7CS5A COMPILER CONSTRUCTION (Evaluation Orders for SDD's) Fall 2018

Lecture 5: Oct. 11, 2018

Instructor: K.R. Chowdhary *: Professor of CS*

5.1 Evaluation Orders for SDD's

"Dependency graphs" are a useful tool for determining an evaluation order for the attribute instances in a given parse tree. While an *annotated parse tree* shows the values of attributes, a *dependency graph* helps us determine how those values can be computed. In the following, in addition to dependency graph, we define two important classes of SDD's: the "S-attributed" and the more general "L-attributed" SDDs. The translations specified by these two classes fit well with the parsing methods we have studied, and most translations encountered in practice can be written as per the requirements of at least one of these classes.

5.1.1 Dependency Graphs

A dependency graph indicate the flow of information among the instances of attributes in a given parse tree. 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. Following is more detailed introduction:

- For each parse-tree node, say a node labeled by grammar symbol X, the dependency graph has a node for each attribute associated with X.

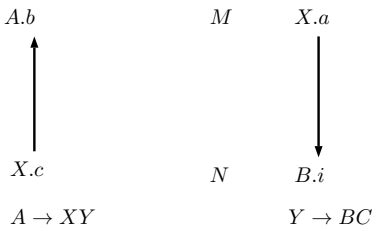

Figure 5.1: Sample Dependency graphs.

- Suppose that a semantic rule associated with a production p defines the value of synthesized attribute $A.b$ in terms of the value of $X.c$ (the rule may define $A.b$ in terms of other attributes in addition to $X.c$) . Then, the dependency graph has an edge from $X.c$ to $A.b$. More precisely, at every node N labeled as A where production

p is applied, create an edge to attribute b at N, from the attribute c at the child of N corresponding to this instance of the symbol X in the body of the production[1].

- Suppose that a semantic rule associated with a production p defines the value of inherited attribute $B.i$ in terms of the value of $X.a$. Then, the dependency graph has an edge from $X.a$ to $B.i$. For each node N labeled B that corresponds to an occurrence of this B in the body of production p, create an edge to attribute i at N from the attribute a at the node M that corresponds to this occurrence of X. Note that M could be either the parent or a sibling of N.

Example 5.1 *Synthesize $E.val$ from $E_1.val$ and $E_2.val$.*

Solution. Consider the following production and rule:

$$\begin{aligned} &\textit{Production Rule}: \\ &E \to E_1 + T \end{aligned}$$

$$\begin{aligned} &\textit{Semantic Rule}: \\ &E.val = E_1.val + T.val \end{aligned}$$

At every node N labeled E, with children corresponding to the body of this production, the synthesized attribute val at N is computed using the values of val at the two children, labeled E and T. Thus, a portion of the dependency graph for every parse tree in which this production is used looks like in Fig. 5.2. As a convention, we shall show the parse tree edges as dotted lines, while the edges of the dependency graph are solid.

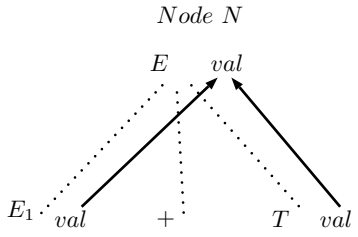

Figure 5.2: $E.val$ is synthesized from $E_1.val$ and $E_2.val$.

□

Example 5.2 *Construct a Dependency graph for the annotated parse tree of Fig. 5.3, whose SDD is given in Table 5.1.*

Solution. The nodes of the dependency graph, represented by the numbers 1 through 9, in Figure 5.4 correspond to the attributes in the annotated parse tree in Fig. 5.3.

[1]Since a node N can have several children labeled X, we again assume that subscripts distinguish among uses of the same symbol at different places in the production.

Table 5.1: An SDD based on a grammar suitable for top-down parsing.

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

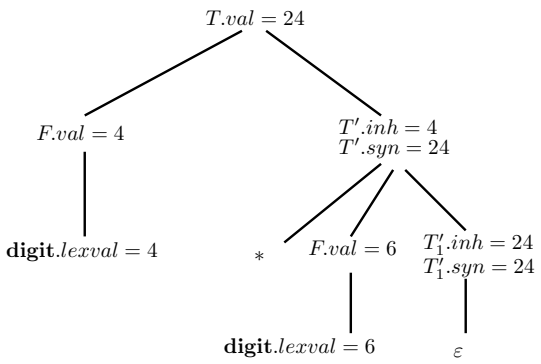

Figure 5.3: Annotated parse tree for $4 * 6$.

Nodes 1 and 2 represent the attribute lexval associated with the two leaves labeled **digit**. Nodes 3 and 4 represent the attribute *val* (i.e., 4 and 6) associated with the two nodes labeled F. The edges to node 3 from 1 and to node 4 from 2 result from the semantic rule that defines $F.val$ in terms of **digit**.$lexua1$. In fact, $F.val$ equals **digit**.$lexval$, but the edge represents dependence, not equality.

Nodes 5 and 6 represent the inherited attribute $T'.inh$ associated with each of the occurrences of nonterminal T' ($T' \to *FT'_1$ and $T' \to \varepsilon$). The edge from 3 to 5 is due to the rule $T'.inh = F.val$, which defines $T'.inh$ at the right child of the root from $F.va1$ at the left child. We see edges to node 6 from node 5 for $T'.inh$ and from node 4 for $F.val$, because these values are multiplied to evaluate the attribute inh at node 6.

Nodes 7 and 8 represent the synthesized attribute *syn* associated with the occurrences of T'. The edge to node 7 from 6 is due to the semantic rule $T'.syn = T'.inh$ associated with production $T' \to \varepsilon$. The edge to node 8 from 7 is due to a semantic rule associated with production 2.

Finally, node 9 represents the attribute $T.val$. The edge from 8 to 9 is due to the semantic rule, $T.val = T'.syn$, associated with production 1 in Table 5.1. □

5.1.2 Ordering the evaluation of Attributes

The dependency graph specify the possible orders in which we can evaluate the attributes at the various nodes of a parse tree. 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

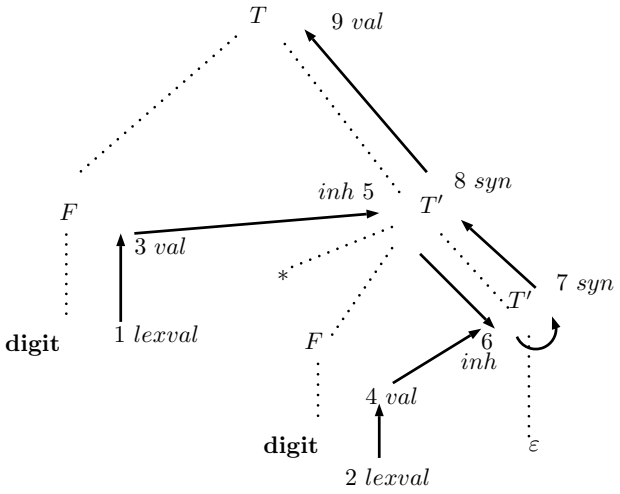

Figure 5.4: Dependency graph for the annotated parse tree of Fig. 5.3.

of N. 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 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, that is, there is no way to evaluate the SDD on this parse tree. If there are no cycles, however, then there is always at least one topological sort. To see why, 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. Make this node the first in the topological order, remove it from the dependency graph, and repeat the process on the remaining nodes.

Example 5.3 *Topological Sort.*

The dependency graph of Fig. 5.4 has no cycles. One topological sort is the order in which the nodes have already been numbered: 1,2,. . . ,9. Notice that every edge of the graph goes from a node to a higher-numbered node, so this order is surely a topological sort. There are other topological sorts as well, such as 1,3,5,2,4,6,7,8,9. □

5.1.3 S-Attributed Definitions

As mentioned earlier, given an SDD, it is very hard to tell whether there exist any parse trees whose dependency graphs have cycles. In practice, translations can be implemented using classes of SDD's that guarantee an evaluation order, since they do not permit dependency graphs with cycles. Moreover, the two classes introduced in this section can be implemented efficiently in connection with top-down or bottom-up parsing.

The first class is defined as follows:

Definition 5.4 *An SDD is S-attributed if every attribute is synthesized.*

Example 5.5 *The SDD of Table. 5.2 is an example of an S-attributed definition. Each attribute, L.val, E.va*1*, T.val, and F.val is synthesized.* □

Table 5.2: 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. It is often especially simple to evaluate the attributes by performing a postorder traversal of the parse tree and evaluating the attributes at a node N when the traversal leaves N for the last time. That is, we apply the function postorder, defined below, to the root of the parse tree:

Algorithm 1 Postorder(N)

```
1: Postorder(N){
2: for (each child C of N, from left to right) postorder(C); do
3:    evaluate the attributes associated with node N;
4: end for
5: }
```

S-attributed definitions can be implemented during bottom-up parsing, since a bottom-up parse corresponds to a postorder traversal. Specifically, postorder corresponds exactly to the order in which an LR parser reduces a production body to its head.

5.1.4 L-Attributed Definitions

The second class of SDD's is called L-attributed definitions. For these definitions, edges of dependency graph, corresponding to the attributes associated with production body, can go from left to right but not from right to left in production symbols. More precisely, each attribute must be either

1. Synthesized, or
2. Inherited, but with the rules limited as follows. Suppose that there is a production $A \to X_1X_2...X_n$, and that there is an inherited attribute $X_i.a$ computed by a rule associated with this production. Then the rule may use only:
   (a) Inherited attributes associated with the head A.
   (b) 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.

(c) Inherited or synthesized attributes associated with this occurrence of X_i itself, but only in such a way that there are no cycles in a dependency graph formed by the attributes of this X_i.

Theorem 5.6 *Show that the SDD given in Table 5.1 is L-attributed.*

Proof.

The SDD in Table 5.1 is L-attributed. To see why, consider the semantic rules for inherited attributes: 1 and 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.

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. □

Example 5.7 *Any SDD containing the following production and rules cannot be L-attributed:*

$$\begin{aligned} &Production: \\ &A \to BC. \end{aligned}$$

$$\begin{aligned} &Semantic\ Rules: \\ &A.s = B.b; \\ &B.i = f(C.c, A.s) \end{aligned}$$

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. While attributes at siblings in a parse tree may be used in L-attributed SDD's, they must be to the left of the symbol whose attribute is being defined. □

5.2 Exercises

1. How a graph can be used for computing attributes? Explain.
2. Construct parse-trees using S-attributed grammar for the following expressions:
   (a) $(3 + 4) * (5 + 6)$

(b) $3 + 4 * (5 + 6)$

(c) $3 + 4 * 5 + 6$

3. What are all the topological sorts for the dependency graph of Fig. 5.4?
4. Explain the process for topological sorting of a graph. Take an example, and execute it.
5. Why the graphs used in attribute grammars are called dependency graphs.
6. Prove that SDD in Table 5.1 is L-attributed grammar.
7. Show that, the SDD with following production and rule is not L-attributed.

$$\begin{aligned} A &\rightarrow BC \\ A.s &= B.b \\ B.i &= f(C.c, A.s) \end{aligned}$$

8. What is the only fundamental difference between S.attributed and L-attributed definitions?
9. In Figure 5.4, explain how the attributes at all the nodes are computed?
10. Does the multiplication in Fig. 5.4 take place at node 5 or 6? Justify.
11. What is search in a topological order in a graph? Explain with example.
12. Describe the method to list the nodes in topological order in a dependency graph, if it exists.
13. Write an algorithm to list the nodes in a dependency graph in topological order. If no such order exists, print "No".
14. List all the possible topological orders of nodes of dependency graph of Figure 5.4. Also, justify these orders.
15. Describe the method you will use to find whether a topological of nodes exists in a dependency graph.
16. What are the differences between S-attributed and L-attributed SDD?