Semantic Analysis - 3

Binary to Decimal – Solution for Modified CFG

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

Example: $110.1010 = 6 + 10/2^4 = 6 + 10/16 = 6 + 0.625 = 6.625$

$N \rightarrow X.X,\ X \rightarrow BX \mid B,\ B \rightarrow 0 \mid 1$

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

1. $N \rightarrow X_1.X_2\ \{N.value\uparrow := X_1.value\uparrow + X_2.value\uparrow / 2^{X_2.length}\}$
2. $X \rightarrow B\ \{X.value\uparrow := B.value\uparrow;\ X.length\uparrow := 1\}$
3. $X_1 \rightarrow BX_2\ \{X_1.length\uparrow := X_2.length\uparrow + 1;$
   $X_1.value\uparrow := B.value\uparrow * 2^{X_2.length\uparrow} + X_2.value\uparrow\}$
4. $B \rightarrow 0\ \{B.value\uparrow := 0\}$
5. $B \rightarrow 1\ \{B.value\uparrow := 1\}$

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
}
```

LAG – Example 1

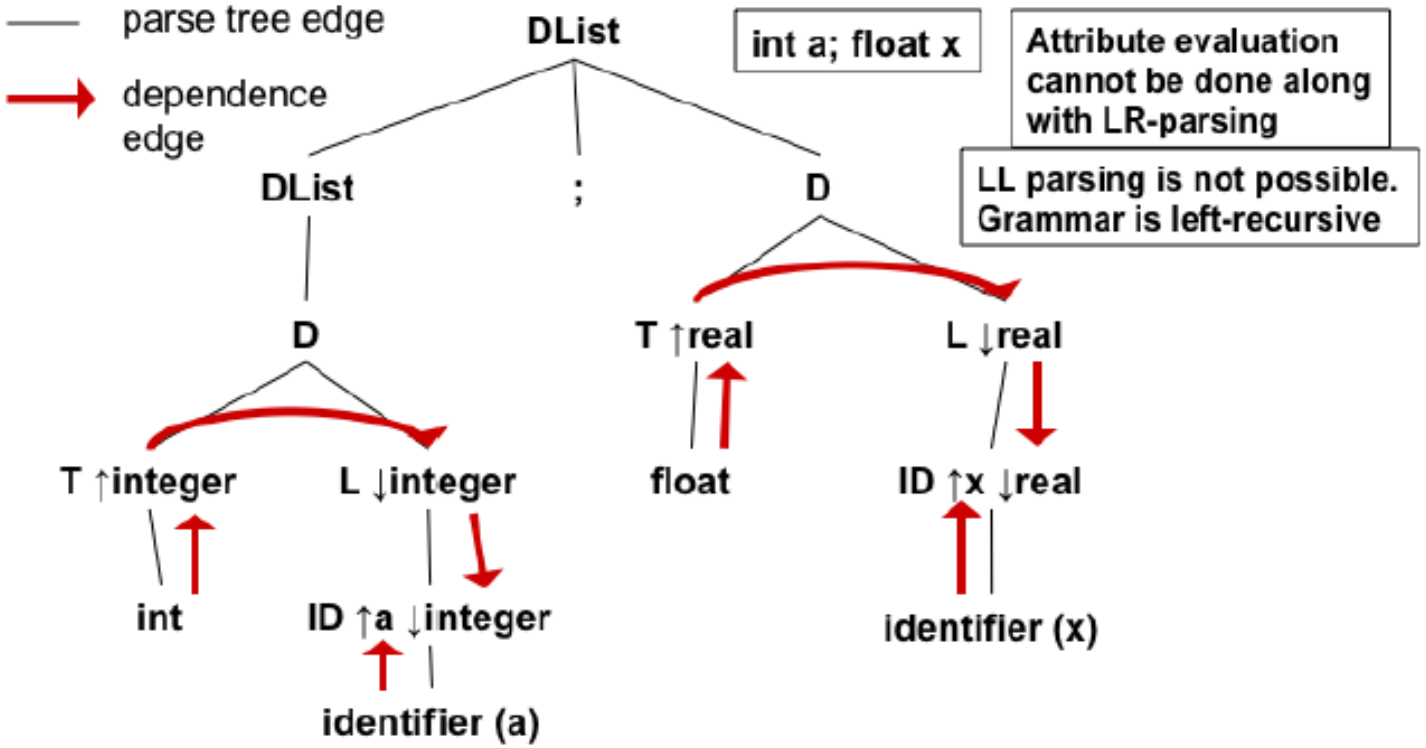
An AG for associating *type* information with names in variable declarations

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

1. $DList \rightarrow D \mid DList\ ;\ D$
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. $ID \rightarrow identifier\ \{ID.name\uparrow := identifier.name\uparrow\}$

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

LAG Example 1 – Attribute Evaluation



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

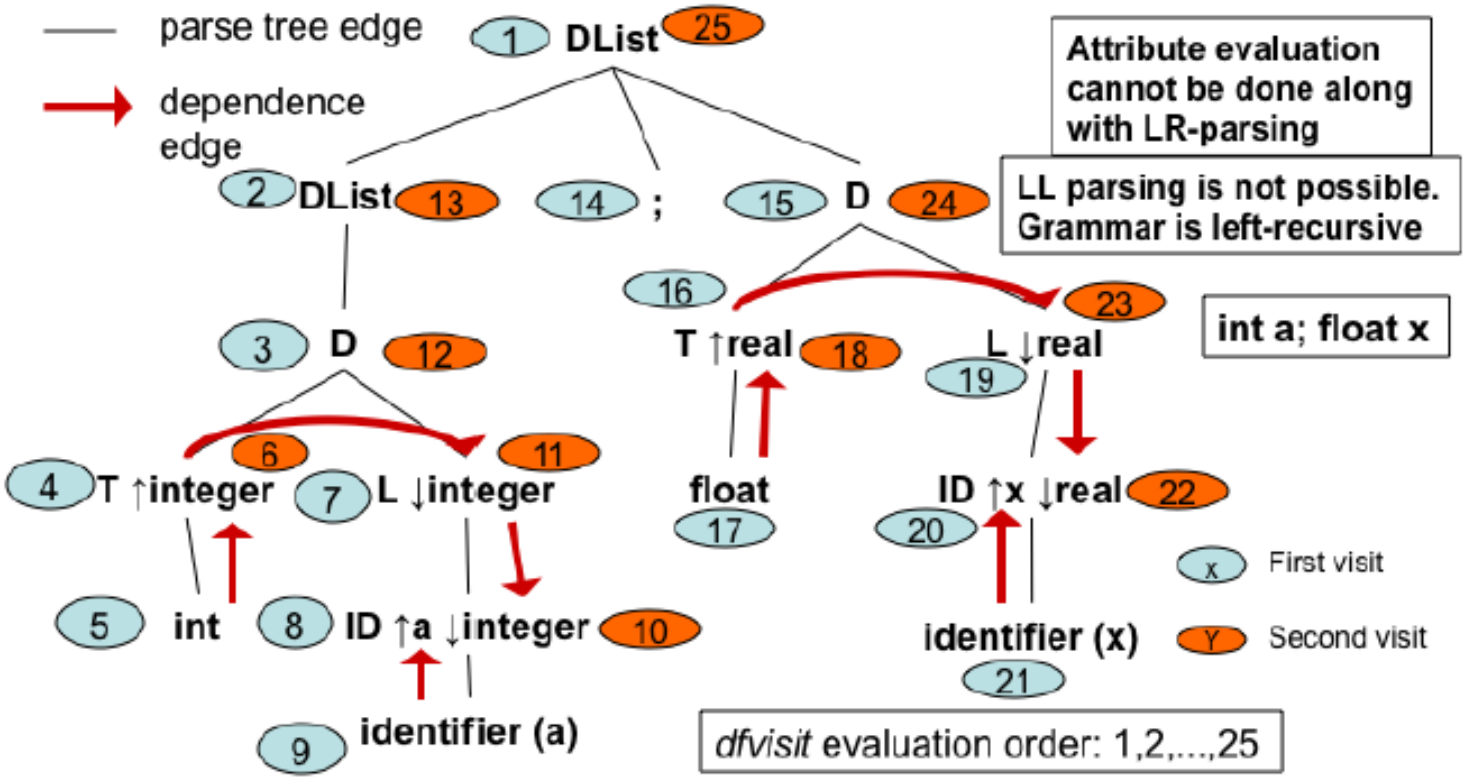
3. $T \rightarrow int\ \{T.type \uparrow := integer\}$ **4.** $T \rightarrow float\ \{T.type \uparrow := real\}$

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

LAG Example 1 – Attribute Evaluation Order



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

LAG Example 2

Let us first consider the CFG for a simple language

1. $S \longrightarrow E$
2. $E \longrightarrow E + T \mid T \mid let\ id = E\ in\ (E)$
3. $T \longrightarrow T * F \mid F$
4. $F \longrightarrow (E) \mid number \mid id$

This language permits expressions to be nested inside expressions and have scopes for the names

- *let A = 5 in ((let A = 6 in (A*7)) - A)* evaluates correctly to 37, with the scopes of the two instances of *A* being different

It requires a scoped symbol table for implementation

An abstract attribute grammar for the above language uses both inherited and synthesized attributes

Both inherited and synthesized attributes can be evaluated in one pass (from left to right) over the parse tree

Inherited attributes cannot be evaluated during LR parsing

LAG Example 2

$$S \longrightarrow E \ \{E.symtab\downarrow := \phi;\ S.val\uparrow := E.val\uparrow\}$$

$$E_1 \longrightarrow E_2 + T \ \{E_2.symtab\downarrow := E_1.symtab\downarrow;$$
$$E_1.val\uparrow := E_2.val\uparrow + T.val\uparrow;\ T.symtab\downarrow := E_1.symtab\downarrow\}$$

$$E \longrightarrow T \ \{T.symtab\downarrow := E.symtab\downarrow;\ E.val\uparrow := T.val\uparrow\}$$

$$E_1 \longrightarrow let\ id = E_2\ in\ (E_3)$$
$$\{E_1.val\uparrow := E_3.val\uparrow;\ E_2.symtab\downarrow := E_1.symtab\downarrow;$$
$$E_3.symtab\downarrow := E_1.symtab\downarrow \setminus \{id.name\uparrow \rightarrow E_2.val\uparrow\}\}$$

$$T_1 \longrightarrow T_2 * F \ \{T_1.val\uparrow := T_2.val\uparrow * F.val\uparrow;$$
$$T_2.symtab\downarrow := T1.symtab\downarrow;\ F.symtab\downarrow := T_1.symtab\downarrow\}$$

$$T \longrightarrow F \ \{T.val\uparrow := F.val\uparrow;\ F.symtab\downarrow := T.symtab\downarrow\}$$

$$F \longrightarrow (E) \ \{F.val\uparrow := E.val\uparrow;\ E.symtab\downarrow := F.symtab\downarrow\}$$

$$F \longrightarrow number \ \{F.val\uparrow := number.val\uparrow\}$$

$$F \longrightarrow id \ \{F.val\uparrow := F.symtab\downarrow [id.name\uparrow]\}$$

LAG Example 2 – Attribute Evaluation Order