EDAP05: Concepts of Programming Languages - Skills Sheet

Karl Hallsby

Last Edited: December 27, 2019

Contents

1 Language Critique

The book provides a convenient common framework for arguing about the benefits and disadvantages of language features and programming languages in general. Table 1.1 and, more generally, Section 1.3 in the book describe a number of axes along which we can try to understand the effect of language features and the qualities of programming languages.

1. You should be able to recognise and describe all the characteristics listed in Table 1.1, as well as the three basic criteria listed therein.
2. For some language feature, you should be able to recognise the characteristics and criteria affected by that feature's presence or absence.
3. For some language feature, you should be able to recognise how the feature's presence or absence affects the language, based on the three criteria from Table 1.1.
4. For two related language features, you should be able to compare them, based on the three criteria from Table 1.1.
5. For some language feature, you should be able to recognise and explain how it can affect the Cost considerations for training, compile time, execution time, and cost of poor reliability as outlined in Section 1.3.7.

2 Language Implementation

You should be familiar with the following concepts:

1. Language implementation via **pure interpretation**, **compilation**, and **hybrid implementation**
2. Language implementation with the help of a **just-in-time compiler**
3. The concept of language **run-time system**

3 Syntax

Syntax describes the possible structure (or form) of programs of a given programming language. Backus-Naur Form (BNF) grammars have emerged as the standard mechanism for describing language syntax. BNF grammars used to describe languages when communicating with language adopters and compiler implementors. There are also many tools (particularly the yacc and antlr families of programs) for automatically generating parsers, programs that recognise whether an input program matches a grammar and, if it does, execute user-defined actions upon encountering certain language constructs.

1. You should be able to determine whether a given property of a language is part of the syntax, of the static semantics, or of the dynamic semantics.
2. You should be able to read a BNF grammar and understand the difference between terminals and nonterminals.
3. Given a BNF grammar, you should be able to write down examples of programs that can be generated by the grammar.
4. Given a BNF grammar, you should be able to tell whether a given program can be generated by the grammar. If the program is generated by the grammar, you should also be able to generate a parse tree for the program.
5. You should be able to determine whether a given (small) BNF grammar is ambiguous (the problem is undecidable in general, so this skill only pertains to practically relevant examples as covered in the textbook).
6. Given a BNF grammar, you should be able to determine the associativity of any operator used therein.
7. You should be able to describe the difference between an object language and a meta-language.
8. Understand **arity**, **fixity**, and **precedence**, and **associativity** of operators

4 Natural Semantics

There are many ways to describe semantics. In this course, we focus on natural semantics (also known as Big-Step Operational Semantics).

1. You should be able to read a specification of natural semantics.
2. You should be able to undestand how **expressions** and **values** relate to each other.
3. Given a natural semantics and a parse tree (or unambiguous expression), you should be able to compute the semantics of the given program.
4. Given a natural semantics and a BNF grammar, you should be able to tell whether any parts of the semantics are undefined.
5. Given an understanding of what a state-free expression language is supposed to do, you should be able to write down a simple natural semantics for it.
6. You should be able to understand the concepts of Environments in the context of natural semantics and be able to utilise it when reading and reasoning about natural semantics.

5 Bindings and Lifetimes

1. You should understand the difference between static and dynamic type binding and be able to take advantage of either property in your programming.
2. Given a syntax, a compiler and a run-time system for a language, you should be able to determine whether the language is using static or dynamic type binding.
3. Concepts: static binding, stack-dynamic binding, explicit heap-dynamic binding, and implicit heap-dynamic binding.
4. Given a syntax, a compiler and a run-time system for a language, you should be able to determine which storage binding(s) the language is using.
5. Concept: lifetime of a variable
6. Concepts: allocation and deallocation of a heap-dynamic variable
7. Concept: binding time, especially the difference between static and dynamic binding times

6 Scoping

1. You should understand the difference between static scoping and dynamic scoping and be able to exploit either in your programming.
2. Given a language implementation, you should be able to write a program to determine whether the language uses static or dynamic scoping.
3. Concept: referencing Environment

7 Types

1. Concepts: the types of integers, floating-point numbers (floats), and decimal numbers (decimals)
2. Concept: the type of booleans
3. Concept: enumeration type
4. Concepts: character and string types
5. Concept: subrange types
6. Concept: record types
7. Concept: tuple types
8. Concept: list types
9. Concept: associative array types
10. Concept: union types, both free and discriminated
11. Concept: operator overloading
12. Concepts: strong typing and weak typing
13. Concepts: type checking and the differences between dynamic type checking, static type checking.
14. Concepts: type equivalence, including the difference between nominal and structural type equivalence
15. Concept: type constructors
16. Concept: typing rules as part of type systems, and how to read such rules and utilise them in your reasoning about program semantics
17. Concept: the type preservation property, also known as subject reduction, of type systems
18. Concepts: type parameters and parametric polymorphism
19. Concept: function types for subroutines
20. Concept: type classes
21. Concept: subtyping
22. Concept: covariance and contravariance of type parameters, and the arrow rule
23. Concept: bounded parametric polymorphism
24. Concept: definition-site variance and use-site variance of type parameters
25. Concept: Algebraic Datatypes as in Standard ML and their use in pattern-matching
26. Concept: Automatic Type Inference

8 Expressions

1. Concept: arithmetic expressions
2. Concepts: boolean expressions and relational expressions
3. Concepts: Different forms of object equality, including reference equality and structural equality
4. Concept: operand evaluation order and how it affects the outcome of programs
5. Concept: short-circuit evaluation and how it affects the outcome of programs

6. Concept: referential transparency and side effects
7. Concept: list comprehensions and their semantics
8. Concept: type coercion expressions, both explicit and implicit, including narrowing conversions and widening conversions
9. Concept: conditional expressions

9 Statements and Control Structures

1. Concept: assignment statements, including compound assignment
2. Concept: two-way selection statements
3. Concept: multiple-selection statements
4. Concept: counter-controlled loops
5. Concept: logically controlled loops
6. Concept: datastructure-controlled loops

10 Subprograms and Parameter Passing

1. Concepts: subprograms, including formal arguments and actual arguments
2. Concept: local variables in subprograms
3. Concept: nested subprograms
4. Concepts: parameter passing modes
   (a) by value
   (b) by result
   (c) by value-result
   (d) by reference
   (e) by name
   (f) by need
5. Concepts: subprograms as parameters, subprograms as return values, Closures
6. Concept: activation records

11 Pointers, References, and Arrays

1. Concepts: pointers and references
2. Concept: the dangling pointer problem
3. Concepts: garbage collection in the forms of reference counting and mark-and-sweep collection
4. Concept: arrays in the forms of
   (a) static arrays
   (b) fixed stack-dynamic arrays
   (c) fixed heap-dynamic arrays
   (d) heap-dynamic arrays

12 Abstract Datatypes

1. Concepts: information hiding and encapsulation
2. Concept: abstract datatypes
3. Concept: generic abstract datatypes, that is, abstract datatypes that take one or more type parameters

13 Object-Oriented Programming

1. Concept: object-oriented language
2. Concept: dynamic dispatch, also known as dynamic binding of methods
3. Concept: inheritance
4. Concept: method overriding
5. Concept: the combined use of static types and dynamic types in statically typed object-oriented languages

14 Functional Programming

1. Concept: local let bindings of variables
2. Concept: anonymous functions, also known as lambda expressions
3. Concept: first-class functions
4. Concepts: pattern matching, including wildcards
5. Concepts: exhaustiveness and redundancy in pattern matching
6. Concept: lists as algebraic datatypes
7. Concept: the map function
8. Concept: currying
9. Functional Programming in Standard ML

15 Exceptions and Continuations

1. Concepts: exceptions and exception handlers