Outline

- Recap
- Syntax-directed translation (Chapter 5)
- Summary and homework
SDD’s

- For synthesized attributes
  - Perform bottom-up tree traversal for attribute evaluation
  - An SDD is \textit{S-attributed} if every attribute is synthesized

- For SDD’s with both inherited and synthesized attributes
  - Dependency graphs
  - An SDD is \textit{L-attributed} if in all of its dependency graphs the edges only go from left to right but not from right to left
Syntax-directed Translation Schemes

- Note that SDD is used for specifications
  - SDD $\Rightarrow$ SDT
- SDT’s are implemented during parsing without building a parse tree
  - S-attributed SDD based on LR-parsable grammar
  - L-attributed SDD based on LL-parsable grammar
S-attributed SDD’s Based on LR-parsable Grammars

- **SDD ⇒ SDT**
  - Construct an SDT where actions are placed at the end of the productions corresponding to semantic rules
    - [[Postfix SDT’s]]
  - **SDD Production**
    - SDD Production: $T \rightarrow T_1 \ast F$
    - Semantic rules: $T.val = T_1.val \ast F.val$
  - **SDD Production**
    - SDD Production: $T \rightarrow F$
    - Semantic rules: $T.val = F.val$
  - **SDD Production**
    - SDD Production: $F \rightarrow \text{digit}$
    - Semantic rules: $F.val = \text{digit}.lexval$
  - **SDT**
    - SDT: $T \rightarrow T_1 \ast F$
    - Action: $\{ T.val = T_1.val \ast F.val; \}$
    - SDT: $T \rightarrow F$
    - Action: $\{ T.val = F.val; \}$
    - SDT: $F \rightarrow \text{digit}$
    - Action: $\{ F.val = \text{digit}.lexval; \}$

- An action is executed along with the reduction of the body to the head of the associated production
Parser-Stack Implementation of Postfix SDT’s

- The attribute(s) of each grammar symbol can be put on the stack along with the symbol
  - \( T \rightarrow T_1 \ast F \)

<table>
<thead>
<tr>
<th></th>
<th>*</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T.val</td>
<td>F.val</td>
<td>...</td>
</tr>
</tbody>
</table>

- When reduction
  - \( \text{stack[top-2].val} = \text{stack[top].val} \ast \text{stack[top-2].val}; \)
  - \( \text{top} = \text{top} - 2; \)
Another SDD to SDT Example

- SDD of **while** to generate 3-address code
  - \( S \rightarrow \text{while ( C ) } S \)
    - \( \text{while}(i<a) \ i=i*2; \)
    - \( j=j*i; \)
  - **label L5**: if \( i<a \) goto L6
    - goto L7
  - **label L6**: \( i=i*2 \)
    - goto L5
  - **label L7**: \( j=j*i \)

- \( S \rightarrow \text{while ( C ) } S \)

  \[ \begin{align*}
  L1 &= \text{new(); } L2 = \text{new();} \\
  S1.\text{next} &= L1; \\
  C.\text{true} &= L2; \quad C.\text{false} = S.\text{next}; \\
  S.\text{code} &= \textbf{label} \ || \ L1 \ || \ C.\text{code} \ || \ \textbf{label} \ || \ L2 \ || \ S1.\text{code}
  \end{align*} \]
Another SDD to SDT Example

- For synthesized attribute computations
  - End of the production body
- For computing inherited attributes of a nonterminal
  - Immediately before the nonterminal occurrence

\[
S \rightarrow \text{while ( } \{ \text{ L1=new(); L2=new();} \\
\quad \text{ C.true=L2; C.false=S.next;} \\
\} \text{ C) S1.next=L1; } \\
S1 \quad \{ \text{ S.code= label || L1 || C.code || label || L2 || S1.code; } \}
\]
SDT’s With Actions Inside Productions

- L-attributed SDD’s $\Rightarrow$ SDT’s with actions inside productions
- When remove left-recursions in an S-attributed SDD for LL parsing $\Rightarrow$ SDT’s with actions inside production
Define A Language and Syntax-Directed Translation (1/23)

- expr → expr + term | expr - term | term
- term → 0 | 1 | ... | 9
- Syntax-directed translation based on semantic actions
  - expr → expr + term { print('+) }
  | expr - term { print('-') }
  | term
  - term → 0 { print('0') }
  | 1 { print('1') }
  |...
  | 9 { print('9') }
SDT’s With Actions Inside Productions

A → A α | β
A → β R
R → α R | ε

- Left recursion removal for top-down parsing
- expr → term rest
- rest → + term { print(‘+’) } rest
  | - term { print(‘-’) } rest
  | ε
- term → 0 { print(‘0’) }
  | 1 { print(‘1’) }
  | ...
  | 9 { print(‘9’) }
SDT’s With Actions Inside Productions

- Any SDT can be implemented as follows
  - Ignore the actions, parse the input and produce a parse tree
  - Add actions into the parser tree corresponding to the SDT
  - Perform a preorder traversal of the tree
Use a recursive-descent parser with one function for each nonterminal

Generate code on the fly using a recursive-descent parser

Implement an SDT in conjunction with an LL-parser

Implement an SDT in conjunction with an LR-parser (in April 19th’s lecture)
Recursive-descent Parser: One Function For Each Nonterminal

- Function arguments: inherited attributes
- Function return: synthesized attributes
- The **while** example
Generate Code on the Fly Using a Recursive-descent Parser

- Incrementally generate pieces of the code into an array or output file
  - Avoid copying or moving long strings
- The **while** example
Implement An SDT in Conjunction with An LL-parse

- If L-attributed SDD is based on an LL-grammar
  - Extend the parser stack to hold actions and items for attribute evaluations
Non-recursive Predictive Parsing

(2/27)

- A stack storing symbols, initialized with $S$
- An input buffer with an input pointer $ip$
- A parsing table $M$ for grammar $G$

Point $ip$ to the 1st input symbol
Set $A$ to the top stack symbol
while ($A \neq \$$) {
    if ($A$ is $a$) pop stack; advance $ip$
    else if ($A$ is a terminal) error();
    else if ($M[A,a]$ is an error entry) error();
    else if ($M[A,a] = A \rightarrow X_1 X_2 \ldots X_k$) {
        output the production or other actions;
        pop the stack;
        push $X_k, \ldots, X_2, X_1$ onto the stack with $X_1$ on top;
    }
    Set $A$ to the top stack symbol;
}
Implement An SDT in Conjunction with An LL-parse

- If L-attributed SDD is based on an LL-grammar
  - Extend the parser stack to hold actions and items for attribute evaluations
    - Action record for inherited attribute computation
      - What should be in the record?
      - Placed above or below the nonterminal?
      - The SDD for while to generate code on the Fly
Implement An SDT in Conjunction with An LL-parse

- If L-attributed SDD is based on an LL-grammar
  - Extend the parser stack to hold actions and items for attribute evaluations
    - Action record for inherited attribute computation
    - Synthesize record for synthesized attribute computation
      - Placed above or below the nonterminal?
Outline

- Recap
- Syntax-directed translation
- Summary and homework
Final Exam Reminder

- THURSDAY, MAY 03, 2007, 08:00-11:00AM
Homework (Due on 04/02)

10.1. (a) Exercise 5.2.4 (page 317);
(b) Exercise 5.2.5 (Page 317).