивается структурно-смысловая (семантическая) информация. А именно, в SLF представлении графа состояний нет возможности понять, какой части фразы (какому члену предложения) соответствует то или иное слово — ребро графа. Например, в приводимой грамматике слово «please» встречается два раза: в начале фразы (правило $startPolite) и в окончании фразы (правило $endPolite):

```
$startPolite =        please |   // rule 5
                      "could you" |
                      "oh mighty computer" {so long?};
$endPolite =          please | "thank you"; // rule 6
```

Соответственно, в графе состояний оно также встречается два раза, в первой и четвёртой строках записи:

```
J=1        S=2        E=1        W=please
J=4        S=0        E=3        W=please
```

При этом в первой строке это слово взято из окончания фразы (правило $endPolite), а в четвёртой строке — из начала фразы (правило $startPolite), что можно определить в конкретном примере по номерам вершин. В общем же случае из записи SLF установить подобное соответствие невозможно. Однако при распознавании речи определение места произносимого в данный момент слова в общей структуре предложения может оказаться важным. Например, слово «please» в начале и конце фразы может произноситься с разной интонацией, а значит, для его распознавания лучше применять разные акустико-фонетические модели.

Можно попытаться решить эту проблему, присваивая токенам уникальные имена, например, «please_start» и «please_end», но такой способ затрудняет работу составителя грамматики. Кроме того, это неэффективно для сложных грамматик с высокой степенью иерархичности, поскольку уникальные имена приписываются токенам т.е. терминальным символам — самому низкому уровню иерархии, а структура верхних уровней иерархии всё равно утрачивается. Для того чтобы сохранить информацию о верхних уровнях иерархии правил, необходимо изменить обработку самих этих правил. Простой способ добиться этого в рамках описанного алгоритма — дополнить его обработкой тегов.

Стандарт SRGS [1] предоставляет большую свободу в использовании тегов. Для поставленной цели это излишне, и использование тегов ограничено следующими правилами. Тег может следовать только непосредственно за каким-либо токеном. (В этом случае тег «приписан» к этому токену в смысле, определённом ниже.) В частности, тег не может следовать за открывающей или закрывающей скобкой, знаком операции или другим тегом (т.е. два тега подряд не допускаются). Теги можно добавлять после любых токенов — и тех, которые являются правилами, и конечных. И в любых местах — как в теле правила, так и заголовке.

Поскольку каждый тег приписан к какому-либо конкретному токену, заголовку правила или ссылке на правило, первый этап трансляции (разбиение грамматики на набор бинарных операций) не изменяется. В приведённом примере с учётом тегов получается следующий список операций:

Номер	Имя	Операция	Первый лист	Второй лист
1	$basicCmd (root)	&	#2	~ $endPolite {end}
2		&	$startPolite {start}	$command {first command}
3	$object	&	~ #4	#5
4		|	**The**	**a**

Номер	Имя	Операция	Первый лист	Второй лист
5		|	#6	**menu**
6		|	**file**	**window**
7	$command {command}	&	$action {action}	$object {object}
8	$action	|	#9	**Move**
9		|	#10	**delete {!!}**
10		|	**open**	**Close**
11	$startPolite	|	#12	**oh mighty computer {so long?}**
12		|	**could you**	**Please**
13	$endPolite	|	**please**	**thank you**

На втором этапе алгоритм изменяется следующим образом: при раскрытии дуги бинарной операцией каждой из дуг-потомков приписываются все теги родителя и теги, содержащиеся в записи соответствующей части операции. Пусть задана дуга $(M->N)$: $X\ \{TMN\}$, где M, N — некоторые вершины графа состояний, X — имя или номер правила в списке бинарных операций, $\{TMN\}$ — тег или список тегов, приписанный к дуге M->N. Пусть эта дуга раскрывается операцией типа «И»: $X\ \{TX\} = Y\ \{TY\}\ \&\ Z\ \{TZ\}$, где Y и Z — теги или номера правил, $\{TX\}$, $\{TY\}$, $\{TZ\}$ — теги, приписанные соответствующим элементам операции. Теги могут отсутствовать. По построению, в бинарных операциях теги могут быть приписаны только к нумерованным элементам. Дуга раскрывается в две дуги: $(M->P)$: $Y\ \{TY\}\{TX\}\{TMN\}$ и $(P->N)$: $Z\ \{TZ\}\{TX\}\{TMN\}$. Аналогично производится обработка при типах операции «исключающее или» и «опция».

Таким образом, происходит наследование тегов правил верхнего уровня при раскрытии их правилами нижнего уровня вплоть до токенов и сохранение информации о структуре фразы. Грамматика, рассматриваемая в качестве примера, с учётом тегов транслируется в такую запись SLF:

```
N=6 L=16
J=0 S=2 E=1 W=$NULL #{end}
J=1 S=2 E=1 W=please #{end}
J=2 S=2 E=1 W=»thank you» #{end}
J=3 S=0 E=3 W=»oh mighty computer» #{so long?}{start}
J=4 S=0 E=3 W=please #{start}
J=5 S=0 E=3 W=»could you» #{start}
J=6 S=3 E=4 W=move #{action}{command}{first command}
J=7 S=3 E=4 W=delete #{!!}{action}{command}{first command}
J=8 S=4 E=5 W=$NULL #{object}{command}{first command}
J=9 S=4 E=5 W=the #{object}{command}{first command}
J=10 S=4 E=5 W=a #{object}{command}{first command}
J=11 S=5 E=2 W=menu #{object}{command}{first command}
J=12 S=3 E=4 W=open #{action}{command}{first command}
J=13 S=3 E=4 W=close #{action}{command}{first command}
J=14 S=5 E=2 W=window #{object}{command}{first command}
J=15 S=5 E=2 W=file #{object}{command}{first command}
```

Видно, что слова «please» в первой и четвёртой строках теперь могут быть явно сопоставлены с соответствующими частями фразы.

Заключение

Предложен алгоритм трансляции автоматной грамматики (в формате SRGS), задающей структуру предложения в виде иерархии правил, в граф состояний (в формате SLF), моделирующий марковский процесс произнесения. Семантическая информация сохраняется при помощи наследуемых тегов. Алгоритм позволяет автоматически генерировать марковские модели сложных предложений, которые невозможно построить вручную. Алгоритм однопроходный, скорость выполнения линейна относительно размера итоговой модели.

Литература

1. Speech Recognition Grammar Specification Version 1.0//http://www.w3.org/TR/2004/REC-speech-grammar-20040316/
2. Augmented Backus-Naur Form//http://en.wikipedia.org/wiki/Augmented_Backus-Naur_Form
3. Backus-Naur Form//http://en.wikipedia.org/wiki/Backus-Naur_Form
4. HTK Standard Lattice Format (SLF)//http://labrosa.ee.columbia.edu/doc/HTKBook21/node257.html
5. Reverse Polish notation//http://en.wikipedia.org/wiki/Reverse_Polish_notation
6. Shunting-yard algorithm//http://en.wikipedia.org/wiki/Shunting-yard_algorithm

Матвеев Иван Алексеевич —
родился в 1973 г. Закончил ФУПМ МФТИ в 1997. Защитил диссертацию на звание к.ф.-м.н. в 1999. Работал в научно-исследовательских отделах компаний SPIRIT (2000–2001), Iritech (2001-н.в.), Samsung (2004–2009), занимающихся обработкой сигналов и биометрической идентификацией. С 2002 года научный сотрудник Вычислительного центра РАН, с 2005 года — заведующий сектором интеллектуальных систем отдела сложных систем Вычислительного центра РАН
Email: matveev@ccas.ru