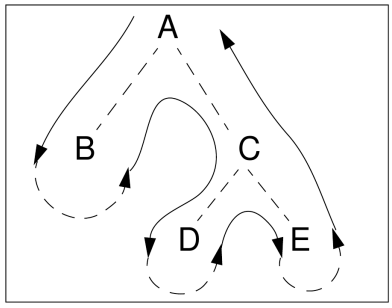
Construction of Attribute Evaluators

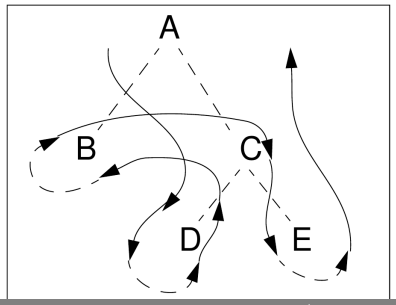
For a given attribute grammar an attribute evaluator is constructed:

- It is **applicable to any tree** that obeys the abstract syntax specified in the rules of the AG.
- It performs a **tree walk** and **executes computations** in visited contexts.
- The execution order obeys the **attribute dependences**.



Pass-oriented strategies for the tree walk:	**AG class:**
k times **depth-first left-to-right**	**LAG (k)**
k times depth-first right-to-left	RAG (k)
alternatingly left-to-right / right-to left	**AAG (k)**
once **bottom-up (synth. attributes only)**	**SAG**

AG is checked if attribute dependences
fit to desired pass-oriented strategy; see LAG(k) check.

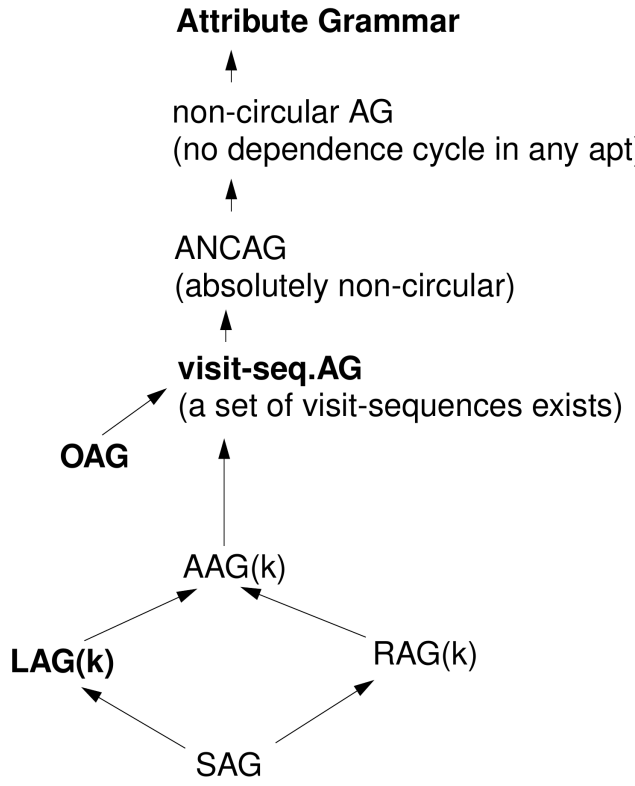


non-pass-oriented strategies:

visit-sequences: **OAG**
an individual plan for each rule of the abstract syntax

A generator fits the plans to the dependences of the AG.

Hierarchy of AG Classes

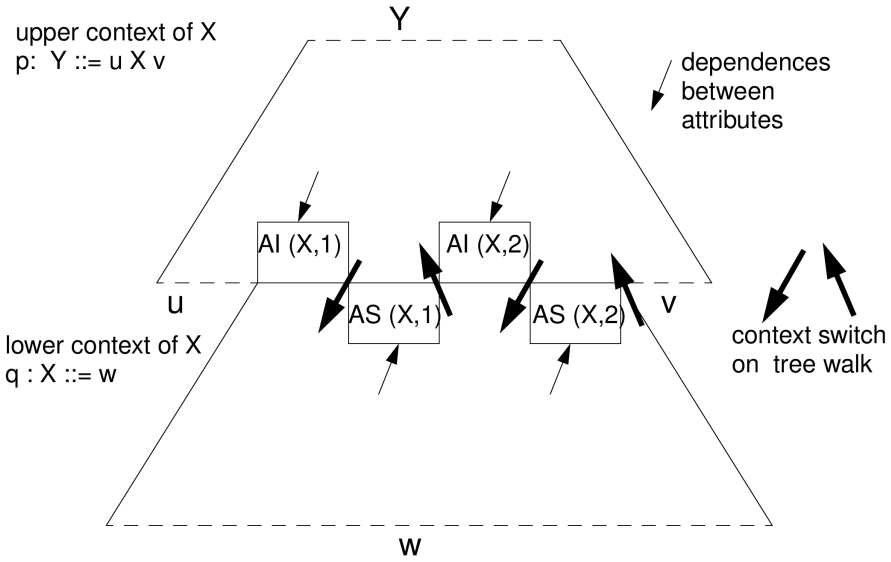


Attribute Partitions

The sets AI(X) and AS(X) are **partitioned** each such that

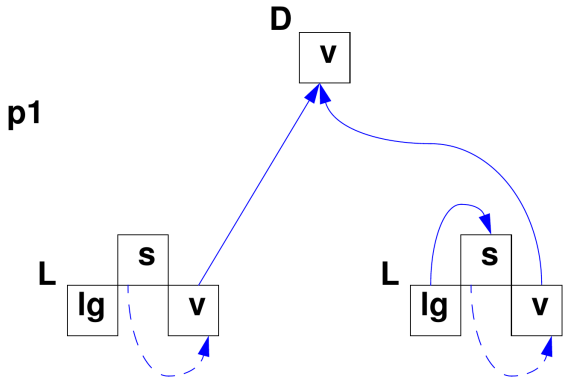
AI (X, i) is computed **before the i-th visit** of X

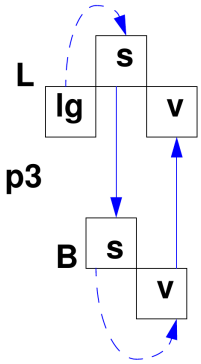
AS (X, i) is computed **during the i-th visit** of X

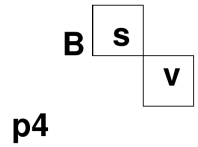


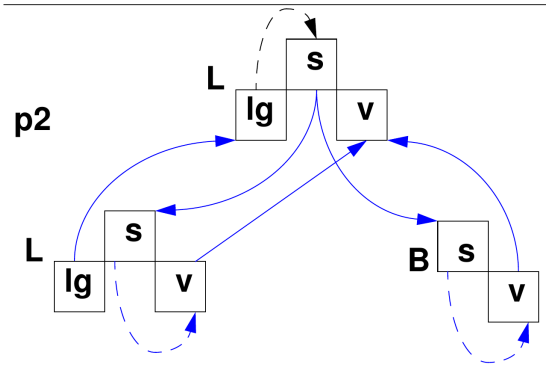
Necessary precondition for the existence of such a partition:
No node in any tree has **direct or indirect dependences that contradict the evaluation order** of the sequence of sets:AI (X, 1), AS (X, 1), ..., AI (X, k), AS (X, k)

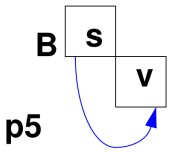
Dependency Analysis for AG "Binary Numbers"











If a tree exists, that has a path from X.a to X.b at some node of Type X, the graphs have an **indirect dependence**

X.a - - - - → X.b

Pass-Oriented Evaluation: LAG(k)

An AG is a LAG(k), if:

For each symbol X there is an **attribute partition** A (X,1), ..., A (X, k),
such that the attributes in **A (X, i) can be computed in the i-th depth-first left-to-right pass**.

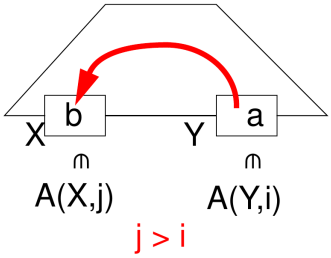
Crucial dependences:

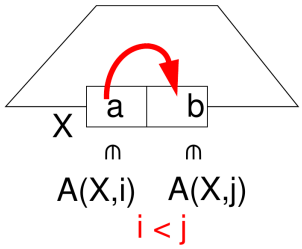
In every dependence graph every dependence

- **Y.a -> X.b** where X and Y occur on the **right-hand side** and **Y is right of X** implies that **Y.a belongs to an earlier pass than X.b**, and
- **X.a -> X.b** where X occurs on the **right-hand side** implies that **X.a belongs to an earlier pass than X.b**

Necessary and sufficient condition over dependence graphs - expressed graphically:

A dependency **from right to left**



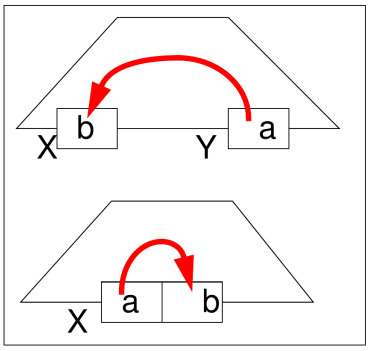


A dependence **at one symbol** on the right-hand side

LAG (k) Algorithm

Algorithm checks whether there is a `k>=1` such that an **AG is LAG(k)**.

Method:
compute iteratively `A(1)`, ..., `A(k)`;
in each iteration try to allocate all remaining attributes to the current pass, i.e. `A(i)`;
remove those which can not be evaluated in that pass



Algorithm:

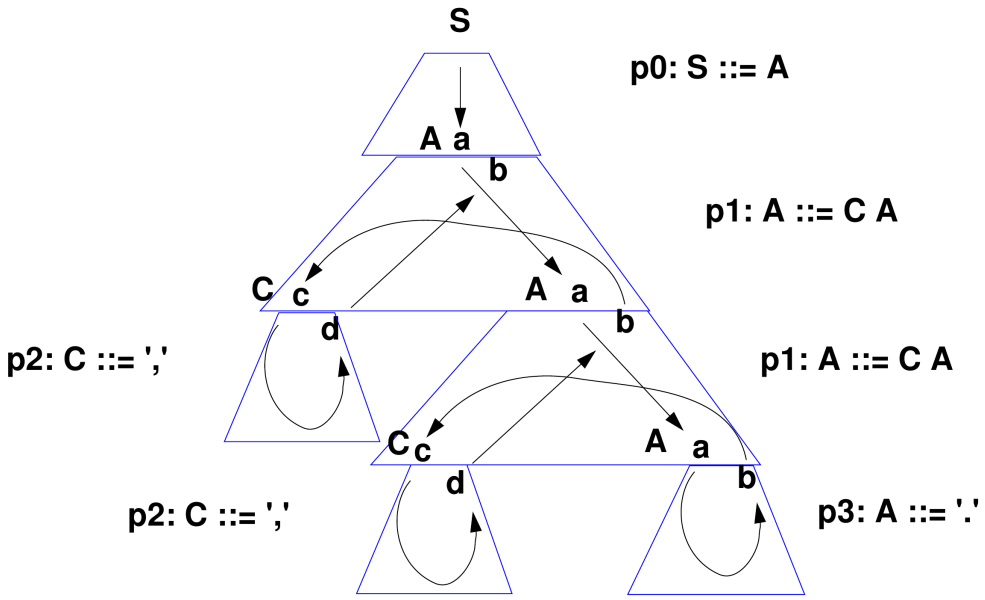
Set `i=1` and `Cand=` all attributes

repeat
set `A(i) = Cand`; set `Cand` to empty;

while still attributes can be removed from `A(i)` do
remove an attribute `X.b` **from** `A(i)` and add it to `Cand` if
- there is a **crucial dependence**
`Y.a -> X.b` s.t.
`X` and `Y` are on the right-hand side, `Y` to the right of `X` and `Y.a` in `A(i)` or
`X.a -> X.b` s.t. `X` is on the right-hand side and `X.a` is in `A(i)`
- `X.b` depends on an attribute that is not yet in any `A(i)`

if `Cand` is empty:	exit: the AG **is LAG(k)** and all attributes are assigned to their passes
if `A(i)` is empty:	exit: the AG **is not LAG(k) for any k**
else:	set i = i + 1

AG not LAG(k) for any k



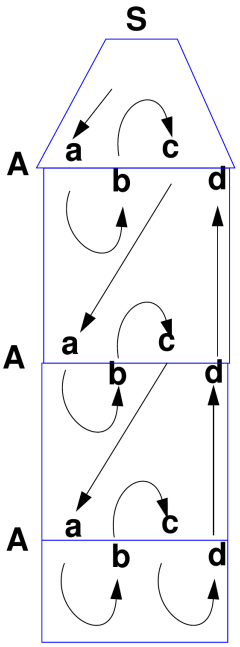
A.a can be allocated to the first left-to-right pass.
C.c, C.d, A.b can not be allocated to any pass.

The AG is RAG(1), AAG(2) and can be evaluated by visit-sequences.

AG not evaluable in Passes

No attribute can be allocated to any pass for any strategy.

The AG can be evaluated by visit-sequences.



p0: S ::= A

p1: A ::= ',' A

p1: A ::= ',' A

p2: A ::= '.'