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

Prof. Dr. Andreas Koch

Embedded Systems & Applications

"Optimierende Compiler"
Aufgabe 2: Erzeugung von CFGs in SSA-Form aus dem DAST
Abgabe bis zum 16.05.2011, 23:59 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 Aufgabe 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 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 durch Aufgabe 1 reduzierte Sprachumfang von Triangle, dieser allerdings *vollständig*. Für die Datentypen gelten folgende Regeln:

1. Variablen primitiver Datentypen (`Integer`, `Boolean`, `Character`) sollen wie in der Vorlesung beschrieben in SSA-Form übersetzt werden.
2. Zusammengesetzte Typen (durch `record` definiert) und Arrays werden *nicht* in die SSA-Form übersetzt sondern unverändert durchgereicht. Dabei darf die Semantik des Programmes aber nicht verändert werden!
3. Selbstdefinierte Typen (via `type`) werden so behandelt, wie die ihnen zugrundeliegenden Typen. Wenn also ein neuer Typ nur einen primitiven Typ umbenennt, werden seine Variablen wie im Fall 1 behandelt. Sind es Records oder Arrays, kommt Fall 2 zum tragen.
4. Funktionen werden *nicht* gesondert in SSA-CFGs übersetzt (wir haben die `if`-Expression ja entfernt), sondern aus den CFGs der Prozeduren lediglich seiteneffektfrei *aufgerufen*. Der Aufruf selber muss allerdings in den CFGs gespeichert werden, da ja in der letzten Aufgabe die CFGs wie-

der in ASTs zurückgewandelt werden müssen. Und dabei dürfen die Funktionsaufrufe natürlich nicht fehlen.

Falls Ihre Lösung diese Vorgaben nicht vollständig umsetzen kann, beschreiben Sie die Einschränkungen bitte in der README-Datei.

Bei der Handhabung von Prozeduren können Sie die Aufrufe ähnlich wie in der Vorlesung skizziert als Pseudo-Zuweisungen behandeln. Beachten Sie, dass Sie für jede Prozedur einen *gesonderten* CFG erzeugen müssen.

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 übernächste Aufgabe 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 werden.

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. Das Kommando soll *vor* dem Prozedurnamen auch einen optionalen Parameter `-o` *filename* verstehen, mit dem die Ausgabe statt auf der Konsole nun in die Datei *filename* erfolgt. Ebenso soll auch noch ein Kommandozeilenparameter `-a` realisiert werden, 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: `cfg_main` 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 (eXit) und Schleifenrücksprung (Loop). 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_main {

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`

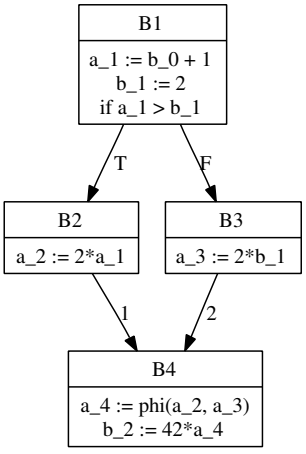

Abbildung 2: Weiterverarbeitung der Beispielausgabe von `dumpcfg`

3 Abgabe

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`- oder `.zip`-Archiv alle Dateien ihrer Version des Triangle-Compilers an

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

mit dem Subject `Abgabe 2 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).

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 Abgaben.
- für alle Beispielprogramme die Ausgaben von `showast` und `dumpast` nach `read` / `check`
- für alle Beispielprogramme die Ausgaben von `ast2ssa` und `dumpcfg -a -o ...` für alle Beispielprogramme nach `read` / `check` als einzelne Dateien.

4 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* dem Angehen der Arbeiten sinnvoll. Nur auf dieser Basis können Sie tatsächlich parallel implementieren.

Die Beispielprogramme werden in den meisten Fällen als Tests zunächst ungeeignet sein, da sie zu kompliziert sind. Entwickeln Sie daher als erstes eine *umfassende* Sammlung von kleineren Testprogrammen, die verschiedene Aspekte des Problems gezielt abbilden. Dabei kann es z.B. gehen um alle Kontrollkonstrukte, Prozeduren mit und ohne `var`, tief verschachtelte Kontrollstrukturen, Arrays, Records, selbstdefinierte Typen, etc. Das Ausarbeiten und Implementieren dieser Testfälle sowie das Durchführen der Tests selber ist keine triviale Aufgabe, kann allerdings unabhängig von der CFG-Datenstruktur vorgenommen werden, da es hier mehr um ein Verständnis der SSA-Grundlagen und das Konstruieren guter Tests geht.

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).

5 Ausblick auf Aufgabe 3

Die nächste Aufgabe wird sich unter anderem mit der Rückwandlung des, möglicherweise modifizierten, CFGs aus der SSA-Form wieder 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.

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 Optimierende Compiler-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.