Allgemeines

Fächer:	Grundlagen des Übersetzerbaus, Optimierungen in Übersetzern
Prüfer:	Prof. Dr. Michael Philippsen
Beisitzer:	Tobias Werth
Note:	1,0

Es herrschte eine sehr angenehme Atmosphäre. Anstatt eine feste Liste von Fragen abzuarbeiten, wurde einem ein Stück Code vorgelegt mit der Aufforderung: „Nun erzählen Sie mal, was Ihnen alles dazu einfällt.“

```
i = 0;
while (i < 10) {
    tmp  = a + b;
    A[i] = tmp * A[i + 1];
    i++;
}
```

Innerhalb eines gewissen Rahmens bestand also die Möglichkeit, das Gespräch in eine Richtung zu lenken, die einem behagte. Die Prüfer ließen einen reden und stellten Zwischenfragen, die in erster Linie auf das Verstehen des Stoffes abzielten und weniger auf das Wiedergeben von auswendig gelerntem Detailwissen[1].

Die Fragen waren zum Teil recht anspruchsvoll, insgesamt aber sehr fair und wurden überaus gut bewertet.

Fragen

Dies ist der etwaige Verlauf des Prüfungsgesprächs. Da die meisten Fragen aus dem Gesprächsverlauf heraus entstanden, sind hier auch alle Antworten kurz skizziert.

Was kann man mit obiger Schleife anstellen, um den erzeugten Code zu verbessern?

Die Berechnung von `tmp` ist invariant und kann herausgezogen werden. Die Schleife muss in `do-while`-Form mit umgebendem `if` transformiert werden, damit der Schleifenausgang dominiert wird (→ Kriterium fürs Herausziehen).

Wann & wie erfolgt die Transformation in `do-while`-Form?

AST-Transformation in der Abbildungsphase.

[1] Was einen aber nicht davon abhalten sollte, so detailliert wie möglich zu antworten. Je mehr (sinnvolles Zeug) man redet, desto besser!

Zeichnen Sie den Kontrolflussgraphen der transformierten Schleife. Zeigen Sie, wohin der invariante Code verschoben wird.

Für den invarianten Code muss ein neuer Grundblock eingefügt werden (zwischen `if` und `do`).

Analysieren Sie den KFG auf Dominanz. Erklären Sie den Algorithmus, der dafür verwendet wird (warum bildet man für jeden Grundblock die Schnittmenge der Dominatoren aller seiner Vorgänger?).

Jeder Pfad von der Wurzel zu einem bestimmten Knoten n muss durch einen Vorgänger von n führen. Ein Knoten, der alle Vorgänger dominiert, ist folglich in jedem Pfaden zu n enthalten. Im umgekehrten Fall existiert ein Pfad zu n „um diesen Knoten herum“, also dominiert er n nicht.

Wie findet man natürliche Schleifen und wie unnatürliche? Worum handelt es sich beim vorliegenden Code? Wodurch können unnatürliche Schleifen entstehen?

- Natürliche Schleife: Rückwärtskante, Worklist-Algorithmus
- Unnatürliche Schleife: Zyklus ohne Rückwärtskante, Test durch Reduktion des KFG

Die gegebene Schleife ist natürlich. Unnatürliche Schleifen können nur bei Sprachen entstehen, die `goto` oder ähnlich üble Konstrukte (z.B. Duff's Device) erlauben.

Welche weiteren Optimierungen kann man mit der gegebenen Schleife anstellen?

Laut nachgedacht:

- Parallelisierung: nicht möglich wegen Anti-Abhängigkeit mit Distanz 1
- Schleife ausrollen: möglich, geht immer

Unter welchen Bedingungen wäre Parallelisierung möglich?

- Keine schleifengetragene Abhängigkeit: entsprechende Stelle im Distanzvektor ist gleich 0
- Äußere Schleife trägt Abhängigkeit: eine Stelle weiter links im Distanzvektor ist positiv

Ist das Ausrollen der Schleife sinnvoll?

Ja, am besten komplett ausrollen. Dadurch spart man den Overhead durch das Inkrementieren von `i` und den bedingten Sprung.

Warum sind bedingte Sprünge problematisch? Was wird dagegen unternommen?

Da das Sprungziel nicht statisch bekannt ist, läuft das Fließband leer. Abhilfemöglichkeit: Sprungzielvorhersage der CPU

- Heuristiken: bedingter Rücksprung wahrscheinlich, da Schleife
- CPU merkt sich für jeden bedingten Sprung die letzten zwei Ergebnisse der Auswertung im Instruktions-Cache

Ein Übersetzer, die die Zielarchitektur genau kennt, kann den Maschinencode so ausgeben, dass die Heuristiken der CPU mit höherer Wahrscheinlichkeit zutreffen.

Wie findet man die erreichenden Definitionen einer Variablenverwendung?

- Fixpunktiteration mit Bitvektoren (kompliziert)
- SSA-Form: einfach ablesen, da nur eine erreichende Definition existiert

Für die Transformation des gegebenen Codes in SSA-Form reicht leider die Zeit nicht mehr. ;-) Wie geschieht die Rücktransformation aus SSA-Form bei einer Φ-Funktion?

- Simpel: Zuweisung am Ende jedes Vorgängerblocks
- Besser: mit Färbealgorithmus; Verschmelzungskanten