БУ ВО Ханты-Мансийского автономного округа – Югры
«Сургутский государственный университет»

Теория языков программирования и методы трансляции

Часть 7

- Нисходящие методы грамматического анализа
- Алгоритм рекурсивного спуска

Гришмановский Павел Валерьевич
доцент кафедры автоматики и компьютерных систем, к.т.н., доцент

Сургут, 2018

Алгоритм рекурсивного спуска

Грамматика $G = (\mathbf{V}^T, \mathbf{V}^N, \mathbf{P}, S)$ называется **грамматикой рекурсивного спуска**, если для $\forall A \in \mathbf{V}^N$ множество ее правил содержит только правила вида

- **либо** $A \to \alpha$, где $\alpha \in \mathbf{V}^*$ и это **единственное** правило для нетерминального символа $A \in \mathbf{V}^N$
- **либо** $A \to a_1\beta_1 \mid a_2\beta_2 \mid \ldots \mid a_n\beta_n$, где $a_i \in \mathbf{V}^T$; $a_i \neq a_j$ при $i \neq j$ для $\forall i,j = 1,2\ldots n$; $\beta_i \in \mathbf{V}^*$

Достоинства:

- восстанавливает левосторонний вывод в прямом порядке
- на каждом шаге однозначно принимается решение
  - при чтении первого символа цепочки можно однозначно установить правило, использованное при выводе
  - при чтении каждого последующего символа он сопоставляется с терминальным символом правой части уже выбранного правила
- время грамматического разбора линейно зависит от длины входной цепочки
- грамматика является однозначной

Недостатки:

- не покрывает всех контекстно-свободных грамматик
- не допускается левая рекурсия

Алгоритм рекурсивного спуска

Построение распознавателя

1. Для каждого нетерминального символа A $\in$ $\mathbf{V}^N$ строится своя процедура разбора *ProcA* (распознавание всех цепочек, выводимых из символа A)
2. Первый символ входного потока, анализируемый каждой процедурой, должен быть считан до вызова этой процедуры (т.е. первый символ строки, выводимой из соответствующего нетерминального символа)
3. При завершении процедуры разбора, ею должен быть считан один символ входного потока, следующий за распознанной цепочкой (т.е. не принадлежащий цепочке, выводимой из него)
4. Алгоритм каждой процедуры разбора строится в соответствии с правыми частями правил (всех правил для соответствующего нетерминального символа)

Алгоритм рекурсивного спуска

Алгоритм процедуры разбора (для каждого нетерминального символа)

- первые символы правых частей всех правил (если их несколько) сравниваются с первым символом, считанным из входного потока и выбирается соответствующее правило (если совпадений нет, то генерируется ошибка), затем считывается очередной входной символ
- если очередной символ правой части правила (в т.ч. первый символ правой части единственного правила) является терминальным, то он сравнивается с текущим считанным символом (при несовпадении генерируется ошибка) и считывается следующий входной символ
- если очередной символ правой части правила (в т.ч. первый символ правой части единственного правила) является нетерминальным, то вызывается соответствующая процедура разбора (текущий считанный символ будет первым символом, анализируемым в этой процедуре, а после завершения вызванной процедуры текущий символ уже будет считан из входного потока)

Алгоритм рекурсивного спуска

Расширение применимости метода рекурсивного спуска

- преобразование к приведенной форме
  - устранение цепных и λ-правил (метод нечувствителен к цепным и λ-правилам, только **если они являются единственными** правилами для некоторого нетерминального символа)
- устранение левой рекурсии (обязательно!)
- преобразование к нормальной форме Грейбах
- факторизация (неповторяющиеся первые символы)
- подстановка правых частей правил вместо нетерминальных символов
- построение конечных автоматов для распознавания лексем (построение сканера – разделение лексического и синтаксического анализа)
- запись правил с использованием метасимволов
- использование регулярных выражений
- эвристический анализ (построение эквивалентных алгоритмов)
- модификация алгоритма построения процедуры разбора («заглядывание вперед» и др.)

Алгоритм рекурсивного спуска (пример)

Множество правил **P** грамматики:

S → I=E;	оператор присваивания
E → E+T | E-T | T	арифметические операции + и -
T → T*M | T/M | M	арифметические операции * и /
M → (E) | -M | I | C	выражение в скобках, унарный минус, переменная, константа
I → A | AK	идентификаторы (переменные)
K → A | AK | D | DK	
C → D | CD	константы (целочисленные значения)
A → a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | | v | w | x | y | z | _	
D → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9	

- Грамматика однозначна
- В грамматике присутствуют:
  - Левая рекурсия
  - Цепные правила

Алгоритм рекурсивного спуска (пример)

Правила для S:

S → I=E;

- существует единственное правило для S – выбор однозначен

```
void ProcS(void)
{                                   // первый символ уже считан!
  long * p = ProcL();               // левый операнд I - переменная
  if (c != '=')                     // затем '='
    Error("'=' missing.", 0);
  Get();                            // затем находим значение
  *p = ProcE();                     // выражения E - правый операнд
  if (c != ';')                     // затем ';'
    Error("';' missing.", 0);
  Get();                            // считывание следующего символа
}
// NB: Функция Error завершает программу!
```

Алгоритм рекурсивного спуска (пример)

Правила для E:

E → E+T | E-T | T

- невозможен выбор правила
- левая рекурсия

Устранение левой рекурсии:

E → TE' | T

E' → +T | -T | +TE' | -TE'

- нарушение порядка вычислений (справа налево)

```
long ProcE(void)
{
  long x = ProcT();                  // первый операнд T
  while (c == '+' || c == '-')       // замена рекурсии циклом
  {
    char p = c;
    Get();
    if (p == '+')
      x += ProcT();                  // последующие операнды
    else
      x -= ProcT();
  }
  return x;
}
```

Алгоритм рекурсивного спуска (пример)

Правила для M:

M → (E) | -M | I | C

- невозможен выбор правила

Устранение цепных правил (подстановка правых частей правил для I и C) влечет

- увеличение количества правил
- формирование одного значения распределено между процедурами

```
long ProcM(void)
{
  long x;

  if (c == '(')                              // выбор правила (E)
  {
    Get();
    x = ProcE();
    if (c != ')') Error("\')\' missing.", 0);
    Get();
  }
  else if (c == '-')                         // выбор правила -M
    {
      Get();
      x = -ProcM();
    }
    else if (c >= '0' && c <= '9')           // «заглядывание вперед»
        x = ProcC();                         // и вызов процедур разбора лексем
      else if ((c >= 'a' && c <= 'z') || c == '_')  // БЕЗ предварительного Get()
          x = ProcI();
        else Error("Syntax error.", 0);
  return x;
}
```