Language Engineering | Lecture 8

Parsing

Typically, languages are defined in terms of a grammar that describes the valid construction of sentences in that language.

The designers of Algol 58 invented BNF to describe the language (Backus-Naur Form)

eg.

```
<expr> ::= <term> | <expr> "+" <term>
```

things in "<" ">" brackets are non-terminals

things in quotes are terminals

this says that the LHS can be produced by the rules on the RHS

this gives us the choice between productions

```
<term> ::= <factor> | <term> "*" <factor>
<factor> ::= <constant> | <variable>
           | "(" <expression> ")"
```

```
<variable> ::= "x" | "y" | "z".

<constant> ::= <digit> | <digit><constant>

<digit> ::= "0" | "1" | "2" | ... | "9".
```

BNF consists of

• non terminals	`<expr>`
• ~~[illegible]~~ terminals	"3"
• alternations	p | q

This was extended to Extended BNF or EBNF, which has a few more constructs:

• optionals :

```
<term> ::= ["-"] <factor>
               ↑
          optional "-".
```

In BNF we could write this as:

```
<term> ::= <factor>
         | "-" <factor>
```

• repetition :

<args> ::= <arg> { "," <arg> }

the {and} brackets tell us that this can be repeated 0 or more times.

• grouping :

<expr> ::= <term> ("+" | "-") <expr>

the (and) allow us to have scope for some BNF rules.

"~~[illegible]~~ " + 3 * 9"

"1 + 9 x 3"

<digit> " + 9 x 3 "

Parsing is traditionally broken up into :

* lexical analysis
* parsing.

* lexical analysis breaks up an input stream into <u>tokens</u>.
  The tokens correspond to terminals

  this stage usually deals with whitespace.
  Not all whitespace is necessarily stripped eg. newlines / tabs might turn into tokens.

  tools that did this phase are called <u>lexers</u> popular tools are lex and flex.

* A perser traditionally consumes tokens from the lexical analysis, and maps rules onto datatypes.
  Tools that do this are deeply tied into the language the compiler is written in.
  Traditionally, the tools bison, yacc, or (antlr java) were used for this

We will be using parser combinators instead. This builds up parsers from smaller one.

We will be looking at parsec.