Bachelor's Thesis
Utility Functions for Kotlin Compiler IR Generation

Student: Mario Birmili
Advisor: Dipl.-Ing. Dr. Markus Weninger, BSc
Start date: October 2024

Dipl.-Ing. Dr. Markus Weninger, BSc
Institute for System Software
T +43-732-2468-4361
markus.weninger@jku.at

Kotlin compiler plugins play a vital role in extending the language's capabilities and providing developers with powerful tools for code manipulation. Our existing compiler plugin generates code but requires complex and repetitive constructs, which can make the development process cumbersome and error-prone. By introducing helper functions and extension methods, we can simplify code generation, enhance readability, and improve maintainability.

The goal of this bachelor's thesis is to enhance the existing Kotlin compiler plugin by developing a set of helper functions and extension methods that streamline the code generation process. The student will focus on creating convenient ways to reference classes and functions, handle data types and nullability issues, and simplify common code generation patterns.

The main objectives are as follows:

1. *Familiarization with the Existing Plugin and Kotlin Compiler Internals* such as `IrClass`, `IrFunction`, `IrProperty`, `IrField`, `IrCall`, and the *plugin context* object.

2. Development of *Helper Functions for Element Referencing*: Find and refactor `.referenceXYZ` calls (such as `referenceFunction(…)`) with less complex functions (such as `findFunction("kotlin/io/println(String)")` or `findClass("kotlin/io/MyClass")`). Also create extension functions for Class Symbols in a similar fashion (such as `stringBuilderClass.findFunction("append(Int)")`, where `stringBuilderClass` was previously found using `findClass`). Replace all `referenceXYZ` calls in the existing plugin with the newly introduced (and easier to use) findXYZ calls.

- When working with types, you must differentiate between built-in and non-built-in classes and manage nullability in types (e.g., `Int` vs. Int?, my.package.Person vs. my.package.Person?).
- Guidance (Pseudo Code):

```
when (typeStr) {
  "Int" -> pluginContext.irBuiltIns.intType
  "Long" -> pluginContext.irBuiltIns.longType
  else -> findClass(typeStr) // typeStr could be "my.package.Person?"
}
```

3. Development of a *Flexible String Concatenation Function*: Create a `concat` function capable of handling a vararg of various kinds of elements (Literals, string fields, string properties, function calls that return strings, and other elements that can be converted to strings using a `.toString` call).

- Guidance (Pseudo Code):

```
fun concat(vararg parameters: Any): IrExpression {
  return irConcat().apply {
    for (parameter in parameters) {
      when (parameter) {
        is String -> addArgument(irString(parameter))
        is IrField -> addArgument(irGetField(null, parameter))
        is IrProperty -> addArgument(irCall(parameter.getter!!))
        is IrFunction -> addArgument(irCall(parameter))
        else -> addArgument(irToString(parameter))
      }
    }
  }
}
```

6. *Facilitating Function Calls on Various Objects*: Implement methods to call functions on objects stored in different locations such as fields or properties. Also enable fluent calling of functions on objects returned from other function calls. For this, develop extension methods for `IrProperty`, `IrField`, `IrCall`, etc., to simplify function invocation (e.g., `IrProperty.call(func: IrFunction, vararg parameters: Any): IrCall { /* ... */ }`)

7. Application and Testing: Apply the developed helper functions and extension methods to simplify existing code in the plugin. Validate the correctness and efficiency of the generated code. Ensure that the improvements lead to more readable and maintainable plugin code.

8. Documentation and Evaluation: Document the implemented functions and methods with clear explanations and usage examples. Evaluate the impact of the changes on the plugin's codebase and developer experience (LOCs, side-by-side comparison of code snippets, etc.). Provide recommendations for future enhancements.

Modalities:
The progress of the project should be discussed at least every three weeks with the advisor. A time schedule and a milestone plan must be set up within the first 3 weeks and discussed with the advisor. It should be continuously refined and monitored to make sure that the thesis will be completed in time. The final version of the thesis must be submitted not later than 31.03.2025.