# Генератор компиляторов Yacc

Yacc – Yet Another Compiler – Compiler.

Описание компилятора имеет три части:

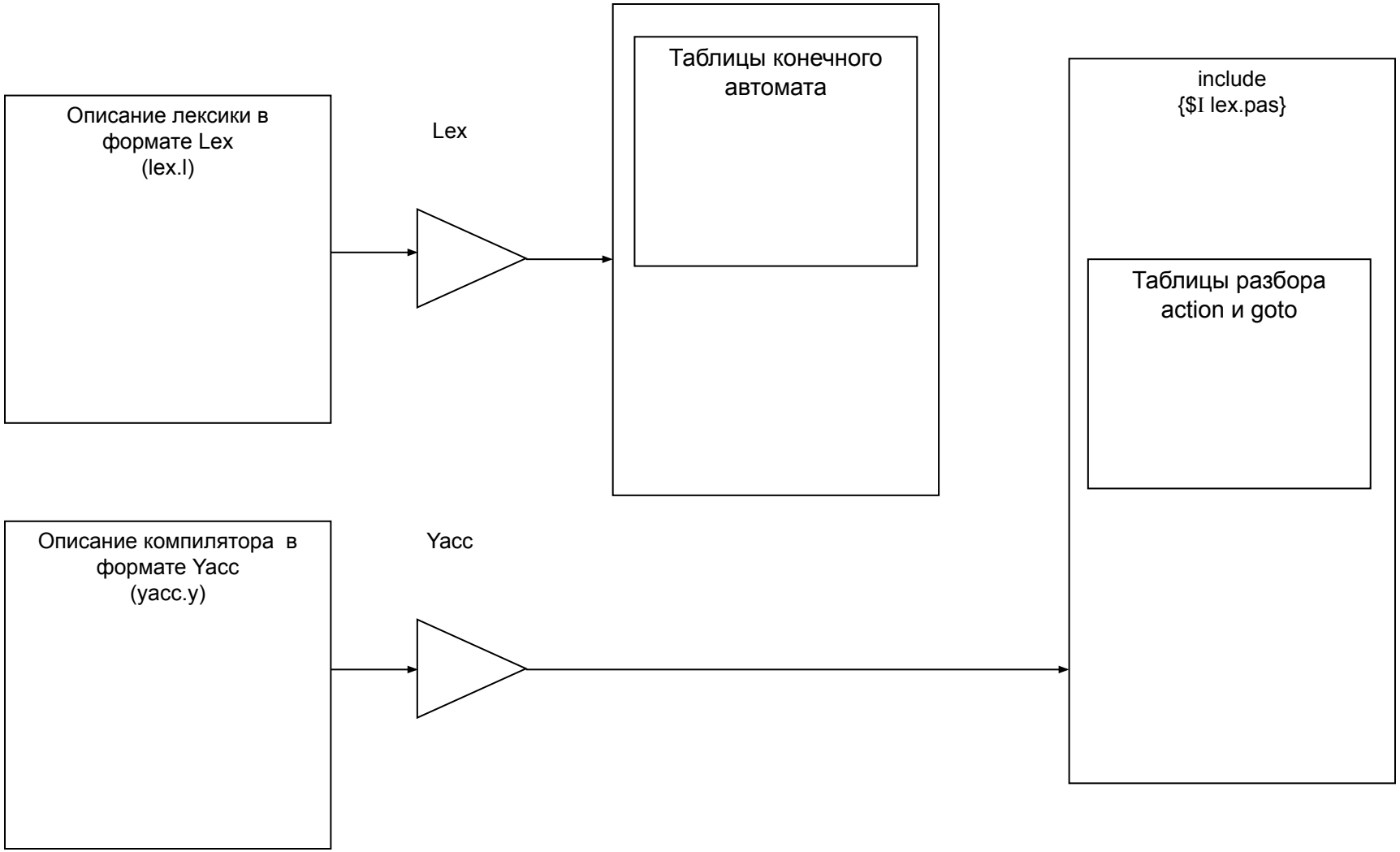
**определения**

%%

**продукции**

%%

**вспомогательные процедуры**


Описание лексики в формате Lex (lex.l)
Lex
Таблицы конечного автомата
include {$I lex.pas}
Таблицы разбора action и goto
Описание компилятора в формате Yacc (yacc.y)
Yacc

# Определения

- Задание стартового нетерминала
  **`%start <symbol>`**
- Определение терминалов
  **`%token <symbol>`**
- Задание ассоциативности операторов
  **`%left <symbol>`**
  **`%right <symbol>`**
  **`%nonassoc <symbol>`**

# Примеры

Задание приоритета и ассоциативности символов

```
%nonassoc ‘<‘ ‘>’ ‘=‘
%left ‘+‘ ‘-’ OR
%left ‘*‘ ‘/’ AND
%right NOT UMINUS
```

Задание типов символов

```
%token <Real> <symbol>
%left <AddOp> <symbol>
%type <Real> expr
```

# Продукции и семантические правила

- Продукция A → B C на языке Yacc записывается:
  **`A : B C;`**
- Если определение нетерминала имеет несколько альтернатив:
  A → B C
  | D E
  | F
  то на языке Yacc оно записывается:

```
A : B C
  | D E
  | F
  ;
```

# Пример

```
%left ‘+‘ ‘-’
%left ‘*‘ ‘/’
%token NUM
%%
expr : expr ‘+‘ expr
     | expr ‘-‘ expr
     | expr ‘*‘ expr
     | expr ‘/‘ expr
     | ‘(‘ expr ‘)‘
     | NUM
     ;
```

# Семантические действия

Имена атрибутов

- $$ - атрибут нетерминала левой части
- $i – атрибут i-того грамматического символа правой части
- $$ := $1 используется по умолчанию

# Пример

```
%left `+` `-`
%left `*` `/`
%token <Real> NUM
%type <Real> expr
%%
expr : expr `+` expr {$$ := $1 + $3}
     | expr `-` expr {$$ := $1 - $3}
     | expr `*` expr {$$ := $1 * $3}
     | expr `/` expr {$$ := $1 / $3}
     | `(` expr `)`  {$$ := $2}
     | NUM
     ;
```

По умолчанию используется действие **{$$ := $1}**

# Действия внутри правой части

Определение

```
x : y { action; } z
```

заменяется на

```
x : y $act z
$act: { action; }
```

Пример

```
x : y { $$:=2*$1 } z { $$:=$2*$3 }
```

# Работа синтаксического анализатора

```
%token NUM
%left '+'
%left '*'
%%
expr : expr '+' expr
     | expr '*' expr
     | '(' expr ')'
     | NUM
     ;
```

# Работа синтаксического анализатора

```
state 5:
  expr : expr '+' _ expr
  '('   shift 2
  NUM   shift 3
  .     error
  exprgoto 8
```

Ситуация
**[expr → expr '+' . expr]**

action

goto

иначе

# Дополнительная продукция

```
state 0:
  $accept : _ expr $end
  '('  shift 2
  NUM  shift 3
  .  error
  exprgoto 1
```

# Пример конфликта сдвиг-свертка

```
%token if then else
%%
stmt : if expr then stmt
         | if expr then stmt else stmt
```

---

Для последовательности

```
if expr-1 then
          if expr-2 then stmt-1 else stmt-2
```

возможны два варианта разбора

```
if expr-1 then
    (if expr-2 then stmt-1 else stmt-2)
if expr-1 then
    (if expr-2 then stmt-1) else stmt-2
```

# Пример конфликта свёртка-свёртка

```
%right SUB SUP
%
EXPR : EXPR SUB EXPR SUP EXPR       (1)
       | EXPR SUB EXPR              (2)
       | EXPR SUP EXPR              (3)
```