

Lecture Outline

1. Syntax-directed translation
2. Variants of syntax trees
3. Three-address code
4. Semantic analysis

1. Syntax-Directed Translation

- Postfix translation schemes
- Translation schemes with actions inside productions
- Producing ASTs with top-down parsing
 - Here is the L-attributed SDD based on an LL(1) grammar for translating arithmetic expressions into ASTs from Lecture 12.

```
E → T A      { E.node = A.s;
                A.i = T.node; }

A → T + T A1  { A1.i = Node('+', A.i, T.node);
                A.s = A1.s; }

A → T μ      { A.s = A.i; }

T → ( E )     { T.node = E.node; }

T → id       { T.node = Leaf(id, id.entry); }
```

2. Variants of Syntax Trees

- Abstract syntax trees
- Directed acyclic graphs
 - Algorithm 6.3: Value-number method for constructing a DAG (p. 361)

3. Three-Address Code

- Three-address instructions
- Representations for three-address code
 - Records
 - Quadruples
 - Triples
- Static single-assignment form

4. Semantic Analysis

- Uses made of semantic information for a variable x :
 - What kind of value is stored in x ?
 - How big is x ?
 - Who is responsible for allocating space for x ?
 - Who is responsible for initializing x ?
 - How long must the value of x be kept?
 - If x is a procedure, what kinds of arguments does it take and what kind of return value does it have?
- Storage layout for local names

5. Practice Problems

1. Construct a DAG for the expression

$((x + y) - ((x + y) * (x - y))) + ((x + y) * (x - y))$

2. Translate the following assignments into (a) syntax trees, (b) quadruples, (c) triples, (d) three-address code:

a. $x = a + -(b+c)$

b. $x[i] = y[i] + z[i]$

c. $x = f(y+1) + 2$

6. Reading

- ALSU, Sections 6.1-6.3

aho@cs.columbia.edu