Attribute Grammars

Specifying Syntax of Programming Languages

- Context-Free Grammars as Language Generator

Context-Free Grammars

- Context-Free Grammar G = (T,N,S,P)
  - T =A finite set of terminal symbols (alphabetic set)
  - N = A finite set of non-terminal symbols
  - S = A starting symbol (A special non-terminal)
  - P = A set of production rule: **lhs** ✍ **rhs**

LHS ✍ **RHS**

Single non-terminal — Can be any string

Semantics of Programming Languages

- **Semantics**
  - **Static semantics**
    - Meanings that can be determined statically (at compile time)
  - **Dynamic semantics**
    - Meanings that can be determined dynamically.

Example: Static Semantics

- Context-free but cumbersome
  - Type checking
- Noncontext-free
  - Variables must be declared before they are used.

Specifying Semantics of Context-Free Languages

Describing Semantics Formally

- CFGs cannot describe all of the static semantics of programming languages.
- Need additions to CFGs to carry some semantic info. along through parse trees.
  - Attribute Grammars (AG)

Attribute Grammars

Attribute Grammars (AG)

- A specification technique to formally define the semantics (semantic properties) of a programming language.
- An extension of CFG:
  - Attributes
  - Attribute evaluation rules

Attribute Grammars

- Attribute Grammar AG = (G, A, R) where
  - G = A CFG G= (T,N,S,P)
  - A = A set of attributes for each grammar symbol
  - R = A set of attribute evaluation rules

Example: Attribute Grammar

AG = (G, A, R)

G:	**A:**	**R:**
E ✍ E + T	val	E.val = E.val + T.val
E ✍ T		E.val = T.val
T ✍ T * F		T.val = T.val * F.val
T ✍ F		T.val = F.val
F ✍ num		F.val = num.val

Attribute Grammar

- An attribute grammar (AG) is a CFG G = (T, N, S, P) with the following additions:
  - Each grammar symbol x has
    - A set A(x) of attributes.
  - Each rule has
    - A set of semantic functions that define certain attributes of the non-terminals in the rule.
    - A (possibly empty) set of predicates to check for attribute consistency.

Attributes

- Each grammar symbol X has a set A(X) of attributes.

Typical Examples of Attributes

- The data type of a variable
- The value of an expression
- The location of a variable in memory
- The object code of a procedure
- The number of significant digits in a number

Static and Dynamic Attributes

- Static attributes
- Dynamic attributes

Types of Attributes

- S(X): **Synthesized** attributes
  - To pass semantic info up a parse tree.
- I(X): **Inherited** attributes
  - To pass semantic info down a parse tree.

Types of Attributes

- **Intrinsic** attributes
  - Synthesized attributes of leaf nodes in a parse tree
  - Whose values are determined outside the parse tree and given.
  - Initially, there are intrinsic attributes on the leaves

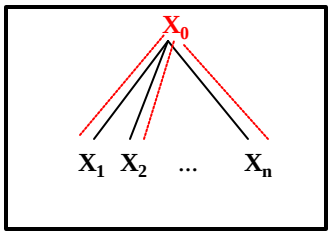
Attribute Evaluation Rules

- A production rule
  - $X_0 \Leftarrow X_1 X_2 \ldots X_n$
- An attribute a for X_i
  - $X_i .a$
- An attribute evaluation rule
  - $X_i .a = f(X_0 X_1 X_2 \ldots X_n)$

Attribute Evaluation Rules - Semantic Functions

- Let X0 ? X1 ... Xn be a rule.
- S(X0) = The synthesized attributes of X0.
- The synthesized attributes of X0 are computed by a semantic function of the form:
  - S(X0) = f(A(X1), ... A(Xn))
  - Depends only on the attributes of the node' s children nodes!

Synthesized Attributes



Attribute Evaluation Rules - Semantic Functions

- Let X0 ? X1 ... Xn be a rule.
- I(Xj) = The inherited attributes of Xj.where 1<=j<=n.
- The inherited attributes of Xj are computed by a semantic function of the form:
  - I(Xj) = f(A(X0), ... , A(Xn))
  - Depends on the attributes of the node' s parent and sibling nodes!

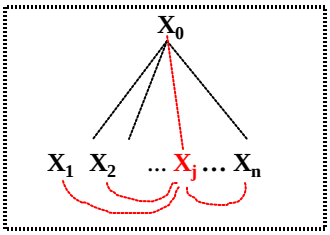
Example: Attribute Grammar

AG = (G, A, R)

num1 + num2 * num3

G:	**A:**	**R:**
E ✍ E + T	val	E.val = E.val + T.val
E ✍ T		E.val = T.val
T ✍ T * F		T.val = T.val * F.val
T ✍ F		T.val = F.val
F ✍ num		F.val = num.val

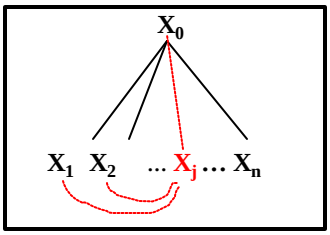
Inherited Attributes



L-Inherited Attributes

- The inherited attributes of Xj are computed by a semantic function of the form:
  - I(Xj) = f(A(X0), ... , A(Xj-1))
  - Depends on the attributes of the node' s parent and left sibling nodes!
  - **L-inherited** attributes.

L-Inherited Attributes



Example: Attribute Grammar

AG = (G, A, R)

G:
Decl ? **int** IdList **;**

Decl ? **real** IdList **;**

Decl ? **char** IdList **;**

IdList ? IdList **,** Id

IdList ? Id

A:
type

R:
{IdList.type = **int**.type**}**

{IdList.type = **real**.type**}**

{IdList.type = **char**.type}

{Id.type = IdList.type
IdList1.type = IdList.type}

{Id.type = IdList.type}

int Id1, Id2, Id3 **;**

Example: Attribute Grammar

AG = (G, A, R)

G:
Stmt ? Id := Expr **;**

Expr ? Expr + Term

Expr ? Term

Term ? num

R:
{Id.val = Expr.val**}**

{Expr.val = Expr.val + Term.val **}**

{Expr.val = Term.val}

{Term.val = num.val}

A:
val

Id = num1 + num2 ;

Predicate Functions

- A predicate function has the form of a Boolean expression on the attribute set {A(X0), …, A(Xn) }.
- Derivations are allowed:
  - Every predicate associated with every non-terminal is true.

Example: Ada Procedures

- The name on the end of an Ada procedure must match the procedure' s name.

CFG:
<proc_def> ? **procedure** <proc_name> <proc_body>
end <proc_name>

Example: Attribute Grammar

- The name on the end of an Ada procedure must match the procedure' s name.

AG:
<proc_def> ? **procedure** <proc_name> <proc_body>
end <proc_name>

Attribute:
string

Semantic function:
<proc_name>.string = <proc_name>.string

Example: Attribute Grammar

- The name on the end of an Ada procedure must match the procedure' s name.

AG:
<proc_def> ? **procedure** <proc_name>[1] <proc_body> **end** <proc_name>[2]
Attribute:
string

Semantic function:
<proc_name>[1].string = <proc_name>[2].string

Example: Assignment Statements

- A simple assignment statement.

CFG:
<assign> ? <var> := <expr>
<expr> ? <var> + <var>
| <var>
<var> ? A | B | C

Example: Type Rules

- The type rules of a simple assignment statement:
  - The variables can be one of two types: int or real.
  - The type of the expression is that of its operands if the same. Otherwise real.
  - The type of LHS of an assignment must match the type of RHS.

A := A + B

A: real
B: int

Example: Attributes

- Attributes:
  - actual_type = A synthesized attribute for <var> and <expr>
  - expected_type = An inherited attribute for <expr>

Example: Attribute Grammar

AG:
Syntax rule:
<assign> ? <var> := <expr>

Semantic rule:

<expr>.expected_type ? <var>.actual_type

Example: Attribute Grammar

AG:
Syntax rule:
<expr> ? <var>[1] + <var>[2]

Semantic rule:

<expr>.actual_type ? if
(<var>[1].actual_type = int) and
(<var>[2].actual_type = int) then int
else real

Predicate:
<expr>.actual_type = <expr>.expected_type

Example: Attribute Grammar

AG:
Syntax rule:
<expr> ? <var>

Semantic rule:

<expr>.actual_type ? <var>.actual_type

Predicate:

<expr>.actual_type = <expr>.expected_type

Example: Attribute Grammar

AG:
Syntax rule:
<var> ? A | B | C

Semantic rule:

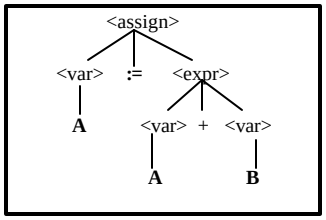
<var>.actual_type ? look-up(<var>.string)

Example: Attribute Grammar

CFG:
<assign> ? <var> := <expr>
<expr> ? <var> + <var>
| <var>
<var> ? A | B | C

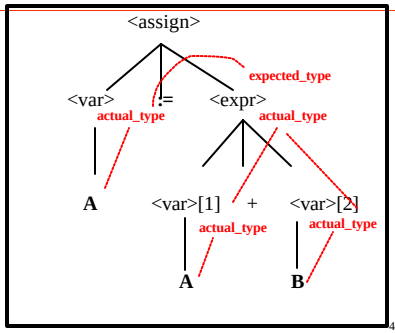
A := A + B

A Parse Tree



Example: Flow of Attributes

A := A + B

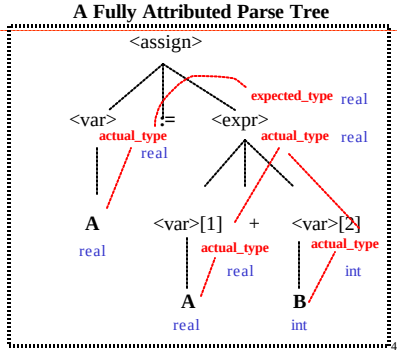


Example: Computing Attributes

A Fully Attributed Parse Tree

A := A + B

A: real
B: int



More Example: Attribute Grammar

- **Example 6.1 (p. 262)**

More Example: Attribute Grammar

- **Example 6.2 (p. 264)**

More Example: Attribute Grammar

- **Example 6.3 (p. 266)**

More Example: Attribute Grammar

- **Example 6.4 (p. 266)**

Computing Attribute Values

- If all attributes were inherited, the tree could be decorated in top-down order.
- If all attributes were synthesized, the tree could be decorated in bottom-up order.
- In many cases, both kinds of attributes are used, and it is some combination of top-down and bottom-up that must be used.

Computing Attribute Values

- Attributes as Parameters and Returned Values
  - Pass the inherited attribute values as parameters to recursive calls on children!
  - Receive synthesized attribute values as returned values of recursive calls on children!

Important Subclasses of AGs

- **S-attributed AG**
- **L-attributed AG**

S-Attributed AG (SAG)

- All attributes are synthesized attributes.
- Can be evaluated at the same time during parsing!!!

Example: SAG

AG = (G, A, R)

G:
E ✍ E + T
E ✍ T
T ✍ T * F
T ✍ F
F ✍ num

A:
val

R:
E.val = E.val + T.val
E.val = T.val
T.val = T.val * F.val
T.val = F.val
F.val = num.val

L-Attributed AG (LAG)

- All attributes are either synthesized attributes or restricted forms of inherited attributes (L-inherited).

Example: LAG

AG = (G, A, R)

G:
Decl ? **int** IdList **;**
Decl ? **real** IdList **;**
Decl ? **char** IdList **;**
IdList ? IdList **,** Id
IdList ? Id

R:
{IdList.type = **int**.type**}**
{IdList.type = **real**.type**}**
{IdList.type = **char**.type}
{Id.type = IdList.type
IdList1.type = IdList.type}
{Id.type = IdList.type}

A:
type

Example: Non-LAG

AG = (G, A, R)

G:
Stmt ? Id := Expr **;**
Expr ? Expr + Term
Expr ? Term
Term ? num

R:
{Id.val = Expr.val**}**
{Expr.val = Expr.val + Term.val **}**
{Expr.val = Term.val}
{Term.val = num.val}

A:
val

More Example: Computing Attribute Values

- **Example 6.11 (p. 277)**

More Example: Computing Attribute Values

- **Example 6.12 (p. 278)**

More Example: Computing Attribute Values

- **Example 6.13 (p. 282)**

More Example: Computing Attribute Values

- **Example 6.14 (p. 283)**

Attribute Grammars and Yacc

Computing Attributes During Parsing

- If an attribute grammar has only synthesized attributes, then
  - They can be evaluated at the same time that the parse tree is generated.

S-Attribute Grammars and Yacc

- Each time yacc determines a production rule to reduce,
  - Its semantic action is executed to apply semantics to the parse tree.

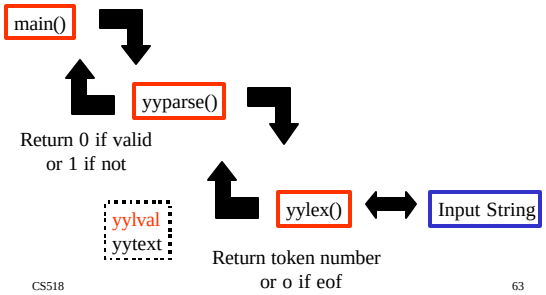
S-Attributes in Yacc

- Attributes
  - yylval
  - %union

S-Attributes in Yacc

- Attributes
  - $$ = The attribute value associated with the LHS of the production rule.
  - $1 = The attribute value associated with the first symbol on the RHS of the production rule.
  - $2 = The attribute value associated with the second symbol on the RHS of the production rule.
  - …

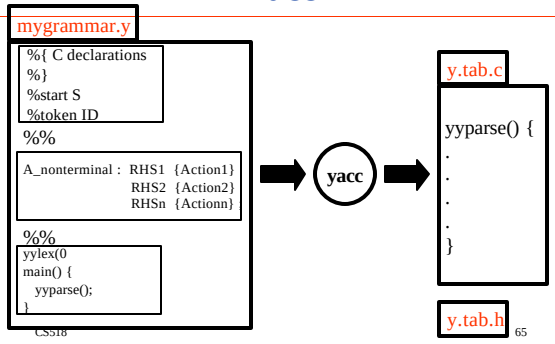
Flow of Control Using Lex and Yacc



S-Attribute Evaluation Rules in Yacc

- Attribute evaluation rules
  - { semantic action}

Attribute Evaluation Rules in Yacc



Applications of Attribute Grammars

- Type Checker
- Interpreter
- Compiler
- Abstract Syntax Tree Builder

Type Checker

- Attributes are types.

Interpreter

- Attributes are values.

Compiler

- Attributes are codes.

AST Builder

- Concrete parse tree vs Abstract syntax tree
- Attributes are abstract syntax trees.

Building Abstract Syntax Tree

- Abstract Syntax Tree Builder

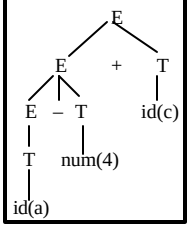
Example: AST Builder

G:

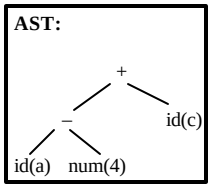
E ✍ E + T
E ✍ E - T
E ✍ T
T ✍ (E)
T ✍ id
T ✍ num

a – 4 + c

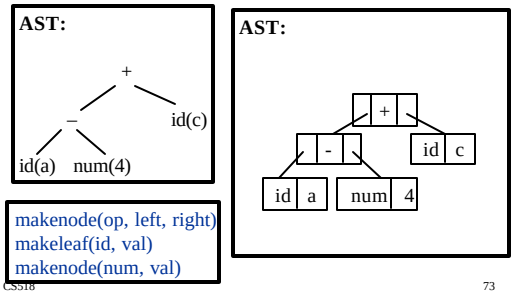
Parse tree:



AST:



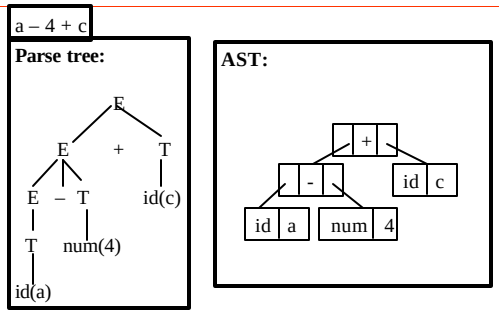
Abstract Syntax Tree



AG for Building AST

G:	R:
E ✍ E + T	{E.ptr = makenode(+, E1.ptr, T.ptr)}
E ✍ E - T	{E.ptr = makenode(-, E1.ptr, T.ptr)}
E ✍ T	{E.ptr = T.ptr}
T ✍ (E)	{T.ptr = E.ptr}
T ✍ id	{T.ptr = makenode(id, id.val)}
T ✍ num	{T.ptr = makenode(num, num.val)}

Building AST



AST Structure for TINY

- See **3.7.2** (p. 134)

Example: AST Building for TINY

- Sample program
  - **Figure 3.8 (p. 137)**
- AST
  - **Figure 3.9 (p. 138)**
- PrintTree
  - **Figure 4.10 (p. 182)**