Semantic Analysis

Sudakshina Dutta

IIT Goa

5^{th} April, 2022

Syntax-Directed Definition (SDD)

- A SDD is a context-free grammar with attributes and rules
  - Attributes are associated with grammar symbols and rules are associated with productions

Attribute Translation Grammar

Two types of attributes at a parse-tree node N for non-terminal A are considered

- Synthesized attributes is defined only in terms of attribute values at the children of N and at N itself. Production must have A as its head.

  — Terminals can have synthesized attributes, not inherited attributes

  — Attributes for terminals have lexical values supplied by lexical analyzer; no semantic rules for terminals

- Inherited attribute is defined only in terms of the attribute values at N's parent, N itself and N's siblings. Production must have A as a symbol in its body.

- A dependency graph depicts the order of evaluation of attributes
  - The graph has attributes as vertices
  - Edges depict order of evaluation
- Circularity is avoided
- Inherited attributes are used when the structure of the parse tree does not match with the underlying syntax tree of the source code

Order of evaluation

- A SDD with only synthesized attributes is called $S-$attributed
- Can be combined with a LR parser e.g., Bison
- For grammars having only synthesized attributes, the evaluation of attributes can proceed in bottom-up manner

  — Post-order traversal

- For grammars with both inherited and synthesized attributes, there is no even order
- The second class of SDD is called L-attributed SDD where dependency graph edges can go from left to right
- Each attribute of this SDD can be either
  - Synthesized or
  - Inherited. If the production is $A \to X_1 X_2 \cdots X_n$, then inherited attribute of X_i is computed using
    — inherited attribute of A
    — inherited or synthesized attributes associated with the symbols X_1, X_2, $\cdots$, X_{i-1}
    — inherited or synthesized attributes associated with X_i, itself (no cycles)
- It ensures left-to-right depth-first evaluation order

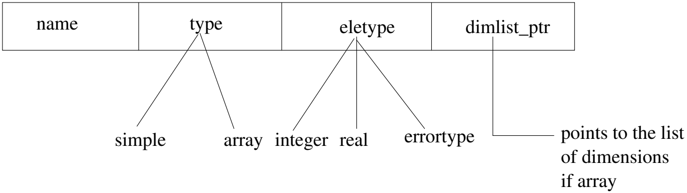
Its a TOP DOWN parsing strategy.

Attributed Translation Grammar

- Attribute translation process can be implemented by building a parse tree and then performing actions in a left-to-right depth-first order
- Generally, attribute translation process is exploited to implement two important classes of attribute definition:
  1. The underlying grammar is LR-parsable and the attribute grammar is S-attributed   Bottom Up
  2. The underlying grammar is LL-parsable and the attribute grammar is L-attributed

Identifier type information in symbol table

Identifier Type Information Record



name

type

eletype

dimlist_ptr

simple

array

integer

real

errortype

points to the list
of dimensions
if array

Example

The grammar

- $D \rightarrow TL$
- $L \rightarrow ID\_ARR | ID\_ARR, L$
- $ID\_ARR \rightarrow id | id[DIMLIST]$
- $DIMLIST \rightarrow num | num, DIMLIST$
- $T \rightarrow Int$

The grammar accepts all the strings of the form *int a* or *int b*, $a[10]$ or *int* $b[50]$, $a[10, 20, 30]$, etc.

- $T \rightarrow Int$ $\{$ $T.type(syn) = integer;$ $\}$
- $D \rightarrow TL$ $\{$ $L.type(inh) = T.type(syn);$ $\}$

Go thru this & below slides when Time is available.
For ENDSEM prep, Its comp.

- $L_1 \to ID\_ARR, L_2$
  $\{ID\_ARR.type(inh) = L_1.type(inh)\}ID\_ARR,$
  $\{\ L_2.type(inh) = L_1.type(inh);\}L_2$
- $L \to \{ID\_ARR.type(inh) = L.type(inh)\}ID\_ARR$
- $ID\_ARR \to id$
  $\{search\_symtab(id.name(syn), found);$
  $if(found)error('declared');$
  $else\{typerec *\ \ t; t \to type = simple;$
  $t \to eletype = ID\_ARR.type(inh);$
  $insert\_symtab(id.name(syn), t);\}$

- $ID\_ARR \rightarrow id[DIMLIST]$
  { search in the symbol table; $if(found) \cdots$;
  $else\{typerec * t; t \rightarrow type = array;$
  $t \rightarrow eletype = ID\_ARR.type(inh);$
  $t \rightarrow dimlist\_ptr = DIMLIST.ptr(syn);$
  $insert\_symtab(id.name(syn), t)\}$
  }
- $DIMLIST \rightarrow num$
  $\{ DIMLIST.ptr(syn) = makelist(num.value(syn))\}$
- $DIMLIST_1 \rightarrow num, DIMLIST_2$
  $\{DIMLIST_1.ptr(syn) =$
  $append(num.value(syn), DIMLIST_2.ptr(syn))\}$