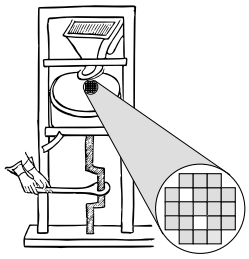
Technische Universität Darmstadt
FG Eingebettete Systeme und
ihre Anwendungen (ESA)

Prof. Dr. Andreas Koch



"Optimierende Compiler"
Optionale Aufgabe 5: Aufräumungsarbeiten
Abgabe bis zum 21.7.2006, 12:00 MET mittags

1 Einleitung

Bei der Rückwandlung des CFGs aus der SSA-Form entstehen häufig eine Unzahl von Kopieranweisungen, die es zu eliminieren gilt. Das Entfernen aller unnötigen Anweisungen, die sich im Laufe der Optimierung angesammelt haben, sowie das Beseitigen von unnützem Kontrollfluß im CFG, sollte daher in in spezialisierten Passes verfügbar sein.

Die Lösung dieser Aufgabe ist optional, *kann* aber zur Verbesserung Ihrer Endnote beitragen (siehe Abschnitt 4).

2 Problemstellung

2.1 Interaktive Oberfläche

Die von Ihnen in Phase 2 erstellte interaktive Oberfläche des Triangle-Compilers soll um bis zu drei Kommandos erweitert werden:

copyprop soll den Copy Propagation-Algorithmus auf den CFG anwenden

dead soll den Algorithmus DEAD auf den SSA-CFG anwenden

clean soll den Algorithmus CLEAN auf den CFG anwenden

Als Voraussetzung für die Implementierung dieser neuen Kommandos müssen Sie die Funktion eines bereits bestehenden Kommandos aufteilen: `ssa2ast` sollte nun in `ssa2cfg` und ein `cfg2ast` unterteilt werden. Ersteres beseitigt die Phi-Funktionen, lässt den CFG aber noch für die Kommandos `copyprop` und `clean` intakt. `cfg2ast` übersetzt den dann weiter optimierten CFG in den AST.

Zur Wertung dieser Aufgabe ist auch eine teilweise Bearbeitung möglich. Das heisst, das Sie auch nur eine Untermmenge der drei oben beschriebenen Algorithmen implementieren können.

2.2 Copy Propagation

Hier soll das in der Vorlesung umrissene Verfahren verwendet werden. Weiterführende Informationen finden sich beispielsweise in "Advanced Compiler Design and Implementation" von S. Muchnick, in Abschnitt 12.5. Kernkomponenten sind das Aufstellen des Datenflußproblems aus dem CFG,

seine iterative Lösung und auch die Implementierung der lokalen und globalen Formen des Copy Propagation-Algorithmus (letzterer aufbauend auf der Lösung des Datenflußproblems). Dieser Algorithmus soll sowohl in der SSA als auch in der nicht-SSA-Form lauffähig sein sollen.

2.3 Dead

Wie in der Vorlesung besprochen, soll diese Phase in jedem Fall auf der SSA-Form laufen (anderenfalls wäre die Berechnung der DU- und UD-Chains zu aufwendig). Am Ende soll eine Statistik ausgegeben werden, wieviele Anweisungen eliminiert werden konnten.

Weiterführende Informationen zu dem Algorithmus finden sich beispielsweise im Paper von Cytron et al., Abschnitte 6 und 7.1.

2.4 Clean

Auch diese Operation soll sich am Algorithmus aus der Vorlesung orientieren. Sie soll sowohl auf der der SSA- als auch in der nicht-SSA-Form lauffähig sein. Die Statistik hier macht Angaben über die Anzahl der aus dem CFG entfernten Knoten und Kanten.

Die Komplexität der Lösung dieser Aufgabe hängt nicht unerheblich davon ab, wie Sie im CFG Sprünge und Verzweigungen zwischen den Blöcken realisiert haben. Bei ungeschickter Modellierung wird das Finden der CFG-Muster für jede Teiltransformation bzw. die tatsächliche Transformation eher unhandlich werden. Dann müssten Sie gegebenenfalls noch ihre bestehenden CFG-Klassen verbessern.

3 Abgabe

Jede Dreiergruppe schickt spätestens zum Abgabezeitpunkt in einem Archiv alle Dateien ihrer Version des Triangle-Compilers an

`oc06@esa.informatik.tu-darmstadt.de`

mit dem Subject `Abgabe 5 Gruppe` *N*, wobei Ihnen *N* bereits via Email mitgeteilt wurde.

In dem Archiv sollen nicht nur die eigenen, sondern *alle* (auch unmodifizierten) Quellen des Triangle-Compilers enthalten sein. Dazu kommen noch die neue(n) Testeingabedatei(en) im Triangle-Subset.

Jede der von Ihnen modifizierten oder neu erstellten Quelldateien trägt oben einen Kommentarkopf, der die Funktion der Datei sowie die im Laufe ihrer Entstehungsgeschichte vorgenommenen Änderungen dokumentiert. Zu jeder Änderung **muß** der entsprechende Autor angegeben werden. Aufgrund dieser Daten erfolgen dann gezielte Nachfragen in den Kolloquien.

Neben dem Kopfkommentar versehen Sie auch die einzelnen von Ihnen neu eingeführten oder veränderten Methoden und Instanzvariablen mit aussagekräftigen Kommentaren entsprechend den JavaDoc-Konventionen. Innerhalb der Methoden beschreiben Sie durch aufschlußreiche Kommentare den allgemeinen Ablauf, der auf einer höheren Abstraktionsebene als die der einzelnen Java-Anweisungen beschrieben werden soll.

Bei der Programmierung verwenden Sie einen einheitlichen, gut lesbaren Stil. Als Vorschlag dazu seien hier die AmbySoft Java Coding Guidelines genannt (auf dem OC06 Web-Site verfügbar).

Neben den Java-Quelltexten enthält das Abgabearchiv eine Datei README.txt, die enthält

- die Namen der Gruppenmitglieder.
- eine kurze Beschreibung, was welches Gruppenmitglied zur Lösung beigetragen hat.

- eine Übersicht über die neuen und geänderten Dateien mit jeweils einer kurzen (eine Zeile reicht) Beschreibung ihrer Funktion.
- Hinweise zur Compilierung der Quellen. Hier geben Sie bitte auch an, falls Sie weitere Bibliotheken (beispielsweise JSAP, log4j, JUnit etc.) verwendet haben. Diese Bibliotheken legen Sie bitte dann auch als .jar Dateien in das abgegebene Archiv.

4 Beurteilung

Da es sich hierbei um eine optionale Aufgabe handelt, ist auch der Eingang in die Bewertung variabel: Die Lösung dieser Aufgabe kann die Note der Kernaufgaben *nicht* verschlechtern. Wenn sie in der Lösungsqualität den Pflichtlösungen entspricht, wird die Endnote damit aufgewertet (wegen des zusätzlichen Arbeitsaufwands). Wenn die optionale Lösung die Qualität der Pflichtlösungen auch noch übertrifft, findet eine weitere Aufwertung statt. Die Pflichtlösungsnote kann aber maximal um eine ganze Note verbessert werden (das hier ist schließlich nur eine von fünf Abgaben).

5 Anregungen zur Gruppenarbeit

Falls alle der Teilaufgaben oben bearbeitet werden sollen, ist folgende Aufteilung naheliegend:

- Copy Propagation
- DEAD
- CLEAN

Wobei Copy Propagation dabei sicherlich die komplizierteste, und CLEAN die einfachste Optimierung ist. Hier bietet sich also eine Verteilung angepasst an die Fähigkeiten der einzelnen Gruppenmitglieder an.

Falls auszugsweise nur eine der optionalen Teilaufgaben gelöst werden soll, könnte z.B. die Copy Propagation wie folgt angegangen werden:

- Aufstellen des Datenflußproblems aus dem CFG und seine iterative Lösung
- Implementierung der Algorithmen für lokale und globale Copy Propagation, aufbauend auf der Lösung des Datenflußproblems
- Test aller Komponenten

Der Tester muss dabei mit der Arbeitsweise beider Verfahren (aber nicht zwangsläufig ihrer Implementierung) vertraut sein und geeignete Testfälle konstruieren, die potentielle Fehlermöglichkeiten weitgehend abdecken. Die Testfälle bestehen jeweils aus dem Quellcode im Triangle-Subset, den erwarteten SSA-CFGs vor und nach der DVNT-Optimierung, den CFGs nach der Rückwandlung aus SSA-Form, den CFGs nach Anwendung der Copy Propagation, sowie dem erwarteten AST nach deren Rückwandlung. Damit ist dann eine Kontrolle des ganzen Compile-Flusses möglich.

6 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 Triangle-Compilers vom OC06 Web-Site 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 an.