519.682.1

СИНТАКСИЧЕСКИЕ МАКРОСЫ И ИХ РЕАЛИЗАЦИЯ В ГЕНЕРАТОРЕ СИНТАКСИЧЕСКИХ АНАЛИЗАТОРОВ

Д. И. Соломатин

Воронежский государственный университет

Поступила в редакцию 1.03.2009 г.

Аннотация. В статье рассматривается применение синтаксических макросов для расширения языков программирования, синтаксис которых задан в виде PEG-грамматики (Parsing Expression Grammar). Описан алгоритм разбора PEG-грамматик (Parsing Expression Grammars) с синтаксическими макросами, реализованный в генераторе синтаксических анализаторов.

Ключевые слова: языки программирования, синтаксический разбор, PEG-грамматики, генератор синтаксических анализаторов.

Abstract. The article describes usage of syntax macros for extension of programming languages whose syntax is specified by PEG-grammars (Parsing Expression Grammars). Also is described the algorithm of parsing PEG-grammars with syntax macros. Algorithm is realized in parser generator.

Key words: programming languages, syntax parsing, PEG-grammars (Parsing Expression Grammars), parser generator.

ВВЕДЕНИЕ

Идея создания языков с расширяемым синтаксисом и семантикой выглядит довольно привлекательной. Действительно, возможность под каждую специфическую задачу реализовать свое подмножество исходного языка, удобное для работы над данной задачей, может значительно сократить время разработки, а также сделать код приложения более прозрачным и содержащим меньшее количество ошибок. В качестве расширений языка могут реализовываться различные языки предметных областей (DSL – Domain Specific Languages), новые для языка парадигмы программирования, поддержка некоторых паттернов проектирования, да и просто различные выражения для упрощения записи кода («синтаксический сахар»).

Рассмотрим задачу реализации механизма расширения языка более подробно. В данной задаче можно выделить две принципиально различных исходных ситуации:

- создание расширения к существующему языку программирования (т.е. к такому, при создании которого не предусматривался механизм расширения);
- разработка нового языка с поддержкой расширения синтаксиса и семантики.

© Соломатин Д. И., 2009

В обоих случаях для новых синтаксических конструкций необходимо каким-либо образом описывать семантику разрабатываемых расширений. Здесь также принципиально возможны два подхода:

- разрабатывать для каждого конкретного синтаксического расширения языка отдельный модуль компилятора или интерпретатора, реализующий семантику данного расширения, т.е. описывать расширения языка сторонними по отношению к собственно языку средствами;
- задавать семантику расширений языка с помощью средств самого языка.

Второй вариант описания семантики может быть реализован в виде так называемых синтаксических макросов, т.е. механизма описания семантики новых синтаксических конструкций языка через существующие, семантика которых определена.

1. СИНТАКСИЧЕСКИЕ МАКРОСЫ В ГЕНЕРАТОРЕ СИНТАКСИЧЕСКИХ АНАЛИЗАТОРОВ PEG-PG

Автором данной статьи была предпринята попытка реализации синтаксических макросов в генераторе синтаксических анализаторов PEG-PG (PEG Parser Generator) [2], т.е. не в конкретном языке программирования, а в инструменте для описания произвольных языков.

В PEG-PG для описания языков используются PEG-грамматики (PEG – Parsing Expression Grammars). PEG-грамматики (PEG – Parsing Expression Grammars) предложены Брайном Фордом (Bryan Ford) в 2002 г. в работе [4] для разбора синтаксиса выражений. 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 [5, 6]. В этих алгоритмах нетерминалы трактуются как процедуры, сообщающие о том, обнаружена или нет подходящая цепочка входных символов. В отличие от других используемых на практике видов грамматик (LL(k), LR(k)) PEG-грамматики всегда порождают строго однозначный разбор предложений языка, поэтому на их основе возможно построение механизма расширения синтаксиса. Данный механизм может быть реализован, например, с виде перегрузки нетерминалов подобно виртуальным методам в объектно-ориентированных языках программирования [3].

В качестве примера описания синтаксического макроса в PEG-PG рассмотрим описание цикла FOR через цикл WHILE для C-подобных языков (рис. 1.).

В приведенном примере предполагается, что до появления данного фрагмента в грамматике уже описан нетерминал expr (выражение), который способен разбирать описание цикла WHILE, а также составное выражение вида “{” последовательность_одиночных_выражений “}”. Второй альтернативой (оператор “/” используется в PEG-PG для перечисления альтернатив, применяемых при разборе в порядке их описания) в определении новой версии expr присутствует вызов предыдущей версии данного нетерминала. Таким образом новая версия expr способна разбирать цикл WHILE, а также все остальные виды выражений, разбираемые предыдущей версии expr. При этом семантика FOR определяется через синтаксические конструкции (WHILE, составное выражение), семантика которых уже каким-либо образом определена.

Данный синтаксический макрос взят из примера, в котором результатом разбора входной строки является построение абстрактного синтаксического дерева (abstract syntax tree – AST) данной строки, соответственно, результатом разбора любой синтаксической конструкции будет AST-поддерево, соответствующего данной конструкции. Здесь “<AstNode>” указывает на возвращаемый нетерминалом expr тип результата разбора (AstNode – Java-класс, описывающий узел AST-дерева). “$result:” – один из способов присваивания возвращаемого нетерминалом значения. В приведенном примере, в первом случае за результат принимается результат применения макроса, во втором – результат применения предыдущей версии нетерминала expr.

Таким образом синтаксический макрос в PEG-PG состоит из:

· *грамматики макроса* (G – grammar), т.е. последовательности корректных PEG-выражений, отвечающих за распознавание части входного текста (входной цепочки символов), подлежащей замене (иной интерпретации);

· *текста замены* или *тела макроса* (S – string), т.е. фрагмента текста, который следует использовать (подставить/интерпретировать) вместо заменяемого исходного фрагмента;

· *правила разбора (для текста замены)* (R – rule), т.е. правила (нетерминала) из PEG-грамматики, которое разбирает текст замены (при применении данного правила в качестве начального к тексту замены).

Т.е. макрос представляет собой тройку (G, S, R).

При этом любой макрос определяется только в виде составной части какого-либо правила

```
<program>
└ ..
  ├ <call>
  │ ├ print
  │ └ "["
  ├ ..
  │ ├ :=
  │ │ ├ i
  │ │ └ 0
  │ └ <while>
  │   ├ <
  │   │ ├ i
  │   │ └ 5
  │   └ ..
  │     ├ <call>
  │     │ ├ print
  │     │ └ i
  │     └ :=
  │       ├ i
  │       └ +
  │         ├ i
  │         └ 1
  └ <call>
    ├ print
    └ "]"
```

Рис. 1.

в PEG-грамматике и описывается в PEG-PG в виде следующей конструкции:

```
#macro(
  // грамматика макроса
<правило_разбора>:=
  // текст замены (тело макроса)
)#
```

Как видно из первого примера, в грамматике макроса вызываемые нетерминалы можно связывать с переменными, а в тексте замены использовать эти переменные как ссылки на фрагменты заменяемого текста. Формат таких ссылок может меняться для того, чтобы гарантировать синтаксическое отличие ссылок от конструкций описываемого языка в тексте замены макроса.

Алгоритм синтаксического разбора с учетом макросов рассмотрен ниже.

2.АЛГОРИТМ ДВУХПРОХОДНОГО РАЗБОРА

Под алгоритмом разбора будем понимать алгоритм синтаксического разбора входной цепочки в соответствии с правилами, заданными в грамматике языка, в совокупности с выполнением семантических действий, описанных в этих правилах.

В PEG-PG в этом смысле применяется двухпроходный алгоритм разбора для PEG-грамматик. Первый шаг алгоритма соответствует алгоритму разбора ЯНРОВ-программ за линейное время [1] с дополнительным сохранения в таблице промежуточных результатов разбора для каждого оператора упорядоченного выбора номера распознанной альтернативы для текущего положения указателя, а для операторов повторения шаблона – числа повторений. Таким образом после первого шага фактически строиться дерево разбора, по которому на втором шаге достаточно еще раз «пройти», попутно вызывая семантически действия, связанные с примененными в процессе разбора правилами. Эти два шага условно можно описать в виде методов синтаксического анализатора:

- ParseResult parse(Rule rule, int pos) и
- ExecuteResult execute(Rule rule, int pos),

где rule – правило, которое необходимо применить в текущей позиции указателя pos, а новые составные типы данных ParseResult и ExecuteResult определены следующим образом:

- ParseResult : (result : bool, newPos : int),
- ExecuteResult : (result : Object, newPos : int)

Таким образом метод parse пытается применить правило грамматики rule к входной цепочке, начиная с позиции pos. В случае успешного разбора в result возвращается значение true, а в newPos – позиция указателя после применения правила rule к входной цепочке, т.е. rule разбирает часть входной цепочки с позиции pos до newPos-1 включительно. Кроме того, во внутренних полях экземпляра анализатора строится соответствующая структура (дерево разбора) для последующего применения метода execute начиная с позиции pos (экземпляр анализатора всегда связан с конкретной разбираемой входной цепочкой).

Метод execute, вызванный для входной цепочки в этой же позиции pos, выполняет в нужной последовательности семантические действия, связанные с правилом rule и правилами, которые рекурсивно вызывались из rule при разборе части входной цепочки с позиции pos в методе parse. Возвращаемый правилом результат, если правило какой-либо результат возвращает, записывается в result (конкретный тип результата зависит от правила rule). Кроме того, метод execute может иметь побочные эффекты, связанные с выполнением семантических действий (здесь под побочными эффектами понимается изменение объектов, глобальных по отношению к вызываемому правилу).

По аналогии с понятием экземпляра синтаксического анализатора следует ввести понятие класса синтаксических анализаторов, которое полностью соответствует понятию класса в объектно-ориентированном программировании. Класс синтаксических анализаторов определяется грамматикой (G) языка, входные цепочки которого могут быть разобраны анализатором данного класса. Стоит заметить, что здесь грамматика – набор всех правил разбора данной грамматики, отличается от понятия грамматики макроса – фрагменте правой части какого-то одного правила из грамматики языка.

Экземпляр синтаксического анализатора в этом случае определяется классом анализатора, т.е. грамматикой, а также входной цепочкой (S). Т.о. экземпляр анализатора определяется двойкой (G, S), условно конкретный экземпляр будем обозначать через $P_{(G, S)}$ или, когда речь идет об одном и том же классе анализаторов, через $P_{(S)}$. Начальное правило разбора не требуется для определения экземпляра анализатора, т.к. является параметром методов parse и execute.

С учетом таких обозначений разбор корректной входной цепочки S языка, который определяется грамматикой G с начальным правилом R, можно записать как последовательный вызов 2-х методов:

$P_{(G, S)}$.parse(R, 0);

$P_{(G, S)}$.execute(R, 0);

3. АЛГОРИТМ ДВУХПРОХОДНОГО РАЗБОРА С УЧЕТОМ СИНТАКСИЧЕСКИХ МАКРОСОВ

Рассмотрим организацию процедуры разбора с учетом вышеизложенных принципов для грамматики с синтаксическими макросами. В случае применения синтаксических макросов должны выполниться семантические действия, соответствующие синтаксическому разбору тела макроса правилом разбора макроса, с корректным вызовом семантических действий, соответствующих синтаксическим фрагментам в разбираемой строке, на которые указывают переменные-ссылки в теле макроса.

Для каждого макроса M (G, S, R) создается отдельный экземпляр синтаксического анализатора $P_{(M.S)}$, где через M.S обозначен текст замены S макроса M, для которого вызывается метод parse с параметрами:

$P_{(M.S)}$.parse(M.R, 0);

Целью этих предварительных действий является подготовка соответствующего экземпляра синтаксического анализатора для макроса M к собственно выполнению макроса M (очевидно, что для любого макроса разбор текста замены (parse) можно выполнить лишь единожды, в то время как выполнение макроса (execute) будет происходить столько раз, сколько раз макрос будет встречаться в исходной входной цепочке). Обозначим полученный в результате экземпляр анализатора за P_M.

На первом шаге разбора (parse) исходной входной цепочки символов в случае, если при разборе были задействованы макросы (грамматики макросов), для каждого встреченного макроса запоминаются позиция окончания данного макроса, а также позиции начала всех связанных (помеченных переменными) нетерминалов, встречающихся в грамматике макроса. На втором шаге разбора (execute) для каждого встреченного макроса M вызывается метод execute соответствующего анализатора макроса:

P_M.execute(M.R, 0);

При этом ссылки в тексте замены макроса заменяются вызовами соответствующих им помеченных нетерминалов в нужных позициях исходной входной цепочки. Рассмотрим суть работы алгоритма на примере приведенного выше макроса для описания цикла FOR (см. рис. 3).

Через P на рисунке обозначен экземпляр синтаксического анализатора для разбора исходной входной цепочки:

```
print("["); for(i=0; i<5; i++) print(i); print("]");
```

P_{FOR} – экземпляр синтаксического анализатора для разбора текста замены макроса для описания цикла FOR; для удобства текст замены из приводимого выше макроса переписан в одну строку:

```
{ $init; while($cond) { $body; $change; } }
```

Переменные $init, $cond, $body, $change связаны с нетерминалами expr, соответственно, при разборе текста замены макроса будут также разобраны как expr.

При разборе входной цепочки анализатором P фрагмент текста "for(i=0; i<5; i++) print(i)" будет разобран грамматикой макроса FOR. Т.о. этот фрагмент на 2-ом этапе разбора будет обрабатываться вызовом выполнения тела макроса FOR, т.е. вызовом метода execute у синтаксического анализатора P_{FOR} (P_{FOR}.execute(expr, 0)); на рисунке обозначен стрелкой с индексом 0. Далее указатель перемещается на позицию, следующую за фрагментом текста, обрабатываемом вызовом макроса; на рисунке – стрелка 0*.

При выполнении макроса FOR ссылки $init, $cond, $body, $change будут обрабатываться вызовом нетерминалов expr для соответствующих фрагментов ("i=0", "i<5", "i++", "print(i)") в исходной входной цепочке, т.е. у экземпляра синтаксического анализатора P (P.execute(expr, pos), где pos – соответствующая позиция); на рисунке – стрелки 1-4.

Результатом разбора рассматриваемой входной строки синтаксическим анализатором, полученным из рассматриваемой грамматики (с синтаксическим макросом FOR), является построение AST-дерева (рис. 2.).

Рассмотренный алгоритм будет корректно рекурсивно работать, если в теле макроса в свою очередь присутствуют фрагменты текста, разбираемые определенными ранее макросами, или же если связываемые в макросе фрагменты

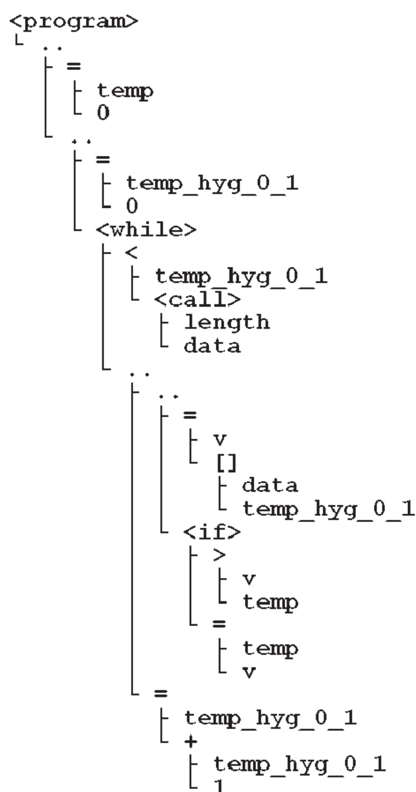

Рис. 2.

текста в свою очередь разбираются макросами (например, цикл FOR в цикле FOR).

4. «ГИГИЕНИЧЕСКИЕ МАКРОСЫ»

Часто возникает необходимость в использовании внутри макроса гарантированно уникальных имен идентификаторов в терминах описываемого языка относительно идентификаторов в тексте основной программы. Например, если в макросе используется идентификатор, имя которого совпадает с идентификатором основной программы, то первый должен быть автоматически переименован, т.е. идентификаторы из разных контекстов должны быть полностью независимы друг от друга. Такое свойство применительно к макросам называется «гигиеной».

В синтаксических макросах PEG-PG по умолчанию идентификаторы не являются “гигиеническими”. Если говорить совсем формально, то при разборе строк, в том числе макросов, невозможно сказать, что является идентификатором, а что нет, т.к. на уровне грамматики и алгоритма синтаксического анализа существуют только нетерминалы. Некоторые из нетерминалов, заданных грамматикой, могут распознавать во входной строке фрагменты, являющиеся идентификаторами в описываемом языке. Т.к. формально такие нетерминалы для конкретного языка отделить от остальных нетерминалов невозможно, их необходимо выделять явно. В PEG-PG для этого при описании нетерминала (правила) используется атрибут hygienic, например:

```
~identifier:
  #options(
    hygienic = true;
  )#
  // собственно описание нетерми-
нала
  letter ( letter / digit )*
;
```

Кроме того, чтобы конкретные идентификаторы в теле макроса сделать “гигиеническими”, их также надо объявлять особым образом, с определенным в грамматике языка суффиксом и префиксом. Например, цикл FOREACH применительно к массиву может быть описан следующим образом:

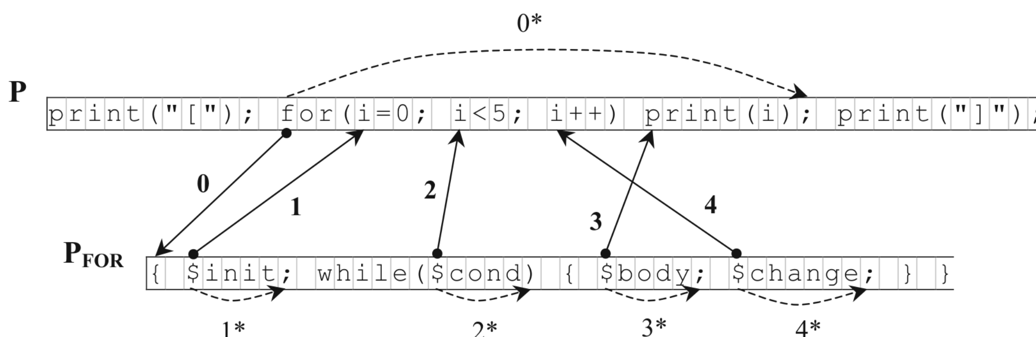

Рис. 1. Последовательность вызовов правил при разборе входной цепочки с макросом FOR

```
<AstNode> expr:
  $result:#macro(
    "foreach" '(' $var:identifier
':' $array:identifier ')'
      $body:expr
  <expr>:=
      for(!temp!=0;  !temp!
<length($array); !temp!=!temp!+1) {
      $var=$array[!temp!];
      $body;
    }
  )#
/
  $result:#prev
;
```

Каждый фрагмент "!temp!" будет на этапе применения данного макроса распознан в виде строки, получаемой при вызове функции hygie nicPreprocess("temp"), которая возвращает уникальный идентификатор для каждого применения макроса. Стандартная реализация функции hygienicPreprocess может быть переопределена в грамматике языка.

В результате при разборе строки

temp=0; foreach(v: data) if(v>0) temp=v;

с учетом предыдущего определения цикла FOR, синтаксическим анализатором будет построено следующее AST-дерево:

```
<program>
└ ..
  ├ =
  │ ├ temp
  │ └ 0
  └ ..
    ├ =
    │ ├ temp_hyg_0_1
    │ └ 0
    └ <while>
      ├ <
      │ ├ temp_hyg_0_1
      │ └ <call>
      │   ├ length
      │   └ data
      └ ..
        ├ ..
        │ ├ =
        │ │ ├ v
        │ │ └ []
        │ │   ├ data
        │ │   └ temp_hyg_0_1
        │ └ <if>
        │   ├ >
        │   │ ├ v
        │   │ └ temp
        │   └ =
        │     ├ temp
        │     └ v
        └ =
          ├ temp_hyg_0_1
          └ +
            ├ temp_hyg_0_1
            └ 1
```

Из приведенного AST-дерева легко видеть, что переменная temp основной программы не пересекается с переменной temp макроса, т.к. последняя в теле макроса описана как "гигиеническая". Если в основной программе в цикле foreach будет вложенный цикл foreach, то переменные temp для двух разных вызовов макроса также будут различными (например, temp_hyg_0_1 и temp_hyg_0_2).

5. СОЗДАНИЕ SOURCE-TO-SOURCE ТРАНСЛЯТОРА С ПОМОЩЬЮ PEG-PG

Синтаксические макросы в PEG-PG не только могут изменять порядок выполнения семантических действий, связанных с синтаксическими конструкциями, но также могут быть использованы для создания различных source-to-source-трансляторов. Например, для рассматриваемой входной строки с конструкцией foreach, в результате связывания начального правила грамматики со строковой переменной будет получен следующий текст:

```
temp=0;

    {
     temp_hyg_0_1=0;
  while(temp_hyg_0_1<length(data)){
        {
      v=data[temp_hyg_0_1];
      if(v>temp) temp=v;
     }
  ;
    temp_hyg_0_1=temp_hyg_0_1+1;
   }
  }
 ;
```

Результирующий текст программы намеренно приведен в том виде, в котором он получен при автоматическом преобразовании. "Нечитаемое" форматирование объясняется тем фактом, что применение макросов к тексту входной программы является исключительно текстовым

преобразованием (комбинацией исходного текста программы с текстом замены макросов), при котором не проводится какого-либо анализа структуры программы, построения AST-дерева с последующей генерации кода программы по такому дереву и т.п.

При этом полученный после применения макросов код полностью соответствует AST-дереву, рассмотренному выше. Т.о. с помощью синтаксических макросов могут быть описаны семантически корректные преобразования исходного кода программ.

ЗАКЛЮЧЕНИЕ

В данной статье рассмотрен механизм определения новых синтаксических конструкций и их семантики с помощью синтаксических макросов, реализованный в генераторе синтаксических анализаторов PEG-PG. Данный механизм в настоящий момент может быть использован при построении расширений языков, описанных с помощью грамматик PEG-PG. В отличие от классического применения макросов как средства дополнительной обработки текста перед его синтаксическим анализом, рассматриваемые синтаксические макросы являются встроенным механизмом в разбор текста на основе PEG-грамматик. При этом не нарушается линейная зависимость времени разбора от длины входной цепочки, характерная для PEG-грамматик. Поэтому данный механизм может быть использован и при построении саморасширяемых, в том числе интерпретируемых, языков программирования.

ЛИТЕРАТУРА

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.

3. *Соломатин Д.И.* Объектно-ориентированный подход к описанию и расширению синтаксиса языков программирования // Вестник Воронежского государственного университета. Серия «Системный анализ и информационные технологии». – Воронеж, 2008. – № 2. – С. 51-55.

4. *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

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

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

Соломатин Дмитрий Иванович – ассистент, кафедра Программирования и информационных технологий, Воронежский государственный университет. Тел.(4732)208-470, 8-905-658-1499.

Solomatin Dmitriy I. – assistant of dept. of the Programming and Information Technologies, Voronezh State University.Tel. .(4732)208-470, 8-905-658-1499.