Programming Languages
UCLA-CS131-S18

Quentin Truong
Taught by Professor Eggert

Spring 2018

Contents

1 Ch1: Programming Languages **2**
1.1 Introduction . . . 2
1.2 The Amazing Variety . . . 2
1.3 The Odd Controversies . . . 2
1.4 The Intriguing Evolution . . . 2
1.5 The Many Connections . . . 2
1.6 A Word about Application Programming Interfaces . . . 3

2 Ch2: Defining Program Syntax **4**
2.1 Introduction . . . 4
2.2 A Grammar Example for English . . . 4
2.3 A Grammar Example for a Programming Language . . . 4
2.4 A Definition of Grammars: Backus-Naur Form . . . 4
2.5 Writing Grammars . . . 4
2.6 Lexical Structure and Phrase Structure . . . 4
2.7 Other Grammar Forms . . . 5
2.8 Conclusion . . . 5

3 Ch3: Where Syntax Meets Semantics **6**
3.1 Introduction . . . 6
3.2 Operators . . . 6
3.3 Precedence . . . 6
3.4 Associativity . . . 6
3.5 Introduction . . . 6
3.6 Introduction . . . 6
3.7 Introduction . . . 6
3.8 Introduction . . . 7
3.9 Introduction . . . 7

4 Ch1: Programming Languages **8**
4.1 Introduction . . . 8

1 Ch1: Programming Languages

1.1 Introduction

- Practical Magic
  - Useful and beautiful
- Programming Languages
  - ML, Java, Prolog

1.2 The Amazing Variety

- Imperative Languages (C)
  - Hallmarks: assignment and iteration
- Functional Languages (ML, Lisp)
  - Hallmarks: recursion and single-valued variables
  - Factorial is natural to functional
- Logic Programming Languages (Prolog)
  - Express program in terms of rules about logical inferences and proving things
  - Factorial is very not natural to logic programming; not well suited to mathematical functions
- Object-oriented Programming Languages (Java)
  - Object is a bundle of data which knows how to do things to itself
  - Helps keep large programs organized
- Other categories
  - Applicative, concurrent, constraint, declarative, definitional, procedural scripting, single-asstgnmen
- Multi-paradigm
  - JavaScript, OCaml, Python, Ruby
- Others
  - FORTH is stack-oriented
  - APL is a unique functional language relying on large character sets with many symbols that most users don't have

1.3 The Odd Controversies

- Partisans
  - For every language b/c some advantages
  - But all languages have advantages and disadvantages
  - Disagreement even on basic terminology, like object oriented

1.4 The Intriguing Evolution

- Programming languages change
  - All change; new ones evolve from old ones
  - Many have several dialects
  - Fortran is entirely only dialects (sequence of standards)

1.5 The Many Connections

- Styles
  - Object Oriented, like Java -¿ objects
  - Functional, like ML -¿ many small functions
  - Logic, like Prolog -¿ express problem as searches in logically defined space of solutions
- Language evolution driven by hardware + applications
  - AI encouraged Lisp; Classes bc Simula

1.6 A Word about Application Programming Interfaces

– Application Programming Interfaces (API)
  - May implement data structures, GUI, network input/output, encryption, security, other services
  - Is much of language; more than the printed specification of the language

2 Ch2: Defining Program Syntax

2.1 Introduction

- Syntax
  - Language definition that says how programs look (form and structure)
  - Appearance, delimiters, etc
- Semantics
  - language definition that says what programs do (behavior and meaning)
  - How it works, what can go wrong, etc
- Formal grammar
  - Used to define programming language syntax

2.2 A Grammar Example for English

- English
  - Article, noun, noun phrase, verb, sentence composes subset of unpunctuated English
  - Grammar used as set of rules that say how to build a parse tree (sentence at root)
  - Language defined by grammar is set of all strings that can be formed as fringes of parse trees

2.3 A Grammar Example for a Programming Language

- Infinite language have arbitrarily long expressions
  - Recursive grammar where exprsesions can be children of expressions
    - Expressions can be sum/product/enclosed/variable of two expressions

2.4 A Definition of Grammars: Backus-Naur Form

- Tokens
  - Smallest units of syntax
  - Strings and symbols not consisting of smaller parts (cat, if, !=)
- Non-terminal symbols
  - Correspond to different language constructs (sentences, noun pharses, statements)
  - Special non terminal symbol <empty¿
- Productions
  - Possible way of building parse tree
  - LHS is non-terminal; RHS is sequence of one or more things
- Start symbol
  - Special non-terminal symbol
- <if-stmt >::= if <expr >then <stmt >else-part >
  - <else-part >::= else <stmt >— <empty >

2.5 Writing Grammars

- Divide and Conquer
  - <var-dec >::= <type-name ><declarator-list >;
  - <declarator-list >::= <declarator >— <declarator >, <declarator-list >
  - BNF syntax defines programming language constructs

2.6 Lexical Structure and Phrase Structure

- Lexical Structure
  - How to divide program text into tokens
- Phrase Structure
  - How to construct parse trees with tokens at leaves

- Separate lexical and phrase structure
  - Otherwise, is ugly, hard to read, and complicated
- Lexer
  - Reads input file and converts to stream of tokens, discarding white space and comments
- Parser
  - Reads stream of tokens and forms parse tree
- Free-format languages
  - End-of-line is no more special than space or tab
  - Most modern languages don't care for column position, so could write program as a single line
  - Python is an exception

2.7 Other Grammar Forms

- Backus-Naur Form (BNF)
  - Has many minor variations, use = or ->instead of ::=
  - Metasymobls are part of language of the definition, not of the language being defined
- Extended Backus-Naur Form (EBNF)
  - Might use brackets, parentheses, etc
  - [optional], {repeatable}, (group)
  - Use quotes to denote tokens as not metasymbols
- Syntax Diagrams (Railroad diagram)
  - Way to express grammars graphically
  - Uses circles, rectangles, and arrows to show flow and possible control flows
  - Railroad diagram bc many many arrows
  - Good for casual use; hard for machines + parse trees
- Formal, Context-free Grammars
  - Formal languages study formal grammars
  - Context-free b/c children of node in parse tree depend only on that node's non-terminal symbol (not on context of neighboring nodes in tree)
  - Regular grammars (less expressive, good for lexical structure) and context-sensitive grammars (more expressive, good for phrase structure) both exist

2.8 Conclusion

- Grammars
  - Used to define syntax (lexical and phrase structure)
  - Lexical is division of program text into meaningful tokens
  - Phrase is organization of tokens into parse tree for meaningful structures
- Good grammars
  - If grammar is in the correct form, can be fed into parser-generator
  - Simple, readable, short grammars are more memorable + easier to learn/use

3 Ch3: Where Syntax Meets Semantics

3.1 Introduction

- Grammar
  - Set of rules for constructing parse trees
  - Language defined by grammar is set of fringes of parse trees
- Equivalent Grammars
  - Different grammars may generate identical languages (bc identical fringes despite different internal structure)
- Internal structure of parse tree
  - Semantics must be unambiguous

3.2 Operators

- Operator (+, *)
  - Refers to both the tokens for the operation, and the operation itself
  - Unary, binary, ternary take one, two, three operands
- Operands
  - Inputs to operator
- Infix Notation
  - Operator between operands
  - Postfix has operator after operands

3.3 Precedence

- Higher precedence performed before lower precedence
  - Use different non-terminal symbol for each precedence level
  - Non-terminal symbols in this chain are in order of precedence, from lowest to highest (is generalizable)
- Precedence Levels
  - Smalltalk has 1 precedence level (no precedence)
  - C has 15, Pascal has 5
  - Can add unnecessary parentheses to make expressions more readable

3.4 Associativity

- Grammar for a language must generate only one parse tree for each expression
  - So need to implement left/right-associative (eliminate the other direction)
  - <exp >::= <exp >+ <exp >
  - <exp >::= <exp >+ <mulexp >| <mulexp >
  - Only recursive on left side, so that it is left associative and grows tree left
- Nonassociative operator
  - Has no defined behavior when used in sequence in expression
  - Prolog 'a :- b :- c' is just a syntax error; also '1 ¡ 2 ¡ 3'

3.5 Other Ambiguities

–

3.6 Introduction

–

3.7 Introduction

–

3.8 Introduction

–

3.9 Introduction

–

4 Ch1: Programming Languages

4.1 Introduction

–