Programming Languages
LECTURER

Joao Almeida-Domingues[*]

University of Glasgow

January 15th, 2021 – March 19th, 2021

Contents

These lecture notes were collated by me from a mixture of sources , the two main sources being the lecture notes provided by the lecturer and the content presented in-lecture. All other referenced material (if used) can be found in the *Bibliography* and *References* sections.

The primary goal of these notes is to function as a succinct but comprehensive revision aid, hence if you came by them via a search engine , please note that they're not intended to be a reflection of the quality of the materials referenced or the content lectured.

Lastly, with regards to formatting, the pdf doc was typeset in LaTeX, using a modified version of Stefano Maggiolo's class

[*]2334590D@student.gla.ac.uk

1 Syntax

syntax , specification , BNF , EBNF , Regex , terminal, nonterminal, sentence symbols , production rule , syntax tree , grammar , phrase

1.1 definition. Syntax concerns the *form* of the program, i.e how constructs of the PL (e.g expressions) need to be arranged to make a *well-formed* program

1.2 definition. Semantics has to do with the *meaning* of the program, i.e its behaviours when run

1.3 definition. Pragmatics how the language is to be used in practice

Syntax is important because it influences how programs are *written* by the programmer, *read* by other programmers, and *parsed* by the computer. On the whole scheme of things though, the other two are more important since they weigh more heavily on *how* a programmer goes about writing her program. Semantics determines how programs are composed by the programmer, understood by other programmers, and interpreted by the computer. Pragmatics influences how programmers are expected to design and implement programs in practice.

it's the content of the message which is important, not the language in which it is written sort of thing

1.1 *Specification*

The syntax of a language must be specified , this can either be done *informally* or *formally* the obvious trade-offs are between accessibility and practicality vs precision and consistency

1.4 example. Informal

A while-command consists of `'while'`, followed by an expression enclosed in parentheses, followed by a command.

1.5 example. Formal

```
while-command = 'while' '(' expression ')' command
```

Formal Specification Notations

REs are good for specifying the syntax of lexical elements of programs (e.g identifiers)

Backus Naur Form (BNF) works well when specifying larger and nested program constructs (e.g expressions, commands)

Extended BNF (EBNF) combines RE and BNF

1.2 *Grammar*

1.6 definition. Grammar describes how to form strings from a language's alphabet that are valid according to the language's syntax

1.7 definition. Context-Free Grammar grammar in which every *production rule* is of the form $A \to \alpha$ where A is a single *nonterminal* symbol, and α is a string of terminals and/or (α can be empty)

1.8 definition. Production Rule rules specifying a symbol substitution that can be recursively performed to generate new symbol sequences .

1.9 example. $\langle \text{Stmt} \rangle \to \langle \text{Id} \rangle = \langle \text{Expr} \rangle$ replaces $\langle \text{Stmt} \rangle$ by $\langle \text{Id} \rangle = \langle \text{Expr} \rangle$

1.10 definition. Terminal Symbols are literal symbols which occur in the language and may show up in the output of the rules but cannot be changed by them, e.g `a`

1.11 definition. Nonterminal symbols are symbols which can be replaced like $\langle \text{Stmt} \rangle$

1.12 definition. Sentence Symbol the nonterminal symbol that stands for a *complete* sentence

We use terminal and nonterminal symbols to specify the production rules.

1.3 *BNF*

BNF is a notation for expressing a grammar, where each element α, β, λ is a sequence of terminal and nonterminal symbols

$$N = \alpha \mid \beta \mid \lambda$$

1.13 example. mini-English

`subject = 'I' | 'a' noun | 'the' noun` with `'I'`, `'a'`, `'the'` being terminal symbols and `noun` nonterminal

1.14 example. Calc

- variables : `[a ... z]`
- expressions made of variables , numerals and arithmetic operators
- assignment and output commands
- terminal : `'put' 'set' '=' '+' '-' '*' '(' ')' '\n' 'a' 'b' 'c' ... 'z' '0' '1' ... '9'`
- nonterminal : `prog expr num com prim id`
- sentence : `prog` (a sequence of 0+ `coms` followed by an `EOF`)
- production rules (two examples)

see lecture 1@28 for full spec

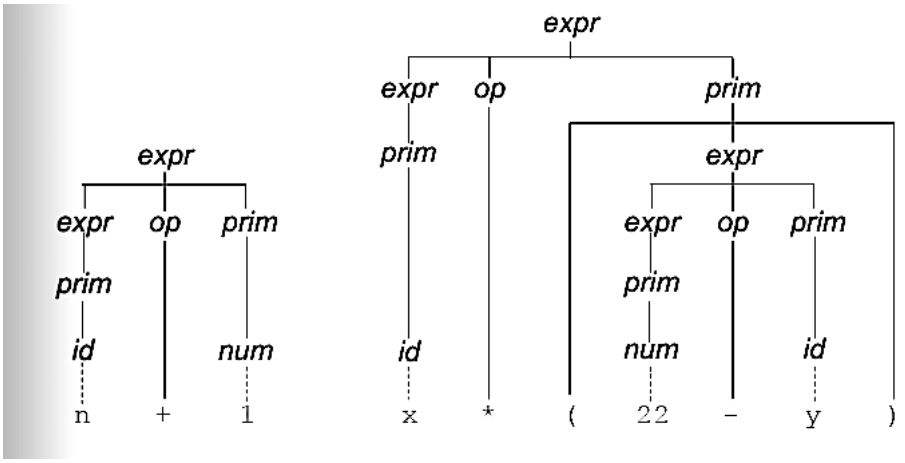
```
expr = prim                          prim = num
     | expr ‘+‘ prim                       | id
     | expr ‘-‘ prim                       | ’(’ expr ’)’
     | expr ‘*‘ prim
```

1.4 *Phrase Structure & Syntax Trees*

1.15 definition. Phrase Structure how phrases may be formed from subphrases

Every phrase in the language has a *syntax tree* that explicitly represents its phrase structure. For a grammar G a syntax tree of G is a tree with the following properties:

- Every terminal node is labeled by a terminal symbol of G
- Every nonterminal node is labeled by a nonterminal symbol of G
- A nonterminal node labeled N may have children labeled X, Y, Z only if G has a production rule $N = X\ Y\ Z$ or $N = \ldots | X\ Y\ Z | \ldots$



References

Watt et al.: Programming language design concepts **watt˙2004**

David Anthony Watt and William Findlay. *Programming language design concepts*. Wiley, 2004.