CSCI 3336 Organization of Programming Languages

DESCRIBING SYNTAX

Topics

- Introduction
- The General Problem of Describing Syntax
- Formal Methods of Describing Syntax
- Context-Free grammars

Introduction

- **Syntax**: the form or structure of the expressions, statements, and program units
- **Semantics**: the meaning of the expressions, statements, and program units
- Syntax and semantics provide a language's definition
  - Example

```
while (<boolean_expr>) <statement>
```


The General Problem of Describing Syntax

- A *sentence* is a string of characters over some alphabet
- A *language* is a set of sentences
- A *lexeme* is the lowest level syntactic unit of a language (e.g., `*`, `sum`, `begin`)
- A *token* is a category of lexemes (e.g., identifier)

Example: Lexemes and tokens

- Consider the following Java statement:

```
index = 2 * count + 17;
```

Lexemes	*Tokens*
index	identifier
=	equal_sign
2	int_literal
*	mult_op
count	identifier
+	plus_op
17	int_literal
;	semicolon

Formal Definition of Languages

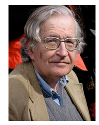
- **Recognizers**
  - A recognition device reads input strings of the language and decides whether the input strings belong to the language
  - Example: syntax analysis part of a compiler
- **Generators**
  - A device that generates sentences of a language
  - One can determine if the syntax of a particular sentence is correct by comparing it to the structure of the generator

Formal Methods of Describing Syntax

- Backus-Naur Form and Context-Free Grammars
  - Most widely known method for describing programming language syntax
- Extended BNF
  - Improves readability and writability of BNF

7

BNF and Context-Free Grammars



Noam Chomsky

- Context-Free Grammars
  - Developed by Noam Chomsky in the mid-1950s
- $G=(V,T,P,S)$
  - » V and T are disjoint finite sets of *variables* and *terminals*
  - » P is a finite set of productions of the from $A \to \alpha$, where A is a variable and α is a string of symbols from $(V \cup T)^*$
  - » Finally, S is a special variable called the *start symbol.*
  - Language generators
  - Define a class of languages called context-free languages

8

Backus-Naur Form (BNF)

- Backus-Naur Form (1959)
  - Invented by John Backus to describe Algol 58
  - BNF is equivalent to context-free grammars
  - BNF is a *metalanguage* used to describe another language
  - In BNF, abstractions are used to represent classes of syntactic structures--they act like syntactic variables (also called *nonterminal symbols*)

John Backus

Peter Naur

9

Context-Free Grammar

- Context-free grammars are powerful enough to describe the syntax of most programming languages; in fact, the syntax of most programming languages is specified using context-free grammars.
- On the other hand, context-free grammars are simple enough to allow the construction of efficient parsing algorithms which, for a given string, determine whether and how it can be generated from the grammar.

Grammars

- Grammars express languages
- Example: The English language grammar

$$\langle sentence \rangle \to \langle noun\_phrase \rangle \ \langle predicate \rangle$$

$$\langle noun\_phrase \rangle \to \langle article \rangle \ \langle noun \rangle$$

$$\langle predicate \rangle \to \langle verb \rangle$$

1-11

The English language grammar

$$\langle article \rangle \to a$$
$$\langle article \rangle \to the$$

$$\langle noun \rangle \to cat$$
$$\langle noun \rangle \to dog$$

$$\langle verb \rangle \to runs$$
$$\langle verb \rangle \to sleeps$$

1-12

Derivation of string "the dog sleeps"

$$
\begin{aligned}
\langle sentence\rangle &\Rightarrow \langle noun\_phrase\rangle\ \langle predicate\rangle \\
&\Rightarrow \langle noun\_phrase\rangle\ \langle verb\rangle \\
&\Rightarrow \langle article\rangle\ \langle noun\rangle\ \langle verb\rangle \\
&\Rightarrow the\ \langle noun\rangle\ \langle verb\rangle \\
&\Rightarrow the\ \ dog\ \ \langle verb\rangle \\
&\Rightarrow the\ \ dog\ \ sleeps
\end{aligned}
$$

1-13

Derivation of string "a cat runs"

$$
\begin{aligned}
\langle sentence\rangle &\Rightarrow \langle noun\_phrase\rangle\ \langle predicate\rangle \\
&\Rightarrow \langle noun\_phrase\rangle\ \langle verb\rangle \\
&\Rightarrow \langle article\rangle\ \langle noun\rangle\ \langle verb\rangle \\
&\Rightarrow a\ \ \langle noun\rangle\ \langle verb\rangle \\
&\Rightarrow a\ \ cat\ \ \langle verb\rangle \\
&\Rightarrow a\ \ cat\ \ runs
\end{aligned}
$$

1-14

Grammar for the English Language

G_L = (S, V, T, P), where
S = <sentence>
V = {<sentence>, <noun_phrase>, <predicate>, <article>, <noun>, <verb>}
T = {a, the, cat, dog, runs, sleeps}
P = {
<sentence> → <noun_phrase> <predicate>
<noun_phrase> → <article> <noun>
<predicate> → <verb>
<article> → a | the
<noun> → cat | dog
<verb> → runs | sleeps }

1-15

Language of grammar G_L

L = { " a cat runs",
"a cat sleeps",
"the cat runs",
"the cat sleeps",
"a dog runs",
"a dog sleeps",
"the dog runs",
"the dog sleeps" }

1-16

A grammar example for a small language L1

```
<program> → begin <stmts> end
<stmts> → <stmt> |
               <stmt> ; <stmts>
<stmt> → <var> = <expr>
<var> → A | B | C
<expr> → <var> + <var> |
          <var> - <var> |
          <var>
```

17

Derivation

- Every string of symbols in the derivation is a sentential form
- A sentence is a sentential form that has only terminal symbols
- A leftmost derivation is one in which the leftmost nonterminal in each sentential form is the one that is expanded

18

An example of derivation

Given the following program:

```
begin
A = B + C
end
```

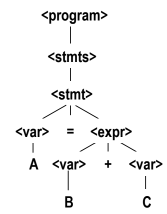
A derivation of the program in L1 follows:

```
<program>   => begin <stmts> end
            => begin <stmt> end
            => begin <var> = <expr> end
            => begin A = <expr> end
            => begin A = <var> + <var> end
            => begin A = B + <var> end
            => begin A = B + C end
```

19

Parse Tree

- A hierarchical representation of a derivation



20

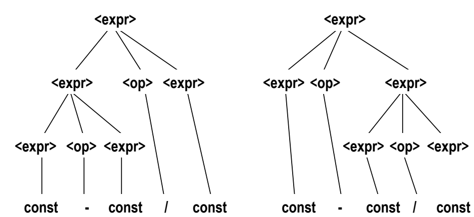
Ambiguity in Grammars

- A grammar is *ambiguous* if and only if it generates a sentential form that has two or more distinct parse trees

21

An Ambiguous Expression Grammar

```
<expr> → <expr> <op> <expr>  |  const
<op> → /  |  -
```

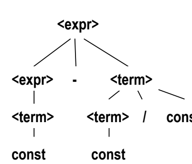


22

An Unambiguous Expression Grammar (cont'd)

- If we use the parse tree to indicate precedence levels of the operators, we cannot have ambiguity

```
<expr> → <expr> - <term>  |  <term>
<term> → <term> / const| const
```

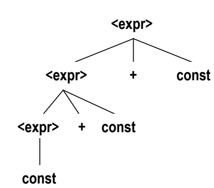


23

Associativity of Operators

- Operator associativity can also be indicated by a grammar

```
<expr> -> <expr> + <expr> |  const  (ambiguous)
<expr> -> <expr> + const  |  const  (unambiguous)
```



24

Extended BNF

- Optional parts are placed in brackets [ ]

```
<proc_call> -> ident [(<expr_list>)]
```

- Alternative parts of RHSs are placed inside parentheses and separated via vertical bars

```
<term> -> <term> (+|-) const
```

- Repetitions (0 or more) are placed inside braces { }

```
<ident> -> letter {letter|digit}
```

25

BNF and EBNF

- BNF

```
<expr> → <expr> + <term>
             | <expr> - <term>
             | <term>
 <term> → <term> * <factor>
             | <term> / <factor>
             | <factor>
```

- EBNF

```
<expr> → <term> {(+ | -) <term>}
 <term> → <factor> {(* | /) <factor>}
```

26

Summary

- BNF and context-free grammars are equivalent meta-languages
  - Well-suited for describing the syntax of programming languages
- An attribute grammar is a descriptive formalism that can describe both the syntax and the semantics of a language
- Three primary methods of semantics description
  - Operation, axiomatic, denotational

27