Aufgabenblatt zum Praktikum Optimierende Compiler

Prof. Dr. Andreas Koch
Jens Huthmann, Florian Stock

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 12
Aufgabenblatt 3 — Abgabedatum: 10.06.2012 23:59 CEST

Einleitung

Als erste einfache Optimierungsverfahren sollen Sie in dieser Aufgabe die Propagierung von Kopieranweisungen (Copy-Propagation) und die Propagierung von Konstanten, auf Basis eines allgemeinen Datenflussanalyse Frameworks implementieren.

Aufgabe 3.1 Interaktive Oberfläche

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

`cp` soll den Copy-Propagation Algorithmus welchen Sie in Aufgabe 3.3 implementieren auf alle CFGs des Quellprogramms anwenden.

`csp` soll den Constant-Propagation Algorithmus welchen Sie in Aufgabe 3.5 implementieren auf alle CFGs des Quellprogramms anwenden.

`dce` soll den bereitgestellten Dead-Code-Elimination Algorithmus (siehe Aufgabe 3.4) auf alle CFGs des Quellprogramms anwenden.

Aufgabe 3.2 Datenflussanalyse Framework

Ihre Aufgabe ist es ein Datenflussanalyse Framework zu entwickeln welches es ermöglicht alle Varianten der Datenflussanalyse zu implementieren. Hierbei soll es einfach möglich sein, neue Analysen, durch Austauschen der Parameter für die c_1, c_2, `in` und `out` Definitionen sowie der Traversierungsrichtung der Kanten im CFG, zu erzeugen.

Das Datenflussproblem kann durch die folgenden Gleichungen abstrakt beschrieben werden. op_1, op_2 und CF stehen hierbei für die Operatoren, c_1 und c_2 für durch die Anweisungen in jedem Block erzeugten Mengen. DIR steht schließlich für die Richtung der Analyse.

$$IN(n) = \underset{d \in DIR}{CF}\, OUT(d)$$
$$OUT(n) = c_1(n)\,\mathrm{op}_1(IN(n)\,\mathrm{op}_2\, c_2(n))$$

Für die in der Vorlesung erläuterte Copy-Propagation sind folgende Zuordnungen gegeben.

$$DIR = PRED$$
$$CF = \cap$$
$$c_1 = COPY$$
$$c_2 = KILL$$
$$op_1 = \cup$$
$$op_2 = -$$

Eine ausführliche Beschreibung können Sie den Vorlesungsfolien entnehmen. In der Vorlesung werden diese nächste Woche durchgenommen.

Hinweise zur Implementierung

Damit Sie Ihre Implementierung der Datenflussanalyse für alle Arten von Datenflussproblemen verwenden können, ist es notwendig, dass Sie die Parameter der Analyse möglichst einfach verändern können.

Beachten Sie hierbei, dass c_2 und c_1 von den TAC Instruktionen abhängig sind, `in` und `out` aber hingegen für ganze Basisblöcke definiert werden. Zusätzlich kann es sein, dass man die Analyse auf einen bestimmten Bereich zwischen Grenz-Basisblöcken einschränken möchte. Im Allgemeinen sind diese Grenzen der Anfangs- und Endblock eines CFGs. Auch die Richtung mit welcher die Kanten verfolgt werden, kann sich bei den verschieden Analysen zwischen Vorwärts und Rückwärts unterscheiden.

Als Beispiel sehen Sie in Listing 1 eine Möglichkeit für eine abstrakte Definition der c_2 und c_1 Mengen. Die Definition von `in` und `out` sowie die weitere Implementierung des Datenflussanalyse Frameworks bleibt Ihnen überlassen. Das Beispiel für c_2 und c_1 ist nur ein Vorschlag, Sie müssen es nicht verwenden.

Listing 1: Abstrakte Datenflussanalyse

```
abstract class DFSet<ResultType> extends TACInstVisitor {
  public Set<ResultType> get(TACInst inst, Set<ResultType> in) {
    return (Set<ResultType) inst.accept(this, in)
  }
}

abstract class C1<ResultType> extends DFSet<ResultType> {
}

abstract class C2<ResultType> extends DFSet<ResultType> {
}
```

Aufgabe 3.3 Copy-Propagation

Mit Hilfe des von Ihnen in Aufgabe 3.2 entwickelten Datenflussanalyse Frameworks sollen Sie nun Ihre erste Optimierung entwickeln.

Diese ist Propagierung von Kopieranweisungen oder Copy-Propagation aus der Vorlesung. Sehen wir uns das Beispiel in Listing 2 an.

Listing 2: Beispielprogramm vor Copy-Propagation

```
a = y + z;
x = y;
b = x + z;
```

Copy-Propagation erlaubt es uns in der dritten Anweisung `x` durch `y` zu ersetzen und die nun unnötige Kopieranweisung mit Dead-Code-Elimination zu entfernen (siehe Listing 3). Dies ermöglicht es uns zum Beispiel zu erkennen dass die Werte `a` und `b` identisch sind. Eine Optimierung für solche Fälle werden Sie später noch kennenlernen.

Listing 3: Beispielprogramm nach Copy-Propagation und Dead-Code-Elimination

```
a = y + z;
b = y + z;
```

Hinweise zur Implementierung

Implementieren Sie das in Vorlesung vorgestellte Verfahren zur Copy-Propagation unter Verwendung Ihres Datenflussframeworks. Sie müssen bei der Copy-Propagation nur die Erkennung und Propagierung der Kopien implementieren. Zur Entfernung der nun unnötigen Kopieranweisungen können Sie die bereits im Compiler enthaltene Dead-Code-Elimination (DCE) verwenden.

Aufgabe 3.4 Dead-Code-Elimination

Verwenden Sie die bereitgestellte Dead-Code-Elimination mit dem in Listing 4 gezeigten Aufruf. Die Klasse `OptimizationFactory` finden Sie im Paket `opt`, in der aktualisierten Version unserer Bantam Distribution.

Listing 4: Aufruf der Dead-Code-Elimination

```
// perform dead code elemination
final Optimization dce = OptimizationFactory.createDCE(false);
optimizer.optimize(dce);
```

Aufgabe 3.5 Constant-Propagation

Als zweite Optimierung sollen Sie Propagierung und Auflösung von konstanten Ausdrücken implementieren. In Listing 5 welches Sie auch bereits aus der Vorlesung kennen, können mit Hilfe der Constant-Propagation einige Operation schon während der Compilierung ausgerechnet werden. Dies führt zu dem Ergebnis in Listing 6

Listing 5: Beispiel vor Constant-Propagation

```
a = 5;
b = 3;
x = a + b;
p = q + a;
d = 0;
if(q > 0) {
  d = 1;
  c = 2 * a;
} else {
  c = 3 * b + 1;
  d = 2;
}
y = c + 7;
z = 4 * d;
```

Listing 6: Beispiel nach Constant-Propagation

```
a = 5;
b = 3;
x = 8;
p = q + a;
d = 0;
if(q > 0) {
  d = 1;
  c = 10;
} else {
  c = 10;
  d = 2;
}
y = 17;
z = 4 * d;
```

Hinweise zur Implementierung

Implementieren Sie das in der Vorlesung vorgestellte Verfahren zur Constant-Propagation unter Verwendung Ihres Datenflussframeworks. Denken Sie daran das Sie bei dem Auflösen der Operatoren auch Überläufe der Zahlenbereiche beachten müssen. Auch dürfen Sie nicht vergessen, dass man auch logische Operationen vorberechnen kann, soweit ihre Operatoren konstant sind.

Es ist nicht Aufgabe dieser Optimierung Code zu entfernen, welcher durch potentiell konstante Bedingungen für einen Branch nicht mehr erreichbar und daher unnötig geworden ist.

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).

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
- Alles zusammen in einem `.jar` oder `.zip`-Archiv. Also kein `.7z`, `.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`