中山大學 SUN YAT-SEN UNIVERSITY 计算机学院（软件学院） SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Compilation Principle
编 译 原 理

第16讲：语义分析(2)

张献伟

xianweiz.github.io

DCS290, 4/18/2023

Quiz Questions

- Q1: for the grammar G, augment and give the initial and final items.

  Add rule-0: S’ -> S. Initial item: S’ -> ·S, final item: S’ -> S·

- Q2: to parse with LR(0), get the first state (i.e., S_0/I_0).

  Closure({S’ -> ·S}) = {S’ -> ·S, S-> ·AB, A-> ·cAa, A-> ·d}

- Q3: give the state of goto(S_0, c)?

  Closure({A -> c·Aa}) = {A -> c·Aa, A -> ·cAa, A -> ·d}

S → AB
A → cAa | d
B → b

- Q4: LR(0), SLR(1), LR(1), LALR(1), what are the differences.

  LR(0): no lookahead, always reduce on complete state

  SLR(1): one lookahead, reduce using FOLLOW

  LR(1): one lookahead, reduce using specified terminals

  LALR(1): a compromise of LR(1) and LR(0)/SLR(1)

- Q5: how to enhance CFG for semantic analysis?

  Add semantic attributes for symbols, rules/actions for productions.

Example: Synthesized Attribute (cont.)

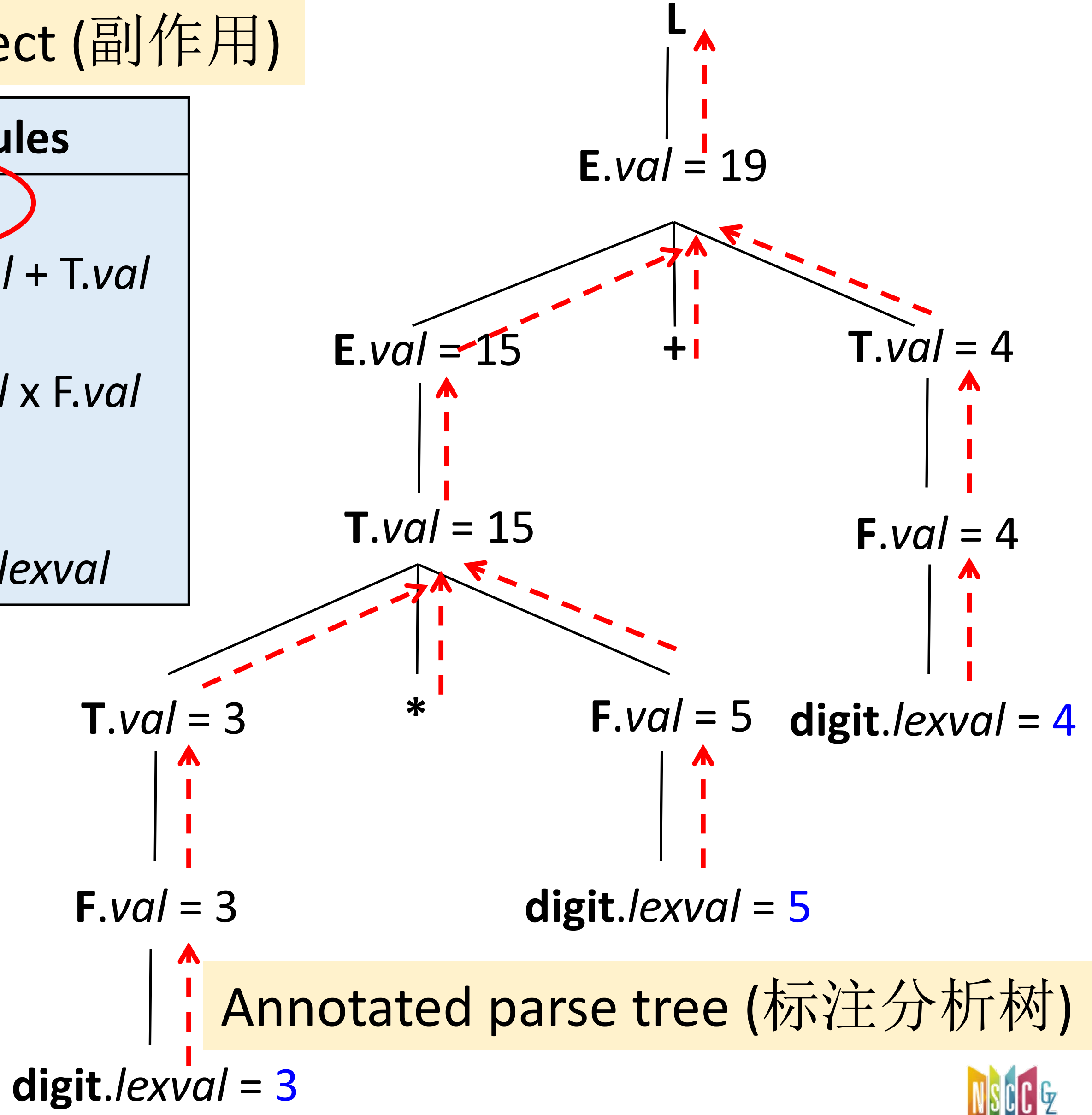
SDD:

Side effect (副作用)

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*

Input:

3 * 5 + 4



Annotated parse tree (标注分析树)

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

Example: Inherited Attribute[继承]

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.*entry*, L.*inh*)

T has synthesized attribute *type*

L has inherited attribute *inh*

Pointing to a symbol-table[符号表] object

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)

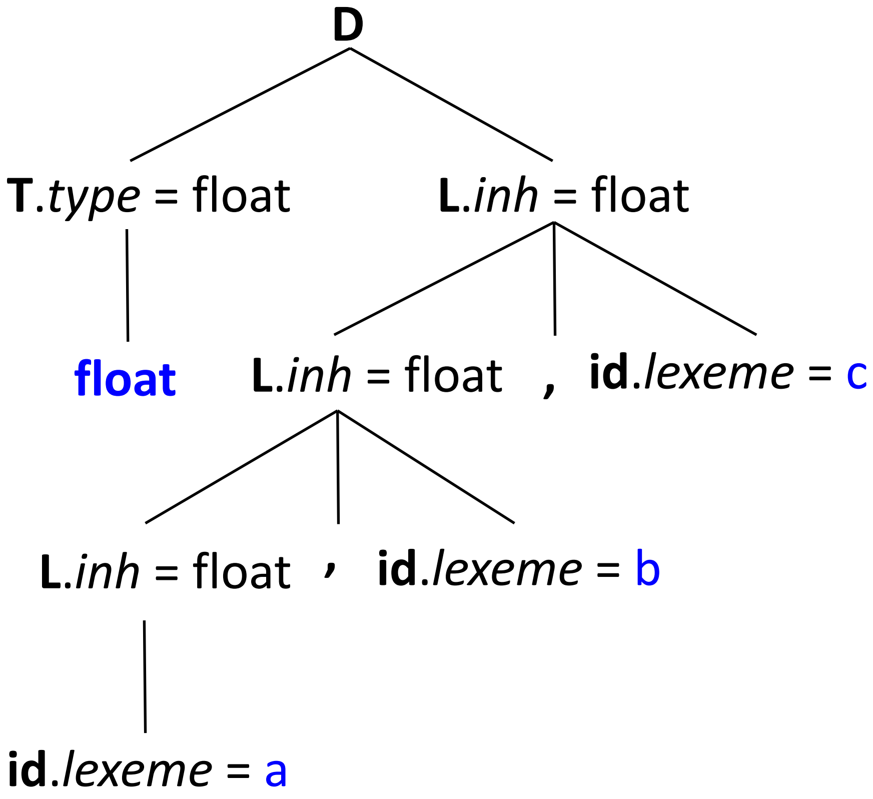
(4) Pass down type, and add type to symbol table entry for the identifier

(5) Add type to symbol table

Example: Inherited Attribute (cont.)

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.*entry*, L.*inh*)



D

T.*type* = float

L.*inh* = float

float

L.*inh* = float

,

id.*lexeme* = c

L.*inh* = float

,

id.*lexeme* = b

id.*lexeme* = a

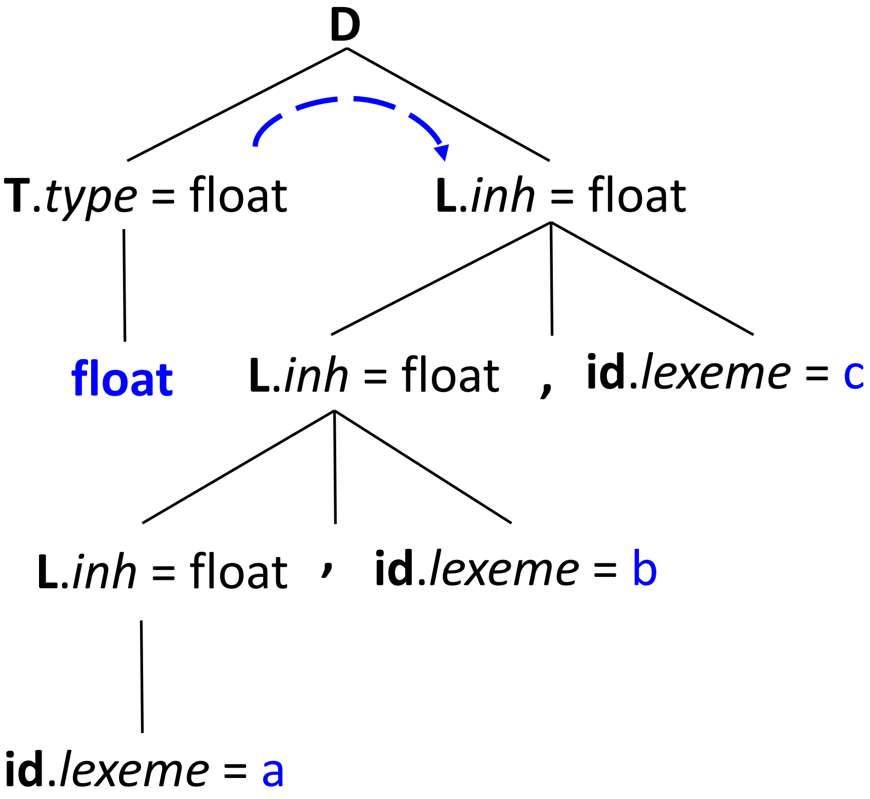
Input:

float a, b, c

Example: Inherited Attribute (cont.)

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.*entry*, L.*inh*)



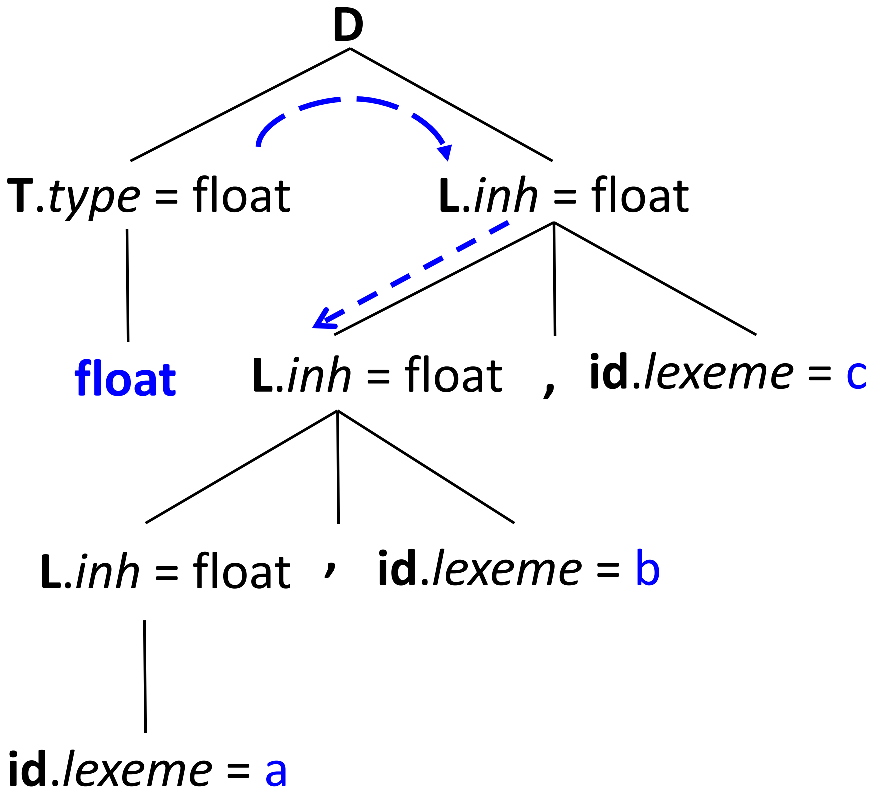
Input:

float a, b, c

Example: Inherited Attribute (cont.)

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.*entry*, L.*inh*)



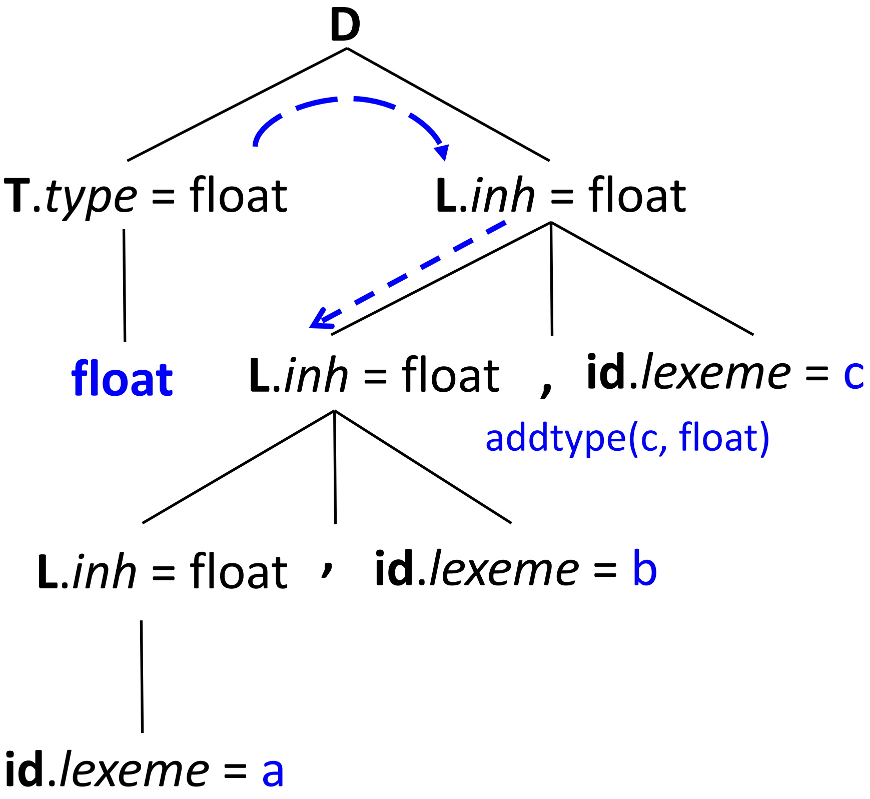
Input:

float a, b, c

Example: Inherited Attribute (cont.)

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.*entry*, L.*inh*)



Input:

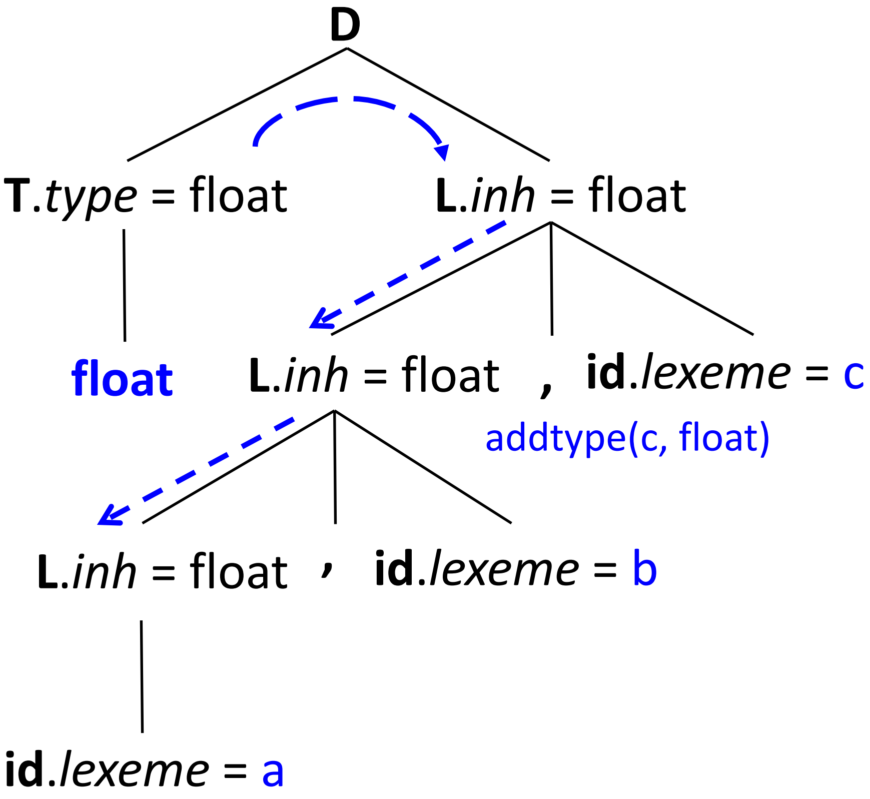
float a, b, c

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

Example: Inherited Attribute (cont.)

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.*entry*, L.*inh*)



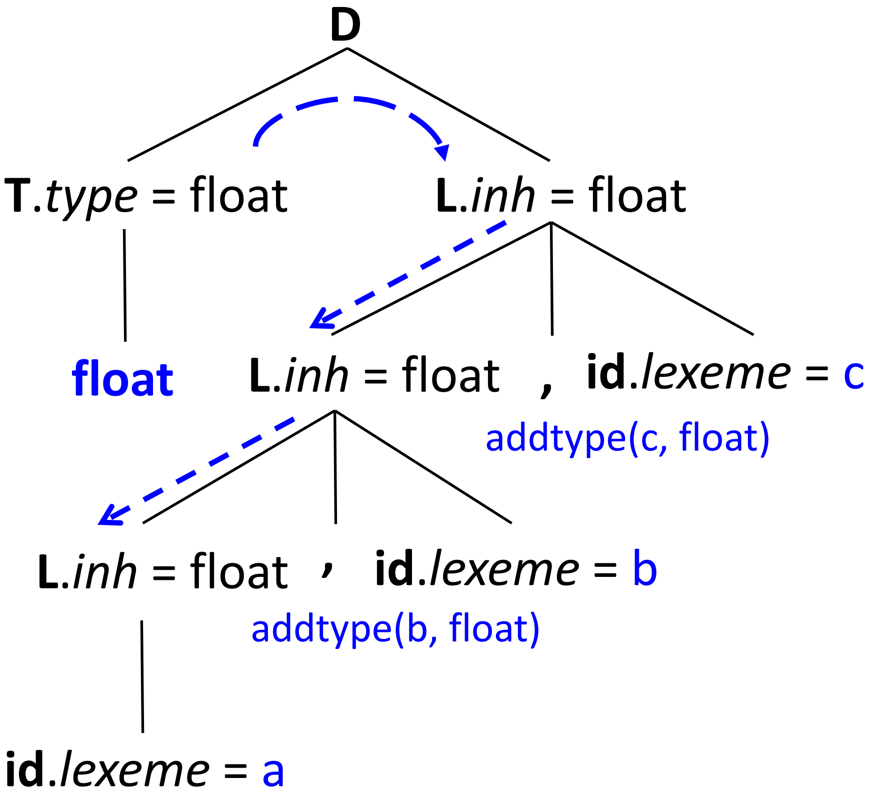
Input:

float a, b, c

Example: Inherited Attribute (cont.)

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.*entry*, L.*inh*)



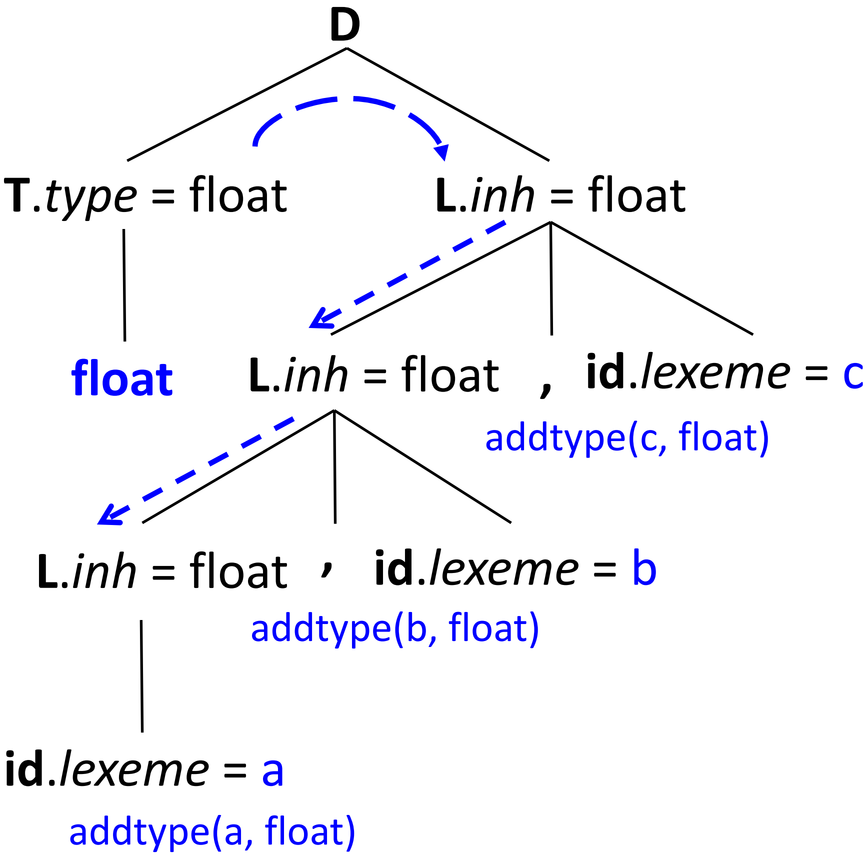
Input:

float a, b, c

Example: Inherited Attribute (cont.)

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.*entry*, L.*inh*)



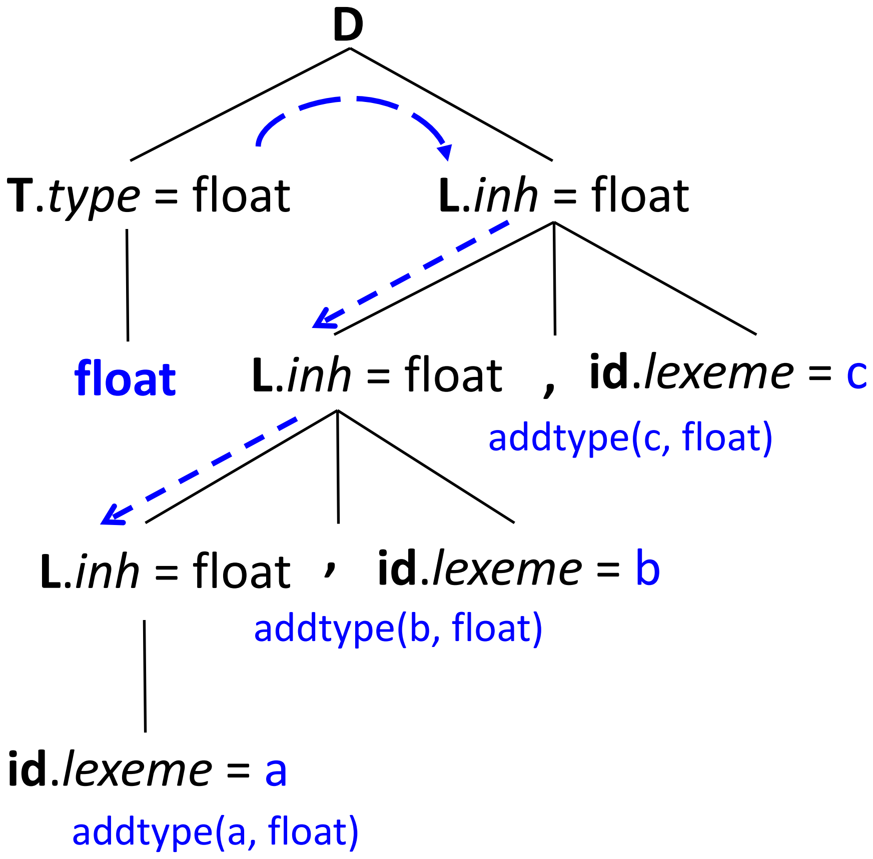
Input:

float a, b, c

Example: Inherited Attribute (cont.)

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.*entry*, L.*inh*)



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]

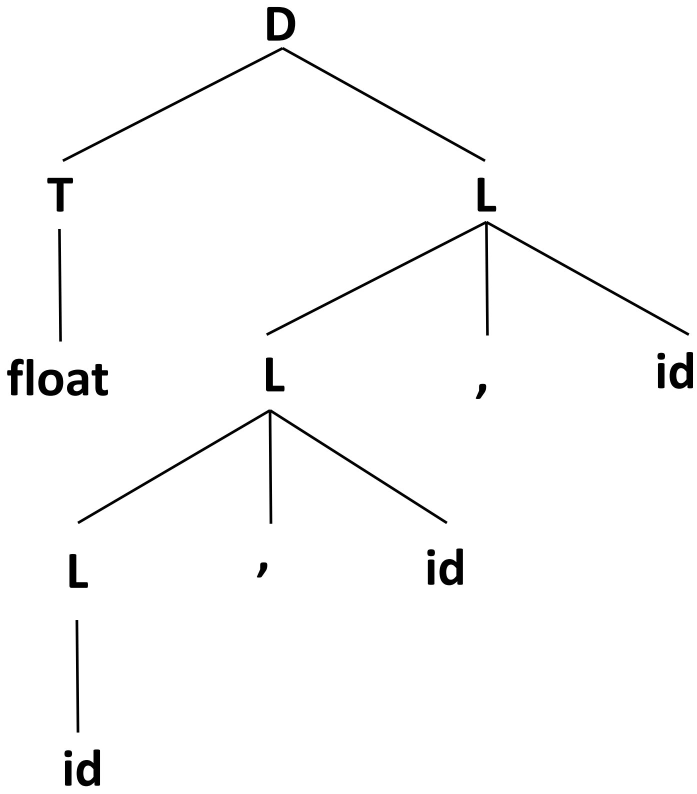
Dependence Graph[依赖图]

- 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.*entry*, L.*inh*)
(5) L -> id	addtype(id.*entry*, L.*inh*)



Input:

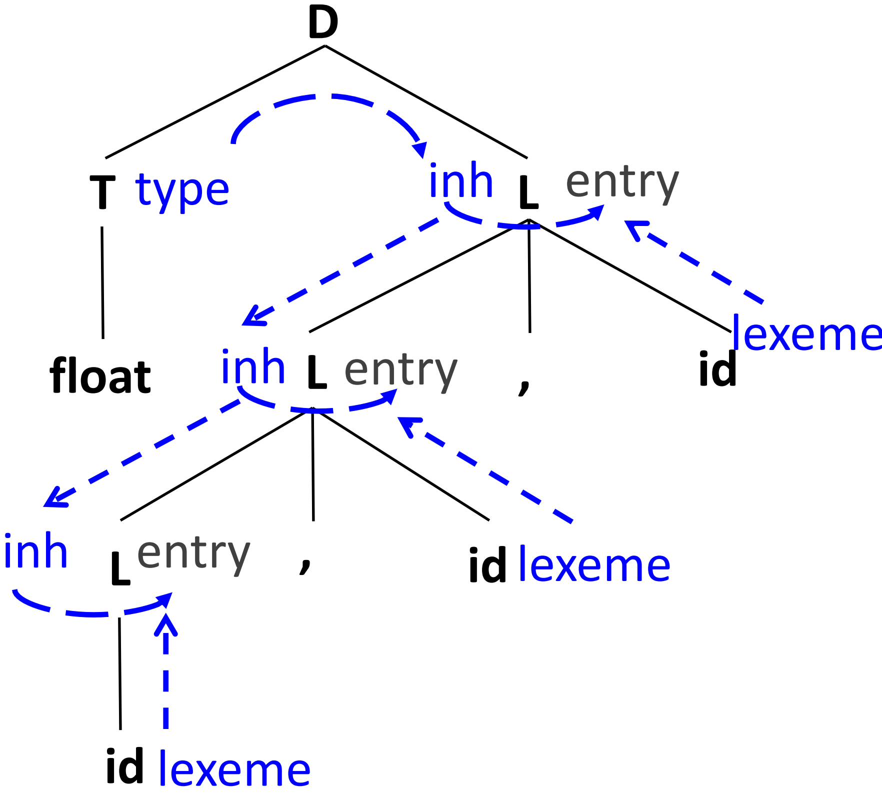
float a, b, c

'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.*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.*entry*, L.*inh*)



Input:

float a, b, c

‘entry’ is dummy attribute for the *addtype()*

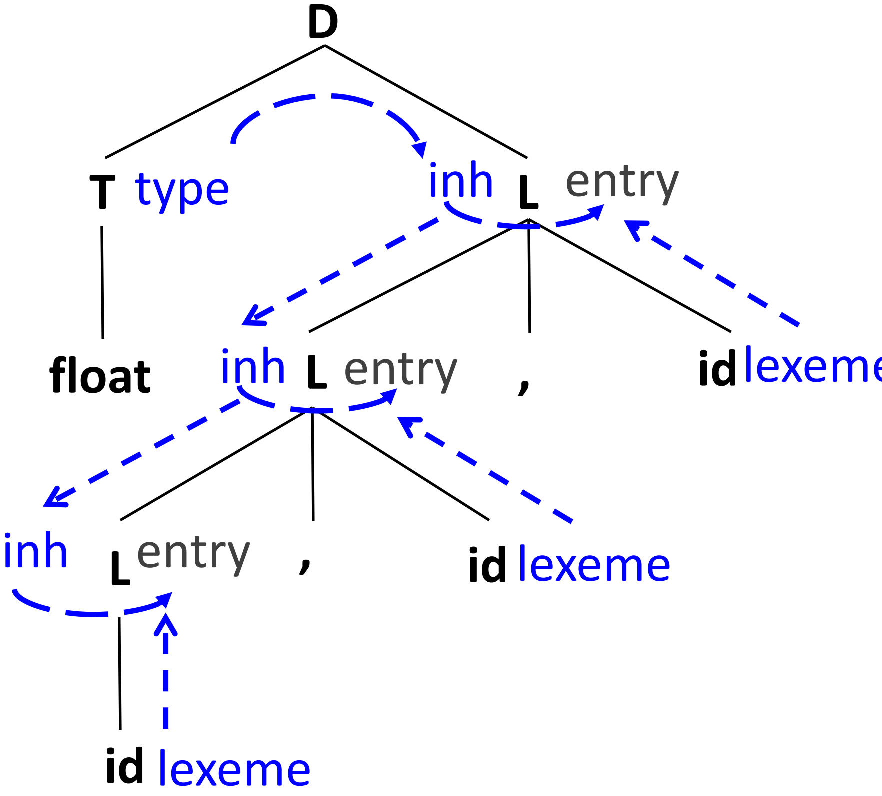
Evaluation Order[属性值计算顺序]

- 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

Example: Evaluation Order

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.*lexeme*, L.*inh*)
(5) L -> id	addtype(id.*lexeme*, L.*inh*)



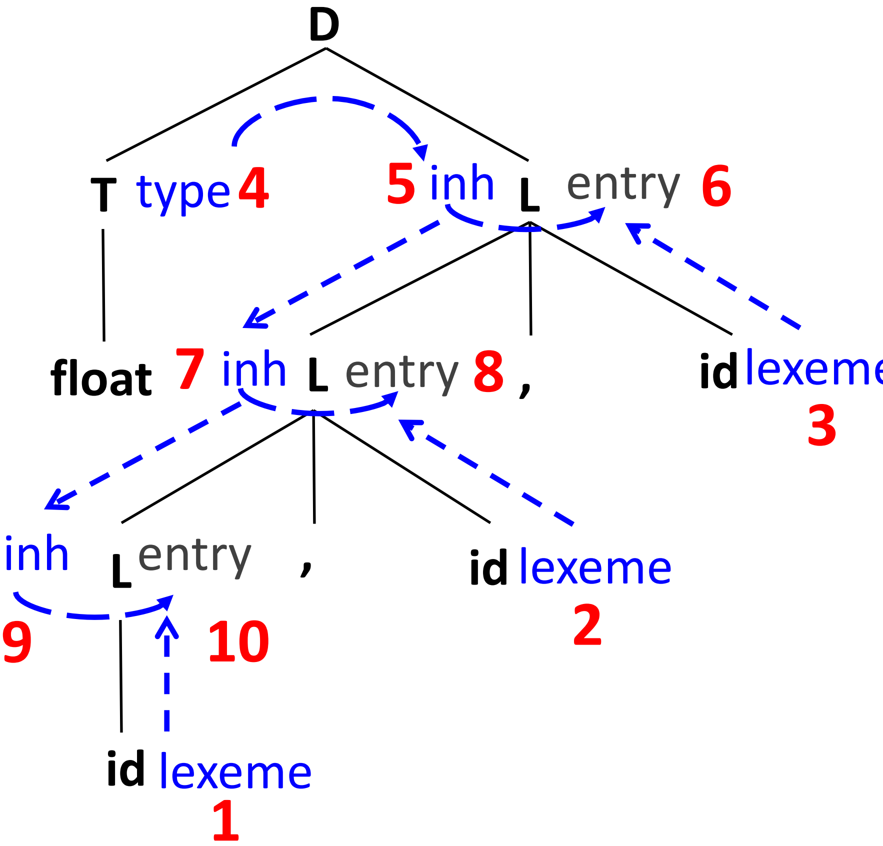
Input:

float a, b, c

Example: Evaluation Order

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.*lexeme*, L.*inh*)
(5) L -> id	addtype(id.*lexeme*, L.*inh*)



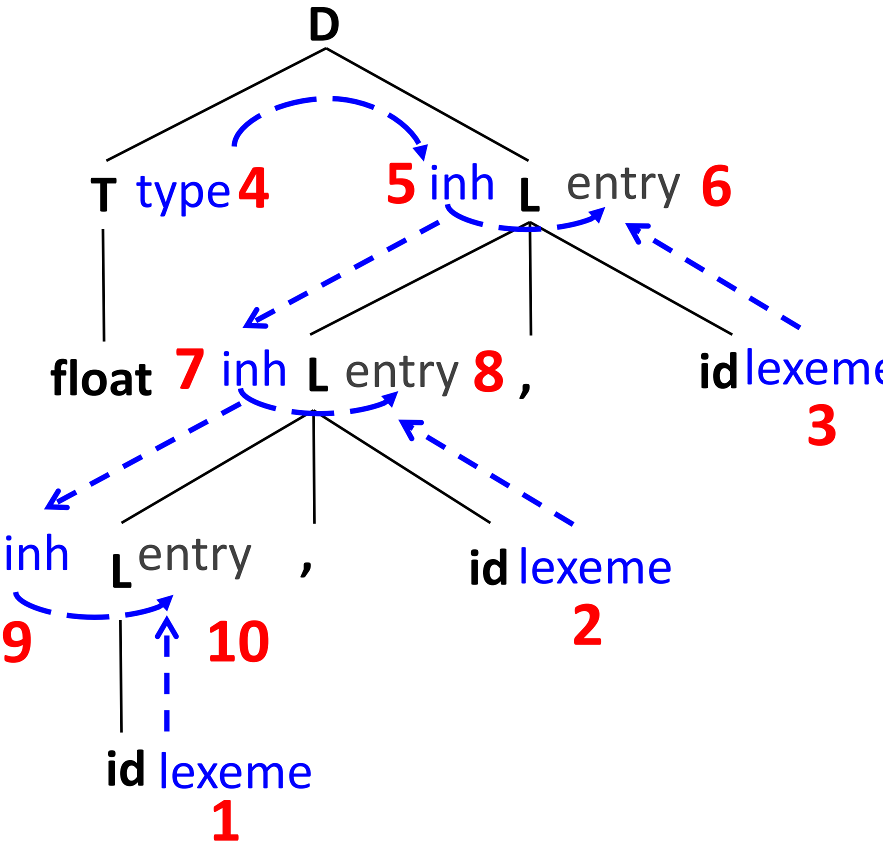
Input:

float a, b, c

Example: Evaluation Order

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.*lexeme*, L.*inh*)
(5) L -> id	addtype(id.*lexeme*, L.*inh*)



Input:

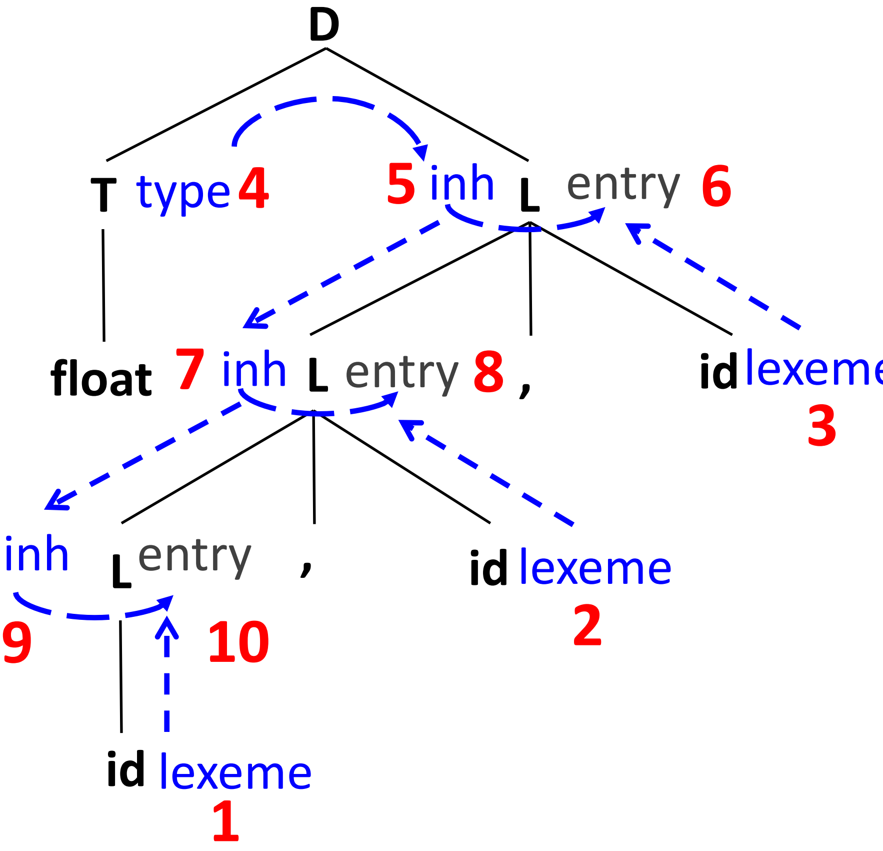
float a, b, c

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

Example: Evaluation Order

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.*lexeme*, L.*inh*)
(5) L -> id	addtype(id.*lexeme*, L.*inh*)



Input:

float a, b, c

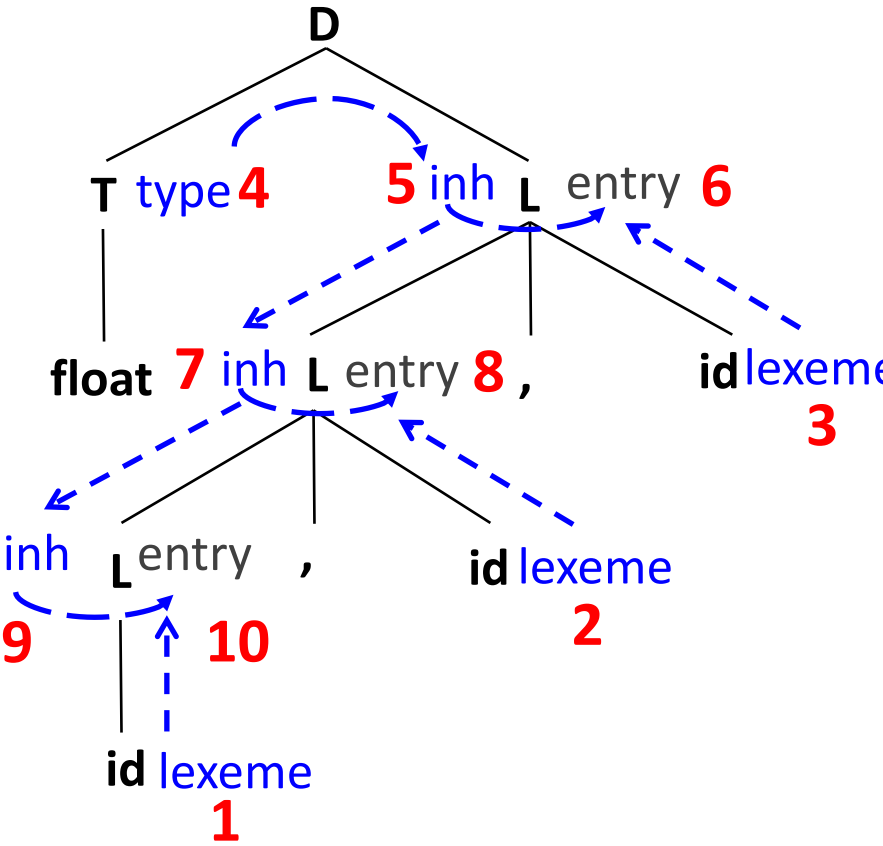
Topological sort:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Example: Evaluation Order

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.*lexeme*, L.*inh*)
(5) L -> id	addtype(id.*lexeme*, L.*inh*)



Input:

float a, b, c

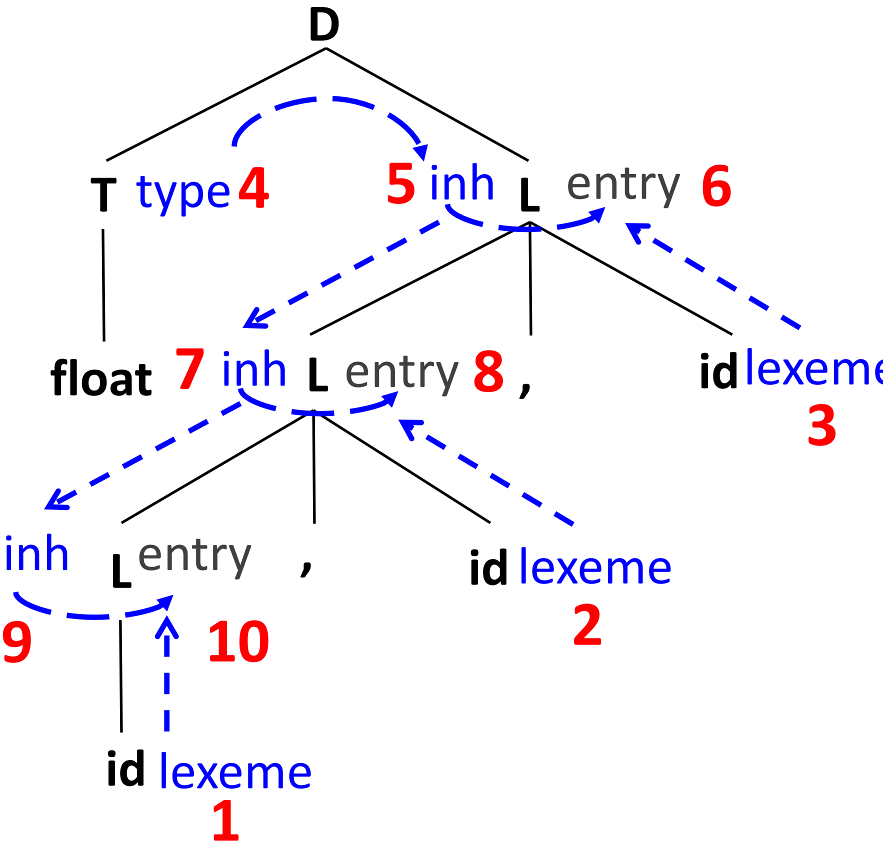
Topological sort:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Example: Evaluation Order

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.*lexeme*, L.*inh*)
(5) L -> id	addtype(id.*lexeme*, L.*inh*)



Input:

float a, b, c

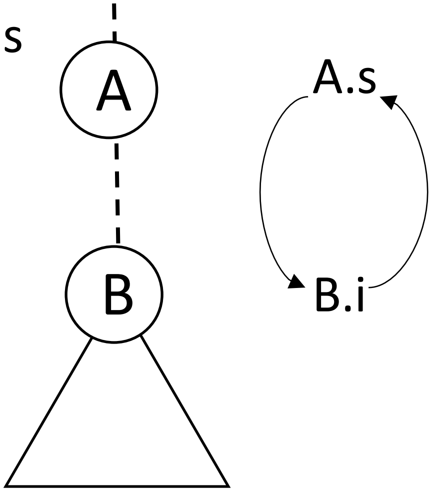
Topological sort:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

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[一些SDD确保无循环]
    - Such classes do not permit graphs with cycles

Production	Semantic Rules
A -> B	A.*s* = B.*i*;
	B.*i* = A.*s* + 1;



S-Attributed Definitions[S-属性定义]

- An SDD is **S-attributed** if every attribute is synthesized[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.*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*

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 -> $X_1X_2\ldots X_n$, the inherited attribute $X_i.a$ only depends on
    - **Inherited** attributes associated with A
    - 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

Why not synthesized?

Cycle: X_i depends on A, A.*s* depends on X_i

- Can be implemented during top-down parsing[LL分析中]

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 -> $X_1X_2…X_n$, the inherited attribute $X_i.a$ only depends on
    - □ **Inherited** attributes associated with A
    - □ 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

Why not synthesized?

Cycle: X_i depends on A, A.*s* depends on 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*)

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 -> $X_1X_2 \dots X_n$, the inherited attribute $X_i.a$ only depends on
    - □ **Inherited** attributes associated with A
    - □ 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

Why not synthesized?

Cycle: X_i depends on A, A.*s* depends on 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 -> $X_1X_2…X_n$, the inherited attribute $X_i.a$ only depends on
    - □ **Inherited** attributes associated with A
    - □ 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

Why not synthesized?

Cycle: X_i depends on A, A.*s* depends on 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*)

Not L-SDD: A.*s* is syn attr

Not S-SDD: B.*i* is inh

Not L-SDD: C is right to B