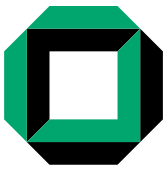


Universität Karlsruhe (TH)

Lehrstuhl für Programmierparadigmen

Compilerpraktikum SS 2009 — http://pp.info.uni-karlsruhe.de/

Dozent: Prof. Dr.-Ing. G. Snelting — snelting@ipd.info.uni-karlsruhe.de

Betreuer: Matthias Braun — braun@ipd.info.uni-karlsruhe.de

Betreuer: Jürgen Graf — graf@ipd.info.uni-karlsruhe.de

Übungsblatt 6 | Ausgabe: 3.06.2009 | Besprechung: 24.06.2009

Die nächsten Wochen geht es darum dem Compiler einige einfache Optimierungen beizubringen.

Tips für die Implementierung:

- Vermeiden Sie es Knoten zu Verändern der Knoten (node.SetXXX), erzeugen Sie stattdessen stets neue Knoten. Dies erleichtert die Arbeit des Common Subexpression Elimination.
- Überlegen Sie sich wie die Optimierungen zusammenspielen und in welcher Reihenfolge Sie geschehen sollten.
- Durchläuft man DAGs (gerichtete azyklische Graphen) mit einer Tiefensuche in Postorder, dann ist sichergestellt, dass man stets alle Vorgänger eines Knotens besucht hat bevor man den Knoten selbst besucht. Warum ist diese Reihenfolge für Optimierungen sinnvoll? An welchen Stellen sind Firm-Graphen keine DAGs? Kann das zu Problemen führen?

Aufgabe 1: *Konstantenfaltung*

Arithmetische Operationen in Firm bei denen alle Vorgänger Konstanten sind können durch eine neue Konstante ersetzt werden.

- Implementieren Sie dies als Optimierung für folgende Knoten: `Add`, `Sub`, `Mul`, `Div`!
- (Freiwillig) Gibt es noch weitere Knoten bei denen Konstantenfaltung Sinn macht? Implementieren Sie diese Konstantenfaltungen!

Aufgabe 2: *Normalisierung*

Normalisierungen sind Optimierungen die für sich genommen keinen Gewinn bringen. Eine typische Optimierung ist zum Beispiel Konstanten stets auf die rechte Seite eines Ausdrucks zu bringen. Beispiel: $23 + x \rightarrow x + 23$. Normalisierungen erleichtern aber häufig die Arbeit anderer Optimierungen, da nur noch nach normalisierten Mustern gesucht werden muss, statt allen möglichen Mustern.

Ausserdem steigert Normalisierung die Effektivität der Common Subexpression Elimination.

- Implementieren Sie die einfache Normalisierung die bei kommutativen Operationen (`Add`, `Mul`) die Konstante auf die rechte Seite der Operation bringt!
- (freiwillig) Denken Sie an die mathemtischen Gesetze der Kommutativität und Assoziativität, fallen ihnen weitere mögliche Normalisierungen ein? Implementieren Sie diese!

Aufgabe 3: *Arithmetische Vereinfachungen*

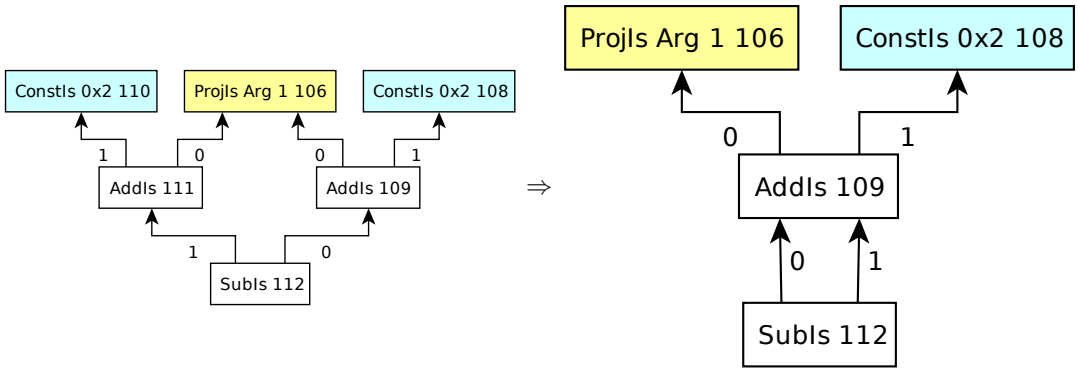
Viele Ausdrücke können mit arithmetischen Regeln vereinfacht werden. Implementieren Sie Optimierungen für die folgenden Fälle:

- Neutrales Element: $x + 0 \rightarrow x$; $x - 0 \rightarrow 0$; $x * 1 \rightarrow x$
- Null: $x * 0 \rightarrow 0$
- $x - x \rightarrow 0$
- (freiwillig) Fallen ihnen weitere Vereinfachungen ein? Implementieren Sie diese!

Aufgabe 4: *Common Subexpression Elimination (geplant für 2. Woche)*

Common Subexpression Elimination vermeidet ein mehrfaches Ausrechnen von Ausdrücken. In firm bedeutet dies, dass man bei 2 identischen Subgraphen nur noch einen der beiden Graphen benutzt. CSE ermöglicht oft weitere arithmetische Vereinfachungen.

Beispiel:



(In diesem Beispiel macht die CSE die Ausführung der arithmetischen Vereinfachung $x - x \rightarrow 0$ möglich)

- Wann berechnen 2 Knoten in Firm den gleichen Wert?
- Überlegen Sie sich passende Datenstrukturen zur Durchführung der CSE.
- Welche Reihenfolge sollte man geschickterweise Wählen?
- (freiwillig, schwer) Gibt es Fälle in denen 2 Knoten das gleiche Ausrechnen, die von ihrer CSE Implementierung aber nicht gefunden werden?