Aufgabenblatt zum Praktikum Optimierende Compiler

Prof. Dr. Andreas Koch
Jens Huthmann, Julian Oppermann

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 13
Aufgabenblatt 3 — Abgabedatum: 16.06.2012 23:59 CEST

Einleitung

In diesem Aufgabenblatt werden Sie zur Vertiefung der SSA Form die Rückwandlung aus dieser mit Hilfe des Briggs Algorithmus implementieren. Zusätzlich gibt es Hinweise auf mögliche Fehler in der LIVE Analyse, welche zu einem Fehlverhalten führen können.

Aufgabe 3.1 Interaktive Oberfläche

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

`briggs` soll die Rückumwandlung aus der SSA-Form, welche Sie in Aufgabe 3.2 implementieren, für alle CFGs des Quellprogramms durchführen.

Aufgabe 3.2 Rückwandlung aus SSA-Form mit dem Briggs Algorithmus

Ihre Aufgabe ist es die Rückwandlung eines CFG in SSA-Form zu einem normalen CFG mit dem in der Vorlesung vorgestellten Briggs Algorithmus zu implementieren. Verwenden Sie hierbei Ihre im vorherigen Aufgabenblatt erstellte Live-Variables Analyse.

Geben Sie am Ende eine Statistik darüber aus wie viele Φ-Anweisungen Sie entfernt haben und wie viele Kopieranweisungen Sie hinzugefügt haben. Verwenden Sie für die Ausgabe die in Listing 1 gegebenen Formatierungsstrings. Setzen Sie ihre Statistik zurück, wenn Ihrer Briggs Implementierung ein neuer Start BasicBlock mit `setEntryBlock` zugewiesen bekommt.

Listing 1: Formatierungsstring für Statistik

```
"Removed phi-instructions: %d"
"Inserted copies         : %d"
```

Hinweise zur Implementierung

- Bauen Sie sich den für Briggs Algorithmus benötigten Dominator Tree aus den Immediate Dominator jedes Blocks zusammen. Den Immediate Dominator erhalten Sie mit `BasicBlock.getImmediateDominator()`.
- Um neue temporäre Variablen zu erstellen verwenden Sie die Methode `Optimizer.getNextTmp()`.
- Eine TAC Kopieranweisung erstellen Sie mit `new LoadVarInst(destination, src)`.
- Mit Hilfe von `Optimizer.getSources` (2 Varianten) können Sie die Operanden einer Instruktion zu ermitteln. Mit `Optimizer.getDestination` ermitteln Sie das Ziel.

- Durch die Anwendung von Optimierungen wie Copy Propagation kann es dazu kommen das Φ-Anweisungen redundant werden. Dies ist der Fall, wenn alle Operanden Φ-Anweisung dem Ziel entsprechen, d.h. $x = \Phi(x, x)$. Solche redundanten Φ-Anweisungen können Sie mit Hilfe der Methode `PhiInst.isCompletelyRedundant()` erkennen und anschließend entfernen.

- Alle Details zur `Optimizer` Klasse finden Sie in der aktualisierten API Beschreibung.

Mögliche Fehlerquellen in der LIVE Analyse

Da eine korrekte LIVE Analyse essentiell für die korrekte Funktionsweise des Briggs Algorithmus ist, ist im folgenden die wahrscheinlichste Fehlerquelle aufgeführt.

- Bei Methodenaufrufen mit `DirCallInsts` und `InDirCallInsts` muss beachtet werden, dass die aufgerufene Methode alle Attribute benutzen (lesen oder schreiben) kann. Ohne eine globale Analyse, die über Methodengrenzen hinweg arbeitet, müssen daher beim Finden eines Methodenaufrufes alle Attribute der Klasse sowohl als Benutzt (USE, gelesen) als auch als Definiert (DEF, geschrieben) angenommen werden. Aufrufe von Bibliotheksfunktionen können ignoriert werden, da diese garantiert keine Attribute verändern.

  Genauer: Dies gilt für `this` und `super`-Aufrufe, da andere Klassen aufgrund der Sichtbarkeit von Attributen in Bantam nicht auf diese zugreifen können. Für `this`-Aufrufe müssen alle im CFG tatsächlich auftretenden Attribute als USE/DEF gesetzt werden, bei `super`-Aufrufen würden nur die aus den Oberklassen ererbten und im CFG verwendeten Attribute ausreichen (diese verfeinerte Unterscheidung sollen Sie aber nicht machen).

  Vorgehen: Zur Vereinfachung können Sie mittels `Optimizer.isCallToBuiltIn` prüfen, ob der Methodenaufruf ignoriert werden kann (Bibliotheksfunktion). Falls nicht, nehmen Sie an, dass alle Variablen, die von `Optimizer.getFieldsUsedInCFG` als im CFG benutzt geliefert werden, an der Stelle des Methodenaufrufes als USE/DEF behandelt werden müssen.

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 aussagekräftige 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 3 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 nocheinmal. 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`