语义分析与中间代码生成

语义分析：概览

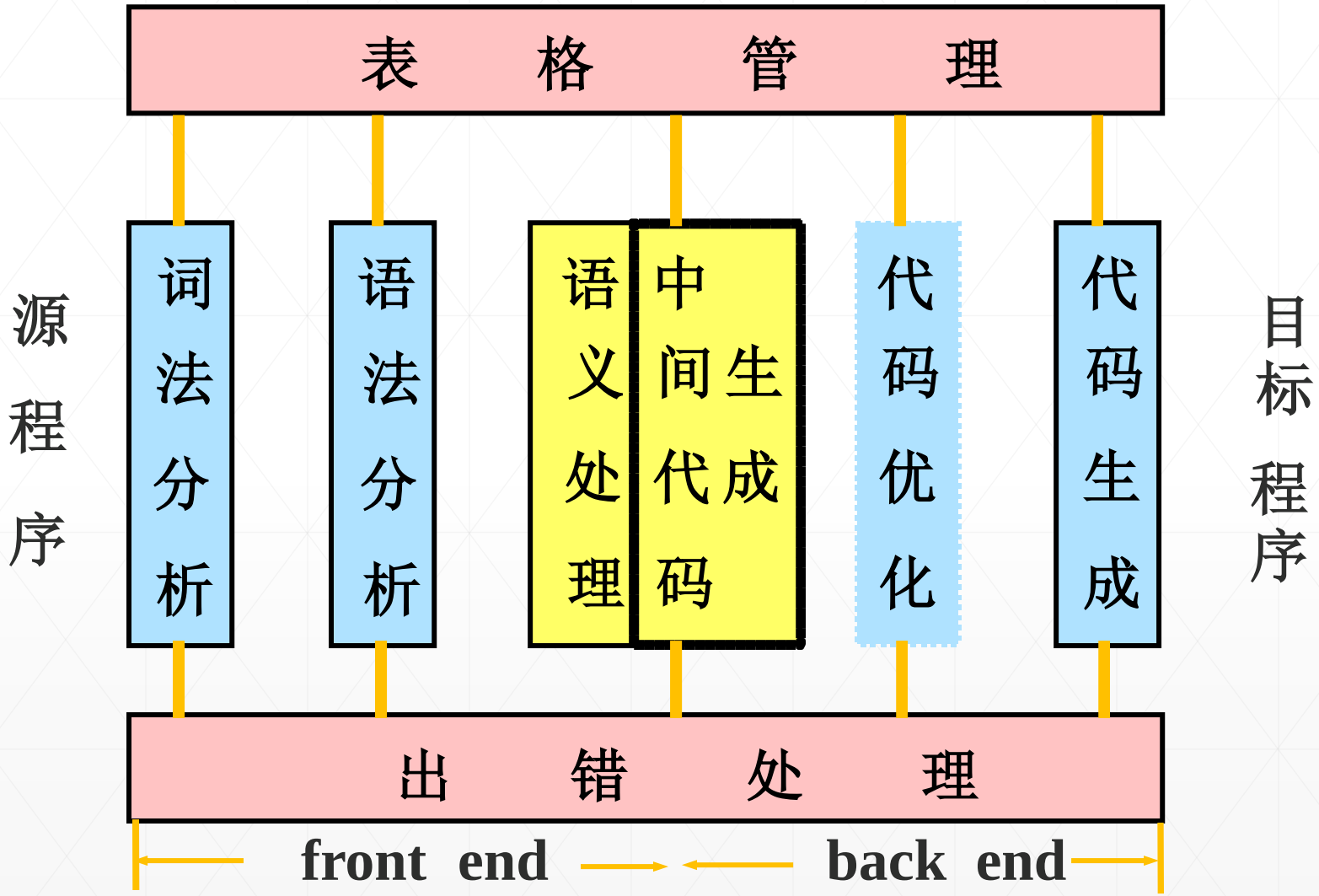
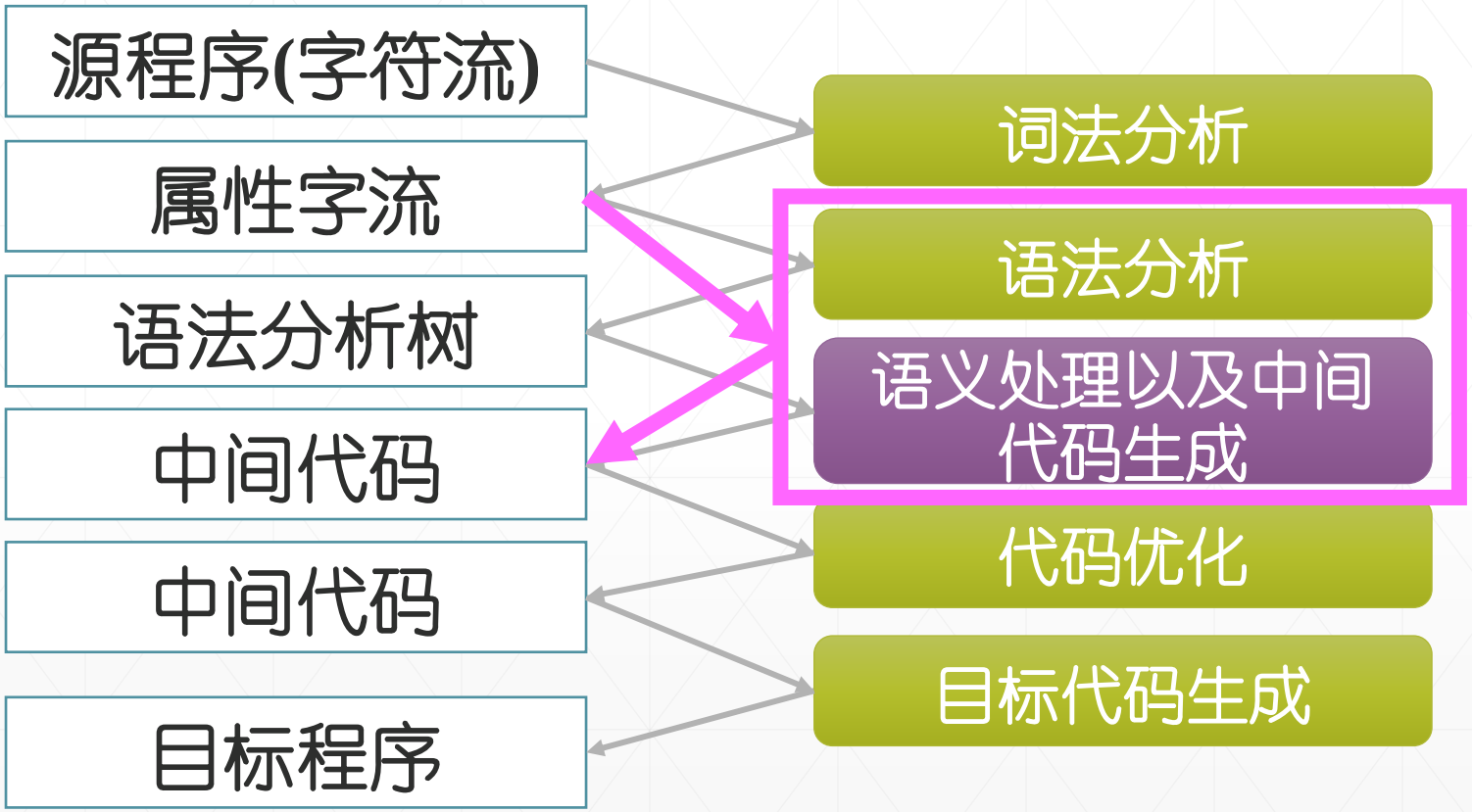
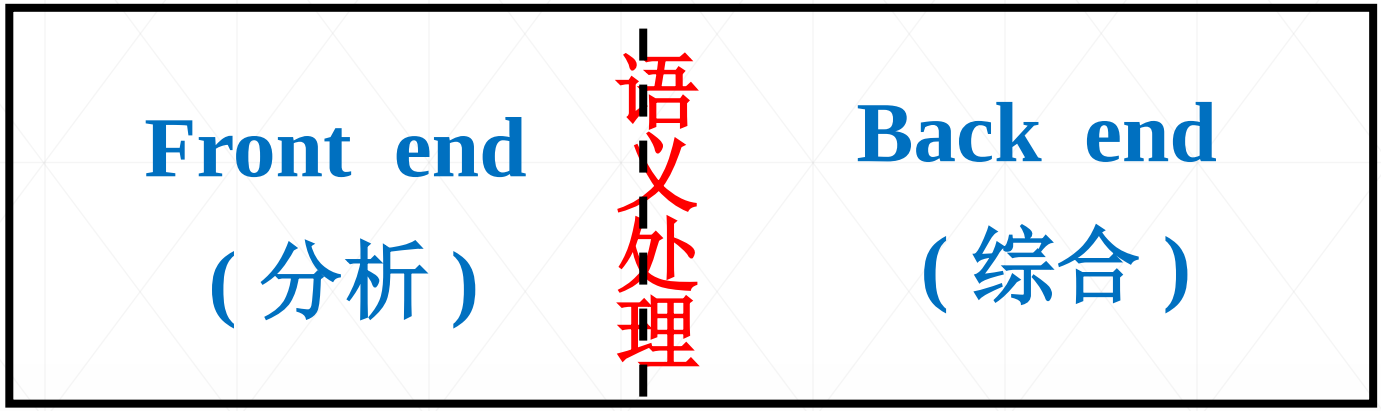


表 格 管 理
源程序
词法分析
语法分析
语义处理
中间代码生成
代码优化
代码生成
目标程序
出 错 处 理
front end
back end

- 基本功能

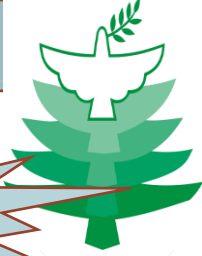

源程序(字符流)
属性字流
语法分析树
中间代码
中间代码
目标程序
词法分析
语法分析
语义处理以及中间代码生成
代码优化
目标代码生成

语义处理的地位



编译程序最实质性的工作；

第一次对源程序的语义作出解释，引起源程序质的变化。



语义处理的任务

输入

按照语法分析器识别的语法范畴结构，

依据其语义规则

依据

进行语义检查和处理，

输出

产生相应的中间代码或目标代码。

语法的局限(上下文无关文法)，
没有在语法中定义的结构通过语义处理描述
语法正确，但是不正确的程序，

- 重复定义的标识符
- 函数参数不匹配
- 类型不兼容的访问
- Break语句的位置
- 未声明的标识符
- Goto的目标不存在
- …

静态语义错误

```
foo(int a, char * s){...}

int bar() {
  int f[3];
  int i, j, k;
  char q, *p;
  float k;
  foo(f[6], 10, j);
  break;
  i->val = 42;
  j = m + k;
  printf("%s,%s.\n",p,q);
  goto label42;
}
```

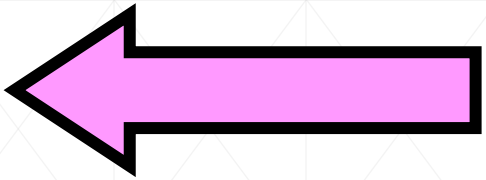
中间代码

介于源语言和目标代码之间的一种代码。

引入中间代码的目的

1. 方便生成目标代码；
2. 便于优化；
3. 便于移植。

第 6 章　语义分析与中间代码生成



6.1　语法制导翻译

6.2　符号表

6.3　类型检查

6.4　中间语言

6.5　中间代码生成

6.1 语法制导翻译

语法结构引导语义的翻译

根据语法结构定义出语义

语法制导定义

6.1 语法制导翻译

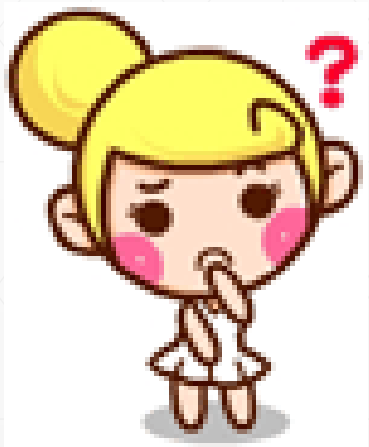
6.1.1 语法制导定义

6.1.2 SDD的求值顺序

6.1.3 语法制导翻译的应用

6.1.4 语法制导的翻译方案SDT

6.1.5 实现L属性的SDD



教材：龙书

属性

属性同变量一样，可以进行计算和传递

- 语义定义在文法结构上
- 结构表示为文法符号
- 结构表达的语义用文法符号的属性表示

文法符号的属性

表示为：

$N.t$ （N是文法符号，t是N的属性）

属性可以为：数、符号串、代码、类型、存储空间…

- 属性的值由产生式的语义规则来定义；
- 属性计算的过程既是语义处理的过程。

非终结符号的两类属性

- **综合属性(Synthesized Attributes,*s*属性)**

产生式左部符号的某些属性由候选式中符号的属性和(或)自己的其他属性定义；

- **继承属性(Inherited Attributes,*i*属性)**

候选式中符号的某些属性由产生式左侧非终结符号的属性和(或)候选式中其他符号的某些属性定义。

终结符号只有综合属性，由词法分析提供

- 语法制导定义 (Syntax-Directed Definition,SDD) 的一种描述形式——属性翻译文法
  - 形式定义 $A = (G, V, F)$

    其中：

    G： 二型文法；

    V： 属性的有穷集；

    F：用属性描述的与产生式相关的语义规则

简单运算的台式计算器的SDD

综合属性

输出E.val

产生式	语义规则
$L \rightarrow E;$	L .val = E .val
$E \rightarrow E + T$	E .val = E_1 .val + T .val
$E \rightarrow T$	E .val = T .val
$T \rightarrow T * F$	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

终结符号的属性

实际应用中：存在非属性描述的语义规则(动作)看做虚属性（副作用）

乘法运算的SDD

产生式	语义规则
$T \rightarrow FT'$	$T'.i = F.val$
	$T.val = T'.s$
$T' \rightarrow *FT'$	$T_1'.i = T'.i \times F.val$
	$T'.s = T_1'.s$
$T' \rightarrow \varepsilon$	$T'.s = T'.i$
$F \rightarrow digit$	$F.val = digit.lexval$

继承属性

简单类型说明语句的SDD

产生式	语义规则
$D \rightarrow T\ L$	$L.type = T.type$
$T \rightarrow int$	$T.type = int$
$T \rightarrow float$	$T.type = float$
$L \rightarrow L, i$	$L_1.type = L.type$
	$Addtype(i.entry, L.type)$
$L \rightarrow i$	$Addtype(i.entry, L.type)$

继承属性

虚属性

6.1 语法制导翻译

6.1.1 语法制导定义

6.1.2 SDD的求值顺序

6.1.3 语法制导翻译的应用

6.1.4 语法制导的翻译方案

6.1.5 实现L属性的SDD

教材：龙书

依赖图

语法分析树中结点属性计算的依赖关系图

在语法分析树的基础上构建

增加属性结点：

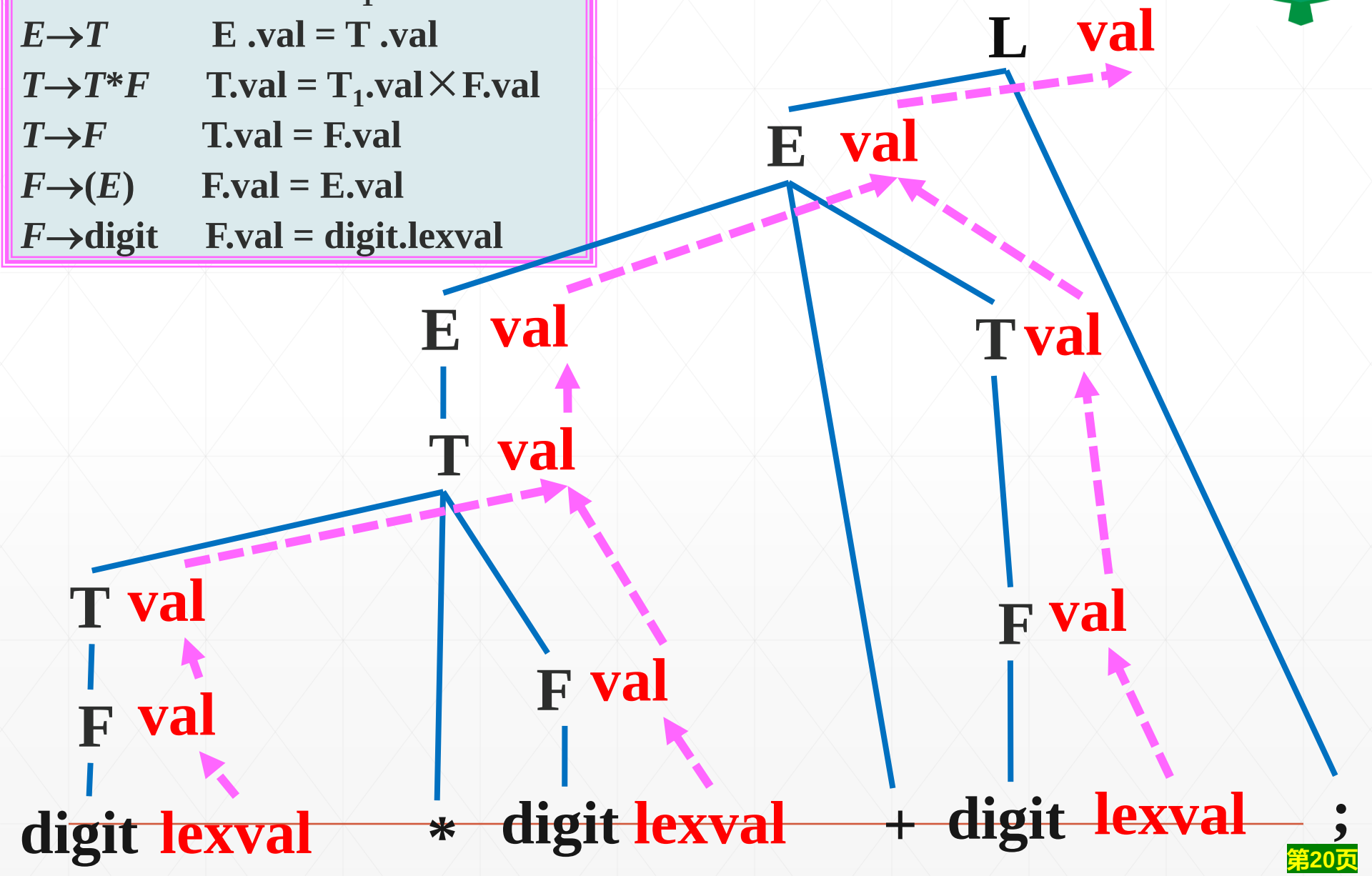
继承属性标在符号的左边，综合属性标在符号的右边

增加属性节点之间的边：

若属性b的计算依赖于属性c，则从结点c到结点b画一条有向边。

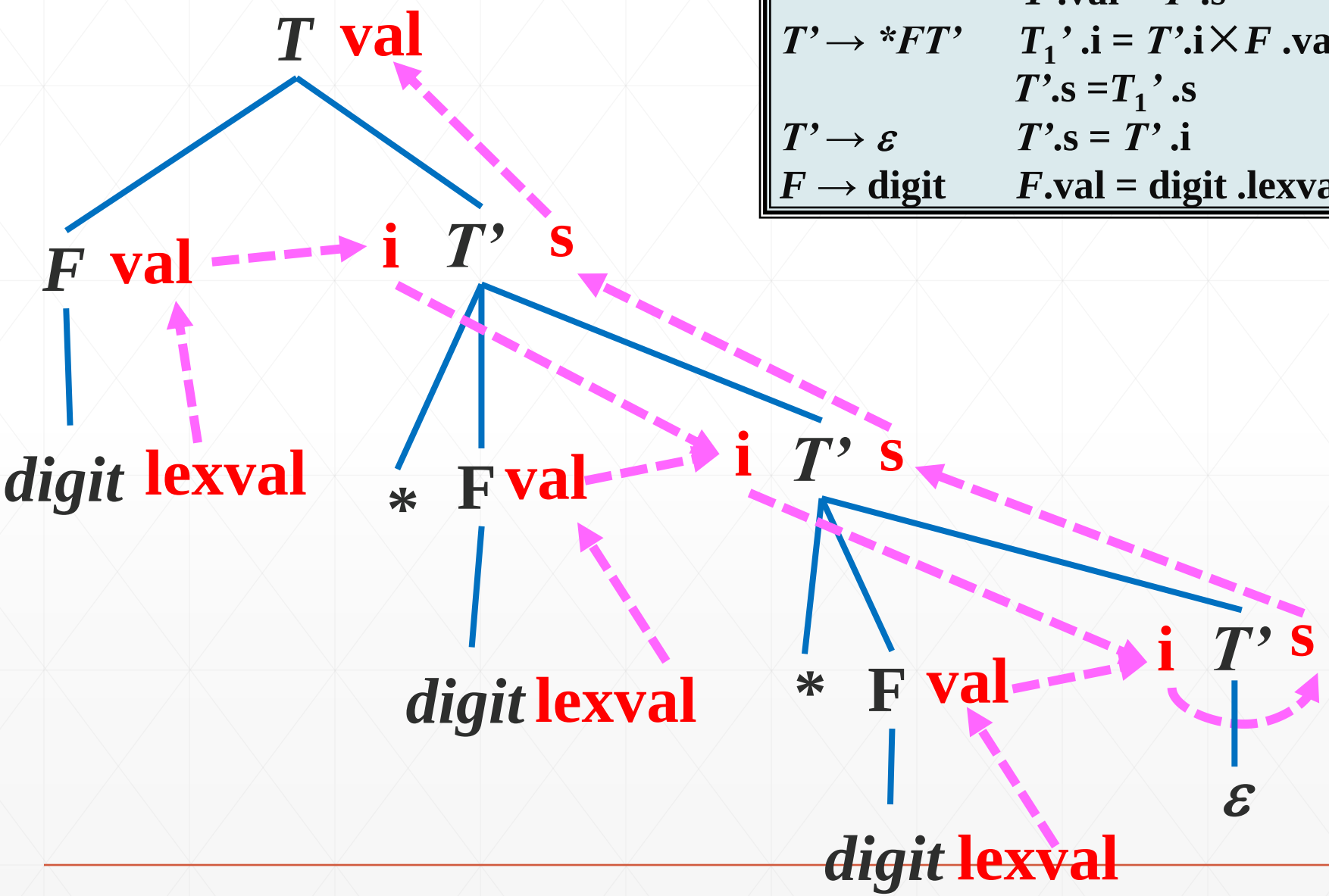
$L \rightarrow E$;	L.val = E .val
$E \rightarrow E+T$	E.val = E_1.val + T.val
$E \rightarrow T$	E .val = T .val
$T \rightarrow T*F$	T.val = T_1.val × F.val
$T \rightarrow F$	T.val = F.val
$F \rightarrow (E)$	F.val = E.val
$F \rightarrow$digit	F.val = digit.lexval

例如， 3 * 5 + 4 ;



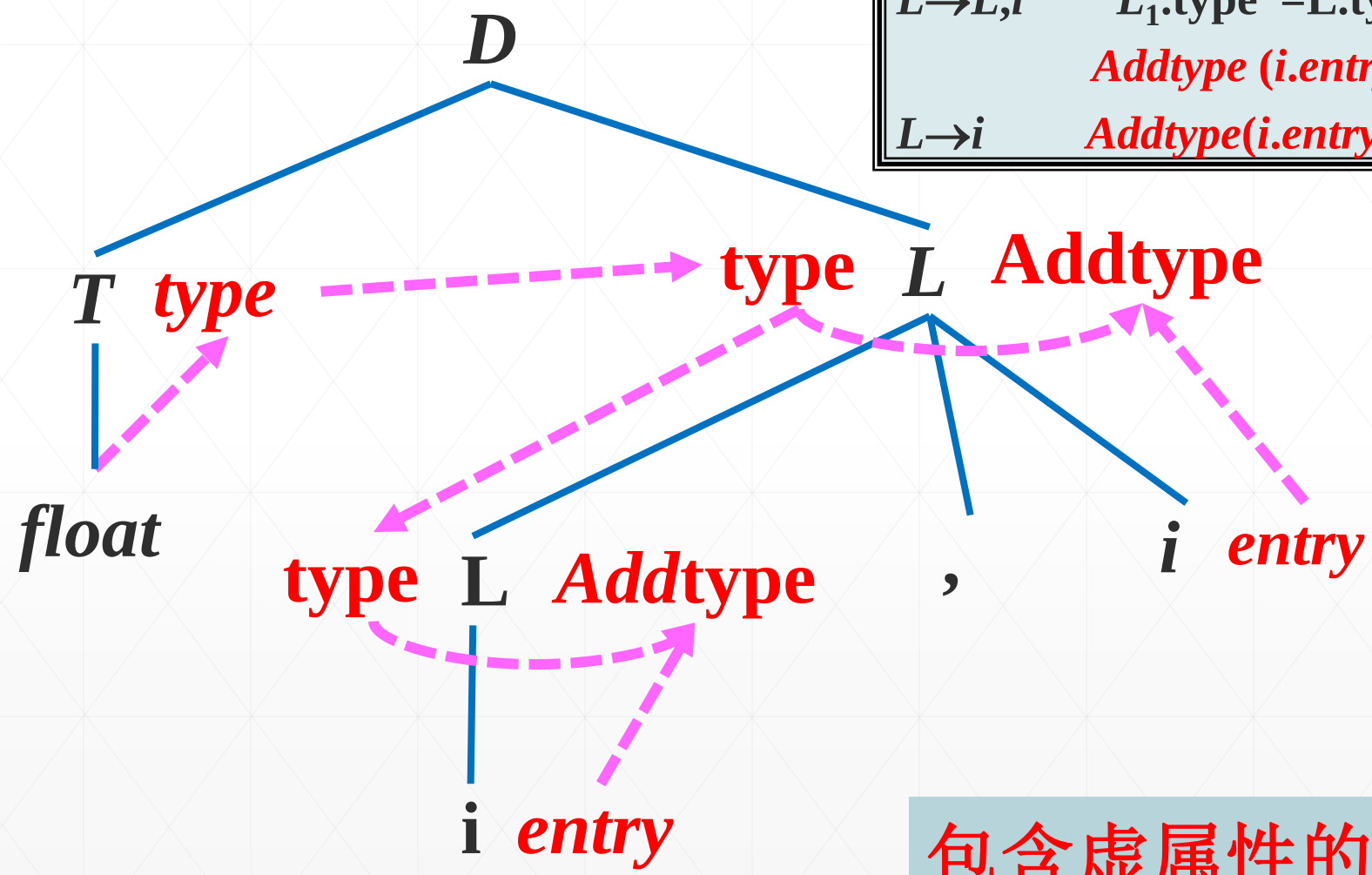
例如，3*5*7

$T \rightarrow FT'$	$T'.i = F.val$
	$T.val = T'.s$
$T' \rightarrow *FT'$	$T_1'.i = T'.i \times F.val$
	$T'.s = T_1'.s$
$T' \rightarrow \varepsilon$	$T'.s = T'.i$
$F \rightarrow digit$	$F.val = digit.lexval$



例如，float i_1，i_2

$D \rightarrow TL$	$L.type = T.type$
$T \rightarrow float$	$T.type = float$
$L \rightarrow L,i$	$L_1.type = L.type$
	$Addtype(i.entry, L.type)$
$L \rightarrow i$	$Addtype(i.entry, L.type)$



包含虚属性的处理

属性计算(加工)的过程即是语义处理的过程

依赖图刻画了属性计算时的一些顺序要求，

属性结点M到结点N有一条边，那么计算结点N对应的属性时，必须计算出M结点对应的属性。

根据依赖图，求出属性结点的一个拓扑排序，此排序就是属性结点的一个计算顺序。

可能无拓扑排序(存在环)

实际应用中，结合语法分析时语法分析树的构造顺序，定义出相对应的属性文法，使属性计算顺序与分析树的展开顺序一致。

根据两种语法分析方法，我们主要介绍两种相对应的属性文法

S-属性文法

L-属性文法

适合在语法分析过程中处理语义的两类属性文法

- S-属性文法 自下而上的属性翻译文法

1.所有非终结符号只有综合属性

可以通过后续遍历语法分析树计算完所有属性

最简单的S属性
描述可能不严格
但实用

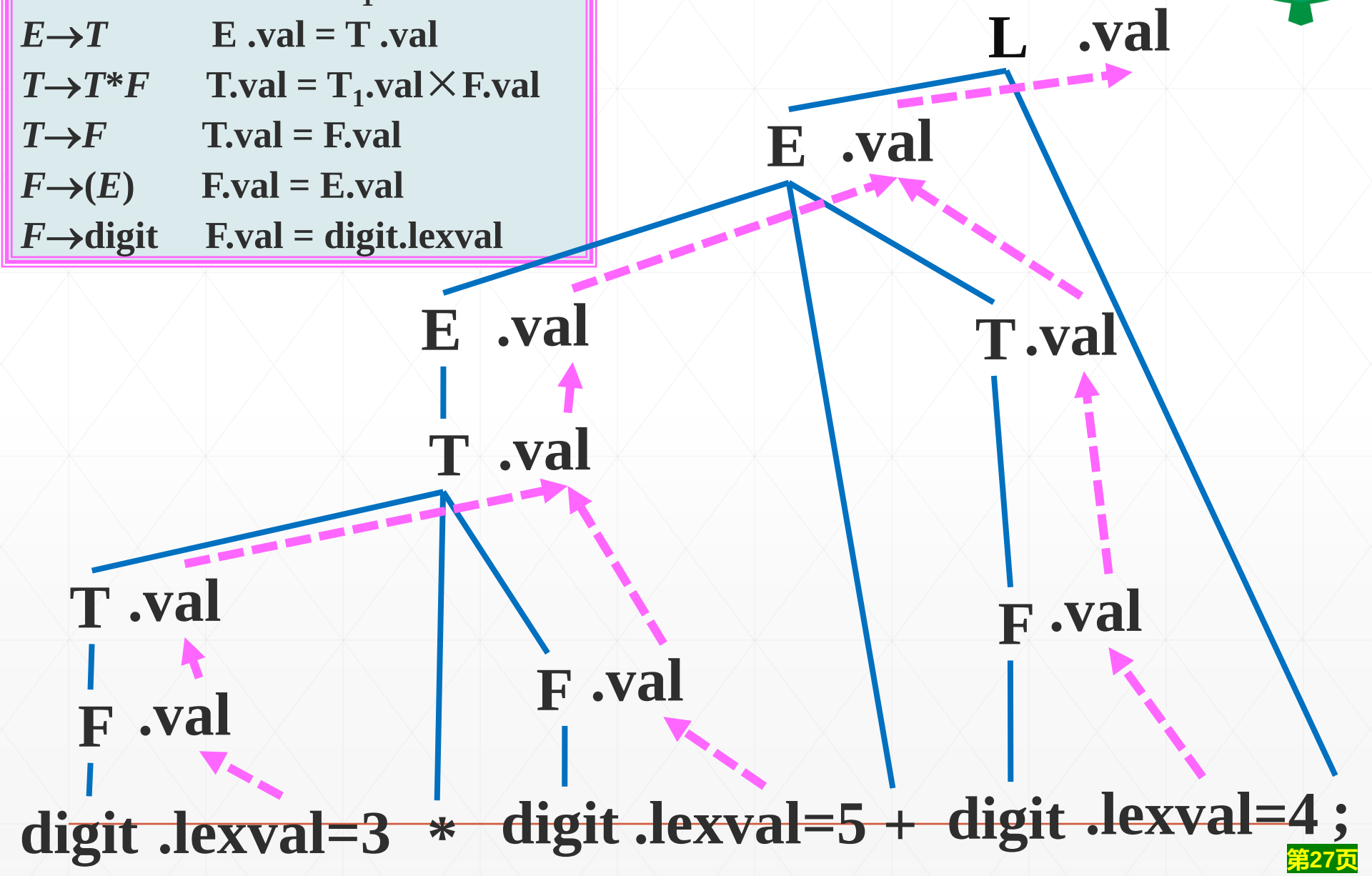
简单运算的台式计算器的SDD

产生式	语义规则
$L \rightarrow E;$	$L.val = E.val$
$E \rightarrow E + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T * F$	$T.val = T_1.val \times F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow digit$	$F.val = digit.lexval$

S属性

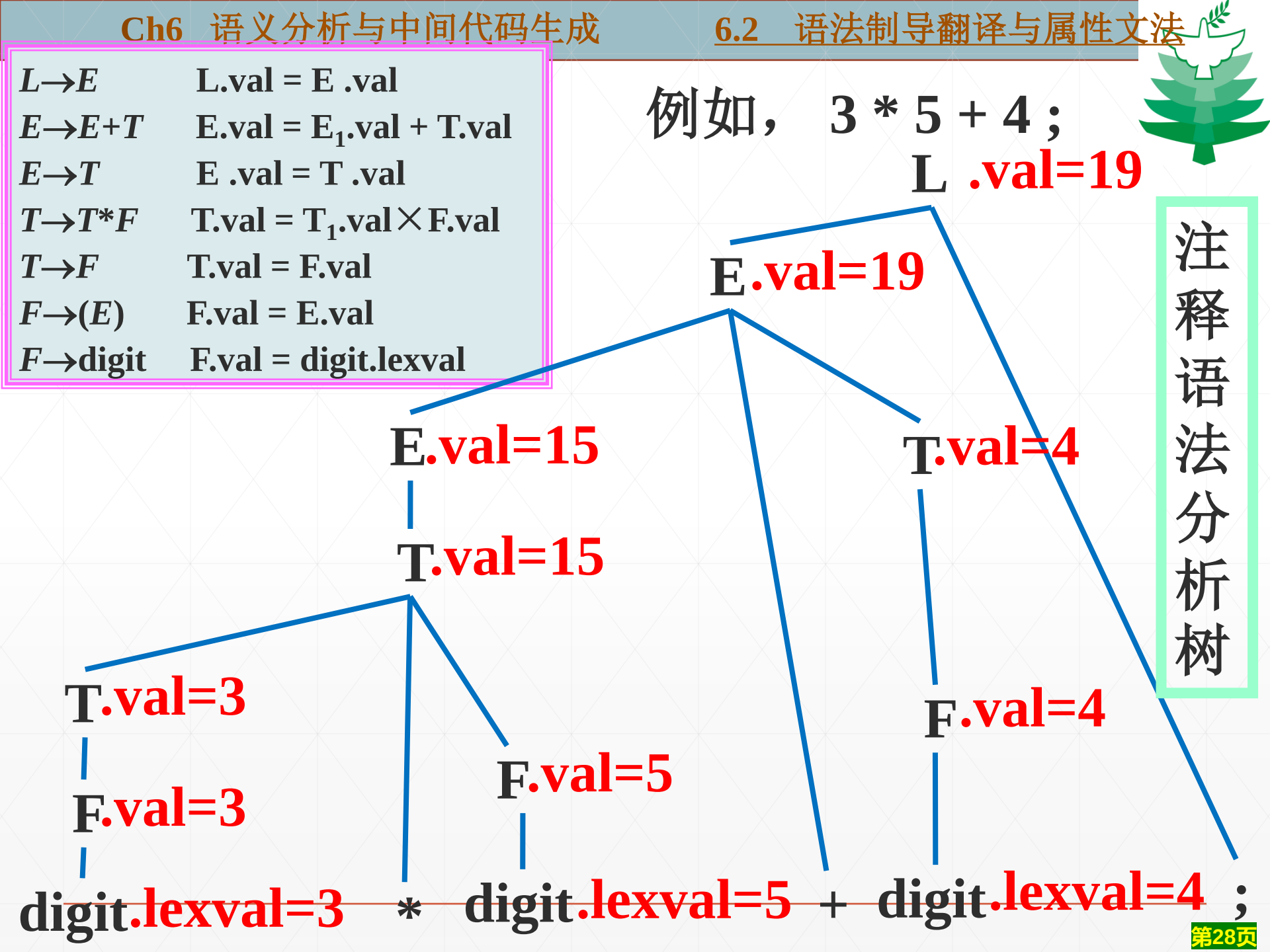
$L \rightarrow E;$	L.val = E .val
$E \rightarrow E+T$	E.val = E_1.val + T.val
$E \rightarrow T$	E .val = T .val
$T \rightarrow T*F$	T.val = T_1.val × F.val
$T \rightarrow F$	T.val = F.val
$F \rightarrow (E)$	F.val = E.val
$F \rightarrow$digit	F.val = digit.lexval

例如， 3 * 5 + 4 ;



$L \rightarrow E$	L.val = E .val
$E \rightarrow E+T$	E.val = E_1.val + T.val
$E \rightarrow T$	E .val = T .val
$T \rightarrow T*F$	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

例如， 3 * 5 + 4 ;



注释语法分析树

适合在语法分析过程中处理语义的两类属性文法

- **L-属性文法** 自上而下的属性翻译文法

1.综合属性

2.继承属性

要求：

或者是左部非终结符号的继承属性

或者是位于其左侧兄弟的任何属性

或者是自己的其他继承属性，但自身属性间存在拓扑排序（不存在环）

可通过深度优先遍历语法分析树计算完所有属性

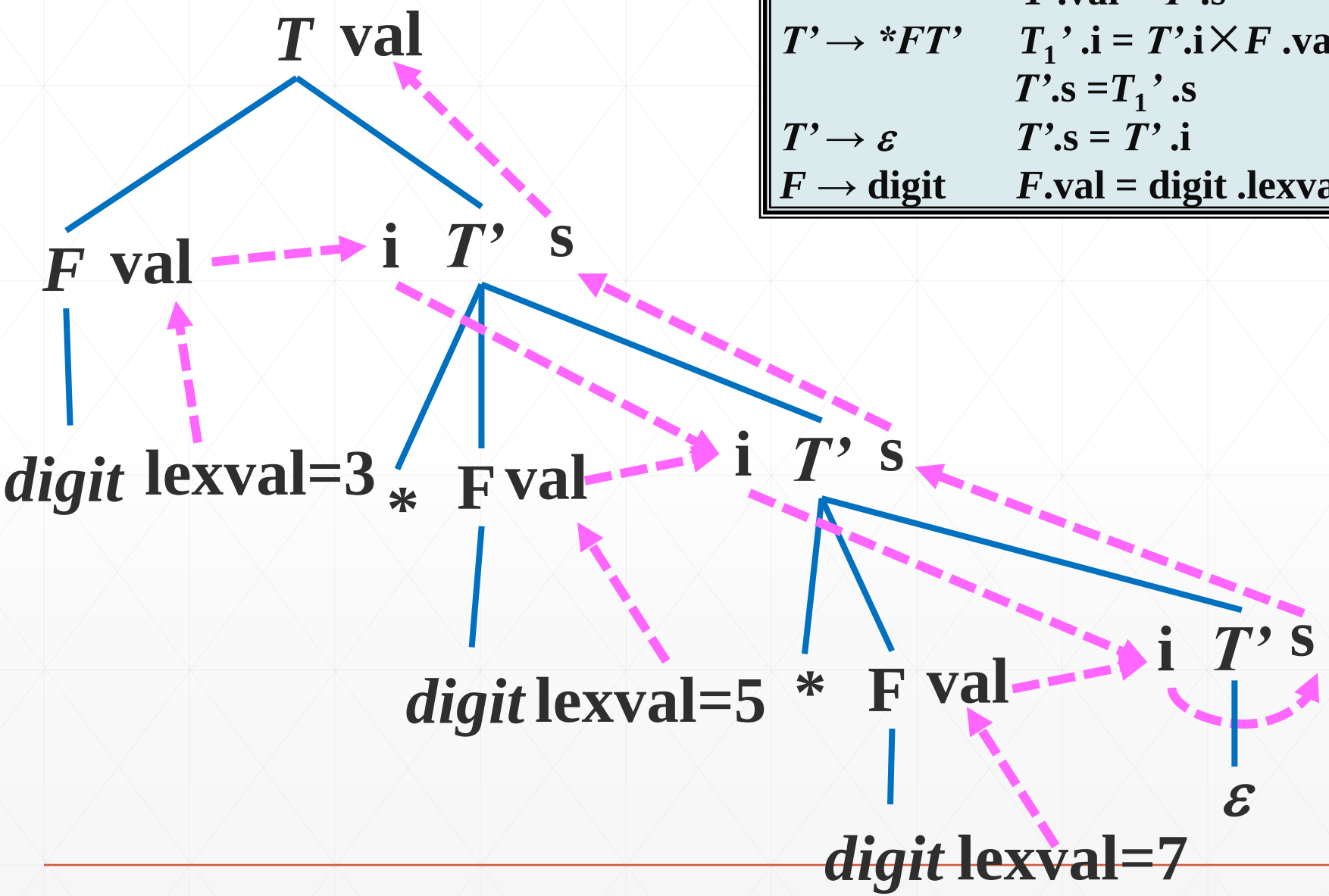
乘法运算的SDD

产生式	语义规则
$T \rightarrow FT'$	$T'.i = F.val$ $T.val = T'.s$
$T' \rightarrow *FT'$	$T_1'.i = T'.i \times F.val$ $T'.s = T_1'.s$
$T' \rightarrow \varepsilon$	$T'.s = T'.i$
$F \rightarrow digit$	$F.val = digit.lexval$

L属性文法

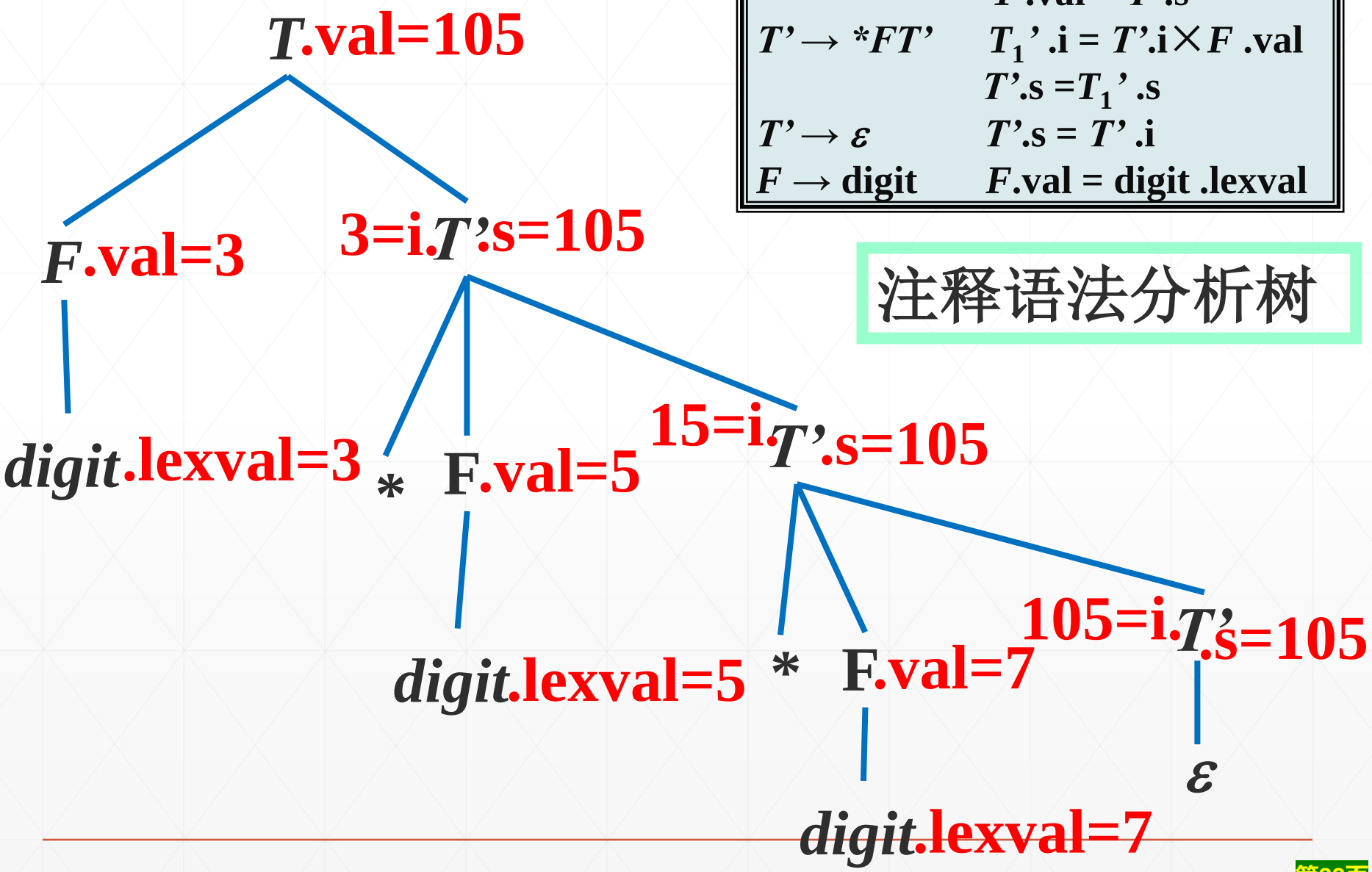
例如，3*5*7

$T \rightarrow FT'$	$T'.i = F.val$
	$T.val = T'.s$
$T' \rightarrow *FT'$	$T_1'.i = T'.i \times F.val$
	$T'.s = T_1'.s$
$T' \rightarrow \varepsilon$	$T'.s = T'.i$
$F \rightarrow digit$	$F.val = digit.lexval$



例如，3*5*7

$T \rightarrow FT'$	$T'.i = F.val$
	$T.val = T'.s$
$T' \rightarrow *FT'$	$T_1'.i = T'.i \times F.val$
	$T'.s = T_1'.s$
$T' \rightarrow \varepsilon$	$T'.s = T'.i$
$F \rightarrow$ digit	$F.val =$ digit.lexval



注释语法分析树

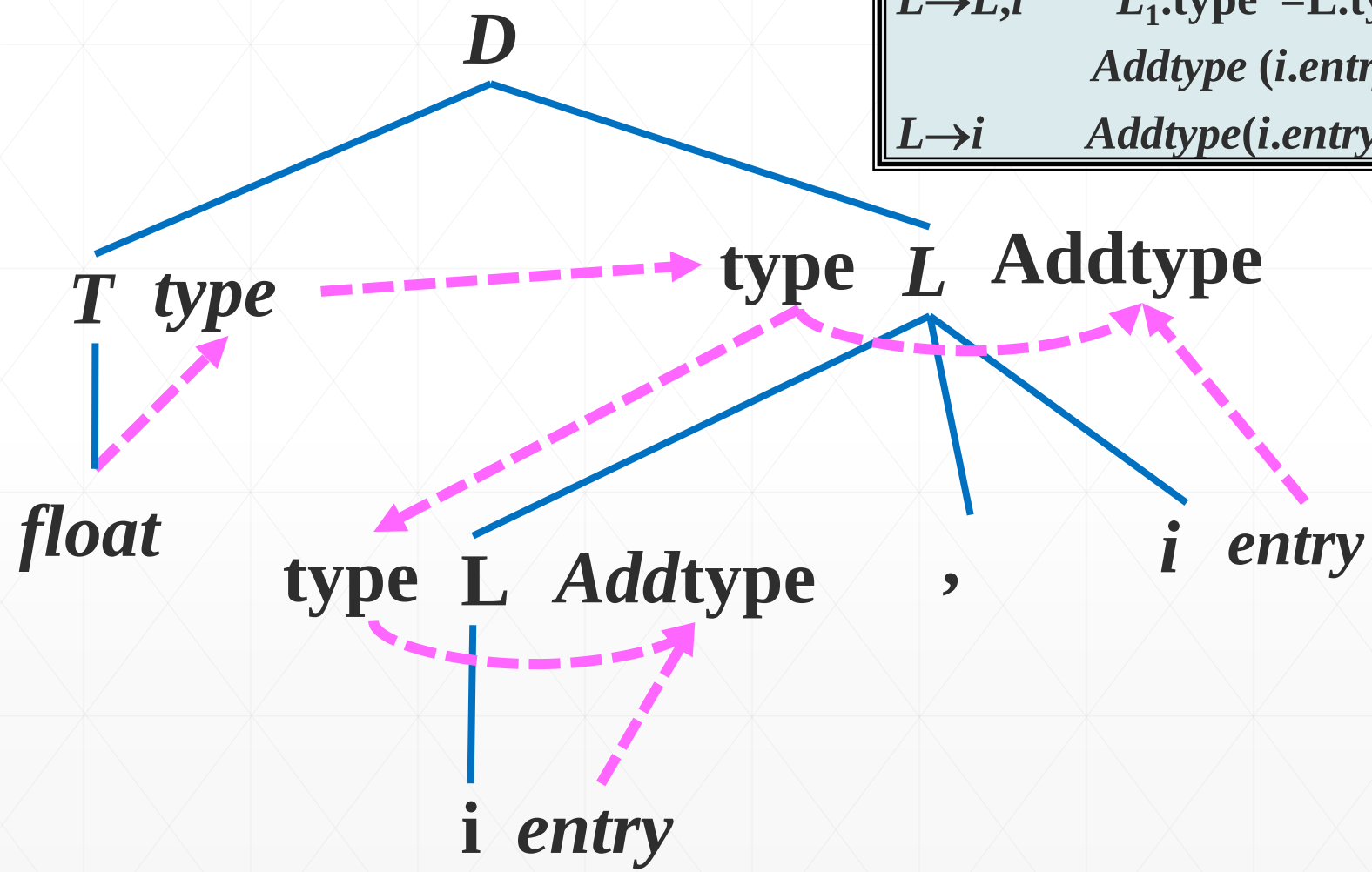
简单类型说明语句的SDD

产生式	语义规则
$D \rightarrow T\ L$	$L.type = T.type$
$T \rightarrow int$	$T.type = int$
$T \rightarrow float$	$T.type = float$
$L \rightarrow L, i$	$L_1.type = L.type$
	$Addtype\ (i.entry, L.type)$
$L \rightarrow i$	$Addtype(i.entry,$ L.type$)$

L属性

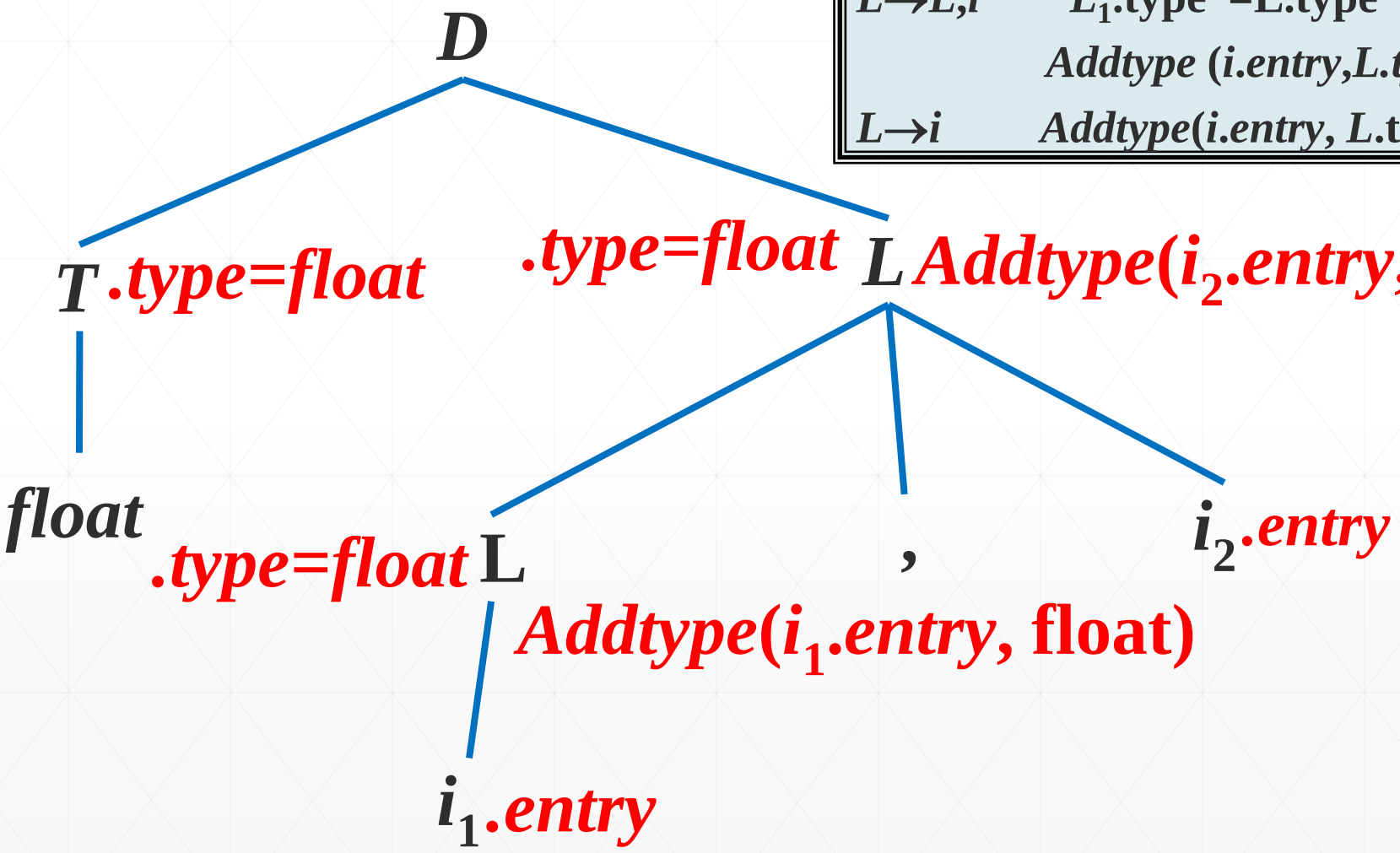
例如，float i_1，i_2

$D \rightarrow TL$	$L.type = T.type$
$T \rightarrow float$	$T.type = float$
$L \rightarrow L,i$	$L_1.type = L.type$
	$Addtype(i.entry, L.type)$
$L \rightarrow i$	$Addtype(i.entry, L.type)$



例如，float i_1，i_2

$D \rightarrow TL$	$L.type = T.type$
$T \rightarrow float$	$T.type = float$
$L \rightarrow L,i$	$L_1.type = L.type$ $Addtype(i.entry, L.type)$
$L \rightarrow i$	$Addtype(i.entry, L.type)$



注释语法分析树

6.1 语法制导翻译

6.1.1 语法制导定义

6.1.2 SDD的求值顺序

6.1.3 语法制导翻译的应用

6.1.4 语法制导的翻译方案

6.1.5 实现L属性的SDD

教材：龙书

抽象语法树的构造

- 抽象语法树（简称语法树）

翻译的一种中间代码

表达式的抽象语法树，内部节点为运算符，子节点为运算符相关的运算分量

一般结构，可以创建一个针对该结构的运算符，与该结构相关的组成部分作为运算分量。

- 语法分析树-语法分析的结果表示

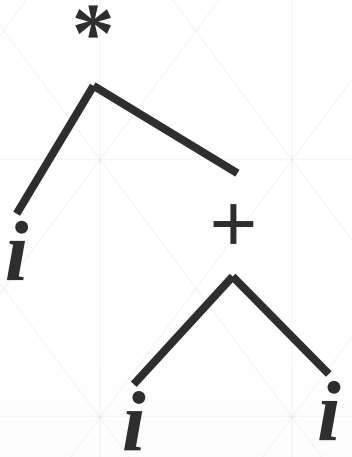
内部节点为非终结符号

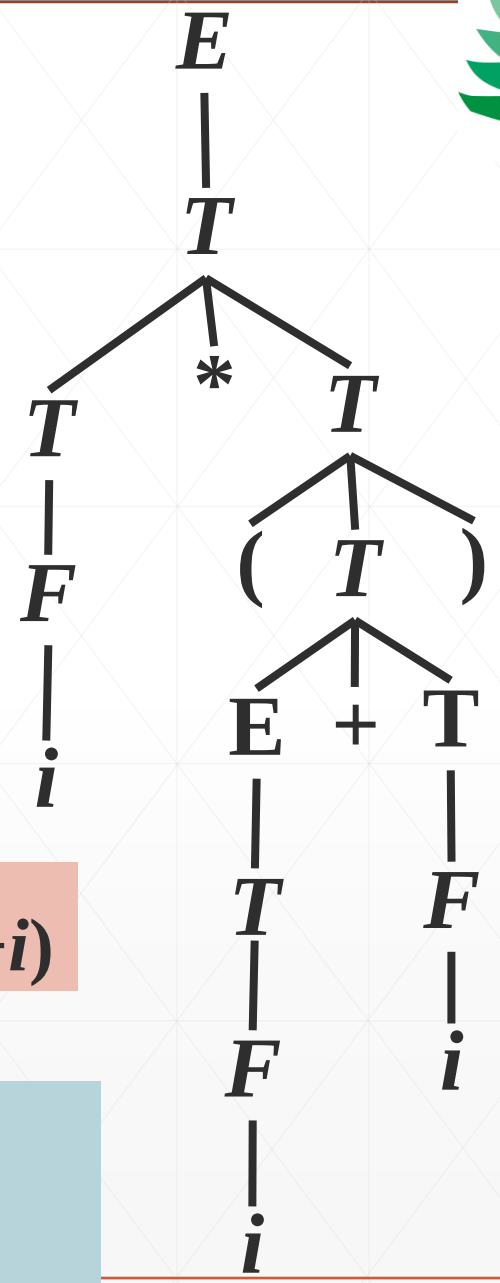
简单算术表达式文法G(*E*)：

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid i$





$i*(i+i)$

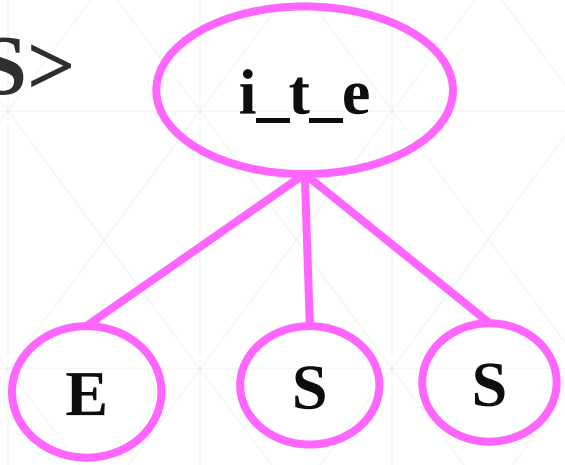
语法分析树庞大的原因：

每步推导都在语法树有反映。

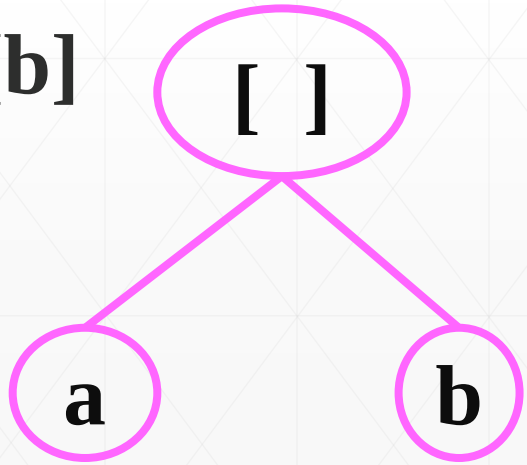
语法分析与语义处理及中间代码生成合并成一遍处理的原因

一般结构对应的抽象语法树例子

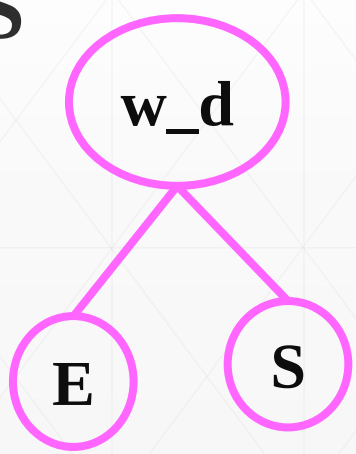
if<E>then<S>else<S>



a[b]



while<E>S



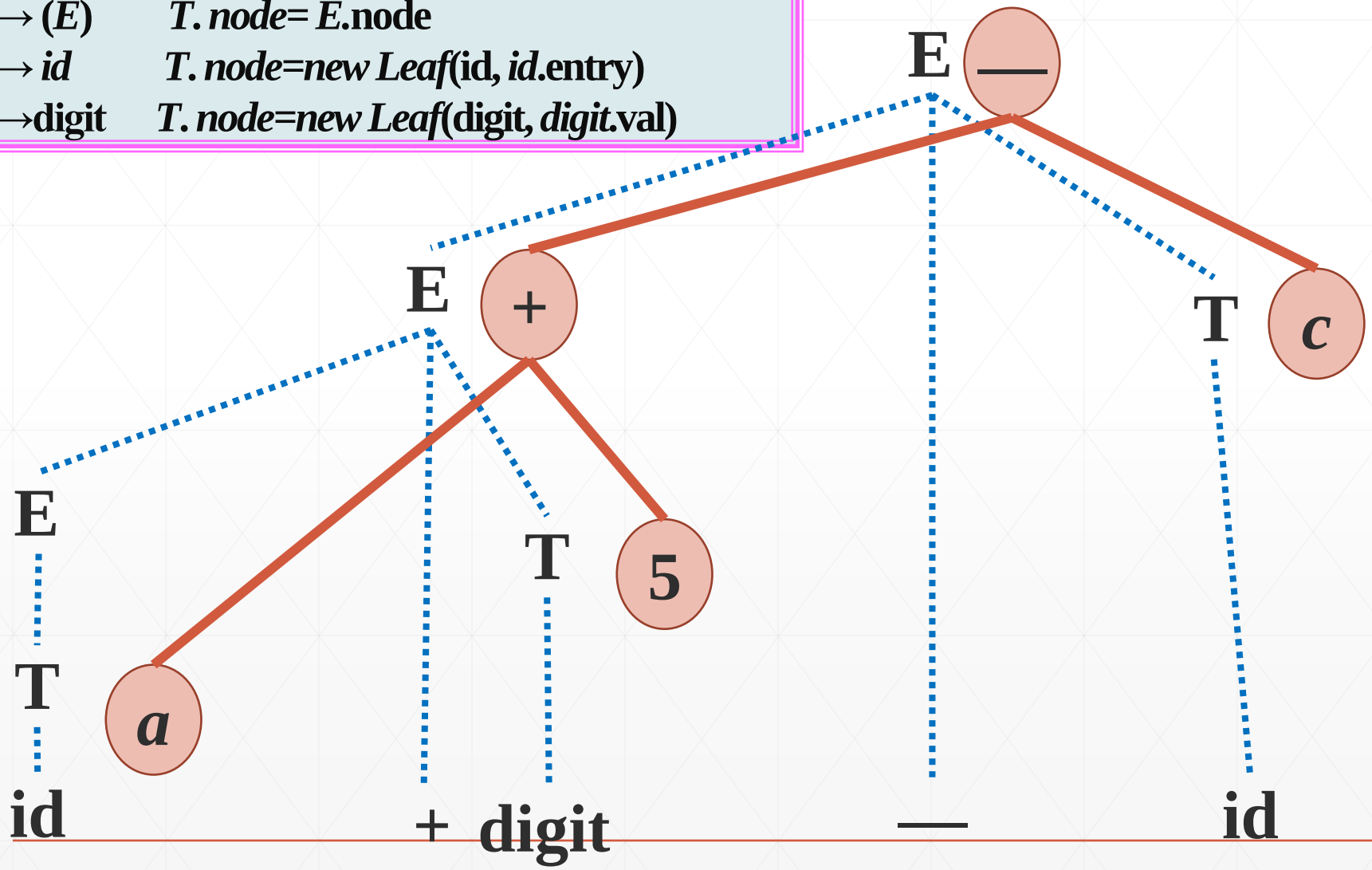
简单表达式构造语法树的SDD

产生式	语义规则
$E \rightarrow E + T$	$E.node = new\ Node('+', E_1.node, T.node)$
$E \rightarrow E - T$	$E.node = new\ Node('-', E_1.node, T.node)$
$E \rightarrow T$	$E.node = T.node$
$T \rightarrow (E)$	$T.node = E.node$
$T \rightarrow id$	$T.node = new\ Leaf(id, id.entry)$
$T \rightarrow digit$	$T.node = new\ Leaf(digit, digit.val)$

S属性

$E \rightarrow E + T$	$E.node = new\ Node('+', E_1.node, T.node)$
$E \rightarrow E - T$	$E.node = new\ Node('-', E_1.node, T.node)$
$E \rightarrow T$	$E.node = T.node$
$T \rightarrow (E)$	$T.node = E.node$
$T \rightarrow id$	$T.node = new\ Leaf(id, id.entry)$
$T \rightarrow digit$	$T.node = new\ Leaf(digit, digit.val)$

例：$a + 5 - c$



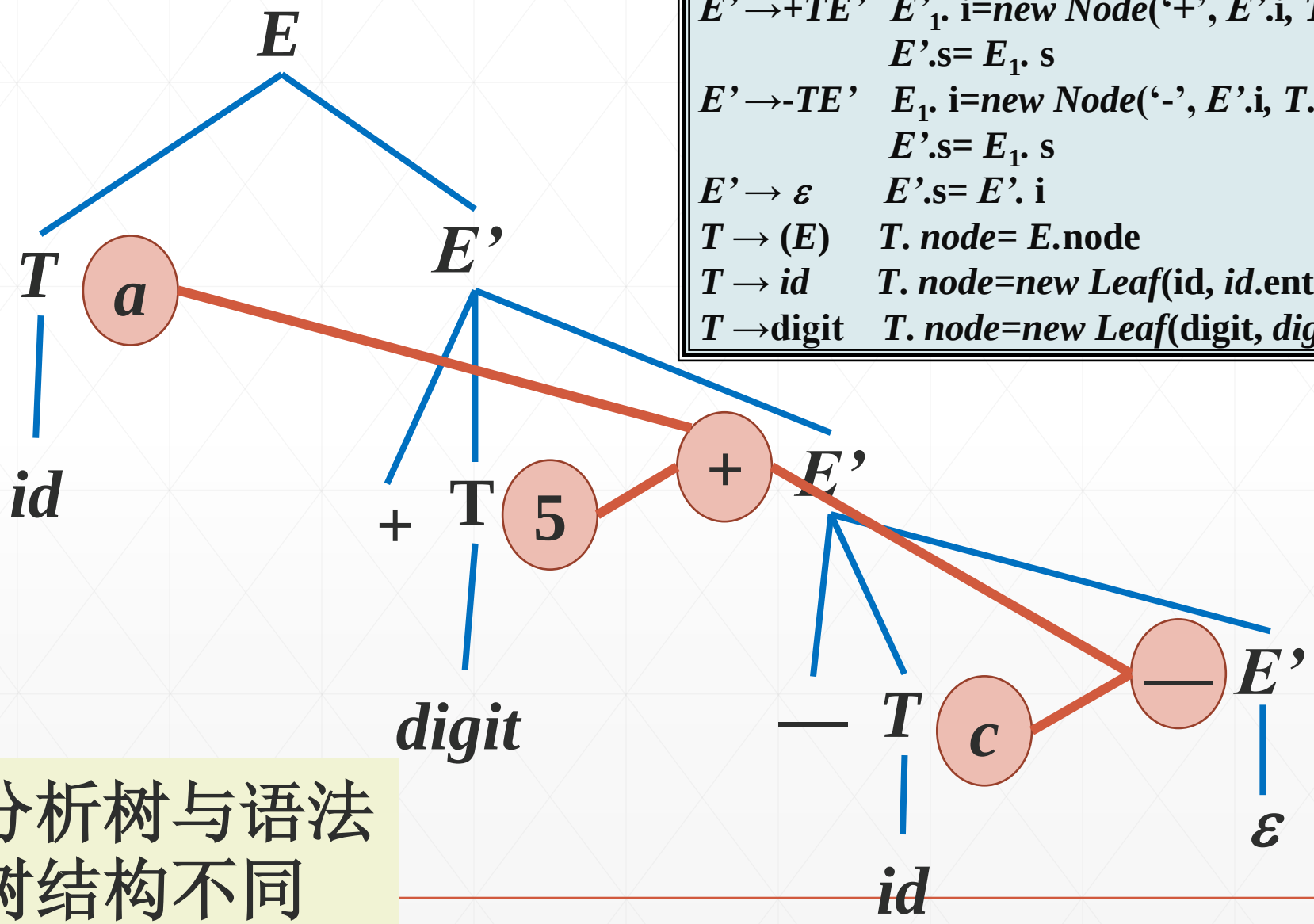
简单表达式构造语法树的SDD

产生式	语义规则
$E \rightarrow T\ E'$	$E.node = E'.s$
	$E'.i = T.node$
$E' \rightarrow +TE'$	$E_1'.i = new\ Node('+', E'.i, T.node)$
	$E'.s = E_1'.s$
$E' \rightarrow -TE'$	$E_1'.i = new\ Node('-', E'.i, T.node)$
	$E'.s = E_1'.s$
$E' \rightarrow \varepsilon$	$E'.s = E'.i$
$T \rightarrow (E)$	$T.node = E.node$
$T \rightarrow id$	$T.node = new\ Leaf(id, id.entry)$
$T \rightarrow digit$	$T.node = new\ Leaf(digit, digit.val)$

L属性

例:a+5-c

$E \rightarrow T\ E'$	$E.node = E'.s$
	$E'.i = T.node$
$E' \rightarrow +TE'$	$E'_1.i = new\ Node('+', E'.i, T.node)$
	$E'.s = E_1.s$
$E' \rightarrow -TE'$	$E_1.i = new\ Node('-', E'.i, T.node)$
	$E'.s = E_1.s$
$E' \rightarrow \varepsilon$	$E'.s = E'.i$
$T \rightarrow (E)$	$T.node = E.node$
$T \rightarrow id$	$T.node = new\ Leaf(id, id.entry)$
$T \rightarrow digit$	$T.node = new\ Leaf(digit, digit.val)$



分析树与语法树结构不同

6.1 语法制导翻译

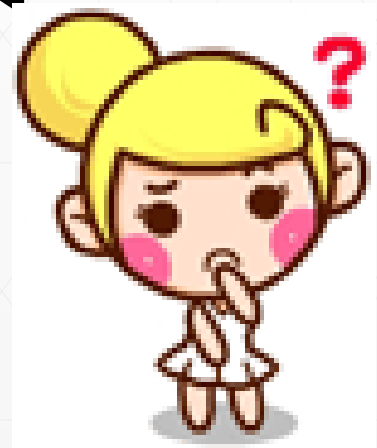
6.1.1 语法制导定义

6.1.2 SDD的求值顺序

6.1.3 语法制导翻译的应用

6.1.4 语法制导的翻译方案

6.1.5 实现L属性的SDD



教材：龙书

语法制导的翻译方案(Syntax-directed translation scheme,SDT)

语法制导定义的补充

给出SDD中的属性计算或语义动作的可行实现方案

SDD→SDT的具体处理：

把属性计算或语义动作用“{}”嵌入在产生式中，嵌入的位置表示相对应的“计算时间”。

SDT的实现方法

一、首先建立一棵嵌入了属性计算与语义动作的语法分析树，然后从左到右深度优先顺序执行这些动作。

任何SDT

二、语法分析过程中实现，不用构造出语法分析树。

部分SDT

判断SDT能否在语法分析过程中实现的方法

将内嵌的语义动作替换为一个独有的标记V_N符号，标记V_N符号只有一个产生式“标记V_N符号$\rightarrow\varepsilon$”；

若带有标记V_N符号的文法可以使用某种方法进行语法分析，则该SDT就可以在语法分析过程中实现。

后缀翻译方案

文法可以用自下而上的语法分析器分析，

SDD是S属性的。

SDT：

把语义动作(属性计算)都放在产生式的最后，

在使用产生式归约时执行对应的语义动作。

后缀SDT

简单运算的台式计算器的SDT

$L \rightarrow E$; {*print*(E .val)}

$E \rightarrow E + T$ {E .val = E_1 .val + T .val}

$E \rightarrow T$ {E .val = T .val}

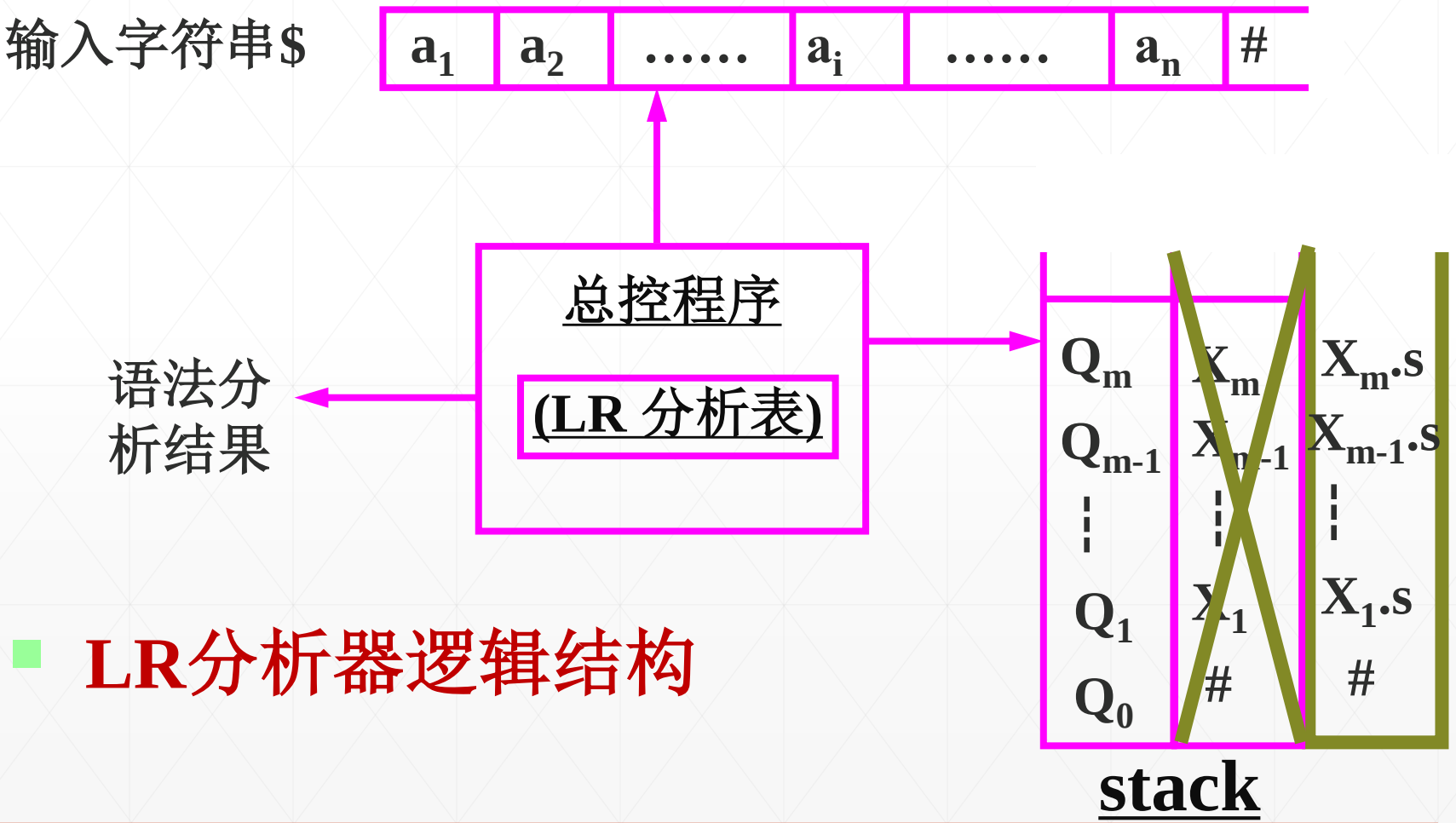
$T \rightarrow T * F$ {T .val = T_1 .val $\times$ F .val}

$T \rightarrow F$ {T .val = F .val}

$F \rightarrow (E)$ {F .val = E .val}

$F \rightarrow$ digit {F .val = digit .lexval}

后缀SDT的语法分析栈实现



LR分析器逻辑结构

简单运算的台式计算器的LR分析的SDT

$L \rightarrow E$; {输出stack[top-1];top=top-1}

$E \rightarrow E + T$

{stack[top-2]=stack[top-2]+stack[top];top=top-2}

$E \rightarrow T$

$T \rightarrow T * F$

{stack[top-2]=stack[top-2]×stack[top];top=top-2}

$T \rightarrow F$

$F \rightarrow (E)$ {stack[top-2]=stack[top-1];top=top-2}

$F \rightarrow$ digit

产生式内部带有语义动作

语义动作的合理执行时间

例：$D \rightarrow T\{L.\text{type}=T.\text{type}\}L$

语法分析过程中执行语义动作的时间：

1.自下而上的分析，归约出T时，执行{}中语义

2.自上而下的分析，在L替换之前，执行{}中语义

不能在语法分析过程中实现的SDT

例：中缀到前缀翻译的SDT

$E \rightarrow \{print(`+')\} E + T$

$E \rightarrow T$

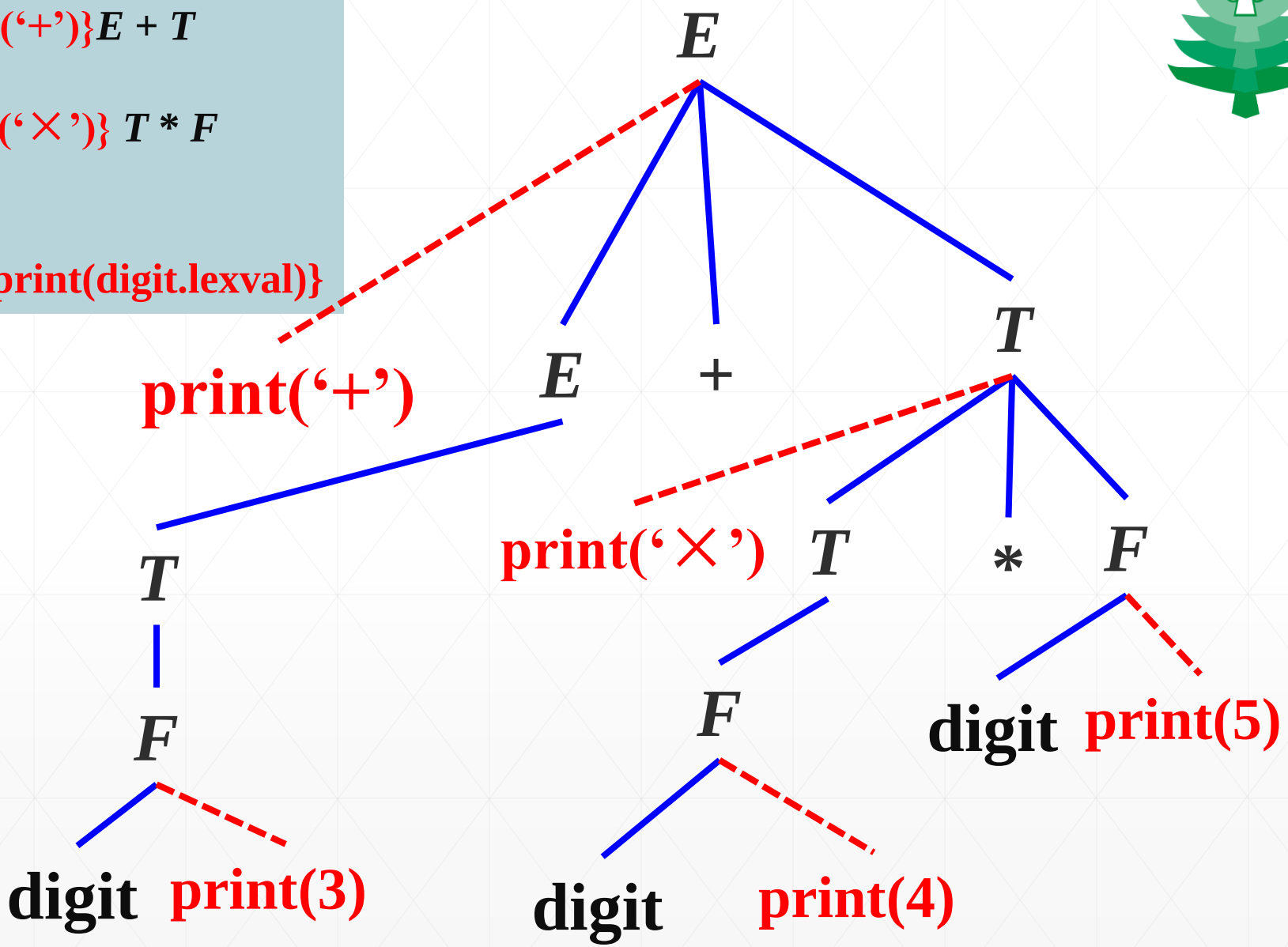
$T \rightarrow \{print(`\times')\}\ T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow digit\{print(digit.lexval)\}$

$E \rightarrow \{print('+')\} E + T$
$E \rightarrow T$
$T \rightarrow \{print('\times')\} T * F$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow digit\{print(digit.lexval)\}$



3+4*5的嵌入翻译动作的语法分析树

不能在语法分析过程中实现的SDT

左递归，不能用自上而下的分析器分析

例:中缀到前缀翻译的SDT

$E \rightarrow$ {print('+')} $E + T$

$E \rightarrow T$

$T \rightarrow$ {print('×')} $T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow$ digit {print(digit .lexval)}

文法修改为:

$E \rightarrow ME + T$

$M \rightarrow \varepsilon$ {print('+')}

$E \rightarrow T$

$T \rightarrow NT * F$

$N \rightarrow \varepsilon$ {print('×')}

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow$ digit {print(digit .lexval)}

I_0={$E'\rightarrow .E$,#
$E\rightarrow .ME + T|.T$,#
$M\rightarrow .$, (/digit
$T\rightarrow .NT * F|.F$,#
$N\rightarrow .$, (/digit
$F\rightarrow .(E)|.$digit ,#
}

$E\rightarrow ME + T$
$M\rightarrow \varepsilon${print('+')}
$E\rightarrow T$
$T\rightarrow NT * F$
$N\rightarrow \varepsilon${print('×')}
$T\rightarrow F$
$F\rightarrow (E)$
$F\rightarrow$digit{print(digit .lexval)}

$M\rightarrow .$与$N\rightarrow .$归约-归约冲突，
不能用LR(0)分析器分析

不能用
LR(1)分析
器分析

Follow(*M*)={(, digit}
Follow(*N*)={(, digit}
不能用*SLR*(1)归约原则解决冲突，
不能用SLR(1)分析器分析

L属性定义的翻译方案

文法可以用自上而下的语法分析器分析

SDD是L属性的

SDT：

1.把继承属性的语义动作插入在符号出现之前，分析开始前执行语义动作。

若有多个继承属性，按拓扑排序排列。

2.综合属性放在产生式的最后，分析完成后执行语义动作。

从SDT中消除左递归

左递归文法不能用自上而下的分析

文法消除左递归的同时语义动作的处理

第一种情况： 翻译结果只与语义动作的执行顺序有关

处理办法： 转换文法的时候，动作当做终结符号

例：$E \rightarrow E+T\{\text{print}(\text{'+'})\}$

$E \rightarrow T$

等价变换为：

$E \rightarrow TE'$

$E' \rightarrow +T\{\text{print}(\text{'+'})\}E'$

$E' \rightarrow \varepsilon$

从SDT中消除左递归

第二种情况：语义动作涉及属性计算

下面给出的处理办法适用条件：

单个递归产生式

单个非递归产生式

左递归非终结符号只有单个属性

从SDT中消除左递归

例：$A \to AY\{A.a=g(A_1.a,Y.y)\}$

$A \to X\ \{A.a=f(X.x)\}$

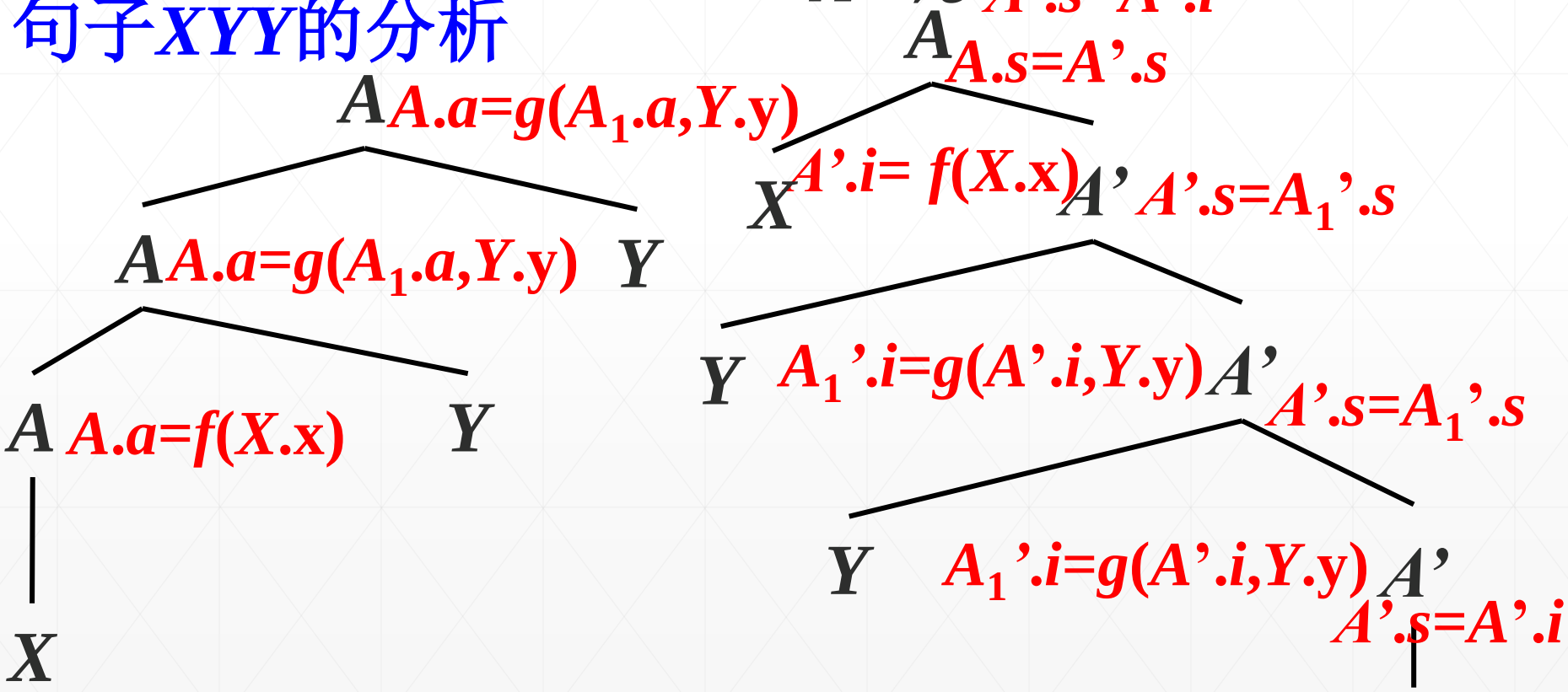
消除左递归：

$A \to XA'$ $A.s=A'.s$
$\{A'.i=f(X.x)\}$

$A' \to Y\ A'$ $A'.s=A_1'.s$
$A_1'.i=g(A'.i,Y.y)$

$A' \to \varepsilon$ $A'.s=A'.i$

句子XYY的分析



从SDT中消除左递归

例：$A \rightarrow AY\{A.a=g(A_1.a,Y.y)\}$

$A \rightarrow X\ \{A.a=f(X.x)\}$

消除左递归：

$A \rightarrow X\ \{A'.i=f(X.x)\}\ A'\{A.s=A'.s\}$

$A' \rightarrow Y\{A_1'.i=g(A'.i,Y.y)\}\ \ A'\{A'.s=A_1'.s\}$

$A' \rightarrow \varepsilon\ \{A'.s=A'.i\}$

左递归文法修改后的语义处理

$E \to E + T$ {E .val = E_1 .val + T .val}
$E \to T$ {E .val = T .val}
$T \to T * F$ {T .val = T_1 .val × F .val}
$T \to F$ {T .val = F .val}
$F \to (E)$ {F .val = E .val}
$F \to$ digit {F .val = digit .lexval}

E'有两个属性值：
继承：前面的结果
综合：最终的结果

$E \to TE'$
$E' \to +TE' | \varepsilon$
$T \to FT'$
$T' \to *FT' | \varepsilon$
$F \to (E)$
$F \to$ digit

只考虑加法的情况

$E \to T$ {E'.i=T.val}
E' {E.val=E'.s}
$E' \to +T$ {E'_1.i=E'.i+T.val}
E' {E'.s=E'_1.s}
$E' \to \varepsilon$ {E'.s=E'.i}

左递归文法修改后的语义处理

$E \to TE'$

$E' \to +TE' \mid \varepsilon$

$T \to FT'$

$T' \to *FT' \mid \varepsilon$

$F \to (E)$

$F \to \mathbf{digit}$

E'和T'有两个属性值：

继承：前面的结果

综合：最终的结果

$E \to T\{E'.i=T.val\}$

$E'\{E.val=E'.s\}$

$E' \to +T\{E'_1.i=E'.i+T.val\}$

$E'\{E'.s=E'_1.s\}$

$E' \to \varepsilon\ \{E'.s=E'.i\}$

$T \to F\{T'.i=F.val\}$

$T'\ \{T.val=T'.s\}$

$T' \to *F\{T'_1.i=T'.i \times F.val\}$

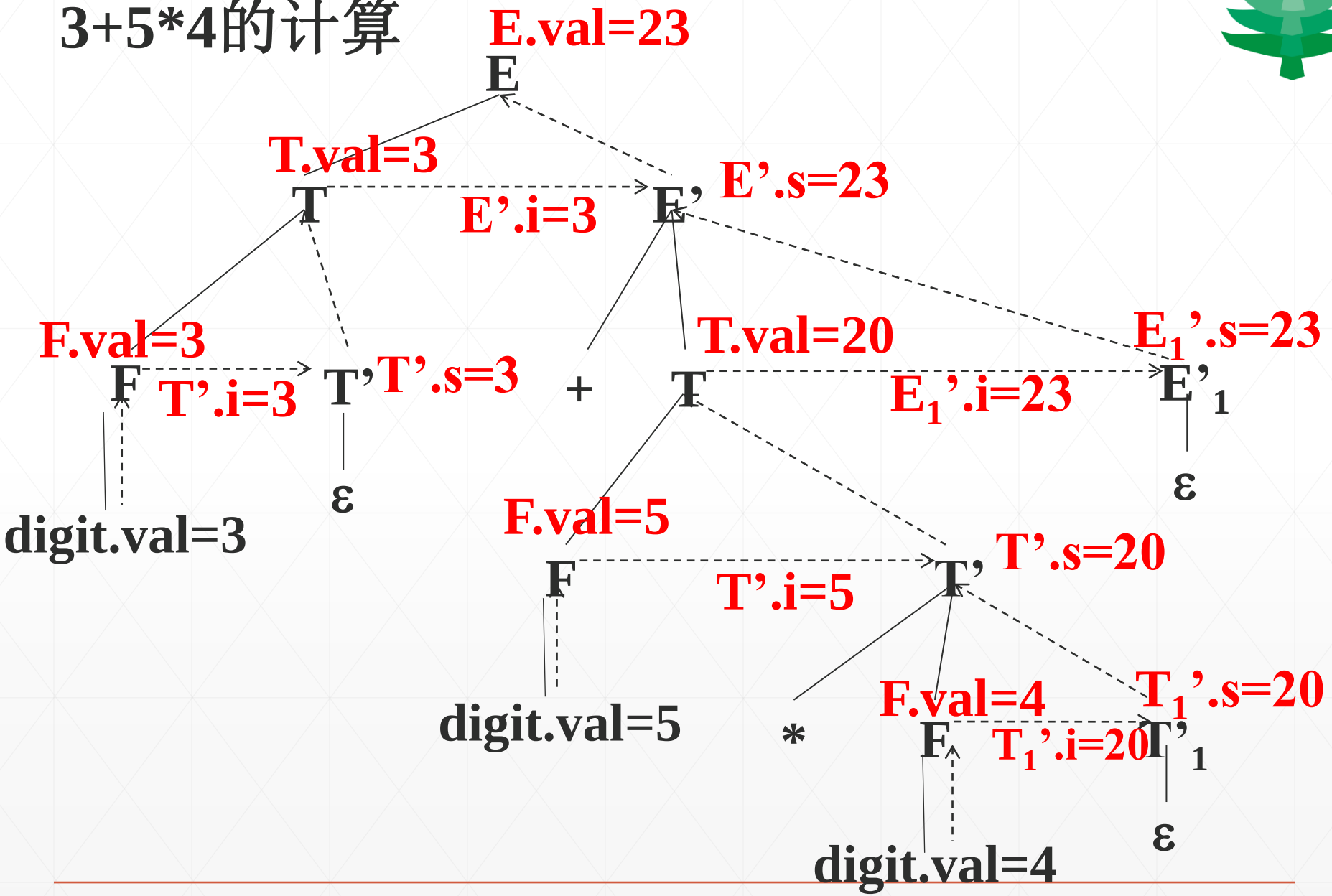
$T'\ \{T'.s=T'_1.s\}$

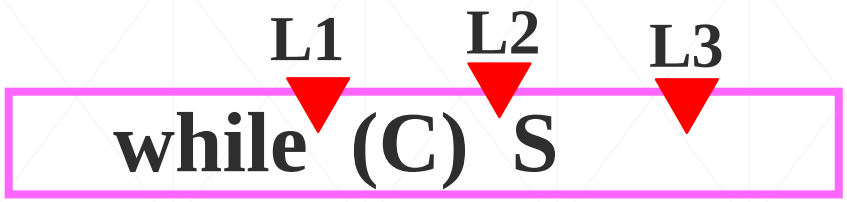
$T' \to \varepsilon\{T'.s=T'.i\}$

$F \to (E)\ \{F.val=E.val\}$

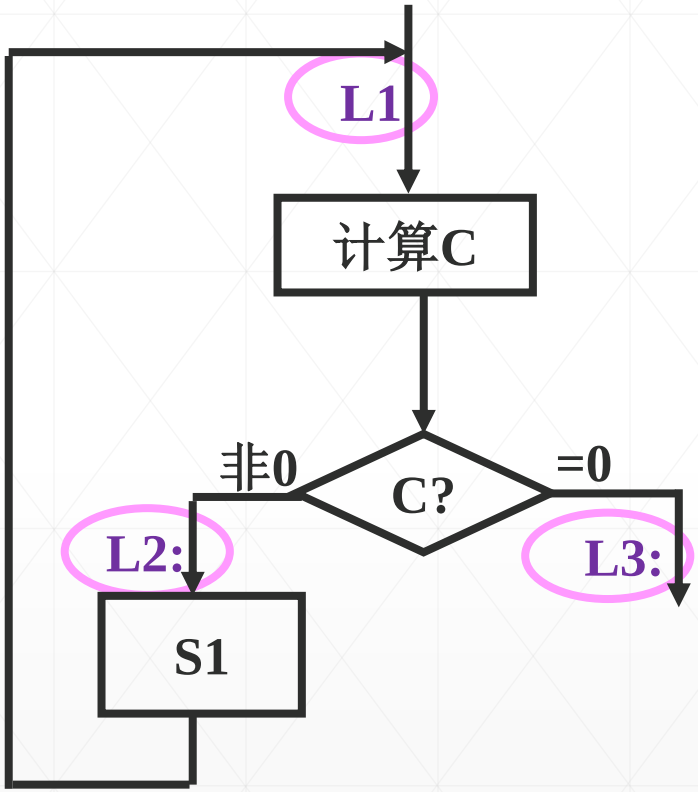
$F \to \mathbf{digit}\{F.val=digit.val\}$

3+5*4的计算


E.val=23
E
T.val=3
T
E'.i=3
E'
E'.s=23
F.val=3
F
T'.i=3
T'
T'.s=3
ε
digit.val=3
+
T.val=20
T
E1'.s=23
E1'.i=23
E'1
ε
F.val=5
F
T'.i=5
T'
T'.s=20
digit.val=5
*
F.val=4
F
T1'.i=20
T'1
T1'.s=20
ε
digit.val=4



产生式	语义规则
S→while(C)S	L1=newlabel() L2=newlabel() S_1.next=L1 C.false=S.next C.true=L2 S.code=label‖L1 ‖C.code ‖label‖L2 ‖S_1.code



文法“S→ while (C) S”的SDT

S→while({L1=newlabel()
L2=newlabel()
C.false=S.next
C.true=L2}
C){S_1.next=L1}
S {S.code=label||L1||C.code||label||L2||
S_1.code}

6.1 语法制导翻译

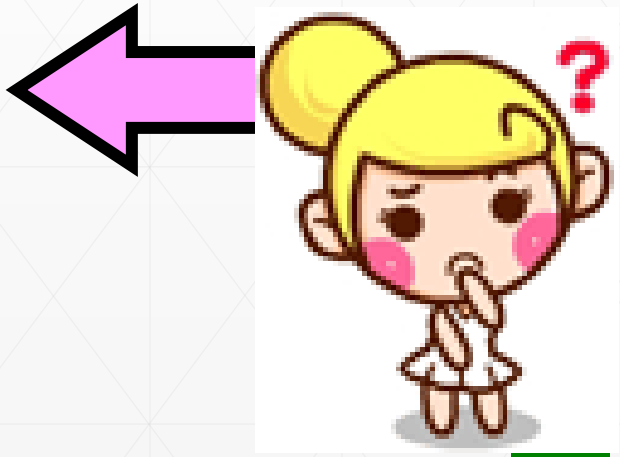
6.1.1 语法制导定义

6.1.2 SDD的求值顺序

6.1.3 语法制导翻译的应用

6.1.4 语法制导的翻译方案

6.1.5 实现L属性的SDD



参考教材：龙书

L属性的SDT的实现方法

一、在递归下降分析器中加入语义动作的处理

二、在LL分析器中加入语义动作的处理

三、在LR分析器中加入语义动作的处理

一、在递归下降分析器中加入语义动作的处理

每个非终结符号的产生式函数处理方式

1.递归下降分析函数的参数是继承属性

2.函数的返回值是综合属性

假设条件

所有必要的属性值都保存在了局部变量中

```
string S(label next){
    string Scode,Ccode;
    label L1,L2;
    if(当前单词==while){
      读下一个单词;
     if(当前单词==‘(‘){
       读下一个单词;
       L1=newlabel();
       L2=newlabel();
       Ccode=C(next,L2);
       if(当前单词==‘)‘){
         读下一个单词;
         Scode=S(L1);
         return(“label”||L1||Ccode||”label”||L2||Scode);}
       else... else…}else…}}
```

S→while({L1=newlabel()
L2=newlabel()
C.false=S.next
C.true=L2}
C){S_1.next=L1}
S {S.code=label||L1||C.code||label||L2||
S_1.code}

每个函数处理返回一段代码，返回语句不停的复制代码

逐步把代码片段添加到一个数组或输出文件

假设条件

以非终结符号*A*对应的代码为例

*A*的代码包含有*A*的侯选式中符号对应的代码，那么在*A*的代码中，侯选式中符号的代码出现的顺序跟他们在侯选式中的顺序一致。

```
void S(label next){
    label L1,L2;
    if(当前单词==while){
        读下一个单词;
        if(当前单词==‘(‘){
            读下一个单词;
            L1=newlabel();
            L2=newlabel();
            print(“label:”,L1);
            C(next,L2);
            if(当前单词==‘)‘){
                读下一个单词;
                print(“label:”,L2);
                S(L1);}
            else... else…}else…}}
```

S→while({L1=newlabel()
L2=newlabel()
C.false=S.next
C.true=L2}
C){S_1.next=L1}
S {S.code=label||L1||C.code||label||L2||
S_1.code}

S→while({L1=newlabel()
L2=newlabel()
C.false=S.next
C.true=L2
print(“label:”,L1)}
C){S_1.next=L1; print(“label:”,L2)}
S

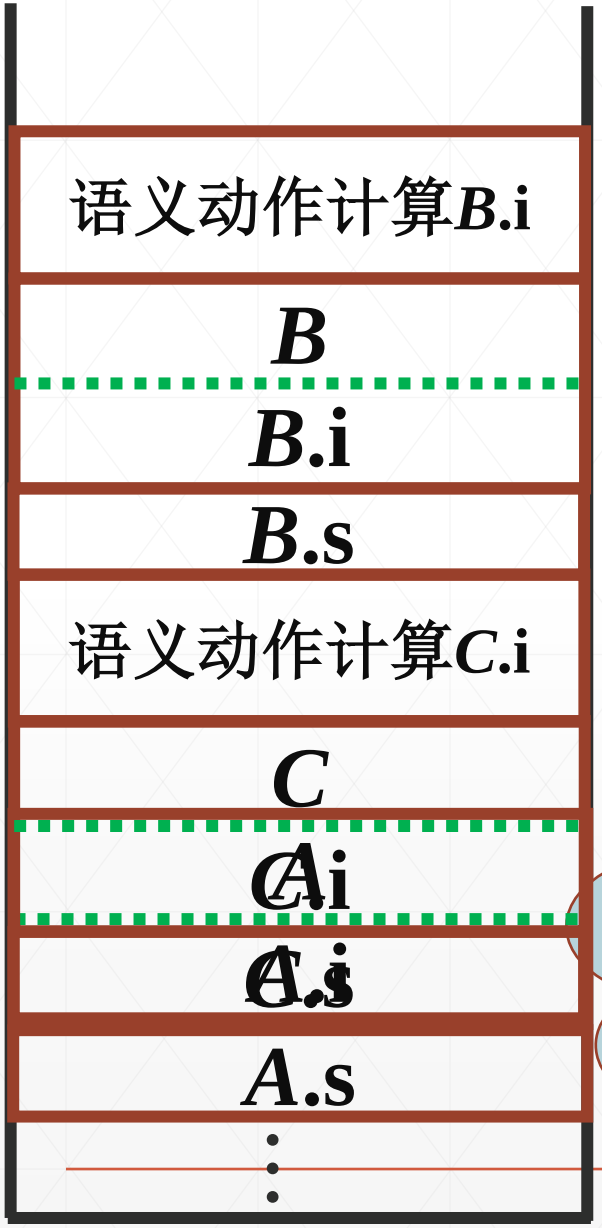
二、在LL分析器中加入语义动作的处理

在分析栈中，增加存储语义动作和属性求值的数据项

继承属性放在符号的记录中，属性求值的代码放在紧靠符号记录之上的动作记录中

综合属性放在紧靠符号记录之下的单独记录中

根据语义动作在SDT中的位置,放置在栈中合适的位置



$A \rightarrow BC$

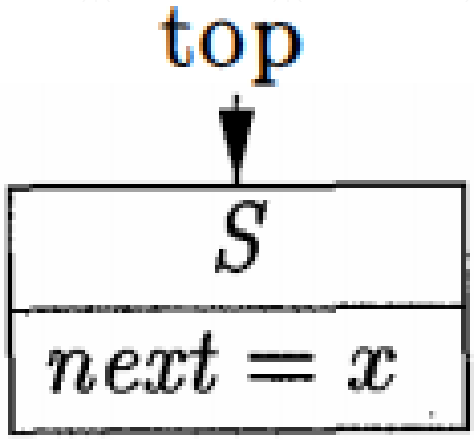
若产生式关联有属性计算

$C.i=f(A.i,B.i,B.s,)$

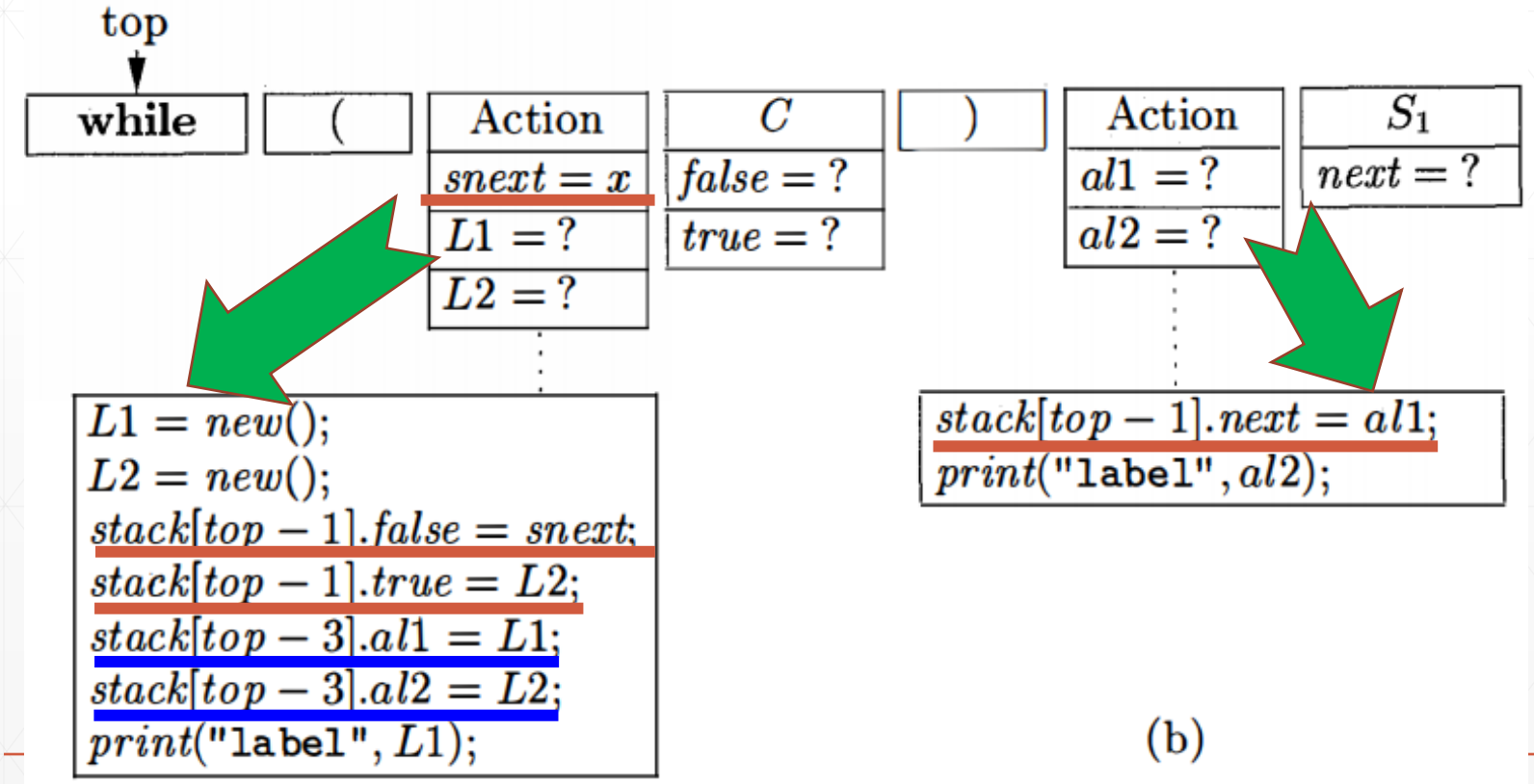
A弹出前要复制其继承属性到栈中

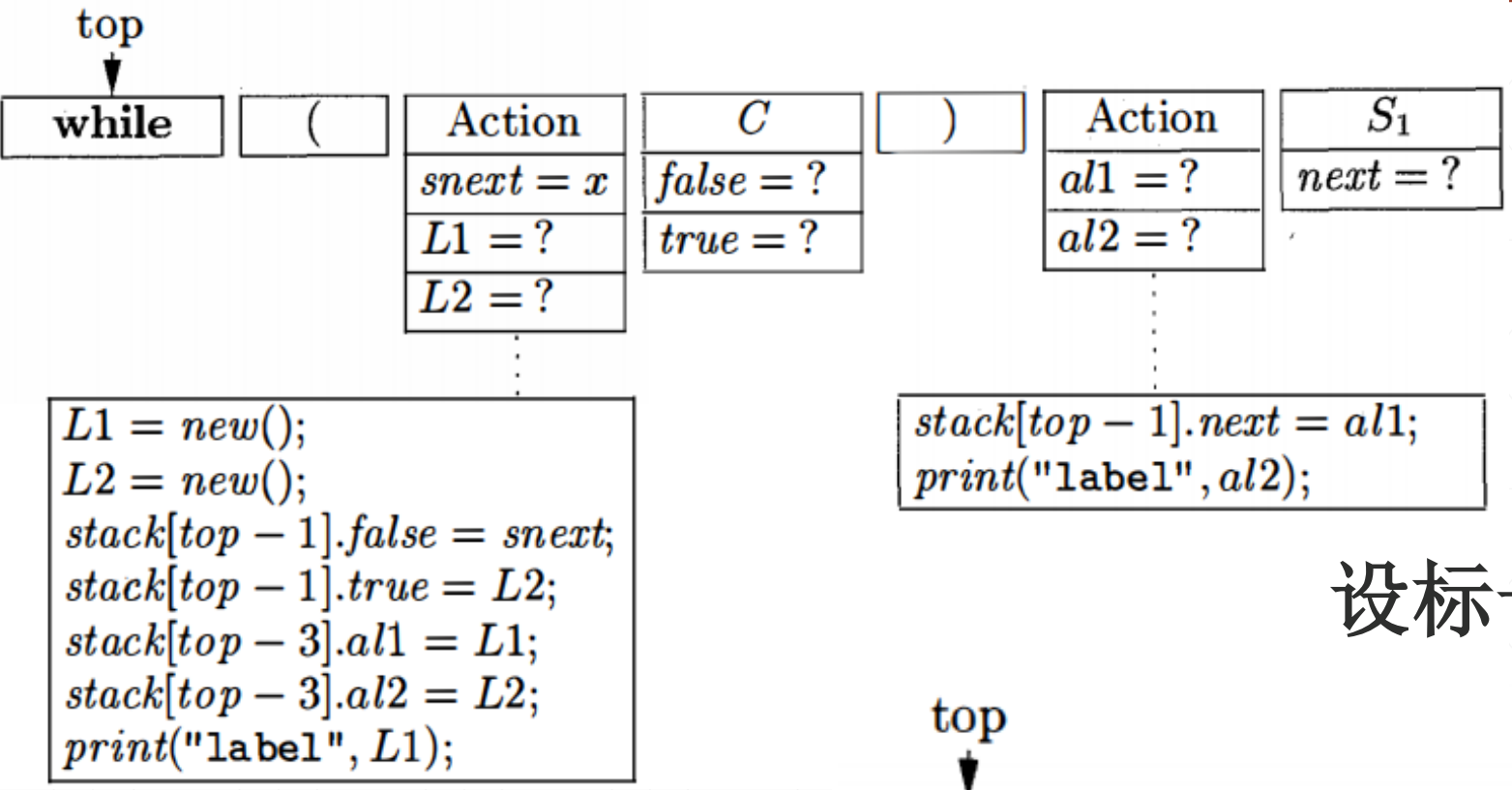
B弹出前要复制其继承属性到C关联的动作中

B.s弹出前同样处理

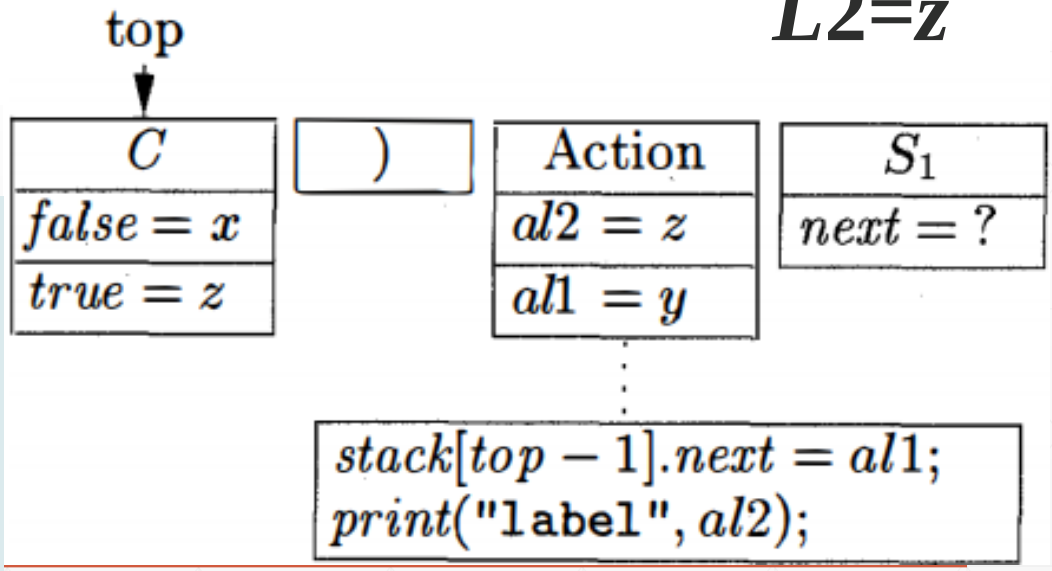


S→while({L1=newlabel()
L2=newlabel()
C.false=S.next
C.true=L2
print("label:",L1)}
C){S_1.next=L1; print("label:",L2)}
S





设标号L1=y
L2=z

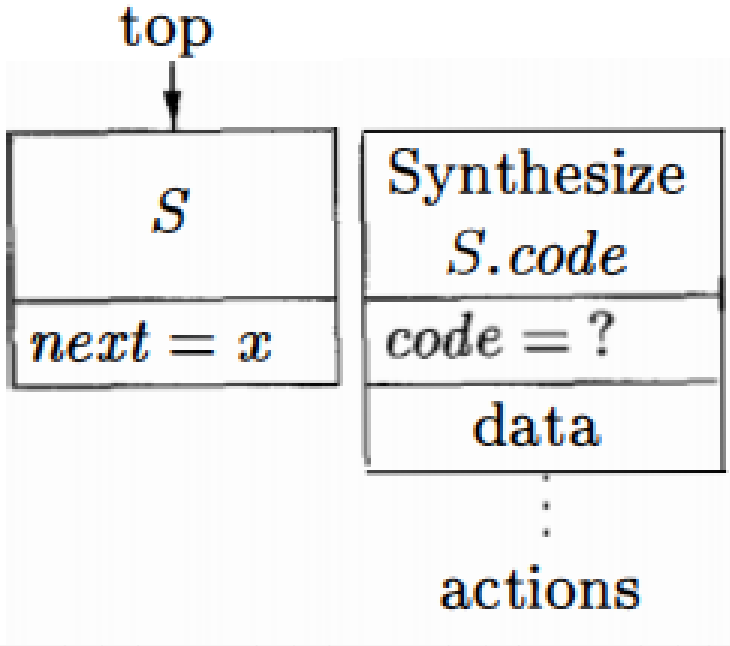


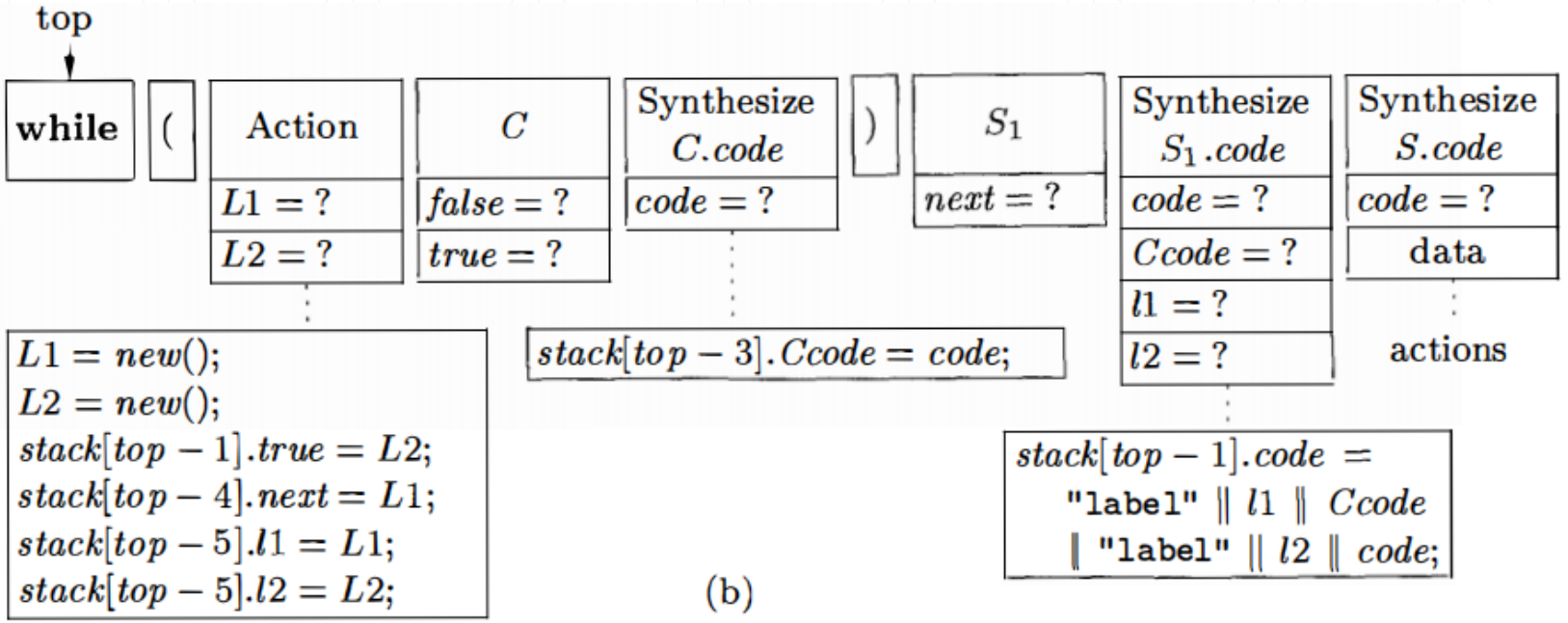
S→while({L1=newlabel()
L2=newlabel()
C.false=S.next
C.true=L2
print(“label:”,L1)}
C){S_1.next=L1; print(“label:”,L2)}
S

翻译方法把输出S.code作为一个综合属性，不是通过边扫描边处理方式生成。

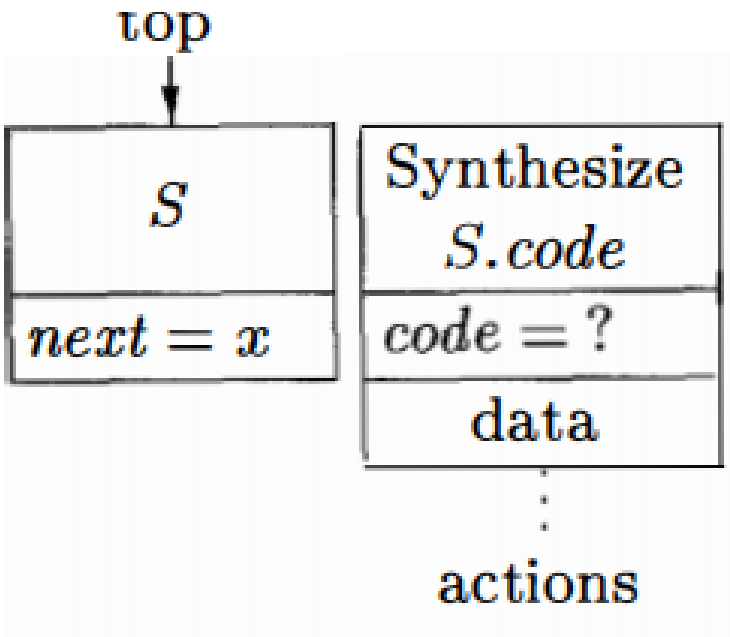
假设每个具有代码的非终结符号都把它的代码存放在栈中该符号的记录下方的综合记录中。

S→while({L1=newlabel()
L2=newlabel()
C.false=S.next
C.true=L2}
C){S_1.next=L1}
S {S.code=label||L1||C.code||label||L2||
S_1.code}





S→while({L1=newlabel()
L2=newlabel()
C.false=S.next
C.true=L2}
C){S_1.next=L1}
S {S.code=label||L1||C.code||label||L2||
S_1.code}



LR文法上的L属性SDD可以在语法分析的同时完成语义处理吗？

以$A \rightarrow BC$为例

$B.i=f(A.i)$

结论：一般不能

L属性的LL(1)文法的SDD的自下而上语法分析

修改文法为S属性文法

修改文法方法

构造SDT

内嵌的语义动作，引入标记非终结符号

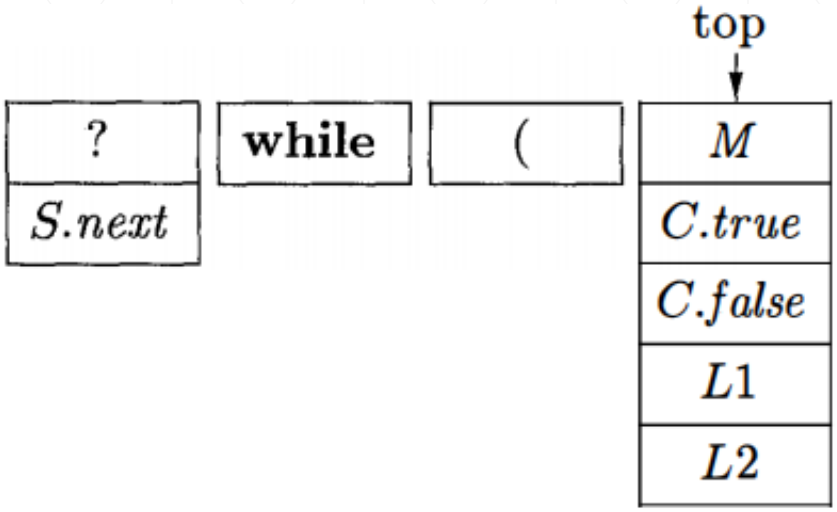
内嵌的语义动作，作为标记非终结符号的综合属性

S→while({L1=newlabel()
L2=newlabel()
C.false=S.next
C.true=L2}
C){S_1.next=L1}
S {S.code=label||L1||C.code||label||L2||
S_1.code}

S→while(MC)NS
{S.code=label||L1||C.code||label||L2||S_1.code}

M→ε{L1=newlabel()
L2=newlabel()
C.false=S.next
C.true=L2}

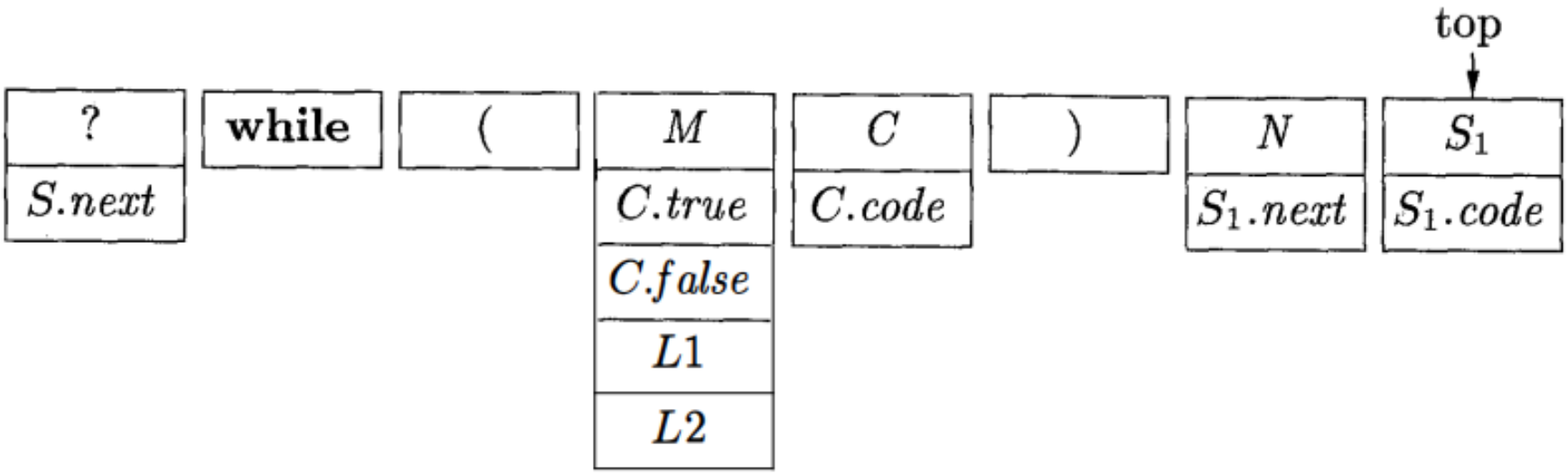
N→ε{S_1.next=L1}



Code executed during reduction of ϵ to M

$L1 = new();$
$L2 = new();$
$C.true = L2;$
$C.false = stack[top - 3].next;$

S→while(MC)NS {S.code=label||L1||C.code||label||L2||S_1.code}
M→ε{L1=newlabel()
L2=newlabel()
C.false=S.next
C.true=L2}
N→ε{S_1.next=L1}



S→while(MC)NS {S.code=label||L1||C.code||label||L2||S_1.code}

M→ε{L1=newlabel()

L2=newlabel()

C.false=S.next

C.true=L2}

N→ε{S_1.next=L1}