Лабораторная работа № 5

Синтаксический анализатор операторного предшествования

1. Цель и задачи работы

Цель работы: приобретение практических навыков реализации таблично управляемых синтаксических анализаторов на примере анализатора операторного предшествования.

Задачи работы:

1. Ознакомиться с основными понятиями и определениями, лежащими в основе синтаксического анализа операторного предшествования.
2. Изучить алгоритм синтаксического анализа операторного предшествования.
3. Разработать, тестировать и отладить программу синтаксического анализа в соответствии с предложенным вариантом грамматики.
4. Включить в программу синтаксического анализ семантические действия для реализации синтаксически управляемого перевода инфиксного выражения в обратную польскую нотацию.

2. Материал для изучения и ознакомления

Перед выполнением работы рекомендуется ознакомиться со следующими материалами:

- Wirth–Weber precedence relationship. URL: http://en.wikipedia.org/wiki/Wirth-Weber_precedence_relationship. (Дата обращения: 05.05.2014).
- Operator-precedence grammar. URL: http://en.wikipedia.org/wiki/Operator-precedence_grammar. (Дата обращения: 05.05.2014).
- Operator-precedence parser. URL: http://en.wikipedia.org/wiki/Operator-precedence_parser. (Дата обращения: 05.05.2014).

3. Теоретическая часть

Метод операторного предшествования относится к классу восходящих таблично-управляемых методов синтаксического анализа на основе алгоритма типа «перенос/свертка». Этот метод основан на отношениях предшествования Вирта-Вебера, но только между терминальными символами грамматики.

Определение 1. Операторной грамматикой называется приведенная КС-грамматика без ε-правил, в которой правые части правил не содержат смежных нетерминалов.

Определение 2. Для операторной грамматики отношения предшествования можно задать на множестве терминалов плюс символ \$, игнорируя нетерминалы. Пусть $G = (N, \Sigma, P, S)$ – операторная грамматика и \$ – новый символ. Зададим отношения операторного предшествования на множестве $\Sigma \cup \{\$\}$:

1) $\boldsymbol{a}=\boldsymbol{b}$, если $A \to \alpha\boldsymbol{a}\gamma\boldsymbol{b}\beta \in P$ и $\gamma \in N \cup \{\varepsilon\}$,
2) $\boldsymbol{a}<\bullet\boldsymbol{b}$, если $A \to \alpha\boldsymbol{a}B\beta \in P$ и $B \Rightarrow^+ \gamma\boldsymbol{b}\delta$, где $\gamma \in N \cup \{\varepsilon\}$,
3) $\boldsymbol{a}\bullet>\boldsymbol{b}$, если $A \to \alpha B\boldsymbol{b}\beta \in P$ и $B \Rightarrow^+ \delta\boldsymbol{a}\gamma$, где $\gamma \in N \cup \{\varepsilon\}$,
4) $\$<\bullet\boldsymbol{a}$, если $S \Rightarrow^+ \gamma\boldsymbol{a}\delta$ и $\gamma \in N \cup \{\varepsilon\}$,
5) $\boldsymbol{a}\bullet>\$$, если $S \Rightarrow^+ \alpha\boldsymbol{a}\gamma$ и $\gamma \in N \cup \{\varepsilon\}$.

Определение 3. Операторная грамматика G называется грамматикой операторного предшествования, если между любыми двумя терминальными символами выполняется не более одного отношения операторного предшествования

Пример 1. Примером грамматики операторного предшествования служит грамматика G_0 с правилами

$$E \to E + T \mid T \quad :1, 2$$
$$T \to T * F \mid F \quad :3, 4$$
$$F \to (E) \mid \boldsymbol{a} \quad :5, 6$$

Матрица отношений операторного предшествования для этой грамматики, будет иметь вид

	(	***a***	*	+	)	$
)			•>	•>	•>	•>
a			•>	•>	•>	•>
*	<•	<•	•>	•>	•>	•>
+	<•	<•	<•	•>	•>	•>
(	<•	<•	<•	<•	=	
$	<•	<•	<•	<•		

Пустая клетка матрицы интерпретируется как **ошибка**.

Идея синтаксического анализа для грамматик операторного предшествования вытекает из теоремы, которая приводится без доказательства.

Теорема. Пусть $G = (N, \Sigma, P, S)$ – операторная грамматика и $\$S\$ \Rightarrow^* \alpha A w \Rightarrow \alpha\beta w$. Тогда

1) *отношение операторного предшествования* $<\bullet$ *или* $=$ *выполняется между последовательными терминалами (и символом* \$*) цепочки* α*;*
2) *отношение* $<\bullet$ *выполняется между самым правым терминалом цепочки* α *и самым левым терминалом цепочки* β*;*
3) *отношение* $=$ *выполняется между последовательными терминалами цепочки* β*;*
4) *отношение* $\bullet>$ *выполняется между самым правым терминалом цепочки* β *и первым символом цепочки* w*.*

С помощью алгоритма разбора типа «перенос-свертка» легко выделить терминальные символы, входящие в основу β. Однако возникают проблемы в связи с нетерминальными символами, поскольку на них не определены отношения операторного предшествования. Тем не менее, тот факт, исходная грамматика является операторной грамматикой, позволяет строить «остовный» правый разбор.

Определение 4. Пусть $G = (N, \Sigma, P, S)$ – операторная грамматика. Остовной грамматикой для G назовем грамматику $G_s = (\{S\}, \Sigma, P', S)$, содержащую каждое правило $S \to X_1X_2 \dots X_m$, для
которого в P найдется такое правило $A \to Y_1Y_2 \dots Y_m$, что для $1 \le i \le m$

1) $X_i = Y_i$, если $Y_i \in \Sigma$,
2) $X_i = S$, если $Y_i \in N$.

Но в P' не должно быть правила $S \to S$.

Следует обратить внимание на то, что $L(G) \subseteq L(G_s)$ и, вообще говоря, $L(G_s)$ может содержать цепочки, не принадлежащие $L(G)$. Теперь можно описать алгоритм типа «перенос-свертка» для грамматик операторного предшествования.

Алгоритм построения анализатора операторного предшествования

Вход. Грамматика операторного предшествования $G = (N, \Sigma, P, S)$.
Выход. Алгоритм разбора $A = (f, g)$ типа «перенос-свертка» для грамматики G_s.
Метод. Пусть β обозначает S или ε.

1) $f(\boldsymbol{a}\beta, \boldsymbol{b}) =$ **перенос**, если $\boldsymbol{a}<\bullet\boldsymbol{b}$ или $\boldsymbol{a}=\boldsymbol{b}$.
2) $f(\boldsymbol{a}\beta, \boldsymbol{b}) =$ **свертка**, если $\boldsymbol{a}\bullet>\boldsymbol{b}$.
3) $f(\$S, \$) =$ **допуск**.
4) $f(\alpha, w)$ – **ошибка** в остальных случаях.
5) $g(\boldsymbol{a}\beta\boldsymbol{b}\gamma, \boldsymbol{w}) = i$, если
   a) β – это S или ε,
   b) $\boldsymbol{a}<\bullet\boldsymbol{b}$,
   c) отношение $=$ выполняется между последовательными терминальными символами цепочки γ, если они существуют,

d) $S \to \beta \boldsymbol{b} \gamma$ – правило с номером i грамматики G_s.

6) $g(\alpha, w)$ - **ошибка** в остальных случаях.

Пример 2. Выполнить разбор цепочки $(\boldsymbol{a}+\boldsymbol{a})^*\boldsymbol{a}$ в соответствии с отношениями операторного предшествования для грамматики G_0. Остовной грамматикой для G_0 будет грамматика G_{0s} с правилами $E \to E + E \mid E * E \mid (E) \mid \boldsymbol{a}$, полученная из G_0 заменой всех нетерминалов символом E и устранением всех цепных правил. (Заметим, что в операторной грамматике правилом, не содержащим терминалов в правой части, может быть только цепное правило.) Грамматика G_{0s}, очевидно, не однозначная, но отношения операторного предшествования гарантируют единственность искомого разбора. Алгоритм разбора A для грамматики G задается определяемыми ниже функциями f и g. Цепочки, которые служат аргументами этих функций, будут состоять только из терминалов грамматики G_0 и символов \$ и E. Далее, γ обозначает E или пустую цепочку, $\boldsymbol{b}$ и $\boldsymbol{c}$ - терминалы или \$.

$f(\boldsymbol{b}\gamma, \boldsymbol{c})$ = если $\boldsymbol{b} \lessdot \bullet \boldsymbol{c}$ или $\boldsymbol{b} = \boldsymbol{c}$, то **перенос**,
иначе если $\boldsymbol{b} \bullet \gtrdot \boldsymbol{c}$, то **свертка**,
иначе если $\boldsymbol{b}$ = \$, $\gamma = E$ и $\boldsymbol{c}$ = \$, то **допуск**,
иначе **ошибка**.

$g(\boldsymbol{b}\gamma\boldsymbol{a}, x) = 6$, если $\boldsymbol{b} \lessdot \bullet \boldsymbol{a}$,
$g(\boldsymbol{b}E^*E, x) = 3$, если $\boldsymbol{b} \lessdot \bullet *$,
$g(\boldsymbol{b}E+E, x) = 1$, если $\boldsymbol{b} \lessdot \bullet +$,
$g(\boldsymbol{b}\gamma(E), x) = 5$, если $\boldsymbol{b} \lessdot \bullet ($,
$g(\boldsymbol{a}, x)$ = **ошибка** в остальных случаях.

Таким образом, для входной цепочки $(\boldsymbol{a}+\boldsymbol{a})^*\boldsymbol{a}$ алгоритм сделает такую последовательность шагов:

$[\$, (\boldsymbol{a}+\boldsymbol{a})^*\boldsymbol{a}\$, \varepsilon] \vdash^s$
$[\$(, \boldsymbol{a}+\boldsymbol{a})^*\boldsymbol{a}\$, \varepsilon] \vdash^s$
$[\$(\boldsymbol{a}, +\boldsymbol{a})^*\boldsymbol{a}\ \$, \varepsilon] \vdash^r$
$[\$(E, +\boldsymbol{a})^*\boldsymbol{a}\$, 6] \vdash^s$
$[\$(E+, \boldsymbol{a})^*\boldsymbol{a}\$, 6] \vdash^s$
$[\$(E+\boldsymbol{a},)^*\boldsymbol{a}\$, 6] \vdash^r$
$[\$(E+E,)^*\boldsymbol{a}\$, 66] \vdash^r$
$[\$(E,)^*\boldsymbol{a}\$, 661] \vdash^s$
$[\$(E), *\boldsymbol{a}\$, 661] \vdash^r$
$[\$E, *\boldsymbol{a}\$, 6615] \vdash^s$
$[\$E^*, \boldsymbol{a}\$, 6615] \vdash^s$
$[\$E^*\boldsymbol{a}, \$, 6615] \vdash^r$
$[\$E^*E, \$, 66156] \vdash^r$
$[\$E, \$, 661563] \vdash$
допуск

Можно убедиться в том, что 661563 действительно остовный правый разбор цепочки $(\boldsymbol{a}+\boldsymbol{a})^*\boldsymbol{a}$ в грамматике G_{0s}. Этот остовный разбор цепочки $(\boldsymbol{a}+\boldsymbol{a})^*\boldsymbol{a}$ можно представить в виде дерева.

Пример 2 представляет собой частный случай метода, работающего для многих грамматик, особенно для тех, которые определяют языки, являющиеся множествами арифметических выражений. Этот метод включает построение новой грамматики, получаемой из старой грамматики заменой всех нетерминалов одним нетерминалом и устранением цепных правил. Для грамматики операторного предшествования всегда можно с помощью алгоритма типа «перенос-свертка» найти один разбор для каждой входной цепочки.

4. Практическая часть

4.1. Эвристический метод нахождения отношений операторного предшествования

Для грамматик математических выражений, наподобие G_0, можно использовать следующий эвристический метод для создания корректного множества отношений операторного предшествования.

1. Если оператор θ_1 имеет более высокий приоритет, чем оператор θ_2, определим, что $\theta_1 \bullet\!> \theta_2$ и $\theta_2 <\!\bullet\ \theta_1$. Например, если оператор * имеет более высокий приоритет, чем оператор +, то * •> + и + <• *. Эти отношения гарантируют, что при получении выражения вида $E + E * E + E$ основой является $E * E$, и именно $E * E$ будет свернуто первым.
2. Если θ_1 и θ_2 представляют собой операторы равного приоритета (это может быть один и тот же оператор), то устанавливаем, что $\theta_1 \bullet\!> \theta_2$ и $\theta_2 \bullet\!> \theta_1$, если операторы лево-ассоциативны, и $\theta_1 <\!\bullet\ \theta_2$ и $\theta_2 <\!\bullet\ \theta_1$, в случае если операторы право-ассоциативны. Например, для лево-ассоциативных операторов + и – устанавливаем, что + •> –, – •> – и – •> +. Для право-ассоциативного оператора возведения в степень ^ (такого оператора в грамматике G_0 нет, но его можно включить с наивысшим приоритетом) следует принять, что ^ <• ^. Такие отношения гарантируют, что в выражении $E - E + E$ основой является $E - E$, а в выражении $E \wedge E \wedge E$ – основой является последнее подвыражение $E \wedge E$.
3. Для всех операторов θ определяем отношения θ <• ***a*** , ***a*** > θ , θ <• (, (<•θ ,) > θ , θ •>), θ •> $ и $ <• θ. Кроме того, считаем, что (=), $ <• (, $ <• ***a***, (<• (, ***a*** •> $,) •> $, (<• ***a***, ***a*** •>),) •>). Эти правила гарантируют, что и ***a***, и (*E*) будут свернуты *E*. Кроме того, $ служит как левым, так и правым маркером конца строки, что заставляет основы находиться между ними.
4. Если грамматика содержит унарный префиксный оператор (например, логическое отрицание ~ для логических выражений), который не имеет бинарного аналога, то θ <• ~ для любого оператора θ, независимо от того, унарный он или бинарный. Кроме того, ~ •> θ, если ~ имеет более высокий приоритет, чем θ, и ~ <• θ в противном случае. Например, если ~ имеет более высокий приоритет, чем логическое умножение &, и & - лево-ассоциативный оператор, то в соответствии с приведенными правилами выражение *E*&~*E*&*E* должно быть сгруппировано как (*E*&(~*E*))&*E*. Правила для постфиксного унарного оператора аналогичны рассмотренным правилам. Ситуация изменяется при рассмотрении операторов типа знака «минус», который может использоваться и как унарный префиксный оператор, и как бинарный инфиксный. Наилучшим решением этой проблемы было бы распознавание типа оператора лексическим анализатором и использование им разных токенов для бинарного и унарного минусов.

4.2. Обработка и нейтрализация ошибок при синтаксическом анализе операторного предшествования

Для каждой пустой клетки матрицы отношений операторного предшествования необходимо определить подпрограмму обработки и нейтрализации ошибок; одна и та же подпрограмма может использоваться в нескольких местах. Для матрицы отношений операторного предшествования из примера 1 можно предложить следующие подпрограммы.

Подпрограмма e_1 Причина ошибки: отсутствует выражение в целом. Действие анализатора: вставить операнд во входной поток. Сообщение об ошибке: пропущен операнд.
Подпрограмма e_2 Причина ошибки: выражение начинается с правой скобки. Действие анализатора: удалить правую скобку из входного потока. Сообщение об ошибке: несбалансированная правая скобка.
Подпрограмма e_3 Причина ошибки: операнд или правая скобка следует за операндом или левой скобкой. Действие анализатора: вставит знак операции (например, +) во входной поток. Сообщение об ошибке: пропущен оператор.
Подпрограмма e_4 Причина ошибки: выражение завершается левой скобкой. Действие анализатора: вытолкнуть левую скобку из стека. Сообщение об ошибке: пропущена правая скобка.

При таком механизме обработки и нейтрализации ошибок матрица отношений операторного предшествования из примера 1 примет следующий вид

	(	***a***	*	+	)	$
)	e_3	e_3	•>	•>	•>	•>
a	e_3	e_3	•>	•>	•>	•>
*	<•	<•	•>	•>	•>	•>
+	<•	<•	<•	•>	•>	•>
(	<•	<•	<•	<•	=	e_4
$	<•	<•	<•	<•	e_2	e_1

Возможны и другие ошибки, которые возникают тогда, когда основа обнаружена, но не существует правила вывода с правой частью, соответствующей основе. Обработка таких ошибок требует творческого подхода.

4.3. Пример выполнения лабораторной работы

Рассматривается грамматика логических выражений, правила вывода которой в BNF имеют вид:

<высказывание> ::= <импликация>
| <высказывание> "=" <импликация>
<импликация> ::= <дизъюнкция>
| <импликация> ">" <дизъюнкция>
<дизъюнкция> ::= <конъюнкция>
| <дизъюнкция> "|" <конъюнкция>
<конъюнкция> ::= <множитель>
| <конъюнкция> "&"<множитель>
<множитель> ::= "~"<множитель>
| "("<высказывание> ")"
| <атом>
<атом> ::= ["A"-"Z", "a"-"z", "0", "1"]

Последовательность операции "=", "> ", "|", "&" выполняется слева направо, последовательность операций "~" выполняется справа налево.
Старшинство операций определяется следующим списком: "~" (имеет наивысшее старшинство), "&", "|", ">", "=".

Программа синтаксического анализатора операторного предшествования может иметь такой вид:

```
PROGRAM OperatorPrecedencePparsing;
// Прототип этой программы создан еще в 1988 году

{$APPTYPE CONSOLE}

uses
  SysUtils;

CONST
  marker = '$';
  blank = ' ';
  max = 1000;
  error_msg : array['1'..'4'] of string = (
    'отсутствует операнд.',
    'несбалансированная правая скобка.',
    'отсутствует оператор.',
    'отсутствует правая скобка.' );
TYPE
// Терминальные символы грамматики
  symbol = (_atom, _not, _and, _or, _imp, _equ, _lpar, _rpar, _doll);
VAR
// Матрица отношений операторного предшествования
  matrix : ARRAY[symbol, symbol] OF char = (
//     |    atm not and or  imp equ lp  rp  dol
// ----+-----------------------------------
{ atm  | } ('3','>','>','>','>','>','3','>','>'),
{ not  | } ('<','<','>','>','>','>','<','>','>'),
{ and  | } ('<','<','>','>','>','>','<','>','>'),
{ or   | } ('<','<','<','>','>','>','<','>','>'),
{ imp  | } ('<','<','<','<','>','>','<','>','>'),
{ equ  | } ('<','<','<','<','<','>','<','>','>'),
{ lp   | } ('<','<','<','<','<','<','<','=','4'),
{ rp   | } ('3','>','>','>','>','>','3','>','>'),
{ dol  | } ('<','<','<','<','<','<','<','2','1') );
      s : ARRAY[0..max] OF char; // Стек разбора
      t : integer; // Указатель (индекс) вершины стека
      symbols : ARRAY[char] OF symbol;
      ch : char;
      flag, done : boolean;
```

```
      i, n : integer;
      prn : ARRAY[1..max] OF char;

PROCEDURE getch;
BEGIN
  REPEAT
    read(ch)
  UNTIL ch > blank;
END; // OF PROCEDURE 'getch'

PROCEDURE gen_prn(c: char);
BEGIN
  IF NOT (c IN ['(',')']) THEN
  BEGIN
    inc(n);
    prn[n] := c
  END
END; // OF PROCEDURE 'gen_prn'

BEGIN
// Инициализация матрицы операторного предшествования 'matrix'
  FOR ch := chr(0) TO chr(255) DO
    IF ch IN ['A'..'Z','a'..'z','0','1'] THEN symbols[ch] := _atom
    ELSE
      CASE ch OF
       '~' : symbols[ch] := _not;
       '&' : symbols[ch] := _and;
       '|' : symbols[ch] := _or;
       '>' : symbols[ch] := _imp;
      '=' : symbols[ch] := _equ;
       '(' : symbols[ch] := _lpar;
       ')' : symbols[ch] := _rpar;
       '$' : symbols[ch] := _doll;
       ELSE // Иначе
         symbols[ch] := _doll;
      END; // CASE

  done := false;
  flag := false;
  REPEAT
    writeln('ВВЕДИТЕ ИНФИКСНОЕ ВЫРАЖЕНИЕ(в конце выражения ''$''; только    ''$'' - выход)');
    n := 0;
    getch;
    IF ch = marker THEN done := true
    ELSE
    BEGIN

//////////////////////////////////////////////////////////
// Начало 'operator precedence parsing algorithm'
      s[0] := marker;
      t := 0;
      WHILE (t > 0) OR (ch <> marker) DO
      BEGIN
        CASE matrix[symbols[s[t]], symbols[ch]] OF
        '<', '=' :
        BEGIN // Перенос
          inc(t);
          s[t] := ch;
          getch;
        END;
        '>' :
        BEGIN // Свертка
          REPEAT
```

```
            gen_prn(s[t]);
            dec(t);
          UNTIL matrix[symbols[s[t]], symbols[s[t+1]]] = '<'
        END;
        ELSE // Иначе
        BEGIN
          writeln('ОШИБКА: ' + error_msg[matrix[symbols[s[t]], symbols[ch]]]);
          readln;
          flag := true;
          break; // Реализация панического анализатора
        END;
        END; // CASE
      END; // WHILE
// Конец 'operator precedence parsing algorithm'
////////////////////////////////////////////////////////

      IF flag THEN flag := false
      ELSE
      BEGIN
        write('ПОСТФИКСНАЯ ЗАПИСЬ: ');
        FOR i := 1 TO n DO write(prn[i]);
        writeln
      END; // ELSE
    END; // ELSE
  UNTIL done;
  readln;
END. // OF PROGRAM OperatorPrecedenceParser
```

Недостатком программы является то, что в ней обнаруживаются не все синтаксические ошибки. Достоинством программы является то, что в процессе синтаксического анализа выполняется преобразование инфиксного логического выражения в обратную польскую нотацию, т. е. реализуется синтаксически управляемый перевод.

5. Варианты заданий на лабораторную работу

Реализовать синтаксический анализатор операторного предшествования и синтаксически управляемый перевод инфиксного выражения в обратную польскую нотацию для грамматики выражений из лабораторной работы № 4.

6. Порядок выполнения работы

1. Ознакомиться с основными понятиями и определениями, лежащими в основе синтаксического анализа операторного предшествования.
2. Изучить алгоритм синтаксического анализа операторного предшествования.
3. Подготовить матрицу отношений операторного предшествования для предложенной грамматики.
4. Сформулировать синтаксически управляемые определения для перевода инфиксного выражения в обратную польскую нотацию.
5. Разработать, тестировать и отладить программу синтаксического анализа для предложенной грамматики.
6. Подготовить отчет о проделанной работе.
7. Подготовить ответы на контрольные вопросы.

7. Требования к отчету

Отчет по лабораторной работе выполняется в электронном виде и должен включать:

1. Идентификатор группы, имя и фамилию студента, дату выполнения работы.
2. Название лабораторной работы.
3. Описание задания – постановку задач, подлежащих выполнению в процессе лабораторной работы.
4. Текст программы, в которой решаются поставленные задачи.
5. Набор тестов и ожидаемые результаты для проверки правильности программы.
6. Результаты выполнения программы.
7. Анализ результатов и краткие выводы по работе.

8. Список дополнительной использованной литературы или дополнительных использованных электронных ресурсов.

8. Контрольные вопросы

1. Что такое операторная грамматика?
2. Что такое грамматика операторного предшествования?
3. Как определяются отношения операторного предшествования?
4. Как выделяется основа в процессе синтаксического разбора операторного предшествования?
5. Какие виды синтаксически ошибок не обнаруживаются в предложенном примере?
6. Какие действия надо предпринять для обнаружения всех синтаксических ошибок в предложенном примере?
7. Как сформулировать синтаксически управляемые определения для перевода инфиксного выражения в последовательность команд стековой машины?
8. Как сформулировать синтаксически управляемые определения для перевода инфиксного выражения в абстрактное синтаксическое дерево?

9. Рекомендуемая литература

1. Ахо А., Ульман Дж., Сети Р. Компиляторы: принципы, технологии и инструменты. – М.: Вильямс, 2001. [Параграф 4.6. Синтаксический анализ приоритета операторов]
2. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции: В 2-х томах. Т.1.: Синтаксический анализ. – М.: Мир, 1978. [Пункт 5.4.3. Грамматики операторного предшествования]
3. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции: В 2-х томах. Т.2.: Компиляция. – М.: Мир, 1978. [Пункт 9.2.1. Простые синтаксически управляемые переводы]