TastyTruffle: A Subtitle

by

James You

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

© James You 2022

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This is the abstract.

Acknowledgements

I would like to thank all the little people who made this thesis possible.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Tables

List of Figures

Abbreviations

TASTy Typed Abstract Syntax Tree 3

Chapter 1

Introduction

Chapter 2

Background

This section should mainly explore type erasure and how it relates to the various sections below.

2.1 Intermediate Representations

2.1.1 Java Bytecode

2.1.2 Scala Typed Abstract Syntax Trees

2.1.3 GraalVM Intermediate Representation

2.2 Managed Runtimes

2.2.1 Type Erasure

2.2.2 Just-in-time Compilation

Chapter 3

Implementation

3.1 TastyTruffle Intermediate Representation

Scala programs in TASTy format are unsuitable for execution in a Truffle interpreter. Programs in must be parsed and transformed into an executable representation in TastyTruffle. As TASTy represents a Scala program close to its equivalent source representation, the transformation of TASTy IR into TastyTruffle IR is not isomorphic. TastyTruffle IR represents a canonicalized executable intermediate representation which can be specialized on demand.

The following sections will introduce the nodes in TastyTruffle IR and how they are derived from Scala source and TASTy.

TODO: insert TASTy tree diagrams.

3.1.1 Root Node

3.1.2 Read and Write Nodes

The retrieval and storage of values in TastyTruffle IR can be divided into the

access or assignment to a local variable, access or assignment to a

```
(x: T)
```

3.1.3 Control Flow Nodes

3.1.4 Call Nodes

3.1.5 Type Nodes

3.1.6 Allocation Nodes

```
new Foo
```

```
new Array[Int]
```

```
new Array[T]
```

3.1.7 Example

```
def checksum[T](data: Array[T]): Int = {
    val sum: Int = 0
    var index: Int = 0
    while (index < data.length) {
        val sum += data[i].##
        index += 1
    }

    return sum
}
```

Figure 3.1: Example implementation of a checksum function.

3.2 Specialization

Cover the types of specializable terms.

3.3 Specializing Methods

3.3.1 Typed Dispatch

```
def checksum(T: Type, data: Array[T]): Int =
    if (T == Int)
        return checksum$Int(data.asInstanceOf[Array[Int]])

    val sum: Int = 0
    var index: Int = 0
    while (index < data.length) {
        val sum += data[i].##
        index += 1
    }
    return sum
}
```

3.3.2 Code Duplication

Specialized method for checksum

```
def checksum$Int(data: Array[Int]): Int = {
    val sum: Int = 0
    var index: Int = 0
    while (index < data.length) {
        val sum += data[i] // hash code is identity for int
        index += 1
    }
    return sum
}
```

3.3.3 Partial Evaluation

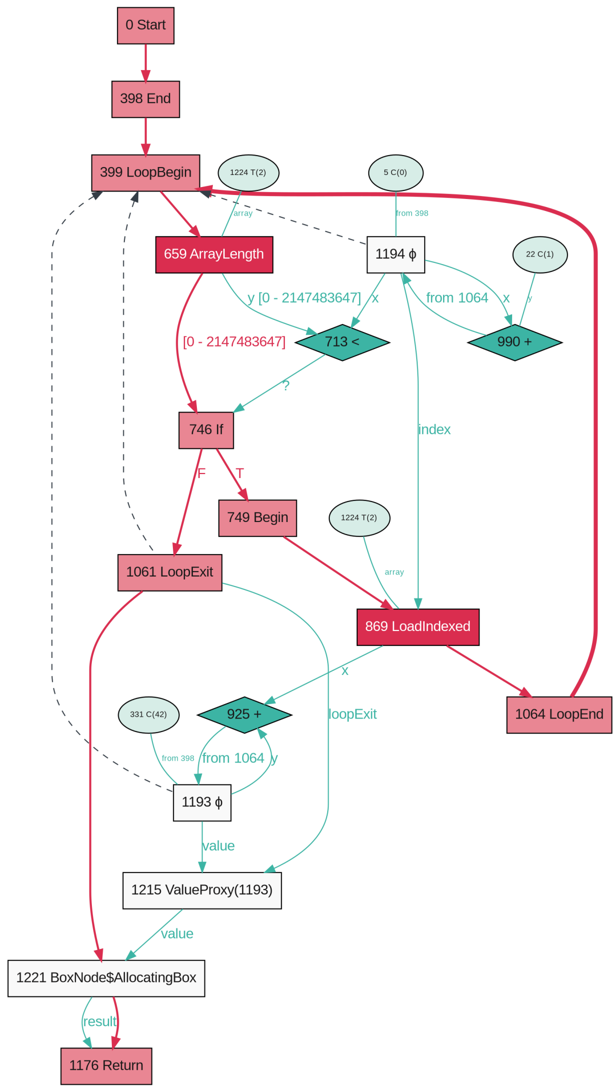

Figure 3.2: Graal IR graph of specialization `checksum[Int]` after Truffle tier.

Chapter 4

Evaluation

Chapter 5

Related Work

Chapter 6

Future Work

Chapter 7

Conclusions

References

[1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion.* Addison-Wesley, Reading, Massachusetts, 1994.

[2] Donald Knuth. *The TeXbook.* Addison-Wesley, Reading, Massachusetts, 1986.

[3] Leslie Lamport. *LaTeX — A Document Preparation System.* Addison-Wesley, Reading, Massachusetts, second edition, 1994.

APPENDICES