А. А. Вылиток

Генератор синтаксических анализаторов *yacc*[1]

Программе *yacc* на вход подается КС-грамматика, на выходе получается синтаксический анализатор, написанный на языке Си.

Анализатор, построенный *yacc*'ом, основан на восходящем методе построения дерева вывода, а не на методе рекурсивного спуска. Построение дерева снизу вверх, от листьев к корню, соответствует построению правого вывода «задом наперед»: анализируемая цепочка постепенно сворачивается к цели грамматики (к начальному символу). Поясним это на примере грамматики для простых выражений.

$S \to S + T \mid T$
$T \to T * F \mid F$
$F \to 0 \mid 1$

Рассмотрим цепочку 0 + 1*1 и построим для нее правый вывод «задом наперед», читая цепочку слева направо и применяя правила грамматики так, что правая часть правила, входящая в сентенциальную форму, заменяется на нетерминал, стоящий в левой части (такая замена называется сверткой):

$$\underline{0} + 1 * 1 \leftarrow \underline{F} + 1*1 \leftarrow \underline{T} + 1*1 \leftarrow S + \underline{1}*1 \leftarrow S + \underline{F}*1 \leftarrow S + \mathrm{T}*\underline{1} \leftarrow S + \underline{\mathrm{T}*\mathrm{F}} \leftarrow \underline{S + T} \leftarrow S$$

Подчеркиванием выделены вхождения правых частей, заменяемые на очередном шаге.

Грамматику можно снабдить действиями (на языке Си). Действие выполняется, когда происходит свертка. Для каждого правила вида $A \to A_1A_2\ldots\mathrm{A}_n$, в действиях можно использовать обозначения \$1, \$2, …,\$n, которые содержат значения, полученные при свертках соответственно к A_1, A_2, …, A_n. (если A_i — терминал, то у него нет значения); \$\$ — это значение, вырабатываемое при свертке к A. Его можно задать в виде присваивания: \$\$=….

Добавим в нашу грамматику действия по вычислению значения выражения.

$S \to S + T \langle \$\$=\$1+\$3; \rangle \mid T \langle \$\$=\$1; \rangle$
$T \to T * F \langle \$\$=\$1*\$3; \rangle \mid F \langle \$\$=\$1; \rangle$
$F \to 0 \langle \$\$=0; \rangle \mid 1 \langle \$\$=1; \rangle$

Рассмотрим вывод для 0 + 1*1, указывая в угловых скобках вычисляемые значения для нетерминалов. Значение для S на последнем шаге будет значением всего выражения.

$$\underline{0} + 1*1 \leftarrow \underline{F\langle 0\rangle} + 1*1 \leftarrow \underline{\mathrm{T}\langle 0\rangle} + 1*1 \leftarrow S\langle 0\rangle + \underline{1}*1 \leftarrow S\langle 0\rangle + \underline{F\langle 1\rangle}*1 \leftarrow S\langle 0\rangle + T\langle 1\rangle*\underline{1} \leftarrow S\langle 0\rangle + \underline{T\langle 1\rangle*F\langle 1\rangle} \leftarrow \underline{S\langle 0\rangle + T\langle 1\rangle} \leftarrow S\langle 1\rangle$$

Запись грамматических правил и действий для программы *yacc* имеет свои особенности. Вместо стрелки для разделения левой и правой частей правил ставится двоеточие. Действия записываются не в угловых, а в фигурных скобках. (Фигурные скобки в роли итераторов в *yacc* не используются.)

Укажем общий вид входного потока для *yacc*:

```
%{
Операторы и директивы Си типа #include, описания и т.д.
```

[1] От англ. «yet another compiler-compiler» (еще один компилятор компиляторов). Так назвал свою программу С. Джонсон, поскольку во время ее разработки уже были подобные программы. Но именно эта, благодаря Unix, дожила до наших дней.

```
Эта часть не обязательна.
%}
yacc-описания: лексемы, грамматические переменные, информация о приоритетах и
ассоциативности.
%%
грамматические правила и действия
еще операторы Си (необязательно):
main() {…;yyparse;…}
yylex(){…}
…
```

Входной поток обычно записывается в файл с расширением *y* (например, *file*.y) и подается программе *yacc* командой Unix:

$ *yacc –o file.c file.y*

Результат будет записан в файл *file.c* (если в команде не указано явно имя выходного файла, то результат запишется в файл *y.tab.c*).

Файл-результат имеет следующую структуру:

```
Операторы на Си между %{ и %} исходного файла, если есть
Операторы на Си из части после второй комбинации %%, если есть:
main() {…;yyparse; …}
yylex(){…}
…
yyparse(){анализатор, который обращается за очередной лексемой к функции
yylex()}
```

Имена функций *yyparse* () и *yylex* () фиксированы. Функцию лексического анализа *yylex* () мы должны написать самостоятельно и вставить в исходный файл. Передача лексемы в синтаксический анализатор осуществляется через переменную *yylval*. Также можно добавить в исходный файл функции, реализующие сложные семантические действия. Обращения к этим функциям указываются в грамматике. Функция синтаксического анализа (с семантическими действиями) *yyparse* () строится программой *yacc* автоматически по грамматике с действиями и добавляется в выходной файл. В исходный файл можно добавить управляющую функцию (например, *main* ()), которая будет вызывать синтаксический анализатор *yyparse* ().

Полученную с помощью *yacc* программу на Си можно откомпилировать (*gcc*), возможно, связать с другими программами на Си и выполнить.

Примеры использования *yacc*

Скобочной системой называется цепочка, порождаемая грамматикой $S \to (S)\ S \mid \varepsilon$.

Пустая цепочка, как следует из определения, тоже считается скобочной системой. Непустая скобочная система неделима, если ее нельзя представить в виде конкатенации (сцепления) двух непустых скобочных систем. Например, (( )( )), ((( ))) — неделимые системы, а системы ( )( )( ) и ( )((( ))) таковыми не являются.

Протяжением скобочной системы называется максимальное число неделимых систем, сцепление которых дает данную систему. Например, ( )(( )( )( )) имеет протяжение 2; (( ))( )( ) — протяжение 3; пустая цепочка ε имеет протяжение 0.

Глубина скобочной системы — это максимальный уровень вложенности скобок. Пустая система имеет глубину 0; глубина ( )( )( ) равна 1; глубина ( )(( )(( )))( )( ) равна 3.

Ширина скобочной системы — это максимальное протяжение среди всех подсистем, входящих в данную систему. Пустая система имеет ширину 0; ширина ((( )))( ) равна 2; ширина ( )(( )( )( )) равна 3.

Построим с помощью *yacc* программу, вычисляющую глубину скобочной системы. В грамматику нужно добавить действия по вычислению глубины: пустая имеет глубину ноль, глубина системы вида (*S*) *S* вычисляется как максимальная из глубин двух частей: неделимой подсистемы (*S*) и «хвостовой» подсистемы *S*. Для удобства введем новый начальный символ *P* и добавим к грамматике правило $P \to S$ с действием вывести результат — глубину скобочной системы.

Приведем текст исходного файла *depth.y* для вычисления глубины.

```
%{
/* Программа вычисления глубины скобочной системы */
#include <stdio.h>
%}
%%
P: S { printf ( "depth: %d\n", $1 ); }
S: '('S')'S { $$ = $2+1; if ( $$<$4 ) $$=$4; }
 | /*empty*/{ $$ = 0; }
%%

main () {
 printf ( "type a string, please: " );
 yyparse ();
}

yylex () {
 int c;
 c = getchar ();
 if ( c=='\n' ) return 0;
 yylval = c;
 return c;
}

yyerror ( char *s ) {
  printf ( "Depth eval: %s\n", s );
}
```

Теперь вычислим ширину скобочной системы с помощью *yacc*. Для вычисления ширины опишем глобальную переменную *width*, которую обнулим сначала, в ней будем хранить текущий максимум для протяжений подсистем. Действия по вычислению: для ε протяжение равно нулю, для (*S*)*S* протяжение на 1 больше, чем для «хвоста» *S*. Если протяжение (*S*)*S* больше чем *widh*, полагаем *widh* равным протяжению (*S*)*S*.

Файл *widh.y* для вычисления ширины скобочной системы:

```
%{
#include <stdio.h>
int width=0;
%}
%%
P: S { printf ( "width: %d\n", width ); }
S: '('S')'S { $$=$4+1; if ( width<$$ ) width=$$; }
```

```
    | /*empty*/{ $$=0; }
%%
main() {
  printf ( "type a string, please: " );
  yyparse ();
}

yylex () {
  int c;
  c = getchar();
  if ( c=='\n' ) return 0;
  yylval = c;
  return c;
};

yyerror ( char *s ) {
  printf ( "Width eval: %s\n", s );
};
```

Задачи для самостоятельного решения

1. С помощью *yacc* построить программу, вычисляющую протяжение скобочной системы.

2. С помощью системы *yacc* построить программу-калькулятор для выражений, содержащих целые числа, операции «+», «*» , «–» , «/» и круглые скобки. Приоритет операций «*» и «/» выше, чем у «+», «*». Все операции левоассоциативны.