计算机学院（软件学院）
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Compilation Principle
编 译 原 理

第17讲：语义分析(3)

张献伟

xianweiz.github.io

DCS290, 4/20/2023

中山大學
SUN YAT-SEN UNIVERSITY

Review Questions

- Why context analysis is not performed in parsing stage?
  Parsing relies on CFG, which is context free.
- What is an synthesized attribute?
  Defined by attribute values of node *N*'s children and *N* itself
- What's the usage of dependence graph?
  To decide the evaluation order of attributes.
- What is S-SDD?
  Synthesized-SDD, with only synthesized attributes.
- Is the SDD a L-SDD?

A -> X Y Z	Y.*i* = f(Z.*z*, A.*s*)

  NO. Z is right to Y, A.s is synthesized attribute.
- S-SDD is suitable for bottom-up or top-down parsing?
  Bottom-up. Natural to evaluate the parent after seeing all children.

SDD vs. SDT

- SDD[语法制导定义]: 是CFG的推广，翻译的高层次规则说明
  - A CFG grammar together with attributes and semantic rules
    - A subset of them are also called **attribute grammars**[属性文法]
      - No side effects, i.e., rules are strictly local to each production
  - Semantic rules imply no order to attribute evaluation
- SDT[语法制导翻译方案]: SDD的补充，具体翻译实施方案
  - An executable specification of the SDD
    - Fragments of programs are attached to different points in the production rules
  - The order of execution is important

Grammar	SDD	SDT
D -> T L	L.*inh* = T.*type*	D -> T { L.*inh* = T.*type* } L
T -> int	T.*type* = int	T -> int { T.*type* = int }
T -> float	T.*type* = float	T -> float { T.*type* = float }
L -> L_1, id	L_1.*inh* = L.*inh*	L -> { L_1.*inh* = L.*inh* }L_1, id
L -> id	id.*type* = L.*inh*	L -> { id.*type* = L.*inh*} id

Syntax Directed Trans. Impl.[实现]

- Learnt how to specify translation: SDD and SDT[定义]
  - SDT is an executable specification of the SDD
    - CFG with semantic actions embedded in production bodies

- SDT can be implemented in two ways[具体实现]
  - Using a parse tree or AST[基于预先构建的分析树]
    - First build a parse tree, and then apply rules or actions at each node while traversing the tree
    - All SDDs (without cycles) and SDTs can be implemented
      - Since the tree can be traversed freely, implements any ordering
  - During parsing, without building a parse tree[语法分析过程中]
    - Apply rules or actions at each production while parsing
    - Only a subset of SDDs and SDTS can be implemented
      - Evaluation ordering restricted to parser derivation order

Syntax Directed Trans. Impl. (cont.)

- Typically, SDD (i.e., semantic analysis) is implemented during parsing[更为高效]
  - Allows compiler to skip parse tree generation
  - Saves time and memory

- Two important classes of SDD's[两个关键子类]
  - SDD is *S-attributed*, the underlying grammar is *LR-parsable*
  - SDD is *L-attributed*, the underlying grammar is *LL-parsable*
  - For both classes, semantic rules in an SDD can be converted into an SDT with actions that are executed at the right time[允许SDD到SDT的转换]
    - During parsing, an action in a production body is executed as soon as all the grammar symbols to the left of the action have been matched

== Implement S-SDD ==

- Convert S-attributed SDD to SDT by[S-SDD->SDT的转换]
  - Placing each action at <u>the end of the production</u>[将每个语义动作都放在产生式的最后]
  - SDTs with all actions at the right ends of the production bodies are called **postfix SDT's**[后缀/尾部SDT]

S-SDD

Production Rules	Semantic Rules
(1) L -> E	print (E.*val*)
(2) E -> E_1 + T	E.*val* = E_1.*val* + T.*val*
(3) E -> T	E.*val* = T.*val*
(4) T -> T_1 * F	T.*val* = T_1.*val* x F.*val*
(5) T -> F	T.*val* = F.*val*
(6) F -> (E)	F.*val* = E.*val*
(7) F -> digit	F.*val* = digit.*lexval*

SDT

CFG with actions
(1) L -> E { print (E.*val*) }
(2) E -> E_1 + T { E.*val* = E_1.*val* + T.*val* }
(3) E -> T { E.*val* = T.*val* }
(4) T -> T_1 * F { T.*val* = T_1.*val* x F.*val* }
(5) T -> F { T.*val* = F.*val* }
(6) F -> (E) { F.*val* = E.*val* }
(7) F -> digit { F.*val* = digit.*lexval* }

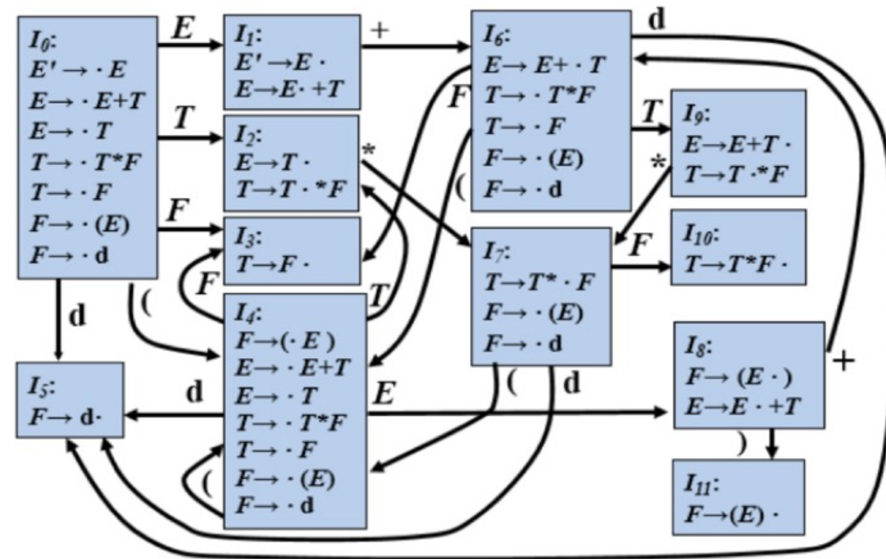
Implement S-SDD (cont.)

- If the underlying grammar of S-SDD is LR parsable
  - Then the SDT can be implemented during LR parsing
- Implement the converted SDT by[借助归约实现]
  - Executing the action along with the reduction of *head <- body*

SDT

CFG with actions
(1) L -> E { print (E.*val*) }
(2) E -> E_1 + T { E.*val* = E_1.*val* + T.*val* }
(3) E -> T { E.*val* = T.*val* }
(4) T -> T_1 * F { T.*val* = T_1.*val* x F.*val* }
(5) T -> F { T.*val* = F.*val* }
(6) F -> (E) { F.*val* = E.*val* }
(7) F -> digit { F.*val* = digit.*lexval* }

SLR Automaton

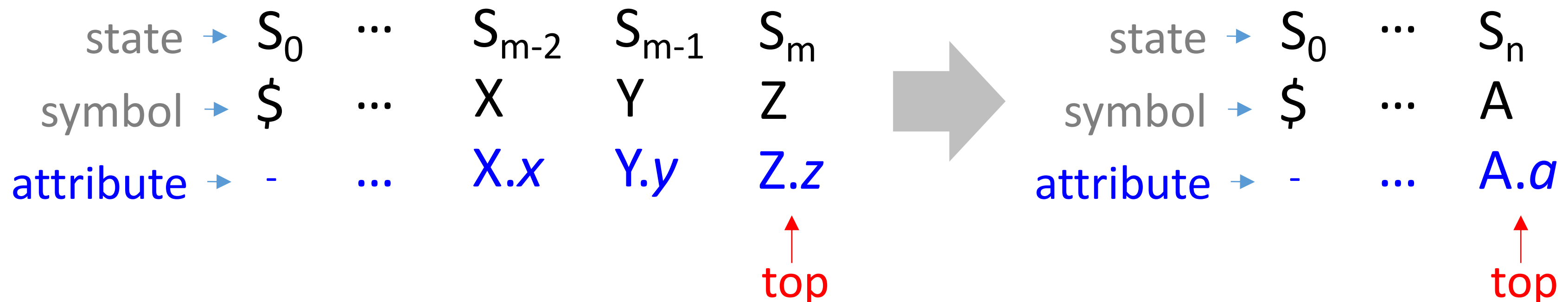


中山大學 SUN YAT-SEN UNIVERSITY

https://www.icourse163.org/learn/HIT-1002123007

Extend LR Parse Stack[扩展分析栈]

- Save synthesized attributes into the stack[栈中额外存放综合属性值]
  - Place the attributes along with the grammar symbols (or LR states that associated with these symbols) in records on stack
  - If there are multiple attributes
    - Make the records large enough or by putting pointers to records on the stack[栈记录足够大，或栈记录中存放指针]
- Example: A -> XYZ {action}
  - x, y, z are attributes of X, Y, Z respectively
  - After the action, A and its attributes are at the top (i.e., m-2)



中山大學 SUN YAT-SEN UNIVERSITY

https://www.icourse163.org/learn/HIT-1002123007

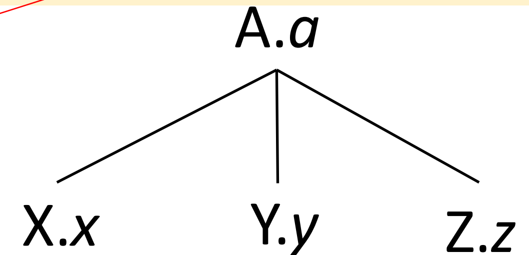
NSCC GZ

Stack Manipulation[栈操作]

- Rewrite the actions to manipulate the parser stack[语义动作]
  - The manipulation can be done automatically by the parser

stack[top-2].*symbol* = A
stack[top-2].*val* = f(stack[top-2].*val*, stack[top-1].*val*, stack[top].*val*)
top = top -2

A -> XYZ { A.a = f(X.*x*, Y.*y*, Z.*z*) }



state →	S_0	…	S_{m-2}	S_{m-1}	S_m
symbol →	$	…	X	Y	Z
attribute →	-	…	X.*x*	Y.*y*	Z.*z*
					↑ top

state →	S_0	…	S_n
symbol →	$	…	A
attribute →	-	…	A.*a*
			↑ top

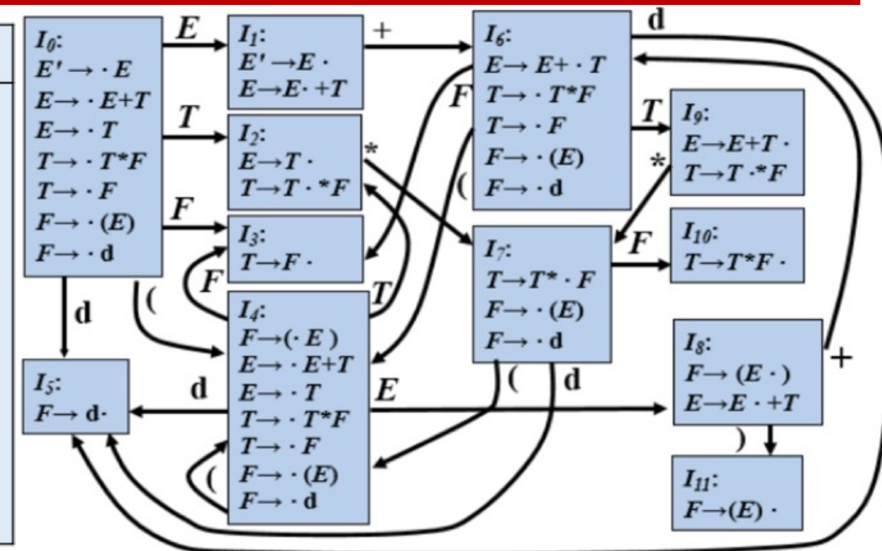
Example

- Rewrite the actions to manipulate the parser stack
  - The manipulation can be done automatically by the parser

Productions	Semantic Rules	Semantic Actions
(1) L -> E	print (E.*val*)	{ print(stack[top].val); }
(2) E -> E_1+T	E.*val* = E_1.*val*+T.*val*	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	E.*val* = T.*val*	
(4) T -> T_1*F	T.*val* = T_1.*val*xF.*val*	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	T.*val* = F.*val*	
(6) F -> (E)	F.*val* = E.*val*	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	F.*val* = digit.*lexval*	

Example

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	



Input: 3 * 5 + 4

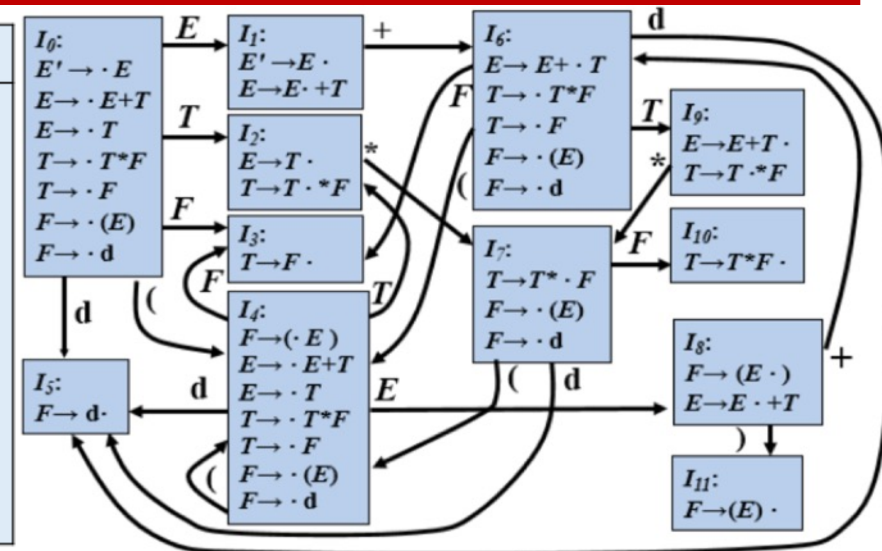
state → S_0

symbol → $

attribute → -

Example

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	



Input: 3 * 5 + 4

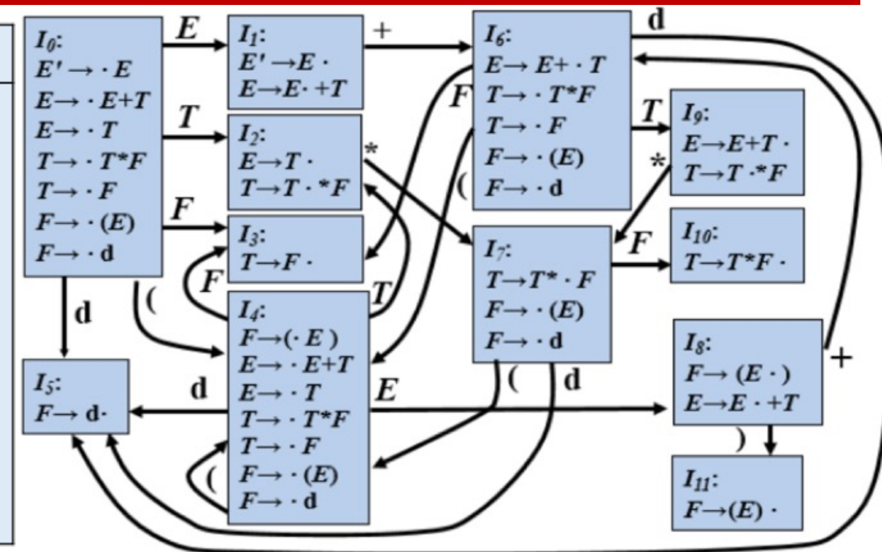
state → S_0

symbol → $

attribute → -

Example

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	

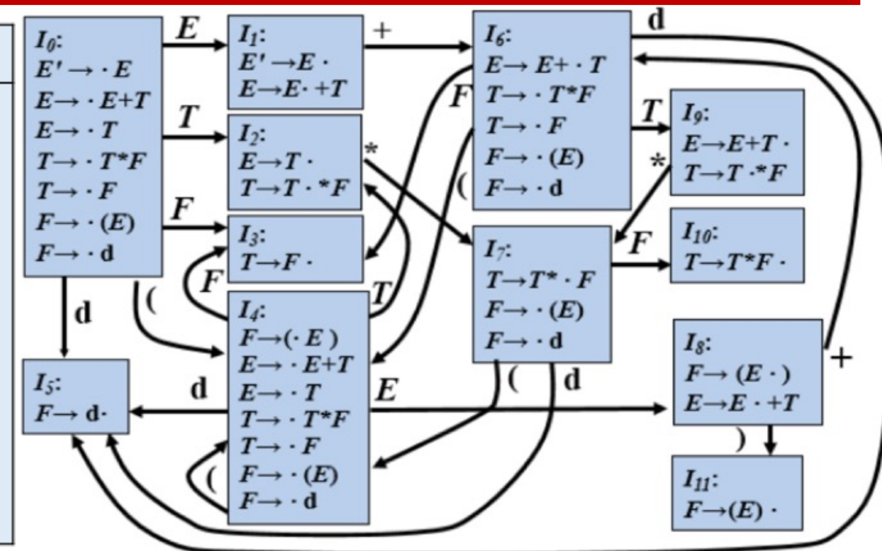


Input: 3 * 5 + 4

state →	S_0	S_5
symbol →	$	d
attribute →	-	3

Example

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	



Input: 3 * 5 + 4

state → S_0

symbol → $

attribute → - 3

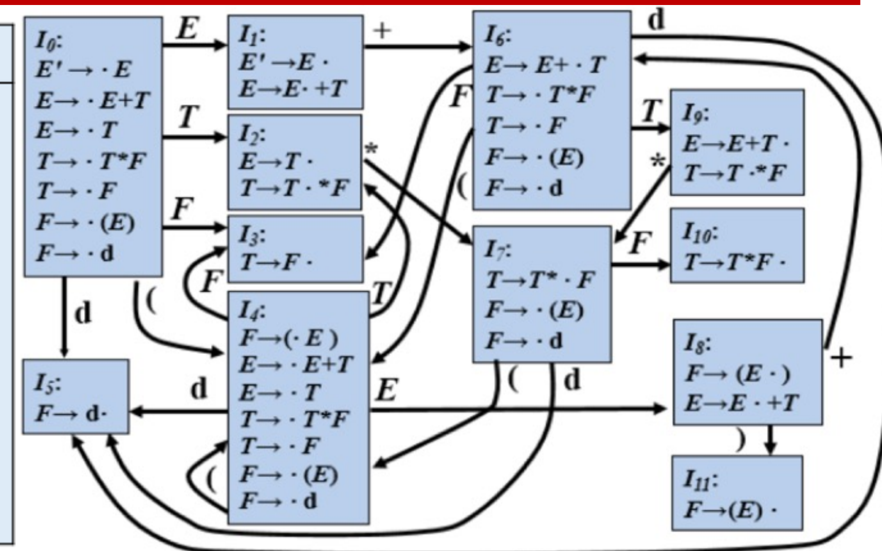
中山大學 SUN YAT-SEN UNIVERSITY

https://www.icourse163.org/learn/HIT-1002123007

NSCC GZ

Example

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	



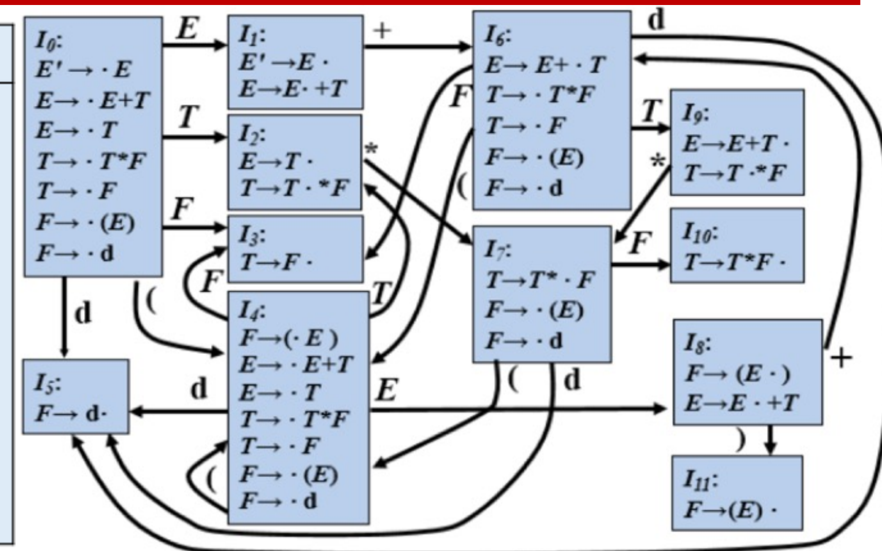
Input: 3 * 5 + 4

state →	S_0	S_3
symbol →	$	F
attribute →	-	3

中山大學 SUN YAT-SEN UNIVERSITY
https://www.icourse163.org/learn/HIT-1002123007
NSCC GZ

Example

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	



Input: 3 * 5 + 4

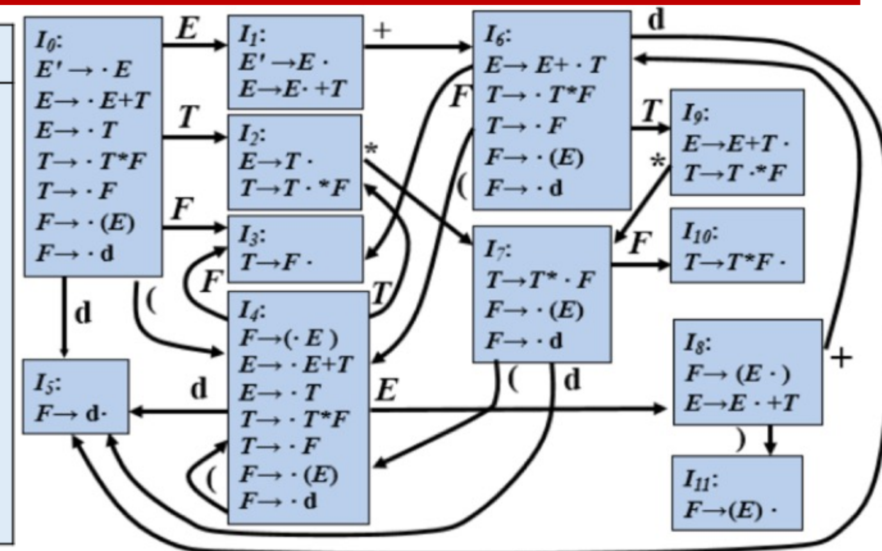
state → S_0

symbol → $

attribute → - 3

Example

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	



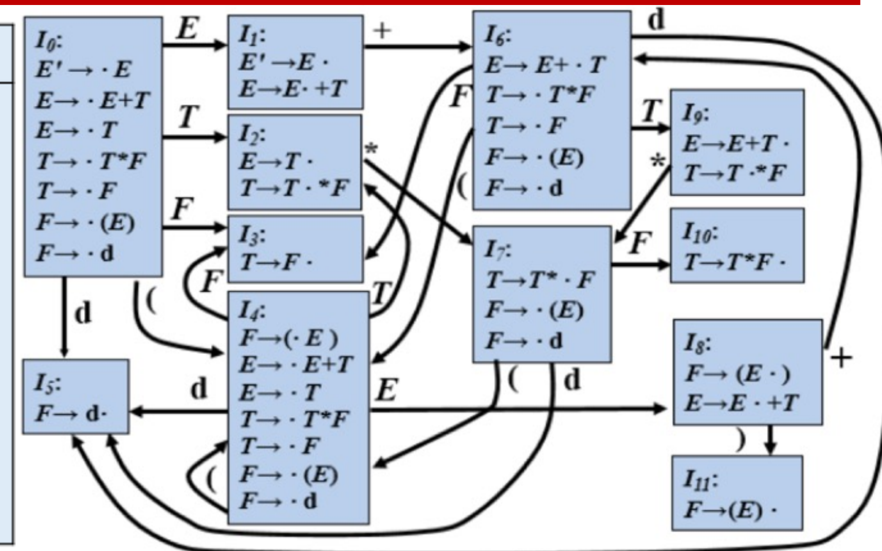
Input: 3 * 5 + 4

state →	S_0	S_2
symbol →	$	T
attribute →	-	3

SUN YAT-SEN UNIVERSITY 中山大學
https://www.icourse163.org/learn/HIT-1002123007
NSCC GZ

Example

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	



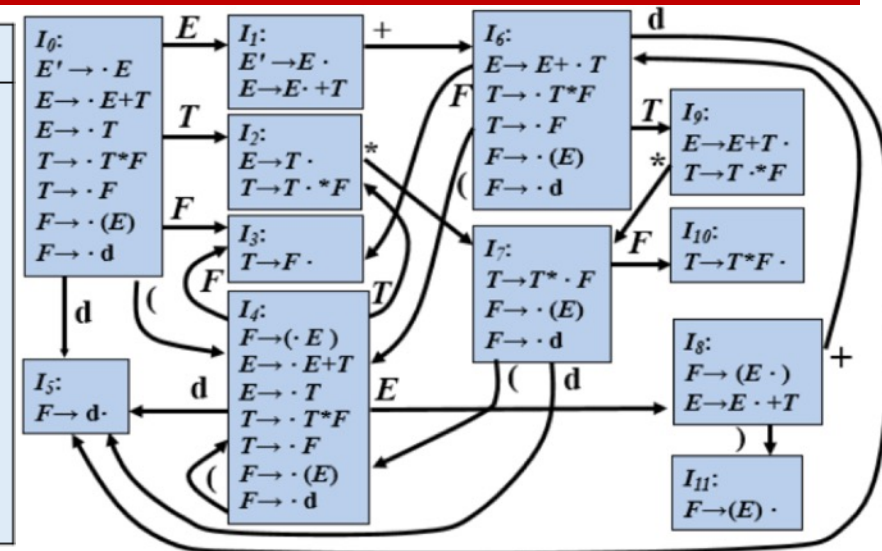
Input: 3 * 5 + 4

state →	S_0	S_2
symbol →	$	T
attribute →	-	3

https://www.icourse163.org/learn/HIT-1002123007
中山大學 SUN YAT-SEN UNIVERSITY
NSCC GZ

Example

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	



Input: 3 * 5 + 4

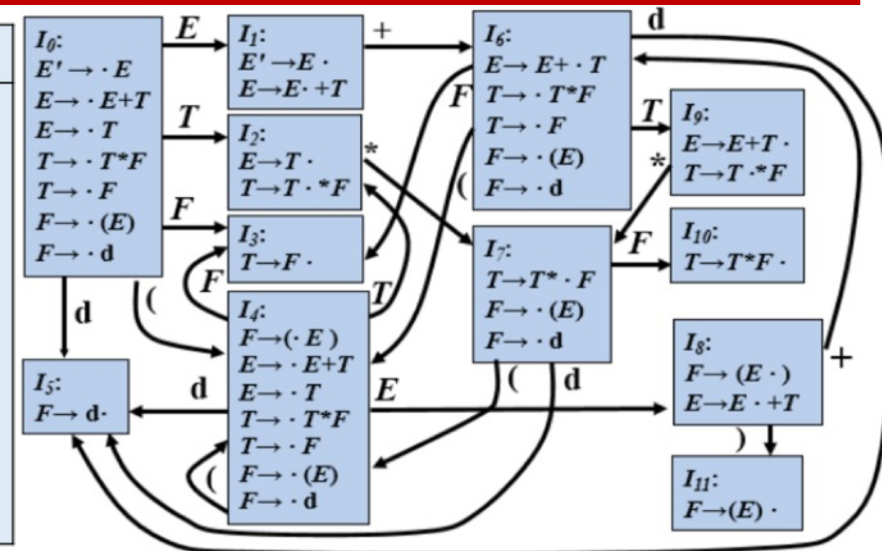
state → S_0 S_2

symbol → \$ T

attribute → - 3

Example

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	

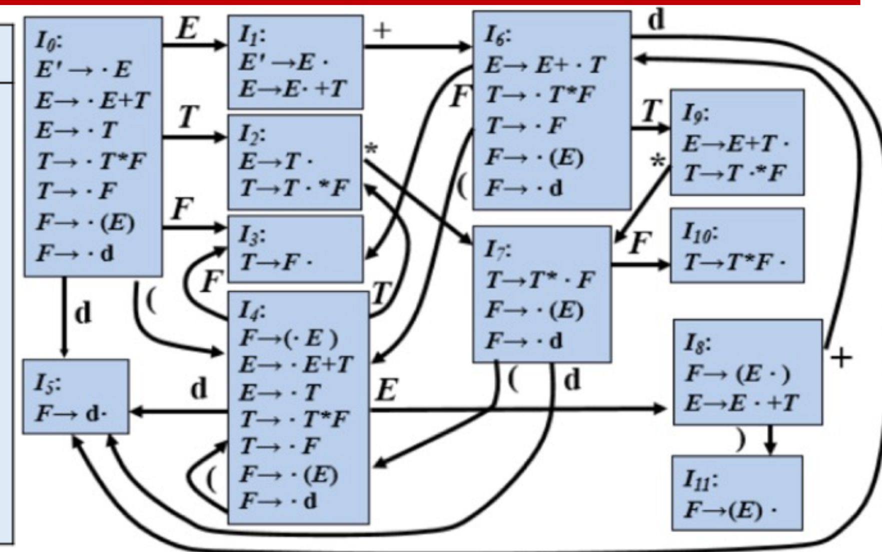


Input: 3 * 5 + 4

state →	S_0	S_2	S_7
symbol →	$	T	*
attribute →	-	3	-

Example

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	



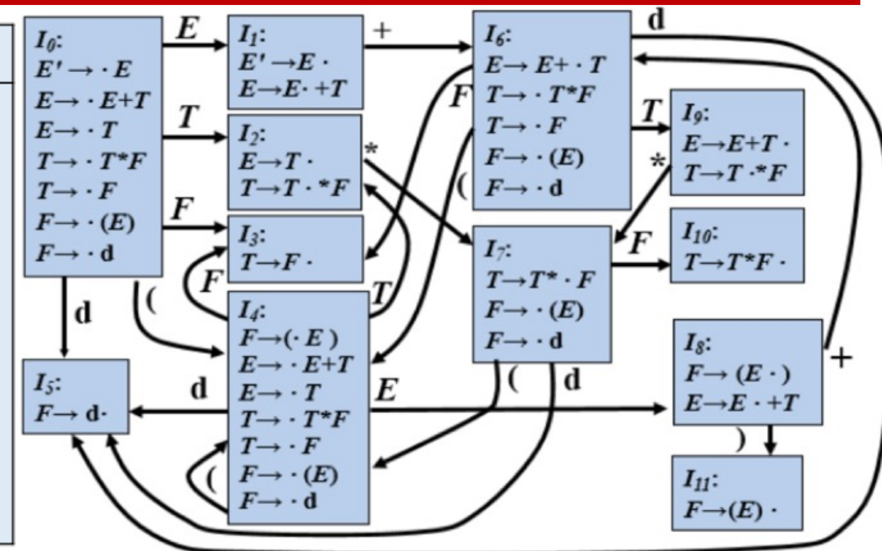
Input: 3 * 5 + 4

state →	S_0	S_2	S_7
symbol →	$	T	*
attribute →	-	3	-

https://www.icourse163.org/learn/HIT-1002123007
中山大學 SUN YAT-SEN UNIVERSITY
NSCC GZ

Example

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	

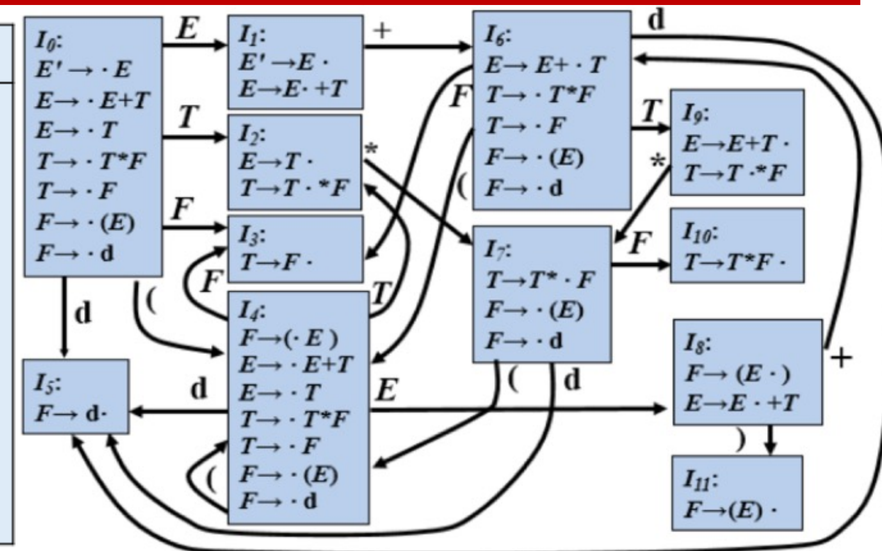


Input: 3 * 5 + 4

state →	S_0	S_2	S_7
symbol →	$	T	*
attribute →	-	3	-

Example

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	

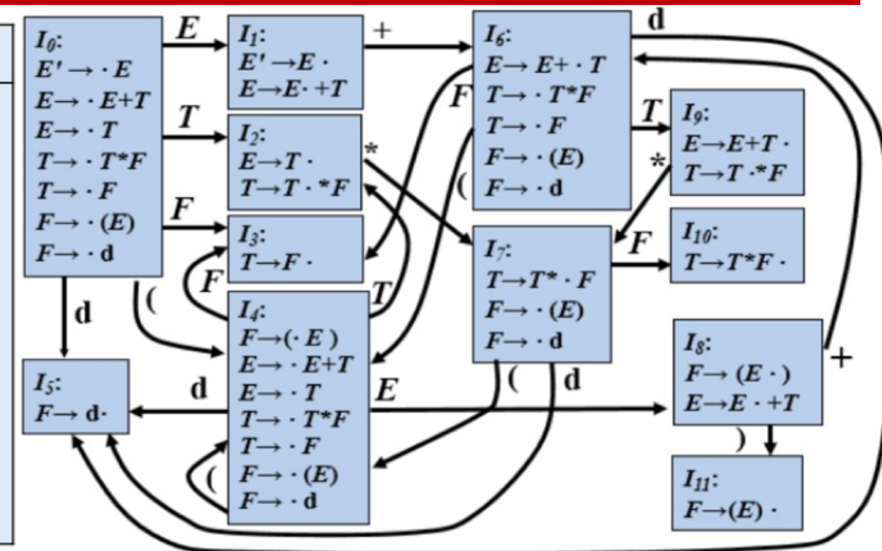


Input: 3 * 5 + 4

state →	S_0	S_2	S_7	S_5
symbol →	$	T	*	d
attribute →	-	3	-	5

Example

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	



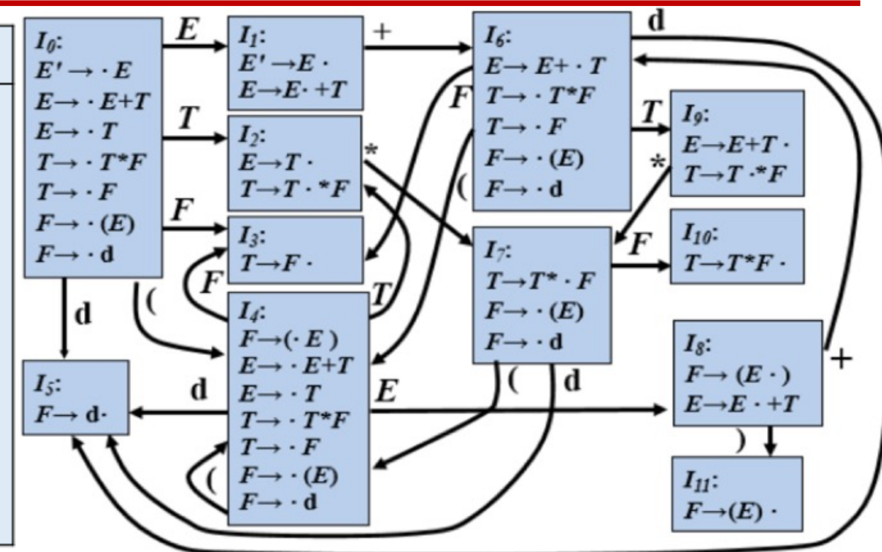
Input: 3 * 5 + 4

state →	S_0	S_2	S_7
symbol →	$	T	*
attribute →	-	3	-

中山大學 SUN YAT-SEN UNIVERSITY
https://www.icourse163.org/learn/HIT-1002123007
NSCCGZ

Example

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	



Input: 3 * 5 + 4

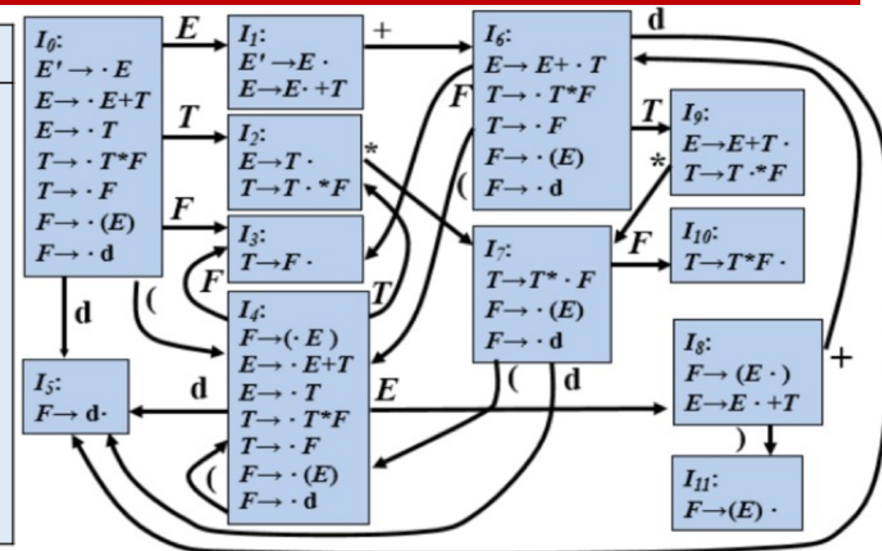
state →	S_0	S_2	S_7	S_{10}
symbol →	$	T	*	F
attribute →	-	3	-	5

中山大學 SUN YAT-SEN UNIVERSITY

NSCC GZ

Example (cont.)

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	



Input: 3 * 5 + 4

state	S_0	S_2	S_7	S_{10}
symbol	$	T	*	F
attribute	-	3	-	5

top

state	S_0
symbol	$
attribute	-

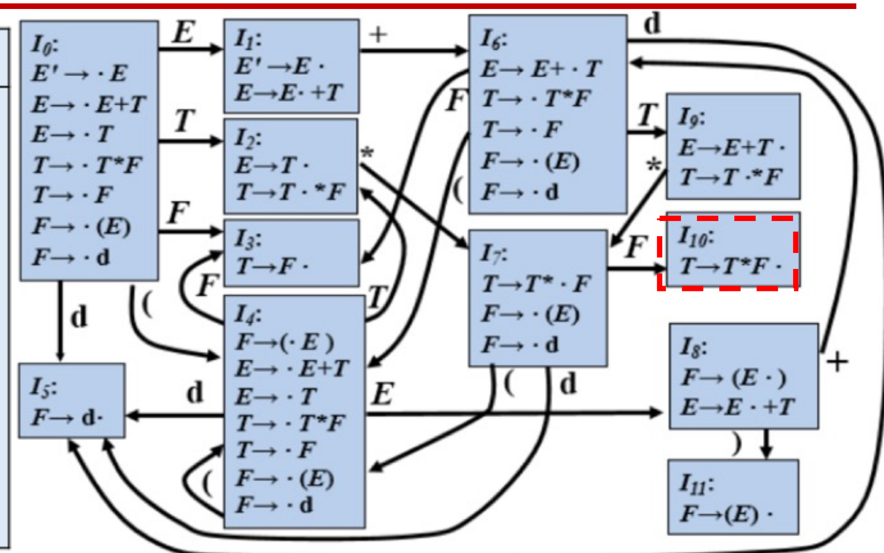
中山大學 SUN YAT-SEN UNIVERSITY

https://www.icourse163.org/learn/HIT-1002123007

NSCC GZ

Example (cont.)

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	



Input: 3 * 5 + 4

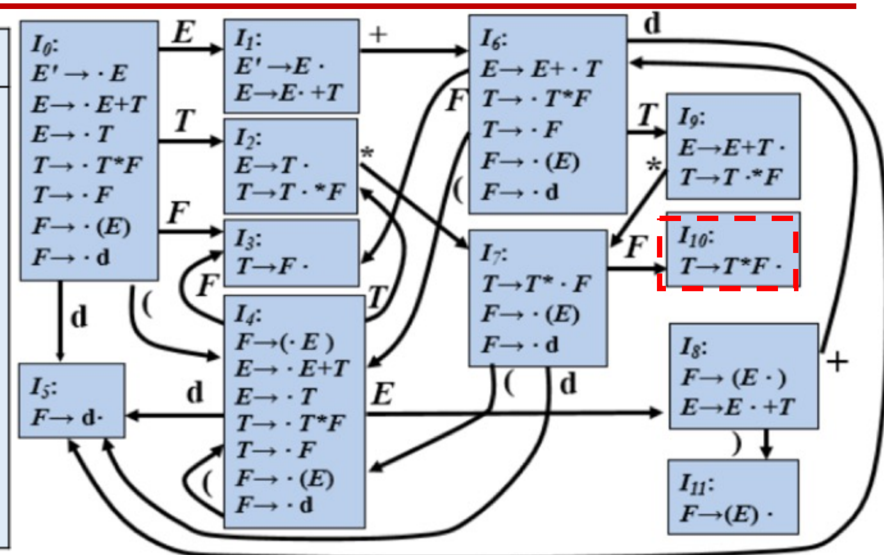
state →	S_0	S_2	S_7	S_{10}
symbol →	$	T	*	F
attribute →	-	3	-	5

top

state →	S_0
symbol →	$
attribute →	-

Example (cont.)

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	



Input: 3 * 5 + 4

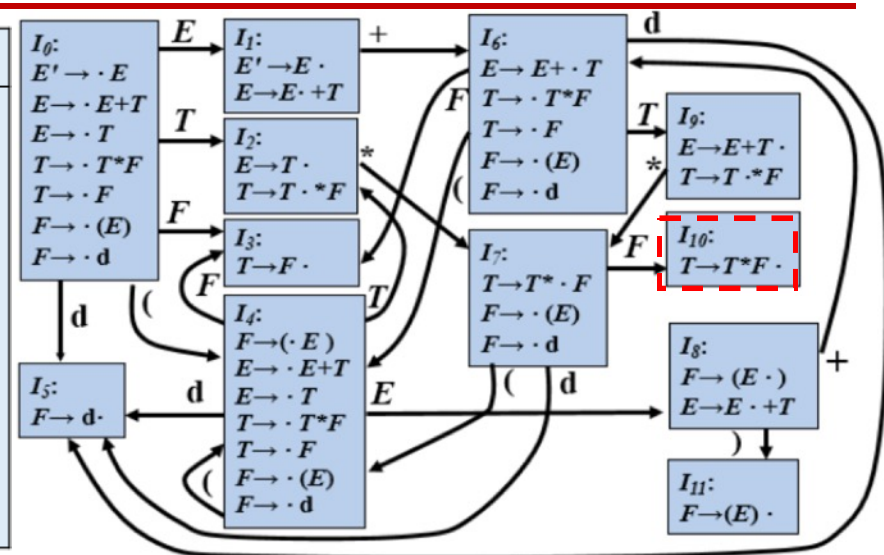
state →	S_0	S_2	S_7	S_{10}
symbol →	$	T	*	F
attribute →	-	3	-	5

top

state →	S_0	
symbol →	$	
attribute →	-	15

Example (cont.)

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	



Input: 3 * 5 + 4

state →	S_0	S_2	S_7	S_{10}
symbol →	$	T	*	F
attribute →	-	3	-	5

top

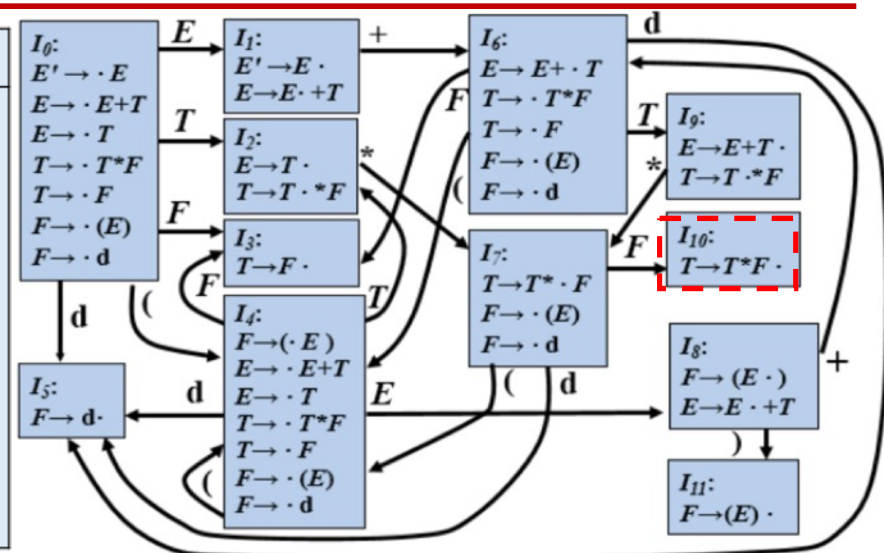
state →	S_0	
symbol →	$	T
attribute →	-	15

top

SUN YAT-SEN UNIVERSITY

Example (cont.)

Productions	Semantic Actions
(1) L -> E	{ print(stack[top].val); }
(2) E -> E_1+T	{ stack[top-2].*val* = stack[top-2].*val* + stack[top].*val*; top = top -2; }
(3) E -> T	
(4) T -> T_1*F	{ stack[top-2].*val* = stack[top-2].*val* x stack[top].*val*; top = top -2; }
(5) T -> F	
(6) F -> (E)	{ stack[top-2].*val* = stack[top-1].*val*; top = top -2; }
(7) F -> digit	



Input: 3 * 5 + 4

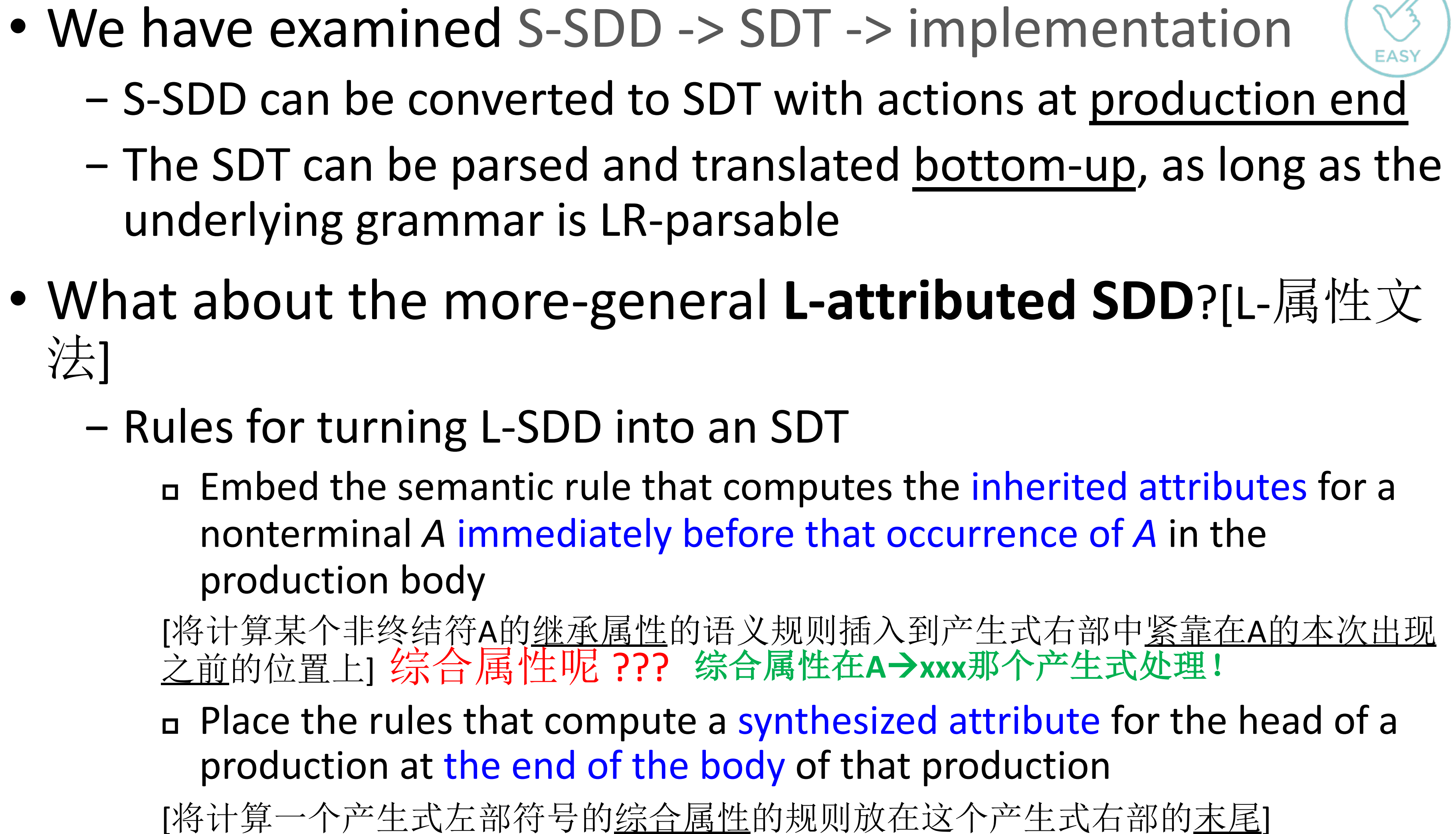
state →	S_0	S_2	S_7	S_{10}
symbol →	$	T	*	F
attribute →	-	3	-	5

top

state →	S_0	S_2
symbol →	$	T
attribute →	-	15

top

== Implement L-SDD ==



- We have examined S-SDD -> SDT -> implementation
  - S-SDD can be converted to SDT with actions at production end
  - The SDT can be parsed and translated bottom-up, as long as the underlying grammar is LR-parsable
- What about the more-general **L-attributed SDD**?[L-属性文法]
  - Rules for turning L-SDD into an SDT
    - Embed the semantic rule that computes the inherited attributes for a nonterminal *A* immediately before that occurrence of *A* in the production body

    [将计算某个非终结符A的继承属性的语义规则插入到产生式右部中紧靠在A的本次出现之前的位置上] 综合属性呢 ??? **综合属性在A→xxx那个产生式处理！**
    - Place the rules that compute a synthesized attribute for the head of a production at the end of the body of that production

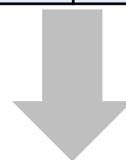
    [将计算一个产生式左部符号的综合属性的规则放在这个产生式右部的末尾]

Example

A -> B C

- C的继承属性：出现之前
- A的综合属性：末尾

Production Rules	Semantic Rules
(1) T -> F T’	T’.*inh* = F.*val* T.*val* = T’.*syn*
(2) T’ -> * F T_1’	T_1’.*inh* = T’.*inh* x F.*val* T’.*syn* = T_1’.*syn*
(3) T’ -> ε	T’.*syn* = T’.*inh*
(4) F -> digit	F.*val* = digit.*lexval*



SDT
(1) T -> F { T’.*inh* = F.*val* } T’ { T.*val* = T’.*syn* }
(2) T’ -> * F { T_1’.*inh* = T’.*inh* x F.*val* } T_1’ { T’.*syn* = T_1’.*syn* }
(3) T’ -> ε { T’.*syn* = T’.*inh* } (4) F -> digit { F.*val* = digit.*lexval* }

Implement the SDT of L-SDD

- If the underlying grammar is LL-parsable, then the SDT can be implemented during LL or LR parsing[若文法是LL可解析的，则可在LL或LR语法分析过程中实现]

- Semantic translation during **LL parsing**, using[LL方式]
  - A recursive-descent parser[LL递归下降]
    - Augment non-terminal functions to both parse and handle attributes
  - A predictive parser[LL非递归的预测分析]
    - Extend the parse stack to hold actions and certain data items needed for attribute evaluation
- A LR parser[LR方式]
  - Involve marker to rewrite grammars

中山大學 SUN YAT-SEN UNIVERSITY

NSCC GZ

L-SDD in LL Parsing[非递归预测]

- Extend the parse stack to hold **actions** and certain **data items** needed for attribute evaluation[扩展语法分析栈]
  - Action-record[动作记录]: represent the actions to be executed
  - Synthesize-record[综合记录]: hold synthesized attributes for non-terminals
  - Typically, the data items are copies of attributes[属性备份]
- Manage attributes on the stack[管理属性信息]
  - The inherited attributes of a nonterminal *A* are placed in the stack record that represents that terminal[符号位放继承属性]
    - Action-record to evaluate these attributes are immediately above *A*
  - The synthesized attributes of a nonterminal *A* are placed in a separate synthesize-record that is immediately below *A*[综合属性另存放]

action	Code
A	Inh Attr.
A.syn	Syn Attr.

中山大學 SUN YAT-SEN UNIVERSITY

NSCC GZ

L-SDD in LL Parsing (cont.)

- Table-driven LL-parser
  - Mimics a leftmost derivation --> stack expansion

- *A* -> *BC*, suppose nonterminal *C* has an inherited attr *C.i*
  - *C.i* may depend not only on the inherited attr. of A, but on all the attrs of *B*
    - Extra care should be taken on the attribute values
  - Since SDD is L-attributed, surely that the values of the inherited attrs of *A* are available when *A* rises to stack top ($X \rightarrow \alpha A \beta$)
    - Thus, available to be copied into C
  - *A*'s synthesized attrs remain on the stack, below *B* and *C* when expansion happens

action	Code
A	Inh Attr.
A.syn	Syn Attr.

SUN YAT-SEN UNIVERSITY 中山大學

NSCC GZ

L-SDD in LL Parsing (cont.)

- *A* -> *BC*: *C*.*i* may depend not only on the inherited attr. of *A*, but on all the attrs of *B*
  - Thus, need to process *B* completely before *C*.*i* can be evaluated
  - Save temporary copies of all attrs needed by evaluate *C*.*i* in the action-record that evaluates *C*.*i*; otherwise, when the parser replaces *A* on top of the stack by *BC*, the inherited attrs of *A* will be gone, along with its stack record
  - 变量展开时（i.e., 变量本身的记录出栈时），若其含有继承属性，则要将继承属性复制给后面的动作记录
  - 综合记录出栈时，要将综合属性值复制给后面的动作记录

action	Code
A	Inh Attr.
A.syn	Syn Attr.

SUN YAT-SEN UNIVERSITY

NSCC GZ

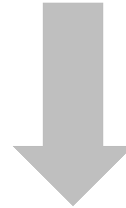
Example

(1) T -> F { T'.*inh* = F.*val* } T' { T.*val* = T'.*syn* }

(2) T' -> * F { T_1'.*inh* = T'.*inh* x F.*val* } T_1' { T'.*syn* = T_1'.*syn* }

(3) T' -> ε { T'.*syn* = T'.*inh* }
(4) F -> digit { F.*val* = digit.*lexval* }

Three kinds of symbols:
1) terminal
2) non-terminal
3) action



(1) T -> F { a_1 } T' { a_2 }	a_1: T'.*inh* = F.*val* a_2: T.*val* = T'.*syn*
(2) T' -> * F { a_3 } T_1' { a_4 }	a_3: T_1'.*inh* = T'.*inh* x F.*val* a_4: T'.*syn* = T_1'.*syn*
(3) T' -> ε { a_5 } (4) F -> digit { a_6 }	a_5: T'.*syn* = T'.*inh* a_6: F.*val* = digit.*lexval*

Example (cont.)

(1) T -> F { a_1 } T' { a_2 }	a_1: T'.*inh* = F.*val* a_2: T.*val* = T'.*syn*
(2) T' -> * F { a_3 } T_1' { a_4 }	a_3: T_1'.*inh* = T'.*inh* x F.*val* a_4: T'.*syn* = T_1'.*syn*
(3) T' -> ε { a_5 }	a_5: T'.*syn* = T'.*inh*
(4) F -> digit { a_6 }	a_6: F.*val* = digit.*lexval*

action	Code
A	Inh Attr.
A.syn	Syn Attr.

Input: 3 * 5

- 变量展开时（i.e., 变量本身的记录出栈时），若其含有继承属性，则要将继承属性复制给后面的动作记录
- 综合记录出栈时，要将综合属性值复制给后面的动作记录

完整步骤见👉：MOOC:语法制导翻译-3

https://www.icourse163.org/learn/HIT-1002123007

中山大學 SUN YAT-SEN UNIVERSITY

NSCC GZ

Example (cont.)

(1) T -> F { a_1 } T' { a_2 }	a_1: T'.*inh* = F.*val* a_2: T.*val* = T'.*syn*
(2) T' -> * F { a_3 } T_1' { a_4 }	a_3: T_1'.*inh* = T'.*inh* x F.*val* a_4: T'.*syn* = T_1'.*syn*
(3) T' -> ε { a_5 }	a_5: T'.*syn* = T'.*inh*
(4) F -> digit { a_6 }	a_6: F.*val* = digit.*lexval*

action	Code
A	Inh Attr.
A.syn	Syn Attr.

Input: 3 * 5

- 变量展开时（i.e., 变量本身的记录出栈时），若其含有继承属性，则要将继承属性复制给后面的动作记录
- 综合记录出栈时，要将综合属性值复制给后面的动作记录

T	Tsyn	$
	val	

完整步骤见👉： MOOC:语法制导翻译-3

Example (cont.)

(1) T -> F { a_1 } T' { a_2 }	a_1: T'.*inh* = F.*val* a_2: T.*val* = T'.*syn*
(2) T' -> * F { a_3 } T_1' { a_4 }	a_3: T_1'.*inh* = T'.*inh* x F.*val* a_4: T'.*syn* = T_1'.*syn*
(3) T' -> ε { a_5 }	a_5: T'.*syn* = T'.*inh*
(4) F -> digit { a_6 }	a_6: F.*val* = digit.*lexval*

action	Code
A	Inh Attr.
A.syn	Syn Attr.

Input: 3 * 5

- 变量展开时（i.e., 变量本身的记录出栈时），若其含有继承属性，则要将继承属性复制给后面的动作记录
- 综合记录出栈时，要将综合属性值复制给后面的动作记录

Tsyn	$
val	

完整步骤见👉： MOOC:语法制导翻译-3

Example (cont.)

(1) T -> F { a_1 } T' { a_2 }	a_1: T'.*inh* = F.*val* a_2: T.*val* = T'.*syn*
(2) T' -> * F { a_3 } T_1' { a_4 }	a_3: T_1'.*inh* = T'.*inh* x F.*val* a_4: T'.*syn* = T_1'.*syn*
(3) T' -> ε { a_5 }	a_5: T'.*syn* = T'.*inh*
(4) F -> digit { a_6 }	a_6: F.*val* = digit.*lexval*

action	Code
A	Inh Attr.
A.syn	Syn Attr.

Input: 3 * 5

- 变量展开时（i.e., 变量本身的记录出栈时），若其含有继承属性，则要将继承属性复制给后面的动作记录
- 综合记录出栈时，要将综合属性值复制给后面的动作记录

F	Fsyn	{ a_1 }	T'	T'syn	{ a_2 }	Tsyn	$
	val		inh	val		val	

完整步骤见👉： MOOC:语法制导翻译-3

Example (cont.)

(1) T -> F { a_1 } T' { a_2 }	a_1: T'.*inh* = F.*val*
	a_2: T.*val* = T'.*syn*
(2) T' -> * F { a_3 } T_1' { a_4 }	a_3: T_1'.*inh* = T'.*inh* x F.*val*
	a_4: T'.*syn* = T_1'.*syn*
(3) T' -> ε { a_5 }	a_5: T'.*syn* = T'.*inh*
(4) F -> digit { a_6 }	a_6: F.*val* = digit.*lexval*

action	Code
A	Inh Attr.
A.syn	Syn Attr.

Input: 3 * 5

- 变量展开时（i.e., 变量本身的记录出栈时），若其含有继承属性，则要将继承属性复制给后面的动作记录
- 综合记录出栈时，要将综合属性值复制给后面的动作记录

Fsyn	{ a_1 }	T'	T'syn	{ a_2 }	Tsyn	$
val		inh	val		val	

完整步骤见👉：MOOC:语法制导翻译-3

Example (cont.)

(1) T -> F { a_1 } T' { a_2 }	a_1: T'.*inh* = F.*val* a_2: T.*val* = T'.*syn*
(2) T' -> * F { a_3 } T_1' { a_4 }	a_3: T_1'.*inh* = T'.*inh* x F.*val* a_4: T'.*syn* = T_1'.*syn*
(3) T' -> ε { a_5 }	a_5: T'.*syn* = T'.*inh*
(4) F -> digit { a_6 }	a_6: F.*val* = digit.*lexval*

action	Code
A	Inh Attr.
A.syn	Syn Attr.

Input: 3 * 5

- <u>变量展开</u>时（i.e., <u>变量本身的记录出栈</u>时），若其含有<u>继承属性</u>，则要将继承属性<u>复制</u>给后面的动作记录
- <u>综合记录出栈</u>时，要将<u>综合属性值复制</u>给后面的动作记录

digit	{ a_6 }	Fsyn	{ a_1 }	T'	T'syn	{ a_2 }	Tsyn	$
lexv=3		val		inh	val		val	

完整步骤见👉： [MOOC:语法制导翻译-3]

中山大學 SUN YAT-SEN UNIVERSITY

https://www.icourse163.org/learn/HIT-1002123007

NSCC GZ

Example (cont.)

(1) T -> F { a_1 } T' { a_2 }	a_1: T'.*inh* = F.*val* a_2: T.*val* = T'.*syn*
(2) T' -> * F { a_3 } T_1' { a_4 }	a_3: T_1'.*inh* = T'.*inh* x F.*val* a_4: T'.*syn* = T_1'.*syn*
(3) T' -> ε { a_5 }	a_5: T'.*syn* = T'.*inh*
(4) F -> digit { a_6 }	a_6: F.*val* = digit.*lexval*

action	Code
A	Inh Attr.
A.syn	Syn Attr.

Input: 3 * 5

Stack top 'digit' matches the input '3'
- pop 'digit', but value copy is needed

- 变量展开时（i.e., 变量本身的记录出栈时），若其含有继承属性，则要将继承属性复制给后面的动作记录
- 综合记录出栈时，要将综合属性值复制给后面的动作记录

digit	{ a_6 }	Fsyn	{ a_1 }	T'	T'syn	{ a_2 }	Tsyn	$
lexv=3		val		inh	val		val	

完整步骤见👉： MOOC:语法制导翻译-3

https://www.icourse163.org/learn/HIT-1002123007

中山大學 SUN YAT-SEN UNIVERSITY
NSCC GZ

Example (cont.)

(1) T -> F { a_1 } T' { a_2 }	a_1: T'.*inh* = F.*val* a_2: T.*val* = T'.*syn*
(2) T' -> * F { a_3 } T_1' { a_4 }	a_3: T_1'.*inh* = T'.*inh* x F.*val* a_4: T'.*syn* = T_1'.*syn*
(3) T' -> ε { a_5 }	a_5: T'.*syn* = T'.*inh*
(4) F -> digit { a_6 }	a_6: F.*val* = digit.*lexval*

action	Code
A	Inh Attr.
A.syn	Syn Attr.

Input: 3 * 5

Stack top 'digit' matches the input '3'
- pop 'digit', but value copy is needed

- 变量展开时（i.e., 变量本身的记录出栈时），若其含有继承属性，则要将继承属性复制给后面的动作记录
- 综合记录出栈时，要将综合属性值复制给后面的动作记录

digit	{ a_6 }	Fsyn	{ a_1 }	T'	T'syn	{ a_2 }	Tsyn	$
lexv=3	d_lexv=3	val		inh	val		val	

完整步骤见👉： MOOC:语法制导翻译-3

https://www.icourse163.org/learn/HIT-1002123007

SUN YAT-SEN UNIVERSITY 中山大學

Example (cont.)

(1) T -> F { a_1 } T' { a_2 }	a_1: T'.*inh* = F.*val* a_2: T.*val* = T'.*syn*
(2) T' -> * F { a_3 } T_1' { a_4 }	a_3: T_1'.*inh* = T'.*inh* x F.*val* a_4: T'.*syn* = T_1'.*syn*
(3) T' -> ε { a_5 }	a_5: T'.*syn* = T'.*inh*
(4) F -> digit { a_6 }	a_6: F.*val* = digit.*lexval*

action	Code
A	Inh Attr.
A.syn	Syn Attr.

Input: 3 * 5

↑

Stack top 'digit' matches the input '3'

- pop 'digit', but value copy is needed

- 变量展开时（i.e., 变量本身的记录出栈时），若其含有继承属性，则要将继承属性复制给后面的动作记录
- 综合记录出栈时，要将综合属性值复制给后面的动作记录

{ a_6 }	Fsyn	{ a_1 }	T'	T'syn	{ a_2 }	Tsyn	$
d_lexv=3	val		inh	val		val	

完整步骤见👉： MOOC:语法制导翻译-3

https://www.icourse163.org/learn/HIT-1002123007

中山大學 SUN YAT-SEN UNIVERSITY
NSCC GZ

Example (cont.)

(1) T -> F { a_1 } T' { a_2 }	a_1: T'.*inh* = F.*val* a_2: T.*val* = T'.*syn*
(2) T' -> * F { a_3 } T_1' { a_4 }	a_3: T_1'.*inh* = T'.*inh* x F.*val* a_4: T'.*syn* = T_1'.*syn*
(3) T' -> ε { a_5 }	a_5: T'.*syn* = T'.*inh*
(4) F -> digit { a_6 }	a_6: F.*val* = digit.*lexval*

action	Code
A	Inh Attr.
A.syn	Syn Attr.

Input: 3 * 5

Stack top 'digit' matches the input '3'
- pop 'digit', but value copy is needed

- 变量展开时（i.e., 变量本身的记录出栈时），若其含有继承属性，则要将继承属性复制给后面的动作记录
- 综合记录出栈时，要将综合属性值复制给后面的动作记录

{ a_6 }	Fsyn	{ a_1 }	T'	T'syn	{ a_2 }	Tsyn	$
d_lexv=3	val		inh	val		val	

完整步骤见👉： MOOC:语法制导翻译-3

https://www.icourse163.org/learn/HIT-1002123007

Example (cont.)

(1) T -> F { a_1 } T' { a_2 }	a_1: T'.*inh* = F.*val* a_2: T.*val* = T'.*syn*
(2) T' -> * F { a_3 } T_1' { a_4 }	a_3: T_1'.*inh* = T'.*inh* x F.*val* a_4: T'.*syn* = T_1'.*syn*
(3) T' -> ε { a_5 }	a_5: T'.*syn* = T'.*inh*
(4) F -> digit { a_6 }	a_6: F.*val* = digit.*lexval*

action	Code
A	Inh Attr.
A.syn	Syn Attr.

Input: 3 * 5

↑

Stack top 'digit' matches the input '3'

- pop 'digit', but value copy is needed

- 变量展开时（i.e., 变量本身的记录出栈时），若其含有继承属性，则要将继承属性复制给后面的动作记录
- 综合记录出栈时，要将综合属性值复制给后面的动作记录

{ a_6 }	Fsyn	{ a_1 }	T'	T'syn	{ a_2 }	Tsyn	$
d_lexv=3	val		inh	val		val	

完整步骤见👉： MOOC:语法制导翻译-3

Example (cont.)

(1) T -> F { a_1 } T' { a_2 }	a_1: T'.*inh* = F.*val* a_2: T.*val* = T'.*syn*
(2) T' -> * F { a_3 } T_1' { a_4 }	a_3: T_1'.*inh* = T'.*inh* x F.*val* a_4: T'.*syn* = T_1'.*syn*
(3) T' -> ε { a_5 }	a_5: T'.*syn* = T'.*inh*
(4) F -> digit { a_6 }	a_6: F.*val* = digit.*lexval*

action	Code
A	Inh Attr.
A.syn	Syn Attr.

Input: 3 * 5

↑

Stack top 'digit' matches the input '3'
- pop 'digit', but value copy is needed

a_6: stack[top-1].*val* = stack[top].*d_lexval*

{ a_6 }	Fsyn	{ a_1 }	T'	T'syn	{ a_2 }	Tsyn	$
d_lexv=3	val		inh	val		val	

- 变量展开时（i.e., 变量本身的记录出栈时），若其含有继承属性，则要将继承属性复制给后面的动作记录
- 综合记录出栈时，要将综合属性值复制给后面的动作记录

完整步骤见👉： MOOC:语法制导翻译-3

Example (cont.)

(1) T -> F { a_1 } T' { a_2 }	a_1: T'.*inh* = F.*val* a_2: T.*val* = T'.*syn*
(2) T' -> * F { a_3 } T_1' { a_4 }	a_3: T_1'.*inh* = T'.*inh* x F.*val* a_4: T'.*syn* = T_1'.*syn*
(3) T' -> ε { a_5 }	a_5: T'.*syn* = T'.*inh*
(4) F -> digit { a_6 }	a_6: F.*val* = digit.*lexval*

action	Code
A	Inh Attr.
A.syn	Syn Attr.

Input: 3 * 5

Stack top 'digit' matches the input '3'
- pop 'digit', but value copy is needed

a_6: stack[top-1].*val* = stack[top].*d_lexval*

{ a_6 }	Fsyn	{ a_1 }	T'	T'syn	{ a_2 }	Tsyn	$
d_lexv=3	val =3		inh	val		val	

- 变量展开时（i.e., 变量本身的记录出栈时），若其含有继承属性，则要将继承属性复制给后面的动作记录
- 综合记录出栈时，要将综合属性值复制给后面的动作记录

完整步骤见👉： MOOC:语法制导翻译-3

SUN YAT-SEN UNIVERSITY 中山大學

https://www.icourse163.org/learn/HIT-1002123007

Example (cont.)

(1) T -> F { a_1 } T' { a_2 }	a_1: T'.*inh* = F.*val* a_2: T.*val* = T'.*syn*
(2) T' -> * F { a_3 } T_1' { a_4 }	a_3: T_1'.*inh* = T'.*inh* x F.*val* a_4: T'.*syn* = T_1'.*syn*
(3) T' -> ε { a_5 }	a_5: T'.*syn* = T'.*inh*
(4) F -> digit { a_6 }	a_6: F.*val* = digit.*lexval*

action	Code
A	Inh Attr.
A.syn	Syn Attr.

Input: 3 * 5

↑

Stack top 'digit' matches the input '3'
- pop 'digit', but value copy is needed

a_6: stack[top-1].*val* = stack[top].*d_lexval*

Fsyn	{ a_1 }	T'	T'syn	{ a_2 }	Tsyn	$
val =3		inh	val		val	

- 变量展开时（i.e., 变量本身的记录出栈时），若其含有继承属性，则要将继承属性复制给后面的动作记录
- 综合记录出栈时，要将综合属性值复制给后面的动作记录

完整步骤见👉： MOOC:语法制导翻译-3

中山大學 SUN YAT-SEN UNIVERSITY
NSCC GZ

Example (cont.)

(1) T -> F { a_1 } T' { a_2 }	a_1: T'.*inh* = F.*val* a_2: T.*val* = T'.*syn*
(2) T' -> * F { a_3 } T_1' { a_4 }	a_3: T_1'.*inh* = T'.*inh* x F.*val* a_4: T'.*syn* = T_1'.*syn*
(3) T' -> ε { a_5 }	a_5: T'.*syn* = T'.*inh*
(4) F -> digit { a_6 }	a_6: F.*val* = digit.*lexval*

action	Code
A	Inh Attr.
A.syn	Syn Attr.

Input: 3 * 5

Stack top 'digit' matches the input '3'
- pop 'digit', but value copy is needed

a_6: stack[top-1].*val* = stack[top].*d_lexval*

- 变量展开时（i.e., 变量本身的记录出栈时），若其含有继承属性，则要将继承属性复制给后面的动作记录
- 综合记录出栈时，要将综合属性值复制给后面的动作记录

Fsyn	{ a_1 }	T'	T'syn	{ a_2 }	Tsyn	$
val =3	val=3	inh	val		val	

完整步骤见👉：MOOC:语法制导翻译-3

https://www.icourse163.org/learn/HIT-1002123007

SUN YAT-SEN UNIVERSITY 中山大學

Example (cont.)

(1) T -> F { a_1 } T' { a_2 }	a_1: T'.*inh* = F.*val* a_2: T.*val* = T'.*syn*
(2) T' -> * F { a_3 } T_1' { a_4 }	a_3: T_1'.*inh* = T'.*inh* x F.*val* a_4: T'.*syn* = T_1'.*syn*
(3) T' -> ε { a_5 }	a_5: T'.*syn* = T'.*inh*
(4) F -> digit { a_6 }	a_6: F.*val* = digit.*lexval*

action	Code
A	Inh Attr.
A.syn	Syn Attr.

Input: 3 * 5

Stack top 'digit' matches the input '3'

- pop 'digit', but value copy is needed

a_6: stack[top-1].*val* = stack[top].*d_lexval*

{ a_1 }	T'	T'syn	{ a_2 }	Tsyn	$
val=3	inh	val		val	

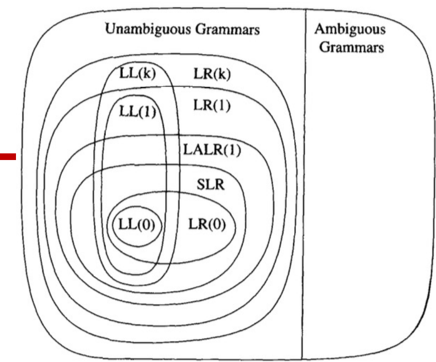
- 变量展开时（i.e., 变量本身的记录出栈时），若其含有继承属性，则要将继承属性复制给后面的动作记录
- 综合记录出栈时，要将综合属性值复制给后面的动作记录

完整步骤见👉： MOOC:语法制导翻译-3

https://www.icourse163.org/learn/HIT-1002123007

中山大學 SUN YAT-SEN UNIVERSITY
NSCC GZ

L-SDD in LR Parsing[LR解析]



- What we already learnt
  - LR > LL, w.r.t parsing power
    - We can do bottom-up every translation that we can do top-down[所有的LL都可以LR]
  - S-attributed SDD can be implemented in bottom-up way
    - All semantic actions are at the end of productions, i.e., triggered in reduce
- For L-attributed SDD on an LL grammar, can it be implemented during bottom-up parsing?
  - Problem: semantic actions can be in anywhere of the production body

(1) T -> F { T'.*inh* = F.*val* } T' { T.*val* = T'.*syn* }
(2) T' -> * F { T_1'.*inh* = T'.*inh* x F.*val* } T_1' { T'.*syn* = T_1'.*syn* }
(3) T' -> ε { T'.*syn* = T'.*inh* }
(4) F -> digit { F.*val* = digit.*lexval* }

The Problem

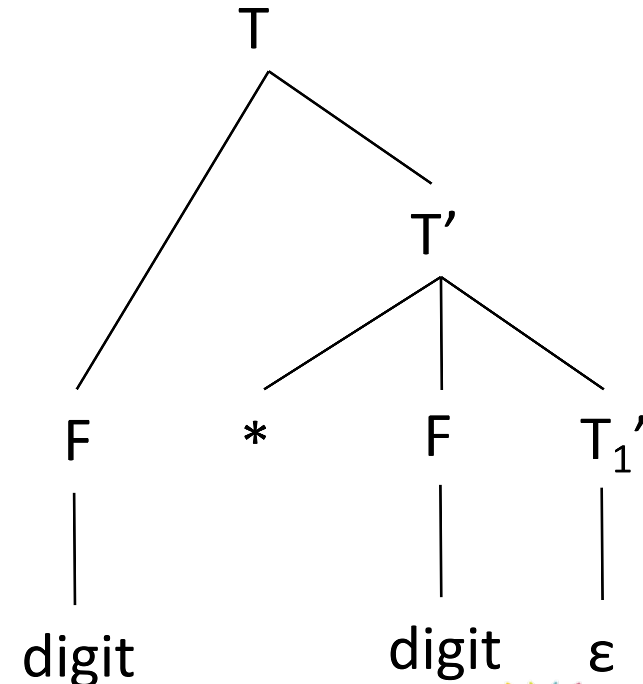
- It is not natural to evaluate inherited attributes
  - Example: how to get T'.*inh*

(1) T -> F { T'.*inh* = F.*val* } T' { T.*val* = T'.*syn* }
(2) T' -> * F { T_1'.*inh* = T'.*inh* x F.*val* } T_1' { T'.*syn* = T_1'.*syn* }
(3) T' -> ε { T'.*syn* = T'.*inh* }
(4) F -> digit { F.*val* = digit.*lexval* }

The Problem

- It is not natural to evaluate inherited attributes
  - Example: how to get T'.*inh*

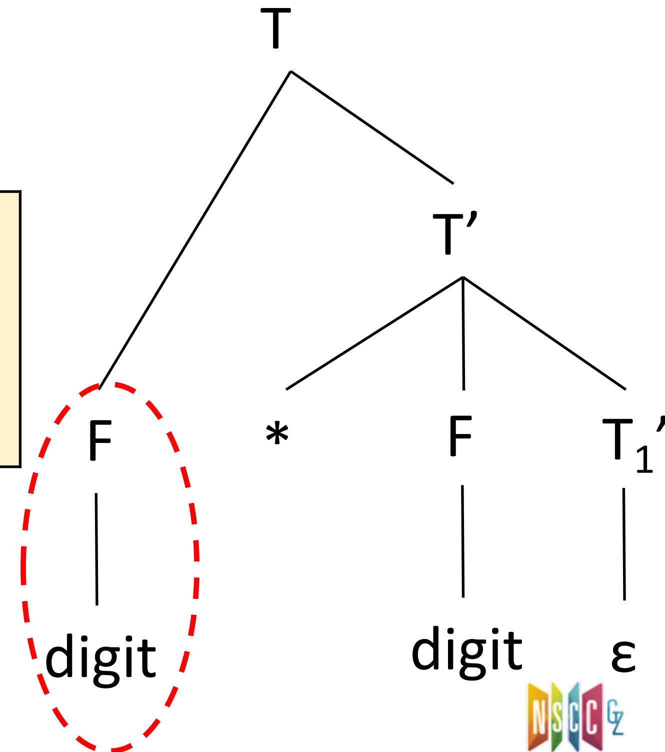
(1) T -> F { T'.*inh* = F.*val* } T' { T.*val* = T'.*syn* }
(2) T' -> * F { T_1'.*inh* = T'.*inh* x F.*val* } T_1' { T'.*syn* = T_1'.*syn* }
(3) T' -> ε { T'.*syn* = T'.*inh* }
(4) F -> digit { F.*val* = digit.*lexval* }



The Problem

- It is not natural to evaluate inherited attributes
  - Example: how to get T'.*inh*

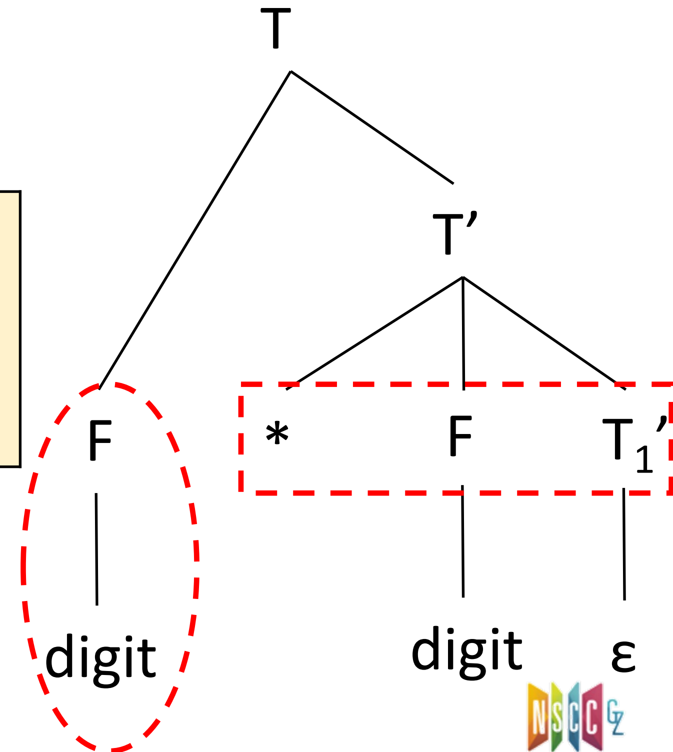
(1) T -> F { T'.*inh* = F.*val* } T' { T.*val* = T'.*syn* }
(2) T' -> * F { T_1'.*inh* = T'.*inh* x F.*val* } T_1' { T'.*syn* = T_1'.*syn* }
(3) T' -> ε { T'.*syn* = T'.*inh* }
(4) F -> digit { F.*val* = digit.*lexval* }



The Problem

- It is not natural to evaluate inherited attributes
  - Example: how to get T'.*inh*

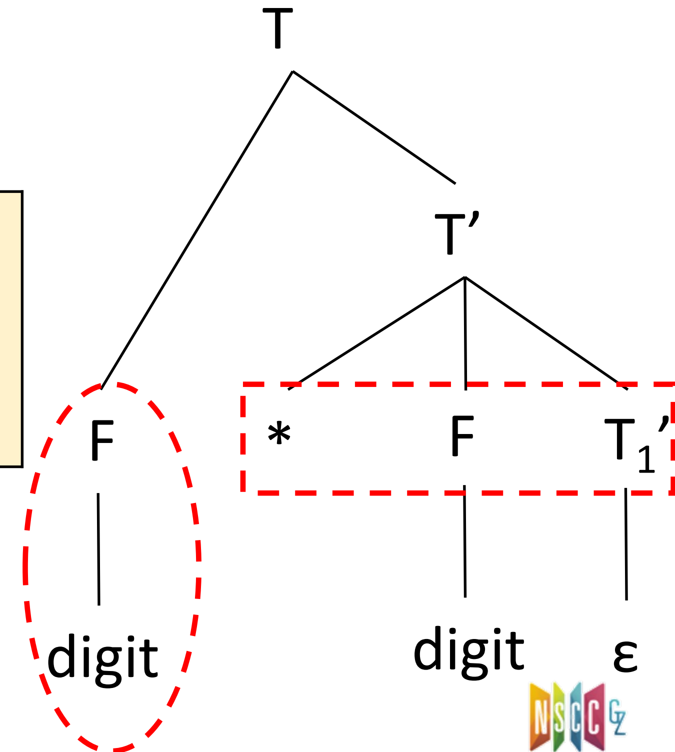
(1) T -> F { T'.*inh* = F.*val* } T' { T.*val* = T'.*syn* }
(2) T' -> * F { T_1'.*inh* = T'.*inh* x F.*val* } T_1' { T'.*syn* = T_1'.*syn* }
(3) T' -> ε { T'.*syn* = T'.*inh* }
(4) F -> digit { F.*val* = digit.*lexval* }



The Problem

- It is not natural to evaluate inherited attributes
  - Example: how to get T'.*inh*

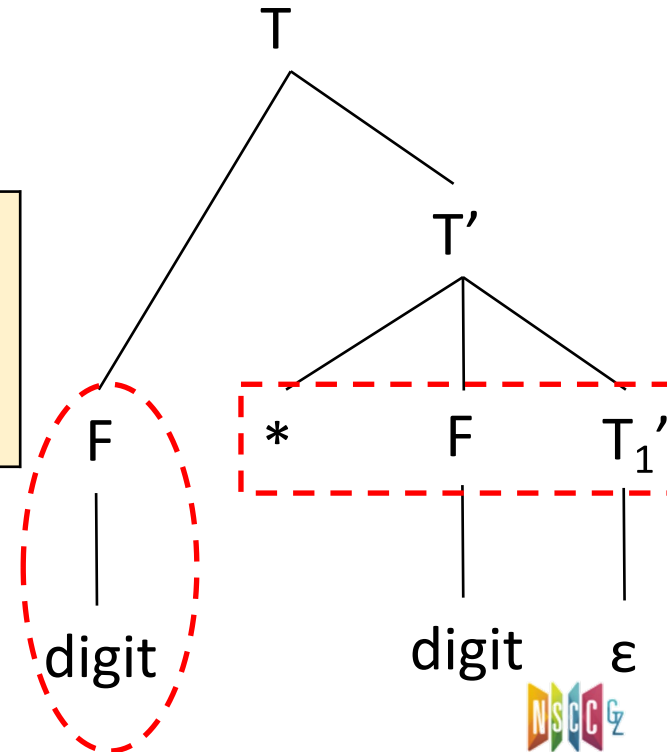
(1) T -> F { T'.*inh* = F.*val* } T' { T.*val* = T'.*syn* }
(2) T' -> * F { T_1'.*inh* = T'.*inh* x F.*val* } T_1' { T'.*syn* = T_1'.*syn* }
(3) T' -> ε { T'.*syn* = T'.*inh* }
(4) F -> digit { F.*val* = digit.*lexval* }



The Problem

- It is not natural to evaluate inherited attributes
  - Example: how to get T’.*inh*
- Claim: inherited attributes are on the stack
  - Left attributes guarantee they’ve already been computed
  - But computed by previous productions – deep in the stack

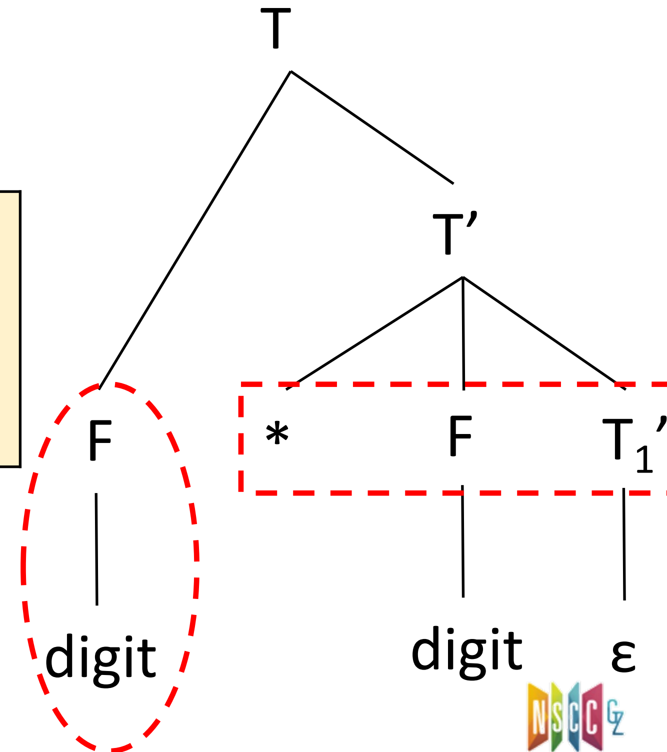
(1) T -> F { T’.*inh* = F.*val* } T’ { T.*val* = T’.*syn* }
(2) T’ -> * F { T_1’.*inh* = T’.*inh* x F.*val* } T_1’ { T’.*syn* = T_1’.*syn* }
(3) T’ -> ε { T’.*syn* = T’.*inh* }
(4) F -> digit { F.*val* = digit.*lexval* }



The Problem

- It is not natural to evaluate inherited attributes
  - Example: how to get T'.*inh*
- Claim: inherited attributes are on the stack
  - Left attributes guarantee they've already been computed
  - But computed by previous productions – deep in the stack
- Solution
  - **Hack the stack to dig out those values**

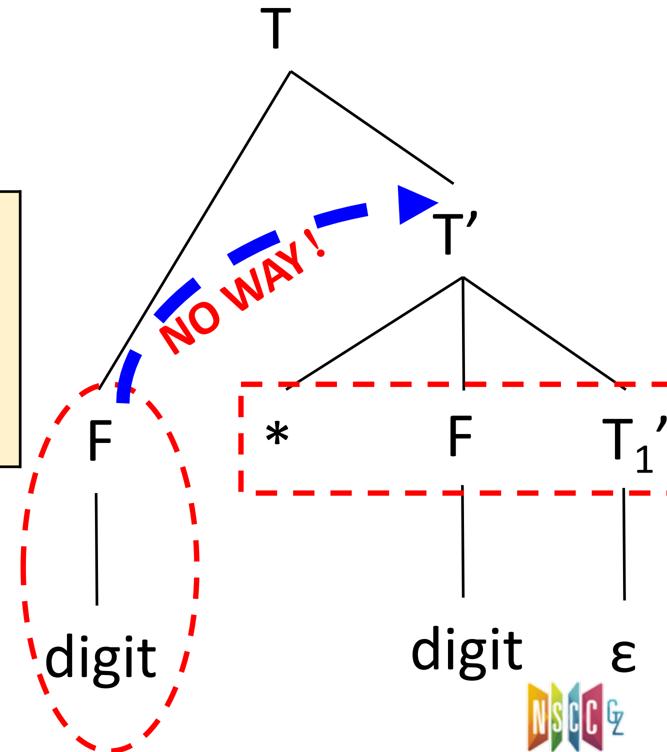
(1) T -> F { T'.*inh* = F.*val* } T' { T.*val* = T'.*syn* }
(2) T' -> * F { T_1'.*inh* = T'.*inh* x F.*val* } T_1' { T'.*syn* = T_1'.*syn* }
(3) T' -> ε { T'.*syn* = T'.*inh* }
(4) F -> digit { F.*val* = digit.*lexval* }



The Problem

- It is not natural to evaluate inherited attributes
  - Example: how to get T'.*inh*
- Claim: inherited attributes are on the stack
  - Left attributes guarantee they've already been computed
  - But computed by previous productions – deep in the stack
- Solution
  - **Hack the stack to dig out those values**

(1) T -> F { T'.*inh* = F.*val* } T' { T.*val* = T'.*syn* }
(2) T' -> * F { T_1'.*inh* = T'.*inh* x F.*val* } T_1' { T'.*syn* = T_1'.*syn* }
(3) T' -> ε { T'.*syn* = T'.*inh* }
(4) F -> digit { F.*val* = digit.*lexval* }



Marker[标记符号]

- Given the following SDD, where $|\alpha| \neq |\beta|$

  A -> X α { Y.*in* = X.*s* } Y | X β { Y.*in* = X.*s* } Y

  Y -> γ { Y.*s* = f(Y.*in*) }

- Problem: cannot generate stack location for Y.*in*
  - Because X.*s* is at different relative stack locations from Y
- Solution: insert markers M_1, M_2 right before Y

  A -> X α M_1 Y | X β M_2 Y

  Y -> γ { Y.*s* = f(stack[top - |γ|].*s*) }   // Y.*s* = M_1.*s* or Y.*s* = M_2.*s*

  M_1 -> ε { M_1.*s* = stack[top - |α|].*s* }   // M_1.*s* = X.*s*

  M_2 -> ε { M_2.*s* = stack[top - |β|].*s* }   // M_2.*s* = X.*s*

- **Marker**: a non-terminal marking a location equidistant from the symbol that has an inherited attribute
  - Always produces ε since its only a placeholder for an action

Modify Grammar with Marker[文法修改]

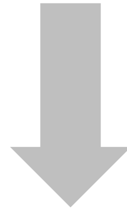
- Given an L-SDD on an LL grammar, we can adapt the grammar to compute the same SDD during a LR parsing
  - Introduce into the grammar a **marker nonterminal**[标记非终结符] in place of each embedded action
    - Each such place gets a distinct marker, and there is one production for any marker *M*, *M* -> ε[空产生式]
  - Modify the action *a* if marker nonterminal *M* replaces it in some production *A* -> α { a } β, and associate with *M* -> ε an action a' that
    - Copies, as inherited attrs of *M*, any attrs of *A* or symbols of α that action a needs (e.g., *M*.*i* = *A*.*i*)[左侧]
    - Computes attrs in the same way as *a*, but makes those attrs be synthesized attrs of *M* (e,.g., *M*.*s* = f(*M*.*i*))

A -> { B.*i* = f(A.*i*); } B C

A -> M B C
M -> ε { M.*i* = A.*i*; M.*s* = f(M.*i*); }

Example

(1) $T \rightarrow F$ { $T'.inh = F.val$ } T' { $T.val = T'.syn$ }
(2) $T' \rightarrow * F$ { $T_1'.inh = T'.inh \times F.val$ } T_1' { $T'.syn = T_1'.syn$ }
(3) $T' \rightarrow \varepsilon$ { $T'.syn = T'.inh$ }
(4) $F \rightarrow digit$ { $F.val = digit.lexval$ }



(1) $T \rightarrow F$ **M** T' { $T.val = T'.syn$ }
    $M \rightarrow \varepsilon$ { $M.i = F.val; M.s = M.i$ }
(2) $T' \rightarrow * F$ **N** T_1' { $T'.syn = T_1'.syn$ }
    $N \rightarrow \varepsilon$ { $N.i_1 = T'.inh; N.i_2 = F.val; N.s = N.i_1 \times N.i_2$ }
(3) $T' \rightarrow \varepsilon$ { $T'.syn = T'.inh$ }
(4) $F \rightarrow digit$ { $F.val = digit.lexval$ }

Stack Manipulation[栈操作]

(1) T -> F { *T'.inh = F.val* } T' { *T.val = T'.syn* }
(2) T' -> * F { *T_1'.inh = T'.inh x F.val* } T_1' { *T'.syn = T_1'.syn* }
(3) T' -> ε { *T'.syn = T'.inh* }
(4) F -> digit { *F.val* = digit.*lexval* }

(1) T -> F **M** T' { *T.val = T'.syn* }
M -> ε { M.i = F.val; M.s = M.i }
(2) T' -> * F **N** T_1' { *T'.syn = T_1'.syn* }
N -> ε { N.i1 = T'.inh; N.i2 = F.val; N.s = N.i1 x N.i2 }
(3) T' -> ε { *T'.syn = T'.inh* }
(4) F -> digit { *F.val* = digit.*lexval* }

(1) T -> F **M** T' { stack[top-2].*val* = stack[top].*syn*; top = top - 2; }
M -> ε { stack[top+1].*T'inh* = stack[top].*val*; top = top + 1; }
(2) T' -> * F **N** T_1' { stack[top-3].*syn* = stack[top].*syn*; top = top -3; }
N -> ε { stack[top+1].*T'inh* = stack[top-2].*T'inh* x stack[top].*val*; top = top + 1; }
(3) T' -> ε { stack[top+1].*syn* = stack[top].T'*inh*; top = top + 1; }
(4) F -> digit { stack[top].*val* = stack[top].*lexval*; }