Aufgaben zum Tut am 09.01.2006

Thomas Pajor

11. Januar 2006

Aufgabe 1)

Das Problem MULTIPROCESSOR SCHEDULING (MPS) ist wie folgt definiert:

Gegeben: Eine Menge $M := \{1, \ldots, m\}$ von m identischen Maschinen oder Prozessoren und eine Menge J von n Jobs mit positiven Bearbeitungsdauern, also einer Abbildung $p : J \to \mathbb{N}$ die jedem Job eine Zeit zuordnet. Außerdem eine Deadline $D \in \mathbb{N}$.

Frage: Existiert ein Schedule s mit Abarbeitungslänge höchstens D? Also existiert eine Zuordnung der Jobs auf die Maschinen $s : J \to M$, so dass

$$\max_{1 \leq j \leq m} \sum_{i \in s^{-1}(j)} p(i) \leq D$$

Zeigen Sie: MPS ist $\mathcal{NP}$-vollständig.

Lösung.

Wir zeigen MPS $\in \mathcal{NPV}$ in zwei Schritten.

1. MPS $\in \mathcal{NP}$:

   Zu einem geratenen Lösungsvorschlag s können wir mit Aufwand $\mathcal{O}(|M| \cdot |J|)$ überprüfen ob er eine gültige Lösung ist in dem wir für jede Maschine die Laufzeiten der zugeordneten Jobs aufaddieren und prüfen ob die Summe höchstens D ist.

   $\Rightarrow$ MPS ist in $\mathcal{NP}$.

2. MPS ist $\mathcal{NP}$-hart:

   Wir reduzieren das Problem PARTITION auf MPS, also PARTITION $\leq_p$ MPS.

   Gegeben sei eine Instanz $I = (A, w)$ des Problems PARTITION. Wir konstruieren eine Instanz $I' = (M, J, p, D)$ von MPS wie folgt:

(1) $|M| := 2$

(2) $|J| := |A|$

(3) $p(i) := w(i)$

(4) $D := \frac{1}{2} \cdot \sum_{i \in A} w(i)$

Der Aufwand für (2), (3) und (4) ist jeweils in $\mathcal{O}(|A|)$. (1) hat einen Aufwand von $\mathcal{O}(1)$. Der Gesamtaufwand der Transformation ist also polynomiell.

Es bleibt noch zu zeigen:

$$I \text{ ist Lösung von PARTITION} \quad \Leftrightarrow \quad I' \text{ ist Lösung von MPS}$$

Sei I eine Lösung von PARTITION und A_1, A_2 die Teilmengen von A mit

$$\sum_{i \in A_1} w(i) = \sum_{i \in A_2} w(i)$$

dann gilt wegen der Konstruktion gerade

$$\sum_{i \in A_1} w(i) = \sum_{i \in A_2} w(i) = D$$

Also ist auch I' mit folgendem Schedule s

$$s(i) := \begin{cases} 1 & \text{falls} \quad i \in A_1 \\ 2 & \text{falls} \quad i \in A_2 \end{cases} \qquad \forall i \in J$$

eine Lösung von MPS.

Sei nun I' eine Lösung von MPS und seien J_1 und J_2 die Mengen der Jobs die durch s auf die erste bzw. zweite Maschine abgebildet wurden. Also gilt:

$$\sum_{i \in J_1} p(i) \leq D \tag{1}$$

$$\sum_{i \in J_2} p(i) \leq D \tag{2}$$

Zusammengefasst gilt nach Konstruktion von D:

$$\sum_{i \in (J_1 \cup J_2)} p(i) \leq 2D = \sum_{i \in A} w(i)$$

Wegen $p(i) = w(i)$ gilt sogar

$$\sum_{i \in (J_1 \cup J_2)} p(i) = 2D$$

Das heißt es muss gelten

$$\sum_{i \in J_1} p(i) = \sum_{i \in J_2} p(i) = D$$

da wir sonst einen Widerspruch zu den Gleichungen (1) und (2) kriegen.

Damit ist auch I eine Lösung von PARTITION.

$\Rightarrow$ MPS ist $\mathcal{NP}$-hart.

Gäbe es nun einen effizienten Algorithmus $\mathcal{A}$ der MPS lösen kann, so könnten wir $\mathcal{A}$ mit Hilfe der obigen Transformation dazu benutzen um für jede Instanz I von PARTITION zu überprüfen ob sie eine Lösung hat.

$\Rightarrow$ MPS ist in $\mathcal{NPV}$. □

Aufgabe 2)

Geben Sie ein WHILE–Programm an, das die Addition zweier natürlicher Zahlen x_1 und x_2 realisiert. Geben Sie außerdem ein WHILE–Programm an, das das Konstrukt

`IF (`$x_1 = 0$`) THEN {` A `}`

berechnet. Dabei sind nur die in der Vorlesung genannten Sprachelemente erlaubt.

Lösung.

```
1  addition(x1, x2) {
2    nat x0;
3    nat v1;
4
5    x0 := x1;
6
7    WHILE (x2 != 0) DO {
8          x0++;
9          x2--;
10   }
11
12   RETURN x0;
13 }
```

Das `IF THEN` Statement könnte so aussehen:

```
y := 1;
WHILE (x != 0) DO {
        y := 0;
        x--;
}
WHILE (y != 0) DO {
        A;
        y--;
}
```

Aufgabe 3)

Sei $\chi : \mathbb{N} \to \{0,1\}$ eine total berechenbare Funktion auf den natürlichen Zahlen. Die beschränkte Generalisierung $\bigwedge_{i=1}^{n}$ ist definiert durch

$$\bigwedge_{i=1}^{n} \chi(i) := \begin{cases} 1 & \text{falls} \quad \forall i \leq n : \chi(i) = 1 \\ 0 & \text{sonst} \end{cases}$$

Zeigen Sie durch Angabe eines WHILE–Programms dass die beschränkte Generalisierung berechenbar ist.

Lösung.

```
gen(x1) {
  nat x0;
  x0++;

  WHILE (x1 != 0) DO {
    IF (chi(x1) = 0) THEN {
      x0 := 0;
    }
    x1--;
  }

  IF (chi(0) = 0) THEN {
    x0 := 0;
  }

  RETURN x0;
}
```