CSE302: Compiler Design

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

January 23, 2007

Today's Outline

- Recap
  - Introduction (Section 2.1)
  - Syntax definition (Section 2.2)
  - Parsing (Section 2.4)
- A simple syntax-directed translator (Chapter 2)
  - Parsing (Section 2.4.5)
  - Syntax directed translation (Section 2.3)
  - A translator for simple expressions (Section 2.5)
- Summary and homework

BNF Grammar and Parse Trees

BNF Functionality
- Describing Lists
- Grammar & Derivation
- Parse Trees
- Avoiding Ambiguity

<program> $\rightarrow$ <stmts>

<stmts> $\rightarrow$ <stmt> | <stmt> ; <stmts>

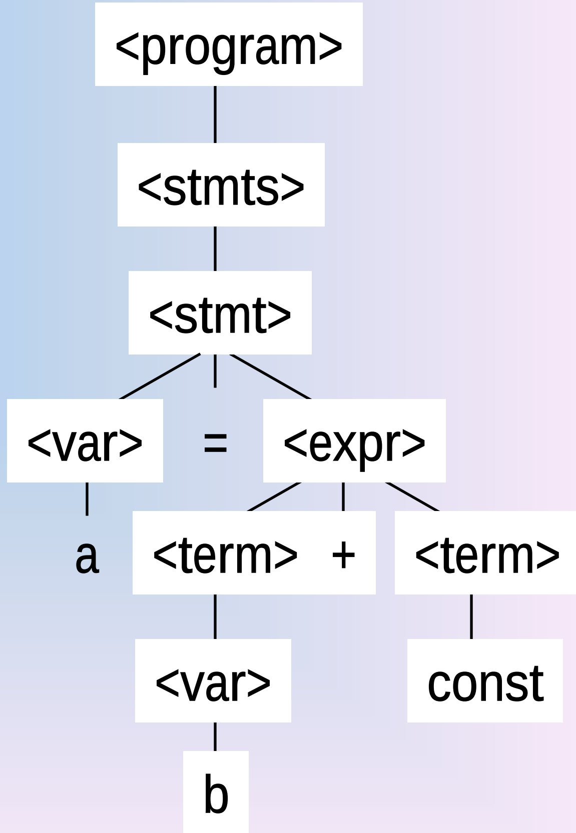
<stmt> $\rightarrow$ <var> = <expr>

<var> $\rightarrow$ a | b | c | d

<expr> $\rightarrow$ <term> + <term> | <term> - <term>

<term> $\rightarrow$ <var> | const

<program>
=> <stmts>
=> <stmt>
=> <var> = <expr>
=> **a** = <expr>
=> **a** = <term> **+** <term>
=> **a** = <var> **+** <term>
=> **a = b +** <term>
=> **a = b +** const



Grammar Ambiguity

- A grammar is ambiguous iff it generates a sentential form that has two or more distinct parse trees
- Use BNF to specify operator precedence and associativity

Language Design

- Design a BNF grammar for a language that could express a one-digit number, an addition of two one-digit numbers, or a subtraction of two one-digit numbers
  - <expr> → <term> + <term> | <term> - <term> | <term>
  - <term> → 0 | 1 | 2 | … | 9

Language Implementation

- A recursive-descent parser
  - Language implementation directly following the BNF grammar
  - <expr> → <term> + <term> | <term> - <term> | <term>
  - <term> → 0 | 1 | 2 | … | 9
- Pseudo code

```
void expr() {
        term();
        if( token==plus_op
         or token==minus_op) {
            match(token);
            term();
        }
        else error();
}
```

```
void term() {
    match(int_literal);
}

void match(expectedToken) {
    if(token==expectedToken)
            getNextToken();
    else error();
}
```

Outline

- Recap
- A simple syntax-directed translator (Chapter 2)
  - Parsing (Section 2.4.5)
  - Syntax directed translation (Section 2.3)
  - A translator for simple expressions (Section 2.5)
- Summary and homework

Remove Left Recursion

- What are the languages defined by the following two BNF grammars?

$A \rightarrow A\ \alpha \mid \beta$	$A \rightarrow \beta\ R$
	$R \rightarrow \alpha\ R \mid \varepsilon$

Remove Left Recursion

- BNF: **<expr> $\rightarrow$ <expr> + <term>**
- Left-recursion to right-recursion

$A \rightarrow A\ \alpha \mid \beta$	$A \rightarrow \beta\ R$
	$R \rightarrow \alpha\ R \mid \varepsilon$

  - <expr> $\rightarrow$ <term> rest
  - rest $\rightarrow$ + <term> rest | - <term> rest | ε
- EBNF: **<expr> $\rightarrow$ <term> { + <term>}**

Outline

- Recap
- A simple syntax-directed translator (Chapter 2)
  - Parsing (Section 2.4.5)
  - Syntax directed translation (Section 2.3)
  - A translator for simple expressions (Section 2.5)
- Summary and homework

What Can Be Done So Far?

- Define language syntax using BNF grammar
- Parsing to detect syntax errors
  - Syntax analysis
- How about translation?
  - Syntax-directed translation
    - Attaching rules or program fragments to productions in a grammar
  - An example of translating infix notation to postfix notation

Postfix Notation

- Inductive definition
  - If E is a variable or constant, then the postfix notation for E is E itself
  - If E is an expression of the form E1 op E2, then the postfix notation for E is E1’ E2’ op
  - If E is of the form (E1), then the postfix notation is E1’
  - Examples
    - The postfix notation (9-5)+2 is 95-2+

Syntax-Directed Definition

- For a BNF grammar
  - Associate each grammar symbol (terminals and non-terminals) with a set of attribute
    - Type information for type checking/conversion
    - Notation representation for notation translation
  - Attach a semantic rule or program fragment to each production in a grammar
    - Computing the values of the attributes associated with the symbols in the production
- The BNF grammar becomes an attribute grammar

Definition of Attribute Grammar

- An attribute grammar is a BNF grammar with additions:
  - For each grammar symbol x: a set A(x) of attribute values
  - Each production in the grammar has a set of semantic rules that define or compute certain attributes of the nonterminals in the production
  - Each production in the grammar has a (possibly empty) set of predicates to check for attribute consistency
- A sentence derivation

Based on BNF

A parse tree

Based on an attribute grammar

A fully attributed parse tree

A Type Checking Example Using Syntax-Directed Definition

- A BNF grammar
  - <assign> → <var> = <expr>
  - <expr> → <var> + <var>
  - <var> → A | B | C
- An attribute grammar
  1. Syntax production: <assign> → <var> = <expr>
     - Semantic rule: <expr>.expected_type ← <var>.actual_type
  2. Syntax production: <expr> → <var> + <var>
     - Semantic rule: <expr>.actual_type ←

       if(<var>[2].actual_type==int) and
(<var>[3].actual_type==int)
then int
else real
endif
     - Predicate: <expr>.actual_type == <expr>.expected_type
  3. Syntax production: <var> → A | B | C
     - Semantic rule: <var>.actual_type ← lookup(<var>.string)

Computing Attribute Values

- Let $X_0 \rightarrow X_1 \ldots X_n$ be a production
  - If the computing rule of X_0's attribute is of the form $A(X_0) = f(A(X_1), \ldots, A(X_n))$
    - Synthesized attribute
  - If the computing rule of X_ji's attribute is of the form $A(X_j) = f(A(X_0), \ldots, A(X_i), \ldots, A(X_{j-1}))$, for i <= j <= n
    - Inherited attribute
- Intrinsic attributes are synthesized attributes of leaf nodes whose values are determined outside the parse tree

A Notation Translation Example Using Syntax-Directed Definition

- <expr> → <expr> + <term> | <expr> - <term> | <term>
- <term> → 0 | 1 | 2 | ... | 9

PRODUCTION	SEMANTIC RULES
$expr \to expr_1 + term$	$expr.t = expr_1.t \parallel term.t \parallel '+'$
$expr \to expr_1 - term$	$expr.t = expr_1.t \parallel term.t \parallel '-'$
$expr \to term$	$expr.t = term.t$
$term \to 0$	$term.t = '0'$
$term \to 1$	$term.t = '1'$
...	...
$term \to 9$	$term.t = '9'$

Tree Traversals

- Perform depth-first traversal of the parse tree to generate a fully attributed parse tree

```
procedure visit(node N) {
    for (each child C of N, from left to right) {
        visit(C);
    }
}
```

Translation Schemes

- We used semantic rules as a translation scheme
- Now we use semantic actions as a translation scheme to get the same translation result

- Syntax-directed definition for a BNF grammar
  - Associate each grammar symbol (terminals and non-terminals) with a set of attribute
    - Type information for type checking/conversion
    - Notation representation for notation translation
  - Attach a semantic rule or program fragment to each production in a grammar
    - Computing the values of the attributes associated with the symbols in the production

New BNF Productions and Parse Trees Using Semantic Actions

- Actions are added in the productions

$expr$	$\rightarrow$	$expr_1 + term$	$\{\text{print}('+')\}$
$expr$	$\rightarrow$	$expr_1 - term$	$\{\text{print}('-')\}$
$expr$	$\rightarrow$	$term$	
$term$	$\rightarrow$	0	$\{\text{print}('0')\}$
$term$	$\rightarrow$	1	$\{\text{print}('1')\}$
		$\cdots$	
$term$	$\rightarrow$	9	$\{\text{print}('9')\}$

- When drawing a parse tree
  - Indicate an action by constructing an extra child for it, connected by a dashed line to the node that corresponds to the head of the production

Actions Translating 9-5+2 into 95-2+

- Perform a postorder traversal of the parse tree

Outline

- Recap
- A simple syntax-directed translator (Chapter 2)
  - Parsing (Section 2.4.5)
  - Syntax directed translation (Section 2.3)
  - A translator for simple expressions (Section 2.5)
- Summary and homework

What Can Be Done Now?

- Define language syntax using BNF grammar
- Parsing to detect syntax errors
  - Syntax analysis
- Syntax-directed translation

Define A Language and Syntax-Directed Translation

- 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') }

Top-Down Parsing

$A \rightarrow A\ \alpha \mid \beta$

$A \rightarrow \beta\ R$
$R \rightarrow \alpha\ R \mid \varepsilon$

- 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') }

Implementing Parsing and Translation

- expr →
  term rest
- rest →
  + term { print('+') } rest
  | - term { print('-') } rest
  | ε
- term →
  0 { print('0') }
  | 1 { print('1') }
  | ...
  | 9 { print('9') }

```
void expr() {
        term(); rest();
}

void rest() {
        if ( lookahead == '+' ) {
                match('+'); term(); print('+'); rest();
        }
        else if ( lookahead == '-' ) {
                match('-'); term(); print('-'); rest();
        }
        else { } /* do nothing with the input */ ;
}

void term() {
        if ( lookahead is a digit ) {
                t = lookahead; match(lookahead); print(t);
        }
        else report("syntax error");
}
```

Outline

- Recap
- A simple syntax-directed translator (Chapter 2)
  - Parsing (Section 2.4.5)
  - Syntax directed translation (Section 2.3)
  - A translator for simple expressions (Section 2.5)
- Summary and homework

You should now be able to ...

- Remove left recursions in BNF;
- Describe syntax-directed definition and attribute grammar;
- Implement a simple language.

Remove Left Recursion

$$A \rightarrow A\ \alpha \mid A\ \beta \mid \gamma$$

$$A \rightarrow \gamma\ R$$

$$R \rightarrow \alpha\ R \mid \beta\ R \mid \varepsilon$$

You should now be able to ...

- Remove left recursions in BNF;
- Describe syntax-directed definition and attribute grammar;
- Implement a simple language.

Syntax-Directed Definition

- For a BNF grammar
  - Associate each grammar symbol (terminals and non-terminals) with a set of attribute
    - Type information for type checking/conversion
    - Notation representation for notation translation
  - Attach a semantic rule or program fragment to each production in a grammar
    - Computing the values of the attributes associated with the symbols in the production
- The BNF grammar becomes an attribute grammar

You should now be able to ...

- Remove left recursions in BNF;
- Describe syntax-directed definition and attribute grammar;
- Implement a simple language.

Implement A Simple Language

- Define language syntax using BNF grammar
- Parse sentences and detect syntax errors
- Use syntax-directed definition to perform language translation

Homework (Due on 01/29 at 11:55 PM)

- 2.1. (20 points)
  - (a) Define a BNF grammar for a language that could express a one-digit number, additions and/or subtractions of multiple one-digit numbers in a prefix notation (e.g., -xy is the prefix notation for x-y and the prefix notation of an infix notation 4+5-2+6 is +-+4526); (5 pts)
  - (b) Construct a syntax-directed translation scheme that translates the above-defined one-digit arithmetic expressions from prefix notation into infix notation; (5 pts)
  - (c) Implement an executable and correct program to perform the above-mentioned translation. (10 pts)