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

Prof. Dr. Andreas Koch

embedded
systems +
applications

"Optimierende Compiler"
Aufgabe 4: Redundanzeliminierung durch Optimierung mit DVNT und Dead-Code Elimination
Abgabe bis zum 5.7.2010, 23:59 Uhr MET DST

1 Einleitung

Als wesentliche Optimierungsverfahren der Vorlesung sollen Sie in dieser Phase die Dominator-basierte Wertnumerierung (DVNT) und das anschließende Aufräumen mit Dead-Code Elimination (DCE) implementieren. Beides soll auf dem SSA-CFG stattfinden.

2 Problemstellung

2.1 Interaktive Oberfläche

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

dvnt soll den DVNT-Algorithmus auf die SSA-CFGs aller Prozeduren anwenden. Dabei können Sie mittels eines optionalen Parameters den Namen nur einer zu bearbeitenden Prozedur angeben. Der Name `main` soll dabei für das Hauptprogramm stehen. Durch einen Parameter `-v` sollen Statistiken ausgegeben werden, sonst soll das Kommando keine Ausgaben haben.

dce soll den DCE-Algorithmus auf die SSA-CFGs aller Prozeduren anwenden. Dabei können Sie mittels eines optionalen Parameters den Namen nur einer zu bearbeitenden Prozedur angeben. Der Name `main` soll dabei für das Hauptprogramm stehen. Durch einen Parameter `-v` sollen Statistiken ausgegeben werden, sonst soll das Kommando keine Ausgaben haben.

2.2 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, die während der SSA-Wandlung nach Brandis und Mössenböck bestimmt werden konnte, 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 zwei Erweiterungen auf, die Sie *optional* mit der Möglichkeit der Notenverbesserung realisieren können:

```
...
if true then
  a := a + 1
else
  a := a + 2
...
```

Abbildung 1: Eingabe der DCE-Vorverarbeitung

```
...
a := a + 1
...
```

Abbildung 2: Ausgabe der DCE-Vorverarbeitung

1. 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.
2. Dieser Algorithmus nimmt vor der eigentlichen Wertnumerierung auch noch eine Vereinfachung von Ausdrücken vor (z.B. wird `x+0` zu `x`). Hier können Sie auch die in Vorlesung vorgestellte Berechnung von konstanten Teilausdrücken zur Compile-Zeit realisieren. Beispiel: `x+4+5` $\rightarrow$ `x+9`. Aber Vorsicht bei Klammerung und Punkt- vor Strichrechnung!

Zur Vereinfachung kann sich Ihre DVNT-Realisierung auf die Bearbeitung *vollständiger* Ausdrücke beschränken. Das heisst, dass bei den Anweisungen

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

keine Wiederverwendung eines Teilausdrucks von `x` nach `y`, sondern nur die des vollständigen Ausdrucks von `x` nach `z` erkannt werden muss. Sie können aber *optional*, ebenfalls mit der Möglichkeit der Notenverbesserung, auch die Bearbeitung von Teilausdrücken realisieren.

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

Geben Sie bei Verwendung von `-v` bei diesem Kommando Statistiken aus, wieviele Ausdrücke wiederverwendet werden konnten.

2.3 DCE

Verwenden Sie das in der Vorlesung gezeigte (und auch im Buch von Cooper & Torczon beschriebene) Verfahren, um nicht benötigte Anweisungen nun tatsächlich zu entfernen.

Stellen Sie dazu sicher, dass Sie Verzweigungen mit konstanten Bedingungen (der Form `if true then ...`, diese können ja nach DVNT entstehen) umschreiben in unbedingte Sprünge und den anderen Zweig (hier also den `else`-Zweig) bis zum passenden Join-Knoten entfernen (im entfernten Zweig können ja wieder Verzweigungen eingeschachtelt sein).

Abbildung 1 und 2 zeigen eine Beispieleingabe und das gewünschte Ergebnis dieses Teilschrittes.

Analoges gilt auch für `while`-Schleifen: Solche mit einer Bedingung von konstant *false* können entfernt werden, bei solchen mit konstant *true* sollten Sie eine Warnung zu einer Endlosschleife ausgeben.

Geben Sie bei Verwendung von `-v` bei Aufruf des Kommandos für die Vorverarbeitung und DCE selbst Statistiken aus, wieviele `if` und `while`-Konstrukte Sie vorweg ersetzen konnten sowie die Anzahl von Anweisungen, die durch DCE entfernt werden konnten.

Für Zweiergruppen: Diese Aufgabe ist für Sie *optional* und kann zur Notenverbesserung dienen. Sie sollten sie aber *nur* in Angriff nehmen, wenn Ihre DVNT-Optimierung fehlerfrei funktioniert. Zwei unsichere Optimierungsverfahren nutzen niemandem!

Allgemeiner Hinweis: Beide Optimierungen dürfen die ausgeführte Funktion des eingegeben Programmes *nicht* beeinflussen. Sie sollten also durch Tests sicherstellen, dass bei den gleichen Eingaben optimierte und unoptimierte Fassung immer noch die gleichen Ausgaben haben!

3 Abgabe

Es gelten auch hier die auf dem ersten Aufgabenblatt beschriebenen Anforderungen an **Programmierstil** und **Dokumentation**. Bitte lesen Sie die entsprechenden Abschnitte falls nötig nocheinmal!

Jede Gruppe schickt spätestens zum Abgabezeitpunkt in einem `.jar`/`.zip`-Archiv alle Dateien ihrer Version des Triangle-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 Triangle-Compilers enthalten sein. Ebenso legen Sie eventuell verwendete zusätzliche externe Bibliotheken in Form ihrer jeweiligen `.jar`-Dateien bei (aber siehe Abschnitt 6).

Bedenken Sie auch, dass in der späteren Endabgabe auch die **Kommentierung** relevant ist: 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. Letzteres kann auch durch Beilegen der Logs ihres Versionskontrollsystems (z.B. SVN oder CVS) geleistet werden. Neben dem Kopfkommentar sind auch die einzelnen von Ihnen neu eingeführten oder veränderten Methoden und Instanzvariablen mit aussagekräftigen Kommentaren entsprechend den JavaDoc-Konventionen versehen. 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 OC Web-Site verfügbar). Es kann später zeitsparend sein, diese Anforderungen auch schon bei dieser Zwischenabgabe zu befolgen.

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

- die Namen der Gruppenmitglieder.
- 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.
- Beschreibungen zu etwaigen Fehlerkorrekturen zu den früheren Phasen.

- für alle Beispielprogramme (die Sie ja in der ersten Aufgabe auf das auch hier zu bearbeitende Subset von Triangle angepasst haben) die Ausgaben von `dumpcfg` für alle Prozeduren (als einzelne Dateien) nach `read` / `check` / `ast2ssa` / `dvnt` / `dce` sowie nach weiteren . . . / `ssa2ast` für `showast`
- den Output der ausgeführten Beispielprogramme, jeweils nach `read` / `check` / `codegen` / `write` (in eine Datei *progname*`-orig.out`) sowie nach `read` / `check` / `ast2ssa` / `dvnt` / `dce` / `ast2ssa` / `codegen` / `write` (in eine Datei *progname*`-opt.out`). Hinweis: Diese Dateien sollten identisch sein!
- Für jedes Beispielprogramm die DVNT und DCE Statistiken in Dateien *progname*`.stat`.

4 Beurteilung

Die Kolloquien finden wieder zu den im Forum entsprechend angebotenen Terminen statt. Diese Kolloquien sind Bestandteil der Prüfungsleistung, es besteht daher **Anwesenheitspflicht** für alle Gruppenmitglieder. Ein Kolloquium dauert jeweils ca. 20 Minuten.

5 Anregungen zur Gruppenarbeit

Eine mögliche Aufteilung ist beispielsweise diese: Nach sorgfältiger gemeinsamer Planung der Datenstrukturen und Schnittstellen können folgende Aufgaben parallel erledigt werden:

- Realisierung des Kern-DVNT-Algorithmus (CFG traversieren, Hashtabelle und Verwaltung von Wertnummern)
- Handhabung von algebraischen Eigenschaften (z.B. $n \cdot 1 = n$, $n + 0 = n$) und konstanten Teilausdrücken (z.B. $4 + 7 = 11$, $3 > 4 =$ false), beides aufbauend auf den Schnittstellen zum Kern-DVNT.
- DCE und Vorverarbeitung (Eliminieren konstanter Verzweigungen)

Da in dieser Aufgabe ein höherer Implementierungsaufwand ansteht, schlagen wir vor, das Testen gleichberechtigt auf alle Teilnehmer zu verteilen. Dabei bietet es sich an, dass die Teilnehmer Tests jeweils für eine andere Teilaufgabe als ihre eigene bereitstellen (Black-Box Tests).

Die Tester müssen dabei mit der *Arbeitsweise* der DVNT- und DCE-Verfahren (aber nicht zwangsläufig ihrer Implementierung) vertraut sein und geeignete Testfälle konstruieren, die potentielle Fehlermöglichkeiten weitgehend abdecken. Die Testfälle bestehen jeweils aus dem Triangle-Quellcode und den erwarteten SSA-CFGs vor und nach den DVNT und DCE-Optimierungen. Hinweis: Die Beispielprogramme sind als Eingaben für die ersten Tests bereits **viel zu kompliziert**! Schreiben Sie *gezielt* kleinere Testprogramme, die "Futter" für die Optimierung enthalten und auch Sonderfälle abdecken.

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 OC Web-Site sowie Code-Bibliotheken für nebensächliche Programmfunktionen frei verwenden. Mit anderen Gruppen dürfen Sie sich gerne über grundlegende Fragen zur Aufgabenstellung austauschen. Detaillierte Lösungsideen dürfen dagegen *nicht vor Abgabe*, Artefakte wie Programm-Code oder Dokumentationsteile *überhaupt nicht* aus-

getauscht werden. Bei Unklarheiten zu diesem Thema (z.B. der Verwendung weiterer Software-Tools oder Bibliotheken) sprechen Sie bitte Ihren Betreuer gezielt an.