Refactoring Scala Programs to Promote Functional Design Patterns

Namrata Malkani; Manas Thakur
Indian Institute of Technology Mandi

KEY POINTS
- Functional Design Patterns - Immutability, declarative style.
- Scala - Hybrid language with excellent functional features.
- Idiomatic Programming - Concise, less error-prone, easy to implement & understand – like the functional features of Scala.
- Loops => Common Sequence Methods.
- Single line solution, no mutation, less errors, more readable.
- Conditionals => Pattern Match.
- Better readability, enforces common return value, efficient byte-code.
- ScalaRT: Scala Refactoring Tool.
- Suggest improvements on code to implement idiomatic style.
- Easily extendable, can be imported in packages in Scala projects based on need.
- Stands at 1300+ lines of code.

FUTURE RESEARCH
- More refactoring, implementing currying, tackling larger OO patterns.
- Evaluating the original and refactored program with respect to a code-quality check system.

REFERENCES

AUTHORS’ CONTACT
Namrata Malkani
Indian Institute of Technology Mandi
Email: nmalkani13@gmail.com

Manas Thakur
Indian Institute of Technology Mandi
Email: manas@iitmandi.ac.in

SCALA CODE ANALYSIS & REFACTORIZING
- Refactoring: Transforming program to improve its design, structure, and implementation, while preserving its functionality.
- It requires a framework that lets the user work on the underlying syntax tree of the source code. We use the metaprogramming library Scalameta.

ScalaRT: TOOL DESIGN
- Loop_replacer Package
  - main
  - loop_replacer
    - checkLoop
    - whileLoop
    - forLoop
    - patternMatch
    - checkConditional
    - loopPatternMatch
  - conditional_replacer
    - Package

LOOPS TO COMMON SEQUENCE METHODS
- For Loop
  - Based on enumerator
  - Condition on loop body
  - Assignment Statement
- While Loop
  - Based on loop body
  - Conditional Statement
  - Assignment Statement
- Multiple Loops
  - Index-Based
  - Non-Index Based

CONDITIONALS TO PATTERN MATCH EXPRESSIONS
- Source Code
  - Parse Syntax Tree
  - Is_if_node
    - F
    - T
  - Is_for_node
    - Construct pred-stmt map
    - Parse loop body
    - Check functional conditions
    - Is nested node
    - Check functional conditions
    - Get tuple set for match_exp
    - Modify each predicate intro case clause
    - Output
- Output
  - Plug relevant stmt into case clause, construct pattern match body
  - Get tuple set for match_exp
  - Modify each predicate into case clause
  - Output

Table 1. Loop refactoring examples. Exact refactored code or likely sequence method replacement can be given from among 20 sequence methods in Scala 2.