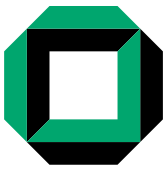

Universität Karlsruhe (TH)

Lehrstuhl für Programmierparadigmen
Sprachtechnologie und Compiler II SS 2009 http://pp.info.uni-karlsruhe.de/
Dozent: Prof. Dr.-Ing. G. Snelting snelting@ipd.info.uni-karlsruhe.de
Übungsleiter: Matthias Braun braun@ipd.info.uni-karlsruhe.de

Übungsblatt 5 | Ausgabe: 17.6.2009 | Besprechung: 23.6.2009

Dies ist ein zusätzliches Übungsblatt zur Demonstration von Datenflußanalaysen in der Praxis.

Aufgabe 1: *Datenflussanalyse - Praxis*

1.1 *Framework*

Unter `http://pp.info.uni-karlsruhe.de/lehre/SS2009/compiler2/DataFlowExercise.zip` finden Sie einen vereinfachten Parser mit zusätzlichen Funktionen zum Aufbau des AST und eines Kontrollfluß-graph.

Unter `http://www.info.uni-karlsruhe.de/software/ycomp/` finden Sie das Visualisierungstool yComp mit dem Sie sich die erzeugten Kontrollflußgraphen mit annotierten Ergebnissen der Analysen betrachten können.

Übersetzen Sie das Projekt und führen Sie es aus! Es sollten 2 `.vcg`-Graphen erzeugt werden die Sie mit yComp betrachten können.

1.2 *Architektur*

Betrachten Sie sich die Architektur der Rahmenwerks. Die Implemntierung der Fixpunktiteration befindet sich in der Klasse `DataFlow`. Konkrete Datenflußanalysen lassen sich Separat durch Implementierung von `DataFlowInterface` erzeugen. Damit sind Fixpunktieration und Datenflußanalyse unabhängig voneinander:

```
package DataFlow;
import CFG.CFG;

/**
 * Abstraction for different dataflow analyses.
 * (This is a generic interface which can be parameterized by the
 *  datatype used to represent dataflow values).
 * @author Matthias Braun
 */
public interface DataFlowInterface<Data, CFGNode> {
    /**
     * Prepare dataflow analysis to work on a CFG.
     */
    void initialize(CFG<CFGNode> cfg);

    /**
     * return true if analysis is a backwards analysis.
     * false if it is a forward analysis
     */
    boolean isBackward();

    /**
     * join/union operation for 2 data values
     * The Data returned must not be null
     * (null is reserved by dataflow algorithm as marker for not calculated values)
     */
    Data union(Data d1, Data d2);
```

```
    /**
     * apply transfer function for a node in the CFG
     * The Data returned must not be null
     * (null is reserved by dataflow algorithm as marker for not calculated values)
     * Warning: It is not allowed to modify the object from the data Parameter!
     *          You have to copy/clone it first.
     */
    Data transfer(CFGNode node, Data data);
}
```

Machen Sie sich mit diesen Klassen vertraut.

- Wo werden Java Generics eingesetzt und wozu?
- Warum ist die Trennung von Fixpunktiteration und Datenflußanalyse sinnvoll?

1.3 *Fixpunktiteration*

- Die Klasse `DataFlow` enthält eine unfertige Implementierung des Worklistalgorithmus in der Method `DataFlow.perform`. Vervollständigen Sie diese.
- Überprüfen Sie die Korrektheit ihrer implementierung, indem Sie die Ergebnisse der Lebendigkeitsanalyse für das Testprogramm betrachten.

1.4 *Reaching Definitions Analysis*

Das Projekt enthält ebenfalls eine unfertige Reaching Definitions Analyse. Die zwar die Grundlegenden Datenstrukturen enthält aber die Transferfunktion und die Unionoperation sind noch nicht implementiert.

- Vervollständigen Sie die Implementierung der Reaching Definitions Analysis.
- Überprüfen Sie die Korrekheit ihrer Analyse.

1.5 *Kür*

- Implementieren Sie eine weitere Datenflußanalyse ihrer Wahl!