Optimierung mittels der Auswahl von String Repräsentationen in Java Bytecode

Markus Wondrak
Goethe Universität Frankfurt am Main
http://www.sepl.informatik.uni-frankfurt.de

August 3, 2014

Contents

List of Figures

List of Tables

Abstract

Es geht um blibla blubb

Chapter 1

Einleitung

In dieser Arbeit soll untersucht werden, ob es möglich ist...

Chapter 2

Werkzeuge

In den folgenden Abschnitten sollen die verwendeten Werkzeuge kurz vorgestellt werden. Dabei handelt es sich zum einen um den Java Bytecode, als auch um die Software Bibliothek *WALA*, auf deren API das von mir entwickelte System basiert.

2.1 Java Bytecode

Die Plattformunabhängigkeit, die in Java geschriebenen Programmen zugesprochen wird, ist vorallem mit der Rolle der Java Virtual Machine zu erklären. Java Programme werden in einen Zwischencode, den Java Bytecode, übersetzt, welcher von der System spezifischen JVM ausgeführt wird. Dabei ist Programmiersprache Java nicht die einzige in Bytecode übersetzbare Sprache. Es existieren neben den bekanntesten Scala, Jython, Groovy, JavaScript noch viele weitere. Einmal in Bytecode übersetzt in diesen Sprachen geschriebene Programme auf jeder der Java Spezifikation entsprechenden JVM aufführen.
Bytecode ist eine Sammlung von Instruktionen welche durch *opcodes* von 2 Byte Länge definiert werden. Zusätzlich können noch 1 bis n Parameter verwendet werden. Die Sprache ist Stack-orientiert, das bedeutet, dass von Operationen verwendete Parameter über einen internen Stack übergeben werden. Als Beispiel dient der folgende Bytecode:

```
ICONST 5     // legt den konstanten int Wert 5 auf den
   Stack
ILOAD 1      // läd die lokale integer Variable 1 und legt
    sie auf den Stack
```

```
IADD        // addiert die ersten beiden Werte auf dem
   Stack und legt das Ergebnis auf den Stack
ISTORE 2    // speichert den Wert auf dem Stack in der
   Variable 2
```

Dabei existiert der Stack nur als Abstraktion für den eigentlichen Prozessor im Zielsystem. Wie die jeweilige JVM den Stack in der Ziel Plattform umsetzt ist nicht definiert. Die Instruktionen lassen sich in folgende Kategorien einordnen:

- Laden und Speichern von lokalen Variablen (`ILOAD`, `ISTORE`)
- Arithmetische und logische ausdrücke (`IADD`)
- Object Erzeugung bzw. Manipulation (`NEW`, `PUTFIELD`)
- Stack Verwaltung (`POP`, `PUSH`)
- Kontrollstruktur (`IFEQ`, `GOTO`)
- Methoden Aufrufe (`INVOKEVIRTUAL`, `INVOKESTATIC`)

2.2 WALA

Bei *WALA* handelt es sich um die "T.J. Watson Library for Analysis". Eine von ehemals von IBM entwickelte Bibliothek für die statische Codeanalyse von Java- und JavaScript Programmen. Das Framework übernimmt dabei das Einlesen von *class* Dateien und stellt eine Repräsentation, die sogenannte *Intermediate Representation*, des Bytecodes zur Verfügung. Diese IR stellt die zentrale Datenstruktur dar und soll in diesem Abschnitt detailiert beschrieben werden.
Für die Manipulation des Bytecodes existiert innerhalb des Frameworks ein Unterprojekt, das diese Aufgabe übernimmt: Shrike. Im Zweiten Abschnitt soll diese API kurz vorgestellt werden.

2.2.1 IR

Die *Intermediate Representation* (IR) eine Abstraktion zum Stack basierten Bytecode. Ein IR ist in Single Static Assignment Form, welche sich dadurch auszeichnet, dass jeder Variablen immer genau **einmal** ein Wert zugewiesen

wird. Zusätzlich besteht die IR aus dem Kontroll Fluß Graphen der Methode, welcher wiederrum aus Basic Blocks zusammengesetzt ist. Ein Basic Block ist eine Zusammenfassung von aufeinander Folgende Instruktionen, welche in jedem Fall nach einander ausgeführt werden.
Die Variablen innerhalb des IRs nennt WALA *value numbers*. Diese beziehen sich immer auf eine Referenz, allerdings kann sich eine Referenz sich auf mehrere tatsächliche value numbers in der IR beziehen. Dies folgt aus der SSA Form, wird eine Variable im Bytecode zweimal ein Wert zugewiesen, wird diese doppelte Zuweisung in der SSA Form durch das Einführen einer neuen value number entfernt. Die Operationen werden auch nur mit Bezug auf die value numbers beschrieben.
Da die Zwischendarstellung vom Stack abstrahieren soll, werden auch alle Operationen, die den Stack betreffen (wie z.B. `LOAD`, `STORE`, `PUSH` oder `POP`) nicht nicht mit in diese Repräsentation übernommen. Dabei werden die Bytecode Indices der überig gebliebenen Instruktionen berücksichtigt und alle anderen Stellen in dem beinhaltenden Array mit `null` Werten aufgefüllt.
Die Verwaltung der value numbers wird von einem Typ namens `SymbolTable` übernommen. Es kommt bei der IR Erstellung zum Einsatz, wenn bei der Simulation des Bytecodes neue Variablen verwendet werden, um neue value numbers zu erzeugen.
Aufgrund der SSA Form der IR lässt sich für jede value number genau eine Definition bestimmen. Zu diesem Zweck bietet WALA den Typ `DefUse` an, welcher für jedes IR-Objekt erstellt werden kann. Er ermöglicht einen einfachen Zugriff aus die Instruktionen, die eine value number definiert (*def*; z.B. als Rückgabe aus einem Methodenaufruf), und eine Menge an Instruktionen, die die entsprechende value number verwenden (*use*; z.B. als Rückgabewert in einem `RETURN` Statement).

Anpassungen

In WALA werden beim Erstellen des IR für alle Konstanten mit demselben Wert dieselbe value numbers erzeugt. Da für die *Analyse* verschiedenen Referenzen getrennt getrennt untersucht werden mussten, wurde für die Erzeugung einer value number für eine Konstante der eingebaute caching Mechanismus umgangen.
Darüber hinaus war für die *Transformation* die Information nötig, an welcher Stelle im Bytecode eine entsprechende Konstante erzeugt wird (z.B. mit-

tels `LCD`). Um dies zu Erreichen wurde dieser Bytecode Index während dem Durchlaufen der Instrukionen innerhalb der `SymbolTable` gespeichert, sodass er beim Klienten des IRs zur Verfügung steht.
Da diese Änderungen nicht in den Haupt Branch von WALA eingepflegt werden durften, benötigt das System den Fork des WALA Projektes [1].

2.2.2 Shrike

Shrike ist ein Unterprojekt innerhalb des WALA Frameworks. Shrike übernimmt dabei das Lesen und das Schreiben von Bytecode aus bzw. in class Dateien. Dabei wird es zum einen beim Erstellen eines IR aus einer Methode verwendet, zum Anderen bietet es eine "Patch-based" API an um den Bytecode einer eingelesenen Methode zu verändern. Dies geschieht über das Einfügen von `Patch`es, welche über einen entsprechenden `MethodEditor` überall im Bytecode einer Methode eingefügt werden oder auch ursprüngliche Instruktionen komplett ersetzen. Zusätzlich enthält es einen `Verifier`, der erzeugten Bytecode überprüft, so dass ungültige Stack Zustände oder Typfehler noch während der Manipulation erkannt werden können.
In dem von mir entwickelten System werden alle Bytecode Manipulationen mit Hilfe von Shrike umgesetzt.

[1] Dieser ist unter `http://github.com/wondee/WALA` zu finden.

Chapter 3

Analyse

Das Folgende Kapitel beschreibt den Analyse Algorithmus, des von mir entworfenen Systems. Im ersten Abschnitt soll die Idee hinter der Analyse beschrieben werden. Darauf folgend wird die Datenstruktur beschrieben, auf der der Algorithmus arbeitet. Im dritten Abschnitt soll das Prinzip des *Labels* präsentiert wird und im vierten Abschnitt schliesslich der der eigentliche Algorithmus vorgestellt werden.

3.1 Idee

3.2 Datenfluß Graph

3.3 Label

3.4 Bestimmung des Bereichs

Chapter 4

Transformation

Chapter 5

Auswertung

Chapter 6

Fazit