Programming Paradigms

Unit 18 — Summary of Basic Concepts

J. Gamper

Free University of Bozen-Bolzano
Faculty of Computer Science
IDSE

Basic Concepts of Programming Languages

- Programming languages are artificial languages designed to communicate with computers
  - Provide most powerful human-computer interface
- There are thousands of different languages, which are more or less appropriate for different problems
- Can be classified according to programming paradigms and abstraction level
- There are many similarities to natural languages, e.g., syntax, semantics
  - Syntax determines whether a programm is well-formed
  - Semantic determines the meaning of lanugage concepts/programs, and can be defined in different ways (operational, aximoatic, denotational semantics)
- Type system in a programming language is needed to organize data and helps to check the correctness of programs
- Different forms of type checking, all having pros and cons
  - Weak typing vs. strong typing
  - Static vs. dynamic type checking
  - Type casting

Imperative and Object-oriented Paradigm

- Imperative paradigm is the oldest programming paradigm, based on von Neumann architecture
  - Assignment statement is a central element, assigning values to memory locations
  - Program consists of sequence of statements that change the program state
- Procedural programming is a refinement that makes it easier to write complex programs
- Machine languages were the earliest imperative languages, followed by FORTRAN and ALGOL
- Abstract Data Types is a further extension of imperative programming
  - Data and operations are encapsulated into a bundle (information hiding)
  - This hides the underlying represenation and implementation
- Object-oriented paradigm extends ADTs
  - Classes are blueprints for objects that encapsulae both data and operations
  - Objects exchange messages
  - Provides encapsulation, information hiding, inheritance, and dynamic dispatching

Object-oriented Programming with Ruby/1

- Design goal of Ruby: simplicity and productivity of the programmer
- Ruby is a pure object-oriented language, treating objects in a consistent way
- Ruby is a strongly typed language, but applies dynamic type checking
- Supports duck typing, and is therefore very flexible when it comes to substitutability
  - Exact class of object is less relevat, but what methods/operations can be performed counts
  - Comparable to interfaces, e.g., in Java
- Some nice features not present in other languages: rich set of methods on arrays, code blocks that can be passed as paraemters to methods, modules, mixins (i.e., modules included in class definitions), accessor methods that can have the same name as variables
- Programmers can be very productive using Ruby, can be used like a scripting language
- Comes with a very successful web development framework: Ruby on Rails
  - The original Twitter implementation was done in Ruby

Object-oriented Programming with Ruby/2

- Performance: Ruby is not the most efficient language
  - All the flexibility makes it difficult to compile programs
- Concurrent programming is difficult to do (with a state-based language)
- Type Safety: duck typing makes it harder to debug code that has type errors in it

Recursion

- Recursion is just a different kind of loop, but as expressive as loops
- Some programming languages are haevily based on recursion, others do not offer recursion at all
- Two important steps in writing recursive programs
  - Base cases
  - Termination
- Often recursion allows you to write elegant code
- With the right language, it is even efficient
- Tail recursion is important to make recursive programs efficient
  - In tail recursion, the recursive call is the last statement before the function terminates, i.e., after returning from the recursive call no other statements need to be executed
  - Therefore, no information need to be stored on the stack

Logic Programming in Prolog/1

- Prolog is a declarative programming language based on first-order logic
  - Specifies what to compute and not how to do it
- A Prolog program/knowledge base consists of facts and rules
- Evaluating a Prolog program means to prove a goal
  - Thereby, key concepts are instantiation, (pattern) matching, and backtracking
- Prolog uses recursion instead of loops
- Lists and structures are two very important data structures
- The cut operator allows to stop backtracking
  - Should be used with care
  - A better programming style is to replace it by negation
- "Generate and test" is a very common programming pattern

Logic Programming in Prolog/2

- The box model shows the execution of a Prolog program
  - Has four ports: CALL, EXIT, REDO, FAIL
- Debugger shows the program execution according to the box model
  - `trace` provides an exhausitive tracing mode
  - `debug` allows to jump to spy points set by the `spy` predicate
- Accumulators are frequently needed to collect intermediate results when traversing structures or lists
  - Helpful to make programs tail-recursive
- Sorting is an important operation
  - Generalized insertion sort, which allows to pass a sorting predicate
  - Constructing structures with the `=..` (univ) operator needed
- Another frequent and powerful operation is mapping structures and lists
  - General map-functions can be used (second-order functions)
- `read` and `write` predicates for simple interactive programs

Logic Programming in Prolog/3

- Prolog has a steeper learning curve compared to other languages
- Fairly focused niche applications, not really a practical general-purpose language
- Mainly used in application areas, such as artificial intelligence, natural language processing, and databases
- There are scalability issues, the basic matching strategy used by Prolog is computationally expensive
  - Has problems to process large data sets
- It is not as declarative as it seems at first glance
  - If you want to write efficient Prolog programs, you have to know what is going on behind the scenes

Functional Programming in Haskell/1

- Haskell is a pure functional language, providing referential transparency
  - function give the same output for the same input
  - functions have no side effects
  - a variable can only be assigned a value once
- The type system (strong/static) prevents you from making a lot of mistakes
  - Nevertheless, it is quite flexible when it comes to extending it with user-defined types
- Haskell offers a lot in terms of expressiveness, yielding very concise code
- Hasekell uses curried functions in combination with partial evaluation of functions,
  - i.e., internally, functions have only one input parameter;
  - functions with multiple input parameters are decomposed into a sequence of partial functions, each having one parameter
- It is easier to show the correctness of your programs, due to the pure functional style

Functional Programming in Haskell/2

- Haskell does lazy evaluation, which gives you an additional tool for writing programs efficiently
- Lazy evaluation supports to work with infinite lists: only those (finite) parts are evaluated that are actually needed and only when they are needed
- Haskell supports list comprehension, a powerful way to specify lists

- The pure functional paradigm also has a price: dealing with messy real-world situations such as IO and state is not easy
- Haskell has a steep learning curve, i.e., it takes a while to learn how to wield the power of Haskell
- This may also explain the fact that the Haskell community is relatively small

Concurrent Programming in Erlang/1

- The shared-nothing, message-passing process model is very powerful when it comes to implementing concurrency
  - Concurrency means any execution order (e.g., parallel or serial) without compromising the correctness of the program
- Erlang offers a lot in terms of reliability and fault tolerance
  - Controlled crash
- Erlang was developed with the aim to achieve industrial-strength high performance
- Erlang processes run on a virtual maching that automatically adapts to the underlying hardware
  - Runs on as many cores/machines as available
- Language supports some powerful features of functional and logic-oriented languages
  - e.g., pattern matching, optimized for tail-recursion
- OTP provides a lot of functionality to make it easier to implement concurrent applications

Concurrent Programming in Erlang/2

- The syntax of the language is a weird mix of Prolog with functional language constructs thrown in
- While Erlang shines when it comes to concurrency, programming simpler (serial) things tend to be harder than in other languages