Классическая теория компиляторов

Лекция 4

ОПЕРАТОРНЫЕ ГРАММАТИКИ

Грамматики простого предшествия

Пусть дана входная цепочка символов "….RS…".

Вопрос: над какой частью цепочки сначала производить операции.

Понятие приоритета, основанное на системе отношений между символами:

Для любой цепочки "….RS…" возможны следующие варианты:

1. Если есть правило, заканчивающееся на R,

$$U \rightarrow \ldots R \quad ,$$

тогда можно сказать, что R > S.

2. Если есть правило, включающее в себя RS,

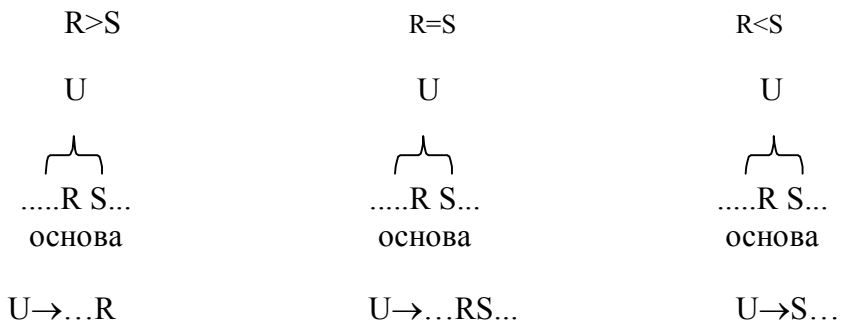
$$U \rightarrow \ldots RS \ldots \quad ,$$

тогда можно сказать, что R = S.

3. Если же есть правило, начинающееся на S,

$$U \rightarrow S \ldots \quad ,$$

тогда можно сказать, что R < S.



Алгоритм разбора

Стек операторов OP и стек аргументов ARG.

S - верхний символ в стеке операторов OP,

R – входной символ.

1. Если R – идентификатор, то поместить его в ARG и пропустить шаги 2,3.
2. Если f(S)<g(R), то поместить R в OP и взять следующий символ.
3. Если f(S)≥g(R), то вызвать семантическую процедуру, определяемую символом S. Эта процедура выполняет семантическую обработку, например, исключает из ARG связанные с S аргументы и заносит в ARG результат операции. Это – т.н. *редукция сентенциальной формы*.

Для грамматики

- $E \rightarrow E+T \mid T$
- $T \rightarrow T*F \mid F$
- $F \rightarrow (E) \mid i$

Строим матрицу, строки которой будут соответствовать f(S), а столбцы – g(R).

Стек F(S) \ g(R)	+	*	(	)	#
+	>	<	<	>	>
*	>	>	<	>	>
(	<	<	<	=	X
)	>	>	X	>	>
#	<	<	<	<	>

Пример разбора

(a*b/c+d)*e#

Стек F(S) \ g(R)	+	*	(	)	#
+	>	<	<	>	>
*	>	>	<	>	>
(	<	<	<	=	X
)	>	>	X	>	>
#	<	<	<	<	>

S	R	ω	Arg	f(S)?g(R)
#		(a*b/c+d)*e#		
#	(	a*b/c+d)*e#		<
#(	a	*b/c+d)*e#	a	<
#(	*	b/c+d)*e#	a	<
#(*	b	/c+d)*e#	a, b	<
#(*	/	c+d)*e#	a, b	>
#(	/	c+d)*e#	(a*b)	<
#(/	c	+d)*e#	(a*b), c	<
#(/	+	d)*e#	(a*b), c	>
#(	+	d)*e#	((a*b)/c)	<
#(+	d	)*e#	((a*b)/c), d	<
#(+	)	*e#	((a*b)/c), d	>
#(	)	*e#	(((a*b)/c)+d)	=
#	*	e#	(((a*b)/c)+d)	<
#*	e	#	(((a*b)/c)+d), e	<
#*	#		(((a*b)/c)+d), e	>
#	#		((((a*b)/c)+d)*e)	>

Алгоритм Дейкстры

Дано:

- $E \rightarrow E + T \mid E - T$ $\qquad E \rightarrow T$
- $T \rightarrow T * F \mid T / F$ $\qquad T \rightarrow F$
- $F \rightarrow F \wedge P$ $\qquad F \rightarrow P$
- $P \rightarrow (E)$ $\qquad P \rightarrow a$

Каждой операции ставится в соответствие некоторый *приоритет*:

P(+, -)=2, P(*, /)=3, P(^) =4

АЛГОРИТМ

- Проверяется очередной символ во входной строке.
- Если это операнд, то он передается в выходную строку.
- Если это открывающая скобка, то она заносится в стек с приоритетом нуль.
- Если это операция, то ее приоритет сравнивается с приоритетом операции, находящейся на вершине стека. Если приоритет новой операции больше, то эта новая операция заносится в стек. В противном случае берется операция с вершины стека и помещается в выходную строку, после этого повторяется сравнение с новыми верхними элементами стека до тех пор, пока на вершине стека не окажется операция с приоритетом, меньшим, чем у текущей операции, или пока стек не станет пустым. После этого текущая операция заносится в стек.
- Если текущий символ во входной строке является закрывающей скобкой, то операции из стека последовательно переносятся в выходную строку до тех пор, пока на вершине стека не появится открывающая скобка; эта открывающая скобка отбрасывается.
- Если выражение закончилось, то из стека последовательно переносятся в выходную строку все оставшиеся в нем операции.

МАТРИЦЫ ПЕРЕХОДОВ

Грамматики высокоуровневых конструкций

- прогр ::= инстр
- инстр ::= инстр; инстр
- инстр ::= перем := выр
- инстр ::= if выр then инстр else инстр end
- инстр ::= while выр do инстр end
- выр ::= выр ±перем | перем
- перем ::= i

```
d := 10;
a := b+c-d;
if a then d := 1 else
    while d do
        e := e-1
    end
end
```

Алгоритм. Исходные данные

<прог> ::= <инстр>

<инстр> ::= IF <выр> THEN <инстр>

<инстр> ::= <перем> := <выр>

<выр> ::= <выр> + <перем> | <перем>

<перем> ::= i

- *стек, переменная, хранящая текущий считываемый символ R, и переменная U, в которой будет храниться значение последней приведенной формы.*

1. *Стек хранит **головы правил** - правые части правил грамматики, заканчивающиеся терминалом.*
2. *Строки матрицы переходов соответствуют тем головам, которые могут появиться в стеке*
3. *Столбцы соотносятся с терминальными символами, включая #.*
4. *Элементами матрицы будут номера или адреса подпрограмм.*

	#	IF	THEN	:=	+	i
#	5	1	0	6	0	1
IF	0	0	9	0	3	1
IF <выр> THEN	7	1	0	6	0	1
<перем> :=	8	0	0	0	3	1
<выр> +	4	0	4	0	4	1
i	2	0	2	2	2	0

Подпрограммы

1. *IF U≠'' THEN ERROR(1);*
   PUSH(R);
   SCAN().
2. *IF U≠'' THEN ERROR(2);*
   POP();
   U := '<перем>'.
3. *IF U≠'<выр>' AND U≠'<перем>' THEN ERROR(3);*
   PUSH('<выр>+')
   U := '';
   SCAN().
4. *IF U≠'<перем>' THEN ERROR(4);*
   POP();
   U := '<выр>'.
5. *IF U≠'<прог>' AND U≠'<инстр>' THEN*
   ERROR(5);
   STOP().
6. *IF U≠'<перем>' THEN ERROR(6);*
   PUSH('<перем>:=')
   U := '';
   SCAN().
7. *IF U≠'<инстр>' THEN ERROR(7);*
   POP();
   U := '<инстр>'.
8. *IF U≠'<выр>' AND U≠'<перем>' THEN ERROR(8);*
   POP();
   U := '<инстр>'.
9. *IF U≠'<выр>' AND U≠'<перем>' THEN ERROR(9);*
   PUSH('IF <выр> THEN')
   U := '';
   SCAN().
10. *ERROR(10);*
    STOP().

	#	IF	THEN	:=	+	i
#	5	1	0	6	0	1
IF	0	0	9	0	3	1
IF <выр> THEN	7	1	0	6	0	1
<перем> :=	8	0	0	0	3	1
<выр> +	4	0	4	0	4	1
i	2	0	2	2	2	0

<прог> ::= <инстр>
<инстр> ::= IF <выр> THEN <инстр>
<инстр> ::= <перем> := <выр>
<выр> ::= <выр> + <перем> | <перем>
<перем> ::= i