TU Delft

Delft University of Technology

Simplify Automated Restructurings with Koncrete

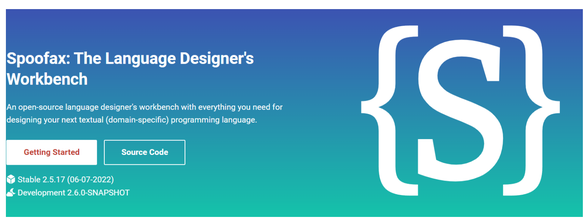
A MasCot Software Restructuring Project (17933)

Luka Miljak Casper Bach

in collaboration with: ESI TU/e PHILIPS

Motivation

- Automated restructuring is necessary for maintaining large and outdated codebases
- Writing automated refactorings is hard; steep learning curve
- How to simplify writing refactoring implementations and improve readability?



Spoofax Language Workbench;
Everything you need to design your own domain-specific programming language
www.spoofax.dev

Koncrete Pattern

Re-uses the syntax of the target language, like C++, Python, etc.

Case Study
API-Migration on C++ codebase of Philips Healthcare

Compile

Black-Box Parser required of the target language (e.g., C++ parser to support C++ patterns)
- Advantage: Easy to interface with any new language
- Disadvantage: Compilation requires some syntactic type hints

Abstract Pattern

Source Code

Match + Patch

Refactored code

Koncrete Pattern Examples

Example: matching recursive function calls

```
Decl```
  int @foo() {
    @<… Expr
      @foo()
    …>
  }
```
```

`Expr Decl Stmt`
Syntactic type hints needed for compilation

**@<… …>**
**Descendant Pattern**
match at any depth

**@|**
**Disjunctive Pattern**
match either left or right

Example: matching both classes and structs

```
Decl```
  <class@|struct> @foo {
    @body
  };
```
```

- - >
Inline Patch
perform a transformation

Example: refactoring test assertions

```
Decl```
  TEST(@f, @t) {
    @<… Stmt
        <EXPECT_TRUE(@e1 == @e2);  --> EXPECT_EQ(@e1, @e2);>
    @| <EXPECT_FALSE(@e1 == @e2); --> EXPECT_NE(@e1, @e2);>
    …>
  }
```
```

### Example Koncrete script in Kotlin

```
// Initialize the (Black-Box) parser
val parser: BlackBoxParser = CDTParser()
// Initialize the library
val lib = Koncrete(parser)

// Source code (Legacy code)
val src = """
  TEST(MyTest, FooIsBar) {
    EXPECT_TRUE(foo(0) == bar(0));
    EXPECT_FALSE(foo(0) == bar(1));
  }
"""

// Koncrete pattern
val pattern = """
  TEST(@f, @t) {
    @<… Statement
        EXPECT_TRUE(@e1 == @e2);
          --> EXPECT_EQ(@e1 == @e2);
     @| EXPECT_FALSE(@e1 == @e2);
          --> EXPECT_NE(@e1 == @e2);
    …>
  }
"""
// Declaring the types of
// metavariables f, t, e1, and e2
val metavars = mapOf(
  "f" to NAME, "t" to NAME,
  "e1" to EXPRESSION, "e2" to EXPRESSION
)

// Pattern matching
val matches: List<MatchResult> = lib.match(
  metavars, DECLARATION, pattern, src
)

// Patching
val result: String = lib.patchAll(
  src, matches
)

// Displaying restructured code
println(result)
```
```