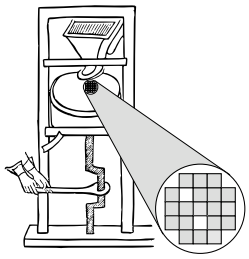
Technische Universität Darmstadt
FG Eingebettete Systeme und
ihre Anwendungen (ESA)

Prof. Dr. Andreas Koch



**"Optimierende Compiler"**
**Aufgabe 3: Erzeugung von CFGs in SSA-Form aus dem DAST**
**Abgabe bis zum 28.06.2007, 18:00 Uhr MET DST**

# 1 Einleitung

Als Grundlage für weitere Optimierungen sollen aus dem AST, der ja die Hauptzwischendarstellung in Triangle ist, Kontrollflußgraphen 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, mit denen Sie die Funktionsfähigkeit Ihrer Lösung überprüfen.

# 2 Problemstellung

## 2.1 Interaktive Oberfläche

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

**ast2ssa** soll aus dem AST mehrere SSA-CFGs erzeugen (je einen pro Prozedur/Funktion und Hauptprogramm)

**dumpcfg** soll einen oder mehrere erstellte CFGs ausgeben (auf Wunsch in eine oder mehrere Dateien)

## 2.2 Erzeugen von SSA-CFGs

Bearbeitet werden soll der vollständige Sprachumfang von Triangle (also auch Prozeduren, Funktionen, Records, Arrays, benutzerdefinierte Typen, etc.). Falls Ihre Lösung hier bekannte Einschränkungen hat, beschreiben Sie diese bitte in der README-Datei.

Bei der Handhabung von Prozeduren greifen Sie auf die Analyseergebnisse der 2. Aufgabe zu und behandeln die Aufrufe ähnlich wie in der Vorlesung skizziert als Pseudo-Zuweisungen. Beachten Sie, dass Sie für jede Prozedur und Funktion einen gesonderten CFG erzeugen müssen. Im folgenden Text wird zur Vereinfachung aber nur von Prozeduren gesprochen.

Bei zusammengesetzten Datentypen (Array, Record) können Sie vereinfachend bei einem Zugriff auch auf ein Unterelement (Array-Element, Mitglied eines Records) einen Zugriff auf die gesamte Variable

```
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`

(komplettes Array, ganzen Record) annehmen. Gleiches gilt auch für benutzerdefinierte Typen. Weitere Details finden sich in der Arbeit von Cytron et al. (Abschnitt 3.1).

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, etc.

Es ist auch sinnvoll beim Entwurf der CFG-Struktur für die nächste 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 alle CFGs "auf einen Satz" in einen AST rücküberführen.

## 2.3 Ausgabe eines SSA-CFG

Zu Debug-Zwecken ist es unerläßlich, eine gut lesbare Ausgabe des CFGs zu bekommen. Das dafür zu implementierende Kommando `dumpcfg` hat als Parameter den Namen der auszugebenden Prozedur. Ein fehlender Name soll zur Ausgabe des Hauptprogramms führen. Optional soll das Kommando *vor* dem Prozedurnamen einen Parameter `-o` *filename* verstehen, mit dem die Ausgabe statt auf der Konsole nun in die Datei *filename* erfolgt. `Optional` können Sie auch noch einen Kommandozeilenparameter `-a` realisieren, der die CFGs `aller` Prozeduren ausgibt (einschliesslich des Hauptprogrammes). Bei gleichzeitiger Verwendung mit `-o` *filename* soll in diesem Fall der CFG jeder Prozedur in eine *eigene* Datei namens `filename-procname.dot` geschrieben werden ( *filename*`-` wird also als Präfix verwendet, um ggf. unterschiedlichen Programme mit gleichen Prozedurnamen auseinanderzuhalten). Als `procname` des Hauptprogrammes soll `main` benutzt werden.

Für die Ausgabe selber verwenden Sie bitte das in Abbildung 1 skizzierte Format.

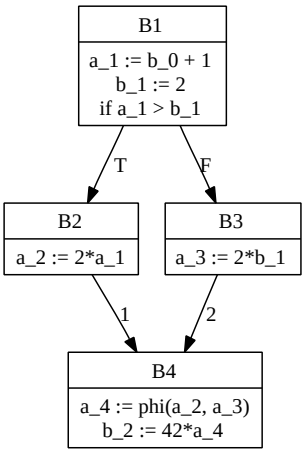



Abbildung 2: Weiterverarbeitung der Beispielausgabe von `dumpcfg`

`cfg` ist der Name des CFGs (hier könnte man auch den Rumpfnamen der Eingabedatei zusammengesetzt mit dem Prozedurnamen 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.

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

`oc07@esa.informatik.tu-darmstadt.de`

mit dem Subject `Abgabe 3 Gruppe` *N*, wobei Ihnen *N* bereits in der Vorlesung mitgeteilt wurde (falls vergessen: siehe nächster Abschnitt). 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 7).

Neben den Java-Quelltexten umfasst 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 Ausgaben von `showast` und `dumpast` nach `read` / `check` / `genrwset`
- für alle Beispielprogramme die Ausgaben von `ast2ssa` und `dumpcfg` (für alle Prozeduren) nach `read` / `check` / `genrwset` als einzelne Dateien.

# 4 Beurteilung

Die Kolloquien zu dieser Abgabe finden am 3.7.07 in der regulären Vorlesungszeit in Raum E103 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 vom 2. Aufgabenblatt.

# 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 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, Prozeduren, tief verschachtelte Strukturen, Arrays, Records, etc.)

Falls in Ihrer Gruppe eine Situation entstehen sollte, in der einzelne Mitglieder deutlich zuwenig (oder zuviel!) 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).

# 6 Ausblick auf Aufgabe 4

Die nächste Aufgabe wird sich unter anderem 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 OC07 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. Bei Unklarheiten zu diesem Thema (z.B. der Verwendung weiterer Software-Tools oder Bibliotheken) sprechen Sie bitte Ihren Betreuer gezielt an.