Prof. Dr. Arnd Poetzsch-Heffter
Dr.-Ing. Jan Schäfer

Technische Universität Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Übungsblatt 11: Übersetzer und sprachverarbeitende Werkzeuge (SS 2011)

Hand Out: 14. Juli 2011
Hand In: 20. Juli 2011

Aufgabe 1 Beispielprogramm erstellen

Die unten abgebildete Katja-Spezifikation (auf der Webseite zum Download als `TAC.katja`) beschreibt die Zwischensprache, die wir auf diesem Übungsblatt verwenden, um Analysen und Optimierungen zu realisieren.

```
root Program Pos

Program = Statements

Statements * Statement

Statement = Assignment
          | While (Value cond, Statements body)
          | If    (Value cond, Statements thenCase, Statements elseCase)

Assignment = Add   (Var dest, Value left, Value right)
           | Minus (Var dest, Value left, Value right)
           | Mult  (Var dest, Value left, Value right)
           | Div   (Var dest, Value left, Value right)

Value = Var   (String name)
      | Const (Integer value)
```

Schreiben Sie ein Java-Programm, welches drei Beispielprogramme, als Katja-Positionsstruktur aufbaut. Eine Schleife und ein If-Statement sollte jeweils mindestens enthalten sein. Kommentieren Sie jeweils kurz, was diese Programme tun.

Aufgabe 2 Liveness-Analyse

Implementieren Sie eine Liveness-Analyse für die Sprache aus Aufgabe 1. Testen Sie die Analyse anhand Ihrer Beispielprogramme, lassen Sie sich für jede Iteration der Fixpunktbildung, den aktuellen Zustand ausgeben.

Hinweise

- Bauen Sie vor der Analyse einen Kontrollflussgraphen auf. Die Basis-Blöcke des Kontrollflussgraphen bestehen nur noch aus Listen von `AssignmentPos`-Objekten und enthalten keine While- und If-Statements.
- (Optional) Versuchen Sie den Kontrollflussgraphen nur mittels Attribute auf der Positionsstruktur zu repräsentieren, ohne den Graphen explizit durch Objekte zu repräsentieren.

Aufgabe 3 Dead-Code-Elimination

Implementieren Sie Dead-Code-Elimination auf Basis der Liveness-Analyse und geben Sie das optimierte Programm jeweils aus.

Aufgabe 4 Integration in den Compiler (Zusatzaufgabe)

Erweitern Sie die Zwischensprache von oben so, dass Sie als Zwischensprache für Ihren MiniJava-Compiler geeignet ist und integrieren Sie die zuvor implementierte Analyse und Optimierung in Ihren MiniJava-Compiler. Dazu benötigen Sie eine Übersetzung aus dem MiniJava-AST in die Zwischensprache und von dort zu abstraktem MIPS-Assembler.