Kotlin Compiler Project - Extension

Tomás Fontes, Jorge Silva

December 11, 2024

1 Project Overview

This project is a simple Kotlin compiler written in Haskell. Initially, the project supported a basic subset of Kotlin, including primitive types, operators, control structures, and simple functions. The compiler was capable of lexical analysis and parsing into an Abstract Syntax Tree (AST). This extension focuses on expanding the language features and improving the efficiency of the generated code.

2 New Features and Enhancements

2.1 Language Subset Enhancement

The language subset has been expanded to include the following new features:

- **Additional Control Structures:**
  - `when` statements are now supported.
  - `for` loops have been added to enhance the control flow capabilities.
- **Advanced Expressions:**
  - More complex expressions, including function calls with parameters and expressions involving multiple operators, are now supported.
- **Function Overloading:**

  - Multiple functions with the same name, differentiated by their parameter types, are now handled.

- **Class Definitions:**
  - Simple class structures with fields and methods are supported, including object instantiation and method calls.

2.2 Compilation Process Enhancement

The compilation process has been improved to support the newly added features. Below is an updated breakdown:

2.2.1 1. Lexer (Scanner)

The lexer, implemented with the Alex library, has been updated to handle new keywords (like `when`, `for`, and `class`) and more complex expressions (such as function calls and method invocations). Token generation has been optimized to account for new syntax rules introduced by the expanded subset.

2.2.2 2. Parser

The parser, using Happy, has been extended to handle the new constructs. It now supports the parsing of `when` statements, `for` loops, class definitions, and function overloading. The parser has been restructured to handle nested and recursive expressions more efficiently, ensuring that function calls and complex expressions are parsed correctly.

2.2.3 3. AST (Abstract Syntax Tree)

The Abstract Syntax Tree has been extended to accommodate new node types:

- `When` Node: Represents a `when` statement in Kotlin.
- `For` Node: Represents a `for` loop.
- `FunctionCall` Node: Represents a function call, including handling overloading by differentiating based on argument types.
- `Class` Node: Represents class definitions, including fields and methods.

These new nodes enable the generation of more sophisticated intermediate representations for the extended language features.

2.2.4 4. Code Generation (Target Architecture)

The code generation phase, which was previously focused on simple variable assignments and function calls, has been enhanced to handle the new constructs. The code generator now produces assembly-like instructions for the new control structures (`when` and `for` loops) and class handling. Function overloading is handled by generating unique labels for each overloaded function based on the parameter types, ensuring proper dispatch at runtime.

3 Example Code

Here's an example of Kotlin code with the new features (e.g., `when`, `for`, and class handling) that the updated compiler can now handle:

```
fun max(a: Int, b: Int): Int {
    return if (a > b) a else b
}

fun max(a: Double, b: Double): Double {
    return if (a > b) a else b
}

fun main() {
    val x = 10
    val y = 20
    val result = max(x, y)
    println(result)

    val list = listOf(1, 2, 3, 4, 5)
    for (item in list) {
        println(item)
    }

    val num = 2
    when (num) {
```

```
        1 -> println("One")
        2 -> println("Two")
        else -> println("Other")
    }

    val obj = MyClass()
    obj.greet()
}

class MyClass {
    fun greet() {
        println("Hello, world!")
    }
}
```

4 Example Command

To build and run the updated compiler with the new features, use the following commands:

```
$ cabal build
$ cabal run compilador input.kt
```

This will process the input Kotlin source file, tokenize it, generate the Abstract Syntax Tree (AST), and produce assembly-like output based on the new constructs.

5 Project Files

The updated project includes the following files:

- `Lexer.x`: The lexer definition using Alex, updated for new language constructs.
- `Parser.y`: The parser definition using Happy, extended to handle additional features.

- `Main.hs`: The main entry point, now handling the new parsing and AST generation logic.
- `Codegen.hs`: A new file added for handling code generation, specifically targeting assembly-like output for the newly added features (e.g., `when`, `for`, and classes).
- `compilador.cabal`: The project configuration file, updated to reflect the new dependencies and modules.

6 Dependencies

The project still depends on the following Haskell packages:

- `alex` – Lexer generator (updated version to support additional constructs).
- `happy` – Parser generator (updated to support the expanded grammar).
- `base` – Core Haskell library.

New Dependencies:

- `containers` – Added for efficient list handling (for loops and collections).

To install the dependencies, run:

```
$ cabal install alex happy containers
```

7 Conclusion

This update to the Kotlin compiler extends its support to include more advanced Kotlin features, including control structures (`when`, `for`), function overloading, and simple class handling. The enhancements to the lexer, parser, and AST generation ensure that the compiler can handle more complex Kotlin code, while the code generation phase has been expanded to support these new features.

Future work could involve improving the efficiency of the generated assembly-like code, supporting more Kotlin features (such as lambda expressions and

higher-order functions), and adding optimizations to the compiler's performance. Additionally, error handling and diagnostics could be enhanced to improve the development experience.