УДК 519.682.1

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД К ОПИСАНИЮ И РАСШИРЕНИЮ СИНТАКСИСА ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Д. И. Соломатин

Воронежский государственный университет

Поступила в редакцию 24.04.2008 г.

Аннотация. В статье рассматривается механизм расширения синтаксиса языков программирования, основанный на применении к PEG-грамматикам (Parsing Expression Grammars) принципов объектно-ориентированного программирования.

Ключевые слова: языки программирования, синтаксис, PEG-грамматики, объектно-ориентированное программирование.

Abstract. The article describes the mechanism for programming language syntax extensibility based on the principles of object-oriented programming as applied to PEG-grammars (Parsing Expression Grammars).

Key words: programming languages, syntaxis, Parsing Expression Grammars, object-oriented programming.

ВВЕДЕНИЕ

Реализация любого языка программирования или же другого однозначного языка (под однозначным языком понимается формальный язык, любые выражения которого поддаются строго однозначном разбору), как правило, начинается с описания синтаксиса этого языка в какой-либо форме. Часто для этих целей используют LL- или LR-грамматики. При этом построение лексического и синтаксического анализаторов языка по таким грамматикам выполняется автоматически с помощью подходящей утилиты построения анализаторов, например, Lex/Yaxx и их развитие Flex/Bison (LALR(1)) для языка C (а также множество клонов для других языков), JavaCC (LL(1)) – для Java, ANTLR (LL(K)) – для нескольких языков (C, Java, C#, Python, Ruby и др.) и т.д. Т.о. способы задания неизменяемого синтаксиса языков широко известны и автоматизированны.

Теперь рассмотрим задачу разбора языка с расширяемым синтаксисом. Под расширением синтаксиса будем понимать возможность добавления в язык новых синтаксических конструкций или изменение существующих. При этом расширение синтаксиса может быть *статическим*, т.е. на момент запуска синтаксического анализатора новый синтаксис полностью известен, и *динамическим*, когда синтаксис может меняться в процессе синтаксического анализа. При этом статическое расширение синтаксиса можно рассматривать как частный случай динамического.

В случае статического расширения синтаксиса ключевым моментом, позволяющим говорить именно о расширении синтаксиса, а не об определении нового языка, можно считать наличие какого-либо формализма, используемого при описании расширений синтаксиса. Т.е. если синтаксис языка описывается в виде грамматики какого-либо вида, то данный формализм должен заключаться в способе описания именно изменений, вносимых в грамматику, в отличие от простого модифицирования исходной грамматики. Для простоты понимания данного тезиса следует рассматривать ситуацию, когда полный исходный текст грамматики языка недоступен (поставляется в скомпилированном виде или является результатом применения к исходной грамматике нескольких заранее неизвестных расширений).

В данной статье рассматривается объектно-ориентированный подход к построению механизма расширяемого синтаксиса, реализованный автором в экспериментальном генераторе синтаксических анализаторов (парсеров) для PEG-грамматик (PEG – Parsing Expression Grammars, были предложены в 2002 г. Брайном Фордом (Bryan Ford) в работе [3]).

1. ТРЕБОВАНИЯ К КЛАССУ ГРАММАТИК

Очевидно, что синтаксис расширяемого языка должен описываться с помощью формализма какого-либо класса грамматик. Обозначим требования, которым должен удовлетворять данный класс грамматик:

1. Объединение описания лексических и синтаксических правил в грамматиках (в отличие, например, от утилит lex и yaxx, для которых отдельно описываются грамматики для лексического и синтаксического анализаторов).

2. Возможность добавления новых правил к существующей грамматике или изменение правил этой грамматики (как вариант, объединение нескольких грамматик), при динамическом расширении синтаксиса – в ходе самого разбора.

3. Строгая однозначность разбора для любых предложений языка, как базового, так и с расширениями.

Первое требование объясняется соображениями простоты конструирования расширений синтаксиса и важно для практического применения.

Второе и третье требования можно строго сформулировать следующим образом: для описания синтаксиса синтаксически расширяемого языка необходимо использовать класс порождающих строго однозначный разбор грамматик, замкнутый относительно операций добавления новых правил в грамматику и изменения существующих.

LL(k)- и LR(k)/LALR(k)-грамматики не удовлетворяют данному определению, т.к. при добавлении (изменении) произвольных правил может получиться грамматика, не удовлетворяющая требованию LL(k) или LR(k). Действительно, при этом могут образоваться несколько правил с одинаковой начальной терминальной цепочкой длины k, т.е. после предпросмотра k символов нельзя будет сделать однозначный выбор правила, которое следует применить.

2. PEG-ГРАММАТИКИ

Указанным в предыдущем разделе требованиям к классу грамматик для синтаксически

Таблица 1.

Выражения, допустимые в PEG-грамматиках

E	пустая строка	Распознается всегда. Указатель не сдвигается.
A	терминальный символ $(a \in Z)$	Может быть в том числе и символом конца последовательности. Распознается, если прочитанный символ совпадает с a. В этом случае указатель сдвигается на один символ.
A	нетерминальный символ $(A \in N)$	Распознается, если распознается последовательность выражений, определяющая нетерминал A.
$e?$	“опционально”	Распознается всегда. Если выражение e было распознано, то указатель сдвигается.
$e*$	“ноль или много”	Распознается всегда. Указатель сдвигается на количество распознанных выражений e.
$e+$	“один или много”	Распознается, если распознано хотя бы одно выражение e. Указатель сдвигается на количество распознанных выражений e.
$\&e$	И-предикат	Распознается, если распознано выражение e. Указатель не сдвигается.
$!e$	НЕ-предикат	Распознается, если не распознано выражение e. Указатель не сдвигается.
$e_1\, e_2 \ldots e_n$	последовательность выражений	Распознается, если распознаны все выражения e_1-e_n. Указатель сдвигается на все распознанные выражения.
$e_1 / e_2 / \ldots / e_n$	упорядоченный выбор	Распознается, если распознано хотя бы одно выражение e_1-e_n. Выражения e_1-e_n распознаются в строго заданном порядке. Если распознано одно из выражений, дальнейшее распознавание прекращается. При этом указатель смещается на распознанное выражение.

расширяемых языков удовлетворяют PEG-грамматики. PEG-грамматики базируются на концепциях ЯНРОВ (язык нисходящего разбора с ограниченными возвратами; англ.: TDPL – top-down parsing language) и ОЯНРОВ (обобщенный ЯНРОВ; англ.: GTDPL – generalized TDPL), предложенных в работах [4, 5]. В этих алгоритмах нетерминалы трактуются как процедуры, сообщающие о том, обнаружена или нет подходящая цепочка входных символов.

В виде ЯНРОВ и ОЯНРОВ можно описать все детерминированные контекстно-свободные (КС) языки с концевыми маркерами (т.е. для любые LL- и LR-языки) и благодаря возможности ограниченных возвратов даже некоторые не КС-языки [1]. В PEG-грамматиках по сравнению с ОЯНРОВ вводятся так называемые синтаксические предикаты для упрощения задания языка, но все PEG-грамматики можно свести к ОЯНРОВ. Таким образом с помощью PEG-грамматик можно описать широкий класс используемых на практике языков.

Ниже в таблице перечислены выражения, допустимые в PEG-грамматиках, с комментариями относительно семантики данных выражений (указатель отделяет уже распознанную часть входной цепочки символов от нераспознанной).

Как видно из таблицы, при описании языка с помощью PEG-грамматик фазы лексического и синтаксического анализа не разделяются.

Для PEG-грамматик может быть применен алгоритм разбора “старьевщика” (packrat parsing algorithm), предложенный в 60-х годах еще для ОЯНРОВ. Этот алгоритм обеспечивает линейное время работы за счет сохранения в памяти предыдущих результатов разбора для каждого нетерминала в текущей позиции указателя.

Недостатком PEG-грамматик является невозможность применения леворекурсивных правил вывода, однако любая леворекурсивная грамматика может быть преобразована в эквивалентную ей грамматику без левой рекурсии.

3. ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД К ОПРЕДЕЛЕНИЮ СИНТАКСИСА

Т.к. правила вывода в PEG-грамматиках могут трактоваться как процедуры, то механизм расширения таких грамматик можно представить в виде добавления в грамматику новых процедур или изменения реализации существующих. Легко заметить аналогию между таким представлением и объектно-ориентированной парадигмой программирования, при этом PEG-грамматики соответствуют классам в ООП, а правила вывода – методам классов (в случае изменяемых правил – виртуальным методам). Расширение грамматики (создание новой грамматики на основе существующей) при этом следует трактовать как наследование новой грамматики от существующей. В дочерней грамматике нет необходимости повторять наследуемые правила, чем обеспечивается повторное использование и модульное представление кода грамматик.

Учитывая, что любая PEG-грамматика однозначно определяет синтаксис языка, ей соответствующего, можно в этом смысле говорить об объектно-ориентированном, модульном подходе к определению синтаксиса расширяемых языков.

Предлагаемый подход был успешно опробован в созданном автором генераторе лексико-синтаксических анализаторов PEG-PG (PEG Parser Generator) [2]. (PEG-PG представляет собой утилиту, которая по входной PEG-грамматике строит java-код парсера, распознающего предложения на языке, описанном данной грамматикой, т.е. так называемый «компилятор компиляторов»).

Следует уточнить, что с помощью PEG-PG можно простым образом создать синтаксические анализаторы для статических расширений синтаксиса языка. Однако предложенный способ может быть использован и для динамического расширения синтаксиса. Действительно, если правила PEG-грамматики во время работы синтаксического анализатора заданы в виде структуры данных в памяти, а выполнение нетерминалов-процедур эмулируется анализатором, то добавлять новые правила или изменять существующие можно, внося изменения в структуру данных. Т.о., добавив в язык конструкции, семантика которых заключается в изменении/добавлении правил вывода в момент разбора, можно получить язык, синтаксис которого можно изменять с помощью средств самого языка. На практике синтаксические расширения для такого языка удобно будет реализовывать в виде подключаемых модулей.

4. ПРИМЕР МОДУЛЬНОГО ОПИСАНИИ И РАСШИРЕНИЯ ЯЗЫКА

Для пояснения идеи ниже приводится пример модульного описания синтаксиса примитивного языка, в котором допустимы только целые числа, записанные в десятичном представлении через произвольное число пробельных символов, и комментарии, а также его синтаксического расширения, допускающего шестнадцатеричный формат записи чисел. На рисунке показана *диаграмма грамматик* для рассматриваемого примера, использующая формализм диаграмм классов (диаграмма классов для классов, полученных в результате трансляции данных грамматик утилитой PEG-PG, если отбросить вспомогательные классы, будет полностью повторять данную диаграмму грамматик).

DecimalIntGrammar описывает правило, применяемое для распознавания десятичной записи целого числа (здесь и далее приводятся PEG-грамматики в формате, принятом для PEG-PG):

```
@class = “DecimalIntGrammar” ;

^~digit: [0-9] ;
~number: digit+ ;
// ^ — отмена сохранения результатов разбора (экономия памяти)
// ~ — “лексическое” правило (не допускает незначащего текста)
```

HexIntGrammar описывает правило, применяемое для распознавания шестнадцатеричной записи целого числа:

```
@class = “HexIntGrammar” ;

^~digit: [0—9] / [a—f] / [A—F] ;
~number: “0x” digit+ ;
```

SkipTextGrammar описывает правило, применяемое для распознавания незначащего текста (пробелы и комментарии):

```
@class = “SkipTextGrammar” ;

~skip:
 (
 [\n,\r,\t,\r,\s]+ // пробельные символы
 / “/*” ( !”*/” . )* “*/” // многострочный комментарий
 / “//” ( !’\n’ . )* // однострочный комментарий
 )*
;
```

IntListGrammar – грамматика с начальным правилом, описывающая первоначальный синтаксис языка:

```
@class = “IntListGrammar” ;

listItem: DecimalIntGrammar.number ;
content: listItem* ;

// skip – зарезервированное правило в PEG—PG, дополнительно
// вызывается между терминалами в “нелексических” правилах
~skip: SkipTextGrammar.skip ;
// start — зарезервированное правило в PEG-PG, с применения
// данного правила начинается разбор входной цепочки
start: content # ; // # — конец входной цепочки
```

HexIntListGrammar – грамматика, описывающая синтаксическое расширение рассматриваемого языка, добавляющее возможность записи чисел в шестнадцатеричном формате. Как видно из приведенной выше диаграммы грамматик, HexIntListGrammar унаследована в терминах ООП от грамматики IntListGrammar:

```
@class = “HexIntListGrammar” ;
@extends = “IntListGrammar” ; // указание родительской грамматики

// перегрузка правила с обращением к предыдущей версии правила
// с помощью ключевого слова super, т.е. фактически
// добавление еще одной альтернативы к перегруженному правилу
listItem: HexIntGrammar.number / super.listItem ;
```

В данном примере различные форматы записи чисел можно рассматривать как различные конструкции языка, следовательно, в этом смысле можно говорить, что расширение синтаксиса языка, описанное в HexIntListGrammar, добавляет новую синтаксическую конструкцию в первоначальный язык.

5. РАСШИРЕНИЕ СУЩЕСТВУЮЩИХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Добавление еще одной альтернативы к перегруженному правилу – наиболее частый прием при расширении синтаксиса языков.

Дело в том, что обычно расширение синтаксиса языка заключается в добавлении новой синтаксической конструкции в язык, а в грамматике языка обычно присутствует правило описания множества допустимых конструкций языка. Поэтому для такого расширения языка достаточно добавить в качестве альтернативы к данному правилу новую синтаксическую конструкцию.

Например, грамматика для языка Java, приведенная в спецификации языка [6], содержит правило Statement, в котором в виде набора альтернатив описаны допустимые в языке Java выражения. Следовательно, добавление новой синтаксической конструкции в язык Java, допустим, для поддержки SQL-запросов, в PEG-грамматике может выглядеть следующим образом:

```
...
Statement:
 Super.Statement /
 “sql” ‘{‘ SqlSpec.Expression ‘}’
;
...
```

где SqlSpec – грамматика, описывающая язык SQL, а Expression – правило, применяемое для распознавания допустимых SQL-запросов (следует заметить, что для данного применения Expression может допускать распознавание и некорректных SQL-запросов, достаточно отделения «как бы» SQL-запросов от Java-кода).

ЗАКЛЮЧЕНИЕ

В данной статье не затрагивались вопросы реализации семантики синтаксических расширений языка. Между тем, семантика значительного класса расширений языков программирования может быть описана с помощью конструкций исходного языка и реализована локальным преобразованием кода программы, т.е. такая задача может быть легко выполнена препроцессором для статических расширений. Разработанный генератор синтаксических анализаторов PEG-PG допускает встраивать в грамматику java-код (*семантические действия*), который будет выполняться по ходу разбора входной цепочки. С помощью данного механизма легко могут быть реализованы такого рода препроцессоры для новых конструкций языка. На данный момент таким способом реализован тестовый препроцессор для обработки встроенных SQL-инструкций в языке Java.

СПИСОК ЛИТЕРАТУРЫ

1. *Ахо А., Ульман Дж.* Теория синтаксического анализа, перевода и компиляции. – М.:Мир, 1978, Т. 1.

2. *Соломатин Д.И.* Разбор для PEG-грамматик. Материалы восьмой международной научно-методической конференции «Информатика: проблемы, методология, технологии». Воронеж, 2008. – С. 279—283.

6. *B. Ford*. Packrat Parsing: a Practical Linear-Time Algorithm with Backtracking – Master’s Thesis Massachusetts Institute of Technology. http://pdos.csail.mit.edu/~baford/packrat/thesis/thesis.pdf

7. *R. M. McClure*. TMG – a syntax directed compiler. In Proceedings of the 20th ACM National Conference, 1965.

8. *D. V. Schorre*. META II, a syntax-oriented compiler writing language. In Proceedings of the 19th ACM National Conference, 1964.

9. The Java Language Specification, Third Edition. http://java.sun.com/docs/books/jls/

Соломатин Дмитрий Иванович – ассистент кафедры Программирования и информационных технологий факультета Компьютерных наук Воронежского государственного университета, тел. (4732) 208-470.

Solomatin D.I. — Assistant, the dept. of Programming and Information Technologies, Voronezh State University. Tel. (4732) 208-470.