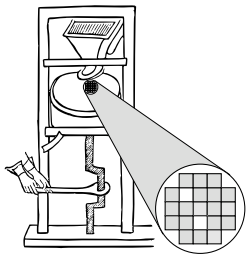
Technische Universität Darmstadt
FG Eingebettete Systeme und
ihre Anwendungen (ESA)

Prof. Dr. Andreas Koch



"Optimierende Compiler"
Aufgabe 3: Erzeugung eines CFG in SSA-Form aus dem DAST
Abgabe bis zum 21.6.2006, 12:00 MET mittags

1 Einleitung

Als Grundlage für weitere Optimierungen soll aus dem AST, der ja die Hauptzwischendarstellung in Triangle ist, ein Kontrollflußgraph in Static Single Assignment-Form (SSA-Form) erzeugt werden. Dabei soll das in der Vorlesung vorgestellte Verfahren von Brandis und Mössenböck verwendet werden. Weiterhin erstellen Sie ein oder mehrere Testprogramme in dem ab jetzt verwendeten Triangle-Subset, mit denen Sie die Funktionsfähigkeit Ihrer Lösung überprüft haben.

2 Problemstellung

2.1 Interaktive Oberfläche

Die von Ihnen in Phase 2 erstellte interaktive Oberfläche des Triangle-Compilers soll um zwei Kommandos erweitert werden.

ast2ssa soll aus dem AST einen SSA-CFG erzeugen

dumpcfg soll den erstellten CFG ausgeben (auf Wunsch in eine Datei)

2.2 Erzeugen des SSA-CFGs

Wie schon in der Vorlesung bekanntgegeben, werden wir uns aus pädagogischen und Zeitgründen ab jetzt nur noch mit einer Untermenge von Triangle befassen.

Speziell sollen jetzt *nicht* mehr zulässig sein:

- Prozedurdefinitionen und -aufrufe ausser `put`, `putint`, `puteol`
- Funktionsdefinitionen und -aufrufe
- Keine inneren `let in` Blöcke
- Keine `let` und `if`-Expressions
- Keine Arrays und Records

- Keine selbstdefinierten Typen

Die bearbeiteten Programme bestehen also aus einem `let in`-Block, der globale Variablen definieren kann, Kontrollflußkonstrukten wie `if`, `while`, sowie Zuweisungen an die deklarierten Variablen. Eingestreut sein dürfen dann auch noch verschiedene `put`-Aufrufe.

Die `put`-Aufrufe sind zu Debug-Zwecken sehr nützlich und verkomplizieren die SSA-Umwandlung nicht wesentlich: Sie haben aus Sicht des Compiler keine Seiteneffekte, und werten lediglich den Wert ihres Argumentes aus. Dieses muß aber, wie beim Verfahren von Brandis und Mössenböck üblich, auf den aktuellen Wert angepasst werden (die richtige Versionsnummer der Variablen verwenden!).

Es wird sich Ihnen die Frage stellen, wie Sie die *Inhalte* der Basisblöcke abspeichern. Eine Möglichkeit ist hier sicherlich, jeden Basisblock mit einer Liste von Anweisungen zu modellieren. Die Anweisungen könnten dann entweder aus einem Verweis in den AST bestehen, oder eine neue Repräsentation haben.

In jedem Fall müssen Sie in der Lage sein, die Phi-Funktionen mit ihrer variablen Anzahl von Elementen abzuspeichern. Der Triangle-AST bietet dafür noch keine Möglichkeit und müsste entsprechend erweitert werden. In einer eigenen Darstellung können Sie natürlich so verfahren, wie es ihnen am besten passt. Sie können in beiden Fällen aber davon ausgehen, dass alle Operanden (=Parameter) einer Phi-Funktion und deren Ergebnis vom gleichen Typ sind. Also alle Boolean, alle Integer, oder alle Character.

Es ist auch sinnvoll beim Entwurf der CFG-Struktur für die übernächste Phase vorzuplanen, in der *Änderungen* vorgenommen werden. Beispielsweise können Anweisungen gelöscht oder verschoben werden (auch über Basisblockgrenzen hinweg), oder auch verändert werden (z.B. Ausdrücke durch vorher berechnete Werte ersetzt werden).

Auch hier müssen Sie überlegen, ob Sie alle Änderungen inkrementell sofort im AST sichtbar machen möchten, oder ob Sie den gesamten CFG "auf einen Satz" in einen AST rücküberführen.

2.3 Ausgabe des SSA-CFG

Zu Debug-Zwecken ist es unerläßlich, eine gut lesbare Ausgabe des CFGs zu bekommen. Das Kommando `dumpcfg`, optional gefolgt von einem Dateinamen, soll eine solche Ansicht in Textform ausgeben bzw. optional in eine Datei schreiben. Dabei verwenden Sie bitte das in Abbildung 1 skizzierte Format.

`cfg` ist der Name des CFGs (hier könnte man auch den Rumpfnamen der Eingabedatei verwenden). Dann folgen mit B1, B2, ...beschriftet die Ausgabe der Basisblock-Knoten. Jeder Basisblockknoten wird beschrieben durch seinen eindeutigen Namen (können Sie frei vergeben), gefolgt nach dem `|`-Zeichen von seinen enthaltenen Anweisungen. Die Anweisungen werden dabei durch Zeilenvorschübe `\n` getrennt. Wichtig: Bei dieser Darstellung sind die Zeichen `\`, `<`, `>`, `{`, `}` und `"` nicht direkt erlaubt. Sie müssen durch voranstellen eines `\`, wie im Beispiel für `a_1 \> b_1` geschehen, geschrieben werden.

Den Knoten folgen dann die gerichteten Kanten zwischen Knoten. Die Kanten von IF-Konstrukten sollen mit T/F für den THEN und den ELSE-Zweig beschriftet werden, die Kanten von Schleifen mit X/L für den Schleifenausgang und Schleifenrücksprung. Kanten zu Join-Knoten sollen mit der Kantennummer (analog zur Phi-Funktion) beschriftet werden.

Weitere Informationen zu diesem eigenartigen Format und seiner praktischen Nutzbarkeit finden Sie unter `www.graphviz.org`, Stichwort *dot*. Abbildung 2 bietet eine kleine Vorschau.

```
digraph cfg {

node [ shape=record ];

/* Knoten des CFGs */

B1 [label="{ B1 | a_1 := b_0 + 1\nb_1 := 2\nif a_1 \> b_1}"];
B2 [label="{ B2 | a_2 := 2*a_1 }"];
B3 [label="{ B3 | a_3 := 2*b_1 }"];
B4 [label="{ B4 | a_4 := phi(a_2, a_3)\nb_2 := 42*a_4 }"];

/* Kanten des CFGs */

B1 -> B2 [label="T"];
B1 -> B3 [label="F"];
B2 -> B4 [label="1"];
B3 -> B4 [label="2"];

}
```

Abbildung 1: Beispielausgabe von `dumpcfg`

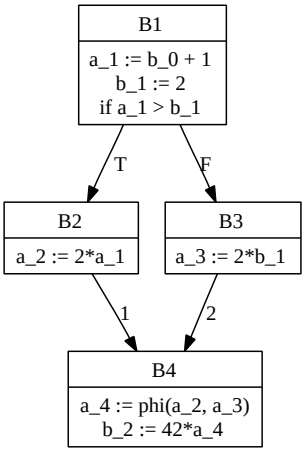


Abbildung 2: Weiterverarbeitung der Beispielausgabe von `dumpcfg`

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 3 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 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

Die Beurteilung dieser Abgabe erfolgt in Kolloquien. Diese finden dann an einem der Dienstagstermine statt. Vorträge sind erst zum Ende des Semesters geplant (nach Abschluß aller Phasen).

5 Anregungen zur Gruppenarbeit

Auch hier ist es so, dass die Ausgabe des CFGs weniger umfangreich als die SSA-Transformation ist. In jedem Fall ist eine sorgfältige Diskussion über die Datenstruktur *vor* Angehen der Arbeiten sinnvoll. Nur auf dieser Basis können Sie tatsächlich parallel arbeiten.

Das Ausarbeiten und Implementieren von Testfällen unter den neuen Randbedingungen (Triangle-Subset) sowie das Durchführen der Tests selber ist auch keine triviale Aufgabe, kann allerdings unabhängig von der CFG-Datenstruktur vorgenommen werden. Hier geht es mehr um ein Verständnis der SSA-Grundlagen und das Konstruieren guter Tests (alle Kontrollkonstrukte, tief verschachtelte Strukturen, etc.)

6 Ausblick auf Aufgabe 4

Die nächste Aufgabe wird sich mit der Rückwandlung des, möglicherweise modifizierten, CFGs aus der SSA-Form in den AST befassen. Falls Sie sich dazu schon einmal vorweg informieren möchten, können Sie einen Blick in das Paper von Briggs et al., ab Seite 20, werfen. Bei der folgenden Aufgabenstellung wird allerdings das "Lost Copy"-Problem (Seite 24-27) keine Rolle spielen, wir werden immer das Aufteilen kritischer Kanten erlauben. Der Abschnitt "If dest $\in$ live_out ..." im Algorithmus in Abbildung 14 des Papers kann daher vereinfacht werden.

7 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.