CSE 309: Compiler

Tanvir Ahmed Khan
takhandipu@gmail.com

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology.

April 12, 2015

Recap

Syntax-Directed Translation

```
207 subprogram_declaration: subprogram_head declarations compound_statement
208             {
209                 printf("\nsubprogram_declaration -> subprogram_head declarations
    compound_statement\n");
210                 char *temp=new char[50];
211                 getTemp(temp);
212                 st.insert(temp,"temp");
213                 SymbolInfo *n=st.uplook(temp);
214                 n->code+="\n";
215                 n->code+=$1->symbol;
216                 n->code+=" proc\npush ax\npush bx\npush cx\npush dx\n";
217                 n->code+=$3->code;
218                 n->code+="\npop dx\npop cx\npop bx\npop ax\nret\n";
219                 n->code+=$1->symbol;
220                 n->code+=" endp\n";
221                 $$=n;
222                 delete [] temp;
223             }
224             ;
225 subprogram_head:     FUNCTION ID arguments COLON standard_type SEMICOLON
```

Syntax-Directed Translation

- source language translation is completely driven by Syntax analyzer or, Parser
- grammar written for parsing is augmented with information to control,
  - Semantic analysis
  - Translation

  attribute grammar

Attribute Grammar

- each grammar symbol is associated with attributes,
  - value
  - type
  - memory location

Today's Topic

Syntax-Directed Translation, Contd.

How to Compute Attribute Values

How to Compute Attribute Values

- using semantic rules associated with each production in a grammar

How to Compute Attribute Values

- using semantic rules associated with each production in a grammar
  - $E \rightarrow E + T$

How to Compute Attribute Values

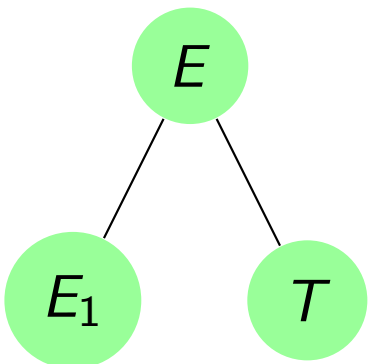
- using semantic rules associated with each production in a grammar
  - $E \rightarrow E + T$
  - $E \rightarrow E_1 + T$

How to Compute Attribute Values

- using semantic rules associated with each production in a grammar
  - $E \rightarrow E + T$
  - $E \rightarrow E_1 + T$
  - $E \rightarrow E_1 + T \ \{E.\texttt{code}=E_1.\texttt{code}||T.\texttt{code}||\texttt{'+'}.\texttt{code}\}$

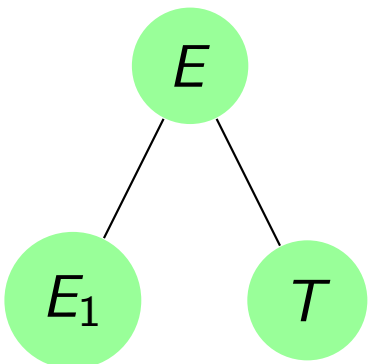
How to Compute Attribute Values

- using semantic rules associated with each production in a grammar
  - $E \rightarrow E + T$
  - $E \rightarrow E_1 + T$
  - $E \rightarrow E_1 + T \ \{E.\texttt{code}=E_1.\texttt{code}||T.\texttt{code}||\texttt{'+'}.\texttt{code}\}$



How to Compute Attribute Values

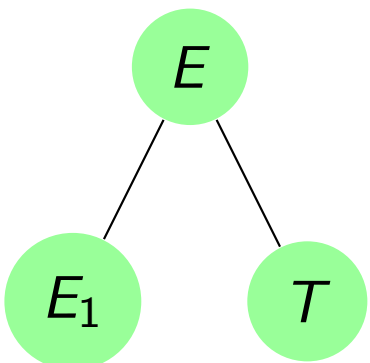
- using semantic rules associated with each production in a grammar
  - $E \rightarrow E + T$
  - $E \rightarrow E_1 + T$
  - $E \rightarrow E_1 + T \ \{E\texttt{.code=}E_1\texttt{.code||}T\texttt{.code||'+'.code}\}$



- attribute value for a parse node may depend on information from its

How to Compute Attribute Values

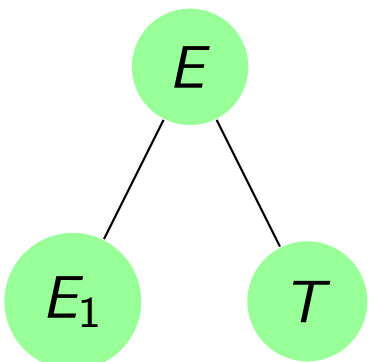
- using semantic rules associated with each production in a grammar
  - $E \rightarrow E + T$
  - $E \rightarrow E_1 + T$
  - $E \rightarrow E_1 + T \ \{E.\texttt{code}=E_1.\texttt{code}||T.\texttt{code}||\texttt{'+'}.\texttt{code}\}$



- attribute value for a parse node may depend on information from its
  - children nodes below, $E.val = E_1.val + T.val$

How to Compute Attribute Values

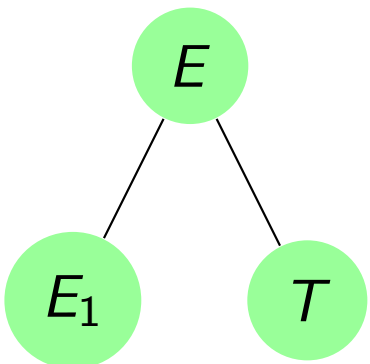
- using semantic rules associated with each production in a grammar
  - $E \rightarrow E + T$
  - $E \rightarrow E_1 + T$
  - $E \rightarrow E_1 + T \ \{E.\texttt{code}=E_1.\texttt{code}||T.\texttt{code}||\texttt{'+'}.\texttt{code}\}$



- attribute value for a parse node may depend on information from its
  - children nodes below, $E.val = E_1.val + T.val$
  - siblings, $T.type = E_1.type$

How to Compute Attribute Values

- using semantic rules associated with each production in a grammar
  - $E \rightarrow E + T$
  - $E \rightarrow E_1 + T$
  - $E \rightarrow E_1 + T \ \{E.\texttt{code}=E_1.\texttt{code}||T.\texttt{code}||\texttt{'+'}.\texttt{code}\}$



- attribute value for a parse node may depend on information from its
  - children nodes below, $E.val = E_1.val + T.val$
  - siblings, $T.type = E_1.type$
  - parent node above, $E_1.scope = E.scope$

Attribute Types

Based on how attributes are computed, two types of attributes,

- synthesized attributes
- inherited attributes

Synthesized Attributes

Synthesized Attributes

- left-side attribute is computed from the right-side attributes

Synthesized Attributes

- left-side attribute is computed from the right-side attributes
  $X \rightarrow Y_1 Y_2 \ldots Y_n$

Synthesized Attributes

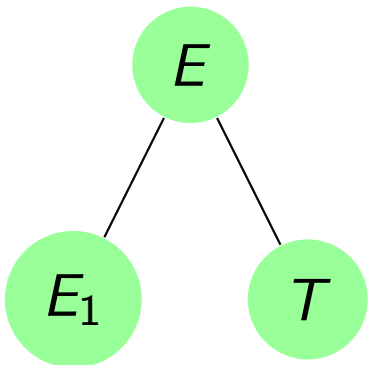
- left-side attribute is computed from the right-side attributes
  $X \to Y_1 Y_2 \ldots Y_n$
  $\{X.a = f(Y_1.a, Y_2.b, \ldots, Y_n.a)\}$

Synthesized Attributes

- left-side attribute is computed from the right-side attributes
  $X \to Y_1 Y_2 \ldots Y_n$
  $\{X.a = f(Y_1.a, Y_2.b, \ldots, Y_n.a)\}$
- attributes that are passed *up* a parse tree

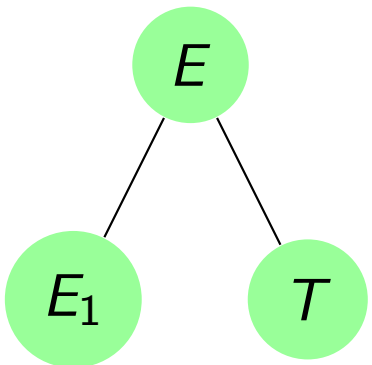
Synthesized Attributes

- left-side attribute is computed from the right-side attributes
  $X \to Y_1 Y_2 \ldots Y_n$
  $\{X.a = f(Y_1.a, Y_2.b, \ldots, Y_n.a)\}$
- attributes that are passed *up* a parse tree



Synthesized Attributes

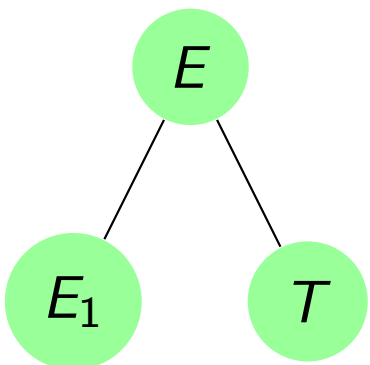
- left-side attribute is computed from the right-side attributes
  $X \to Y_1 Y_2 \ldots Y_n$
  $\{X.a = f(Y_1.a, Y_2.b, \ldots, Y_n.a)\}$
- attributes that are passed *up* a parse tree



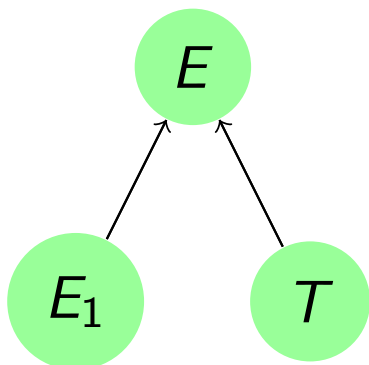
- $E.val = E_1.val + T.val$

Synthesized Attributes

- left-side attribute is computed from the right-side attributes
  $X \to Y_1 Y_2 \ldots Y_n$
  $\{X.a = f(Y_1.a, Y_2.b, \ldots, Y_n.a)\}$
- attributes that are passed *up* a parse tree



- $E.val = E_1.val + T.val$



Inherited Attributes

Inherited Attributes

- right-side attribute is computed from the left-side attribute or, other right-side attributes

Inherited Attributes

- right-side attribute is computed from the left-side attribute or, other right-side attributes
  $X \rightarrow Y_1 Y_2 \ldots Y_n$

Inherited Attributes

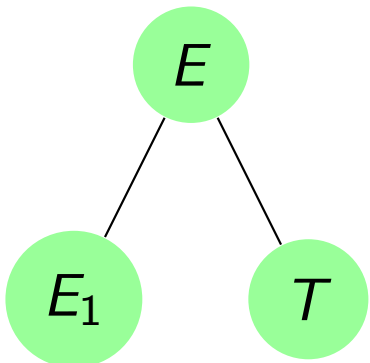
- right-side attribute is computed from the left-side attribute or, other right-side attributes
  $X \rightarrow Y_1 Y_2 \ldots Y_n$
  $\{Y_k.a = f(X.a, Y_1.a, Y_2.c, \ldots, Y_n.a)\}$

Inherited Attributes

- right-side attribute is computed from the left-side attribute or, other right-side attributes
  $X \rightarrow Y_1 Y_2 \ldots Y_n$
  $\{Y_k.a = f(X.a, Y_1.a, Y_2.c, \ldots, Y_n.a)\}$
- attributes that are passed *down* a parse tree

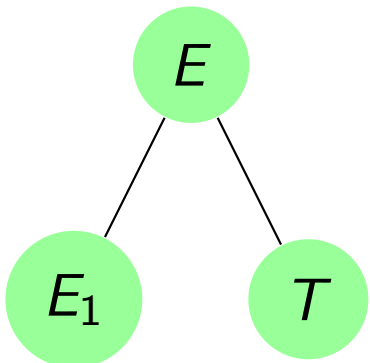
Inherited Attributes

- right-side attribute is computed from the left-side attribute or, other right-side attributes
  $X \rightarrow Y_1 Y_2 \ldots Y_n$
  $\{Y_k.a = f(X.a, Y_1.a, Y_2.c, \ldots, Y_n.a)\}$
- attributes that are passed *down* a parse tree



Inherited Attributes

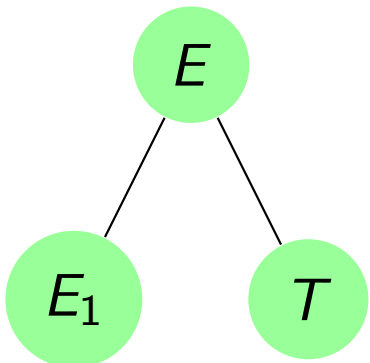
- right-side attribute is computed from the left-side attribute or, other right-side attributes
  $X \rightarrow Y_1 Y_2 \ldots Y_n$
  $\{Y_k.a = f(X.a, Y_1.a, Y_2.c, \ldots, Y_n.a)\}$
- attributes that are passed *down* a parse tree



- $E_1.type = T.type$

Inherited Attributes

- right-side attribute is computed from the left-side attribute or, other right-side attributes
  $X \rightarrow Y_1 Y_2 \ldots Y_n$
  $\{Y_k.a = f(X.a, Y_1.a, Y_2.c, \ldots, Y_n.a)\}$
- attributes that are passed *down* a parse tree



- $E_1.type = T.type$
- $E_1.scope = E.scope$

Attribute Grammar Example

Parsing Grammar,

$$
\begin{array}{lcl}
P & \rightarrow & DS \\
D & \rightarrow & \mathit{var}\ V;\ D \\
D & \rightarrow & \epsilon \\
S & \rightarrow & V\ :=\ E;\ S \\
S & \rightarrow & \epsilon
\end{array}
$$

Attribute Grammar Example

Parsing Grammar,

$$
\begin{array}{lcl}
P & \rightarrow & DS \\
D_1 & \rightarrow & \mathit{var}\ V;\ D_2 \\
D & \rightarrow & \epsilon \\
S_1 & \rightarrow & V\ :=\ E;\ S_2 \\
S & \rightarrow & \epsilon
\end{array}
$$

Attribute Grammar Example

Syntax-Directed Translation,

Attribute Grammar Example

Syntax-Directed Translation,

$$P \quad \rightarrow \quad DS \qquad\qquad \{S\texttt{.dList=}D\texttt{.dList}\}$$

Attribute Grammar Example

Syntax-Directed Translation,

$$
\begin{array}{lcll}
P & \rightarrow & DS & \{S\texttt{.dList=}D\texttt{.dList}\} \\
D_1 & \rightarrow & \mathit{var}\ V;\ D_2 & \{D_1\texttt{.dList=}D_2\texttt{.dList.append(}V\texttt{.name)}\}
\end{array}
$$

Attribute Grammar Example

Syntax-Directed Translation,

P	$\rightarrow$	DS	$\{S.\texttt{dList}=D.\texttt{dList}\}$
D_1	$\rightarrow$	$var\ V;\ D_2$	$\{D_1.\texttt{dList}=D_2.\texttt{dList.append}(V.\texttt{name})\}$
D	$\rightarrow$	ϵ	$\{D.\texttt{dList}=\{\}\}$

Attribute Grammar Example

Syntax-Directed Translation,

P	$\rightarrow$	DS	$\{S.\texttt{dList}=D.\texttt{dList}\}$
D_1	$\rightarrow$	$var\ V;\ D_2$	$\{D_1.\texttt{dList}=D_2.\texttt{dList.append}(V.\texttt{name})\}$
D	$\rightarrow$	ϵ	$\{D.\texttt{dList}=\{\}\}$
S_1	$\rightarrow$	$V := E;\ S_2$	$\{\texttt{check}(V.\texttt{name}, S_1.\texttt{dList});$ $S_2.\texttt{dList}=S_1.\texttt{dList};\}$

Attribute Grammar Example

Syntax-Directed Translation,

P	$\rightarrow$	DS	$\{S.\texttt{dList}=D.\texttt{dList}\}$
D_1	$\rightarrow$	$var\ V;\ D_2$	$\{D_1.\texttt{dList}=D_2.\texttt{dList.append}(V.\texttt{name})\}$
D	$\rightarrow$	ϵ	$\{D.\texttt{dList}=\{\}\}$
S_1	$\rightarrow$	$V := E;\ S_2$	$\{\texttt{check}(V.\texttt{name}, S_1.\texttt{dList});$ $S_2.\texttt{dList}=S_1.\texttt{dList};\}$
S	$\rightarrow$	ϵ	$\{\}$

Attribute Grammar Example

Syntax-Directed Translation,

P	$\rightarrow$	DS	$\{S.\texttt{dList}=D.\texttt{dList}\}$
D_1	$\rightarrow$	$var\ V;\ D_2$	$\{D_1.\texttt{dList}=D_2.\texttt{dList.append}(V.\texttt{name})\}$
D	$\rightarrow$	ϵ	$\{D.\texttt{dList}=\{\}\}$
S_1	$\rightarrow$	$V := E;\ S_2$	$\{\texttt{check}(V.\texttt{name}, S_1.\texttt{dList});$ $S_2.\texttt{dList}=S_1.\texttt{dList};\}$
S	$\rightarrow$	ϵ	$\{\}$

Reference

- 16: Syntax-Directed Translation
  From CS 143 Compilers Course