Attribute Grammar

Context-free grammar $G = (V, \Sigma, P, S)$

Attribute set A
each nonterminal X has set of attributes $A(X)$ divided into disjoint sets $I(X)$ inherited and $S(X)$ synthesized.

Semantic domain SD
consists of collection of types and functions on those types. Every $a \epsilon A$ has a type.

Finite set R of attribute evaluation rules
associated with productions

Given $p : X_{p_0} \rightarrow X_{p_1} ... X_{p_{n_p}}$,
(a, p, k) is an attribute occurrence if $a \epsilon A(X_{p_k})$

It is a defining occurrence if $(s, p, 0)$ and $s \epsilon S(X_{p_0})$ or if (i, p, k) where $i \epsilon I(X_{p_k})$ and $1 \leq k \leq n_p$.

In the evaluation rule

$$(a, p, k) = f((a_1, p, k_1), ...(a_m, p, k_m)),$$

(a, p, k) depends each on $(a_i, p, k_i), 1 \leq i \leq m$

An attribute grammar is in normal form if each attribute definition depends only on

$$I(X_{p_0}) \cup S(X_{p_i}), 1 \le i \le n_p$$

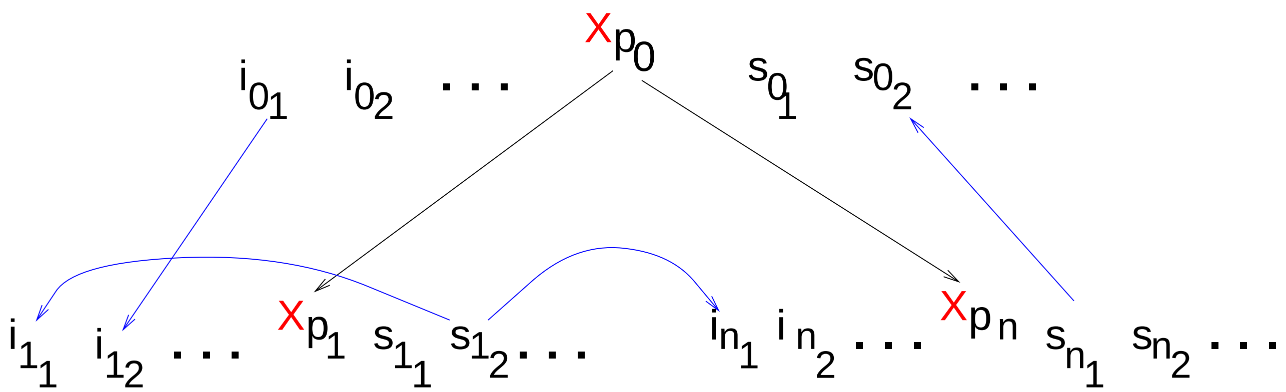
(i.e., only on attributes defined by other productions)

Attribute instances can be evaluated in any dependency order.

Given a parse tree T, attribute instances are associated with nodes in the obvious way.

$D(p)$ - dependency graph for production

$$p : X_{p_0} \to X_{p_1} \ldots X_{p_{n_p}}$$



$\longrightarrow$ is a dependency edge from use to definition

Given a parse tree T, D(T) can be constructed from the local dependency graphs D(p)

Theorem An attribute grammar can be put in normal form if every D(p) is acyclic

An attribute grammar is circular if there exists a parse tree T such that D(T) has a circular dependency path

Circularity testing is exponential in the number of possible dependency subgraphs between some inherited attribute i of X and some synthesized attribute s of X, for each $X \epsilon V$. (In practice, this number is small)

If D(T) is noncircular, it can be evaluated in any dependency order.

Evaluation Orders

Static evaluator - depends only on grammar (not on instances)

Two single-visit evaluators:

S-attributed definition
All attributes are synthesized.

Can be implemented by single bottom-up tree walk or by SDT using semantic stack

L-attributed definition (L-AG)
All attributes either

1. synthesized

2. dependent on attributes of children or inherited attributes of self

3. inherited, but dependent only on inherited attributes of parent and any attributes of siblings *to the left*

Can be implemented by top down left-to-right depth-first traversal of tree

```
procedure eval (u : node);
begin
    let p be the production rooted at u

    for c ← 1 to n do
        j ← c {determines visit order}
        evaluate all i-attributes of u_j
            {can depend on siblings}
        eval (u_j)
    end

    evaluate all s-attributes of u

end eval;
```

eval(root)

for visit to u, order of evaluation is
i-attributes of u; $\underbrace{\text{visits to children; s} - \text{attributes of } u}_{eval(u)}$

Can be implemented by bottom-up SDT using a semantic stack and some extra ϵ-rules