Protokoll CB1-Prüfung

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
1.0	13.04.2012		CL

Contents

1 Meta-Info

Studiengang	Master Inf
Dauer	29 Minuten
Prüfer	Michael Philippsen
Beisitzer	Thorsten Blaß
Thema	Grundlagen des Übersetzerbaus
Datum	13.04.2012
Ergebnis	1.0

2 Allgemeines

Stift und Papier liegen bereit, Codebeispiele sind vorbereitet und ausgedruckt. Die Fragen stellen sowohl Prof. Philippsen als auch der Beisitzer. Die Prüfung kam mir im Vergleich zu anderen relativ lang vor. Es wurde viel Wert auf Details und Verständnis gelegt, die Fragen waren offen und manchmal (absichtlich) irreführend gestellt. Ich bin gerade am Anfang der Prüfung sehr viel geschwommen und zwischen Themen hin- und hergesprungen, oft war mir nicht ganz klar, worauf er hinauswollte und habe mich das ein- oder andere Mal in einen (scheinbaren) Widerspruch verstrickt (der dann auch garantiert bemerkt und einem Vorgehalten wird).

3 Inhalt

Ich bekam den AST von

```
k := ((a + 3) * (c + d)) * (e + f)
```

vorgelegt und sollte erklären, wie ich daraus Code erzeugen kann. Ich wollte nach Möglichkeit Graham&Glanville oder Codeerzeugung mit DP machen, also hab ich die beiden genannt, Prof. Philippsen wollte aber Codeerzeugung durch Baumtransformation haben. Beim Lernen hatte ich das als "wir machen halt G&G mit Bäumen auf der rechten Seite der Produktionen" abgetan, was so wohl nicht ganz richtig war. Der Unterschied (und der war gesucht und gefragt) liegt in der Behandlung der Konflikte: Während G&G Shift-Reduce-Konflikte hat sucht man bei der Codeerzeugung durch Baumtransformation immer die Transformation, die am nähesten an der Wurzel liegt und wendet diese an. Die Entscheidung, welche Regel angewendet wird fällt bei der Baumtransformation also Top-Down.

G&G durfte ich dann später an einem Teil des ASTs und der Grammatik machen. Prof. Philippsen wollte wissen, woher man denn bei dem LR-Parser in G&G weiß, dass man nochmal Shiften kann (gesucht war natürlich der Lookahead). Außer Konkurrenz wurde ich noch gefragt, ob man denn Backtracking machen müsste, wenn man an einem Shift-Reduce-Konflikt ein Shift macht und danach die Regel doch nicht zutrifft; ich wusste es nicht und ließ mir erklären, dass der Fall einfach nicht auftritt, weil man das Lookahead für die Maschinenarchitektur so wählt, dass man keine falschen Shifts macht.

Prof. Philippsen wollte wissen, welches Kriterium denn bei der Codeerzeugung durch Baumtransformation optimiert wird. Habe Codegröße genannt und erzählt, dass man andere Kriterien durch eine Kostenfunktion repräsentieren kann, dann die Auswahl der richtigen Operationen aber nicht mehr trivial ist.

Dann wurde ich zu Registervergabe befragt. Genannt habe ich als Verfahren Graphfärben (getreg ist mir später zwar eingefallen, habs dann aber nicht mehr gebraucht) und der Registernutzung in Ausdrucksbäumen mit DP (das durfte ich dann auch kurz umreißen, wurde dann aber unterbrochen). Auch erwartet wurde anscheinend, dass ich die Abhängigkeit zwischen Codeerzeugung und Registervergabe nenne und dass die beiden Phasen nicht immer strikt trennbar sind, wenn man die optimale Lösung haben will. Dann hab ich erzählt, dass man da ja vielleicht einen iterativen Fixpunkt-Algorithmus nehmen könnte und immer wieder zwischen Codeerzeugung und Registervergabe hin- und herspringen könnte, solange bis der Code nicht besser wird. Ob das richtig war weiß ich nicht, das wurde nur mit "interessant" kommentiert. :)

In den letzten fünf Minuten hab ich dann noch ein Beispiel mit inneren Klassen bekommen:

```
class Outer {
        public int a;
        private int b;
```

```
        void foo() {
                int l = 42;

                class Inner {
                        private int q = l + a;
                        private int f = b;
                }
        }
}
```

Hier hab ich erwähnt, dass die Hierarchie plattgeklopft wird und bei der semantischen Analyse festgestellt wird, dass l nicht in Inner benutzt werden kann, weil es nicht final ist. Dann durfte ich beliebige Transformationen aussuchen und erklären, die bei der Abbildung von inneren Klassen passieren:

```
class Outer {
        public int a;
        private int b;

        static int access$001(Outer o) {
                return o.b;
        }

        void foo() {
                // ...
        }
}

class Outer$Inner {
        Outer this$0;
        private final int l;

        Outer$Inner(Outer o, int l) {
                this$0 = o;
                this.l = l;
        }
}
```

Ich hatte zunächst fälschlicherweise gesagt der Wert von Outer.foo().l würde zur Compile-Zeit kopiert (was nur möglich ist, wenn bekannt ist, dass der Wert kein Funktionsaufruf o.Ä. ist, d.h. keine Seiteneffekte hat), woraufhin Prof. Philippsen die 42 durch einen Methodenaufruf bar() ersetzt hat. Dann kam mir, dass man dann wohl doch einen Parameter am Konstruktor dafür braucht.

4 Vorbereitung

Ich hatte mich im Vorfeld der Prüfung etwa eine Woche lang vorbereitet und bin zusammen mit einem Kommilitonen den Foliensatz komplett und detailliert durchgegangen. Wir haben alle Beispiele diskutiert und nachvollzogen und alle unklaren Punkte ausdiskutiert, öfter auch mal den `javac` angeworfen und uns dann den Bytecode zeigen lassen (`javap -c Klassenname`, hilfreich v.a. bei der Transformation von inneren Klassen und Generics). Zusätzlich haben wir die Verfahren, die in der Übung besprochen wurden an mehreren Beispielen durchexerziert.