ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

NGUYÊN LÝ NGÔN NGỮ LẬP TRÌNH (MỞ RỘNG)

Sinh mã JVM cho toán tử + của ngôn ngữ BKOOL

GVHD: Nguyễn Hứa Phùng
SV: Ngô Lê Quốc Dũng - 1910101

TP. HỒ CHÍ MINH, THÁNG 3/2022

Mục lục

1 Giới thiệu

Như đã được học trong môn Nguyên lý Ngôn ngữ Lập trình, quá trình biên dịch mã nguồn diễn ra trong hai giai đoạn: frontend và backend. Trong giai đoạn frontend, mã nguồn sẽ được chuyển về một dạng biểu diễn trung gian là cây cú pháp trừu tượng (AST), cây này sau đó sẽ là đầu vào cho bộ sinh mã trung gian Jasmin. Đoạn mã này sau đó sẽ được chuyển về mã bytecode có thể thực thi được trên máy ảo JVM trong giai đoạn backend.

Bài báo cáo ngắn này sẽ trình bày kiến trúc hiện tại của bộ sinh mã trung gian, cũng như kết quả hiện thực được cho toán tử + trên ngôn ngữ BKOOL.

2 Kiến trúc hiện tại của bộ sinh mã

Sau khi sinh ra cây AST và được kiểm tra lỗi tại thời gian dịch, thì cây AST sẽ được dùng làm đầu vào cho bộ sinh mã trung gian. Với kiến trúc được cho, bộ sinh mã trung gian gồm có 4 phần:

1. **Bộ sinh mã phụ thuộc máy**: file `MachineCode.py`, mỗi hàm sinh ra một dòng mã Jasmin
2. **Khung**: file `Frame.py`, mô phỏng lại stack khi thực thi chương trình, là nơi tạo ra và quản lý các nhãn (Label0, Label1,...)
3. **Bộ sinh mã trung gian**: file `Emitter.py`, chứa một số hàm hỗ trợ cho quá trình sinh mã (như sinh các dòng chỉ thị, sinh toán tử tương ứng với kiểu,...) nhằm trừu tượng hóa trên bộ sinh mã phụ thuộc máy
4. **Bộ sinh mã độc lập máy**: file `CodeGenerator.py`, dùng một visitor để sinh mã cho ngôn ngữ BKOOL dựa trên đầu vào là cây AST

3 Kết quả hiện thực bộ sinh mã cho toán tử +

Để sinh được toán tử + cho ngôn ngữ BKOOL, ta cần điều chỉnh mã nguồn `initial` tại những điểm sau:

- `main/bkool/parser/BKOOL.g4`: tạo ra văn phạm phù hợp cho ngôn ngữ BKOOL.
- `main/bkool/utils/AST.py`: định nghĩa những lớp cần thiết để xây dựng được cây AST.

- `main/bkool/astgen/ASTGeneration.py`: tạo ra cây AST từ một đoạn mã hợp văn phạm.
- `main/bkool/utils/Visitor.py`: định nghĩa những phương thức visit để buộc phía `CodeGenerator` phải hiện thực những phương thức đó
- `main/bkool/codegen/CodeGenerator.py`: sinh mã từ chương trình cho tới toán tử + trong một biểu thức nhị phân.
- `test/CodeGenSuite.py`: sinh ra các testcase cho toán tử +.

Vì xử lý trên cùng một cây AST nên việc sinh mã cho ngôn ngữ BKOOL không quá khác biệt so với việc sinh mã cho ngôn ngữ MC đã được viết sẵn trong initcode. Bằng việc chỉnh sửa lại file `CodeGenerator.py` để hoạt động trên ngôn ngữ hướng đối tượng, sửa lại `visitCallExpr` thành `visitCallStmt`, và thêm phương thức `visitBinaryOp` để xử lý được một biểu thức nhị phân có toán tử + cho cả hai kiểu int và float là công việc đã hoàn thành. Dưới đây là so sánh cấu trúc cây AST và mã Jasmin tương ứng.

```
Program([
    ClassDecl(Id(Program),[
        MethodDecl(Id(main),Static,VoidType,[],Block([
            CallExpr(Id(io),Id(putInt),[
                BinaryOp(+,IntLit(20),IntLit(30))
            ])
        ]))
    ])
])
```

```
.source BKOOLClass.java
.class public BKOOLClass
.super java.lang.Object

.method public static main([Ljava/lang/String;)V
.var 0 is args [Ljava/lang/String; from Label0 to Label1
Label0:
    bipush 20
    bipush 30
    iadd
    invokestatic io/putInt(I)V
Label1:
    return
.limit stack 2
.limit locals 1
.end method

.method public <init>()V
.var 0 is this LBKOOLClass; from Label0 to Label1
Label0:
    aload_0
    invokespecial java/lang/Object/<init>()V
Label1:
    return
.limit stack 1
.limit locals 1
.end method
```

Hình 1: So sánh cây AST (bên trái) và mã Jasmin (bên phải) cho phép toán 20+30 là tham số của lời gọi phương thức `io.putInt()`

Để chạy được thành công các test, em đã thực hiện 2 thay đổi sau:

- Sửa lại phương thức `check` ở `TestUtils.py` để chạy được trên hệ máy Windows
- Thêm kiểu float cho phương thức `getJVMtype` trong `Emitter.py` để có thể tính toán trên số thực