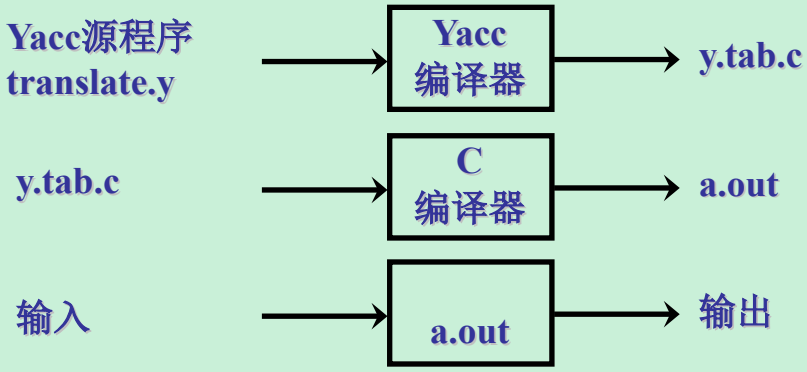
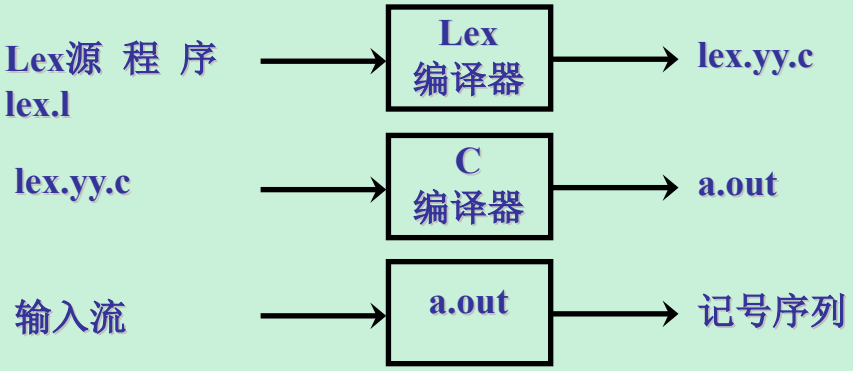
3.7 分析器的生成器

3.7.1 分析器的生成器Yacc



用Lex建立词法分析器的步骤



3.7 分析器的生成器

- Yacc程序包括三个部分
  - 声明
  - %%
  - 翻译规则
  - %%
  - 支持例程

3.7 分析器的生成器

E→E + T | T

T→T * F | F

F→(E) | digit

- 例---声明部分

```
%{
#include <ctype.h>                    /* 常量、变量的声明*/
%}
%token DIGIT
%%
```

3.7 分析器的生成器

E→E + T | T

T→T * F | F

F→(E) | digit

- 例---翻译规则部分

```
line    : expr '\n'              {printf("%d\n", $1);}
        ;
expr    : expr '+' term          {$$ = $1+$3;}
        | term
        ;
term    : term '*' factor        {$$ = $1 * $3;}
        | factor
        ;
factor : '(' expr ')'    {$$ = $2;}
        | DIGIT
        ;
%%
```

3.7 分析器的生成器

E→E + T | T

T→T * F | F

F→(E) | digit

- 例---支持例程部分

```
yylex(){
    int c;
    c = getchar();
    if (isdigit(c)){
        yylval = c – ‘0’;
        return DIGIT;
    }
    return c;
}
```

3.7 分析器的生成器

3.7.2 用Yacc处理二义文法

解决分析动作冲突的两大默认规则：

- 对于归约-归约冲突，选择在Yacc 程序中最先出现的那个产生式归约
- 对于移进-规约冲突，优先移进

3.7 分析器的生成器

3.7.2 用Yacc处理二义文法

例 台式计算器

- 输入一个表达式并回车，显示计算结果。
- 也可以输入一个空白行。

lines→ lines expr ‘\n’ | lines ‘\n’ | e

E→ E + E | E – E | E * E | E / E | (E) | -E | number

3.7 分析器的生成器

```
%{
# include  <ctype .h>
# include  <stdio.h >
# define YYSTYPE double  /* 将栈定义为double类型 */
%}

%token  NUMBER
%left  '+'  '–'
%left  '*'  '/ '
%right  UMINUS
%%
```

lines→ lines expr '\n' | lines '\n' | e

E→ E + E | E – E | E * E | E / E | (E) | -E | number

3.7 分析器的生成器

```
lines       : lines expr '\ n'    {printf ( "%g \ n", $2 ) }
            | lines '\ n'
            |  /* ε */
            ;
expr        : expr '+' expr         {$$ = $1 + $3; }
            | expr '–' expr         {$$ = $1 – $3; }
            | expr '*' expr         {$$ = $1 * $3; }
            | expr '/ ' expr        {$$ = $1 / $3; }
            | '(' expr ')'          {$$ = $2; }
            | '–' expr  %prec UMINUS {$$ = –$2; }
            | NUMBER
            ;
%%
```

lines→ lines expr '\n' | lines '\n' | e

E→ E + E | E – E | E * E | E / E | (E) | -E | number

3.7 分析器的生成器

```
yylex ( ) {
        int c;
        while ( ( c = getchar ( ) ) = = ‘  ’ );
        if ( ( c = = ‘.’ ) | | (isdigit (c) ) ) {
                ungetc (c, stdin);
                scanf ( “% lf ”,  &yylval);
                return  NUMBER;
        }
        return c;
}
```

lines→ lines expr ‘\n’ | lines ‘\n’ | e

E→ E + E | E – E | E * E | E / E | (E) | -E | number

4.1 语法制导的定义

例　简单台式计算器的语法制导定义

产 生 式	语 义 规 则
$L \rightarrow E$ n	*print* (*E.val*)
$E \rightarrow E_1 + T$	*E.val* := E_1 *.val* + *T.val*
$E \rightarrow T$	*E.val* := *T.val*
$T \rightarrow T_1 * F$	*T.val* := T_1*.val* * *F.val*
$T \rightarrow F$	*T.val* := *F.val*
$F \rightarrow (E)$	*F.val* := *E.val*
$F \rightarrow$ digit	*F.val* := digit.*lexval*

4.1 语法制导的定义

4.1.1 语法制导定义的形式

- 基础文法
- 每个文法符号有一组属性
- 每个文法产生式$A \rightarrow \alpha$有一组形式为$b := f(c_1, c_2, \ldots, c_k)$的语义规则，其中$f$是函数，$b$和$c_1, c_2, \ldots, c_k$是该产生式文法符号的属性
  - 综合属性：如果b是A的属性，$c_1, c_2, \ldots, c_k$是产生式右部文法符号的属性或A的其它属性。
  - 继承属性：如果b是产生式右部某个文法符号X的属性。

属性值由分析树中它的子结点的属性值来计算

属性值由结点的兄弟结点及父结点的属性值来计算。

4.1 语法制导的定义

4.1.2 综合属性

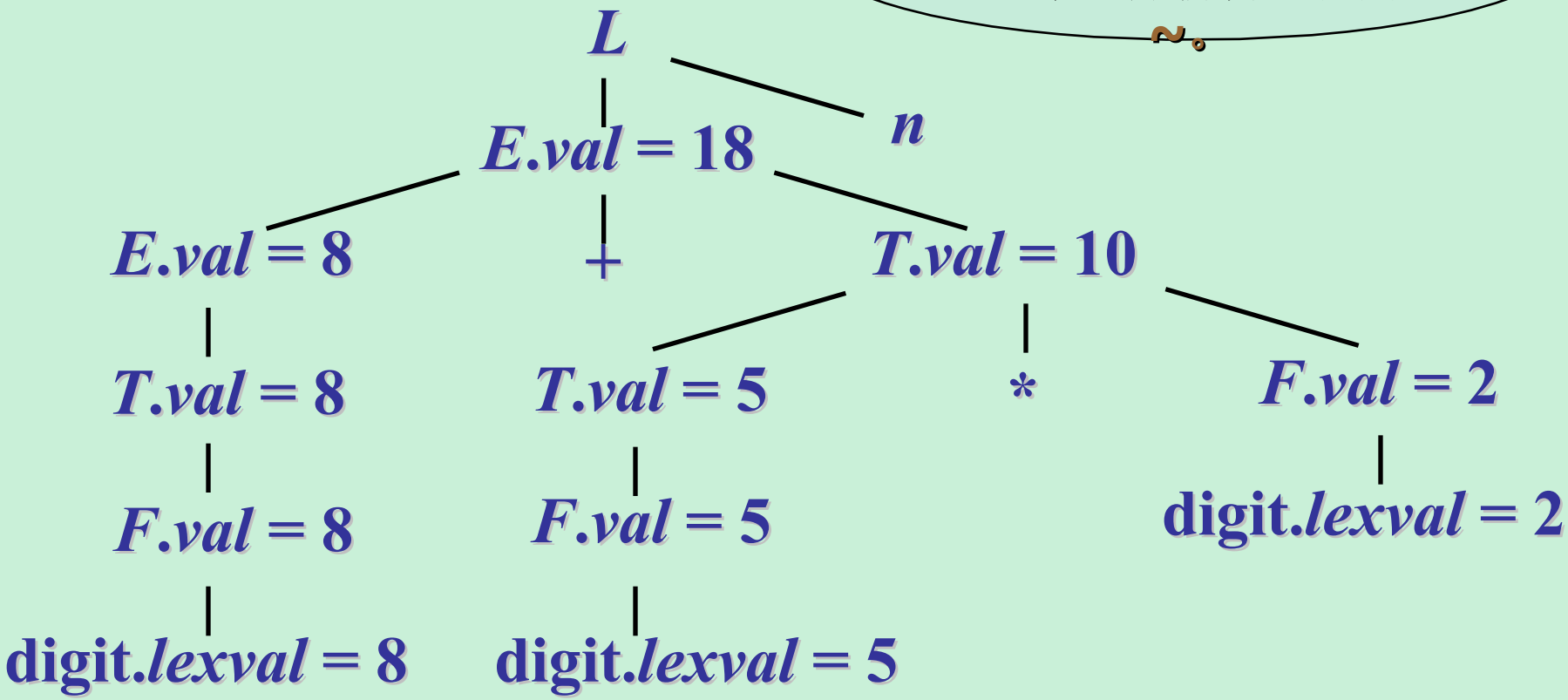
S属性定义：仅仅使用综合属性的语法制导定义

产 生 式	语 义 规 则
$L \rightarrow E$ n	*print* (*E.val*)
$E \rightarrow E_1 + T$	*E.val* := E_1 *.val* + *T.val*
$E \rightarrow T$	*E.val* := *T.val*
$T \rightarrow T_1 * F$	*T.val* := T_1*.val* * *F.val*
$T \rightarrow F$	*T.val* := *F.val*
$F \rightarrow (E)$	*F.val* := *E.val*
$F \rightarrow$ digit	*F.val* := digit.*lexval*

4.1 语法制导的定义

8+5*2 n的注释分析树

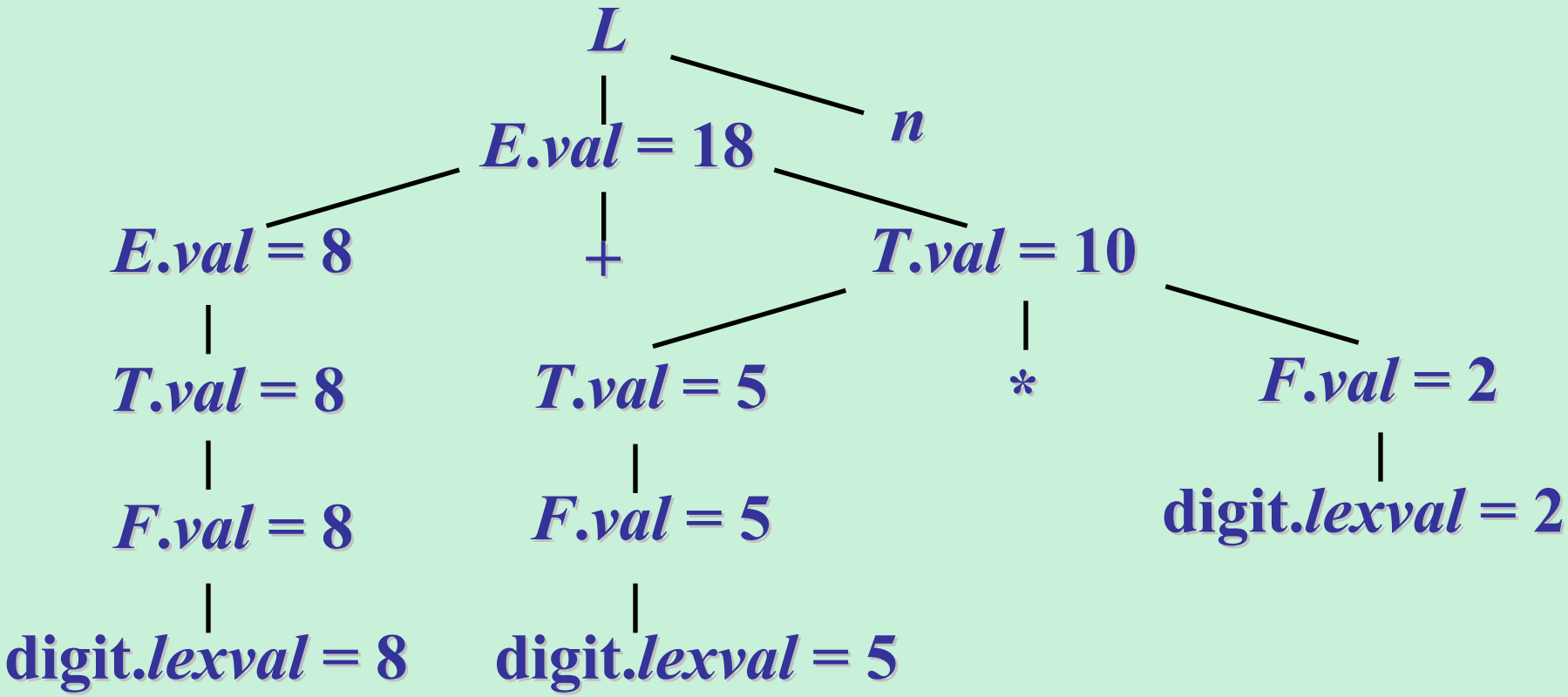
每个结点的属性值都标注出来的分析树，称为~。



4.1 语法制导的定义

8+5*2 n

分析树各结点属性的计算可以自下而上地完成



4.1 语法制导的定义

8+5*2 n

分析树各结点属性的计算可以自下而上地完成