

Aufgabenblatt zum Praktikum Optimierende Compiler

Prof. Dr. Andreas Koch
Jens Huthmann, Julian Oppermann



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Sommersemester 2013

Aufgabenblatt 4 — Abgabedatum: 30.06.2013 23:59 CEST

Einleitung

In diesem Aufgabenblatt werden Sie die Dominator Value Numbering Technique zur Redundanzeliminierung implementieren.

Aufgabe 4.1 Interaktive Oberfläche

Die von Ihnen in Aufgabe 1 erstellte interaktive Oberfläche des Bantam-Compilers soll um ein Kommando erweitert werden.

`dvnt` soll die Redundanzeliminierung mit Hilfe der Dominator Value Numbering Technique, welche Sie in Aufgabe 4.2 implementieren, für alle CFGs des Quellprogramms durchführen. Achten Sie darauf, dass hierzu der CFG in SSA-Form vorliegen muss.

Aufgabe 4.2 DVNT

Hier soll das in der Vorlesung umrissene Verfahren verwendet werden (im Buch von Cooper & Torczon aus Abschnitt 8.5.2). Dabei muss zunächst aus der IDom-Relation ein Dominator-Baum erzeugt werden. Dieser steuert dann die Bearbeitungsreihenfolge (Vorgänger müssen vor Nachfolgern behandelt werden).

Das Paper (auf der Web-Seite) “Value Numbering” von Briggs, Cooper und Simpson kann zum Nacharbeiten der Vorlesung verwendet werden. Auf den Seiten 1 bis 9 beschreibt es die Verfahren allgemein und stellt in Abbildung 4 einen konkreten Algorithmus für DVNT vor.

Dieser weist gegenüber der vereinfachten Variante aus der Vorlesung eine Erweiterung auf, die Sie ebenfalls realisieren sollen.

- Überflüssige Phi-Funktionen können vereinfacht werden. Solche Phi-Funktionen haben dieselbe Wertnummer bei allen Operanden, oder das Ergebnis hat die gleiche Wertnummer wie eine andere Phi-Funktion in diesem Block. Ersteres impliziert, dass eine Wertnummer durch die in die Phi-Funktion eingehende Operandenkante auch von einem unmittelbaren Vorgänger in den Join-Knoten übernommen werden kann, nicht nur vom IDom des Join-Knotens! Der so erweiterte Algorithmus macht also etwas mehr, als der einfache DVNT-Algorithmus.

Neben dem transformierten SSA-CFG, den man sich ja mit `dumpcfg` anzeigen lassen kann, soll Ihre Implementierung ausgeben, wie viele Berechnungen von Ausdrücken vermieden werden konnten (im Beispiel von Listing 1 also 4) und wie viele Phi-Funktionen eliminiert werden konnten (im Beispiel also 0).

Hinweise zur Implementierung

Beachten Sie, dass die Wertenummerierung von nicht lokalen Variablen durch Methodenaufrufe ungültig werden, da die Variablen durch den Aufruf verändert worden sein können.

Listing 1: Beispiel für DVNT aus Vorlesung

```
q = a + b;
r = c + d;
if (b > c) {
    e = b + 18;
    s = a + b;
    u = e + f;
} else {
    e = a + 17;
    t = c + d;
    u = e + f;
}
v = a + b;
w = c + d;
x = e + f;
```

Programmierstil

Die von Ihnen erstellten Programme werden in der Endfassung erfahrungsgemäß einige Tausend Zeilen Java umfassen. Um dem Betreuer das Verständnis und Ihnen die Wartung zu erleichtern, sollen Sie von Anfang an einen sauberen und disziplinierten Programmierstil praktizieren.

Bei der Implementierung sind die Konventionen aus Writing Robust Java Code weitgehend einzuhalten. Dieses Dokument liegt als PDF auch auf der Web-Seite der Vorlesung. Ergänzend soll folgendes beachtet werden:

- Achten Sie darauf, dass Klassen nicht zu komplex werden (zu viele Instanzvariablen, zu viele Methoden). Bei deutlich mehr als 20 dieser Konstrukte sollten Sie die Klasse aufteilen.
- Analoges gilt für die Komplexität von einzelnen Methoden. Auch hier sollten Sie bei mehr als 100 Programmzeilen Länge die Methode aufteilen.
- Verwenden Sie statt Abfragen von `instanceof` echte objekt-orientierte Konstrukte (z.B. polymorphe Methoden).

Der Test und die Abnahme Ihrer Programme wird vom Betreuer auf Linux mit dem SUN Java Development Kit (JDK) Version 1.7 erfolgen.

Dokumentation

Die Lösungen werden nur durch das unten beschriebene **README** und die in das Java-Programm eingebetteten JavaDoc-Direktiven und Kommentare dokumentiert. Achten Sie daher darauf, dass Sie von diesen beiden Möglichkeiten ausreichend und aussagekräftig Gebrauch machen!

Kommentare sollen am Anfang jeder von Ihnen modifizierten oder neu erstellten Quelldatei, pro Klasse und pro Instanzvariable und Methode verfasst werden. Bei Verwendung relativ kurzer Methoden und aussagekräftiger Bezeichner können sich Kommentare innerhalb von Methoden auf wenige wirklich wichtige Stellen beschränken. Bei komplizierteren Methoden soll der Ablauf aber durch eine größere Anzahl an aussagekräftigen Kommentaren im Methodenrumpf verdeutlicht werden.

Der Dateikopfkommentar muss neben einer allgemeinen Beschreibung auch eine Historie von Änderungen enthalten. Jeder Eintrag in dieser Historie beschreibt unter Angabe von Datum/Uhrzeit und

Namen des Autors auf 1-2 Textzeilen die Natur der Änderungen. Alternativ kann hier auch das Log Ihres Versionskontrollsystems (z.B. CVS oder besser SVN) verwendet werden.

Diese Angaben sind für den Betreuer wichtig, damit im Kolloquium die für ein Thema passenden Ansprechpartner gefunden werden!

Abgabe

Grundsätzlich schickt jede Gruppe spätestens zum Abgabetermin in einem **.jar-** oder **.zip-**Archiv die Quelldateien Ihrer Version des Bantam-Compilers an

oc@esa.informatik.tu-darmstadt.de

mit dem Subject **Abgabe 4 Gruppe N**, wobei Ihnen N bereits in der Vorlesung mitgeteilt wurde. In dem Archiv sollen nicht nur die eigenen, sondern alle (auch unmodifizierten) Quellen des Bantam-Compilers enthalten sein. Ebenso legen Sie eventuell verwendete zusätzliche externe Bibliotheken in Form ihrer jeweiligen **.jar-**Dateien bei (aber siehe Abschnitt Plagiarismus).

Am Lehrstuhl wird zur Zeit ein automatisches Testsystem entwickelt. Bitte geben Sie daher zusätzlich auch ein mit **java -jar** ausführbares **.jar-**Archiv ihres Compilers mit ab. Sie müssen also eine Manifest-Datei erstellen, die als Main-Class die Hauptklasse Ihrer Shell spezifiziert, und externe Bibliotheken auf dem Classpath ausweist.

Wichtig: Um unseren Testaufwand überschaubar zu halten, lesen Sie bitte die letzten drei Absätze noch einmal. Sie sollen abgeben:

- Die vollständigen Quellen (also die **.java-**Dateien!)
- Aber nur diese, nicht noch irgendwelche Altdaten Ihrer eigenen Testläufe (soweit nicht explizit in der Aufgabenstellung angefordert). Räumen Sie also Ihre Verzeichnisse auf, bevor Sie sie in ein Archiv packen!
- Die eventuell benötigten Fremdbibliotheken
- Ein ausführbares **.jar-**Archiv
- Alles zusammen in einem **.jar** oder **.zip-**Archiv. Also kein **.7z**, **.rar**, **.tar.gz** oder **.tbz**

Daneben enthält das Abgabearchiv eine Datei **README.txt**, die enthält

- die Namen der Gruppenmitglieder und die jeweils bearbeiteten Themen
- eine Übersicht über die neuen und geänderten Dateien mit jeweils einer kurzen (eine Zeile reicht) Beschreibung ihrer Funktion.
- Hinweise zur Compilierung der Quellen. Geben Sie eine **javac-**Kommandozeile an bzw. verweisen Sie auf mitgelieferte Makefiles oder ANT Build-Dateien. Nicht ausreichend ist ein Hinweis auf eine von Ihnen verwendete IDE (wie Eclipse, NetBeans etc.).
- Angaben über weitere Bibliotheken (beispielsweise JSAP, log4j, JUnit etc.), die Sie eventuell verwendet haben. Diese Bibliotheken legen Sie bitte dann auch als **.jar** Dateien in das abgegebene Archiv.

Gruppenarbeit

Bei den Tests von Optimierungen ist es häufig so, dass auf den ersten Blick alles funktioniert. Aber dann

in einem Quelltext ein “Sonderfall” auftaucht, der vom bisherigen Optimierungscode noch nicht oder nur fehlerhaft bearbeitet wird. Unterschätzen Sie also den Testaufwand nicht!

Falls in Ihrer Gruppe eine Situation entstehen sollte, in der einzelne Mitglieder deutlich zu wenig (oder zu viel!) der anfallenden Arbeitslast bewältigen, sprechen Sie den Betreuer bitte frühzeitig auf die Problematik an. Nur so kann durch geeignete Maßnahmen in Ihrem Interesse gegengesteuert werden. Nach der Abgabe ist es dafür zu spät und Sie tragen die Konsequenzen selber (z.B. wenn sich eines Ihrer Team-Mitglieder wegen seiner Verpflichtungen beim Wasser-Polo nur stark eingeschränkt den Mühen der Programmierung widmen konnte, und Sie daher eine unvollständige Lösung abgeben mussten).

Plagiarismus

Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Mit der Abgabe einer Lösung zu den Programmierprojekten bestätigen Sie, dass Ihre Gruppe die alleinigen Autoren des neuen Materials bzw. der Änderungen des zur Verfügung gestellten Codes sind. Im Rahmen dieser Veranstaltung dürfen Sie den Code des Bantam-Compilers von der FG ESA Web-Seite zu Optimierende Compiler sowie Code-Bibliotheken für nebensächliche Programmfunktionen (Beispiele siehe oben) frei verwenden. Mit anderen Gruppen dürfen Sie sich über grundlegenden Fragen zur Aufgabenstellung austauschen. Detaillierte Lösungsideen dürfen dagegen nicht vor Abgabe, Artefakte wie Programm-Code oder Dokumentationsteile überhaupt nicht ausgetauscht werden. Bei Unklarheiten zu diesem Thema (z.B. der Verwendung weiterer Software-Tools oder Bibliotheken) sprechen Sie bitte Ihren Betreuer gezielt an.

Weitere Infos unter www.informatik.tu-darmstadt.de/plagiarism