Recitation #10

Administrative

- Assignment 4 – Late Penalty changed
  - 5% off for each day late \(\rightarrow\) up to 4 days late, i.e. 20% off when submitted on Friday
  - If you submitted before the deadline, WebCourses doesn’t let you submit again \(\rightarrow\) send e-mail with attached files
  - Any more questions on assignment?
- Final Project is coming up
  - Dr. Hughes will discuss in class
  - Will build on and extend Assignment 4

SLR Parser - Worksheet

- Continue from last week and finish worksheet
- Review some terminology and CLOSURE and GOTO functions
- See notes for Recitation #9 and solution for worksheet

Syntax-Directed Definition

- Associate information with language construct by attaching attributes to grammar symbols
- Attributes can represent e.g. data types, definition scope, function visibility etc.
- Syntax-Directed Definition (SDD) specifies values of attributes by associating semantic rules with the grammar productions, e.g.

\[
E \rightarrow E_1 + T \quad E.val = E_1.val + T.val
\]

\[
E \rightarrow E_1 - T \quad E.val = E_1.val - T.val
\]

- Bison/Yacc has built-in notion of attributes, referred to as $$, $1 \rightarrow\) should be familiar with them from Assignment 4
- Annotated Parse Tree:
  - Adds attributes to nodes
  - Also called attributed or decorated parse tree
  - Depth-first search tree traversal
  - Each step in tree traversal performs one of these steps: Process node data (N), traverse child n (C..)
  - Three common traversal types:
1. **Pre-order traversal (NC..)**: Each node is visited before its children are visited, e.g. A, B, C, E, F, D
2. **Post-order traversal (C..N)**: The children of a node are visited before the node itself, e.g. B, E, F, C, D, A
3. **Inorder traversal**: Makes only sense in binary tree. Traverse left subtree, before node (N), then traverse right subtree.

- How do we determine the order of evaluation of the attributes?
  - Build parse tree
  - Create dependency graph between attributes
  - Find a topological sort on that graph and evaluate attributes in that order

**Synthesized Attributes**

- Attributes are synthesized when they are defined at a node labeled A using attributes of the node and its children
- The usual synthesized attributes start as token values, which are the leaves of the parse tree. Terminals are only allowed to have synthesized attributes
- Everything on right-hand side of grammar rule are children non-terminals and terminals in parse tree
- In Bison, each action synthesizes the value of its resulting symbol ($$) from the values of the symbols on the right-hand side of the rule, e.g. $1, $2 etc.
- In bottom-up parser with synthesized attributes, information passes from leaves to root

**S-attributed SDD**

- SDD with only synthesized attributes
- Evaluates up the tree (typically post-order traversal)
- Example:

```
E \rightarrow E_1 + T  \quad E.val := E_1.val + T.val
E \rightarrow E_1 - T  \quad E.val := E_1.val - T.val
E \rightarrow T  \quad E.val := T.val
T \rightarrow T_1 * F  \quad T.val := T_1.val * F.val
T \rightarrow T_1 / F  \quad T.val := T_1.val / F.val
T \rightarrow F  \quad T.val := F.val
F \rightarrow \neg F_1  \quad F.val := \neg F_1.val
F \rightarrow (E)  \quad F.val := E.val
F \rightarrow id  \quad F.val := id.entry
F \rightarrow \text{unsigned_integer}  \quad F.val := \text{unsigned_integer}.val
```

- Annotated Parse Tree for string **3 * (5 + 4):**
**Inherited Attributes**

- Information sometimes needs to pass from parent to children
- Attributes are inherited when they are defined at node labeled A using attributes of the node, its parent and its children.
- **When can they be important?**
  - Example (in Bison):

```
1 declaration:
2   class type namelist
3   ;
4
5 class:
6   GLOBAL { $$ = 1; } |
7   LOCAL { $$ = 2; } |
8   ;
9
10 type:
11   REAL { $$ = 1; } |
12   INTEGER { $$ = 2; } |
13   ;
14
15 namelist:
16   NAME { mksymbol($0, $1, $2); } |
17   namelist NAME { mksymbol($0, $1, $2); } |
18   ;
```

- It would be useful to have class and type available in the actions for ‘namelist’, e.g. for error checking and entering in symbol table
- Bison allows access to symbols on its internal stack (to the left of current rule)
- Inherited attributes can be source of hard-to-find bugs → Make sure correct symbols always precede ‘namelist’ in new grammar rules

- **L-attributed SDD**
  - SDD with mixed synthesized and inherited attributes
  - Can be evaluated left-to-right and depth-first
Example (Expression grammar that is right-recursive and left factored):

1. \[ T \rightarrow F \ FT \quad FT.\text{inh} := F.\text{val}; T.\text{val} := FT.\text{syn} \]
2. \[ FT \rightarrow *F \ FT_i \quad FT_i.\text{inh} := FT.\text{inh} * F.\text{val}; FT.\text{syn} := FT_i.\text{syn} \]
3. \[ FT \rightarrow \varepsilon \quad FT.\text{syn} = FT.\text{inh} \]
4. \[ F \rightarrow \text{unsigned_integer} \quad F.\text{val} := \text{unsigned_integer}.\text{val} \]

The following explanation is taken from the [Aho] book pages 308 – 309:

- The semantic rules are based on the idea that the left operand of the operator * is inherited.
- The head \( FT \) of the production \( FT \rightarrow *F \ FT_i \) inherits the left operand of * in the production body.
- E.g. given a term \( x \ast y \ast z \), the root of the subtree for \( y \ast z \) inherits \( x \). Then, the root of the subtree for \( z \) inherits the value of \( x \ast y \), and so on. Once all the factors have been accumulated, the result is passed back up the tree using synthesized attributes.

Annotated Parse Tree for string \( 3 \ast 5 \):

- Leftmost leaf in the parse tree has value 3 from the lexical analyzer.
- Its parent is for production (4), only semantic rule takes value 3.
- At the second child of the root, the inherited attribute \( FT.\text{inh} \) is defined by the rule \( FT.\text{inh} := F.\text{val} \) associated with production (1). Thus, the left operand, 3, for the * operand is passed from left to right across the children of the root.
- The production (2) applies at node \( FT \). The inherited attribute \( FT_i.\text{inh} \) is defined by the rule \( FT_i.\text{inh} := FT.\text{inh} * F.\text{val} \).
- With \( FT_i.\text{inh} := 3 \) and \( F.\text{val} := 5 \), we get \( FT_i.\text{inh} := 15 \).
- At the lower node for \( FT_i \), production (3) applies. The semantic rule \( FT_i.\text{syn} = FT_i.\text{inh} \) defines \( FT_i.\text{syn} := 15 \).
- The syn attributes at the nodes for \( FT \) pass the value 15 up the tree to the node for \( T \), where \( T.\text{val} := 15 \).
Bigger Example:

\[
\begin{align*}
E \rightarrow & \ T \ TT & TT\ inh := T.\ val; E.\ val := TT.\ syn \\
TT \rightarrow & \ +T \ TT_1 & TT_1.\ inh := TT.\ inh + T.\ val; TT.\ syn := TT_1.\ syn \\
TT \rightarrow & \ -T \ TT & TT_1.\ inh := TT.\ inh - T.\ val; TT.\ syn := TT_1.\ syn \\
TT \rightarrow & \ \epsilon & TT.\ syn := TT.\ inh \\
T \rightarrow & \ F \ FT & FT.\ inh := F.\ val; T.\ val := FT.\ syn \\
FT \rightarrow & \ \ast F \ FT_1 & FT_1.\ inh := FT.\ inh \ast F.\ val; FT.\ syn := FT_1.\ syn \\
FT \rightarrow & \ / F \ FT_1 & FT_1.\ inh := FT.\ inh / F.\ val; FT.\ syn := FT_1.\ syn \\
FT \rightarrow & \ \epsilon & FT.\ syn = FT.\ inh \\
F \rightarrow & \ -F_1 & F.\ val := -F.\ val \\
F \rightarrow & \ (E) & F.\ val := E.\ val \\
F \rightarrow & \ id & F.\ val := id.\ entry \\
F \rightarrow & \ unsigned\_integer & F.\ val := unsigned\_integer.\ val
\end{align*}
\]

- Annotated Parse Tree for string $3 \ast (5 + 4)$: