CS352s2014 Project 3: Translation

February 28, 2014

1 Objective

The objective of this project is to write a translator from MiniJava's AST to its SSA, as represented in the classes `edu.purdue.cs352.minijava.ssa.*`.

2 Code

The project's code environment is available on the course web page. We are providing Java class files for a complete compiler, including the parser, but are not providing accompanying Java source. Through the course of these projects, it will be your job to rewrite the Java source for each component, creating a complete compiler of your own design. You are advised to keep a copy of the Java class files so that you can use them to generate correct output and test against your own.

You are free to use the Java class files provided in lieu of your own implementations of the lexer and parser, or to use your own if you prefer.

2.1 Testing

The compiler provides a number of tools for testing each stage. They are available in the `bin` directory. Two are of value for testing the translator.

- `bin/mjcompile-ssa`: Compiles MiniJava code into SSA form and writes it to standard output.
- `bin/mjcompile-mips`: Compiles MiniJava code into MIPS assembler, using SSA as an intermediary.

When your translator is in place, the output of these programs should be *similar* to that of the versions provided. Please note that there is no single correct SSA form for any program, so your output is not expected to be *identical*. In particular, the names of labels are arbitrary and random, and reproducing them precisely as the original does would be impossible. You are graded on correctness, not similarity to the original.

3 SSA

Static Single Assignment form is a restricted form of Three-Address Code in which every instruction implicitly creates its own temporary variable. You are not expected to reimplement the SSA classes, `edu.purdue.cs352.minijava.ssa.*`. Their source code will be provided.

`SSAStatement` represents a single SSA instruction, and, for the purpose of this assignment, has three important components: The operation, in field `op`, the operands, in fields `left` and `right`, and, for those SSA instructions that do not perfectly fit TAC, special data, stored in the field `special`. The number of operands and type of the `special` field are documented in the `enum Op` section of `edu.purdue.cs352.minijava.ssa.SSAStatement`. Additionally, `SSAStatement`s have an `ast` field, which simply associates them with the AST node that produced them.

Please note that `SSAProgram` has more fields (`register`, `registerPinned`, `type`) which will be used in later projects. See `SSAStatement`'s constructors for the fields which are relevant to this assignment.

`SSAStatement`s are organized into `SSAMethod`s, which correspond to method declarations in the AST, or to the special main method declaration. `SSAMethod`s, in turn, are organized into `SSAClass`es, which additionally store the fields of their class, and correspond to class declarations in the AST. Finally, the top level of the AST corresponds to an `SSAProgram`.

4 Goals

For this assignment, you must implement one Java classes: `edu.purdue.cs352.minijava.SSACompiler`. `SSACompiler` should extend `edu.purdue.cs352.minijava.ast.ASTVisitor`, for compiling AST nodes into `SSAStatement` lists, as well as implementing `compile` methods for the AST types `Program`, `ClassDecl`, `Main` and `MethodDecl`. A template will be provided in `edu/purdue/cs352/minijava/SSACompiler.java-template` to clarify. The only error a conventional translator might report is if a variable has not been declared. To simplify this translator, you are expected only to keep a local (i.e., method) symbol table, and to assume that any variable references which are not to local variables refer to `this.`*var*. Incorrect programs will be rejected in a future project.

A number of examples are provided in the `examples` directory. These examples will help in testing, but are *not* the examples that will be used to grade your submission. In particular, none of these examples are incorrect MiniJava. The frontend used to test the SSA compilation is `bin/mjcompile-ssa`, which in turn uses the `edu.purdue.cs352.minijava.SSACompilerFrontend` class. You are not expected to reimplement `SSACompilerFrontend`, only `SSACompiler`.

4.1 Example

Given the following file in `examples/test.java`:

```
class Simple {
    public static void main(String[] a) {
        System.out.println(42);
    }
}
```

`SSACompilerFrontend` should output something similar to:

```
program:
main:
method main:
0:  Int *42
1:  Print 0
```

5 Submission

You will submit your project code via the `turnin` command. Please include the entire project code directory, including all the class files provided by us, the `bin` and `examples` directories, etc.

To turn in a project directory: `turnin -c cs352 -p proj3 <project directory>`

This project is due by Friday, March 14, 2014 at 12:30PM Purdue time. i.e., before class, *not* before midnight. Late submissions will not be accepted.

6 Grading

We will test your code on machines similar to those in the Linux labs used by this course's PSOs. Your grade will be based on correctness in the following criteria for both the generated and hand-written lexers:

- Correct organization of programs (`SSAProgram, SSAClass, SSAMethod`)
- Correct generation of `SSAStatement`s without side effects.
- Correct handling of local variables and their association with `SSAStatement`s.
- Generation of correct statement sequences with control flow.
- Use of SSA's *unify* operator where necessary.