
Languages

More on languages:

- Notation for Grammar (BNF notation)
- Pictures for Grammar (rail road diagram)

How to build a lexical analyser

How to build a parser

Ways to define a grammar

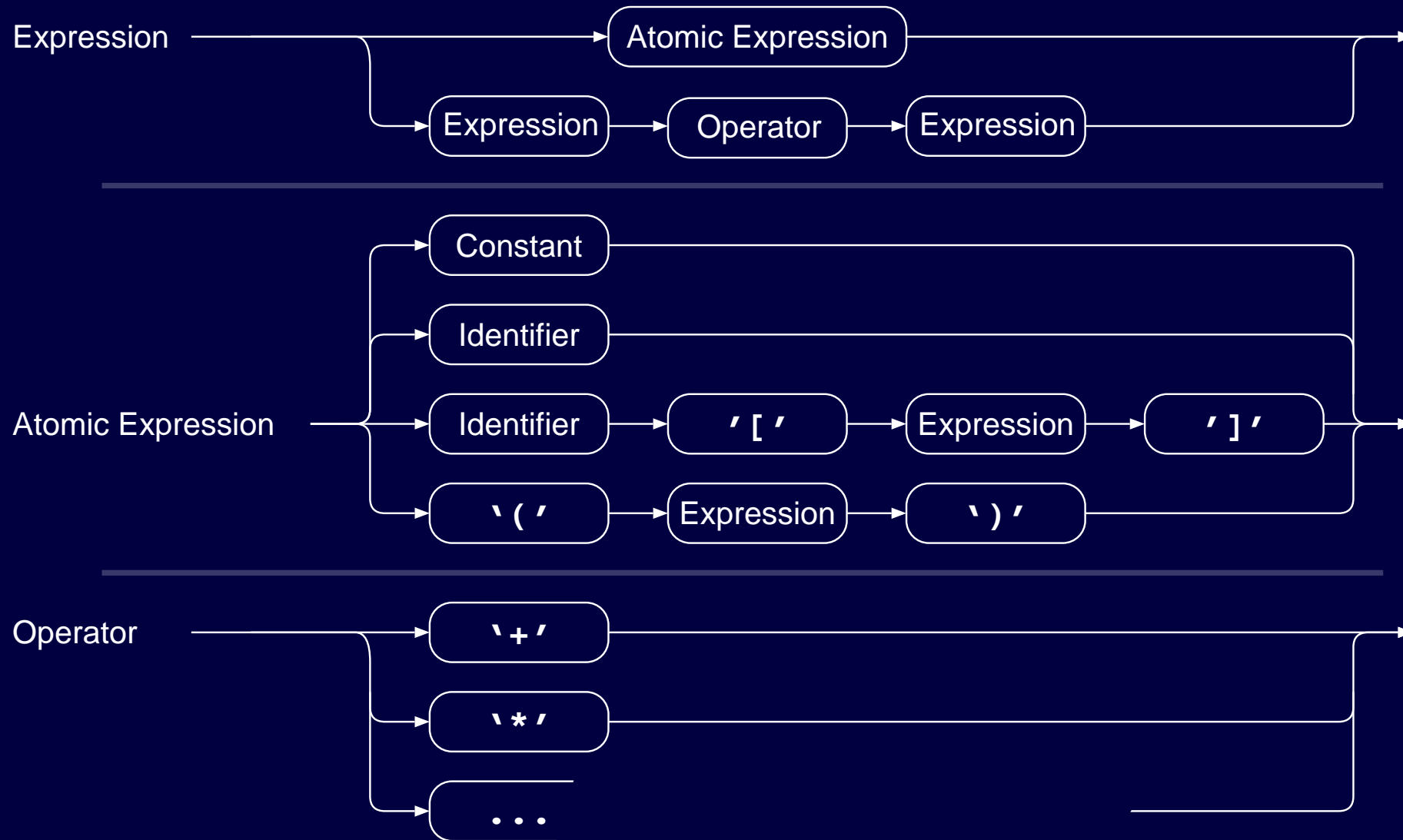
How do we specify computer languages?

1. Rail road diagrams
2. BNF Syntax

Rail road diagrams.

- Pictures
 - an X looks like this picture.
 - Follow the arrows
 - You are not allowed to go the wrong way
- There is one place to start.

Example: C expression



Notation for Grammar

Rail roads cannot be typed into computers.

- There is a language for describing languages
- Called Backus-Naur format, or BNF for short.

BNF:

- specifies production rules.
- A production rule says how to make a sentence, or another way to interpret it
- A production rule says how to parse a sentence.

BNF syntax

BNF syntax: no more than two symbols needed

- $X ::= Y$ An X consists of a Y .
- $X ::= Y \mid Z$ An X consists of a Y or a Z .
- $X ::= Z \mid X Y$ An X consists of a Z or an X followed by a Y
(recursion)

BNF example: C expressions

Expression ::=
 AtomicExpression
 | Expression Operator Expression

Operator ::= Plus | Star | ...

AtomicExpression ::=
 Constant
 | Identifier
 | Identifier LeftBracket Expression RightBracket
 | LeftParenthesis Expression RightParenthesis

Terminals, Non Terminals (one start symbol), BNF syntax. (2+3)*8
?

Terminals

Terminals are symbols recognised by the lexical analyser.

```
LeftParenthesis = '('  
RightParenthesis = ')'  
Constant = '[0-9]'  
Plus = '+'  
Minus = '-'  
Slash = '/'  
Star = '*'
```

Third one is known as a regular expression

- [] means any of the characters in between.

Non Terminals

Non terminals are produced by the language.

Expression

AtomicExpression

Operator

Expression is either an atomic expression or a thing with an operator.

Operator is either a plus, minus, star or slash.

You must have a place to start, the start symbol.

⇒ Expression in this case.

⇒ Program for C.

Parsing or producing a program

- A syntactically valid program can be produced by the BNF production rules
- A syntactically invalid program cannot be produced by the BNF production rules

- Lets take another look at an expression, say, $5 * (8 - 3)$.
 - Is it valid?
 - How is it produced?

Production

5 * (8-3) is an **Expression**

5 is a **Constant** (terminal)

* is an **Operator** (non-terminal)

* is an **star** (terminal)

(8-3) is an **Expression** (non terminal)

(8-3) is an **Atomic Expression** (non terminal)

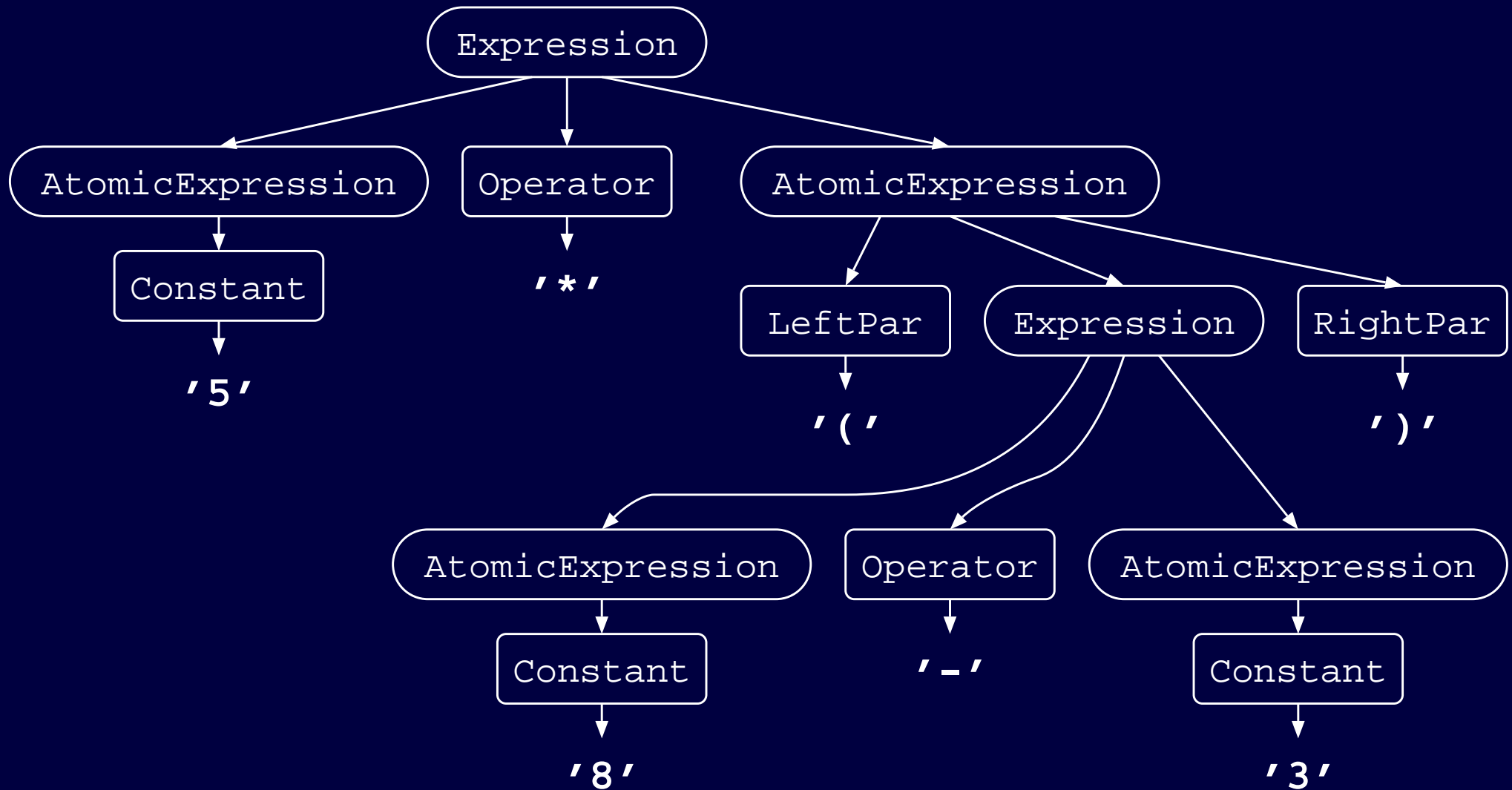
(is a **LeftParenthesis** (terminal)

8-3 is an **Expression** (non terminal)

) is a **RightParenthesis** (terminal)

...

Parse tree



Yet another tree

Ok, tree is upside down

- Most trees in computer science are upside down.
- Indeed, you can make a parser by calling functions.

We need

- a data type to represent “tokens” (words of the language, such as `]`, `while`, `19`)
- a function to read words, the lexical analyser.
- a function to string words together, the parser.

Data type for tokens

```
typedef enum {  
    LeftParenthesis, RightParenthesis, LeftBracket, Ri  
    Constant, Identifier, Plus, Star  
} Token ;
```

Lexer

```
Token readtoken() {
    c = getchar() ;
    switch( c ) {
        case '(': return LeftParenthesis ;
        case ')': return RightParenthesis ;
        case ' ': case '\t': case '\n': return readtoken()
        case '0': case '1': case '2': case '3': case '4':
        case '5': case '6': case '7': case '8': case '9':
            do {
                c = getchar() ;
            } while( isdigit( c ) ) ;
            ungetc( c, stdin ) ; /* Woops, one too far */
            return Constant ;
    }
}
```

Parser

```
expression() {
    Token z = readtoken() ;
    if( z == LeftParenthesis ) {
        expression() ;
        z = readtoken() ;
        if( z != RightParenthesis ) {
            error( "Expected a ')'" ) ;
        }
    } else if( ... )
        ...
}
```

- COMS22100 deals with all the gory details, such as the lookahead.

Conclusions

How do we specify computer languages?

1. Rail road diagrams
2. BNF Syntax

Terminals

- Symbols with which the thing ends.
- Recognised by the lexical analyser
- Example: '+', 'monkey'

Non Terminals

- Symbols which are to be produced further

Start Symbol: a Non Terminal.