Semantic Analysis -2

Attribute Grammars - Recap

Let $G = (N, T, P, S)$ be a CFG and let $V = N \cup T$.

Every symbol X of V has associated with it a set of *attributes*

Two types of attributes: *inherited* and *synthesized*

Each attribute takes values from a specified domain

A production $p \in P$ has a set of attribute computation rules for

- synthesized attributes of the LHS non-terminal of p
- inherited attributes of the RHS non-terminals of p

Rules are strictly local to the production p (no side effects)

Attribute Grammars - Recap

An attribute cannot be both synthesized and inherited, but a symbol can have both types of attributes

Attributes of symbols are evaluated over a parse tree by making passes over the parse tree

Synthesized attributes are computed in a bottom-up fashion from the leaves upwards

- Always synthesized from the attribute values of the children of the node
- Leaf nodes (terminals) have synthesized attributes (only) initialized by the lexical analyzer and cannot be modified

Inherited attributes flow down from the parent or siblings to the node in question

Attribute Evaluation Algorithm

Input: A parse tree T with unevaluated attribute instances
Output: T with consistent attribute values

```
{ Let (V, E) = DG(T);
  Let W = {b | b ∈ V & indegree(b) = 0};
  while W ≠ φ do
    { remove some b from W;
      value(b) := value defined by appropriate attribute
                        computation rule;
      for all (b, c) ∈ E do
        { indegree(c) := indegree(c) − 1;
          if indegree(c) = 0 then W := W ∪ {c};
        }
    }
}
```

Attribute Grammars - Example

AG for the evaluation of a real number from its bit-string representation

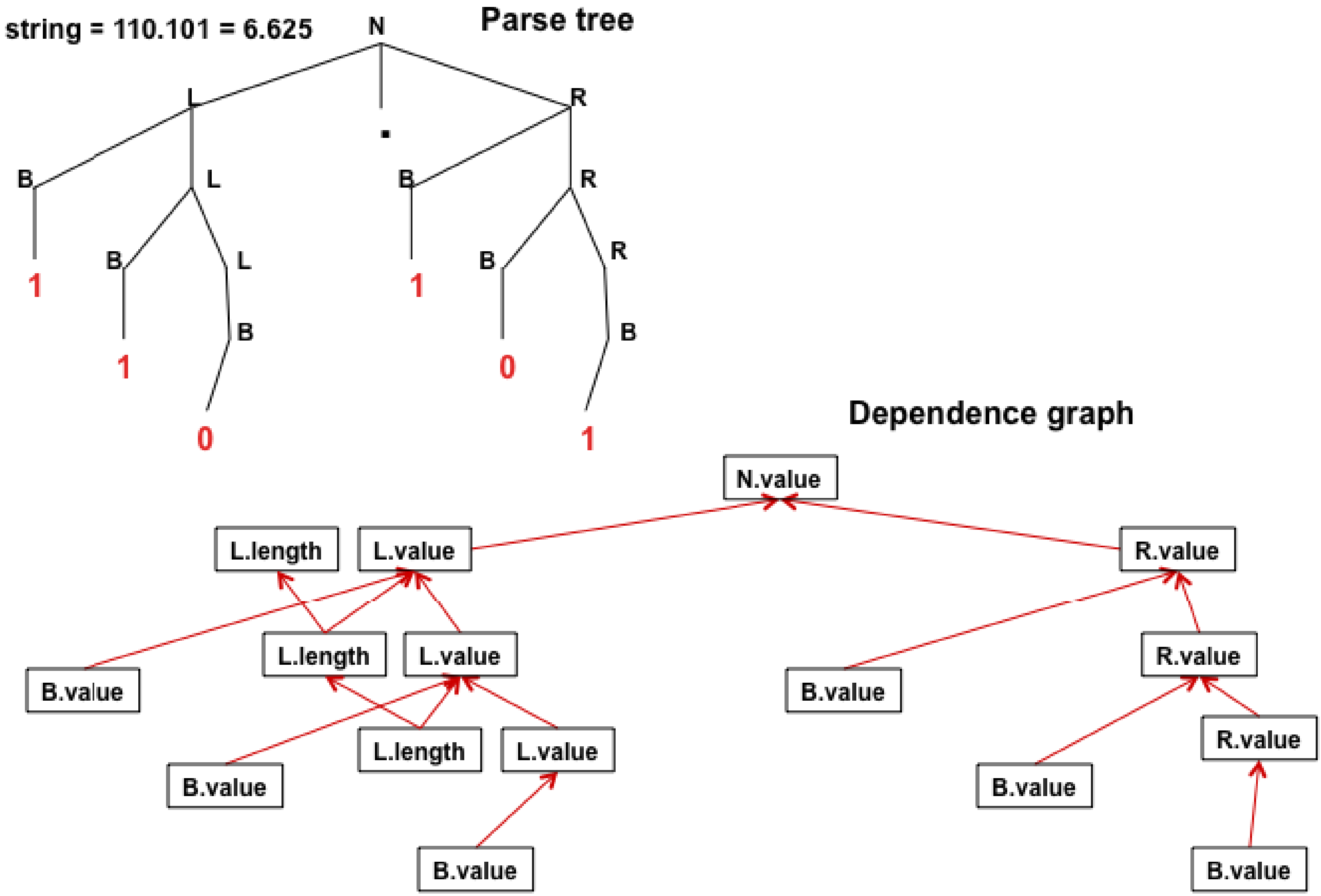
Example: 110.101 = 6.625

$N \rightarrow L.R,\ L \rightarrow BL \mid B,\ R \rightarrow BR \mid B,\ B \rightarrow 0 \mid 1$

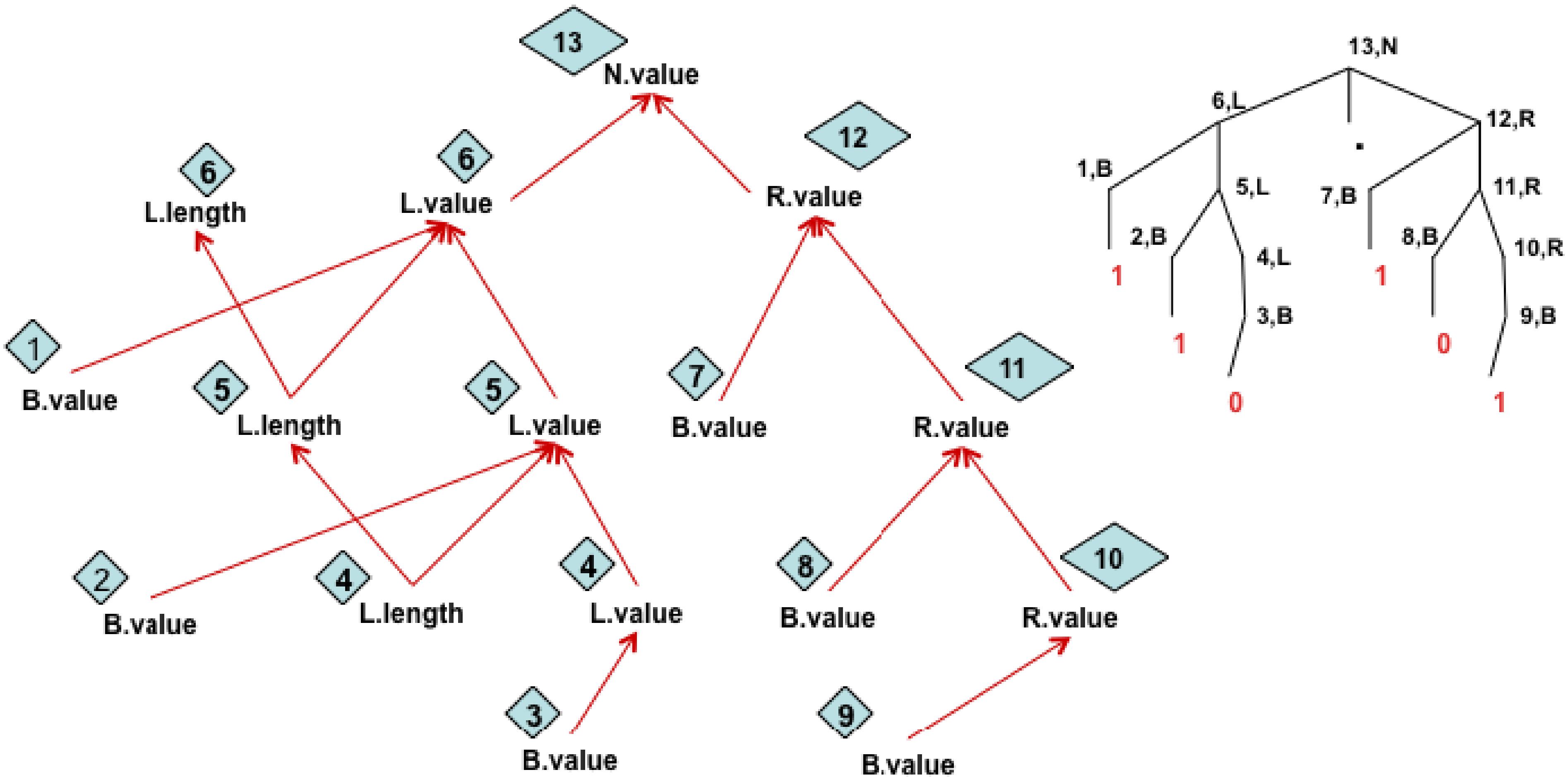
$AS(N) = AS(R) = AS(B) = \{value \uparrow : real\},$
$AS(L) = \{length \uparrow : integer,\ value \uparrow : real\}$

1. $N \rightarrow L.R\ \{N.value \uparrow := L.value \uparrow + R.value \uparrow\}$
2. $L \rightarrow B\ \{L.value \uparrow := B.value \uparrow;\ L.length \uparrow := 1\}$
3. $L_1 \rightarrow BL_2\ \{L_1.length \uparrow := L_2.length \uparrow + 1;$
   $L_1.value \uparrow := B.value \uparrow * 2^{L_2.length\uparrow} + L_2.value \uparrow\}$
4. $R \rightarrow B\ \{R.value \uparrow := B.value \uparrow /2\}$
5. $R_1 \rightarrow BR_2\ \{R_1.value \uparrow := (B.value \uparrow + R_2.value \uparrow)/2\}$
6. $B \rightarrow 0\ \{B.value \uparrow := 0\}$
7. $B \rightarrow 1\ \{B.value \uparrow := 1\}$

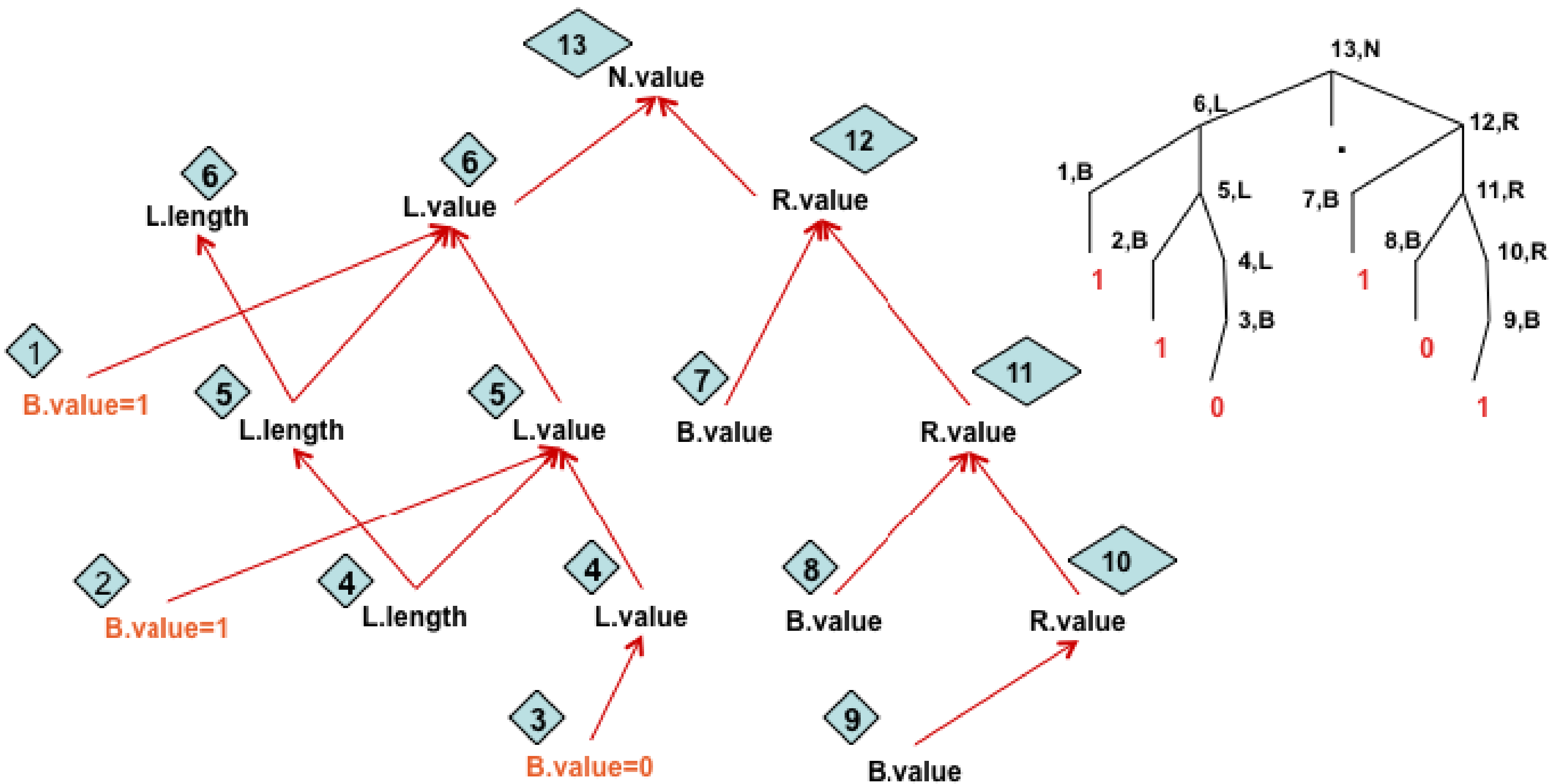
Example – Dependence Graph



Example – Attribute Evaluation



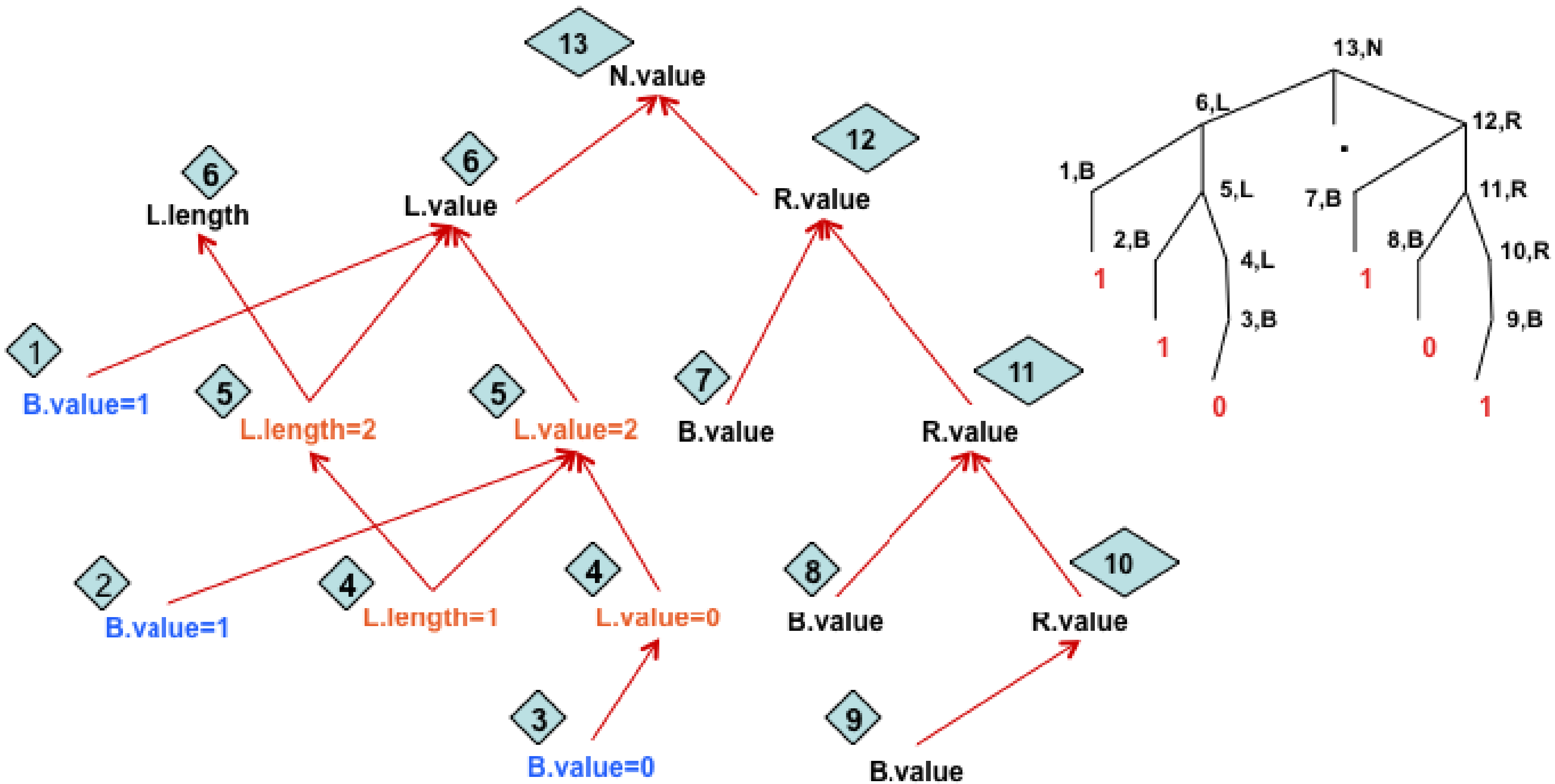
Example – Attribute Evaluation



Nodes 1,2: $B \rightarrow 1 \; \{B.value \uparrow := 1\}$

Node 3: $B \rightarrow 0 \; \{B.value \uparrow := 0\}$

Example – Attribute Evaluation

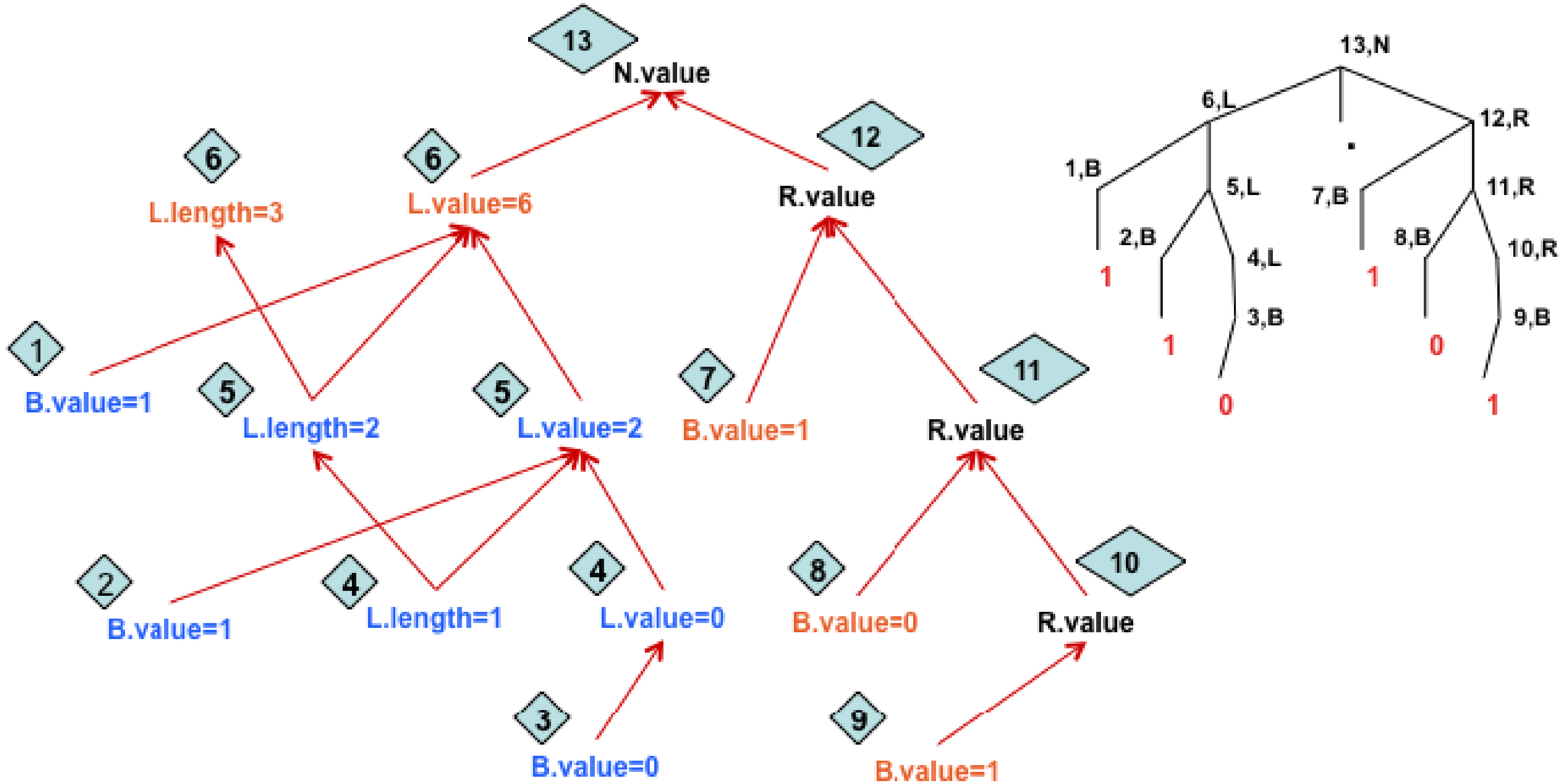


Node 4: $L \rightarrow B$ $\{L.value \uparrow := B.value \uparrow;\ L.length \uparrow := 1\}$

Node 5: $L_1 \rightarrow BL_2$ $\{L_1.length \uparrow := L_2.length \uparrow +1;$

$L_1.value \uparrow := B.value \uparrow * 2^{L_2.length\uparrow} + L_2.value \uparrow\}$

Example – Attribute Evaluation



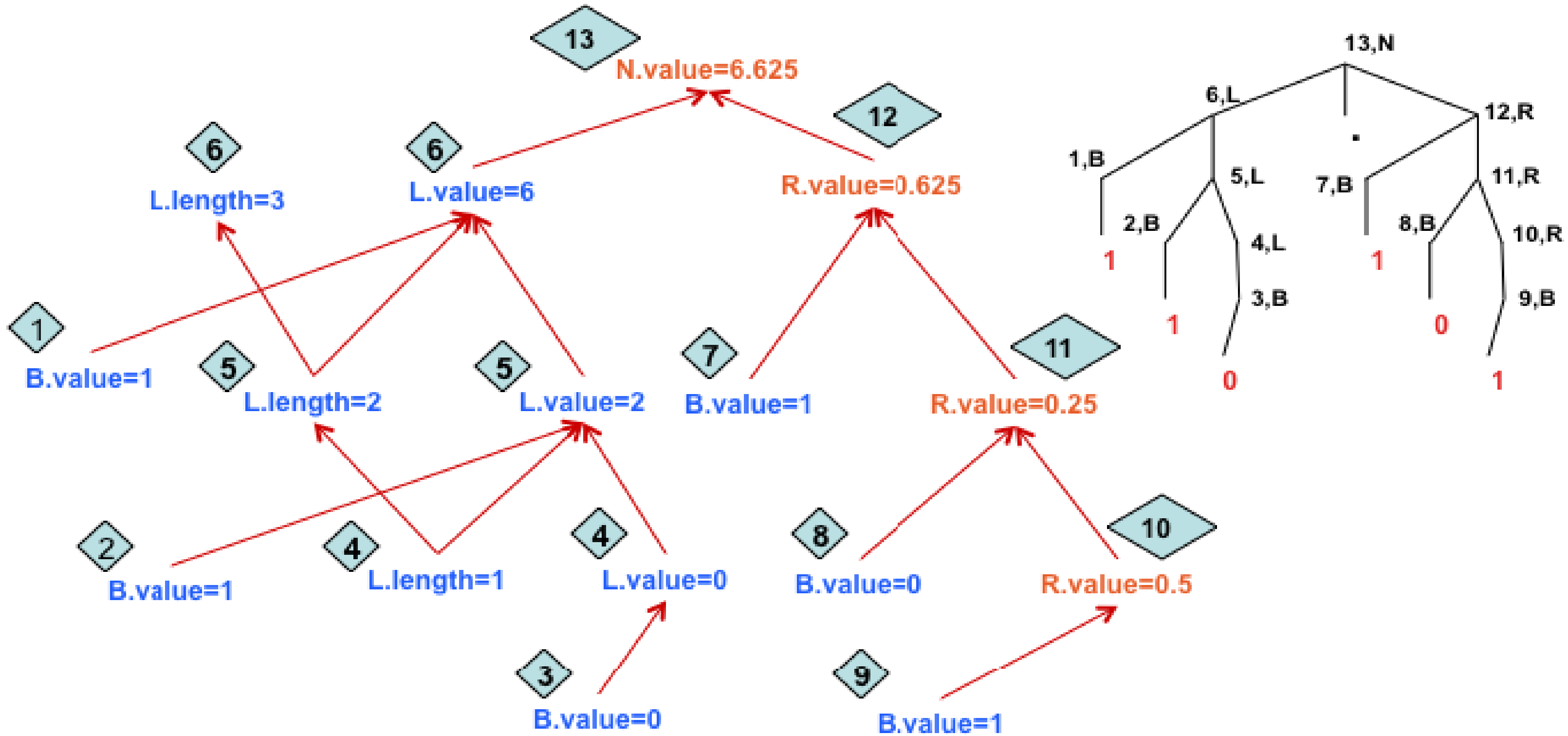
Node 6: $L_1 \to BL_2$ $\{L_1.length\uparrow := L_2.length\uparrow + 1;$

$L_1.value\uparrow := B.value\uparrow * 2^{L_2.length\uparrow} + L_2.value\uparrow\}$

Nodes 7,9: $B \to 1$ $\{B.value\uparrow := 1\}$

Node 8: $B \to 0$ $\{B.value\uparrow := 0\}$

Example – Attribute Evaluation



Node 10: $R \rightarrow B$ $\{R.value\uparrow := B.value\uparrow /2\}$

Nodes 11,12:

$R_1 \rightarrow BR_2$ $\{R_1.value\uparrow := (B.value\uparrow + R_2.value\uparrow)/2\}$

Node 13: $N \rightarrow L.R$ $\{N.value\uparrow := L.value\uparrow + R.value\uparrow\}$

Another Example

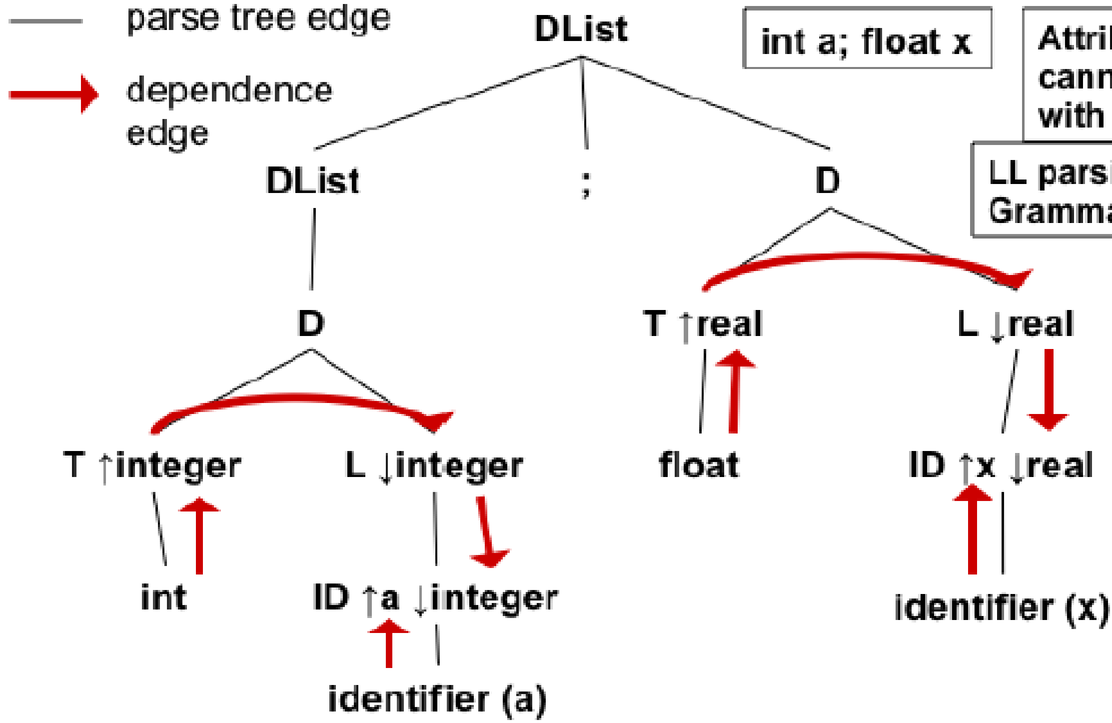
An AG for associating *type* information with names in variable declarations

$AI(L) = AI(ID) = \{type \downarrow : \{integer, real\}\}$
$AS(T) = \{type \uparrow : \{integer, real\}\}$
$AS(ID) = AS(identifier) = \{name \uparrow : string\}$

1. $DList \to D \mid DList \; ; \; D$
2. $D \to T \; L \; \{L.type \downarrow := T.type \uparrow\}$
3. $T \to int \; \{T.type \uparrow := integer\}$
4. $T \to float \; \{T.type \uparrow := real\}$
5. $L \to ID \; \{ID.type \downarrow := L.type \downarrow\}$
6. $L_1 \to L_2 \; , \; ID \; \{L_2.type \downarrow := L_1.type \downarrow; ID.type \downarrow := L_1.type \downarrow\}$
7. $ID \to identifier \; \{ID.name \uparrow := identifier.name \uparrow\}$

Example: *int* a,b,c; *float* x,y
a,b, and c are tagged with type *integer*
x and y are tagged with type *real*

Another Example – Attribute Evaluation



int a; float x

Attribute evaluation cannot be done along with LR-parsing

LL parsing is not possible. Grammar is left-recursive

1. $DList \rightarrow D \mid DList\ ;\ D$ **2.** $D \rightarrow T\ L\ \{L.type\downarrow := T.type\uparrow\}$

3. $T \rightarrow int\ \{T.type\uparrow := integer\}$ **4.** $T \rightarrow float\ \{T.type\uparrow := real\}$

5. $L \rightarrow ID\ \{ID.type\downarrow := L.type\downarrow\}$

6. $L_1 \rightarrow L_2\ ,\ ID\ \{L_2.type\downarrow := L_1.type\downarrow; ID.type\downarrow := L_1.type\downarrow\}$

7. $ID \rightarrow identifier\ \{ID.name\uparrow := identifier.name\uparrow\}$

L-Attributed and S-Attributed Grammars

An AG with only synthesized attributes is an S-attributed grammar

- Attributes of SAGs can be evaluated in any bottom-up order over a parse tree (single pass)
- Attribute evaluation can be combined with LR-parsing (YACC)

In L-attributed grammars, attribute dependencies always go from *left to right*

More precisely, each attribute must be

- Synthesized, or
- Inherited, but with the following limitations: consider a production $p: \; A \rightarrow X_1 X_2 ... X_n$. Let $X_i.a \in AI(X_i)$. $X_i.a$ may use only
  - elements of $AI(A)$
  - elements of $AI(X_k)$ or $AS(X_k)$, $k = 1, ..., i-1$ (*i.e.*, attibutes of $X_1, ..., X_{i-1}$)

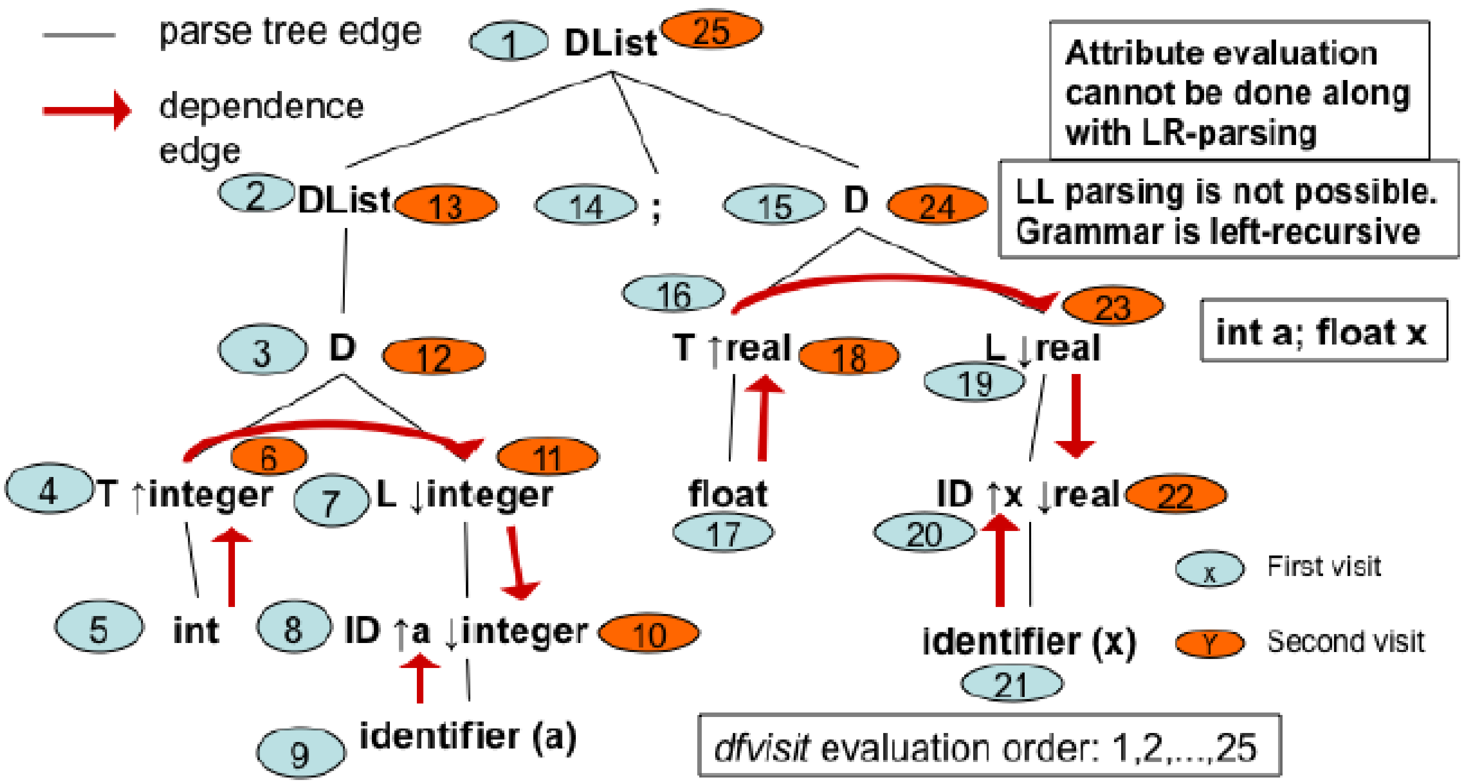
We concentrate on SAGs, and 1-pass LAGs, in which attribute evaluation can be combined with LR, LL or RD parsing

Attribute Evaluation Algorithm for LAGs

Input: A parse tree *T* with unevaluated attribute instances
Output: *T* with consistent attribute values

```
void dfvisit(n: node)
{  for each child m of n, from left to right do
     {  evaluate inherited attributes of m;
        dfvisit(m)
     };
   evaluate synthesized attributes of n
}
```

Attribute Evaluation order in Last Example



1. $DList \rightarrow D \mid DList \; ; \; D$ 2. $D \rightarrow T \; L \; \{L.type \downarrow := T.type \uparrow\}$

3. $T \rightarrow int \; \{T.type \uparrow := integer\}$ 4. $T \rightarrow float \; \{T.type \uparrow := real\}$

5. $L \rightarrow ID \; \{ID.type \downarrow := L.type \downarrow\}$

6. $L_1 \rightarrow L_2 \; , \; ID \; \{L_2.type \downarrow := L_1.type \downarrow ; ID.type \downarrow := L_1.type \downarrow\}$

7. $ID \rightarrow identifier \; \{ID.name \uparrow := identifier.name \uparrow\}$