Praktische Informatik I WS 2006/07

Übungsblatt 9

Abgabe: 15.01.2007

Aufgabe 1: Beweis durch Semantik (50%)

> Wenn CD aber AB nicht misst, und man nimmt bei AB, CD abwechselnd immer das kleinere vom grösseren weg, dann muss (schliesslich) eine Zahl übrig bleiben, die die vorangehende misst .

[Aus Euklid, Die Elemente, Herausgegeben von Clemens Thaer, Wissenschaftliche Buchgesellschaft Darmstadt, VII Buch, §2]

Dies ist wohl eine der ältesten Algorithmenbeschreibungen der Welt (ca. 300 v.u.Z.) mit der der *grösste gemeinsame Teiler* (ggT) zweier positiver ganzer Zahlen ermittelt wird. Dabei gelten die folgenden Axiome:

1. $ggT(x, x) = x$.
2. $ggT(x, y) = ggT(y, x)$.
3. $ggT(x, y) = ggT(x, y - x)$ für $x < y$.

Die folgende JAVA Methode soll diesen Algorithmus implementieren.

```
public static int ggt(int wert1, int wert2)
{
  while (wert1 != wert2)
  {
    if (wert1 > wert2)
    wert1 = wert1 -wert2;
    else
    wert2 = wert2 - wert1;
  }
  return wert1;
}
```

Überprüft mit Hilfe der Regeln der imperativen Semantik, dass diese Methode ihren Zweck erfüllt und exemplarisch den $ggT(44, 12)$ berechnet.

Aufgabe 2: Interpretation von Assemblerbefehlen (50%)

Gegeben ist folgende Liste vom Assemblerbefehlen.

LOAD x	Lade den Inhalt von Adresse x in den Akkumulator
LOADA	Lade den Inhalt der Adresse, deren Wert im Akkumulator steht, in den Akkumulator
STORE x	Speichere den Inhalt des Akkumulators in der Speicherzelle mit der Adresse x
ADD x	Addiere den Wert an Adresse x zum Inhalt des Akkumulators
SUB x	Subtrahiere den Wert von Adresse x vom Inhalt des Akkumulators
MULT x	Multipliziert den Wert von Adresse x mit dem Inhalt des Akkumulators
JMPNEG x	Springe zur Marke x, wenn der Inhalt des Akkumulators < 0
JMPEQ x	Springe zur Marke x, wenn der Inhalt des Akkumulators $= 0$
JLE x	Springe zur Marke x, wenn der Inhalt des Akkumulators $<= 0$
JMP x	Springe zur Marke x
NOT	logische Negation - bitweises Komplement des Wertes im Akkumulator
NEG	unäres Minus angewendet auf den Wert im Akkumulator.
HALT	Beendet das Programm

Mit jedem Assemblerbefehl ist eine Semantikregel assoziiert, die angibt, wie der aktuelle Zustand verändert wird, so ist beispielsweise

$$[\![LOAD\ x]\!](\sigma) = \sigma \oplus \{Akk \mapsto \sigma(Mem[x]), pc \mapsto \sigma(pc) + 1\}$$

(*Akk* sei dabei die Variable, die den Akkumulatorinhalt bezeichnet, Mem das Speicherarray, *pc* der Programmzähler).

a) Gebt für jeden Befehl die entsprechende semantische Transformation an.

b) Der obige Befehlssatz wird um den Befehl

PRINT x	Gebe den Inhalt von Adresse x aus

ergänzt, für den keine Semantikregel definiert werden soll.

Schreibt ein JAVA Programm, mit dem ein Assemblerprogramm, das diesen Befehlssatz verwendet, ausgeführt werden kann. Dabei soll der Speicher als `int []` Array modelliert werden. Um die Zustandsübergänge zu veranschaulichen, soll zu jedem Befehl der aktuelle Zustand ausgegeben werden. Als Programmrahmen und Testfall dient dabei das Programm `AsmInterpreter.java` auf der PI1 - Homepage.

c) Gebt eine kurze Beschreibung des Verhaltens des Pseudo-Assemblerprogramms.