CS412/CS413

Introduction to Compilers

Tim Teitelbaum

Lecture 15: Partitioned Attribute Grammars

22 Feb 08

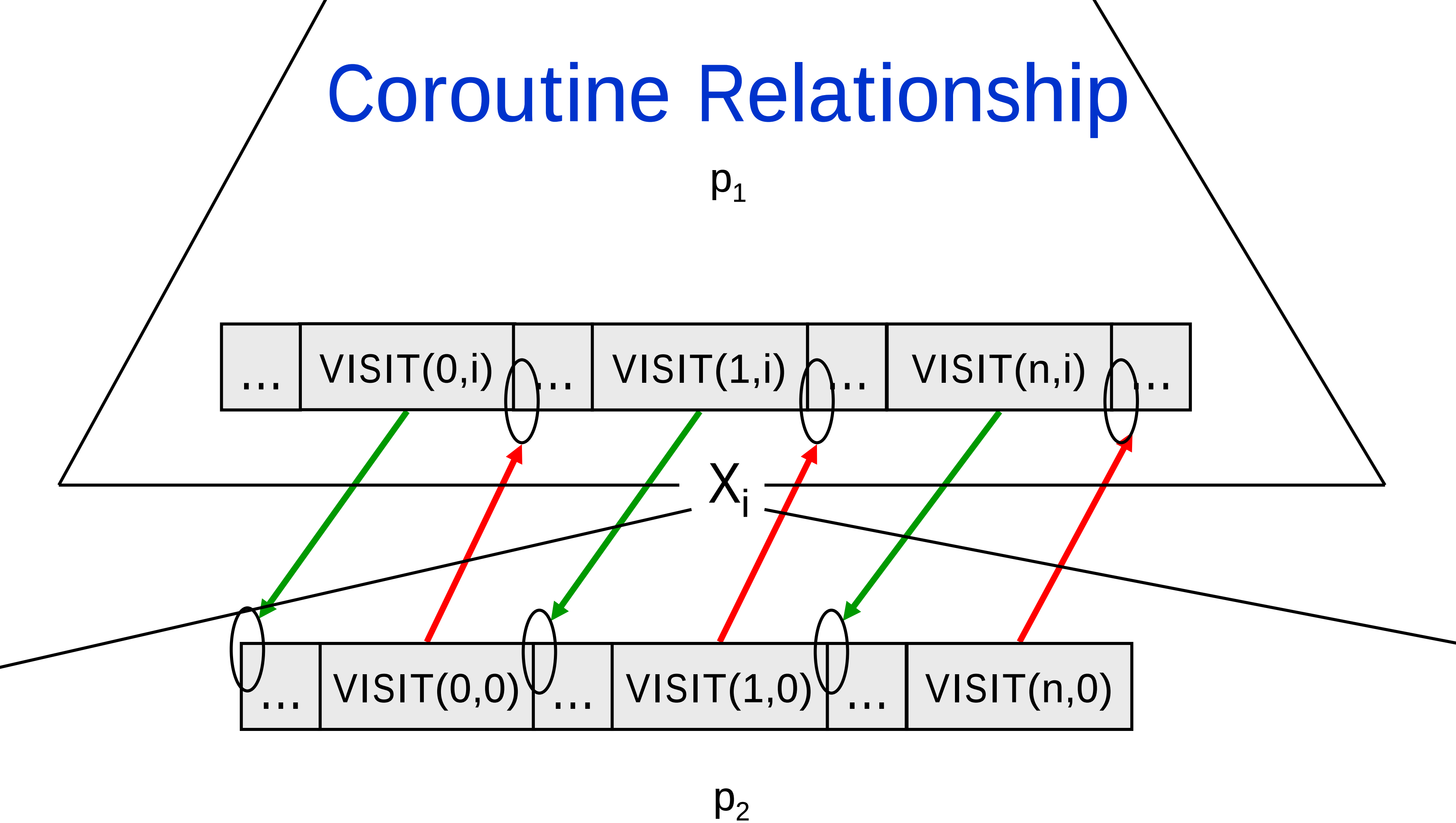
Static Attribute Evaluation

- Analyze the grammar and determine a fixed tree traversal scheme (with interleaved evaluations) such that for any possible derivation tree, evaluations will be in topological order
- Partitioned attribute grammars are a large class that lends itself to efficient analysis and evaluation

Plans

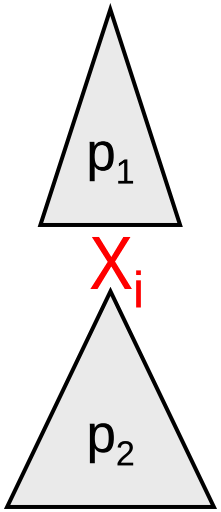
- Each production $X_0 \rightarrow X_1 \ldots X_n$ will have one associated plan
- A plan is a linear sequence of instructions, where an instruction is one of
  - EVAL $X_i.a$ evaluate attribute a of symbol X_i
  - VISIT(r,i) r-th visit to neighbor i [child 0 = parent]
- If-then-else's in plans would permit different execution orders in different contexts, but we chose to allow only straight-line plans for simplicity and efficiency

Coroutine Relationship



- VISIT instructions act as coroutine calls

Interface

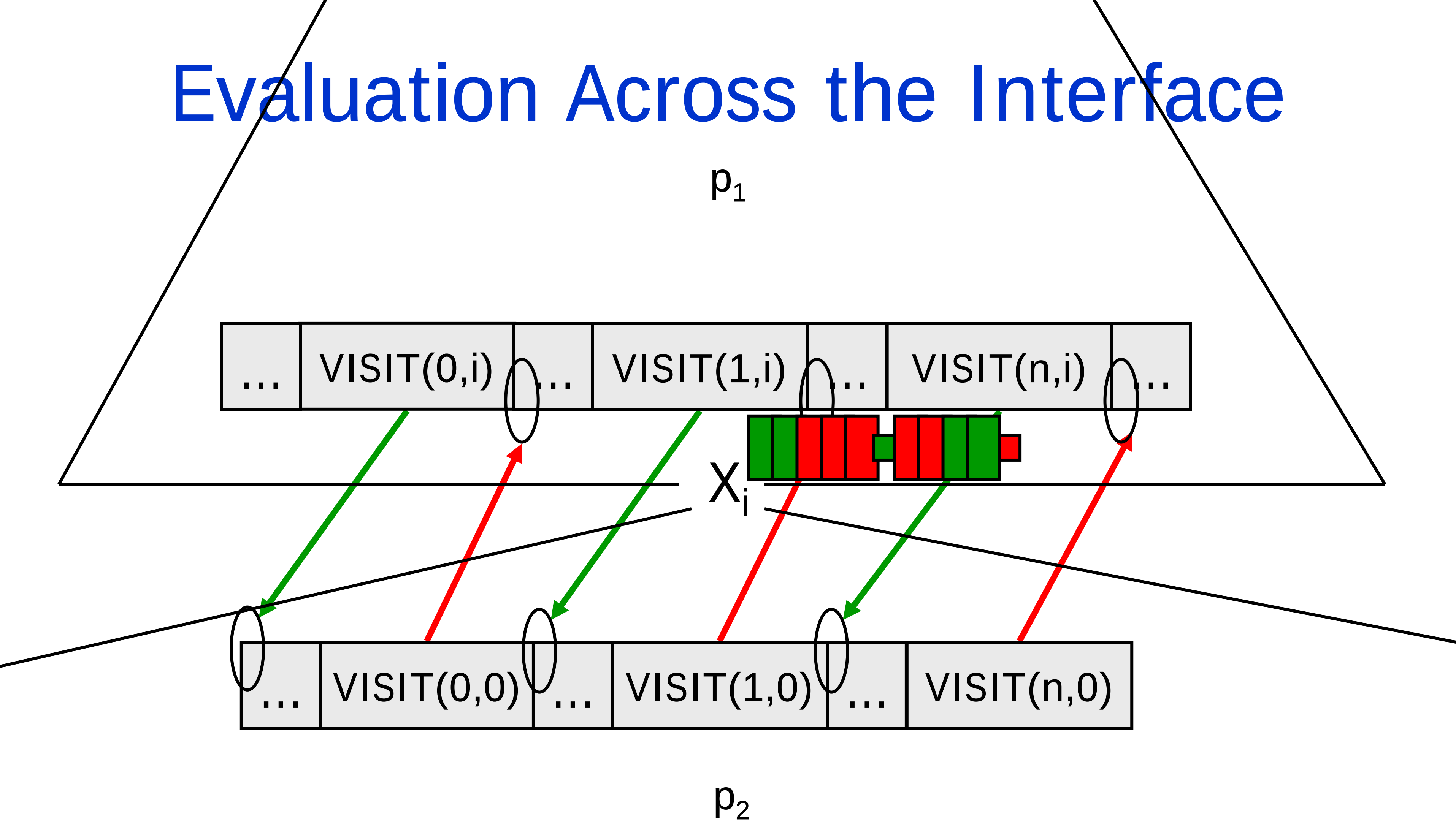


p_1: $X_0 \rightarrow X_1 \ldots X_i \ldots X_n$

p_2: $X_i \rightarrow \ldots$

- The attributes of X_i constitute an interface between the plans for p_1 and p_2.
  - The plan for p_1 evaluates inherited attributes of X_i
  - The plan for p_2 evaluates synthesized attributes of X_i

Evaluation Across the Interface



- VISIT instructions act as coroutine calls

Consistency of Plans

- The plan for p_1 must be consistent with the plans for all productions

$$X_i \rightarrow \alpha$$

- The plan for p_2 must be consistent with the plans for all productions

$$A \rightarrow \alpha\ X_i\ \beta$$

Plans are Fragments of Topological Orders

- The plans must be constructed so that for any derivation tree T, when the plan instances are "wired up" by VISITs, the order of EVALs are a topological order for D(T)

AG for which no such plans exist

$Z \rightarrow s\ X_1\ X_2$

$x_1.a = x_2.d$

$X_1.c = 1$

$x_2.a = x_1.d$

$X_2.c = 2$

$Z \rightarrow t\ X_1\ X_2$

$X_1.a = 3$

$X_1.c = X_2.b$

$X_2.a = 4$

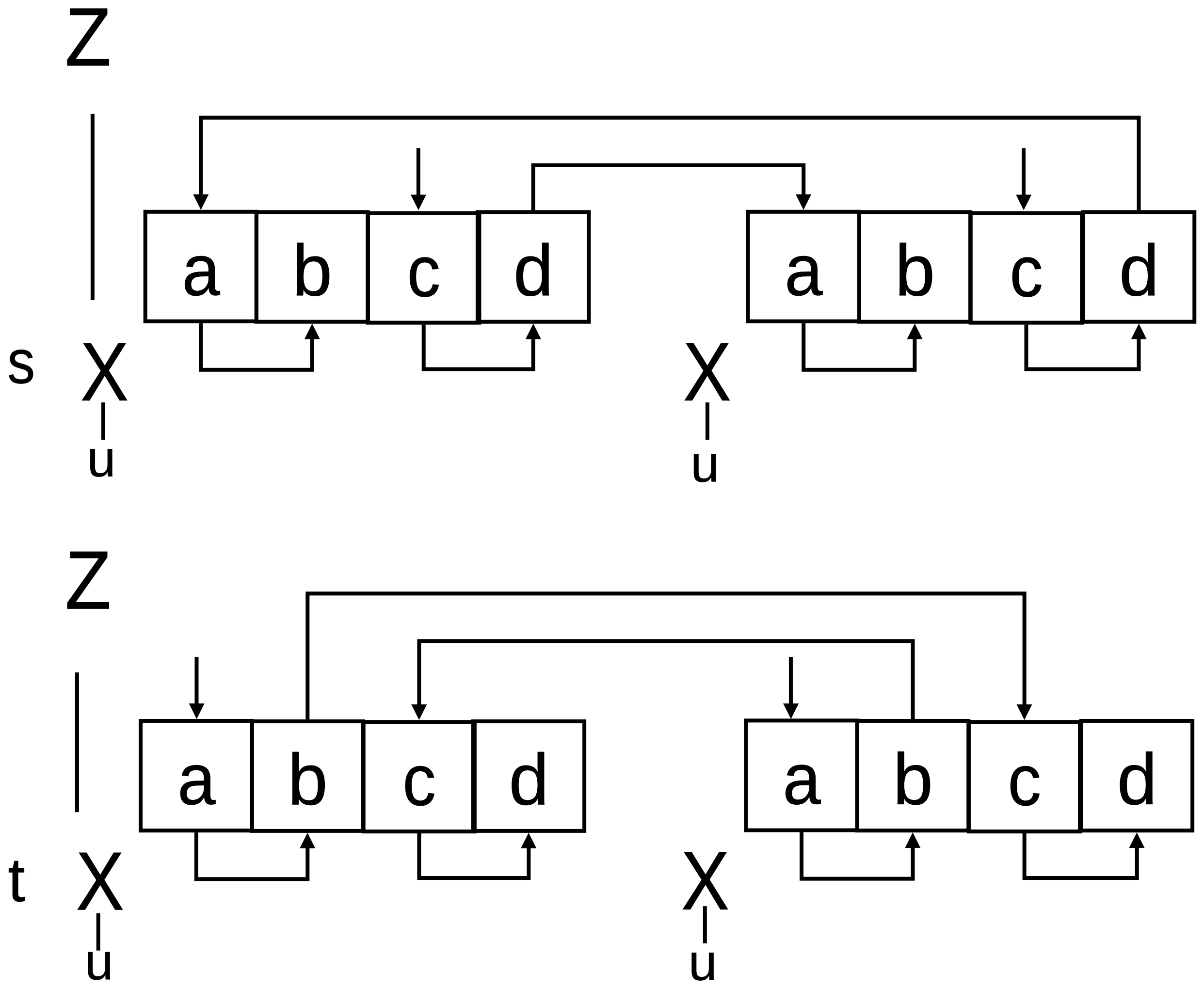
$X_2.c = X_1.b$

$X \rightarrow u$

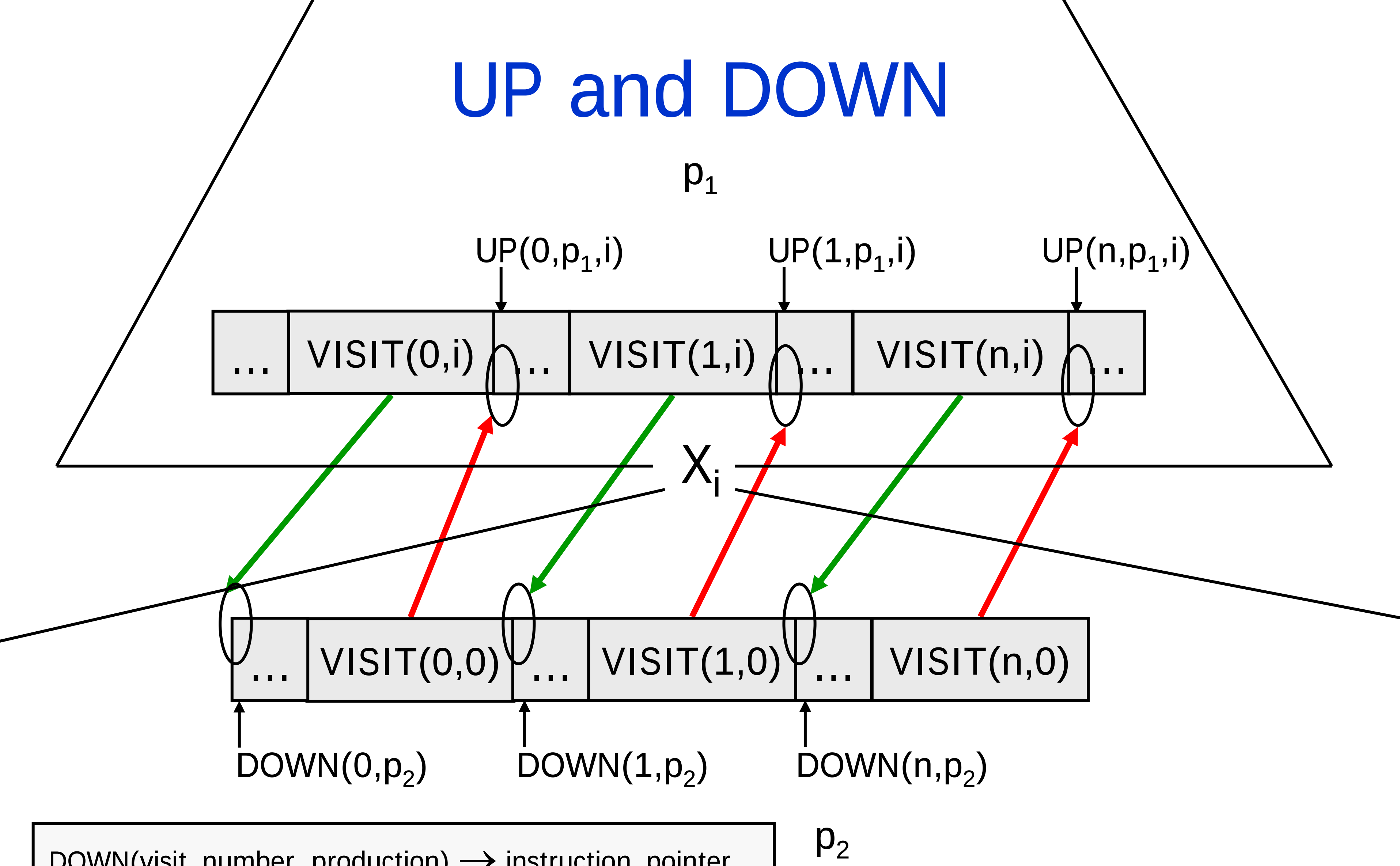
$X.b = X.a$

$X.d = X.c$

AG for which no such plans exist



UP and DOWN



DOWN(visit_number, production) $\longrightarrow$ instruction_pointer

UP(visit_number, production, child) $\longrightarrow$ instruction_pointer

Evaluator

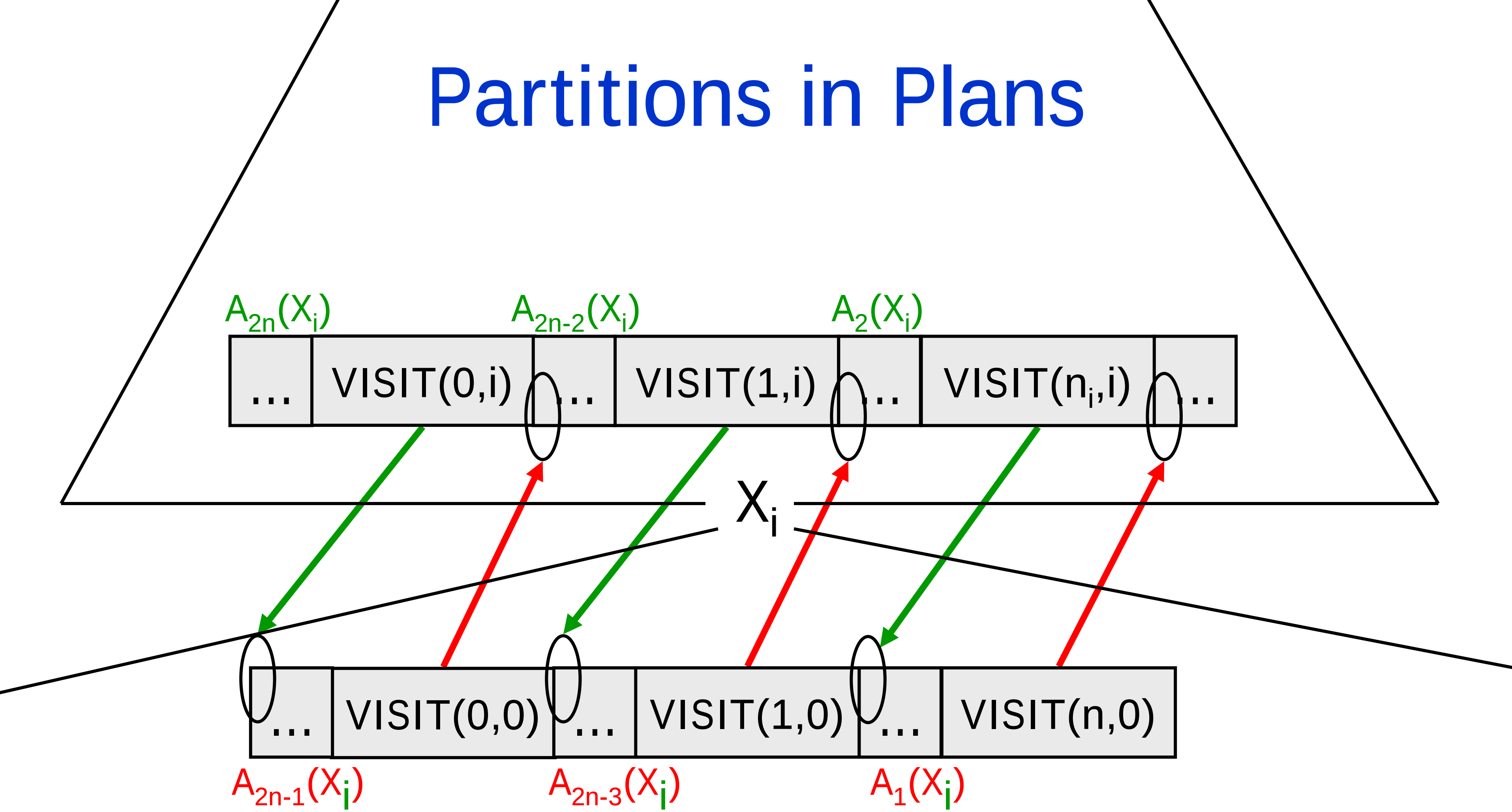
```
node := root;
ip := DOWN(0, root.rule);
repeat
   case state of
         Xi.a:     {
                   evaluate Xi.a;
                   increment ip;
                   }
         VISIT(r,i), i>0: {                    /* child visit */
                   ip := DOWN(r, Xi.rule);
                   node := Xi;
                   }
         VISIT(r,0): {                         /* parent visit */
                   ip := UP(r, node.parent.rule, node.child_number);
                   node = node.parent;
                   }
   end case
until node = root and instruction at ip = VISIT(1,0);
```

Partitions

- If such plans are to exist, there must exist, for each nonterminal X, a partition of A(X) into classes $A_{2n}(X)$, $A_{2n-1}(X)$, ..., $A_2(X)$, $A_1(X)$, where
  - even $A_i(X)$ are subsets of IA(X)
  - odd $A_i(X)$ are subsets of SA(X)

  s.t., for every derivation tree T, and every nonterminal instance X in T, the attribute instances of X can be evaluated in the order $A_{2n}(X)$, $A_{2n-1}(X)$, ..., $A_2(X)$, $A_1(X)$. Within each $A_i(X)$, the order or evaluation is unconstrained and may differ from plan to plan.

Partitions in Plans



- Every plan involving X_i must respect the partitioning of $A(X_i)$

Partitioned Attribute Grammar

- If, in addition to the existence of the partitions, the grammar is locally acyclic, then it is called partitioned.

Computing Plans from Partitions

- Suppose, for each nonterminal X, we know valid partition

  $A_{2n}(X)$, $A_{2n-1}(X)$, ..., $A_2(X)$, $A_1(X)$

- To compute the plan for production p: $X_0 \rightarrow X_1 \dots X_n$
  - Start with D_p, the direct depencency graph of p
  - For each i in [0..n], and each partition j for attributes of X_i, collapse all input attribute occurrences of the partition class $A_j(X_i)$ into one node labeled VISIT(*,i) merging edges
  - Add edges between consecutive partition classes of the given X_i
  - Topological sort the resulting graph and fill in visit numbers in place of *s

Example

$Z \rightarrow X_1\ X_2$

$X_1.a = 1$

$X_2.a = X_1.b$

$X_1.c = X_2.b$

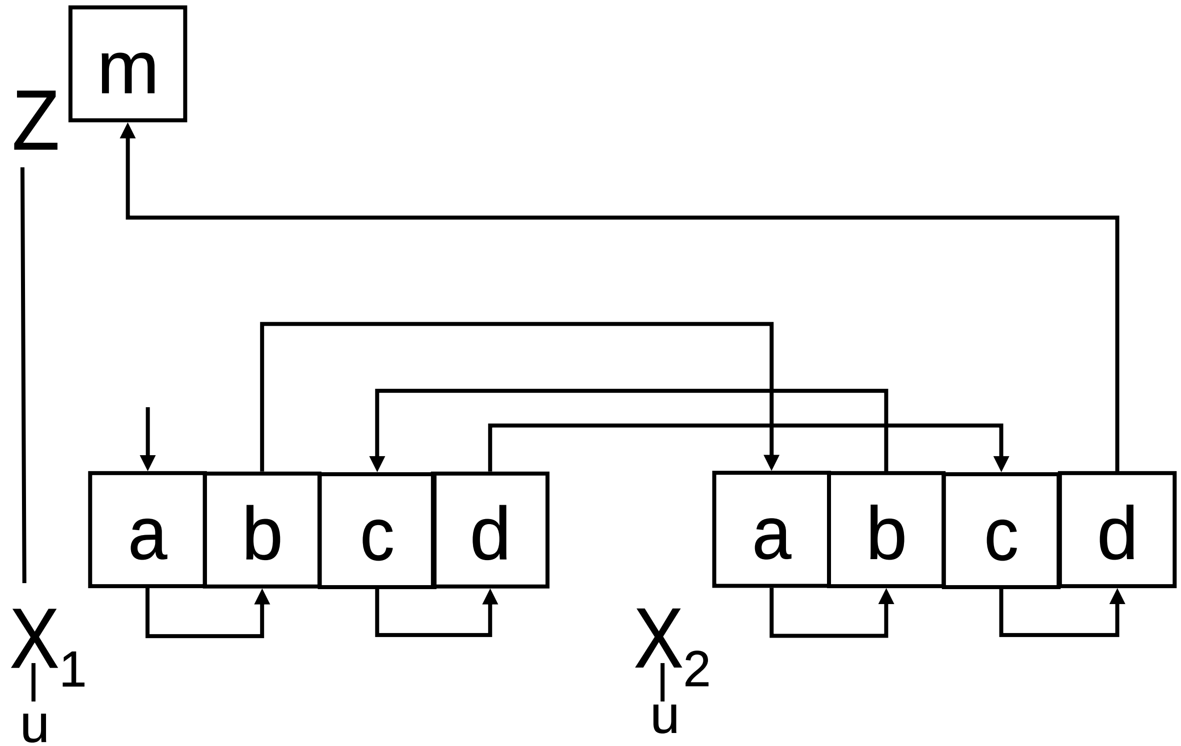
$X_2.c = X_1.d$

$S.m = X_2.d$

$X \rightarrow u$

$X.b = X.a$

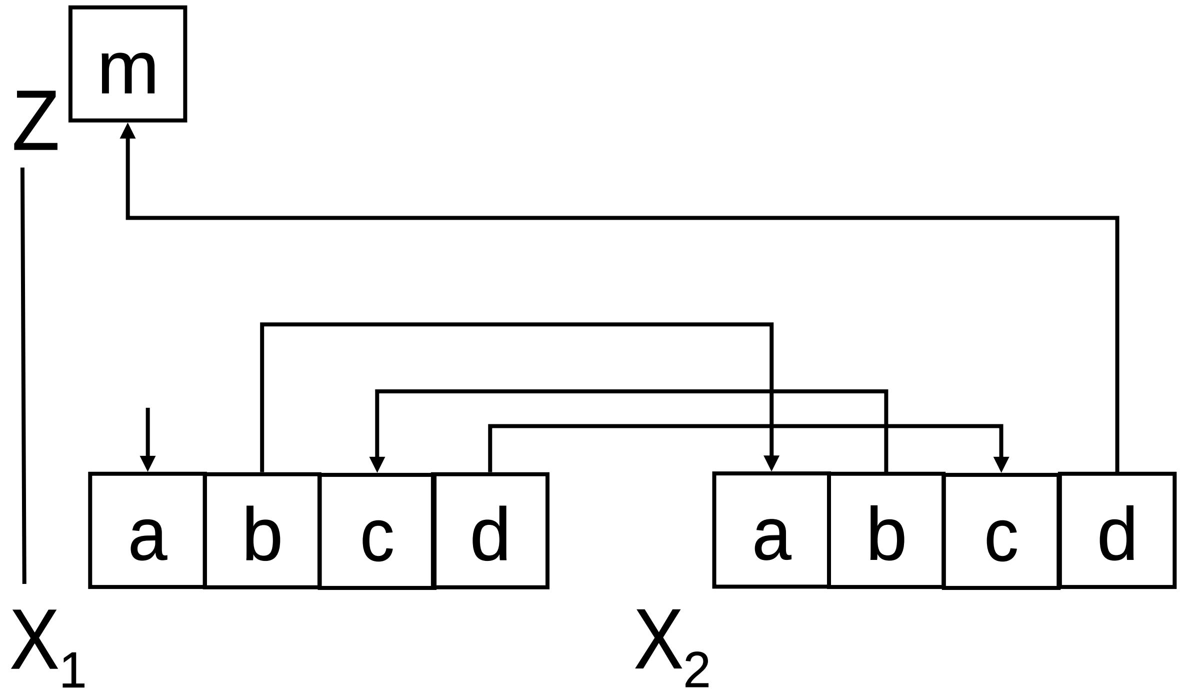
$X.d = X.c$



Partitioning of A(X) = {{a}{b}{c}{d}}

Example

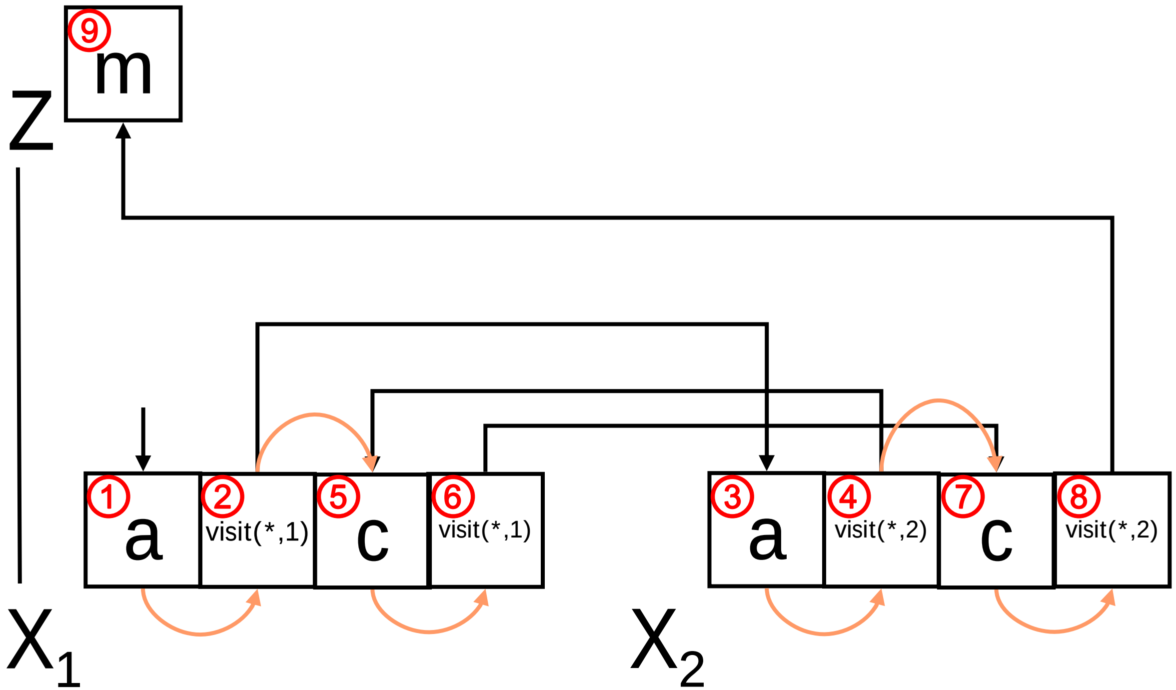
$Z \rightarrow X_1\ X_2$



Partitioning of A(X) = {{a}{b}{c}{d}}

Example

$Z \rightarrow X_1\ X_2$

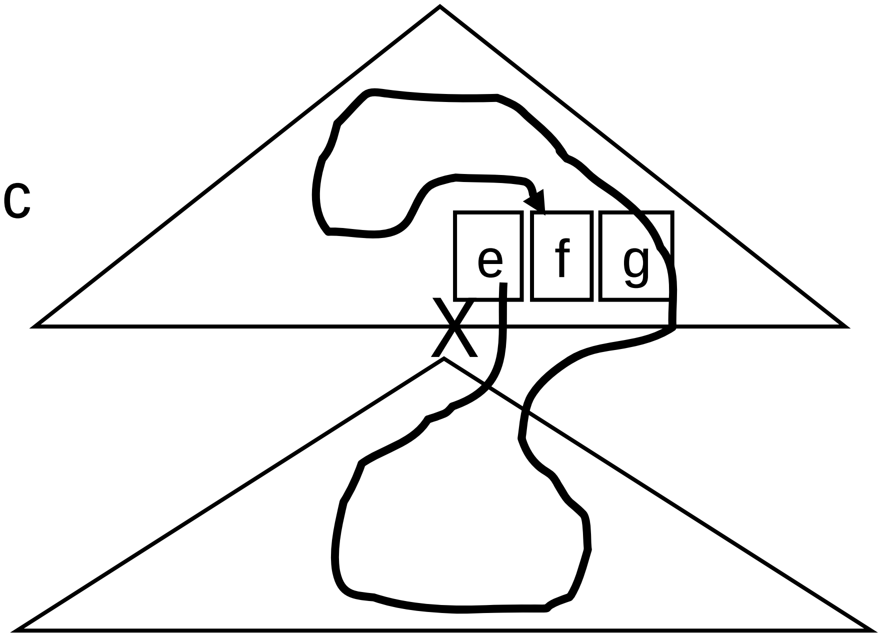


Partitioning of A(X) = {{a}{b}{c}{d}}

Plan: Eval(X_1.a); Visit(0,1); Eval(X_2.a); Visit(0,2);
Eval(X_1.c); Visit(1,1); Eval(X_2.c); Visit(1,2);
Eval(Z.m); Visit(0,0)

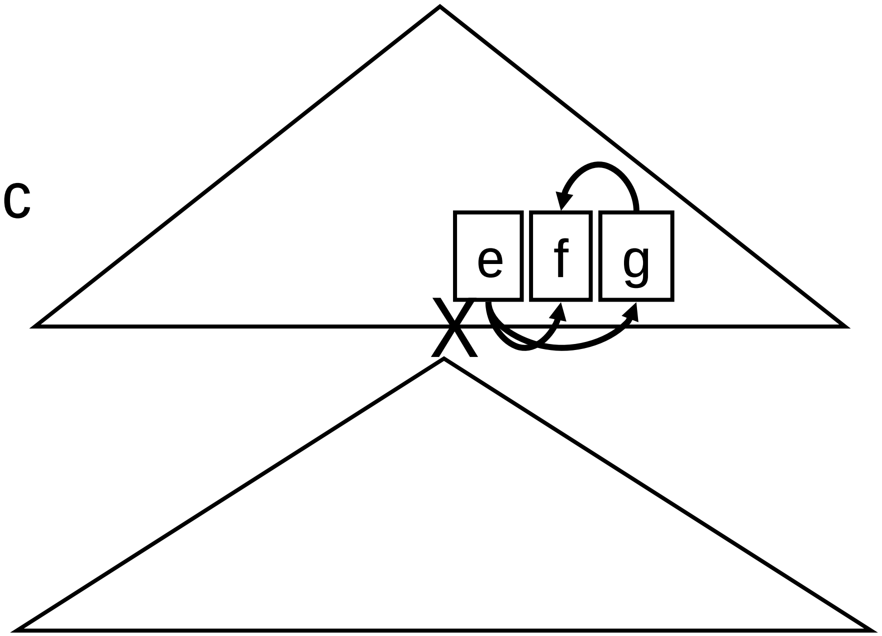
Ordered Attribute Grammars

- For each nonterminal X
  - Construct graph DS(X) = <A(X), E> that over-approximates the transitive dependences that may arise among the attributes of X in some derivation tree
  - Defer how.
  - If DS(X) is cyclic for any X give up.

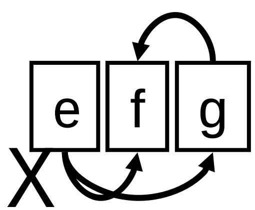


OAG: step 1

- For each nonterminal X
  - Construct a graph DS(X) = <A(X), E> that over-approximates the transitive dependences that may arise among the attributes of X in some derivation tree
  - Defer how.
  - If DS(X) is cyclic for any X give up.



OAG: step 2

- Attempt to compute a partition of A(X) from DS(X) without reference to the productions in which X occurs, as follows:
  - Topological sort DS(X) minimizing alternations between IA(X) and SA(X).
  - Each switch from inherited to synthesized (or vice versa) is a boundary between classes of the partition.
- For example, from 

we get the partition {{e}{g}{f}}

OAG: step 3

- Use the given method for finding a plan from the partitions. If this fails (because topological sort discovers a cycle), then fail.

OAG: step 1, cont.

- To compute DS(X) for all X
  - Simultaniously take transitive closures of the direct dependence graphs D_p for all p, and whenever an edge between two attributes of the same nonterminal occurrence is created, add it to every occurrence of X.
  - When finished, choose the attributes and edges of an arbitrary occurrence of each nonterminal X as DS(X)