Assignment 9: Using Truffle and Graal

Patrick S. Li

November 24, 2015

1 Introduction

As you have discovered from writing your own virtual machine, many programming languages are trivial to implement if performance is not a concern. However, obtaining good performance from dynamically typed languages is not trivial. Furthermore, the effort involved in writing a high performance virtual machine is often tied intricately to the target programming language and cannot be shared amongst different language implementations.

In this assignment, we will take advantage of Oracle's Truffle framework and Graal compiler to implement a subset of the Feeny language. Truffle/Graal is a general virtual machine framework that allows for many languages to easily take advantage of the HotSpot dynamically-optimizing compiler to obtain good performance.

2 Harness

The system infrastructure required for this assignment has been provided to you in the form of a VirtualBox image. Start it up and launch the Eclipse IDE located at:

```
/home/patrick/eclipse/java-mars/eclipse/eclipse
```

. A default workspace under

```
/home/patrick/Documents/feeny-workspace
```

has been prepared for you which contains the Truffle and Graal frameworks, and a skeleton project for Feeny.

On the left hand side of the Eclipse editor, in the Package pane, is a listing of projects. The Feeny skeleton project is listed under `feeny`. The application driver is in the `feeny.Feeny` class. Open it and navigate to the `main` function which contains a number of simple examples for creating and executing Truffle nodes.

3 Creating and Executing a Simple Node

Look at the definition of `test_hello` for an example of implementing a simple Truffle node that simply prints "Hello" to the screen.

A new `HelloNode` object is created with a fresh `FrameDescriptor`, and then executed. A `FrameDescriptor` is a description of the arguments and slots of the local frame of a given scope. It plays a similar role as the `Env` object in the operational semantics of Feeny.

Open the definition of `feeny.nodes.HelloNode` to study the implementation of a simple Truffle node. The first thing to notice is that `HelloNode` inherits from `RootNode`. This is required by the Truffle framework.

A default constructor is defined that takes a `FrameDescriptor` as an argument and passes it along to the superclass constructor.

Finally, the behaviour of `HelloNode` is implemented in the overridden `execute` method. It takes a `VirtualFrame` as input, which holds the values of all the local variables in the current frame, and returns the result of evaluating the node. The `HelloNode` node simply prints "Hello" and returns the null object.

4 Nodes Containing Child Nodes

Look at the definition of `test_plus` for an example of implementing a Truffle node that contains children nodes. The `PlusNode` performs a binary addition operator on two child nodes.

Look at the definition of `feeny.nodes.PlusNode` and notice the definitions of the children `RootNode` nodes. To facilitate Truffle's automatic partial evaluation capabilities, the children nodes must be marked with the `@Child` annotation. In the `execute` method for `PlusNode`, the children nodes are executed via recursive calls to `execute`.

5 Creating, Writing, and Reading from Variables

Look at the definition of `test_variables` for an example of managing variables. The constructor for `LocalsNode` creates a slot in the `FrameDescriptor` for a variable called "i". The implementation of `LocalsNode` simply executes the two children nodes, `WriteLocalNode` and `ReadLocalNode` in sequence.

The `WriteLocalNode` assigns a value to the slot in the frame corresponding to the variable "i", and the `ReadLocalNode` reads the current value assigned to the slot for variable "i".

6 Creating and Calling Functions

Look at the definition of `test_feeny` for an example of creating and calling a function. In this example, we will create a simple function that simply prints out the value of its first argument, and then call this function with the string "Hello World".

The body of the function is represented by the `PrintArgNode`. Please read the implementation of its `execute` method for an example of how to read a function argument.

To create a function from the `PrintArgNode` we wrap it in a `RootCallTarget`. This target is then passed to the constructor for `CallNode`. The `execute` method for `CallNode` calls its given target with the argument "Hello World".

7 Implementing Feeny in Truffle and Graal

The above examples provides you the necessary techniques for implementing a subset of the Feeny language using Truffle and Graal. You will not need to support the `array` or `object` Feeny language constructs, and consequently only need to support the Null object and Integer values.

A reader for reading the Feeny AST binary format is provided for you in the `feeny.reader` package. The datastructure returned by the reader is a mirror of the C datastructure provided for you on previous assignments. See the definition of `test_feeny` for an example on using the reader.

To implement Feeny, you will write a tree transformer that takes the Feeny AST as input and outputs a corresponding Truffle AST that implements the semantics of Feeny.

8 Optimization

The techniques described so far enables you to write a functional version of a subset of Feeny using Truffle and Graal. However, to fully enable all of Truffle's optimizations, you will need to take advantage of additional annotations. Read the implementation of the Simple Language in

```
com.oracle.truffle.sl.SLLanguage
```

for examples of the more advanced annotations. These additional annotations will be necessary to obtain the highest performance from the Truffle framework and will be useful to you should you choose to submit your Truffle implementations for the end-of-class competition.

9 Deliverables

Students may work in pairs or alone. Please ensure that your implementation works on the provided `fibonacci.feeny` test program. Zip all source files in the `feeny` package together in a file called *assign9_XX.zip*. Replace *XX*

above with your initials. Mail the zip file to `patrickli.2001@gmail.com` with `[Feeny9]` in the subject header.