УДК 681.3.06

Т. В. Грищук, к. т. н., доц.; Н. М. Быков, к. т. н., проф.

МОДЕЛИРОВАНИЕ ДИСКУРСА В СИСТЕМАХ С ГОЛОСОВЫМИ ИНТЕРФЕЙСАМИ

Представлен модифицированный синтаксис КС-грамматик. Показана возможность использования данного синтаксиса для получения интерпретируемого кода входящих команд.

***Ключевые слова**: КС-грамматика, деревья вывода, распознавание речи, голосовой интерфейс.*

Для человека речь является естественной формой общения. Поэтому реализация человеко-машинного интерфейса в современных технических системах на основе речевого ввода-вывода информации представляет собой перспективное направление развития интеллектуальных систем управления.

Обработка естественной речи – это формулирование и исследование компьютерно-эффективных механизмов для обеспечения коммуникации с ЭВМ на естественном языке. Объектами исследования являются:

- сами естественные языки;
- использование естественного языка как в общении людей, так и в общении человека с ЭВМ.

Задача исследований – создание компьютерно-эффективных моделей общения человека с ЭВМ. Именно такая постановка задачи отличает обработку естественного языка (Natural Language Processing, NLP) от задач традиционной лингвистики и других дисциплин, изучающих естественные языки, и позволяет отнести ее к области искусственного интеллекта [1].

Различают общую и прикладную NLP. Задачей прикладной NLP является разработка моделей использования речи человеком. Прикладная NLP занимается не моделированием, а непосредственно возможностью коммуникации человека с ЭВМ на естественном языке.

Большинство систем распознавания речи (СРР) предназначены для решения конкретных задач управления (приборами, процессами и т. п.), поэтому их грамматический уровень предусматривает формализацию естественного языка. В общем случае достаточным является формальное описание дискурса с помощью контекстно-свободных грамматик (КС-грамматик). Наиболее часто используемым синтаксисом записи КС-грамматик являются формы Бэкуса-Наура (БНФ) [2].

Общепринятым методом представления вывода некоторой непустой цепочки считается дерево вывода (иногда его называют деревом синтаксического разбора). Рассмотрим простое правило для вывода одного предложения: *Main (GoCommand (GoVerb ("Go"), HomeDir ("home")))*. Дерево вывода данного правила представлено на рис. 1.

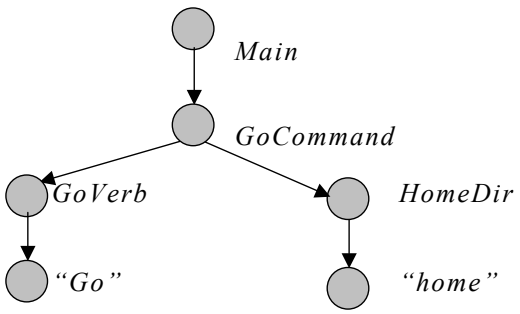

Рис. 1. Пример дерева вывода

Алгоритмическое и математическое обеспечение программного модуля голосового интерфейса должны решать две основные задачи. Во-первых, необходимо обеспечить такую организацию входной грамматики, которая бы позволила отказаться от сохранения самих правил. Во-вторых, синтаксис КС-грамматики и математическая модель организации фраз КС-грамматики должны предоставлять возможность преобразования любой входной фразы в интерпретированный код с возможностью применения структурного или объектно ориентированного подходов.

В данной статье представлен новый способ записи КС-грамматик, на основе которого можно построить граф грамматики. Вывод фраз грамматики происходит путем обхода графа в ширину, что дает возможность параллельно выводить все фразы грамматики для увеличения скорости распознавания [3].

Вершинами данного графа являются терминалы, а дуги – это правила грамматики. Основным недостатком данного подхода является то, что мы теряем информацию о размещении нетерминальных символов, которые необходимы для контроля процесса распознавания при обходе всех грамматических цепочек на представленном графе. Нетерминальные символы представляют собой точки разветвления. Поскольку один и тот же терминал может выступать в грамматике как отдельно, так и в роли части нетерминала, то при генерировании цепочек на графе возникает необходимость в сложном грамматическом контроле. Покажем это на примере дискурса, который описывает процесс форматирования некоторого текста.

В данном примере мы описываем ситуацию, когда пользователь может сменить стиль написанного им текста на «полужирный» (*bold*) или на «курсив» (*italic*).

Запишем грамматику в БНФ:

Grammar “Format”

< main > : make < object > < style > |

bold it;

< object > : it | sentence;

< style > : bold | italic;

Данная грамматика описывается через 5 (*make, it, sentence, bold, italic*) терминальных символов и 2 нетерминальных (*object*, *style*).

Дискурс в данном случае описывается таким графом:

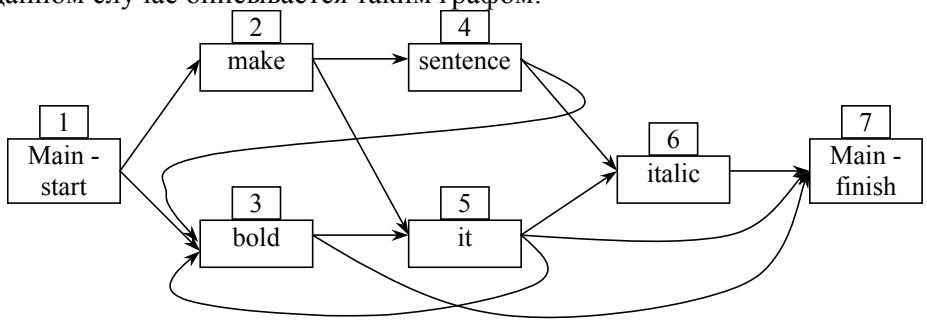

Рис. 2. Граф грамматики “Format”

В данном случае возникает необходимость выполнять грамматический контроль цепочки после выхода из каждого терминала. Так, исходя из рис. 2, для представленного графа возможны циклические повторения фразы «*bold it*», что, естественно, запрещено грамматикой.

Существующие методы грамматического вывода характеризуются также сложностью по разным типам неопределенности, главной из которых является синтаксическая неопределенность, когда один и тот же символ входит одновременно в несколько правил

грамматики. Для устранения такой неопределенности необходимо вводить дополнительные операции контроля синтаксиса.

Для того, чтобы упростить процедуру грамматического контроля, в статье предлагается заменить вид графа грамматики путем внесения дополнительных вершин, которые будут означать входы и выходы нетерминальных символов. Так, каждое правило типа $< S >= a < b >$ будет переведено в форму $< s\, a < b\, b > s >$.

На рис. 3 показан модифицированный граф грамматики. Несложный визуальный анализ данного графа свидетельствует, что теперь нет необходимости в дополнительных ссылках на правила грамматики.

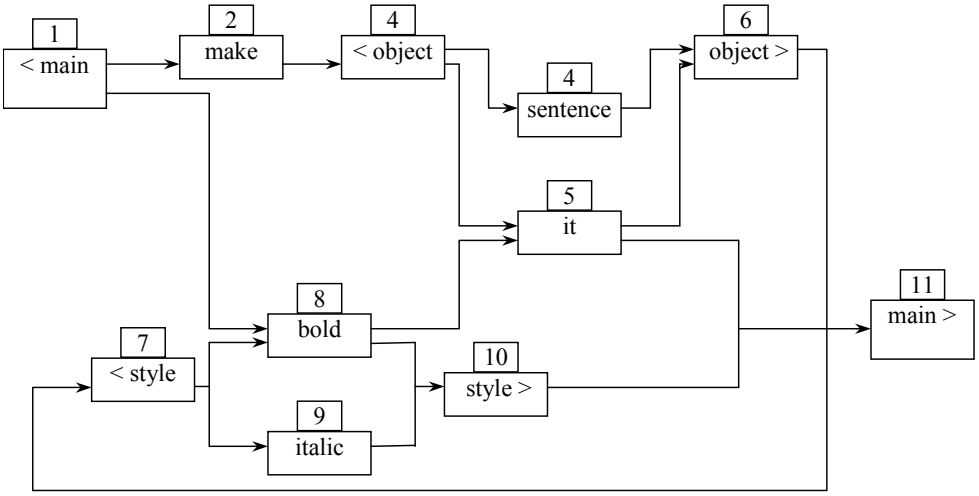

Рис. 3. Граф грамматики в модифицированном синтаксисе

Основной принцип действия голосового интерфейса – это преобразование входной команды пользователя, поданной естественным языком, в программный код на языке интерпретатора, который сразу же будет исполнен. Из предыдущего примера видно, что классическая форма записи позволяет просто описать входные команды пользователя, но нуждается в использовании дополнительных средств для преобразования входной команды в выходной программный код. Для выполнения необходимого преобразования необходимо на основе начального дерева вывода грамматики получить такое дерево вывода, на ветках которого будут находится терминальные символы, которые фактически являлись бы частями программного интерпретируемого кода.

Классический подход имеет следующие преимущества:

- описание входной части отделено от части построения выходной грамматики;
- пошаговое формирование выходного дерева позволяет отслеживать процесс формирования, что помогает при редактировании грамматики;
- наличие возможности определения некорректных или нежелательных команд.

Недостатками подхода являются:

- при описании правил преобразования необходимо четко представлять себе структуру дерева и осознавать влияние каждого шага на результирующее дерево;
- сложность изменения проявляется в том, что внесение изменений на одном уровне влечёт за собой необходимость в изменении правил на следующих уровнях.

Вторым преимуществом предложенного скобочного синтаксиса описания грамматик является то, что на его основе можно строить графы атрибутивных грамматик. В этом случае с вершинами графа ассоциируются элементы выходного языка (скрипта, который в конечном итоге выполнит команду). Данный способ записи позволяет реализовать в процессе разработки сложных грамматических конструкций принцип структурного программирования. Так, на верхних уровнях правил грамматики можно описывать общую структуру исполняемого скрипта, а на нижних уровнях грамматики – непосредственно

получать фактические параметры.

Рассмотрим пример атрибутивной грамматики. Данная грамматика описывает четыре фразы, каждая из которых транслируется в некоторый условный скрипт.

```
<main(script: my_color, my_object)>:
        draw <color(my_color)> <object(my_object)>
                {script = “Draw (” + my_object + “,” + my_color “);”;};
        <color(color)>:
                red     {color = “red”;} |
                green {color = “green”;};
        <object(object)>:
                rectangle {object = “rectangle”;} |
                circle       {object = “circle”;}
```

В скобочном синтаксисе данная граматика имеет следующий вид:

<main draw <color color> <object object> {α_1} main>
<color red {α_2} color>
<color green {α_3} color>
<object rectangle {α_4} object>
<object circle {α_5} object>,

где:

{α_1} = {script = “Draw (” + my_object + “,” + my_color “);”;}
{α_2} = {color = “red”;}
{α_3} = {color = “green”;}
{α_4} = {object = “rectangle”;}
{α_5} = {object = “circle”;}.

На рис. 4 приведен граф рассматриваемой грамматики.

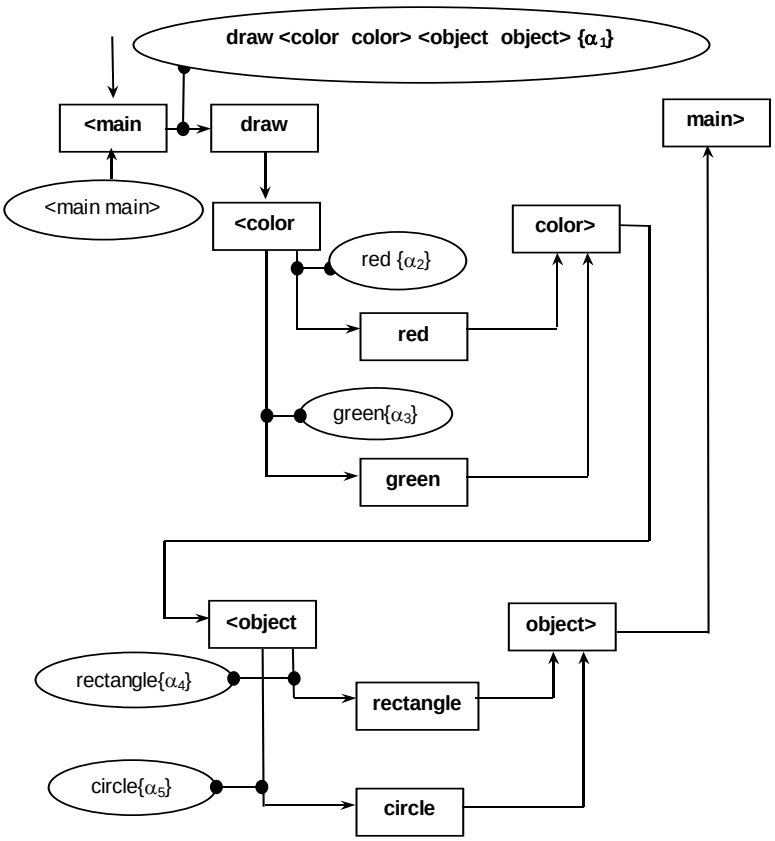

Рис. 4. Пример графа для атрибутивной грамматики

В результате генерации фразы на грамматической сети существует возможность получить не только фразу, и её вывод в скобочной форме. Например, для фразы:

draw red rectangle

получаем вывод:

<main draw <color red {α_2} color> <object rectangle {α_4} object> {α_1} main>.

Интерпретация скобочного синтаксиса процесса вывода приводит к генерации кода на внутреннем языке (Intermediate Translation Language – ITL). Для рассматриваемой фразы получаем ITL код, который на высоком уровне детализации эквивалентен следующему:

```
Entry point()
{
        Declare translation;
        Main(translation);
}
Main(script)
{
        Declare my_color, my_object;
        Color(my_color);
        Object(my_object);
        script = “Draw (” + my_object + “,” + my_color “);”
}
Color(color)
{
        color = “red”;
}
Object(object)
{
oject = “rectangle”;
}
```

Интерпретация ITL кода, который получен на грамматической сети, дает следующий результат трансляции: Draw (rectangle, red);.

Выводы

Таким образом, рассмотренный способ записи КС-грамматик позволяет не только представлять грамматику в виде ориентированного взвешенного графа, но и выполнять параллельный вывод фраз грамматики без дополнительного грамматического контроля для устранения грамматической неопределенности. Еще одним преимуществом данного похода является возможность получать интерпретированный код на этапе вывода входной команды, используя основные принципы структурного программирования.

СПИСОК ЛІТЕРАТУРЫ

1. Encyclopaedia of Artificial Intelligence. Entry Natural Language Understanding. – P. 660 – 677. – Режим доступу: http://sabia.tic.udc.es/encyclopediaAI/.

2. Льюис Ф. Теоретические основы проектирования компиляторов / Льюис Ф., Розенкранц Д., Стирнз Р. – М.: Мир, 1979. – 654 с.

3. Грищук Т. В. Розпізнавання природної мови на граматичних марковських мережах / Т.В. Грищук // Наукові праці Донецького національного технічного університету. Серія: “Обчислювальна техніка та автоматика”. – Донецьк: ДонНТУ, 2005. – С. 181 – 187.

Грищук Татьяна Викторовна **–** к. т. н., доцент кафедры компьютерных систем управления, тел.: (0432)-598222, thryshuk@mail.ru

Быков Николай Максимович **–** к. т. н., профессор кафедры компьютерных систем управления Винницкий национальный технический университет.