Compiler Design

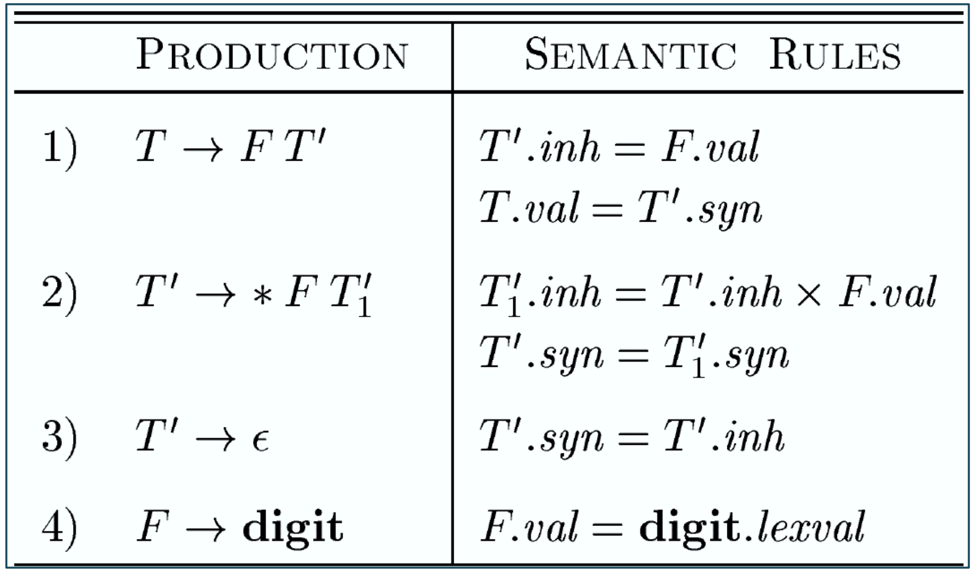
Fatemeh Deldar

Isfahan University of Technology

1403-1404

Evaluating an SDD at the Nodes of a Parse Tree

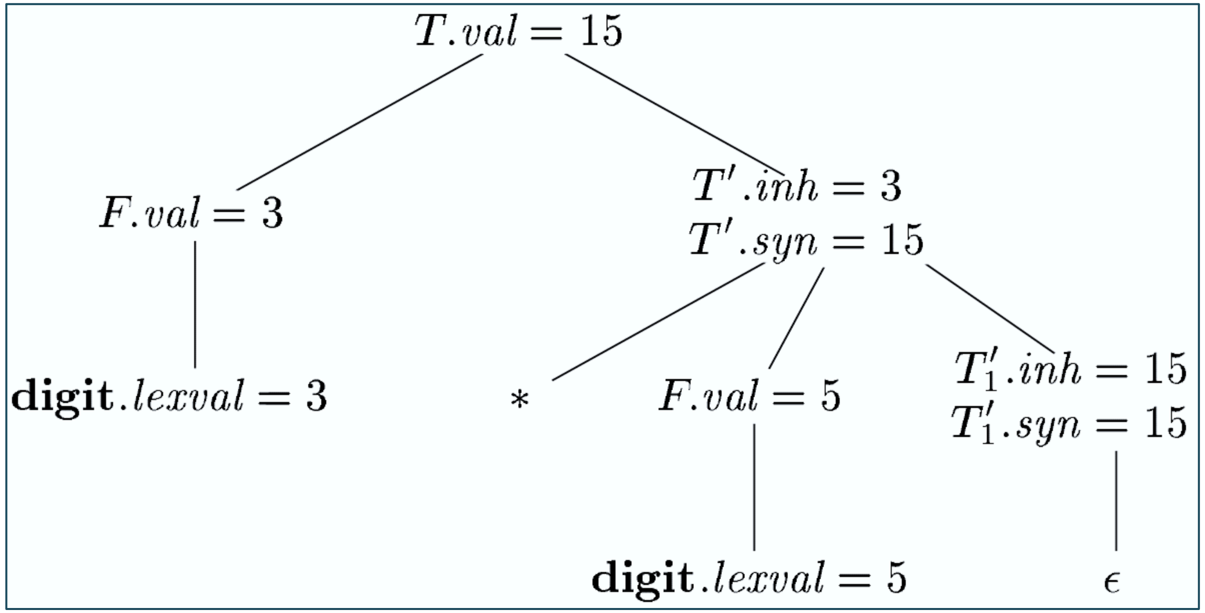
- **Example**
  - In the following SDD, the top-down parse of input $3 * 5$ begins with the production $T \rightarrow FT'$
  - F generates the digit 3, but the operator $*$ is generated by T'
  - Thus, the left operand 3 appears in a different subtree of the parse tree from $*$
  - An inherited attribute will therefore be used to pass the operand to the operator



	PRODUCTION	SEMANTIC RULES
1)	$T \rightarrow F\ T'$	$T'.inh = F.val$ $T.val = T'.syn$
2)	$T' \rightarrow *\ F\ T'_1$	$T'_1.inh = T'.inh \times F.val$ $T'.syn = T'_1.syn$
3)	$T' \rightarrow \epsilon$	$T'.syn = T'.inh$
4)	$F \rightarrow \textbf{digit}$	$F.val = \textbf{digit}.lexval$

Evaluating an SDD at the Nodes of a Parse Tree

- **Example**
  - The semantic rules are based on the idea that the left operand of the operator * is inherited
  - Given a term $x * y * z$, the root of the subtree for $* y * z$ inherits x
  - Then, the root of the subtree for $* z$ inherits the value of $x * y$, and so on
  - Once all the factors have been accumulated, the result is passed back up the tree using synthesized attributes
  - **Annotated parse tree for $3 * 5$**

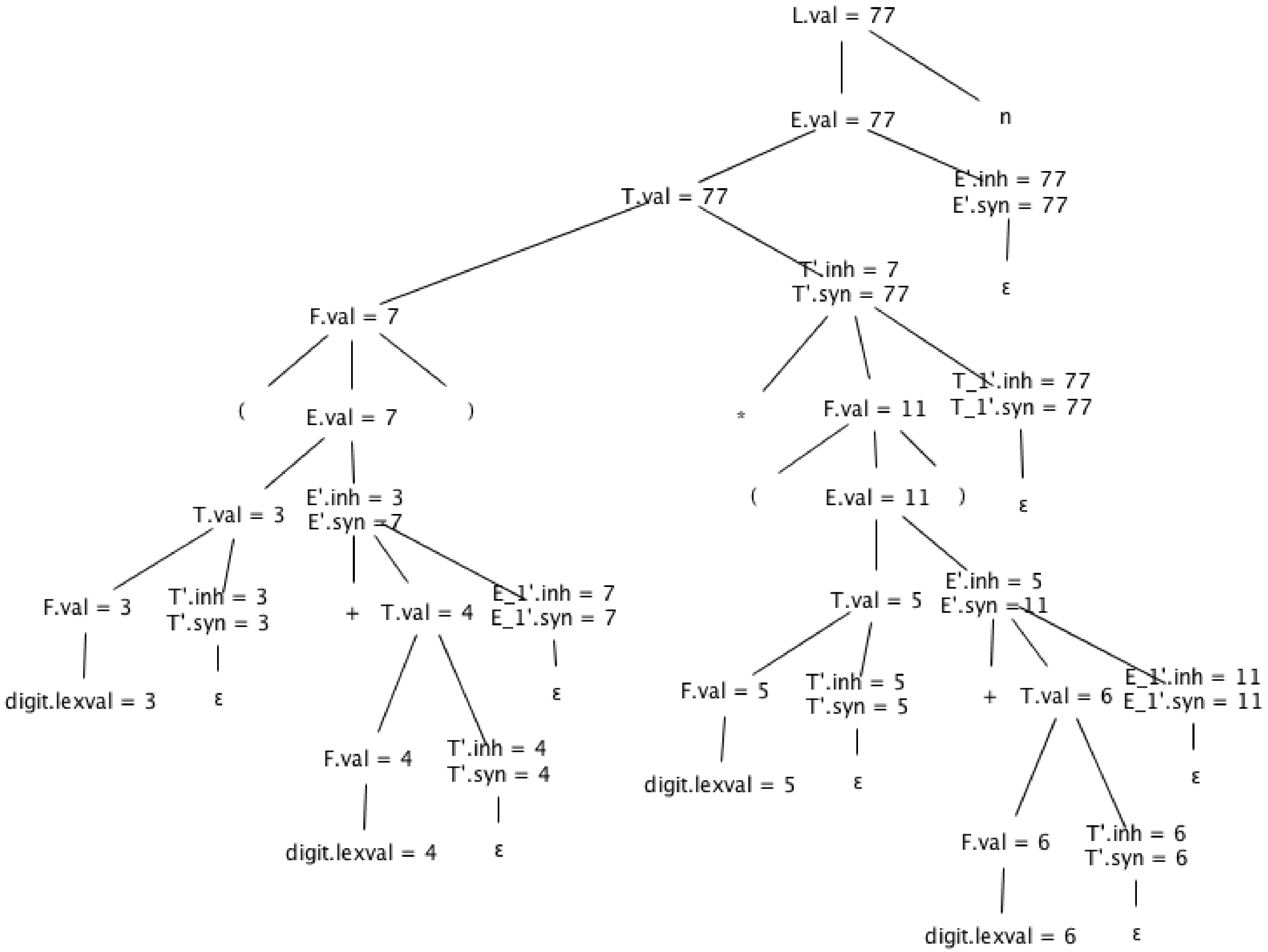


Evaluating an SDD at the Nodes of a Parse Tree

- **Exercise**

	Production	Semantic rules
1)	$L \rightarrow E\ \boldsymbol{n}$	$L.val\ =\ E.val$
2)	$E -> TE'$	$E'.inh\ =\ T.val$ $E.val\ =\ E'.syn$
3)	$E' -> +TE_1'$	$E_1'.inh = E'.inh\ +\ T.val$ $E'.syn = E_1'.syn$
4)	$E' -> \varepsilon$	$E'.syn = E'.inh$
5)	$T -> FT'$	$T'.inh = F.val$ $T.val\ =\ T'.syn$
6)	$T' -> * FT_1'$	$T_1'.inh\ = T'.inh * F.val$ $T'.syn = T_1'.syn$
7)	$T' -> \varepsilon$	$T'.syn = T'.inh$
8)	$F -> (E)$	$F.val = E.val$
9)	$F -> digit$	$F.val = digit.lexval$

- **Exercise (Cont.)**
  - Annotated parse tree for the following expression, using the previous SDD
    - $(3 + 4) * (5 + 6)\ \boldsymbol{n}$

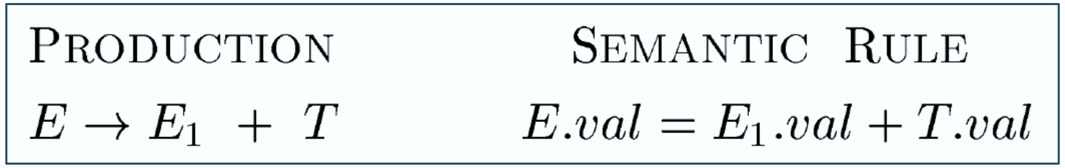


Dependency Graphs

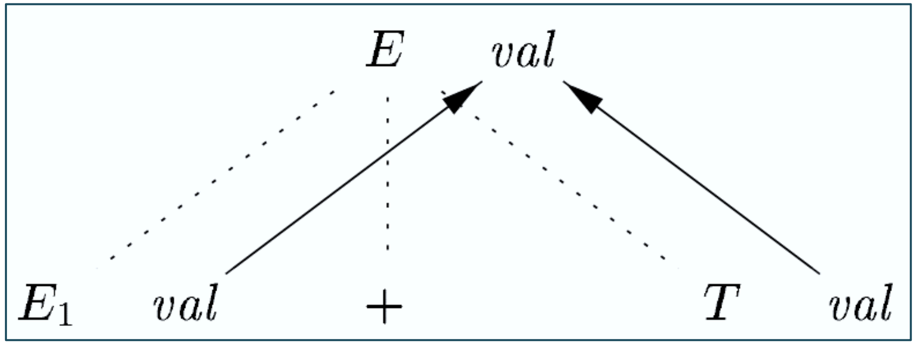
- **Dependency graphs** are a useful tool for determining an evaluation order for the attribute instances in a given parse tree
- ***While an annotated parse tree shows the values of attributes, a dependency graph helps us determine how those values can be computed***
- A dependency graph depicts the flow of information among the attribute instances in a particular 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

Dependency Graphs

- **Example**

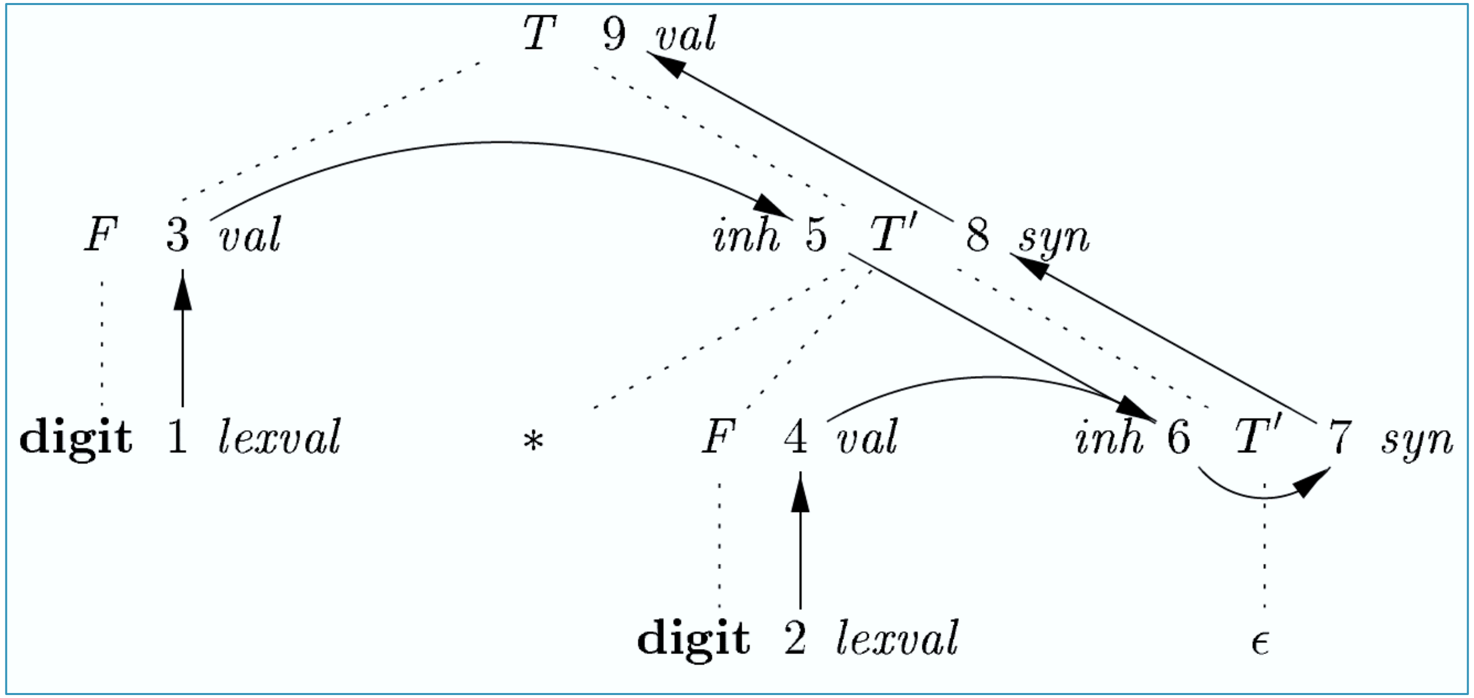


PRODUCTION	SEMANTIC RULE
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$



- **Example**

	Production	Semantic Rules
1)	$T \rightarrow F\,T'$	$T'.inh = F.val$ $T.val = T'.syn$
2)	$T' \rightarrow *\,F\,T'_1$	$T'_1.inh = T'.inh \times F.val$ $T'.syn = T'_1.syn$
3)	$T' \rightarrow \epsilon$	$T'.syn = T'.inh$
4)	$F \rightarrow$ **digit**	$F.val =$ **digit**$.lexval$



Ordering the Evaluation of Attributes

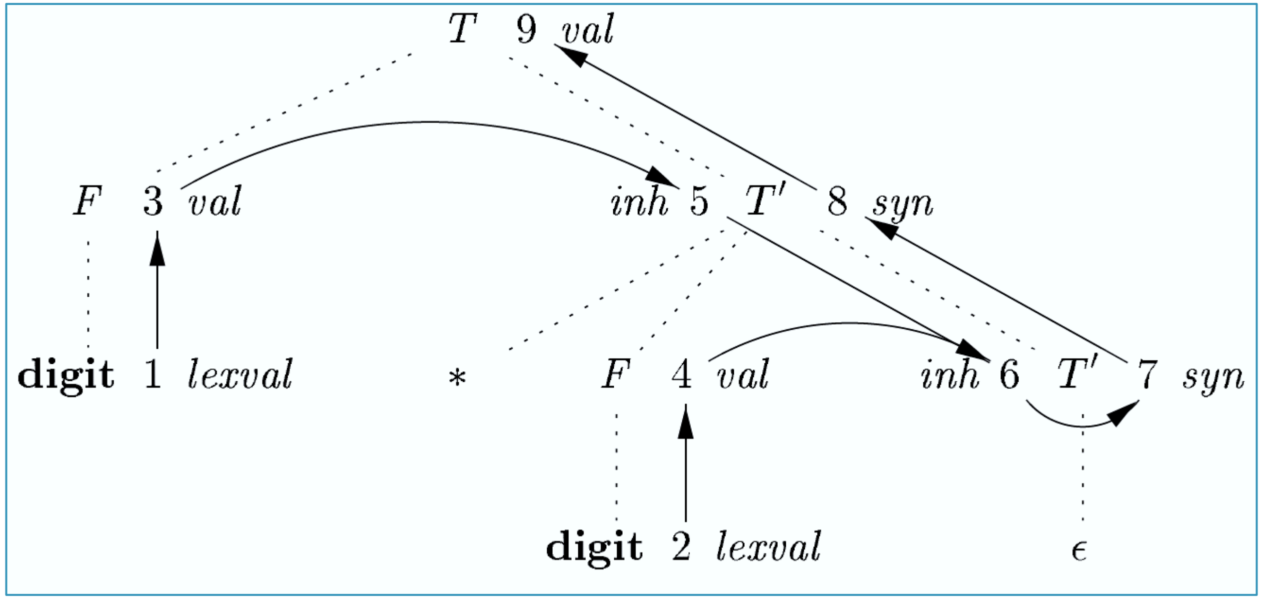
- The dependency graph characterizes the possible orders in which we can evaluate the attributes at the various nodes of a parse tree

- Topological sorts
  - ***Sequences of nodes $N_1, N_2, \dots, N_k$ such that if there is an edge of the dependency graph from N_i to N_j, then $i < j$***

- **If there is any cycle in the graph**, then there are no topological sorts; that is, there is no way to evaluate the SDD on this parse tree

Ordering the Evaluation of Attributes

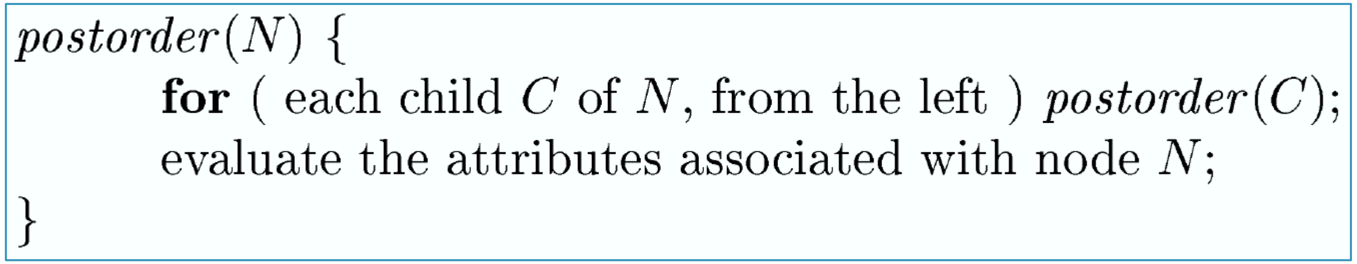
- **Example:** **All topological sorts of the following dependency graph**
  - [1, 2, 3, 4, 5, 6, 7, 8, 9]
  - [1, 2, 3, 5, 4, 6, 7, 8, 9]
  - [1, 2, 4, 3, 5, 6, 7, 8, 9]
  - [1, 3, 2, 4, 5, 6, 7, 8, 9]
  - [1, 3, 2, 5, 4, 6, 7, 8, 9]
  - [1, 3, 5, 2, 4, 6, 7, 8, 9]
  - [2, 1, 3, 4, 5, 6, 7, 8, 9]
  - [2, 1, 3, 5, 4, 6, 7, 8, 9]
  - [2, 1, 4, 3, 5, 6, 7, 8, 9]
  - [2, 4, 1, 3, 5, 6, 7, 8, 9]



S-Attributed Definitions

- In practice, translations can be implemented using classes of SDDs that guarantee an evaluation order, since they do not permit dependency graphs with cycles

- **An SDD is S-attributed if every attribute is synthesized**
  - When an SDD is S-attributed, we can evaluate its attributes in any bottom-up order of the nodes of the parse tree

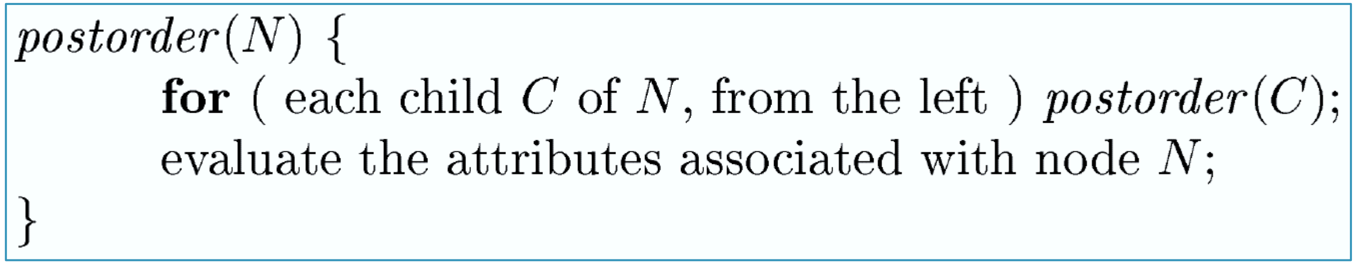


```
$postorder(N)$ {
        for ( each child $C$ of $N$, from the left ) $postorder(C)$;
        evaluate the attributes associated with node $N$;
}
```

S-Attributed Definitions

- In practice, translations can be implemented using classes of SDDs that guarantee an evaluation order, since they do not permit dependency graphs with cycles

- **An SDD is S-attributed if every attribute is synthesized**
    - When an SDD is S-attributed, we can evaluate its attributes in any bottom-up order of the nodes of the parse tree



```
$postorder(N)$ {
        **for** ( each child $C$ of $N$, from the left ) $postorder(C)$;
        evaluate the attributes associated with node $N$;
}
```

S-Attributed Definitions

- تعاریف S-attributed می‌توانند در طی تجزیه پایین به بالا پیاده‌سازی شوند، زیرا تجزیه پایین به بالا متناظر با پیمایش پسوندی است

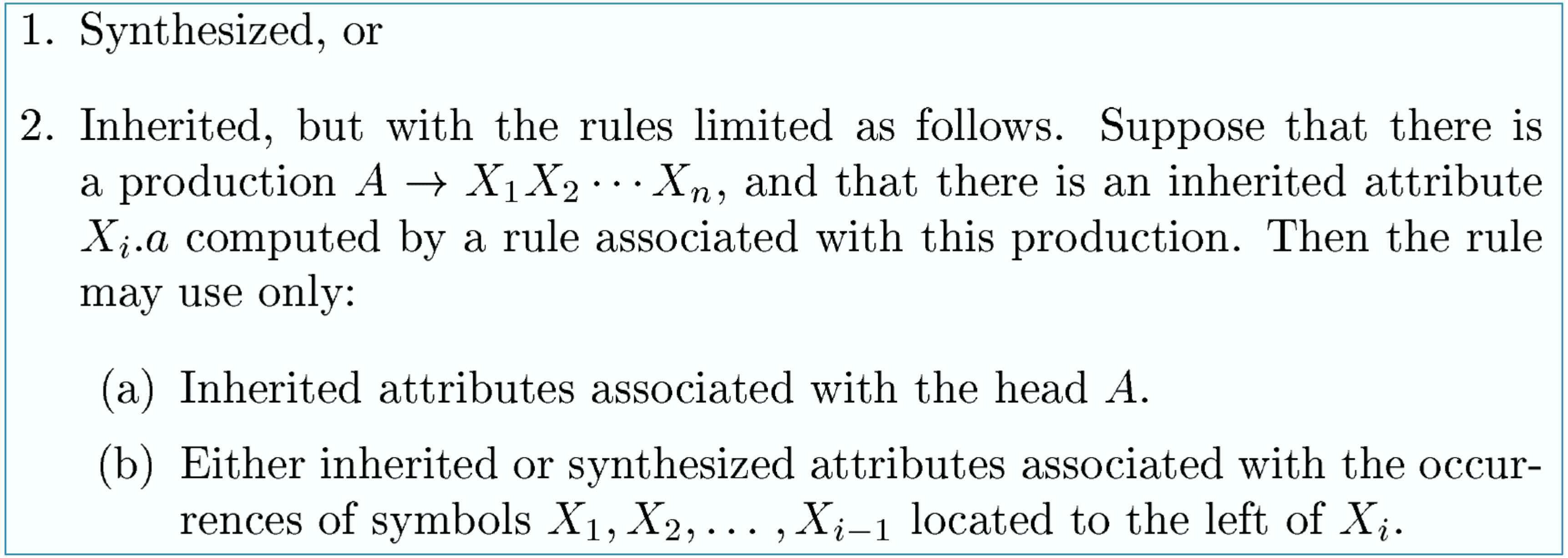
- بنابراین وقتی برای تعریف S-attributed از روش تجزیه پایین به بالا استفاده می‌کنیم، ترجمه می‌تواند در حین عمل تجزیه در طی یک گذر انجام شود و سرعت کامپایل افزایش یابد

- با توجه به اینکه تجزیه پایین به بالا دقیقاً بر اساس پیمایش پسوندی عمل می‌کند، بنابراین ترتیب ارزیابی خصیصه‌ها از قبل مشخص است و نیازی به گراف وابستگی برای تعیین ترتیب ارزیابی خصیصه‌ها نیست

- نکته: تعاریف S-attributed می‌تواند برای تمام گرامرهای LR(1) استفاده شود
  - برای این کار می‌توان یک فیلد اضافی به پشته تجزیه جهت نگهداری مقدار خصیصه‌های synthesized اضافه کرد یا کنار پشته تجزیه یک پشته مقدار جهت نگهداری مقدار خصیصه‌ها داشت

L-Attributed Definitions

- The idea behind this class is that, between the attributes associated with a production body, dependency-graph edges can go from left to right, but not from right to left
- **An SDD is L-attributed if every attribute is**



1. Synthesized, or

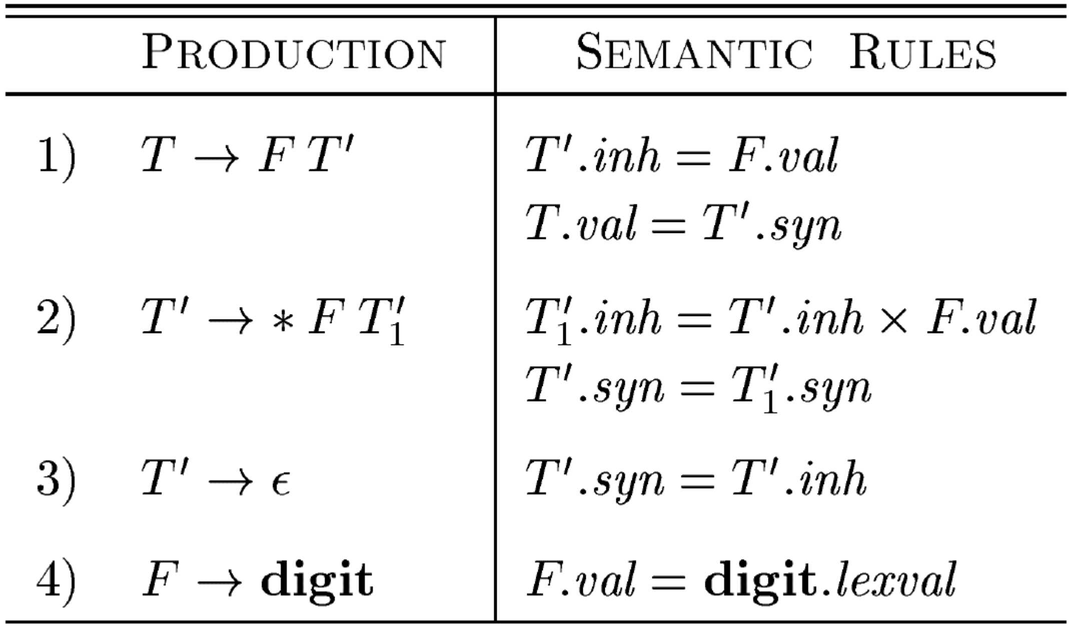
2. Inherited, but with the rules limited as follows. Suppose that there is a production $A \rightarrow X_1 X_2 \cdots X_n$, and that there is an inherited attribute $X_i.a$ computed by a rule associated with this production. Then the rule may use only:

   (a) Inherited attributes associated with the head A.

   (b) Either inherited or synthesized attributes associated with the occurrences of symbols $X_1, X_2, \ldots, X_{i-1}$ located to the left of X_i.

L-Attributed Definitions

- **Example**
  - This SDD is L-attributed



	PRODUCTION	SEMANTIC RULES
1)	$T \rightarrow F\,T'$	$T'.inh = F.val$ $T.val = T'.syn$
2)	$T' \rightarrow *\,F\,T'_1$	$T'_1.inh = T'.inh \times F.val$ $T'.syn = T'_1.syn$
3)	$T' \rightarrow \epsilon$	$T'.syn = T'.inh$
4)	$F \rightarrow \textbf{digit}$	$F.val = \textbf{digit}.lexval$