EECS 368 Programming Language Paradigms

Dr. Andy Gill

Department of Electrical Engineering & Computer Science University of Kansas

September 8, 2015



Syntax for Languages

```
C++
#include <iostream.h>
main()
{
    cout << "Hello World!";
    return 0;
}</pre>
```

Java

JavaScript

console.log("Hello, World");

Scheme

(display "Hello World!")
(newline)

Haskell

```
main :: IO ()
main = putStrLn "Hello World!"
```



```
\verb"java ::= public class name { methods } \\
```

java ::= public class name { methods }

```
java ::= public class name { methods }
 methods ::= method methods
 method ::= public static void name ( method_args ) { statement ; }
 method_args ::= non_empty_method_args
non_empty_method_args ::= method_arg
                       | method_arg , non_empty_method_args
```

```
java ::= public class name { methods }
methods ::= method methods
method ::= public static void name ( method_args ) { statement ; }
method_args ::= non_empty_method_args
non_empty_method_args ::= method_arg
                       | method_arg , non_empty_method_args
method_arg ::= ...
```

Summary

BNF is a way of expressing valid programs.

BNF can express arbitrary sized sequences.

Real languages can have large BNF's.





Examples of Comma-Separated Sequences


```
Zero or More
seq ::= m_seq
m_seq ::= number
        | m_seq , m_seq
seq ::= m_seq
m_seq ::= number
        | number , m_seq
seq ::= m_seq
m_seq ::= number
        | m_seq , number
```

Another Comma-Separated Sequence

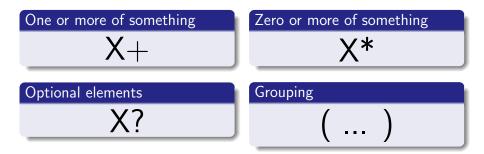
There are other ways of representing sequences

```
An alternative right leaning
seq ::= numbers number
numbers ::= number , numbers
```

What is the common pattern? How many more are there? We should be able to think in terms of "What does this BNF accept?"

EBNF - Extended BNF

There are some understood extensions, called EBNF.



? Optional

Optional elements

X?

X can be a terminal, non-terminal, or a parenthesized sequence of terminals and non-terminals.

Sometimes written [X]

BNF

abc ::= xyz vwx www | xyz www

EBNF

abc ::= xyz vwx? www

? More Optional

BNF

```
seq ::= m_seq
```

EBNF

```
seq := m_seq?
```

BNF

EBNF

```
seq ::= m_seq?
m_seq ::= number ( , m_seq )?
```



* Zero or More

Zero or More



X can be a terminal, non-terminal, or a parenthesized sequence of terminals and non-terminals.

Sometimes written { X }

BNF

```
seq ::= numbers number
numbers ::= number , numbers
|
```

EBNF

```
seq ::= numbers number
numbers ::= ( number , )*
```

EBNF

```
seq ::= ( number , )* number
```



+ One or More

One or More



X can be a terminal, non-terminal, or a parenthesized sequence of terminals and non-terminals.

BNF (no commas)

EBNF

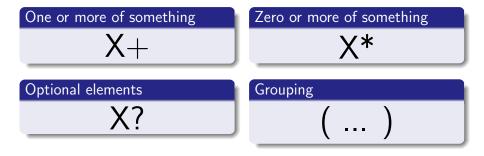
seq ::= number+

Linked List EBNF

seq ::= number seq?



EBNF - Extended BNF



New EBNF Grammar for Mini-Java

```
java ::= public class name { method* }
method ::= (public|private)
           static* type name ( method_args ) { statements }
method_args ::= method_arg , method_args
method_arg ::= type name
type ::= type [ ] | name | "int" | "float" | "boolean"
statements ::= ( statement ; )*
statement ::= full_name ( fun_args )
full_name ::= ( name . )* name
fun_args ::= expr , fun_args
expr ::= string | int | expr + expr | expr * expr | ( expr )
```

In Class

```
S ::= A S | A
A ::= A S | B
B ::= C S D | E
```

- What are the terminals and non-terminals for this grammar? (In class)
- Show that this grammar is ambiguous by giving a token sequence that has two possible concrete syntax trees (two derivations)
- Construct an unambiguous grammar to describe the same language, using BNF.
- Construct another unambiguous grammar to describe the same language, using EBNF.

(Adapted from Fundamental Structures of Computer Science, Wulf, Shaw, Hilfinger, Flon, pp 370)