20CYS312 - Principles of Programming Languages
Exploring Programming Paradigms

Assignment-01

Presented by Suvetha D P
CB.EN.U4CYS21078
TIFAC-CORE in Cyber Security
Amrita Vishwa Vidyapeetham, Coimbatore Campus

Feb 2024

Outline

Introduction

- A programming paradigm is a fundamental approach or style of programming that provides a set of principles, concepts, and techniques for designing and implementing computer programs.
- It defines the structure, organization, and flow of the code, as well as the methodologies for problem-solving and expressing computations.
- Different paradigms have their strengths and weaknesses, and choosing the right paradigm for a given task can greatly impact the efficiency, maintainability, and scalability of a program.

Introduction

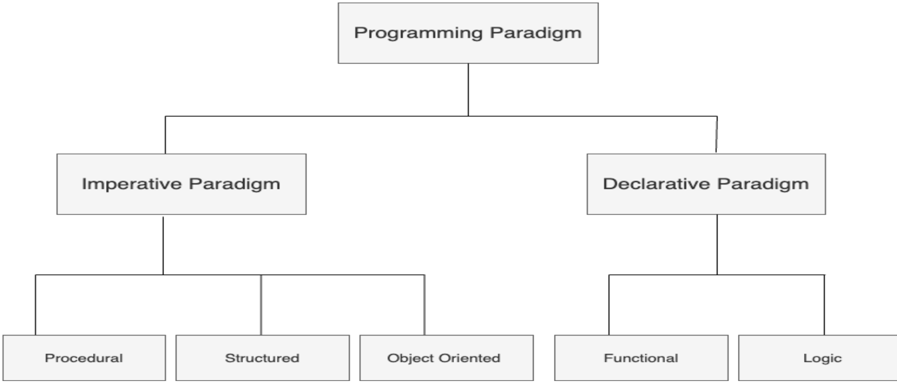


Figure: Programming Paradigm

Functional Paradigm

- Functional programming is a programming paradigm which has its roots in mathematics, primarily evolved from lambda calculus.It is a declarative type of programming style. Its main focus is on **“what to solve”**.
- This is the style of writing programs through the compilation of a **set of functions**. Its basic principle is to wrap almost everything in a function, write many small reusable functions, and then simply call them one after another.
- The primary aim of this style of programming is to avoid the problems that come with shared state, mutable data and side effects which are commonplace in object oriented programming. Functional programming tends to be **more predictable and easier** to test than object oriented programming.

Principles and Concepts of Functional Paradigm

- Purity
- Immutability
- Disciplined State
- First class functions and high order functions
- Type systems
- Referential transparency
- Recursion

Principles and Concepts of Functional Paradigm

Purity

- A pure function is a function which always returns the same output when called with the same argument values.
- Pure functions are predictable and reliable. Most of all, they only calculate their result.

Immutability

- It means that once you assign a value to something, that value won't change.
- Immutability helps to maintain state throughout the runtime of a program. Since your function has a disciplined state and does not change other variables outside of the function, you don't need to look at the code outside the function definition.

Disciplined State

- Shared mutable state is challenging to maintain due to multiple functions having direct access, making it prone to errors. It complicates code readability and maintenance, requiring tracking functions using shared variables for understanding. Debugging becomes difficult in such scenarios.
- **state discipline: you use state,but in a very disciplined way.**

Principles and Concepts of Functional Paradigm

First class functions and high order functions

- Functional programming treats functions as first-class citizens, enabling them to be passed as arguments, returned as values, stored in data structures, and assigned to variables.
- Higher order function is the function that takes one or more functions as arguments or returns a function as its result.

Type System

- Type systems help the compiler in making sure that you only have the right types as arguments, turn statements, function composition, and so on. The compiler will not allow you to make any basic mistakes.

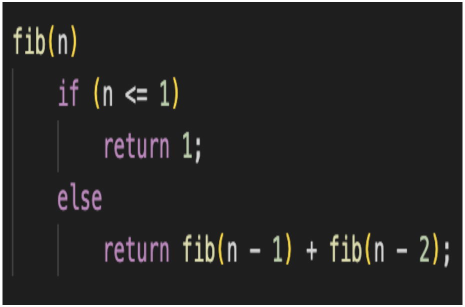
Referential transparency

- An expression is said to be referentially transparent if it can be replaced with its corresponding value without changing the program's behaviour.
- To achieve referential transparency, a function must be pure. This has benefits in terms of readability and speed. Compilers are often able to optimize code that exhibits referential transparency.

Principles and Concepts of Functional Paradigm

Recursion

- There are no “for” or “while” loop in functional languages. Iteration in functional languages is implemented through recursion. Recursive functions repeatedly call themselves, until it reaches the base case.



```
fib(n)
    if (n <= 1)
        return 1;
    else
        return fib(n - 1) + fib(n - 2);
```

Principles and concepts of functional programming enhance code readability, predictability, and testability, resulting in codebases with fewer bugs. This approach facilitates smoother onboarding and contributes to a more pleasant and maintainable code base.

Functional Paradigm: F#

- F# is pronounced as F Sharp. It is a functional programming language that supports approaches like object oriented and imperative programming approach. It is a cross-platform and .Net Framework language. The filename extension for F# source file is .fs.
- It was designed and developed by Microsoft. It was first appeared in 2005. Current stable version of F# is 4.0.1.20 which was released on November 13, 2016.
- It is influenced by Python, Haskell and Scala. It influenced to C#, F* etc.
- F# interacts with .Net library so that it can access the entire library and tools.

Characteristics

- Lightweight syntax
- Immutable by default
- Type inference and automatic generalization
- First-class functions
- Powerful data types
- Pattern matching
- Async programming

Features

- Conciseness
- Convenience
- Correctness
- Concurrency
- Completeness

Features

Conciseness

F# provides clean and nice code to write no curly brackets, no semicolons and so on. Even you don't have to specify type in your code just because of type inference. Moreover, you can finish your code in less line compared to other languages.

Convenience

Common programming tasks are much simpler in F#. You can easily define and process your complex problems. Since functions are first class object so it is very easy to create powerful and reusable code by creating functions that uses other functions as a parameter.

Correctness

F# provides powerful type system which helps to deal with common type errors like null reference exception etc. F# is a strongly typed language which helps to write error free code. It is easily caught at compile time as a type error.

Features

Concurrency

F# provides number of built-in functions and libraries to deal with programming system when multiprocessing is occurred. F# also supports asynchronous programming, message queuing system and support for event handling. Data in F# is immutable by default so sharing of data is safe. It avoids lock during code communication.

Completeness

F# is a functional programming language but it also supports other programming approaches like object oriented, imperative etc. which makes it easier to interact with other domains. Basically, we can say that F# is designed as a hybrid language by which you can do almost everything that you can do with other programming languages like C#, Java etc.

Applications

F# has a rich set of library and supports multi-paradigm which helps to deal with every domain whether it is desktop based application, web based application or mobile based application.

- Data analysis
- Scientific research
- Data statistical
- Design games
- Artificial Application
- Desktop application
- Mobile application

Logic Paradigm

- A logic programming paradigm is a set of principles and techniques that guide the design and implementation of logic programs.
- A logic program consists of a collection of facts and rules that describe the relationships and properties of entities, and a query language that allows asking questions and obtaining answers from the program.
- A logic programming paradigm defines the syntax and semantics of the facts, rules, and queries, as well as the inference mechanism that derives new facts and rules from the existing ones.
- Major logic programming language families include Prolog, Answer Set Programming (ASP) and Datalog.

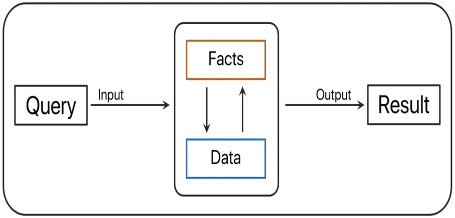


Figure: logic Workflow

Principles and Concepts of Logic Paradigm

- Declarative Nature
- Logic Rules and Facts
- Predicate Logic
- Unification
- Backtracking
- Negation as Failure

Principles and Concepts of Logic Paradigm

Declarative Nature

Logic programming emphasizes what needs to be achieved rather than how to achieve it. Programs are expressed as a set of logical statements that declare relationships and conditions, leaving the execution details to the underlying logic programming system.

Logic Rules and Facts

Rules: Logical relationships are defined using rules, typically in the form of Horn clauses. Rules consist of a head and a body, with the body specifying conditions that, when satisfied, lead to the conclusion stated in the head.
Facts: Facts represent basic truths or knowledge in the form of simple statements.
Eg: father(charles, william) -> Charles is William's father.

Predicate Logic

Predicate logic is a formal system that uses predicates to express relationships and conditions. Predicates are statements that involve variables and constants, and they evaluate to true or false.

Principles and Concepts of Logic Paradigm

Unification

Unification is a process used in logic programming to find substitutions for variables that make two logical expressions equivalent. It plays a crucial role in pattern matching and solving queries.
Eg: For the query "parent(mary, X)" and the rule "parent(Y, Z) :- mother(Y, Z)," unification binds Y to "mary" and Z to "X," making the query and rule heads identical.

Backtracking

Backtracking is a search strategy employed by logic programming systems to explore alternative paths when a failure occurs. It involves undoing decisions and exploring different branches of the search space.

Negation as Failure

In logic programming, negation is often implemented as failure. If a query cannot be proven true, it is assumed false. This concept, known as "negation as failure," is a way of expressing negation in a logic programming context.
Eg: If the program tries to prove "friend(john, bob)" using various rules but fails in all attempts, it can then infer "not friend(john, bob)" under negation as failure.

Logic Paradigm - Mercury

- Mercury, a general-purpose programming language, originated from the **University of Melbourne, Australia**. It embodies the paradigm of purely **declarative programming**, designed for creating robust real-world applications by seamlessly integrating logic and functional programming.
- It enforces a disciplined approach by necessitating type, mode, and determinism declarations for predicates and functions. The compiler rigorously checks these declarations, enhancing **reliability and productivity**. The correctness of declarations not only prevents certain errors but also serves as valuable documentation for maintenance, while optimizing the efficiency of the generated code.
- Mercury stands out as a language that prioritizes reliability, productivity, and efficiency through its declarative paradigm, stringent type checking, and the **integration of logic and functional programming features**.

Characteristics and Features of Mercury

Declarative

Mercury is purely declarative, ensuring that predicates and functions have no non-logical side effects. It manages I/O through specific predicates operating on an old state, enforcing a destructive update model. I/O is confined to sections where backtracking is unnecessary. Mercury handles dynamic data structures with abstract data types in the standard library and allows the creation of custom types.

Strongly Typed Language

Mercury's type system is based on many-sorted logic with parametric polymorphism, very similar to the type systems of modern functional languages such as ML and Haskell.

Characteristics and Features of Mercury

Strongly Moded Language
The programmer must declare the instantiation state of the arguments of predicates at the time of the call to the predicate and at the time of the success of the predicate. Currently only a subset of the intended mode system is implemented. This subset effectively requires arguments to be either fully input (ground at the time of call and at the time of success) or fully output (free at the time of call and ground at the time of success).

Module System
Mercury programs are composed of modules, each with an interface section declaring exported types, functions, and predicates. The implementation section defines these entities and local types or predicates. Exported type names without definitions are accessible only within the defining module, implementing abstract data types. Mercury supports automatic inference for type, mode, and determinism in non-exported predicates and functions.

Characteristics

- Mercury supports higher-order programming, with closures, currying, and lambda expressions.

- Strong types, modes, and determinism provide the compiler with the information it needs to generate very efficient code.

- The Mercury compiler is written in Mercury itself. It was bootstrapped using NU-Prolog and SICStus Prolog. This was possible because after stripping away the declarations of a Mercury program, the syntax of the remaining part of the program is mostly compatible with Prolog syntax.

- Mercury compiler compiles to C with GNU C extensions for better performance. It uses features like global register variables and inline assembler. However, it remains functional without these extensions for portability.

Applications

- Medical Diagnosis and Decision Support
- Financial Risk Management and Trading
- Robotics and Automation
- Logistics and Supply Chain Management
- Software Development and Testing

Advantages of Functional Programming

1. **Simplicity:** Pure functional languages simplify code structure, making it easy to understand.

2. **Scalability:** Supports concurrent thread handling for scalability.

3. **Code Reusability**: Functions are designed for reuse, passed as parameters, or returned.

4. **Testing and Debugging:** Testing is simplified with pure functions, aiding error identification.

5. **Readability:** Code is less complex, and readability is increased with immutable values.

6. **Lazy Evaluation:** Enhances performance by preventing unnecessary evaluations.

7. **No Side Effects:** Avoids side effects, ensuring consistent outcomes for the same input.

Disadvantages of Functional Programming

1. **Performance**: Immutability can lead to garbage generation, impacting performance.
2. **Memory Consumption:** Immutability results in data duplication, increasing memory usage.
3. **Tools Framework Support:** Limited support and tools compared to languages like JavaScript or Java.
4. **Loops Compatibility:** Lacks traditional loops, relying on recursion, which can be complex.
5. **Data Duplication:** Struggles with data duplication, impacting performance.
6. **Mathematical Concepts:** Connected to complex mathematical concepts, requiring time to grasp.
7. **Community Support:** Smaller community may lead to reliance on third-party vendors for support.

Advantages of Functional Programming-F#

1. **Algebraic Data Types (ADT):** F# promotes ADT, enhancing data structure precision and reducing misinterpretation.

2. **Transformations and Mutations:** F# favors transformations over mutations, preserving the original entity and simplifying code.

3. **Concise Syntax:** F# features a so-
phisticated type inference scheme, allowing omission of types for concise and readable code.

Disadvantages of Functional Programming-F#

1. **Naming Challenges:** Lack of method overload feature makes naming conventions challenging in F#.

2. **Complex Data Structures:** Effective manipulation of complex data structures, such as binary trees, requires a transformation-over-mutation approach.

3. **Less Advanced Tools:** Compared to C#, F# lacks comprehensive tools and resources, making coding less convenient, with limited support for refactoring .

Notable Features of Functional Paradigm - F#

- Immutability
- Pure Functions
- Higher-Order Functions
- Pattern Matching
- Strong Typing
- Type Inference
- Concise and Expressive Syntax

Advantages of Logic Programming:

- Logic programming facilitates conveying knowledge independently of implementation, enhancing flexibility and understandability.

- Information separation allows machine architecture changes without affecting programs or their code.

- Natural adaptation and extension for accommodating specialized knowledge types.

- Applicable in non-computational disciplines relying on reasoning and precise expression.

Disadvantages of Logic Programming

- Initial underserving of consumers due to insufficient investment in complementary technologies.

- Early lack of facilities for supporting arithmetic, types, etc., discouraging the programming community.

- Inadequate representation method for computational concepts in built-in state variable mechanisms.

- Some programmers prefer the overtly operational character of machine-operated programs.

Advantages Logic Paradigm in Mercury

- Declarative approach focusing on "what" needs to be achieved rather than "how."

- Powerful reasoning and inference capabilities through rules and facts.

- Expressive and concise syntax for readable code.

- Suitable for complex domains like natural language processing and constraint satisfaction.

- Static type checking for error prevention and code robustness.

- Tabling for efficiency to avoid redundant computations.

Disadvantages Logic Paradigm in Mercury

- Non-deterministic execution due to backtracking, potentially leading to unpredictable behavior and debugging challenges.

- Limited concurrency support compared to some other languages.

- Steeper learning curve for those unfamiliar with logic programming paradigms.

- Complex error handling, especially for intricate problems.

- Smaller community and ecosystem with fewer resources and libraries available.

Notable Features of Logic Paradigm - Mercury

- Declarative Approach
- Facts and Rules
- Unification
- Backtracking and Reasoning
- Pure Logic
- Static Type Checking

Similarities

Declarative Style: Both prioritize "what" over "how" in problem-solving.
Immutability: Promote the use of immutable data structures for predictability and thread safety.
Strong Typing: Employ static type systems for early error detection and improved code robustness.
Conciseness and Readability: Strive for clean and concise syntax to enhance code readability and maintainability.
Expressive Power: Offer powerful abstractions and data manipulation capabilities for handling complex problems.

Differences

Core Paradigm:
- F#: Primarily functional, influenced by imperative languages, focuses on data transformations.
- Mercury: Pure logic language built on facts and rules, emphasizing reasoning through unification and backtracking.

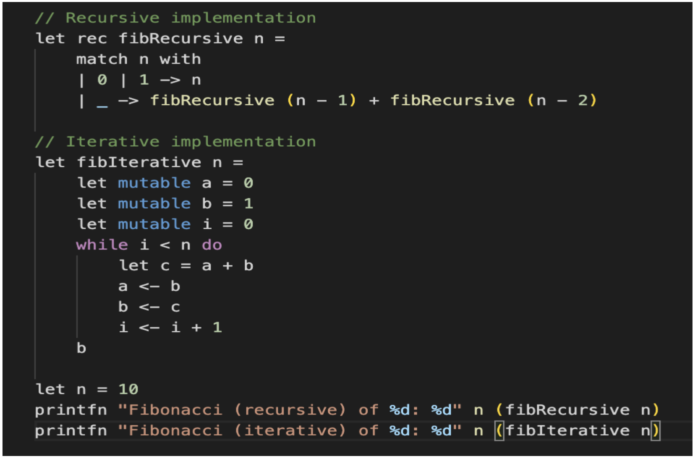
Execution and Control Flow:
- F#: Follows traditional execution order with explicit constructs like loops and conditionals, supports recursion.
- Mercury: Non-deterministic with backtracking, exploring alternative solutions and implicit control flow driven by rule matching.

Error Handling:
- F#: Requires explicit handling of exceptions and potential error states.
- Mercury: Non-deterministic execution and negation as failure may lead to complex debugging scenarios.

Concurrency:
- F#: Built-in features for parallel and asynchronous programming.
- Mercury: Limited built-in support, requires specific constructs and libraries for parallel processing.

Program

Functional Paradigm - F#

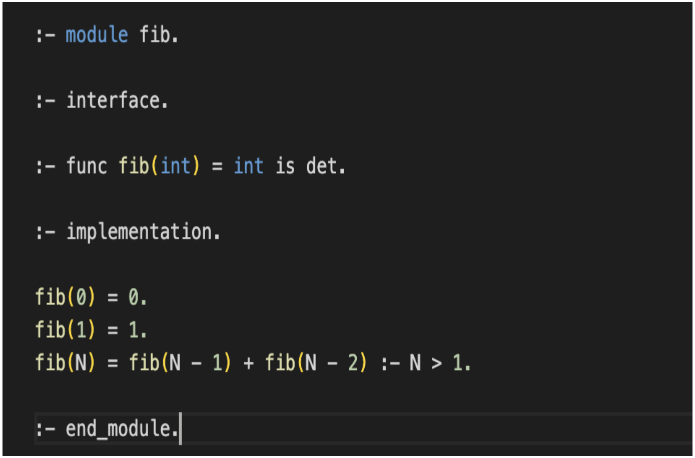


```
// Recursive implementation
let rec fibRecursive n =
    match n with
    | 0 | 1 -> n
    | _ -> fibRecursive (n - 1) + fibRecursive (n - 2)

// Iterative implementation
let fibIterative n =
    let mutable a = 0
    let mutable b = 1
    let mutable i = 0
    while i < n do
        let c = a + b
        a <- b
        b <- c
        i <- i + 1
    b

let n = 10
printfn "Fibonacci (recursive) of %d: %d" n (fibRecursive n)
printfn "Fibonacci (iterative) of %d: %d" n (fibIterative n)
```

Figure: Fibonacci in F#

Program

Logic Paradigm - Mercury



```
:- module fib.

:- interface.

:- func fib(int) = int is det.

:- implementation.

fib(0) = 0.
fib(1) = 1.
fib(N) = fib(N - 1) + fib(N - 2) :- N > 1.

:- end_module.
```

Figure: Fibonacci in Mercury

References

1. https://www.geeksforgeeks.org/functional-programming-paradigm/
2. https://www.freecodecamp.org/news/
an-introduction-to-programming-paradigms/
3. https://hackr.io/blog/functional-programming
4. https://www.tutorialspoint.com/fsharp/fsharp_overview.htm
5. https://www.javatpoint.com/what-is-f-sharp
6. earn.microsoft.com/en-us/dotnet/fsharp/what-is-fsharp
7. https://www.linkedin.com/advice/0/
what-main-characteristics-examples-logic-programming
8. https://www.allassignmenthelp.com/blog/
logic-programming-what-are-its-techniques/
9. https://mercurylang.org/about.html