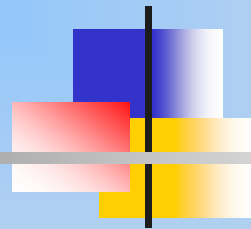
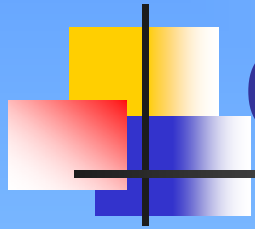


CSE302: Compiler Design



Instructor: Dr. Liang Cheng
Department of Computer Science and Engineering
P.C. Rossin College of Engineering & Applied Science
Lehigh University

April 5, 2007



Outline

- Recap
- Syntax-directed translation (Chapter 5)
- Intermediate-code generation
- Summary and homework



Syntax-directed Translation Schemes

- SDD \Rightarrow SDT
 - SDT's can be implemented during parsing without building a parse tree
 - S-attributed SDD based on LR-parsable grammar
 - L-attributed SDD based on LL-parsable grammar



An L-attributed SDD Example

- SDD of **while** to generate 3-address code

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

- $S \rightarrow \text{while} (C) S$ `L1=new(); L2=new();
S1.next=L1;
C.true=L2; C.false=S.next;`

`S.code= label || L1 || C.code || label || L2 || S1.code`



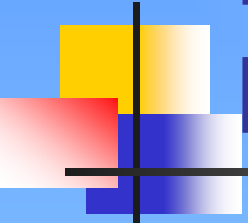
The Corresponding SDT

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

```
S → while ( { L1=new(); L2=new();  
             C.true=L2; C.false=S.next;  
             }
```

```
  C)  S1.next=L1;
```

```
  S1  { S.code= label || L1 || C.code || label || L2 || S1.code; }
```



Recursive-descent Parser: One Function For Each Nonterminal

```
S → while ( { L1=new(); L2=new();  
              C.true=L2; C.false=S.next;  
            }  
C)   S1.next=L1;  
S1   { S.code= label || L1 || C.code || label || L2 || S1.code; }
```

```
String S(label next) {  
    String Scode, Ccode;    Label L1, L2;  
    if(inputToken == whileToken) {  
        advanceInput();    match('(');    L1=new(); L2=new();  
        Ccode = C(next, L2);    match(')');    Scode = S(L1);  
        return("label"||L1||Ccode||"label"||L2||Scode);  
    }  
}
```

Generate Code on the Fly Using a Recursive-descent Parser

```
String S(label next) {
    String Scode, Ccode;      Label L1, L2;
    if(inputToken == whileToken) {
        advanceInput();      match('(');      L1=new(); L2=new();
        Ccode = C(next, L2);  match(')');    Scode = S(L1);
        return("label"||L1||Ccode||"label"||L2||Scode);
    }
}
```

```
String S(label next) {
    String Scode, Ccode;      Label L1, L2;
    if(inputToken == whileToken) {
        advanceInput();      match('(');      L1=new(); L2=new();
        print("label",L1); C(next, L2); match(')');    print("label",L2); S(L1);
        return("label"||L1||Ccode||"label"||L2||Scode);
    }
}
```

■ New SDD



Implement An SDT in Conjunction with An LL-parser

- 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 ($M[A,a] = A \rightarrow X_1X_2\dots X_k$) {

 output the production or other actions;

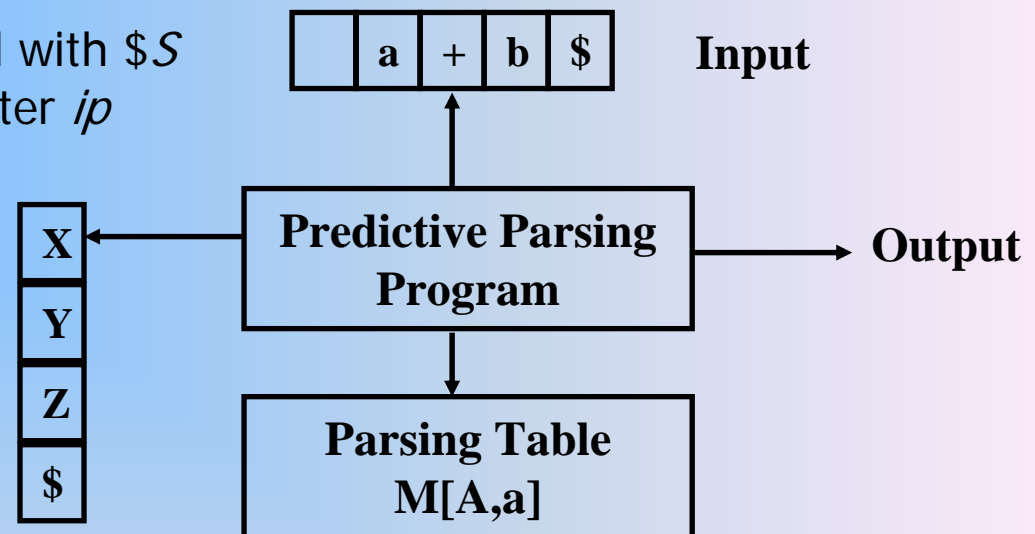
 pop the stack;

 push X_k, \dots, X_2, X_1 onto the stack with X_1 on top;

 }

 Set A to the top stack symbol;

} Instructor: Dr. Liang Cheng





Implement An SDD 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?

```
S → while ( { L1=new(); L2=new(); print("label",L1);  
              C.true=L2; C.false=S.next;  
            }  
C) {print("label",L2); S1.next=L1;}  
S1
```



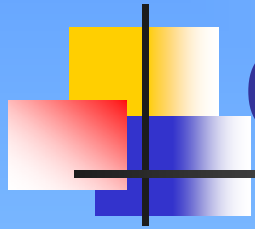
Implement An SDT in Conjunction with An LL-parse

- 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?

```
S → while ( { L1=new(); L2=new();  
             C.true=L2; C.false=S.next;  
             }
```

```
    C)  S1.next=L1;
```

```
    S1  { S.code= label || L1 || C.code || label || L2 || S1.code; }
```



Outline

- Recap
- Syntax-directed translation
- **Intermediate-code generation**
- Summary and homework



Major Topics in Chapter 6

- Intermediate representations
- Static checking
- Intermediate code generation



Variant of Syntax Trees

- Directed acyclic graph (DAG) for expression
 - Identify common subexpressions
 - $a + a * (b - c) + (b - c) * d$
 - Before creating a new node, check whether an identical node already exists
 - Node(op, left, right)
 - Stored in an array of records

■ Production	Semantic rules
$A \rightarrow T = E$	$A.\text{node} = \mathbf{new} \text{Node}('=', T.\text{node}, E.\text{node})$
$E \rightarrow T_1 + T_2$	$E.\text{node} = \mathbf{new} \text{Node}('+', T_1.\text{node}, T_2.\text{node})$
$T \rightarrow \mathbf{id}$	$T.\text{node} = \mathbf{new} \text{Leaf}(\mathbf{id}, \mathbf{id}.\text{entry})$
$T \rightarrow \mathbf{num}$	$T.\text{node} = \mathbf{new} \text{Leaf}(\mathbf{num}, \mathbf{num}.\text{val})$

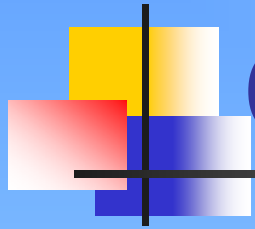


Three-Address Code

- Three-address code for the DAG example
 - $a + a * (b - c) + (b - c) * d$
- An address
 - A name
 - A pointer to its symbol-table entry
 - A constant
 - A compiler-generated temporary
- Symbolic labels
 - L1, L2, ...

Three-Address Instruction Forms

- $x = y \text{ op } z$
- $x = \text{op } y$
- $x = y$
- goto L
- $x = y[i]$
- $x[i] = y$
- $x = \&y$
- $x = *y$
- $*x = y$
- if x goto L and ifFalse x goto L
- if x relop y goto L
- Procedure calls and returns
 - param x_1
 - param x_2
 - ...
 - param x_n
 - call p, n or $y = \text{call } p, n$
 - return w



Outline

- Recap
- Syntax-directed translation
- Intermediate-code generation
- **Summary and homework**



Final Exam Reminder

- THURSDAY, MAY 03, 2007,
08:00-11:00AM



Homework (Due on 04/09)

- HW11.1 and HW11.2 are posted at the Blackboard.