Extended BNF

Objectives

+ Discuss the most common method of describing syntax - context-free grammars (also known as Backus-Naur Form). This includes a discussion of derivations, parse trees, ambiguity, descriptions of operator precedence and associativity, and extended Backus-Naur Form.

1. The Extended BNF

– Because of a few minor inconveniences in BNF, it has been extended in several ways. Most extended versions are called Extended BNF, or simply EBNF, even though they are not all exactly the same.
– The extensions do not enhance the descriptive power of BNF; they only increase its readability and writability. Three extensions are commonly included in the various versions of EBNF.
– The first of these denotes an optional part of an RHS, which is delimited by brackets. For example, a C if-else construct can be described as

```
<if_stmt> → if (<expression>) <statement> [else <statement>]
```

– Without the use of the brackets, the syntactic description of this statement would require the following two rules:

```
<if_stmt> → if (<expression>) <statement>
          |if (<expression>) <statement> else <statement>
```

– The second extension is the use of braces in a RHS to indicate that the enclosed part can be repeated indefinitely or left out altogether. This extension allows lists to be built with a single rule, instead of using recursion and two rules.

– For example, lists of identifiers separated by commas can be described by the following rule:

```
<ident_list> → <identifier> {, <identifier>}
```

– This is a replacement of the recursion by a form of implied iteration; the part enclosed within braces can be iterated any number of times.
– The third common extension deals with multiple-choice options. When a single element must be chosen from a group, the options are placed in parentheses and separated by the OR operator, |.
– For example,

```
<term> → <term> (* | / | %) <factor>
```

– In BNF, a description of this <term> would require the following three rules:

```
<term> → <term> * <factor>
       | <term> / <factor>
       | <term> % <factor>
```

– The brackets, braces, and parentheses in the EBNF extensions are **metasymbols**, which means they are notational tools and not terminal symbols in the syntactic entities they help describe.
– In cases where these metasymbols are also terminal symbols in the language being described, the instances that are terminal symbols can be underlined or quoted. Example 1 illustrates the use of braces and multiple choices in an EBNF grammar.

Example 1 BNF and EBNF Versions of an Expression Grammar

BNF:

```
<expr> → <expr> + <term>
       | <expr> - <term>
       | <term>
<term> → <term> * <factor>
       | <term> / <factor>
       | <factor>
<factor> → <exp> ** <factor>
           <exp>
<exp> → (<expr>)
      | id
```

EBNF:

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

– The BNF rule

```
<expr> → <expr> + <term>
```

clearly specifies—in fact forces—the + operator to be left associative. However, the EBNF version,

```
<expr> → <term> {+ <term>}
```

does not imply the direction of associativity.

– This problem is overcome in a syntax analyzer based on an EBNF grammar for expressions by designing the syntax analysis process to enforce the correct associativity.
– Some versions of EBNF allow a numeric superscript to be attached to the right brace to indicate an upper limit to the number of times the enclosed part can be repeated.
– Also, some versions use a plus (+) superscript to indicate one or more repetitions. For example,

```
<compound> → begin <stmt> {<stmt>} end
```

– and

```
<compound> → begin {<stmt>}+ end
```

– are equivalent.
–
– In recent years, some variations on BNF and EBNF have appeared. Among these are the following:
    - In place of the arrow, a colon is used and the RHS is placed on the next line.
    - Instead of a vertical bar to separate alternative RHSs, they are simply placed on separate lines.
    - In place of square brackets to indicate something being optional, the subscript opt is used. For example,

- Constructor `Declarator → SimpleName (FormalParameterList`$_{opt}$`)`
- Rather than using the | symbol in a parenthesized list of elements to indicate a choice, the words “one of” are used. For example,

```
AssignmentOperator → one of = *= /= %= += -=
                            <<= >>= &= ^= |=
```

– There is a standard for EBNF, ISO/IEC 14977:1996 (1996), but it is rarely used. The standard uses the equal sign (=) instead of an arrow in rules, terminates each RHS with a semicolon, and requires quotes on all terminal symbols. It also specifies a host of other notational rules.

2. Grammars and Recognizers

– Earlier on, we suggested that there is a close relationship between generation and recognition devices for a given language. In fact, given a context-free grammar, a recognizer for the language generated by the grammar can be algorithmically constructed.

– A number of software systems have been developed that perform this construction. Such systems allow the quick creation of the syntax analysis part of a compiler for a new language and are therefore quite valuable.

– One of the first of these syntax analyzer generators is named yacc (*y*et *a*nother *c*ompiler *c*ompiler). There are now many such systems available.