DATE: 22/10/2013

Compilers Report
Lexical Analyzer

In previous class we learnt about synthesized attributes. Now let us learn about inherited attributes.

INHERITED ATTRIBUTES

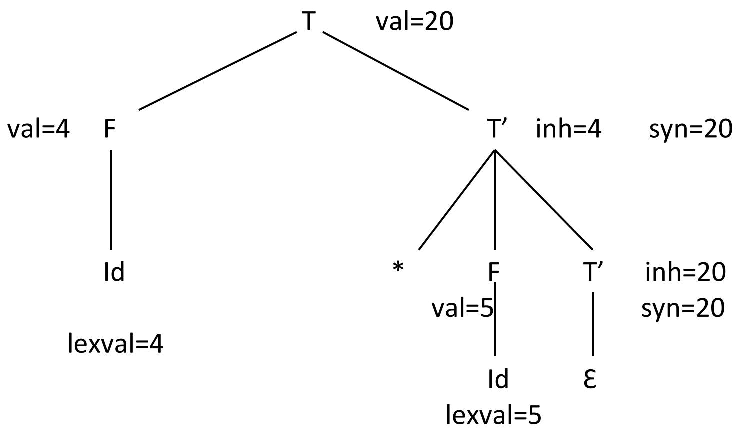
Inherited attributes for a non-terminal is defined by a rule associated with the production at the parent of it. Inherited attributes is defined only in terms of Attributes of Parents, itself and its siblings. It cannot be defined in terms of attributes of the child of node.

Let us understand it with an example.

PRODUCTION	SEMANTIC RULES
T->FT’	T’.inh=F.val T.val=T’.syn
T’->*FT_1’	T_1’.inh=T’.inh*F.val T’.syn=T_1’.syn
T’->Ɛ	T’.syn=T’.inh
F->id	F.val=id.lexval

Non terminal F and T has one synthesized attribute val. id has attribute lexval. Non-Terminal T’ has two attributes. An inherited Attribute inh and Synthesized attribute syn.
The rules are based on fact that left operand of the operator is present on the left sub tree.

Let us use this SDD to compute 4*5.



F->id   id.lexval is the lexval returned by the Lexical Analyzer. F.val = 4. Similarly F.val = 5 in other sub tree.

T->FT'   T' takes the F.val i.e., T'.inh=F.val=4. The left operand is transferred to right sub tree.

T'->*FT1'   T1'.inh takes the value F.val*T'.inh. i.e.,   T1'.inh=4*5 =20

T'-> Ɛ   T'.syn=T'.inh=20

T->FT'   T.syn= T1'.syn=5.

T->FT'   T.val=T.syn=20

4*5 is evaluated.

HOW TO EVALUATE ATTRIBUTES

A parse tree containing or showing the values of its attributes is called Attributed Parse Tree
1) For synthesized attributes we can evaluate the attributes in Bottom up manner.
2) For SDD's with both inherited and synthesized attributes, it can't be ensured that there exists even own order to evaluate the attributes.
Let us consider the following example:

PRODUCTION	SEMANTIC RULE
X->Y	X.syn=f(Y.inh)
	Y.inh=g(X.syn)

$$\begin{tikzcd}
X.syn \arrow[d, shift right] \\
Y.inh \arrow[u, shift right]
\end{tikzcd}$$

In this case we can't evaluate the attributes.

3) Dependency Graphs are used to find the evaluation order for evaluating the attributes.

DEPENDENCY GRAPHS

It depicts the path of flow of information

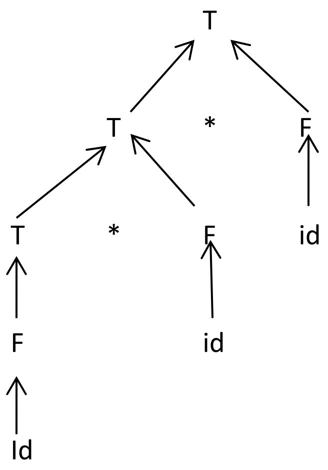
1) For each node in the Parse Tree the Dependency graph has a node for each attribute in the node.
2) For each production of the form X.a=f(Y.a,Z.b,….) create an edge from Y.a node to X.a node,Z.b to X.a and so on. This means Y.a and Z.b are to be evaluated before evaluating X.a

S Attributed SDD

An S Attributed SDD is a SDD if all the attributed is synthesized.

When the SDD is S attributed, it can evaluated in any bottom-up order of the nodes of the parse tree. Generally post order traversal is used to evaluate where attributes of child are evaluated first and then the attribute of it.

```
POSTORDER (N)
{
        for (each child C of N, from the left)
                POSTORDER(C);
        Evaluate the attributes of node N
}
```



An example to S attributes SDD

It is easy to implement the S attributed SDD with Bottom up Parser as Bottom up Parser corresponds to Post Order traversal. In fact using Bottom up Parser, Synthesized attributes can be evaluated without creating the nodes explicitly.

INHERITED ATTRIBUTES

The Dependency Graph for the Grammar which we defined previously is shown below. This is done using the above rules.

$$\begin{tikzcd}
 & T.val & & & & & \\
F.val \arrow[rr] & & T'.inh \arrow[drrr] & & T'.syn \arrow[ulll] & & \\
Id \arrow[u] & & * & F.val \arrow[rr] & & T'.inh \arrow[r] & T'.syn \arrow[ull] \\
 & & & Id \arrow[u] & & \varepsilon &
\end{tikzcd}$$

TOPOLOGICAL SORT

A topological sort of a Directed Acyclic Graph (DAG) is a linear ordering of its vertices such that for every directed edge *uv* from vertex *u* to vertex *v*, *u* comes before *v* in the ordering.
The vertices may represent the task to be done and the edges constraints that one order should be completed before another. Top Sort is possible only is the Graph is Acyclic or else there doesn't any order. Any DAG has at least one topological ordering. It can more than 1 order.

$$\begin{tikzcd}
 & 5 \arrow[dl] \arrow[dr] & & 4 \arrow[dl] \arrow[dr] & \\
2 \arrow[drr] & & 0 & & 1 \arrow[dll] \\
 & & 3 & &
\end{tikzcd}$$

The graph shown above has many valid topological sorts, including:

5, 4, 2, 0, 1, 3
4, 5, 1, 0, 2, 3
5, 2, 4, 0, 1 ,3

How to topological sort a DAG??

Let us find the answer to the question in next class