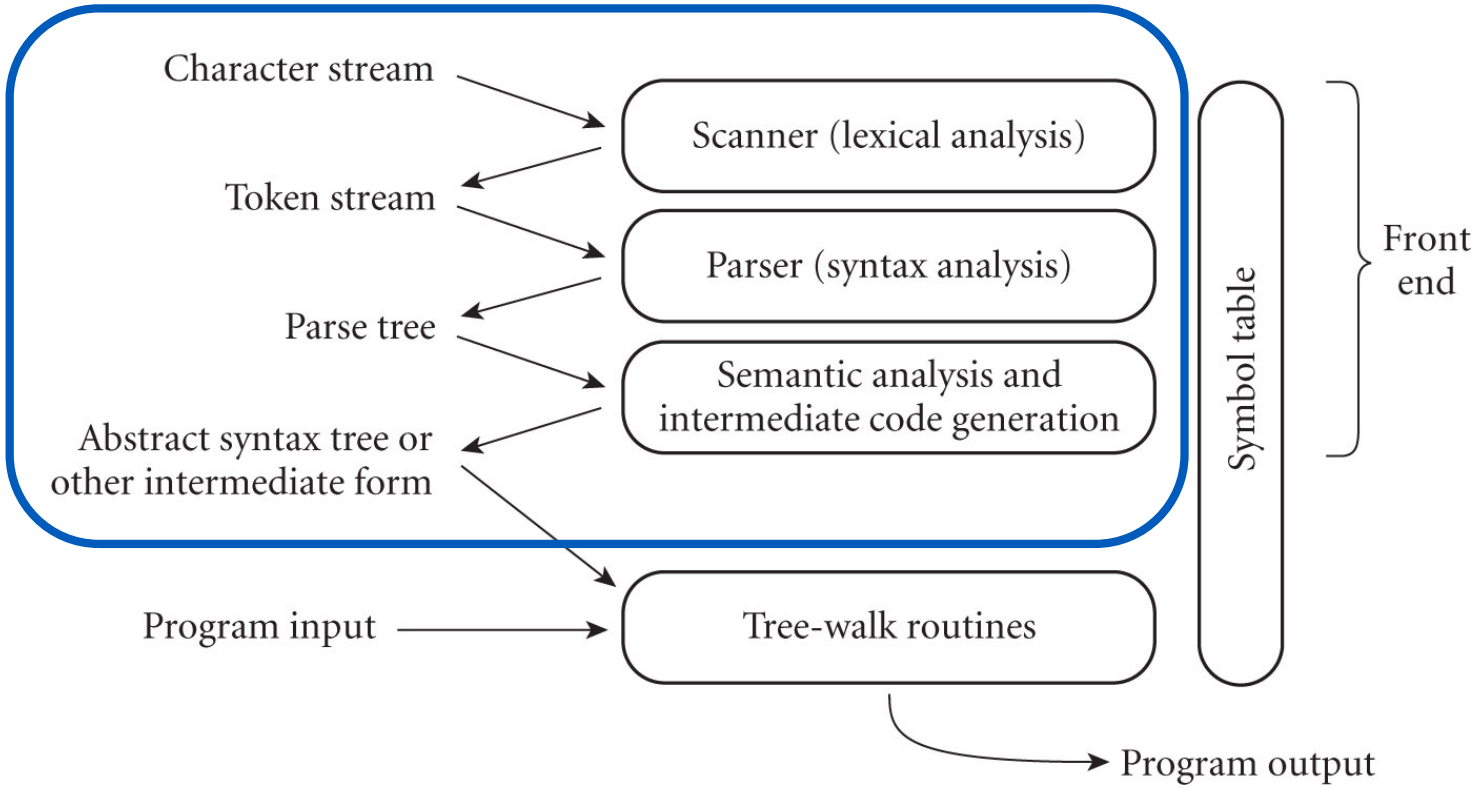
CS 320 : Formal Grammars

Marco Gaboardi
CDS 1019
gaboardi@bu.edu

Parsing and semantic analysis



BNF

- Here another example from a programming languages application.

```
<program> ::= <stmts>
<stmts> ::= <stmt> | <stmt> ; <stmts>
<stmt> ::= <var> = <expr>
<var> ::= a | b | c | d
<expr> ::= <term> + <term> | <term> - <term>
<term> ::= <var> | const
```

Generator vs Recognizer

```
<program> ::= <stmts>
<stmts> ::= <stmt> | <stmt> ; <stmts>
<stmt> ::= <var> = <expr>
<var> ::= a | b | c | d
<expr> ::= <term> + <term> | <term> - <term>
<term> ::= <var> | const
```

Recognize a sentence

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

Generate a sentence

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

Let's consider a simple one

```
<expression> ::= <term> + <term> | <term> - <term>
<term> ::= <var> | <const>
<var> ::= a | b | c | d
<const> ::= 0 | 1
```

How would we implement this in OCaml?

Let's consider a more difficult one

```
<expr> ::= (<expr>+<expr>) | <digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

How would we implement this in OCaml?

Is a BNF grammar specific enough for an interpreter to recognize sentences from a formal programming language?

Some of the challenges:

- There is a (potentially) infinite number of source programs that we need to recognize.
  - An infinity of words
  - An infinity of sentences
- There should be no ambiguity in the way the program is interpreted.
  - Unique vocabulary,
  - Uniquely determine sentences
- The source program may contain syntax errors and the compiler/interpreter has to recognize them.
  - Lexical errors (errors in the choice of words)
  - Grammatical errors (errors in the construction of sentences)

Is a BNF grammar specific enough for an interpreter to execute it?

Here a simple grammar for expressions:

```
<expr> ::= <expr> <op> <expr>
<expr> ::= 1|2|3|4|5|6|7|8|9|0
<op>   ::= +|-|*|/
```

How shall the interpreter/compiler execute the following expression?

```
2+3*4
```

This can be interpreted as

```
(2+3)*4
```

or as

```
2+(3*4)
```

Note: here the parenthesis are just to show the possible ambiguity, they are not part of the grammar.

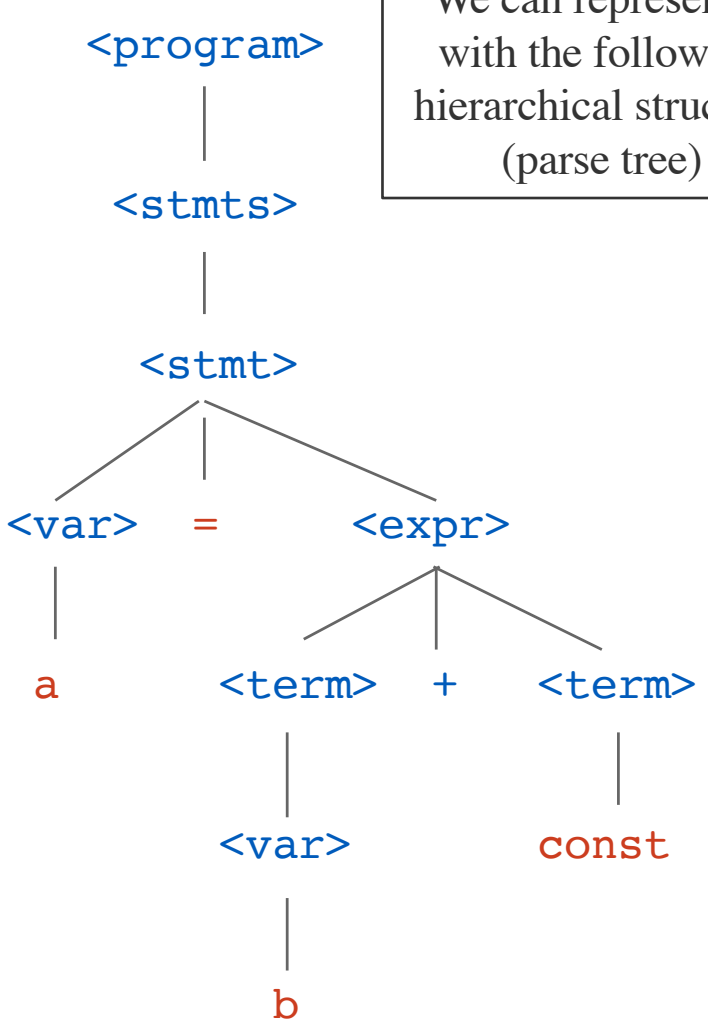
Parse Tree

- A parse tree is a hierarchical representation of a derivation

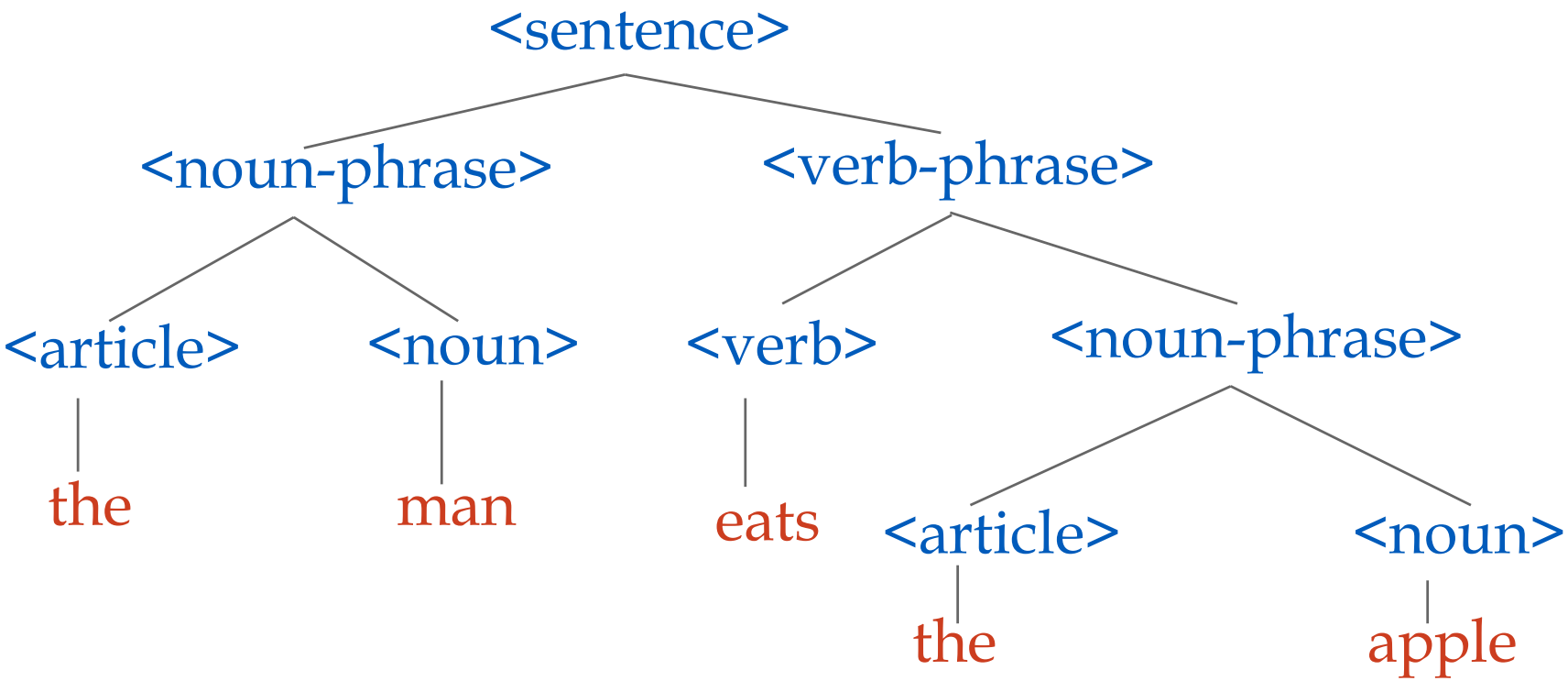
Suppose we have the following derivation

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

We can represent it with the following hierarchical structure (parse tree)



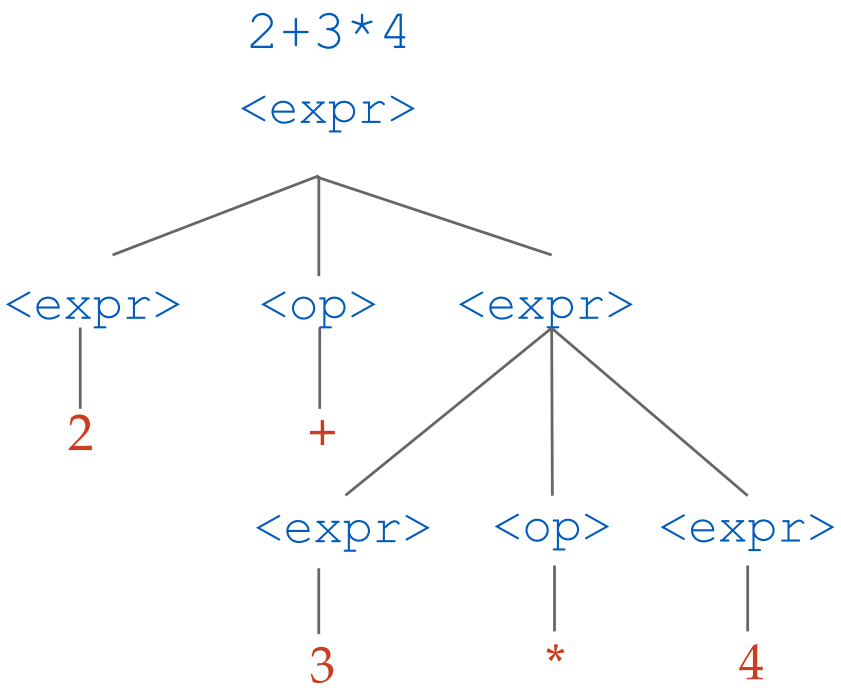
Parse Tree – another example

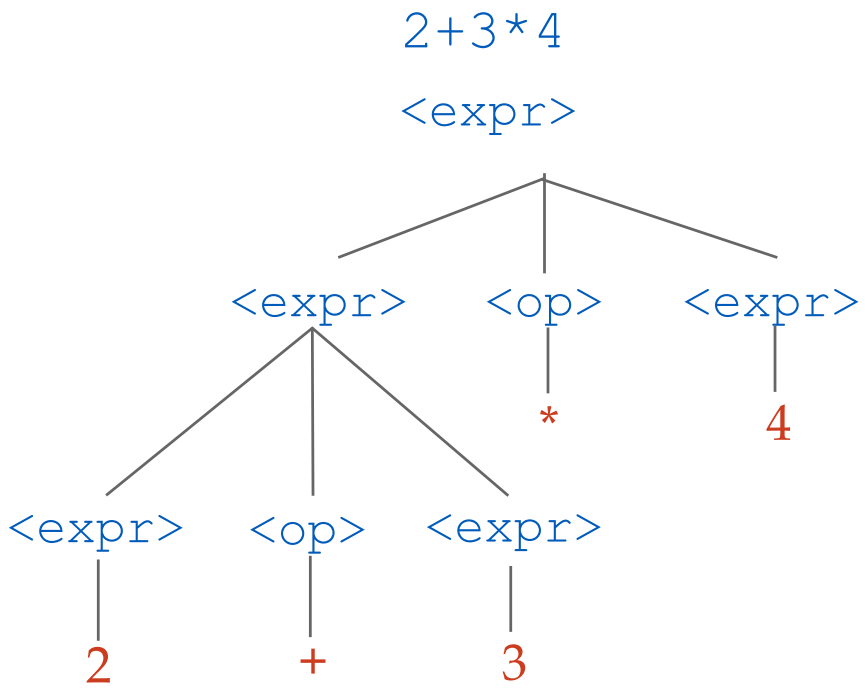


Ambiguous Grammars

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

```
<expr> ::= <expr> <op> <expr>
<expr> ::= 1|2|3|4|5|6|7|8|9|0
<op>   ::= +|-|*|/
```





Ambiguous Grammars

Ambiguous grammars are, in general, undesirable in formal languages.

Why?

It makes parsing difficult – and more error prone.

Ambiguity can have different sources.

Good news: we can usually eliminate the ambiguity by revising the grammar.