



Compilation Principle 编译原理

第14讲: 语义分析(2)

张献伟

xianweiz.github.io

DCS290, 4/12/2022





Review Questions

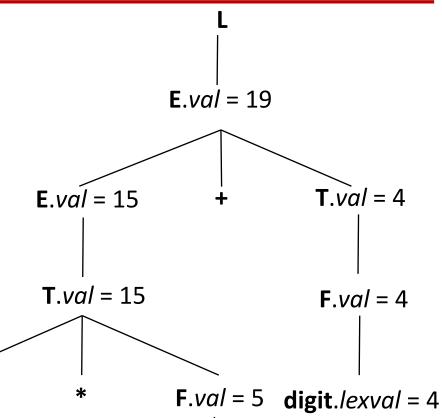
- Why context analysis is not performed in parsing stage?
 Parsing relies on CFG, which is context free.
- Give some examples of semantic analysis.
 Def-before-use, no redefinition, same type, scoping ...
- What is Syntax Directed Translation? The parsing process and parse trees are to direct semantic analysis and the translation of the program (a.k.a., CFG-driven translation)
- How to augment grammar for semantic analysis?
 Semantic attributes for symbols, rules/actions for productions
- What are SDD and SDT?
 SDD = Syntax Directed Definitions, SDT = SD Translation Schemes
- What is an synthesized attribute?
 Defined by attribute values of node N's children and N itself





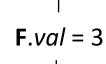


| Production Rules | Semantic Rules |
|-----------------------------|--------------------------------|
| (1) L -> E | print(E. <i>val</i>) |
| (2) $E \rightarrow E_1 + T$ | $E.val = E_1.val + T.val$ |
| (3) E -> T | E.val = T.val |
| (4) T -> $T_1 * F$ | $T.val = T_1.val \times F.val$ |
| (5) T -> F | T.val = F.val |
| (6) F -> (E) | F.val = E.val |
| (7) F -> digit | F.val = digit.lexval |



Input:

$$3*5+4$$



T.val = 3

digit.lexval = 5

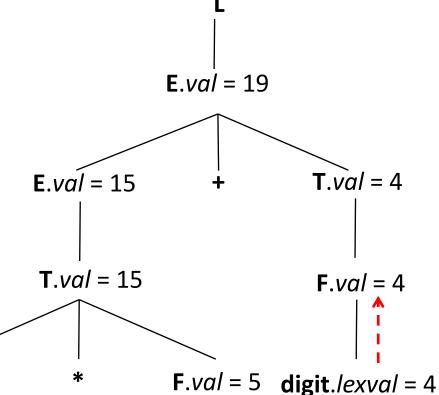






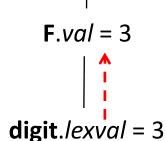


| Production Rules | Semantic Rules |
|-----------------------------|--------------------------------|
| (1) L -> E | print(E. <i>val</i>) |
| (2) $E \rightarrow E_1 + T$ | $E.val = E_1.val + T.val$ |
| (3) E -> T | E.val = T.val |
| (4) T -> $T_1 * F$ | $T.val = T_1.val \times F.val$ |
| (5) T -> F | T.val = F.val |
| (6) F -> (E) | F.val = E.val |
| (7) F -> digit | F.val = digit.lexval |



Input:

$$3*5+4$$



T.val = 3

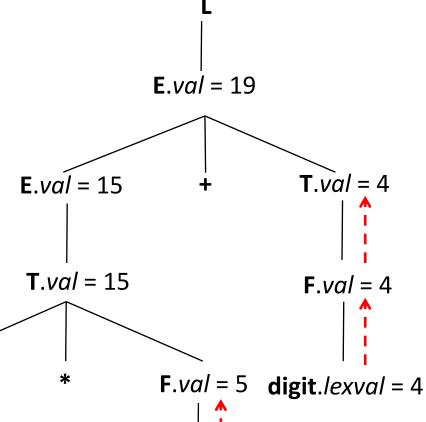
digit.*lexval* = 5



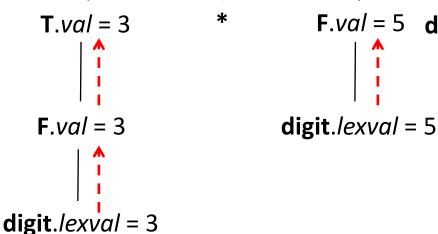




| Production Rules | Semantic Rules |
|-----------------------------|--------------------------------|
| (1) L -> E | print(E. <i>val</i>) |
| (2) $E \rightarrow E_1 + T$ | $E.val = E_1.val + T.val$ |
| (3) E -> T | E.val = T.val |
| (4) T -> $T_1 * F$ | $T.val = T_1.val \times F.val$ |
| (5) T -> F | T.val = F.val |
| (6) F -> (E) | F.val = E.val |
| (7) F -> digit | F.val = digit.lexval |



$$3*5+4$$

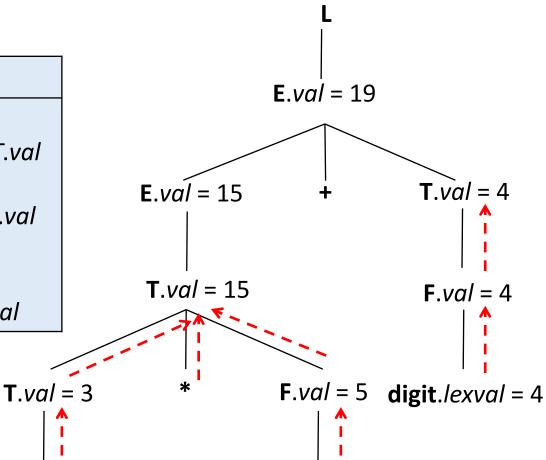




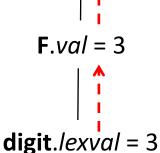


SDD:

| Production Rules | Semantic Rules |
|-----------------------------|--------------------------------|
| (1) L -> E | print(E. <i>val</i>) |
| (2) $E \rightarrow E_1 + T$ | $E.val = E_1.val + T.val$ |
| (3) E -> T | E.val = T.val |
| (4) T -> T ₁ * F | $T.val = T_1.val \times F.val$ |
| (5) T -> F | T.val = F.val |
| (6) F -> (E) | F.val = E.val |
| (7) F -> digit | F.val = digit.lexval |



$$3*5+4$$



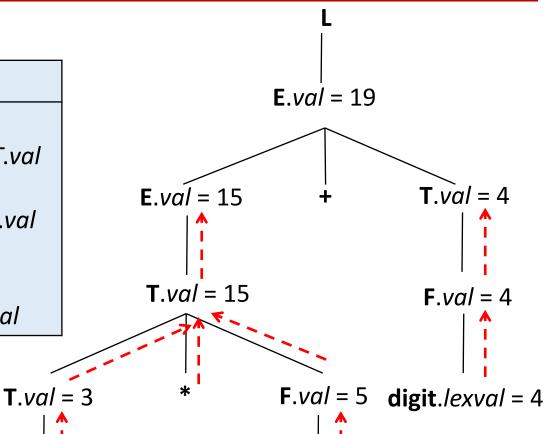




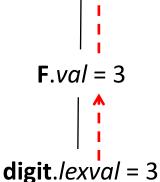




| Production Rules | Semantic Rules |
|-----------------------------|--------------------------------|
| (1) L -> E | print(E. <i>val</i>) |
| (2) E -> E ₁ + T | $E.val = E_1.val + T.val$ |
| (3) E -> T | E.val = T.val |
| (4) T -> T ₁ * F | $T.val = T_1.val \times F.val$ |
| (5) T -> F | T.val = F.val |
| (6) F -> (E) | F.val = E.val |
| (7) F -> digit | F.val = digit.lexval |



$$3*5+4$$



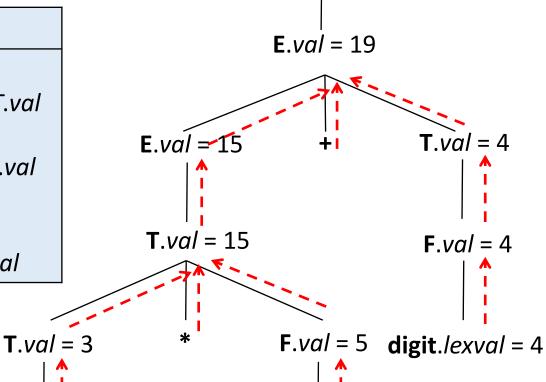






SDD:

| Production Rules | Semantic Rules |
|-----------------------------|--------------------------------|
| (1) L -> E | print(E. <i>val</i>) |
| (2) $E \rightarrow E_1 + T$ | $E.val = E_1.val + T.val$ |
| (3) E -> T | E.val = T.val |
| (4) T -> $T_1 * F$ | $T.val = T_1.val \times F.val$ |
| (5) T -> F | T.val = F.val |
| (6) F -> (E) | F.val = E.val |
| (7) F -> digit | F.val = digit.lexval |



digit.lexval = 5

Input:

$$3*5+4$$

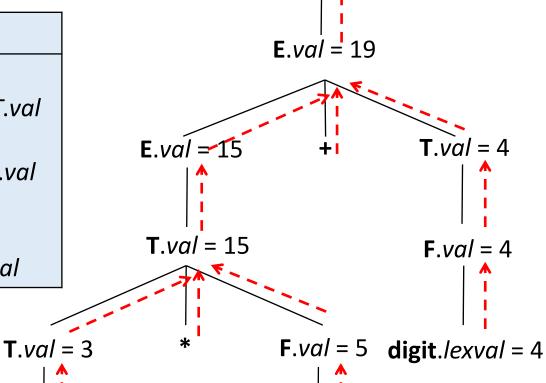




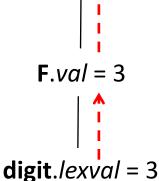
 $\mathbf{F}.val = 3$

SDD:

| Production Rules | Semantic Rules |
|-----------------------------|--------------------------------|
| (1) L -> E | print(E. <i>val</i>) |
| (2) E -> E ₁ + T | $E.val = E_1.val + T.val$ |
| (3) E -> T | E.val = T.val |
| (4) T -> T ₁ * F | $T.val = T_1.val \times F.val$ |
| (5) T -> F | T.val = F.val |
| (6) F -> (E) | F.val = E.val |
| (7) F -> digit | F.val = digit.lexval |



$$3*5+4$$









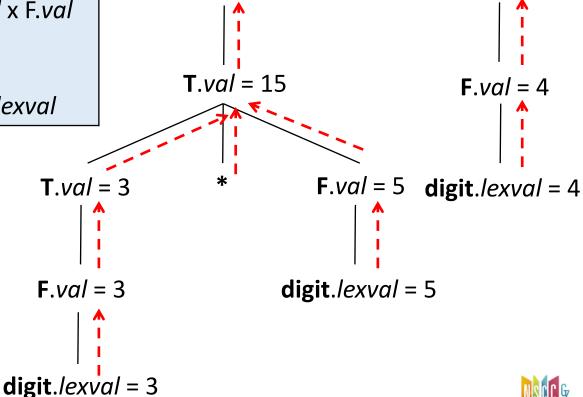


Side effect (副作用)

| Production Rules | Semantic Rules |
|-----------------------------|--------------------------------|
| (1) L -> E | print(E. <i>val</i>) |
| (2) $E \rightarrow E_1 + T$ | $E.val = E_1.val + T.val$ |
| (3) E -> T | E.val = T.val |
| (4) T -> T ₁ * F | $T.val = T_1.val \times F.val$ |
| (5) T -> F | T.val = F.val |
| (6) F -> (E) | F.val = E.val |
| (7) F -> digit | F.val = digit.lexval |



$$3*5+4$$



E.val = 15

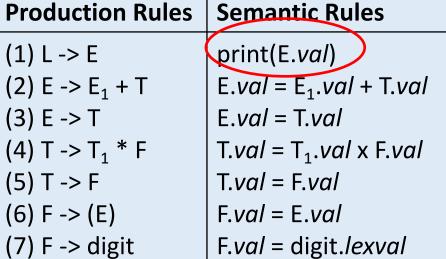
E.*val* = 19

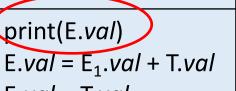


T.val = 4



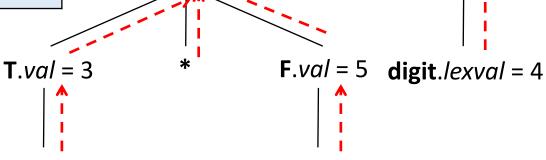
Side effect (副作用)





Input:

$$3*5+4$$



E.val = 15

T.val = 15

Annotated parse tree (标注分析树)

digit.lexval = 5

E.*val* = 19

digit.lexval = 3

 $\mathbf{F}.val = 3$



T.val = 4

 $\mathbf{F}.val = 4$



Example: Inherited Attribute[继承]

SDD:

| Production Rules | Semantic Rules |
|------------------------------|---|
| (1) D -> T L | L.inh = T.type |
| (2) T -> int | T. <i>type</i> = int |
| (3) T -> float | T.type = float |
| (4) L -> L ₁ , id | $L_1.inh = L.inh$ |
| | addtype(id. <i>entry,</i> L. <i>inh</i>) |
| (5) L -> id | addtype(id. <i>entry,</i> L. <i>inh</i>) |

T has synthesized attribute typeL has inherited attribute inh

Variable declaration of type int/float followed by a list of IDs:

- (1) Declaration: a type *T* followed by a list of *L* identifiers
- (2) Evaluate the synthesized attribute *T.type* (int)
- (3) Evaluate the synthesized attribute *T.type* (float)
- (4) Pass down type, and add type to symbol table entry for the identifier
- (5) Add type to symbol table





Example: Inherited Attribute[继承]

SDD:

| Production Rules | Semantic Rules | | |
|--|--|-------------------|--|
| (1) D -> T L (2) T -> int (3) T -> float | L.inh = T.type T.type = int T.type = float | | Thas synthesized attribute type Lhas inherited attribute inh |
| ` ' | $L_1.inh = L.inh$ | Pointing | to a symbol-table[符号表] object |
| | addtype (id.entry) | | |
| (5) L -> id | addtype(id. <i>entry</i> , | , L. <i>inh</i>) | |

Variable declaration of type int/float followed by a list of IDs:

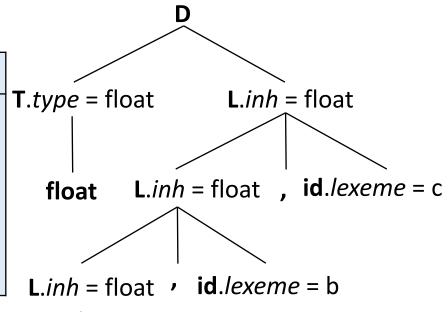
- (1) Declaration: a type *T* followed by a list of *L* identifiers
- (2) Evaluate the synthesized attribute *T.type* (int)
- (3) Evaluate the synthesized attribute *T.type* (float)
- (4) Pass down type, and add type to symbol table entry for the identifier
- (5) Add type to symbol table





SDD:

| Production Rules | Semantic Rules |
|------------------------------|--|
| (1) D -> T L | L.inh = T.type |
| (2) T -> int | T. <i>type</i> = int |
| (3) T -> float | T.type = float |
| (4) L -> L ₁ , id | $L_1.inh = L.inh$ |
| | addtype(id.entry, L.inh) |
| (5) L -> id | addtype(id. <i>entry</i> , L. <i>inh</i>) |



Input:

float a, b, c

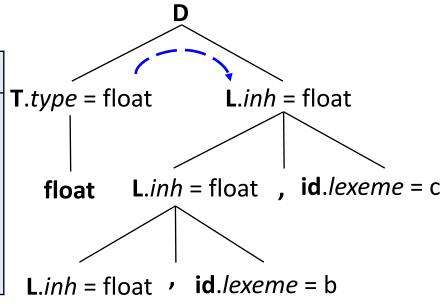






SDD:

| Production Rules | Semantic Rules |
|------------------------------|--|
| (1) D -> T L | L.inh = T.type |
| (2) T -> int | T. <i>type</i> = int |
| (3) T -> float | T.type = float |
| (4) L -> L ₁ , id | $L_1.inh = L.inh$ |
| | addtype(id. <i>entry</i> , L. <i>inh</i>) |
| (5) L -> id | addtype(id. <i>entry</i> , L. <i>inh</i>) |



Input:

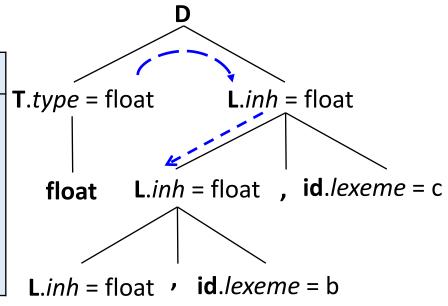
float a, b, c





SDD:

| Production Rules | Semantic Rules |
|------------------------------|---|
| (1) D -> T L | L.inh = T.type |
| (2) T -> int | T.type = int |
| (3) T -> float | T.type = float |
| (4) L -> L ₁ , id | $L_1.inh = L.inh$ |
| | addtype(id.entry, L.inh) |
| (5) L -> id | addtype(id. <i>entry,</i> L. <i>inh</i>) |



Input:

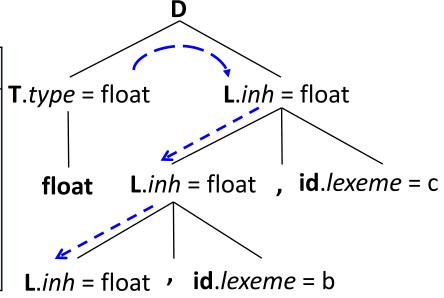
float a, b, c





SDD:

| Production Rules | Semantic Rules |
|------------------------------|--|
| (1) D -> T L | L.inh = T.type |
| (2) T -> int | T. <i>type</i> = int |
| (3) T -> float | T.type = float |
| (4) L -> L ₁ , id | $L_1.inh = L.inh$ |
| | addtype(id. <i>entry</i> , L. <i>inh</i>) |
| (5) L -> id | addtype(id. <i>entry</i> , L. <i>inh</i>) |



Input:

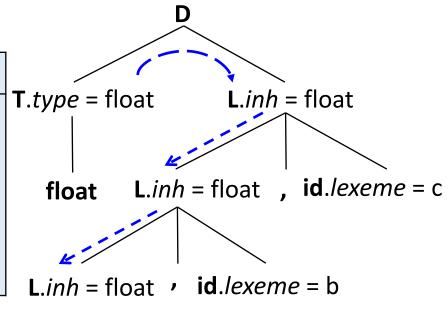
float a, b, c





SDD:

| Production Rules | Semantic Rules |
|------------------------------|--|
| (1) D -> T L | L.inh = T.type |
| (2) T -> int | T. <i>type</i> = int |
| (3) T -> float | T.type = float |
| (4) L -> L ₁ , id | $L_1.inh = L.inh$ |
| | addtype(id.entry, L.inh) |
| (5) L -> id | addtype(id. <i>entry</i> , L. <i>inh</i>) |



Input:

float a, b, c

type depends on child inh depends on sibling or parent





The Concepts

- Side effect[副作用]
 - 一般属性值计算(基于属性值或常量进行的)之外的功能
 - 例如: code generation, print results, modify symbol table ...
- Attribute grammar[属性文法]
 - 一个没有副作用的SDD
 - The rules define the value of an attribute purely in terms of the value of other attributes and constants[属性文法的规则仅仅通过其他属性值和常量来定义一个属性值]
- Annotated parse-tree[标注分析树]
 - 每个节点都带有属性值的分析树
 - A parse tree showing the value(s) of its attribute(s)
 - a.k.a., attribute parse tree[属性分析树]
 - Can also have actions being annotated[也可标注语义动作]





Dependence Graph[依赖图]

- Dependence relationship[依赖关系]
 - Before evaluating an attribute at a node of a parse tree, we must evaluate all attributes it depends on[按照依赖顺序计算]
- Dependency graph[依赖图]
 - While the annotated parse tree shows the values of attributes,
 a dependency graph helps determine how those values can be
 computed[依赖图决定属性值的计算]
 - Depicts the flow of info among the attribute instances in a particular parse tree[描绘了分析树的属性信息流]
 - Directed graph where edges are dependence relationships between attributes
 - For each parse-tree node X, there's a graph node for each attr of X
 - If attr X.a depends on attr Y.b, then there's one directed edge from Y.b to X.a

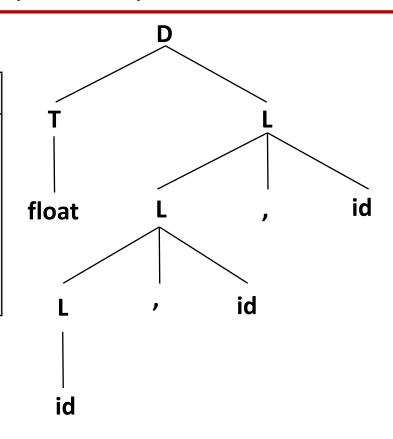




Example: Dependency Graph

SDD:

| Production Rules | Semantic Rules |
|------------------------------|--|
| (1) D -> T L | L.inh = T.type |
| (2) T -> int | T.type = int |
| (3) T -> float | T.type = float |
| (4) L -> L ₁ , id | $L_1.inh = L.inh$ |
| | addtype(id. <i>entry</i> , L. <i>inh</i>) |
| (5) L -> id | addtype(id. <i>entry</i> , L. <i>inh</i>) |



Input:

float a, b, c

'entry' is dummy attribute for the addtype()

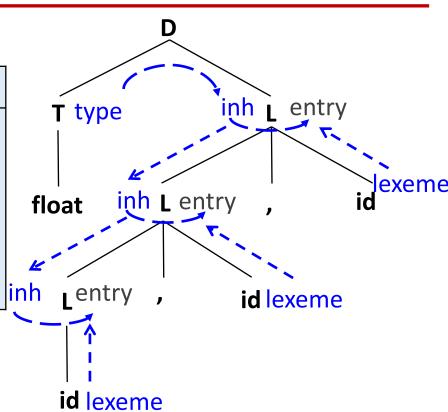




Example: Dependency Graph

SDD:

| Production Rules | Semantic Rules |
|-------------------------|--|
| (1) D -> T L | L.inh = T.type |
| (2) T -> int | T. <i>type</i> = int |
| (3) T -> float | T.type = float |
| (4) L -> L_1 , id | $L_1.inh = L.inh$ |
| | addtype(id. <i>entry,</i> L. <i>inh</i>) |
| (5) L -> id | addtype(id. <i>entry</i> , L. <i>inh</i>) |



Input:

float a, b, c

'entry' is dummy attribute for the addtype()





Evaluation Order[属性值计算顺序]

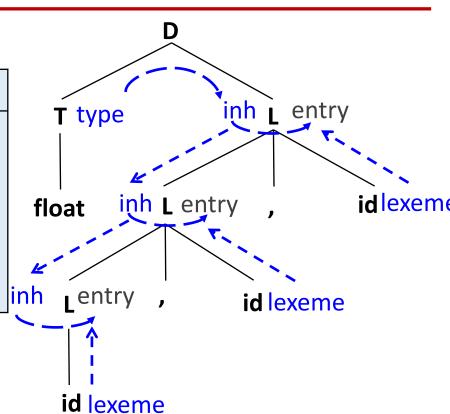
- Ordering the evaluation of attributes[计算顺序]
 - Dependency graph characterizes possible orders in which we can evaluate the attributes at the various nodes of a parse-tree
- If the graph has an edge from node *M* to node *N*, then the attribute associated with *M* must be evaluated before *N*[用图的边来确定计算顺序]
 - Thus, the only allowable orders of evaluation are those sequences of nodes N_1 , N_2 , ..., N_k such that if there is an edge of the graph from N_i to N_j , then i < j
 - Such an ordering embeds a directed graph into a linear order,
 and is called a topological sort[拓扑排序] of the graph
 - If there's any cycle in the graph, then there are no topological sorts, i.e., no way to evaluate the SDD on this parse tree
 - If there are no cycles, then there is always at least one topological sort





SDD:

| Production Rules | Semantic Rules |
|------------------------------|---|
| (1) D -> T L | L.inh = T.type |
| (2) T -> int | T. <i>type</i> = int |
| (3) T -> float | T.type = float |
| (4) L -> L ₁ , id | $L_1.inh = L.inh$ |
| | addtype(id. <i>lexeme</i> , L. <i>inh</i>) |
| (5) L -> id | addtype(id. <i>lexeme</i> , L. <i>inh</i>) |



Input:

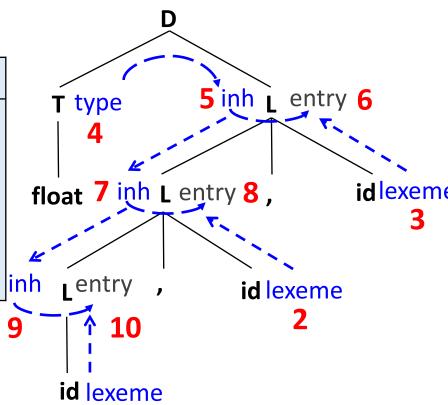
float a, b, c





SDD:

| Production Rules | Semantic Rules |
|------------------------------|---|
| (1) D -> T L | L.inh = T.type |
| (2) T -> int | T. <i>type</i> = int |
| (3) T -> float | T.type = float |
| (4) L -> L ₁ , id | $L_1.inh = L.inh$ |
| | addtype(id. <i>lexeme</i> , L. <i>inh</i>) |
| (5) L -> id | addtype(id. <i>lexeme</i> , L. <i>inh</i>) |



Input:

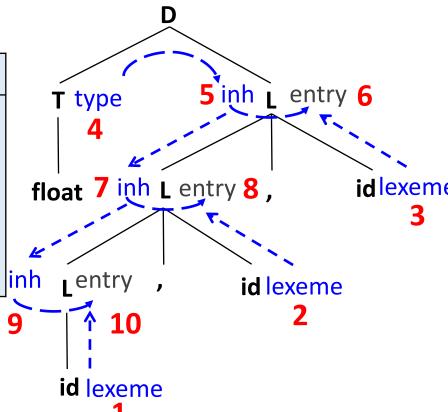
float a, b, c





SDD:

| Production Rules | Semantic Rules |
|------------------------------|---|
| (1) D -> T L | L.inh = T.type |
| (2) T -> int | T. <i>type</i> = int |
| (3) T -> float | T.type = float |
| (4) L -> L ₁ , id | $L_1.inh = L.inh$ |
| | addtype(id. <i>lexeme</i> , L. <i>inh</i>) |
| (5) L -> id | addtype(id. <i>lexeme</i> , L. <i>inh</i>) |



Input:

float a, b, c

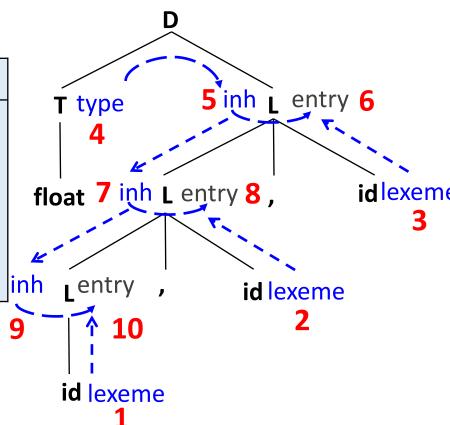
Topological sort: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10





SDD:

| Production Rules | Semantic Rules |
|------------------------------|---|
| (1) D -> T L | L.inh = T.type |
| (2) T -> int | T. <i>type</i> = int |
| (3) T -> float | T.type = float |
| (4) L -> L ₁ , id | $L_1.inh = L.inh$ |
| | addtype(id. <i>lexeme</i> , L. <i>inh</i>) |
| (5) L -> id | addtype(id. <i>lexeme</i> , L. <i>inh</i>) |



Input:

float a, b, c

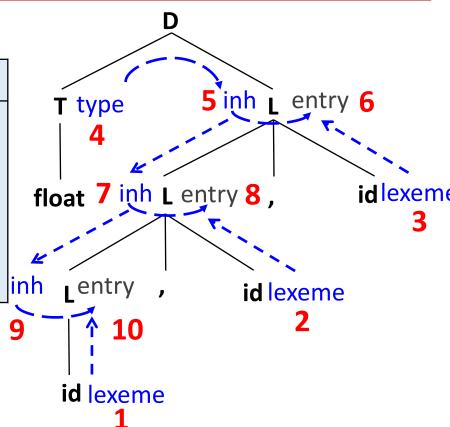
Topological sort: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10





SDD:

| Production Rules | Semantic Rules |
|------------------------------|---|
| (1) D -> T L | L.inh = T.type |
| (2) T -> int | T. <i>type</i> = int |
| (3) T -> float | T.type = float |
| (4) L -> L ₁ , id | $L_1.inh = L.inh$ |
| | addtype(id. <i>lexeme</i> , L. <i>inh</i>) |
| (5) L -> id | addtype(id. <i>lexeme</i> , L. <i>inh</i>) |



Input:

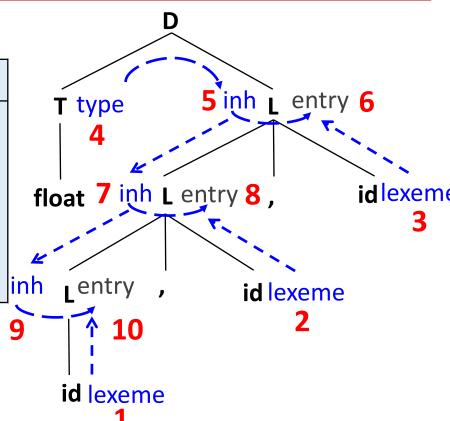
float a, b, c





SDD:

| Production Rules | Semantic Rules |
|------------------------------|---|
| (1) D -> T L | L.inh = T.type |
| (2) T -> int | T. <i>type</i> = int |
| (3) T -> float | T.type = float |
| (4) L -> L ₁ , id | $L_1.inh = L.inh$ |
| | addtype(id. <i>lexeme</i> , L. <i>inh</i>) |
| (5) L -> id | addtype(id. <i>lexeme</i> , L. <i>inh</i>) |



Input:

float a, b, c





Evaluation Order (cont.)

- Before evaluating an attribute at a node of a parse tree, we must evaluate all attributes it depends on[依赖关系]
 - Synthesized: evaluate children first, then the node itself
 Any bottom-up order is fine
 - For SDD's with both inherited and synthesized attributes,
 there's no guarantee that there is even one evaluation order
- Difficult to determine whether exist any circularities[非常难确定是否有循环依赖]
 - But, there are useful subclasses of SDD's that are sufficient to guarantee that an evaluation order exists
 - Such classes do not permit graphs with cycles

Production

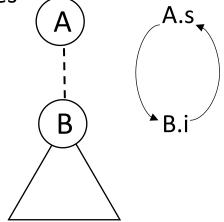
 $A \rightarrow B$

Semantic Rules

$$A.s = B.i$$
;

$$B.i = A.s + 1;$$







- An SDD is **S-attributed** if every attribute is <u>synthesized</u>[只 具有综合属性]
- If an SDD is S-attributed (S-SDD)
 - We can evaluate its attributes in any bottom-up order of the nodes of the parse-tree[任何自底向上的顺序计算属性值]
 - Can be implemented during bottom-up parsing[LR分析中实现]

| Production Rules | Semantic Rules |
|-----------------------------|--------------------------------|
| (1) L -> E | print(E. <i>val</i>) |
| (2) E -> E ₁ + T | $E.val = E_1.val + T.val$ |
| (3) E -> T | E.val = T.val |
| (4) T -> T ₁ * F | $T.val = T_1.val \times F.val$ |
| (5) T -> F | T.val = F.val |
| (6) F -> (E) | F.val = E.val |
| (7) F -> digit | F.val = digit.lexval |





- An SDD is L-attributed (L-SDD) if
 - Between the attributes associated with a production body, dependency-graph edges can go from <u>left to right</u>, but not from right to left[依赖图的边只能从左到右]
 - More precisely: each attribute must be either synthesized, or inherited but with the rules limited as follows: suppose A -> X₁X₂...X_n, the inherited attribute X_i.a only depends on
 - Inherited attributes associated with A
 - \blacksquare Either syn or inh attributes of $X_1, X_2, ..., X_{i-1}$ located to the left of X_i
 - Either syn or inh attributes of X_i itself, but no cycles formed by the attributes of this X_i
- Can be implemented during top-down parsing[LL分析中]





- An SDD is L-attributed (L-SDD) if
 - Between the attributes associated with a production body, dependency-graph edges can go from left to right, but not from right to left[依赖图的边只能从左到右]
 - More precisely: each attribute must be either **synthesized**, or **inherited** but with the rules limited as follows: suppose A -> X₁X₂...X_n, the inherited attribute X_i.a only depends on Why not synthesized?

 - \blacksquare Either syn or inh attributes of $X_1, X_2, ..., X_{i-1}$ located to the left of X_i
 - □ Either syn or inh attributes of X_i itself, but no cycles formed by the attributes of this X_i
- Can be implemented during top-down parsing[LL分析中]





- An SDD is L-attributed (L-SDD) if
 - Between the attributes associated with a production body, dependency-graph edges can go from left to right, but not from right to left[依赖图的边只能从左到右]
 - More precisely: each attribute must be either **synthesized**, or **inherited** but with the rules limited as follows: suppose A -> X₁X₂...X_n, the inherited attribute X_i.a only depends on Why not synthesized?

 Inherited attributes associated with A Cycle: X_i depends on A, A.s depends on X_i

 - \blacksquare Either syn or inh attributes of X_1 , X_2 , ..., X_{i-1} located to the left of X_i
 - □ Either syn or inh attributes of X_i itself, but no cycles formed by the attributes of this X_i
- Can be implemented during top-down parsing[LL分析中]





- An SDD is L-attributed (L-SDD) if
 - Between the attributes associated with a production body, dependency-graph edges can go from left to right, but not from right to left[依赖图的边只能从左到右]
 - More precisely: each attribute must be either **synthesized**, or **inherited** but with the rules limited as follows: suppose A -> X₁X₂...X_n, the inherited attribute X_i.a only depends on Why not synthesized?

 Inherited attributes associated with A Cycle: X_i depends on A, A.s depends on X_i

 - \blacksquare Either syn or inh attributes of X_1 , X_2 , ..., X_{i-1} located to the left of X_i
 - □ Either syn or inh attributes of X_i itself, but no cycles formed by the attributes of this X_i
- Can be implemented during top-down parsing[LL分析中]

| Production Rules | Semantic Rules |
|-------------------------|-------------------|
| A -> B C | A.s = B.b |
| | B.i = f(C.c, A.s) |





- An SDD is L-attributed (L-SDD) if
 - Between the attributes associated with a production body, dependency-graph edges can go from left to right, but not from right to left[依赖图的边只能从左到右]
 - More precisely: each attribute must be either **synthesized**, or **inherited** but with the rules limited as follows: suppose A -> X₁X₂...X_n, the inherited attribute X_i.a only depends on Why not synthesized?

 Inherited attributes associated with A Cycle: X_i depends on A, A.s depends on X_i

 - \blacksquare Either syn or inh attributes of X_1 , X_2 , ..., X_{i-1} located to the left of X_i
 - □ Either syn or inh attributes of X_i itself, but no cycles formed by the attributes of this X_i
- Can be implemented during top-down parsing[LL分析中]

S-SDD or L-SDD?

| Production Rules | Semantic Rules |
|-------------------------|-------------------|
| A -> B C | A.s = B.b |
| | B.i = f(C.c, A.s) |





L-Attributed Definitions[L-属性定义]

- An SDD is L-attributed (L-SDD) if
 - Between the attributes associated with a production body, dependency-graph edges can go from left to right, but not from right to left[依赖图的边只能从左到右]
 - More precisely: each attribute must be either **synthesized**, or inherited but with the rules limited as follows: suppose A -> X₁X₂...X_n, the inherited attribute X_i.a only depends on Why not synthesized?

 Inherited attributes associated with A Cycle: X_i depends on A, A.s depends on X_i

Rules

- □ Either syn or inh attributes of X_1 , X_2 , ..., X_{i-1} located to the left of X_i
- □ Either syn or inh attributes of X_i itself, but no cycles formed by the attributes of this X_i
- Can be implemented during top-down parsing[LL分析中]

| | Production Rules | Semantic Rules |
|-----------------|-------------------------|-------------------|
| S-SDD or L-SDD? | | A.s = B.b |
| | | B.i = f(C.c, A.s) |
| _ | 1-+ C CDD, D !!- ! - = | |

Not S-SDD: B.i is inh



L-Attributed Definitions[L-属性定义]

- An SDD is L-attributed (L-SDD) if
 - Between the attributes associated with a production body, dependency-graph edges can go from left to right, but not from right to left[依赖图的边只能从左到右]
 - More precisely: each attribute must be either **synthesized**, or **inherited** but with the rules limited as follows: suppose A -> X₁X₂...X_n, the inherited attribute X_i.a only depends on Why not synthesized?

 Inherited attributes associated with A Cycle: X_i depends on A, A.s depends on X_i

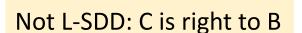
 - \blacksquare Either syn or inh attributes of X_1 , X_2 , ..., X_{i-1} located to the left of X_i
 - □ Either syn or inh attributes of X_i itself, but no cycles formed by the attributes of this X_i
- Can be implemented during top-down parsing[LL分析中]

S-SDD or L-SDD?

| Semantic Rules |
|-------------------------------|
| A.s = B.b $B.i = f(C.c, A.s)$ |
| |



Not S-SDD: B.i is inh





L-Attributed Definitions[L-属性定义]

- An SDD is L-attributed (L-SDD) if
 - Between the attributes associated with a production body, dependency-graph edges can go from left to right, but not from right to left[依赖图的边只能从左到右]
 - More precisely: each attribute must be either **synthesized**, or **inherited** but with the rules limited as follows: suppose A -> X₁X₂...X_n, the inherited attribute X_i.a only depends on Why not synthesized?

 Inherited attributes associated with A Cycle: X_i depends on A, A.s depends on X_i

 - \blacksquare Either syn or inh attributes of X_1 , X_2 , ..., X_{i-1} located to the left of X_i
 - □ Either syn or inh attributes of X_i itself, but no cycles formed by the attributes of this X_i
- Can be implemented during top-down parsing[LL分析中]

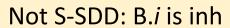
S-SDD or L-SDD?

Production Rules Semantic Rules

A -> B C

A.s = B.b

Not L-SDD: A.s is syn attr





Not L-SDD: C is right to B



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 <u>semantic actions</u> 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 <u>S-attributed</u>, the underlying grammar is <u>LR-parsable</u>
 - SDD is <u>L-attributed</u>, the underlying grammar is <u>LL-parsable</u>
 - 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[SDD->SDT的转换]
 - Placing each action at the end of the production[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. <i>val</i>) |
| (2) E -> E ₁ + T | $ E.val = E_1.val + T.val $ |
| (3) E -> T | E.val = T.val |
| (4) T -> T ₁ * F | $T.val = T_1.val \times F.val$ |
| (5) T -> F | T.val = F.val |
| (6) F -> (E) | F.val = E.val |
| (7) F -> digit | F.val = digit.lexval |





== Implement S-SDD ==

- Convert S-attributed SDD to SDT by[SDD->SDT的转换]
 - Placing each action at the end of the production[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

| | | 201 |
|---|--|-----|
| 1 | | |

| Production Rules | Semantic Rules |
|-----------------------------|--------------------------------|
| (1) L -> E | print (E. <i>val</i>) |
| (2) $E \rightarrow E_1 + T$ | $E.val = E_1.val + T.val$ |
| (3) E -> T | E.val = T.val |
| (4) T -> T ₁ * F | $T.val = T_1.val \times F.val$ |
| (5) T -> F | T.val = F.val |
| (6) F -> (E) | F.val = E.val |
| (7) F -> digit | F.val = digit.lexval |



| CFG with actions |
|---|
| (1) L -> E { print (E.val) } |
| (1) L -> E { print (E.val) } (2) E -> E ₁ + T { E.val = E_1 .val + T.val } |
| (2) $E \rightarrow E_1 + I \{ E.val = E_1.val + I.val \}$ (3) $E \rightarrow T \{ E.val = T.val \}$ (4) $T \rightarrow T_1 * F \{ T.val = T_1.val \times F.val \}$ (5) $T \rightarrow F \{ T.val = F.val \}$ |
| (4) $T \rightarrow T_1 * F \{ T.val = T_1.val \times F.val \}$ |
| (5) T -> F { T. val = F. val } |
| (b) F -> (E) { F.Val = E.Val } |
| (7) $F \rightarrow digit \{ F.val = digit.lexval \}$ |





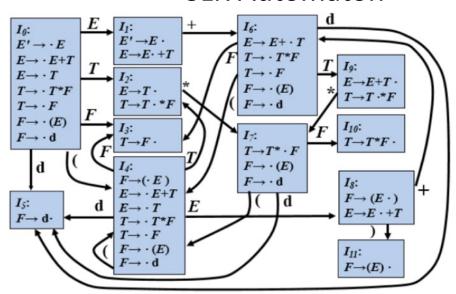
Implement S-SDD (cont.)

- If the underlying grammar of S-SDD is <u>LR parsable</u>
 - 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

(1) L -> E { print (E.val) } (2) E -> E₁ + T { E.val = E₁.val + T.val } (3) E -> T { E.val = T.val } (4) T -> T₁ * F { T.val = T₁.val x F.val } (5) T -> F { T.val = E.val } (6) F -> (E) { F.val = E.val } (7) F -> digit { F.val = digit.lexval }

SLR Automaton







- 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)

state
$$\rightarrow$$
 S_0 symbol \rightarrow $\$$





- 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)

```
state \rightarrow S_0 symbol \rightarrow \$ attribute \rightarrow -
```





- 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)

state
$$\rightarrow$$
 S₀ ... S_{m-2} S_{m-1} S_m symbol \rightarrow \$... X Y Z attribute \rightarrow - ... X.x Y.y Z.z





- 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)

state
$$\rightarrow$$
 S₀ ... S_{m-2} S_{m-1} S_m symbol \rightarrow \$... X Y Z attribute \rightarrow - ... X.x Y.y Z.z





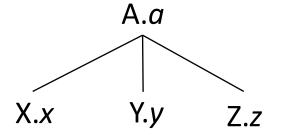
- Save synthesized attributes into the stack[栈中额外存放综合属性值]
 - Place the attributes along with the grammar symbols (or LR states that associated with these symbols) in records on stack
 - If there are multiple attributes
 - Make the records large enough or by putting pointers to records on the stack[栈记录足够大,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)





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

$$A \rightarrow XYZ \{ A.a = f(X.x, Y.y, Z.z) \}$$



state
$$\rightarrow$$
 S₀ ... S_{m-2} S_{m-1} S_m symbol \rightarrow \$... X Y Z attribute \rightarrow - ... X.x Y.y Z.z

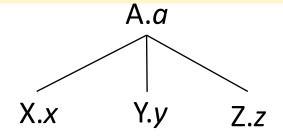




- Rewrite the actions to manipulate the parser stack[语义动作]
 - The manipulation can be done automatically by the parser

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

$$A \rightarrow XYZ \{ A.a = f(X.x, Y.y, Z.z) \}$$



state
$$\rightarrow$$
 S₀ ... S_{m-2} S_{m-1} S_m symbol \rightarrow \$... X Y Z attribute \rightarrow - ... X.x Y.y Z.z





- 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 \rightarrow XYZ \{ A.a = f(X.x, Y.y, Z.z) \}
X.x \qquad Y.y \qquad Z.z
```





- 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.a
          A \rightarrow XYZ \{ A.a = f(X.x, Y.y, Z.z) \}
                                                             X.x
       state \rightarrow S<sub>0</sub> ··· S<sub>m-2</sub> S<sub>m-1</sub> S<sub>m</sub>
                                                              state • S<sub>0</sub> ···
     symbol → $ ··· X Y Z
                                                            symbol • $ ····
    attribute \rightarrow - ... X.x Y.y Z.z
                                                                                  A.a
                                                          attribute → - ····
```





top

top

- 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. <i>val</i>) | { 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 | $T.val = T_1.val \times F.val$ | { 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 | |





| Productions | Semantic Actions | $\begin{bmatrix} I_{\theta} : & E' \to E \\ E' \to E \end{bmatrix} \xrightarrow{I_{\theta}} \begin{bmatrix} I_{\theta} : & E \\ E \to E + \cdot T \end{bmatrix}$ |
|----------------------------|---|--|
| (1) L -> E | { print(stack[top].val); } | $ \begin{bmatrix} E \to \cdot E + T \\ E \to \cdot T \end{bmatrix} T $ $ \begin{bmatrix} E \to E \cdot + T \\ T \to \cdot T * F \end{bmatrix} $ $ T \to \cdot T * F $ $ T \to \cdot T *$ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; | $ \begin{array}{c ccccccccccccccccccccccccccccccccccc$ |
| | top = top -2; } | $F \rightarrow (F)$ F |
| (3) E -> T | | $ F \rightarrow \cdot d $ $T \rightarrow F$. |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; | $ \begin{array}{ c c c c c c c c c c c c c c c c c c c$ |
| | top = top -2; } | $F \rightarrow \cdot d$ I_8 : |
| (5) T -> F | | $\begin{vmatrix} I_5 \\ \vdots \end{vmatrix}$ d $E \rightarrow T$ E |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; | $ \begin{array}{c c} \hline F \rightarrow d \cdot \\ \hline T \rightarrow \cdot T^*F \\ \hline T \rightarrow \cdot F \end{array} $ |
| | top = top -2; } | |
| (7) F -> digit | | $F \rightarrow \cdot \mathbf{d}$ $F \rightarrow (E) \cdot$ |

Input: 3 * 5 + 4

```
state \rightarrow S_0 symbol \rightarrow $ attribute \rightarrow -
```





| | | d |
|----------------------------|---|---|
| Productions | Semantic Actions | $ \begin{vmatrix} I_{\theta} : & E \\ E' \to E \end{vmatrix} \xrightarrow{E} \xrightarrow{I_1} \begin{vmatrix} E \\ E' \to E \end{vmatrix} \xrightarrow{I_2} $ |
| (1) L -> E | { print(stack[top].val); } | $\begin{bmatrix} E \rightarrow \cdot E + T \\ E \rightarrow \cdot T \end{bmatrix} T \xrightarrow{E \rightarrow E \cdot + T} \begin{bmatrix} F & T \rightarrow \cdot T * F \\ T \rightarrow \cdot F \end{bmatrix} \xrightarrow{I_0:} T \xrightarrow$ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; | $T \rightarrow T^*F$ $E \rightarrow T \cdot $ $*$ $E \rightarrow E + T \cdot $ $T \rightarrow T \cdot F$ $*$ $T \rightarrow T \cdot F$ |
| | top = top -2; } | $F \to F$ |
| (3) E -> T | | $ F \rightarrow \cdot d $ $\uparrow \uparrow_{\rightarrow F}$. |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; | |
| | top = top -2; } | $F \rightarrow (\cdot E)$ $F \rightarrow \cdot d$ $I_g:$ |
| (5) T -> F | | $\begin{array}{c ccccccccccccccccccccccccccccccccccc$ |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; | $ \begin{array}{c c} \hline F \rightarrow d \cdot \\ \hline T \rightarrow \cdot T^*F \\ \hline T \rightarrow \cdot F \end{array} $ |
| | top = top -2; } | |
| (7) F -> digit | | $F \rightarrow \cdot d$ $F \rightarrow (E) \cdot$ |

```
state \rightarrow S_0 symbol \rightarrow $
```

attribute → -





| | | d |
|----------------------------|--|---|
| Productions | Semantic Actions | $\begin{bmatrix} I_{\theta} : & E' \to E \\ E' \to E \end{bmatrix} \xrightarrow{E} \begin{bmatrix} I_{1} : & + \\ E' \to E \end{bmatrix} \xrightarrow{I_{\theta}} \begin{bmatrix} I_{\theta} : & + \\ E \to E + \cdot T \end{bmatrix}$ |
| (1) L -> E | { print(stack[top].val); } | $ \begin{bmatrix} E \to E + T \\ E \to T \end{bmatrix} $ $ T \to T * F $ $ T \to T \to T * F $ $ T \to T * F $ $ T \to T \to T * F $ $ T \to T \to T \to T \to T $ $ T \to T \to T \to T$ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; top = top -2; } | $ \begin{array}{c ccccccccccccccccccccccccccccccccccc$ |
| (3) E -> T | | $ F \rightarrow d \nearrow I_{7}$ |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; top = top -2; } | $ \begin{array}{ c c c c c c c c c c c c c c c c c c c$ |
| (5) T -> F | | $ I_5:$ $ \mathbf{d} _{E \to T} E $ $ \mathbf{d} _{E \to F \to T} E $ |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; top = top -2; } | $ \begin{array}{c c} & T \to T^*F \\ & T \to F \\ & F \to \cdot (E) \end{array} $ |
| (7) F -> digit | | $F \rightarrow \cdot d$ $F \rightarrow (E) \cdot$ |

state
$$\rightarrow$$
 S₀ S₅ symbol \rightarrow \$ d attribute \rightarrow - 3





| | * | E — d |
|----------------------------|---|--|
| Productions | Semantic Actions | $\begin{bmatrix} I_{\theta} : & E' \to E \\ E' \to E \end{bmatrix} \xrightarrow{E} \begin{bmatrix} I_{1} : & + \\ E' \to E \end{bmatrix} \xrightarrow{I_{\theta}} \begin{bmatrix} I_{\theta} : & + \\ E \to E + \cdot T \end{bmatrix}$ |
| (1) L -> E | { print(stack[top].val); } | $ \begin{bmatrix} E \to \cdot E + T \\ E \to \cdot T \end{bmatrix} $ $ T \longrightarrow I_{2}: $ $ T \longrightarrow T^{*F} $ $ T \longrightarrow F $ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; | $T \rightarrow T^*F$ $E \rightarrow T \cdot E$ $*$ $F \rightarrow \cdot (E)$ $*$ $T \rightarrow T \cdot F$ |
| | top = top -2; } | $F \rightarrow \cdot (F)$ F |
| (3) E -> T | | $ F \rightarrow \cdot \mathbf{d} \nearrow T_{\rightarrow F}$. |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; | $ \begin{array}{ c c c c c c c c c c c c c c c c c c c$ |
| | top = top -2; } | $F \rightarrow \cdot d$ I_8 : |
| (5) T -> F | | I_{5} : d $I_{E \to T}$ I_{E} I_{E} I_{E} |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; | $ \begin{array}{c c} \hline F \rightarrow d \cdot \\ \hline T \rightarrow \cdot T^*F \\ \hline T \rightarrow \cdot F \end{array} $ |
| | top = top -2; } | |
| (7) F -> digit | | $F \rightarrow \cdot d$ $F \rightarrow (E) \cdot$ |

```
state \rightarrow S<sub>0</sub>
symbol \rightarrow $
attribute \rightarrow - 3
```





| | | d |
|----------------------------|---|--|
| Productions | Semantic Actions | $ \begin{vmatrix} I_{\theta} : & E' \to E \\ E' \to -E \end{vmatrix} \xrightarrow{E} \begin{vmatrix} I_{1} : & E' \to E \\ E \to E + \cdot T \end{vmatrix} $ |
| (1) L -> E | { print(stack[top].val); } | $\begin{bmatrix} E \to \cdot E + T \\ E \to \cdot T \end{bmatrix} T \xrightarrow{E \to E \cdot + T} \begin{bmatrix} F & T \to \cdot T^*F \\ T \to \cdot F \end{bmatrix} \xrightarrow{I_0:} T $ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; | $T \rightarrow T^*F$ $E \rightarrow T \cdot $ $*$ $F \rightarrow \cdot (E)$ $*$ $E \rightarrow E + T \cdot $ $T \rightarrow T \cdot ^*F$ |
| | top = top -2; } | $F \rightarrow \cdot (F)$ $F \rightarrow F$ |
| (3) E -> T | | $ F \rightarrow \cdot d $ $T \rightarrow F$. |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; | $ \begin{array}{ c c c c c c c c c c c c c c c c c c c$ |
| | top = top -2; } | $F \rightarrow (\cdot E)$ $E \rightarrow \cdot E + T$ $F \rightarrow \cdot d$ $I_{S}:$ $F \rightarrow (F_{+})$ |
| (5) T -> F | | $\begin{bmatrix} I_5: \\ F \end{bmatrix} \qquad \begin{bmatrix} \mathbf{d} \\ E \to F + T \end{bmatrix} $ |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; | $ \begin{array}{c c} \hline F \rightarrow d \cdot \\ \hline T \rightarrow \cdot T * F \\ \hline T \rightarrow \cdot F \end{array} $ |
| | top = top -2; } | |
| (7) F -> digit | | $F \rightarrow \cdot d$ $F \rightarrow (E) \cdot$ |

state
$$\rightarrow$$
 S₀ S₃ symbol \rightarrow F attribute \rightarrow - 3





| | * | E — d |
|----------------------------|---|--|
| Productions | Semantic Actions | $\begin{bmatrix} I_{\theta} : & E' \to E \\ E' \to E \end{bmatrix} \xrightarrow{E} \begin{bmatrix} I_{1} : & + \\ E' \to E \end{bmatrix} \xrightarrow{I_{\theta}} \begin{bmatrix} I_{\theta} : & + \\ E \to E + \cdot T \end{bmatrix}$ |
| (1) L -> E | { print(stack[top].val); } | $ \begin{bmatrix} E \to \cdot E + T \\ E \to \cdot T \end{bmatrix} $ $ T \longrightarrow I_{2}: $ $ T \longrightarrow T^{*F} $ $ T \longrightarrow F $ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; | $T \rightarrow T^*F$ $E \rightarrow T \cdot E$ $*$ $F \rightarrow \cdot (E)$ $*$ $T \rightarrow T \cdot F$ |
| | top = top -2; } | $F \rightarrow \cdot (F)$ F |
| (3) E -> T | | $ F \rightarrow \cdot \mathbf{d} \nearrow T_{\rightarrow F}$. |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; | $ \begin{array}{ c c c c c c c c c c c c c c c c c c c$ |
| | top = top -2; } | $F \rightarrow \cdot d$ I_8 : |
| (5) T -> F | | I_{5} : d $I_{E \to T}$ I_{E} I_{E} I_{E} |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; | $ \begin{array}{c c} \hline F \rightarrow d \cdot \\ \hline T \rightarrow \cdot T^*F \\ \hline T \rightarrow \cdot F \end{array} $ |
| | top = top -2; } | |
| (7) F -> digit | | $F \rightarrow \cdot d$ $F \rightarrow (E) \cdot$ |

```
state \rightarrow S<sub>0</sub>
symbol \rightarrow $
attribute \rightarrow - 3
```





| | | d |
|----------------------------|--|--|
| Productions | Semantic Actions | $\begin{bmatrix} I_{\theta} : & E' \to E \\ E' \to E \end{bmatrix} \xrightarrow{E} \begin{bmatrix} I_{1} : & + \\ E' \to E \end{bmatrix} \xrightarrow{I_{\theta} : E} \begin{bmatrix} I_{\theta} : & + \\ E \to E + \cdot T \end{bmatrix}$ |
| (1) L -> E | { print(stack[top].val); } | $ \begin{bmatrix} E \to E + T \\ E \to F + T \end{bmatrix} $ $ T \to T * F $ $ T \to T * F $ $ T \to F \to F \to F \to F $ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; top = top -2; } | $ \begin{array}{c c} T \to \cdot T^*F \\ T \to \cdot F \end{array} $ $ \begin{array}{c c} F \to \cdot (E) \\ T \to T \cdot *F \end{array} $ $ \begin{array}{c c} T \to T \cdot *F \end{array} $ |
| (3) E -> T | | $ F \rightarrow d \nearrow I_{7}$ |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; top = top -2; } | $ \begin{array}{c c} \hline d & (F) \\ \hline I_{f:} \\ F \to \cdot (E) \\ F \to \cdot d \end{array} $ $ I_{g:}$ |
| (5) T -> F | | I_{5} : d $\begin{vmatrix} E \rightarrow E + I \\ E \rightarrow T \end{vmatrix}$ E I |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; top = top -2; } | $ \begin{array}{c c} & T \to T^*F \\ & T \to F \\ & F \to \cdot (E) \end{array} $ |
| (7) F -> digit | | $F \rightarrow \cdot \mathbf{d}$ $F \rightarrow (E) \cdot$ |

state
$$\rightarrow$$
 S₀ S₂ symbol \rightarrow \$ T attribute \rightarrow - 3





| | | d |
|----------------------------|--|---|
| Productions | Semantic Actions | $\begin{bmatrix} I_{\theta} : & E' \to E \\ E' \to E \end{bmatrix} \xrightarrow{E} \begin{bmatrix} I_{1} : & + \\ E' \to E \end{bmatrix} \xrightarrow{I_{\theta}} \begin{bmatrix} I_{\theta} : & + \\ E \to E + \cdot T \end{bmatrix}$ |
| (1) L -> E | { print(stack[top].val); } | $ \begin{bmatrix} E \to E + T \\ E \to T \end{bmatrix} $ $ T \to T * F $ $ T \to T \to T * F $ $ T \to T * F $ $ T \to T \to T * F $ $ T \to T \to T \to T \to T $ $ T \to T \to T \to T$ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; top = top -2; } | $ \begin{array}{c c} T \to \cdot T^*F \\ T \to \cdot F \end{array} $ $ \begin{array}{c c} F \to \cdot (E) \\ T \to T \cdot *F \end{array} $ $ \begin{array}{c c} F \to \cdot (E) \\ F \to \cdot d \end{array} $ $ \begin{array}{c c} F \to E + T \cdot \\ T \to T \cdot *F \end{array} $ |
| (3) E -> T | | $ F \rightarrow d \nearrow I_{7}$ |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; top = top -2; } | $ \begin{array}{c c} \hline d & (F) \\ \hline I_{f:} \\ F \to \cdot (E) \\ F \to \cdot d \end{array} $ $ \begin{array}{c c} \hline I_{g:} \\ \hline I_{g:} \end{array} $ |
| (5) T -> F | | I_{5} : d $\begin{vmatrix} E \rightarrow E + T \\ E \rightarrow T \end{vmatrix}$ E] (d $\begin{vmatrix} F \rightarrow (E \cdot) \\ F \rightarrow F \rightarrow T \end{vmatrix}$ |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; top = top -2; } | $ \begin{array}{c c} & T \to T^*F \\ & T \to F \\ & F \to \cdot (E) \end{array} $ |
| (7) F -> digit | | $F \rightarrow \cdot d$ $F \rightarrow (E) \cdot$ |

Input: 3 * 5 + 4

state
$$\rightarrow$$
 S₀ S₂ symbol \rightarrow \$ T attribute \rightarrow - 3





| | | , d |
|----------------------------|---|---|
| Productions | Semantic Actions | $ \begin{vmatrix} I_{\theta} : \\ E' \to \cdot E \end{vmatrix} \xrightarrow{E} \begin{vmatrix} I_{1} : \\ E' \to E \cdot \\ E \to E + \cdot T \end{vmatrix} \xrightarrow{I_{\theta}} $ |
| (1) L -> E | { print(stack[top].val); } | $\begin{bmatrix} E \to \cdot E + T \\ E \to \cdot T \end{bmatrix} T \xrightarrow{E \to E \cdot + T} \begin{bmatrix} F & T \to \cdot T^*F \\ T \to \cdot F \end{bmatrix} \xrightarrow{I_g:} T \xrightarrow{F} T \xrightarrow{F} T$ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; | $ \begin{array}{ c c c c c c c c c c c c c c c c c c c$ |
| (3) E -> T | top = top -2; } | $F \rightarrow \cdot (E)$ F I_3 : |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; | $ \begin{array}{c c} & & & & & & & & & & & & & & & & & & &$ |
| | top = top -2; } | $F \rightarrow (\cdot E)$ $F \rightarrow \cdot d$ I_g : |
| (5) T -> F | | I_5 : d $E \to E + I$ E) d $E \to E + I$ |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; | $ \begin{array}{c c} F \to d \cdot \\ \hline T \to \cdot T^*F \\ T \to \cdot F \end{array} $ |
| | top = top -2; } | |
| (7) F -> digit | | $F \rightarrow \cdot d$ $F \rightarrow (E) \cdot$ |

state
$$\rightarrow$$
 S₀ S₂ symbol \rightarrow \$ T attribute \rightarrow - 3





| | | d |
|----------------------------|---|--|
| Productions | Semantic Actions | $\begin{bmatrix} I_{\emptyset} : & E' \to E \\ E' \to E \end{bmatrix} \xrightarrow{E} \begin{bmatrix} I_{1} : & + \\ E' \to E \end{bmatrix} \xrightarrow{I_{\emptyset} : E \to E + \cdot T} $ |
| (1) L -> E | { print(stack[top].val); } | $\begin{bmatrix} E \to \cdot E + T \\ E \to \cdot T \end{bmatrix} T \xrightarrow{E \to E \cdot + T} \begin{bmatrix} F & T \to \cdot T^*F \\ T \to \cdot F \end{bmatrix} \xrightarrow{I_0:} T $ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; | $T \rightarrow T^*F$ $E \rightarrow T \cdot $ $*$ $E \rightarrow E + T \cdot $ $T \rightarrow T \cdot *F$ $*$ $T \rightarrow T \cdot *F$ |
| | top = top -2; } | $F \rightarrow \cdot (F)$ F |
| (3) E -> T | | $ F \rightarrow \cdot d \nearrow I_{7}: \nearrow F I_{7}: \nearrow F I_{7}: \nearrow F I_{7}: \nearrow F I_{7}: F $ |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; | $ \begin{array}{ c c c c c c c c c c c c c c c c c c c$ |
| | top = top -2; } | $F \to (\cdot E)$ $F \to \cdot d$ $I_{\mathcal{S}}$ |
| (5) T -> F | | $ I_5:$ d $E \rightarrow T$ E |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; | $ \begin{array}{c c} \hline F \rightarrow d \\ \hline T \rightarrow \cdot T^*F \\ \hline T \rightarrow \cdot F \end{array} $ |
| | top = top -2; } | |
| (7) F -> digit | | $F \rightarrow \cdot \mathbf{d}$ |

state
$$\rightarrow$$
 S₀ S₂ S₇ symbol \rightarrow \$ T * attribute \rightarrow - 3 -





| | | d |
|----------------------------|--|---|
| Productions | Semantic Actions | $\begin{bmatrix} I_{\theta} : & E' \to E \\ E' \to E \end{bmatrix} \xrightarrow{E} \begin{bmatrix} I_{1} : & + \\ E' \to E \end{bmatrix} \xrightarrow{I_{\theta}} \begin{bmatrix} I_{\theta} : & + \\ E \to E + \cdot T \end{bmatrix}$ |
| (1) L -> E | { print(stack[top].val); } | $ \begin{bmatrix} E \to E + T \\ E \to T \end{bmatrix} $ $ T \to T * F $ $ T \to T \to T * F $ $ T \to T * F $ $ T \to T \to T * F $ $ T \to T \to T \to T \to T $ $ T \to T \to T \to T$ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; top = top -2; } | $ \begin{array}{c c} T \to \cdot T^*F \\ T \to \cdot F \end{array} $ $ \begin{array}{c c} F \to \cdot (E) \\ T \to T \cdot *F \end{array} $ $ \begin{array}{c c} F \to \cdot (E) \\ F \to \cdot d \end{array} $ $ \begin{array}{c c} F \to E + T \cdot \\ T \to T \cdot *F \end{array} $ |
| (3) E -> T | | $ F \rightarrow d \nearrow I_{7}$ |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; top = top -2; } | $ \begin{array}{c c} \hline d & (F) \\ \hline I_{f:} \\ F \to \cdot (E) \\ F \to \cdot d \end{array} $ $ \begin{array}{c c} \hline I_{g:} \\ \hline I_{g:} \end{array} $ |
| (5) T -> F | | I_{5} : d $\begin{vmatrix} E \rightarrow E + T \\ E \rightarrow T \end{vmatrix}$ E] (d $\begin{vmatrix} F \rightarrow (E \cdot) \\ F \rightarrow F \rightarrow T \end{vmatrix}$ |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; top = top -2; } | $ \begin{array}{c c} & T \to T^*F \\ & T \to F \\ & F \to \cdot (E) \end{array} $ |
| (7) F -> digit | | $F \rightarrow \cdot d$ $F \rightarrow (E) \cdot$ |

Input: 3 * 5 + 4

state
$$\rightarrow$$
 S₀ S₂ S₇ symbol \rightarrow \$ T * attribute \rightarrow - 3 -





| | | d |
|----------------------------|--|--|
| Productions | Semantic Actions | $\begin{bmatrix} I_{\theta} : & E' \to E \\ E' \to E \end{bmatrix} \xrightarrow{E} \begin{bmatrix} I_{1} : & + \\ E' \to E \end{bmatrix} \xrightarrow{I_{\theta} : E} \begin{bmatrix} I_{\theta} : & + \\ E \to E + \cdot T \end{bmatrix}$ |
| (1) L -> E | { print(stack[top].val); } | $ \begin{bmatrix} E \to E + T \\ E \to F + T \end{bmatrix} $ $ T \to T * F $ $ T \to T * F $ $ T \to F \to F \to T $ $ T \to F \to F \to F $ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; top = top -2; } | $ \begin{array}{c c} T \to \cdot T^*F \\ T \to \cdot F \end{array} $ $ \begin{array}{c c} F \to \cdot (E) \\ T \to T \cdot *F \end{array} $ $ \begin{array}{c c} T \to T \cdot *F \end{array} $ |
| (3) E -> T | | $ F \rightarrow d \nearrow I_{7}$ |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; top = top -2; } | $ \begin{array}{c c} \hline d & (F) \\ \hline I_{f:} \\ F \to \cdot (E) \\ F \to \cdot d \end{array} $ $ I_{g:}$ |
| (5) T -> F | | I_{5} : d $\begin{vmatrix} E \rightarrow E + I \\ E \rightarrow T \end{vmatrix}$ E I |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; top = top -2; } | $ \begin{array}{c c} & T \to T^*F \\ & T \to F \\ & F \to \cdot (E) \end{array} $ |
| (7) F -> digit | | $F \rightarrow \cdot \mathbf{d}$ $F \rightarrow (E) \cdot$ |

state
$$\rightarrow$$
 S₀ S₂ S₇ symbol \rightarrow \$ T * attribute \rightarrow - 3 -





| I | | d |
|----------------------------|---|---|
| Productions | Semantic Actions | $\begin{bmatrix} I_{\theta} : & E' \to E \\ E' \to \cdot E \end{bmatrix} \xrightarrow{E} \xrightarrow{I_{1} : E' \to E \cdot T} \xrightarrow{I_{\theta} : E \to E + \cdot T}$ |
| (1) L -> E | { print(stack[top].val); } | $\begin{bmatrix} E \to \cdot E + T \\ E \to \cdot T \end{bmatrix} T \xrightarrow{E \to E \cdot + T} \begin{bmatrix} F & T \to \cdot T^*F \\ T \to \cdot F \end{bmatrix} \xrightarrow{I_g:} T \xrightarrow{F} T \xrightarrow{F} T$ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; | $T \rightarrow T^*F$ $E \rightarrow T \cdot $ $*$ $F \rightarrow \cdot (E)$ $*$ $E \rightarrow E + T \cdot $ $T \rightarrow T \cdot *F$ |
| | top = top -2; } | $F \rightarrow F$ |
| (3) E -> T | | $F \rightarrow d$ |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; | $ \begin{array}{ c c c c c c c c c c c c c c c c c c c$ |
| | top = top -2; } | $F \rightarrow (\cdot E) \qquad F \rightarrow \cdot d \qquad I_g: \qquad \downarrow$ |
| (5) T -> F | | $\begin{array}{c ccccccccccccccccccccccccccccccccccc$ |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; | $ \begin{array}{c c} \hline F \rightarrow d \\ \hline T \rightarrow \cdot T^*F \\ \hline T \rightarrow \cdot F \end{array} $ |
| | top = top -2; } | $\begin{bmatrix} F \to \cdot (E) \\ F \to \cdot \mathbf{d} \end{bmatrix}$ $I_{II}:$ |
| (7) F -> digit | | $F \rightarrow c d$ |

state
$$\rightarrow$$
 S₀ S₂ S₇ S₅ symbol \rightarrow \$ T * d attribute \rightarrow - 3 - 5





| | | d |
|----------------------------|--|--|
| Productions | Semantic Actions | $\begin{bmatrix} I_{\theta} : & E' \to E \\ E' \to E \end{bmatrix} \xrightarrow{E} \begin{bmatrix} I_{1} : & + \\ E' \to E \end{bmatrix} \xrightarrow{I_{\theta} : E} \begin{bmatrix} I_{\theta} : & + \\ E \to E + \cdot T \end{bmatrix}$ |
| (1) L -> E | { print(stack[top].val); } | $ \begin{bmatrix} E \to E + T \\ E \to F + T \end{bmatrix} $ $ T \to T * F $ $ T \to T * F $ $ T \to F \to F \to T $ $ T \to F \to F \to F $ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; top = top -2; } | $ \begin{array}{c c} T \to \cdot T^*F \\ T \to \cdot F \end{array} $ $ \begin{array}{c c} F \to \cdot (E) \\ T \to T \cdot *F \end{array} $ $ \begin{array}{c c} T \to T \cdot *F \end{array} $ |
| (3) E -> T | | $ F \rightarrow d \nearrow I_{7}$ |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; top = top -2; } | $ \begin{array}{c c} \hline d & (F) \\ \hline I_{f:} \\ F \to \cdot (E) \\ F \to \cdot d \end{array} $ $ I_{g:}$ |
| (5) T -> F | | I_{5} : d $\begin{vmatrix} E \rightarrow E + I \\ E \rightarrow T \end{vmatrix}$ E I |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; top = top -2; } | $ \begin{array}{c c} & T \to T^*F \\ & T \to F \\ & F \to \cdot (E) \end{array} $ |
| (7) F -> digit | | $F \rightarrow \cdot \mathbf{d}$ $F \rightarrow (E) \cdot$ |

state
$$\rightarrow$$
 S₀ S₂ S₇ symbol \rightarrow \$ T * attribute \rightarrow - 3 -





| | | d |
|----------------------------|--|---|
| Productions | Semantic Actions | $\begin{vmatrix} I_{\theta} : & E \\ E' \to \cdot E \end{vmatrix} \xrightarrow{E} \begin{vmatrix} I_{1} : & + \\ E' \to E \cdot & E \end{vmatrix} \xrightarrow{I_{\theta}} \begin{vmatrix} I_{\theta} : & E \\ E \to E + \cdot T \end{vmatrix}$ |
| (1) L -> E | { print(stack[top].val); } | $\begin{bmatrix} E \to \cdot E + T \\ E \to \cdot T \end{bmatrix} T \xrightarrow{E \to E \cdot + T} \begin{bmatrix} F & T \to \cdot T * F \\ T \to \cdot F \end{bmatrix} \xrightarrow{I_0:} T \xrightarrow$ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; top = top -2; } | $ \begin{array}{c ccccccccccccccccccccccccccccccccccc$ |
| (3) E -> T | | $F \rightarrow \cdot (E)$ $F \rightarrow \cdot d$ $I_{3}:$ $T \rightarrow F$ $I_{10}:$ $T \rightarrow T^{*}F$ |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; top = top -2; } | $ \begin{array}{c c} \hline d & (F) \\ \hline I_{f}: \\ F \to \cdot (E) \\ F \to \cdot d \end{array} $ $ I_{g}: $ |
| (5) T -> F | | I_5 : d $E \to E \to I$ $E \to I$ $E \to I$ $E \to I$ |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; | $ \begin{array}{c c} F \to \mathbf{d} \\ \hline \end{array} $ $ \begin{array}{c c} T \to \cdot T^*F \\ T \to \cdot F \end{array} $ |
| | top = top -2; } | $\begin{bmatrix} F \to \cdot (E) \\ F \to \cdot \mathbf{d} \end{bmatrix}$ |
| (7) F -> digit | | $F \rightarrow c$ |

state
$$\rightarrow$$
 S₀ S₂ S₇ S₁₀ symbol \rightarrow \$ T * F attribute \rightarrow - 3 - 5





| | | d |
|----------------------------|---|---|
| Productions | Semantic Actions | $\begin{bmatrix} I_0: & & E \\ E' \to E \end{bmatrix} \xrightarrow{I_1:} \begin{bmatrix} E' \to E \\ E' \to E \end{bmatrix} \xrightarrow{I_2:} \begin{bmatrix} I_2: & & & & & & & & & & & & & & & & & & &$ |
| (1) L -> E | { print(stack[top].val); } | $ \begin{bmatrix} E \to \cdot E + T \\ E \to \cdot T \end{bmatrix} $ $ T \to T^*F $ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; | $ \begin{array}{ c c c c c c c c c c c c c c c c c c c$ |
| | top = top -2; } | |
| (3) E -> T | | $F \rightarrow d$ |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; | $ \begin{array}{c c} & & & \\ \hline \end{array} $ |
| | top = top -2; } | $F \rightarrow (\cdot E)$ $F \rightarrow \cdot d$ I_{S} : |
| (5) T -> F | | $ I_{\mathcal{S}}:$ $ \mathbf{d} _{E \to T} E = \mathbf{d} _{E \to F \to T} E $ |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; | $ \begin{array}{c c} F \to d \cdot \\ T \to F \end{array} $ $ \begin{array}{c c} T \to T^*F \\ T \to F \end{array} $ |
| | top = top -2; } | |
| (7) F -> digit | | $F \rightarrow \cdot \mathbf{d}$ $F \rightarrow (E) \cdot$ |





| | | F - d |
|----------------------------|---|--|
| Productions | Semantic Actions | $\begin{bmatrix} I_{\theta} : & & E \\ E' \to \cdot E & & E' \to E \cdot \end{bmatrix} \xrightarrow{I_{\theta} : E \to E + \cdot T} \xrightarrow{\mathbf{G}} \begin{bmatrix} I_{\theta} : & & & & & & & & & & & & & & & & & & $ |
| (1) L -> E | { print(stack[top].val); } | $ \begin{bmatrix} E \to \cdot E + T \\ E \to \cdot T \end{bmatrix} T $ $ \begin{bmatrix} E \to E \cdot + T \\ T \to \cdot T * F \end{bmatrix} $ $ T \to \cdot T * F $ $ T \to \cdot T *$ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; | $T \rightarrow T^*F$ $E \rightarrow T$ $*$ $T \rightarrow T^*F$ $*$ $T \rightarrow T^*F$ $*$ $T \rightarrow T^*F$ |
| (2) F . T | top = top -2; } | $F \rightarrow \cdot (E)$ F I_{10} : |
| (3) E -> T | | $T \rightarrow T \rightarrow$ |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; | $ \begin{array}{c c} \hline d & (& F & \hline \\ \hline & I_f & \hline \\ \hline \end{array} $ |
| | top = top -2; } | $F \rightarrow (\cdot E)$ $F \rightarrow \cdot d$ I_8 : |
| (5) T -> F | | I_5 : d $E \to E + I$ E] (d $F \to (E \cdot)$ |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; | $ \begin{array}{c c} F \to d \cdot \\ \hline T \to \cdot T^*F \\ T \to \cdot F \end{array} $ |
| | top = top -2; } | I_{tt} : |
| (7) F -> digit | | $F \rightarrow \cdot d$ $F \rightarrow (E) \cdot$ |





| | | F - d |
|----------------------------|---|--|
| Productions | Semantic Actions | $I_0: \atop E' \to \cdot E \qquad \stackrel{L}{\longleftarrow} I_1: \atop E' \to E \cdot \qquad \stackrel{+}{\longleftarrow} I_6: \atop E \to E + \cdot T \qquad \stackrel{U}{\longleftarrow}$ |
| (1) L -> E | { print(stack[top].val); } | $ \begin{bmatrix} E \to \cdot E + T \\ E \to \cdot T \end{bmatrix} $ $ \begin{bmatrix} E \to E \cdot + T \\ I_{2} \cdot \end{bmatrix} $ $ \begin{bmatrix} F \\ T \to \cdot T * F \end{bmatrix} $ $ \begin{bmatrix} T \to \cdot T * F \end{bmatrix} $ $ T \to F $ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; | $ \begin{array}{c c} E \to T \\ T \to T^*F \\ T \to F \end{array} $ $ \begin{array}{c c} \downarrow \\ F \to CE \\ F \to CE \end{array} $ $ \begin{array}{c c} \downarrow \\ T \to T^*F \\ T \to T^*F \end{array} $ |
| (2) F . T | top = top -2; } | $F \rightarrow \cdot (E)$ F I_{10} : |
| (3) E -> T | | $T \rightarrow T$ |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; | I |
| | top = top -2; } | $F \rightarrow (\cdot E)$ $F \rightarrow \cdot d$ I_{g} : |
| (5) T -> F | | $\begin{bmatrix} I_5: & \mathbf{d} & E \to T & E \end{bmatrix} \begin{pmatrix} \mathbf{d} & F \to (E \cdot) \\ E \to F \cdot + T & E \end{bmatrix}$ |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; | $ \begin{array}{c c} F \to d \cdot \\ \hline T \to \cdot T^*F \\ \hline T \to \cdot F \end{array} $ |
| | top = top -2; } | I_{II} : |
| (7) F -> digit | | $F \to \cdot d$ $F \to (E) \cdot$ |





| | | d |
|----------------------------|---|---|
| Productions | Semantic Actions | $\begin{bmatrix} I_{\theta} : & E \\ E' \to \cdot E \end{bmatrix} \xrightarrow{E} \xrightarrow{I_{1} : E' \to E} \cdot \begin{bmatrix} + & I_{\theta} : & E \\ E \to E + \cdot T \end{bmatrix}$ |
| (1) L -> E | { print(stack[top].val); } | $\begin{bmatrix} E \to \cdot E + T \\ E \to \cdot T \end{bmatrix} T \xrightarrow{E \to E \cdot + T} \begin{bmatrix} F & T \to \cdot T * F \\ T \to \cdot F \end{bmatrix} \xrightarrow{I_0:} T \xrightarrow{F} T \xrightarrow$ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; | $ \begin{array}{c ccccccccccccccccccccccccccccccccccc$ |
| (2) F > T | top = top -2; } | $F \rightarrow \cdot (E)$ F $I_{2:}$ $I_{10:}$ |
| (3) E -> T | | $ F \rightarrow \cdot d $ $T \rightarrow F$. $ F $ $ F $ |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; | $ \begin{array}{c c} & & & \\ \hline & & \\ \hline & & & \\ \hline \\ \hline$ |
| | top = top -2; } | $F \rightarrow (\cdot E)$ $F \rightarrow \cdot d$ I_8 : |
| (5) T -> F | | $\begin{array}{c ccccccccccccccccccccccccccccccccccc$ |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; | $ \begin{array}{c c} F \to d \cdot \\ \hline T \to \cdot T^*F \\ T \to \cdot F \end{array} $ |
| | top = top -2; } | |
| (7) F -> digit | | $F \rightarrow \cdot d$ $F \rightarrow (E) \cdot$ |





| | | d |
|----------------------------|---|---|
| Productions | Semantic Actions | $\begin{bmatrix} I_{\theta} : & E \\ E' \to \cdot E \end{bmatrix} \xrightarrow{E} \xrightarrow{I_{1} : E' \to E} \cdot \begin{bmatrix} + & I_{\theta} : & E \\ E \to E + \cdot T \end{bmatrix}$ |
| (1) L -> E | { print(stack[top].val); } | $\begin{bmatrix} E \to \cdot E + T \\ E \to \cdot T \end{bmatrix} T \xrightarrow{E \to E \cdot + T} \begin{bmatrix} F & T \to \cdot T * F \\ T \to \cdot F \end{bmatrix} \xrightarrow{I_0:} T \xrightarrow{F} T \xrightarrow$ |
| (2) E -> E ₁ +T | { stack[top-2].val = stack[top-2].val + stack[top].val; | $ \begin{array}{c ccccccccccccccccccccccccccccccccccc$ |
| (2) F > T | top = top -2; } | $F \rightarrow \cdot (E)$ F $I_{2:}$ $I_{10:}$ |
| (3) E -> T | | $ F \rightarrow \cdot d $ $T \rightarrow F$. $ F $ $ F $ |
| (4) T -> T ₁ *F | { stack[top-2].val = stack[top-2].val x stack[top].val; | $ \begin{array}{c c} & & & \\ \hline & & \\ \hline & & & \\ \hline \\ \hline$ |
| | top = top -2; } | $F \rightarrow (\cdot E)$ $F \rightarrow \cdot d$ I_8 : |
| (5) T -> F | | $\begin{array}{c ccccccccccccccccccccccccccccccccccc$ |
| (6) F -> (E) | { stack[top-2].val = stack[top-1].val; | $ \begin{array}{c c} F \to d \cdot \\ \hline T \to \cdot T^*F \\ T \to \cdot F \end{array} $ |
| | top = top -2; } | |
| (7) F -> digit | | $F \rightarrow \cdot d$ $F \rightarrow (E) \cdot$ |



