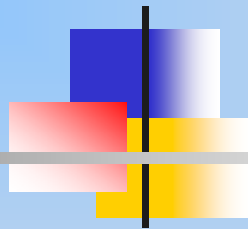
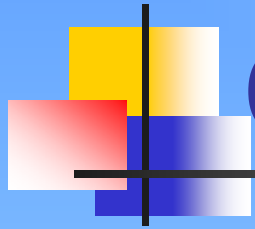


CSE302: Compiler Design



Instructor: Dr. Liang Cheng
Department of Computer Science and Engineering
P.C. Rossin College of Engineering & Applied Science
Lehigh University

March 29, 2007

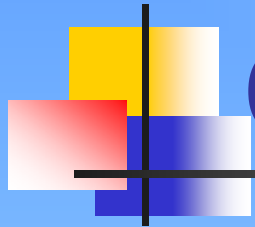


Outline

- Recap
 - Yacc
- Syntax-directed translation (Chapter 5)
- Summary and homework



- Take a specification file (grammar) and produce an output file for the parser
 - Input: <filename>.y
 - {definitions}
 - %%
 - {productions/rules}
 - %%
 - {auxiliary routines}
 - Output: y.tab.c
 - LALR parser



Outline

- Recap
- **Syntax-directed translation (Chapter 5)**
- Summary and homework



Syntax-Directed Techniques

- **Syntax-directed definition**
 - Attach a **semantic rule** to each production
- **Syntax-directed translation**
 - Add **program fragment(s)** to some production(s)
- **Applications of SDT**
 - Compute the values of the attributes associated with the symbols in the productions
 - Type checking
 - Generate side effects
 - Code generation, print results, modify symbol table, ...



Inherited & Synthesized Attribute

- Let $X_0 \rightarrow X_1 \dots X_n$ be a production
 - If the computing rule of X_0 's attribute is of the form $A(X_0) = f(A(X_1), \dots, A(X_n))$
 - **Synthesized attribute**
 - If the computing rule of X_j 's attribute is of the form $A(X_j) = f(A(X_0), \dots, A(X_i), \dots)$
 - **Inherited attribute**
 - Terminals have **intrinsic attributes**
 - Lexical values supplied by the lexical analyzer

Definition of Attribute Grammar (1/23)

- An **attribute grammar** is a BNF grammar with additions:
 - For any grammar symbol X : a set $A(X)$ of **attribute** values
 - Each production in the grammar has a set of **semantic rules** that define or compute certain attributes of the nonterminals in the production
 - Each production in the grammar has a (possibly empty) set of **predicates** to check for attribute consistency

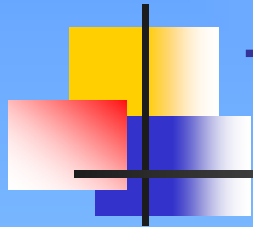
- A sentence derivation

Based on BNF

A parse tree

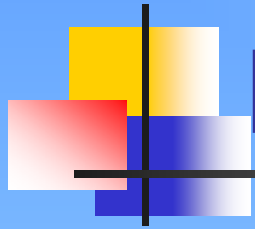
Based on an attribute grammar

A fully attributed parse tree
or an annotated parse tree



Tree Traversals

- For synthesized attributes
 - Perform bottom-up tree traversal for attribute evaluation
 - An SDD is **S-attributed** if every attribute is synthesized
- For SDD's with both inherited and synthesized attributes
 - Dependency graphs
 - No guarantee that there is even one order
 - Circular dependency
 - Production $A \rightarrow B$
 - Semantic rules
 - $A.s = B.i$
 - $B.i = A.s + 1$



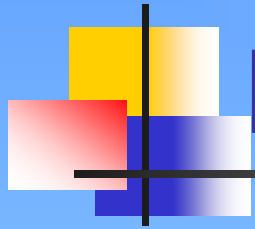
Dependency Graphs

- Determine how attributes can be evaluated in **parse trees**
 - For each symbol X , the dependency graph has a node for each attribute associated with X
 - An edge from node A to node B means that the attribute of A is needed to compute the attribute of B
 - How to diff syn. attributes from inh. attributes



A Type Checking Example Using Syntax-Directed Definition (1/23)

- A BNF grammar
 - $\langle \text{assign} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expr} \rangle$
 - $\langle \text{expr} \rangle \rightarrow \langle \text{var} \rangle + \langle \text{var} \rangle$
 - $\langle \text{var} \rangle \rightarrow A \mid B \mid C$
- An attribute grammar
 1. Syntax production: $\langle \text{assign} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expr} \rangle$
 - Semantic rule: $\langle \text{expr} \rangle.\text{expected_type} \leftarrow \langle \text{var} \rangle.\text{actual_type}$
 2. Syntax production: $\langle \text{expr} \rangle \rightarrow \langle \text{var} \rangle + \langle \text{var} \rangle$
 - Semantic rule: $\langle \text{expr} \rangle.\text{actual_type} \leftarrow$
if($\langle \text{var} \rangle[2].\text{actual_type} == \text{int}$) and
($\langle \text{var} \rangle[3].\text{actual_type} == \text{int}$)
then int
else real
endif
 - Predicate: $\langle \text{expr} \rangle.\text{actual_type} == \langle \text{expr} \rangle.\text{expected_type}$
 3. Syntax production: $\langle \text{var} \rangle \rightarrow A \mid B \mid C$
 - Semantic rule: $\langle \text{var} \rangle.\text{actual_type} \leftarrow \text{lookup}(\langle \text{var} \rangle.\text{string})$



L-Attributed SDD's

- An SDD is **L-attributed** if in all of its dependency graphs the edges only go from left to right but not from right to left
 - No circular dependency
 - Guarantee that there is an evaluation order



Computing Attribute Value (1/23)

- Let $X_0 \rightarrow X_1 \dots X_n$ be a production
 - If the computing rule of X_0 's attribute is of the form $A(X_0) = f(A(X_1), \dots, A(X_n))$
 - **Synthesized attribute**
 - If the computing rule of X_j 's attribute is of the form $A(X_j) = f(A(X_0), \dots, A(X_i), \dots, A(X_{j-1}))$, for $i <= j <= n$
 - **Inherited attribute**
 - **Or** $A(X_j) = f(A(X_0), \dots, A(X_i), \dots, A(X_{j-1}), A(X_j))$
 - Inherited or synthesized attributes associated with X_j itself can be but without cycles in the dependency graphs



SDD Examples

- Production

$A \rightarrow B C$

Semantic rules

$A.i = B.l$

$B.m = F(C.x, A.j)$

- Is this an S-Attributed or L-Attributed SDD?
- Another SDD example
- More SDD examples



Semantic Rules with Side Effects

- Note that SDD is used for specifications
 - Semantic rules can contain actions that generate side effects
 - Production Semantic rules
 - $D \rightarrow T L$ $L.inh = T.type$
 - $T \rightarrow \mathbf{int}$ $T.type = \mathbf{int}$
 - $T \rightarrow \mathbf{float}$ $T.type = \mathbf{float}$
 - $L \rightarrow L_1, \mathbf{id}$ $L_1.inh = L.inh$
 $\mathbf{addType(id.entry, L.inh)}$
 - $L \rightarrow \mathbf{id}$ $\mathbf{addType(id.entry, L.inh)}$
- More SDD's with side effects



Outline

- Recap
- Syntax-directed translation
- **Summary and homework**



Final Exam Reminder

- THURSDAY, MAY 03, 2007,
08:00-11:00AM



Homework (Due on 04/02)

- 10.1. (a) Exercise 5.2.4 (page 317);
(b) Exercise 5.2.5 (Page 317).