Attribute Grammars

- Definitions: synthesized, inherited, dependence graph
- Example: syntax-directed translation
- S-attributed grammars
- L-attributed grammars
- Bottom Up evaluation of inherited attributes
- Top Down translation

Attribute Grammars

- **Attributes**: properties associated with nonterminal symbols of a context free grammar
- **E.G., Binary numbers**
  1. $B \rightarrow 0 \quad val(B) = 0$
  2. $B \rightarrow 1 \quad val(B) = 1$
  3. $L \rightarrow B \quad val(L) = val(B); \ len(L) = 1$
  4. $L \rightarrow L_1 \ B \quad val(L) = 2 \cdot val(L_1) + val(B)$
     \[ len(L) = len(L_1) + 1 \]
  5. $N \rightarrow L \quad val(N) = val(L); \ len(N) = len(L)$
Parse Tree of $101_2$

Represent dependences between semantic functions of attributes

Evaluate(Decorate) Parse Tree

Dependence Graph: acyclic, evaluated in topsort order, contains attribute instances

Initial values
Evaluate Parse Tree

Evaluate val

Evaluate len;
Full evaluation yields val(N) = 5; len(N) = 3.
Classifications

• **Inherited attributes:**
  – Values based on attributes of parent (LHS nonterminal) or siblings (nonterminals on RHS of same production)

• **Synthesized attributes:**
  – Values based on attributes of descendents (child nonterminals in same production)

Classifications

• Local context: always within focus of a single production
  – Dependence edges go only one level in parse tree
• Terminals can be associated with values returned by the scanner
• Distinguished nonterminal cannot have inherited attributes
**Example - Identifiers**

Identifiers with no letters repeated (e.g., moon - illegal, money - legal)

\[ D \rightarrow I \quad \text{str}(I) = \{\}; \text{val}(D) = \text{val}(I); \]

\[ \text{accept, if } \text{val}(D) \neq \text{error} \]

\[ I \rightarrow L \quad I_1 \quad \text{str}(L) = \text{str}(I); \text{str}(I_1) = \text{val}(L); \]

\[ \text{val}(I) = \text{val}(I_1) \]

\[ I \rightarrow L \quad \text{str}(L) = \text{str}(I); \text{val}(I) = \text{val}(L) \]

\[ L \rightarrow a \mid b \mid \ldots \mid z \quad \text{val}(L) = \text{concatenation of } \text{val} \]

(returned by scanner to \( \text{str}(L) \), if this character is not a repeated letter, else \( \text{error} \).

(note: any comparison to \( \text{error} \) returns \( \text{error} \).)

**Inherited Attributes**

\[ D \rightarrow I \quad \text{str}(I) = \{\}; \text{val}(D) = \text{val}(I); \]

\[ \text{accept, if } \text{val}(D) \neq \text{error} \]

\[ I \rightarrow L \quad I_1 \quad \text{str}(L) = \text{str}(I); \text{str}(I_1) = \text{val}(L); \]

\[ \text{val}(I) = \text{val}(I_1) \]

\[ I \rightarrow L \quad \text{str}(L) = \text{str}(I); \text{val}(I) = \text{val}(L) \]

\[ L \rightarrow a \mid b \mid \ldots \mid z \quad \text{val}(L) = \text{concatenation of } \text{val} \]

(returned by scanner to \( \text{str}(L) \), if this character is not a repeated letter, else \( \text{error} \).

(note: any comparison to \( \text{error} \) returns \( \text{error} \).)
Synthesized Attributes

\[
\begin{align*}
\text{D} & \rightarrow \text{I} & \text{str}(\text{I}) = \{\}; \quad \boxed{\text{val}(\text{D}) = \text{val}(\text{I})}; \\
& \quad \text{accept, if } \text{val}(\text{D}) \neq \text{error} \\
\text{I} & \rightarrow \text{L} \; \text{I}_1 & \text{str}(\text{L}) = \text{str}(\text{I}); \; \text{str}(\text{I}_1) = \text{val}(\text{L}); \\
& \quad \boxed{\text{val}(\text{I}) = \text{val}(\text{I}_1)} \\
\text{I} & \rightarrow \text{L} & \text{str}(\text{L}) = \text{str}(\text{I}); \quad \boxed{\text{val}(\text{I}) = \text{val}(\text{L})} \\
\text{L} & \rightarrow a \mid b \mid \ldots \mid z & \boxed{\text{val}(\text{L})} = \text{concatenation of } \text{val} \\
& \quad \text{returned by scanner to } \text{str}(\text{L}), \text{if this character is not a repeated letter, else error.} \\
& \quad \text{(note: any comparison to } \text{error} \text{ returns } \text{error.)}
\end{align*}
\]

Parse Tree of abc

```
D  val
  I  
    str  val
      str  val
        str  val
          str  val
            str  val
              str  val
                str  val
                  str  val
                    str  val
                      str  val
                        str  val
                          str  val
                            str  val
                              str  val
                                str  val
                                  str  val
                                    str  val
                                      str  val
                                        str  val
                                          str  val
                                            str  val
                                              str  val
                                                str  val
                                                  str  val
                                                    str  val
                                                      str  val
                                                        str  val
                                                          str  val
                                                            str  val
                                                              str  val
                                                                str  val
                                                                  str  val
                                                                    str  val
                                                                      str  val
                                                                        str  val
                                                                          str  val
                                                                            str  val
                                                                              str  val
                                                                                str  val
                                                                                  str  val
                                                                                    str  val
                                                                                      str  val
                                                                                     str  val
```

Attribute Grammars © BGR, Fall05
**Decorated Parse Tree**

**Compiler Example**

$symb$ is the symbol table gathered from the declarations and checked in the statements.
Syntax-directed Translation

• Idea: to use attribute grammars to cover some of the context-sensitive issues in translation

• Syntax-directed definition: an attributed grammar such that every grammar symbol has an attribute.

• Conceptually, attribute evaluation is
  – Build parse tree
  – Find attribute dependences
  – Decorate parse tree

Evaluation Methods

• Want to interleave attribute evaluation with parsing

• Use dependence graph (but cannot handle circular dependences)

• Predetermine evaluation order at compiler construction time, using knowledge of grammar

• Ad-hoc: chosen parsing method imposes evaluation order when interleaved with parsing; restricts grammars that can be handled
**S-attributed Grammars**

- **S-attributed grammars**: all attributes are synthesized
- Easy to interleave with BU parsing by using a parallel stack for attribute values
  - Evaluate as do a reduction
- Important: can code semantic functions *a priori*, because know all the handles from the grammar, so *know where the associated attributes will be in the stack when a reduction is about to take place.*

---

**Attribute Grammars**

- **Attributes**: properties associated with nonterminal symbols of a context free grammar

  **E.G., Binary numbers**

  1. B → 0 \( \text{val}(B) = 0 \)
  2. B → 1 \( \text{val}(B) = 1 \)
  3. L → B \( \text{val}(L) = \text{val}(B); \text{len}(L) = 1 \)
  4. L → L₁ B \( \text{val}(L) = 2 \ast \text{val}(L₁) + \text{val}(B) \)
     \( \text{len}(L) = \text{len}(L₁) + 1 \)
  5. N → L \( \text{val}(N) = \text{val}(L); \text{len}(N) = \text{len}(L) \)

  | Semantic rules defining attributes are side-effect free |
Example - Binary Nos

1. B → 0 \quad \text{val}(B) = 0
2. B → 1 \quad \text{val}(B) = 1
3. L → B \quad \text{val}(L) = \text{val}(B); \text{len}(L) = 1
4. L → L_b \quad \text{val}(L) = 2^*\text{val}(L_b) + \text{val}(B)
   \quad \text{len}(L) = \text{len}(L_b) + 1
5. N → L \quad \text{val}(N) = \text{val}(L); \text{len}(N) = \text{len}(L)

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ $</td>
<td>1 1 $</td>
<td>shift</td>
</tr>
<tr>
<td>$ (B 1 _)</td>
<td>1 $</td>
<td>red(2), find B</td>
</tr>
<tr>
<td>$ (L 1 1)</td>
<td>1 $</td>
<td>red(3), find L</td>
</tr>
<tr>
<td>$ (L 1 1) (1 1 _)</td>
<td>$</td>
<td>shift</td>
</tr>
<tr>
<td>$ (L 1 1) (B 1 _)</td>
<td>$</td>
<td>red(2), find B</td>
</tr>
<tr>
<td>$ (L 3 2)</td>
<td>$</td>
<td>red(5), find L</td>
</tr>
<tr>
<td>$ (N 3 2)</td>
<td>$</td>
<td>accept</td>
</tr>
</tbody>
</table>

<symbol> val() len()</symbol>

L-attributed Grammars

- Every attribute in the grammar is synthesized, or for production \( A → X_1 ... X_k \) an inherited attribute \( X_k \) only depends on attributes of \( X_1 ... X_{k-1} \) or inherited attributes of \( A \).
- Can use depth-first evaluation scheme on parse tree
- Includes all syntax-directed definitions from LL(1) grammars
L-attributed Grammars

Translation scheme: embeds semantic actions to evaluate attributes in RHS of productions (use {...} to delimit actions) to accomplish depth-first evaluation order

1. An inherited attributed for a nonterminal on RHS of production, must be computed in an action BEFORE that symbol

2. An action cannot refer to a synthesized attribute of a symbol to the right of the action

3. A synthesized attribute of the LHS nonterminal can only be computed after all attributes it refers to are computed; place this action at the end of the RHS of the production
Example - Identifiers as a Translation Scheme

D → \{ \text{str}(I) = \varepsilon \} \quad I \quad \{ \text{val}(D) = \text{val}(I) \}
\quad \{ \text{accept, if } \text{val}(D) \neq \text{error} \}

I → \{ \text{str}(L) = \text{str}(I) \} \quad L \quad \{ \text{str}(I_1) = \text{val}(L) \} \quad I_1
\quad \{ \text{val}(I) = \text{val}(I_1) \}

I → \{ \text{str}(L) = \text{str}(I) \} \quad L \quad \{ \text{val}(I) = \text{val}(L) \}

L → a \mid b \mid \ldots \mid z \quad \{ \text{val}(L) = \text{concatenation of val} \}

\text{returned by scanner to } \text{str}(L), \text{ if this character is not a repeated letter, else } \text{error} \}

Try to evaluate earlier example abc with depth-first walk and these rules.

Intuition

- Can see TD parsing relates well to L-attributed grammars
- Can see BU parsing relates well to S-attributed grammars
BU Eval of Inherited Attrbs

• *Idea*: transform grammar so all embedded actions of translation scheme occur at end of RHS of some production (at a reduction) without changing LR(k) nature of the grammar

• Can handle all L-attributed defns corresponding to LL(1) grammars plus some LR(1)

Marker Nonterminals

*Used to move all actions to end of RHS of productions*

Always $X \rightarrow \epsilon$ for $X$, a marker nonterminal.

Replace an embedded action by a unique marker nonterminal that generates $\epsilon$

Make the action for that nonterminal the same as the embedded action removed

But: grammar must stay LR(k) after these changes (this needs to be checked.)

Language accepted is same.

Actions occur in same order during parse.
Example, ASU p 309

\[
\begin{align*}
S & \rightarrow E \\
E & \rightarrow E + T \mid E - T \mid T \\
T & \rightarrow num
\end{align*}
\]

becomes after recursion removal with actions:

\[
\begin{align*}
S & \rightarrow E \\
E & \rightarrow T \ R \\
R & \rightarrow + T \ {\{\text{ print "+"}\}} \ R \\
R & \rightarrow - T \ {\{\text{ print "-"}\}} \ R \\
R & \rightarrow \varepsilon \\
T & \rightarrow num \ \{\text{print num}\}
\end{align*}
\]

Marker Nonterminals

\[
\begin{align*}
\text{LR(1) grammar} & \\
S & \rightarrow E \\
E & \rightarrow T \ R \\
R & \rightarrow + T \ {\{\text{ print "+"}\}} \ R \\
R & \rightarrow - T \ {\{\text{ print "-"}\}} \ R \\
R & \rightarrow \varepsilon \\
T & \rightarrow num \ {\{\text{print num}\}}
\end{align*}
\]

\[
\begin{align*}
\text{After transformation} & \\
S & \rightarrow E \\
E & \rightarrow T \ R \\
R & \rightarrow + T \ M \ R \\
R & \rightarrow - T \ N \ R \\
R & \rightarrow \varepsilon \\
M & \rightarrow \varepsilon \ {\{\text{ print "+"}\}} \\
N & \rightarrow \varepsilon \ {\{\text{print "-"}\}} \\
T & \rightarrow num \ {\{\text{print num}\}}
\end{align*}
\]
Marker Nonterminals (Copies)

- Handling copy rules with marker nonterminals
  
  \[ A \rightarrow X \ Y \text{ where } i(Y) = s(X) \]
  
  Translation scheme would be:
  
  \[ A \rightarrow X \ \{i(Y) = s(X)\} \ Y \]

- Example of this in our identifier grammar
  
  \[ str(I_1) = val(L) \text{ in } I \rightarrow L \ I_1 \]
  
  would become \( I \rightarrow L \ \{str(I_1) = val(L)\} \ I_1 \)

Example

```
D
 / \ 
I   I
 / \ / 
L a L a
 /    / 
I   I
 / \ / 
L a L ab
 /    / 
b
```
Example

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Attribute Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>a b $</td>
<td>$</td>
</tr>
<tr>
<td>$ a</td>
<td>b $</td>
<td>$</td>
</tr>
<tr>
<td>$ L</td>
<td>b $</td>
<td>a (val(L))</td>
</tr>
<tr>
<td>$ L b</td>
<td>$</td>
<td>a</td>
</tr>
<tr>
<td>$ L L_1</td>
<td>$</td>
<td>ab (val(L_1))</td>
</tr>
<tr>
<td>$ L I</td>
<td>$</td>
<td>ab (val(I))</td>
</tr>
<tr>
<td>$ I</td>
<td>$</td>
<td>ab (val(I))</td>
</tr>
<tr>
<td>$ D</td>
<td>$</td>
<td>ab (val(D))</td>
</tr>
</tbody>
</table>

Marker Nonterminals,(Copies ii)

- In previous example, copies never need to be performed as value is at top of attribute stack due to shape of grammar rules
- Not always this lucky
  
  $S \rightarrow a\ A\ C$ \hspace{1cm} i(C) = s(A) [1.]
  $S \rightarrow b\ A\ B\ C$ \hspace{1cm} i(C) = s(A) [2.]
  $C \rightarrow c$ \hspace{1cm} s(C) = g (i(C))

  Problem: in 1., s(A) is in stack(top) when find C but in 2., s(A) is in stack(top-1). Must rewrite grammar to try to make attribute value end up in same place in both rules.
**Grammar Transformation**

S → a A C       \( i(C) = s(A) \) [1.]
S → b A B M C   \( i(M) = s(A); \ i(C) = s(M) \) [2. ‘]
C → c           \( s(C) = g(i(C)) \)
M → \( \varepsilon \)     \( s(M) = i(M) \)

M saves the value of \( s(A) \) so it goes on the value stack at the same place in both rules 1., 2.’; when encounter C, makes \( i(C) \) in same stack position.

**Marker Nonterminals, (Non-copies)**

Previous transformation works even for non-copy actions:
if \( S \rightarrow b \ A \ C \) has action \( i(C) = f(s(A)) \)
then \( s(A) \) is on stack, not \( f(s(A)) \).

Fix:
\[ S \rightarrow a A N C, \quad i(N) = s(A), i(C) = s(N) \]
\[ N \rightarrow \varepsilon, \quad s(N) = f(i(N)) \]

Problem: can destroy the LR(1) property of grammar with added markers; LL(1) grammars remain okay.
Top Down Translation

• L-attributed grammars work well with TD translation, but when remove left recursion must also transform attributes
• Involves changing all synthesized attributes to a mixture of inherited and synthesized

Example

\[
\begin{align*}
S & \rightarrow E & \text{val}(S) = \text{val}(E) \\
E & \rightarrow E_1 + T & \text{val}(E) = \text{val}(E_1) + \text{val}(T) \\
E & \rightarrow E_1 - T & \text{val}(E) = \text{val}(E_1) - \text{val}(T) \\
E & \rightarrow T & \text{val}(E) = \text{val}(T) \\
T & \rightarrow \text{int} & \text{val}(T) = \text{int\_value}
\end{align*}
\]

All synthesized attributes (L-attributed).
Removing Left Recursion

\[
E \rightarrow E \alpha | \beta \quad E \rightarrow \beta A; \quad A \rightarrow \alpha A | \epsilon
\]

Want to specify attribute transformations as well!

Example

\[
\begin{align*}
S & \rightarrow E \quad \{ \text{val}(S) = \text{val}(E) \} \\
E & \rightarrow T \quad \{ \text{i}(R) = \text{val}(T) \} \quad R \quad \{ \text{val}(E) = \text{s}(R) \} \\
R & \rightarrow + T \quad \{ \text{i}(R_1) = \text{i}(R) + \text{val}(T) \} \quad R_1 \\
& \quad \quad \quad \quad \quad \{ \text{s}(R) = \text{s}(R_1) \} \\
R & \rightarrow - T \quad \{ \text{i}(R_1) = \text{i}(R) - \text{val}(T) \} \quad R_1 \\
& \quad \quad \quad \quad \quad \{ \text{s}(R) = \text{s}(R_1) \} \\
R & \rightarrow \epsilon \quad \{ \text{s}(R) = \text{i}(R) \} \\
T & \rightarrow \text{int} \quad \{ \text{val}(T) = \text{int}_\text{const} \}
\end{align*}
\]
Corresponding Parse Trees

Transformation, \(\text{ASUp}^{304}\text{ff}\)

\[\begin{align*}
A & \rightarrow A_1 Y \quad \{a(A) = g(a(A_1), y(Y))\} \\
A & \rightarrow X \quad \{a(A) = f(x(X))\}
\end{align*}\]

Becomes
\[\begin{align*}
A & \rightarrow X \quad \{i(R) = f(x(X))\} \quad R \quad \{a(A) = s(R)\} \\
R & \rightarrow Y \quad \{i(R_1) = g(i(R), y(Y))\} \quad R_1 \quad \{s(R) = s(R_1)\} \\
R & \rightarrow \varepsilon \quad \{s(R) = i(R)\}
\end{align*}\]
Transformation

\[ g\{g(f(x(X)),y(Y_1)),y(Y_2)\} \]

\[ g(f(x(X)),y(Y_1)) \]

\[ f(x(X)) \]

\[ X \]

\[ Y_1 \]

\[ Y_2 \]

\[ g(g(f(x(X)),y(Y_1)),y(Y_2)) \]

\[ g(f(x(X)),y(Y_1)) \]

\[ g(g(f(x(X)),y(Y_1)),y(Y_2)) \]

\[ e \]