COMPILER CONSTRUCTION
(Evaluation Orders for SDD's)

Prof. K R Chowdhary
Email: kr.chowdhary@jietjodhpur.ac.in

Campus Director, JIET, Jodhpur

Thursday 18th October, 2018

Dependency graphs

"Dependency graphs" are a useful tool for determining an evaluation order for the attribute instances in a given parse tree. A *dependency graph* helps us determine how attributes values can be computed. Two important classes of SDDs are: "S-attributed" and the more general "L-attributed" SDDs.

An edge from one attribute instance to another means that the value of the first is needed to compute the second. Edges express constraints implied by the semantic rules.

Dependency graphs

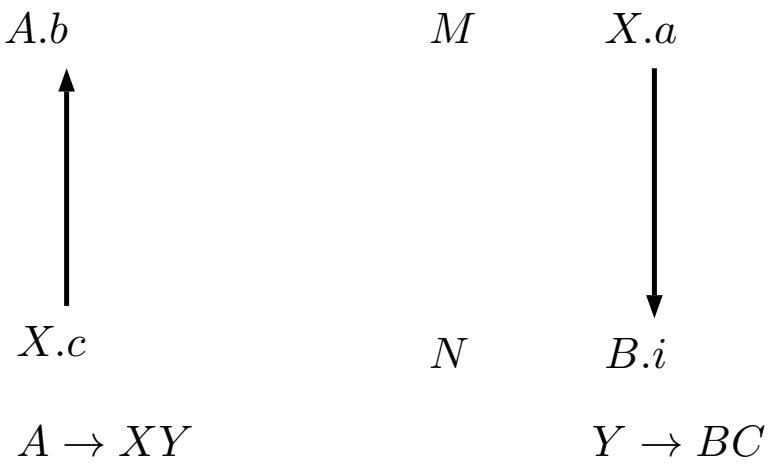

Figure 1: Sample Dependency graphs.

Annotated Parse tree and Dependency graph

Example

Synthesize $E.val$ from $E_1.val$ and $E_2.val$.

Solution. Consider the following production and rule:

$$Production\ Rule: \quad E \rightarrow E_1 + T$$

$$Semantic\ Rule: \quad E.val = E_1.val + T.val$$

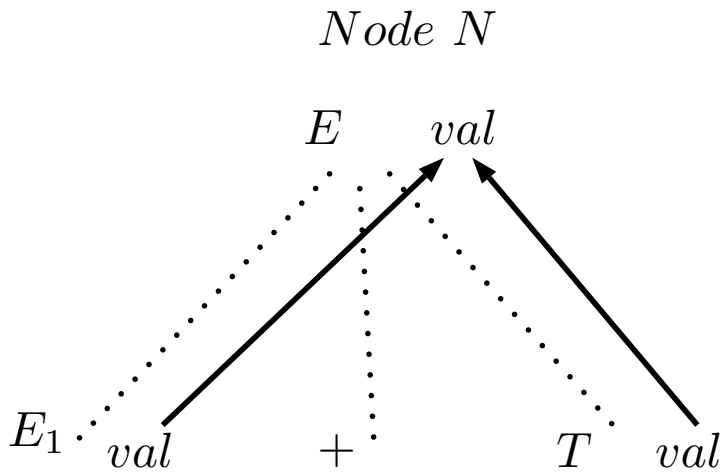

Figure 2: $E.val$ is synthesized from $E_1.val$ and $E_2.val$.

SDD for "a * b"

Example

Construct a Dependency graph for the annotated parse tree of Fig. 3, whose SDD is given in Table 1.

Table 1: An SDD based on a grammar suitable for top-down parsing.

S.No.	Production	Semantic Rules
1.	$T \rightarrow FT'$	$T'.inh = F.val$
		$T.val = T'.syn$
2.	$T' \rightarrow *FT'_1$	$T'_1.inh = T'_1.inh \times F.val$
		$T'.syn = T'_1.syn$
3.	$T' \rightarrow \varepsilon$	$T'.syn = T'.inh$
4.	$F \rightarrow \textbf{digit}$	$F.val = \textbf{digit}.lexval$

Annotated Parse tree

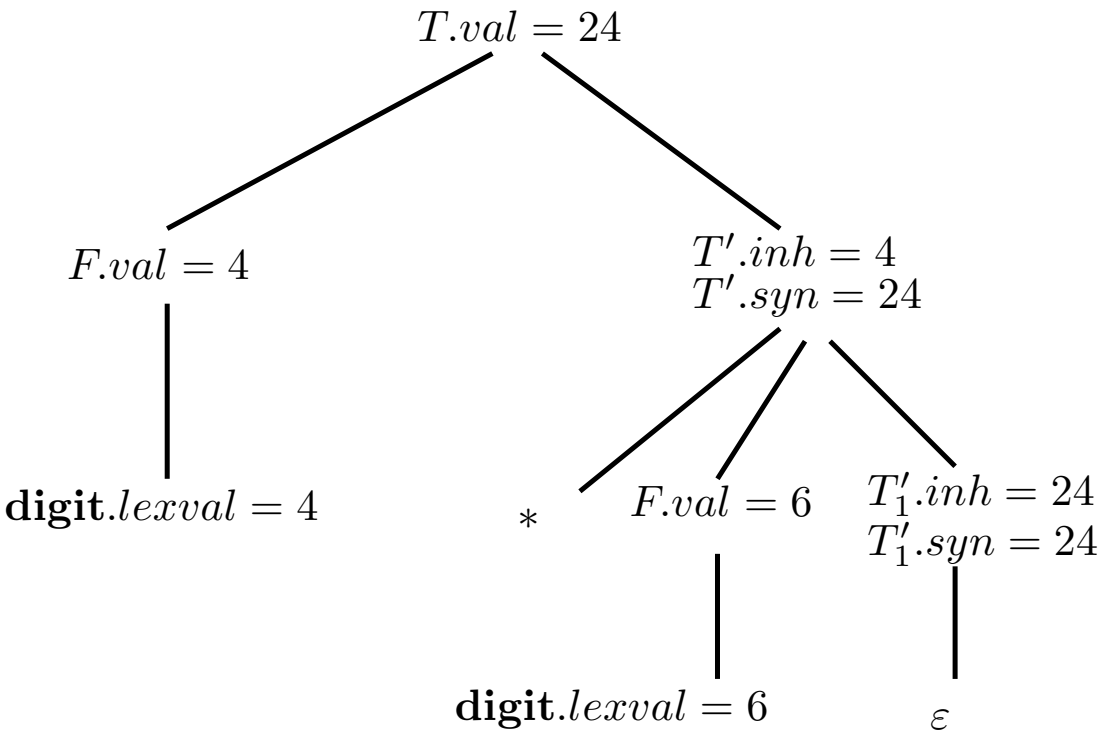

Figure 3: Annotated parse tree for $4 * 6$.

Annotated parse tree and its dependency graph

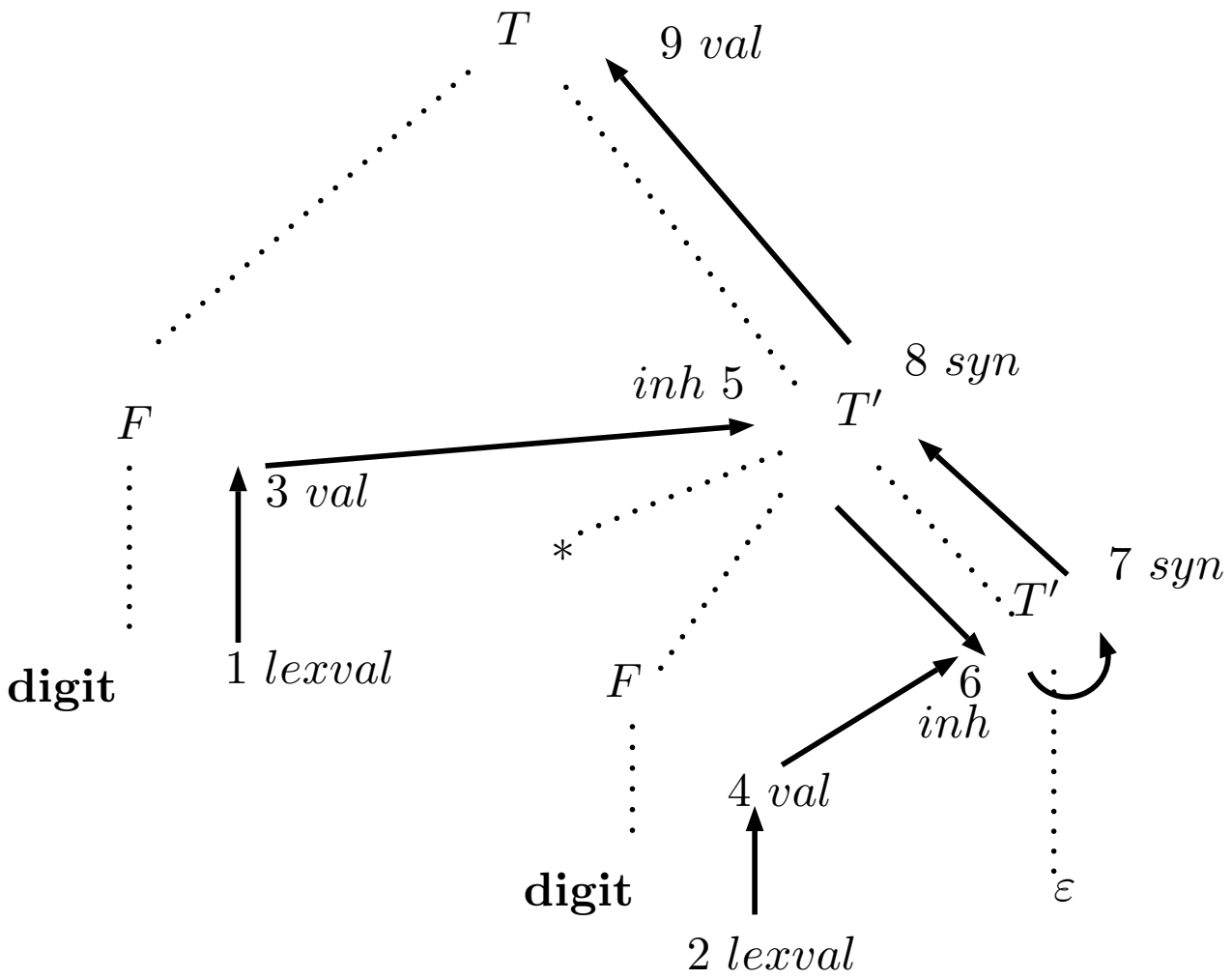

Figure 4: Dependency graph (→) and annotated parse (...) tree of Fig. 3.

Ordering the evaluation of Attributes

- If the dependency graph has an edge from node M to node N, then the attribute corresponding to M must be evaluated before the attribute of N.
- Sequences of nodes N_1, N_2, ..., N_k such that if there is an edge of the dependency graph from N_i to N_j; where $i < j$. Such an ordering transforms a directed graph into a linear order, called *topological sort* of the graph.
- If there is any cycle in the graph, then there are no topological sorts possible. Since there are no cycles, we can surely find a node with no edge entering. For if there were no such node, we could proceed from predecessor to predecessor until we came back to some node we had already seen, yielding a cycle.

Topological Sort

Example

Topological Sort.

- The dependency graph of Fig. 4 has no cycles. One topological sort is the order in which the nodes have already been numbered: 1,2,. . . ,9.
- Every edge of the graph goes from a node to a higher-numbered node, so this order is surely a topological sort. The other topological sort is 1,3,5,2,4,6,7,8,9.

S-Attributed Definitions

Translations can be implemented using classes of SDD's that guarantee an evaluation order, since they do not permit dependency graphs with cycles.

Definition

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

Example

The SDD of Table. 2 is an example of an S-attributed definition. Each attribute, $L.val$, $E.va1$, $T.val$, and $F.val$ is synthesized.

(See the table in next slide.)

S-Attributed Definitions

Table 2: Syntax-directed definition of a simple desk calculator (S-attributed).

	Production	Semantic Rules
1.	$L \to E$ **n**	$L.val = E.val$
2.	$E \to E_1 + T$	$E.val = E_1.val + T.val$
3.	$E \to T$	$E.val = T.val$
4.	$T \to T_1 * F$	$T.val = T_1.val \times F.val$
5.	$T \to F$	$T.val = F.val$
6.	$F \to (E)$	$F.val = E.val$
7.	$F \to$ **digit**	$F.val =$ **digit**$.lexval$

When an SDD is S-attributed, we can evaluate its attributes in any bottom-up order of the nodes of the parse tree. □

L-Attributed Definitions

For L-attributed definitions, edges of dependency graph corresponding to the attributes can go from left to right in productions. An attribute must be either

1. Synthesized, or
2. Inherited, but with the rules limited as follows. let $A \rightarrow X_1X_2...X_n$, and an inherited attribute $X_i.a$ is computed by a rule associated with this production. Then the rule may use only:
    1. Inherited attributes associated with the head A.
    2. Either inherited or synthesized attributes associated with the occurrences of symbols X_1, X_2, ... , X_{i-1} occurring to the left of X_i.
    3. Inherited or synthesized attributes associated with of X_i itself, such that there are no cycles in a dependency graph of this X_i.

L-Attributed Definitions

Theorem

Show that the SDD given in Table 1 is L-attributed.

Proof.

1. The SDD in Table 1 is L-attributed. To see why, consider the semantic rules for inherited attributes: 1 and 2.
2. The first of these rules defines the inherited attribute $T'.inh$ using only $F.val$, and F appears to the left of T' in the production body, as required. The second rule defines $T'_1.inh$ using the inherited attribute $T'.inh$ associated with the head, and $F.va1$, where F appears to the left of T'_1 in the production body.
3. In each of these cases, the rules use information "from above or from the left ," as required by the class. The remaining attributes are synthesized. Hence, the SDD is L-attributed. □

L-Attributed Definitions

Example

Any SDD containing the following production and rules **cannot** be L-attributed:

$Production : A \to BC$
$Semantic\ Rules : A.s = B.b; B.i = f(C.c, A.s)$
Solution. The first rule, $A.s = B.b$, is a legitimate rule in either an S-attributed or L-attributed SDD. It defines a synthesized attribute $A.s$ in terms of an attribute b at child B (that is, a symbol within the production body).
The second rule defines an inherited attribute $B.i$, so the entire SDD cannot be S-attributed. Further, although the rule is legal, the SDD cannot be L-attributed, because the attribute $C.c$ is used to help define $B.i$, and C is to the right of B in the production body. □