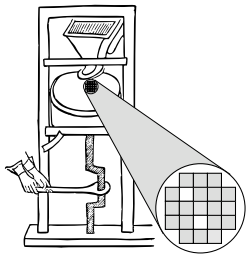
Technische Universität Darmstadt
FG Eingebettete Systeme und
ihre Anwendungen (ESA)

Prof. Dr. Andreas Koch



"Optimierende Compiler"
Aufgabe 4: Optimierung in SSA-Form und Rückwandlung in AST
Abgabe bis zum 30.06.2008, 23:59 Uhr MET DST

1 Einleitung

Zum Abschluss der Veranstaltung sollen Sie drei Techniken zu einer Gesamtlösung kombinieren. Es handelt sich dabei in um Copy Propagation und Rückwandlung aus der SSA-Form, beides aufbauend auf der Datenflussanalyse.

Durch Copy Propagation werden verwendete Kopien einer Variable durch die Originalvariable ersetzt. Für die Erzeugung von Maschinencode ist es anschließend erforderlich, den optimierten SSA-CFG wieder in einen DAST rückzuwandeln. Dabei müssen in erster Linie die Phi-Funktionen aufgelöst werden.

2 Problemstellung

2.1 Interaktive Oberfläche

Die von Ihnen in Phase 1 erstellte interaktive Oberfläche des Triangle-Compilers soll um vier Kommandos erweitert werden. Zwei davon geben die Ergebnisse der Datenflußanalyse aus, auf die die beiden anderen Kommandos aufbauen.

flowcopy soll auf die SSA-CFGs aller Prozeduren (einschliesslich des Hauptprogramms) eine Datenflußanalyse ausführen und für jeden Block des CFG die Mengen COPY, KILL, CPIN und CPOUT ausgeben. Bei Angabe eines optionalen Prozedurnamens als Parameter wird nur diese Prozedur analysiert.

flowlive soll auf die SSA-CFGs aller Prozeduren (einschliesslich des Hauptprogramms) eine Datenflußanalyse ausführen und für jeden Block des CFG die Mengen UEVAR, VARKILL und LIVEOUT ausgeben.Bei Angabe eines optionalen Prozedurnamens als Parameter wird nur diese Prozedur analysiert.

copyprop soll auf SSA-CFGs aller Prozeduren (einschliesslich des Hauptprogramms), aufbauend auf den Ergebnissen der `flowcopy`-Datenflussanalyse eine globale Copy-Propagation durchführen.Bei Angabe eines optionalen Prozedurnamens als Parameter wird nur diese Prozedur optimiert.

ssa2ast soll die SSA-CFGs aller Prozeduren (einschliesslich des Hauptprogramms) wieder in die normale Form (ohne Phi-Funktionen) umwandeln und das Ergebnis als für die Code-Erzeugung

geeigneten DAST ablegen. Bei Angabe eines optionalen Prozedurnamens als Parameter wird nur diese Prozedur wieder in den DAST rückübersetzt, die anderen Prozeduren sollen im DAST unverändert bleiben.

2.2 Datenflussanalyse

Die `flowcopy` und `flowlive` Kommandos sollen ihre Ergebnisse sowohl in Ihren internen Datenstrukturen ablegen (damit sie weiter verwendet werden könen, siehe unten) als auch auf die Konsole ausgeben. Dabei soll als Ausgabeformat das im 6. Vorlesungsblock für die Copy Propagation gezeigte für beide Kommandos benutzt werden. Für einen CFG für die Prozedur foo mit den Blöcken B1, B2 und B3 könnte `flowcopy` folgende Ausgabe liefern (sinnlose Beispiele):

```
CFG: foo
Copy(B1) = {<d\_4,c\_3,B1,2>}
Copy(B2) = {<d\_4,g\_2,B2,2>}
Copy(B3) = {}
Kill(B1) = {<d\_4,g\_2,B2,2>}
Kill(B2) = {}
Kill(B3) = {}
CPin(B1) = {}
CPin(B2) = {<d\_4,c\_3>,<g\_2,e\_1>}
CPin(B3) = {<d\_4,c\_3>}
CPout(B1) = {<d\_4,c\_3>}
CPout(B2) = {<d\_4,c\_3>,<g\_2,e\_1>}
CPout(B3) = {<d\_4,c\_3>}
```

Ein Beispiel für `flowcopy` könnte liefern:

```
CFG: foo
UEVar(B1) = {a\_2}
UEVar(B2) = {b\_3,c\_1}
UEVar(B3) = {c\_1}
VarKill(B1) = {}
VarKill(B2) = {a\_2}
VarKill(B3) = {b\_3}
LiveOut(B1) = {}
LiveOut(B2) = {c\_1}
LiveOut(B3) = {}
```

In beiden Fällen sind die Versionsnummern der SSA-Wertinstanzen durch einen Unterstrich vom Variablennamen getrennt.

Behandeln Sie zusammengesetzte (`record`) und Array-Typen wie in Aufgabe 2 (Erzeugung der SSA-Form) beschrieben. Definieren Sie neben den oben gezeigten interaktiven Kommandos auch ein API, so dass die nachfolgenden Teilaufgaben eine Analyse auslösen können und die Ergebnisse abrufen. Dazu ist es sinnvoll, keine Textdarstellung zu verwenden (Strings sind böse!), sondern beispielweise den DAST bzw. Ihren CFG um entsprechend verkettete Objekte zu erweitern.

Zur Lösung der Datenflussprobleme können Sie einen einfachen iterativen Algorithmus verwenden. Eine Optimierung auf niedrige Iterationszahl (durch geeignete Rechenreihenfolge) ist *nicht* verlangt, kann aber optional realisiert werden und könnte damit die Gesamtbewertung verbessern.

Als weitere optionale Aufgabe können Sie `dumpcfg` so erweitern, dass auch die Ergebnisse der Datenflußanalysen an die ausgegebenen CFGs annotiert werden und so auch in den dot-Grafiken auftaucht.

2.3 Globale Copy-Propagation

Hier sollen Sie, wie im 6. Block der Vorlesung vorgestellt, eine globale Copy-Propagation als Kombination von globalen und lokalen Phasen realisieren. Dabei bauen Sie auf auf den Ergebnissen der

`flowcopy`-Datenflussanalyse auf. Starten Sie (falls nötig) bei Ausführen des `copyprop`-Kommandos diese Analyse automatisch.

Die Ergebnisse sollen Sie als modifizierten Code direkt in den CFG eintragen, so dass ein Kommando wie `dumpcfg` dann zur Ausgabe des optimierten Programmes genutzt werden kann. Beachten Sie, dass diese Aufgabenstellung *keine* Dead Code Elimination (DCE) von Ihnen verlangt. Die überflüssigen Kopieranweisungen stehen also nach wie vor in Ihrem Code, nur ihre Ergebnisse werden nicht mehr verwendet. Falls Sie optional DCE einbauen wollen, siehe Abschnitt 8.

2.4 SSA-CFG nach AST rückwandeln

Hier soll das im 9. Block der Vorlesung beschriebene Verfahren zur Rückwandlung eingesetzt werden. Da Sie ja hoffentlich (wie dringend empfohlen) schon bei der Konzeption Ihrer CFG-Datenstruktur die Rückrichtung im Auge behalten haben, liegt die Hauptschwierigkeit hier im Auflösen der Phi-Funktionen in geeignet platzierte Kopieranweisungen. Verwenden Sie dazu den in der Vorlesung vorgestellten erweiterten Briggs-Algorithmus, der auch die Variablen in Kontrollbedingungen korrekt behandelt. Bauen Sie für die dafür erforderliche Liveness-Analyse auf den Ergebnissen des `flowlive`-Passes auf, den Sie nötigenfalls automatisch aufrufen.

Die Funktionsweise dieser Phase sollten Sie durch `showast`, `check`, `codegen` sowie dem Laufenlassen Ihrer Testprogramme überprüfen.

3 Hinweise zum Kürzen

Falls Sie sich ausserstande sehen, die Aufgabenstellung im geforderten Umfang zu bearbeiten (aus Zeitmangel, Verständnisprobleme, etc.), sind hier einige Vorschläge für *sinnvolles* Kürzen. Das Implementieren des reduzierten Leistungsumfangs wird aber eine Abwertung nach sich ziehen (z.B. Basisnote nicht mehr 2,0 sondern 3,0 o.ä.).

1. Implementieren Sie statt der globalen eine lokale Copy-Propagation.
2. Wandeln Sie den SSA-CFG nicht mit dem Briggs-Algorithmus zurück, sondern mit dem Verfahren aus dem 4. Block der Vorlesung. Vermeiden Sie Fehlerfälle durch das Aufspalten aller kritischen Kanten.
3. Wenn Sie Varianten 1 *und* 2 wählen, können Sie auch die Datenflußanalyse (`flowcopy` und `flowlive`) komplett weglassen, Ihren Compiler aber wenigstens mit eingeschränktem Funktionsumfang vorführen.

Die vereinbarten Zweiergruppen können als Arbeitserleichterung **ohne Abwertung** die Variante 2 verwenden. Sollte hier das volle Briggs-Verfahren erfolgreich implementiert werden, würde dies zu einer Aufwertung führen.

4 Testen

Um Ihnen das Testen zu erleichtern und das bekannte Prinzip auszunutzen, dass man selbstgeschriebenen Code am besten mit fremdgeschriebenen Testfällen erprobt, dürfen Sie ab jetzt Ihre Triangle-Testprogramme auch gruppenübergreifend **austauschen** (z.B. über das Forum). Sie sollten dabei neben den Programmen selbst auch Angaben über die Ihrer Meinung nach richtigen Ergebnisse machen.

Vergessen Sie nicht, dass Ihr Compiler nach `ssa2ast` nun auch mit `codegen` gültigen Code für Ihren rückgewandelten CFG erzeugen muss. Wenn `codegen` abstürzt oder der optimierte Code (den Sie

jetzt ja ausführen können!) falsche Berechnungen durchführt, haben Sie noch Fehler in Ihrem Compiler.

5 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`-Archiv alle Dateien ihrer Version des Triangle-Compilers an

`oc08@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 9).

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 (in `triangle-examples.tar.gz`) die Ausgaben von `flowcopy`, `floweval` für alle Prozeduren (als einzelne Dateien) nach `read` / `check` / `ast2ssa`.
- dann jeweils die `dumpcfg` Ausgaben nach `copyprop` als einzelne Dateien.
- für alle Beispielprogramme daran anschliessend die Ausgaben von `showast` und `dumpast` nach `ssa2ast` als einzelne Dateien

6 Beurteilung

Die Kolloquien zu dieser Abgabe finden am 03.07.08 in der regulären Vorlesungszeit in Raum E202 statt. Diese Kolloquien sind Bestandteil der Prüfungsleistung, es besteht daher **Anwesenheitspflicht** für alle Gruppenmitlglieder.

Ein Kolloquium dauert jeweils ca. 15 Minuten. Anfangszeiten und Gruppennummern sind unverändert von früheren Aufgaben.

7 Anregungen zur Gruppenarbeit

Bei diesem Endspurt zu einem guten Compiler sind *alle* Gruppenmitglieder bei der Implementierung gefordert! Es bietet sich folgende Aufteilung an:

- Datenflussanalyse (`flowcopy` und `flowlive`)
- Copy Propagation (`copyprop`)
- SSA-Rückwandlung (`ssa2ast`)

Vom Programmierumfang sind die Aufgaben ungefähr vergleichbar, `ssa2ast` hat aber den aufwendigsten Algorithmus. `copyprop` und `ssa2ast` können vollständig unabhängig voneinander entwickelt werden. Beide brauchen aber wohldefinierte Schnittstellen zum Datenflussanalyseteil: `copyprop` interessiert sich für die gesammelten Daten von `flowcopy`, während `ssa2ast` die Liveness-Daten von `flowlive` benötigt. Hier lohnen sich *sorgfältige* Absprachen zu einem geeigneten API! Danach kann auch die Datenflussanalyse unabhängig von den beiden anderen Teilen entwickelt werden.

Zum Testen verwenden Sie eigene Testprogramme oder die von anderen Gruppen zur Verfügung gestellten. Die Testfälle könnten jeweils bestehen aus dem Triangle-Quellcode, den erwarteten SSA-CFGs vor und nach der DVNT- Optimierung und Copy Propagation, sowie dem erwarteten AST nach der Rückwandlung. Damit ist dann eine Kontrolle des ganzen Compile-Flusses möglich.

Hinweis: Die Beispielprogramme sind als Eingaben für die ersten Tests bereits **viel zu kompliziert**! Schreiben Sie *gezielt* kleinere Testprogramme, die Anwendungsstellen für die Optimierung enthalten und auch Sonderfälle (z.B. kritische Kanten) abdecken. Lassen Sie sich für's Erste von den Code-Beispielen des 9. Vorlesungsblockes inspirieren!

8 Ausblick auf Endabgabe

Diese 4. Aufgabe ist die letzte, die neue funktionale Anforderungen an den Compiler stellt. In der Endabgabe geht es primär um Funktionsfähigkeit, Robustheit und Dokumentation. Statt einem nur abstürzenden bzw. falsch rechnenden Code erzeugenden Compiler, der aber theoretisch alles kann, sollten Sie sich darauf konzentrieren, einen vollständig *funktionierenden* Compiler zu implementieren, der aber dann vielleicht einige Optimierungen nicht anbietet (siehe Abschnitt 3).

Nach dieser Abgabe erhalten Sie optional noch einmal Zeit zum abschliessenden Nachbessern. Die genauen Modalitäten werden noch in der Vorlesung abgesprochen, Sie können aber mit gut zwei Wochen zusätzlicher Zeit rechnen. Trotzdem sollten Sie sich nach Kräften bemühen, den oben genannten Abgabetermin **einzuhalten**. Nur so können wir Ihnen in den Kolloquien Rückmeldung geben, was für Schwächen (oder Stärken) Ihr Compiler hat und was Sie für die Endabgabe noch in Angriff nehmen sollten. Nach der Endabgabe wird es keine Nachbesserung mehr geben, wir bewerten den dann vorliegenden Code!

In der Endabgabe ist dann auch die **Kommentierung** relevant: 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.

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 OC08 Web-Site verfügbar).

Sollten Sie trotzdem noch Interesse an weitergehenden Arbeiten haben (z.B. zur Erarbeitung eines weiteren Notenbonus) , sei Ihnen hier die Dead Code Elimination (DCE) empfohlen, die die unnötigen Kopieranweisungen tatsächlich entfernt (das macht die Copy Propagation ja selbst nicht). Die dafür nötigen Verfahren, namens CLEAN und DEAD werden in Cooper & Torczon, Abschnitt 10.3.1 recht gut beschrieben. Ihre Vorstellung ist auch noch in der Vorlesung geplant, dann aber sehr gegen Ende der Veranstaltung.

9 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 OC08 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* ausgetauscht werden. Von letzterer Einschränkung explizit ausgenommen sind Triangle-Beispielprogramme zum Testen des Compilers. Diese **dürfen** ausgetauscht werden (z.B. im Forum der Veranstaltung). Bei Unklarheiten zu diesem Thema (z.B. der Verwendung weiterer Software-Tools oder Bibliotheken) sprechen Sie bitte Ihren Betreuer gezielt an.