Script generated by TTT

Title: Nipkow: Theo (11.07.2019)

Date: Thu Jul 11 14:16:22 CEST 2019

Duration: 87:12 min

Pages: 114

Satz 6.22
3KNF-SAT ist NP-vollständig.

Satz 6.22
3KNF-SAT ist NP-vollständig.

Beweis:
Wir zeigen SAT $\leq_p$ 3KNF-SAT mit einer polynomiellen Reduktion $F \mapsto F'$ so dass F' in 3KNF ist und

$$F \text{ ist erfüllbar } \Leftrightarrow \ F' \text{ ist erfüllbar}$$

Satz 6.22
3KNF-SAT ist NP-vollständig.

Beweis:
Wir zeigen SAT $\leq_p$ 3KNF-SAT mit einer polynomiellen Reduktion $F \mapsto F'$ so dass F' in 3KNF ist und

$$F \text{ ist erfüllbar } \Leftrightarrow \ F' \text{ ist erfüllbar}$$

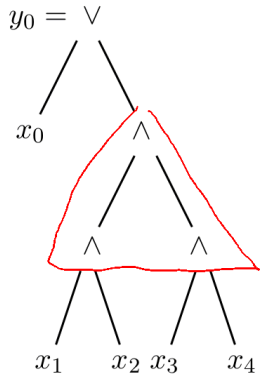
NB F und F' sind erfüllbarkeitsäquivalent, aber nicht notwendigerweise auch äquivalent.

1. Transformiere F in Negations-Normalform (NNF) durch fortgesetze Anwendung der de Morganschen Gesetze

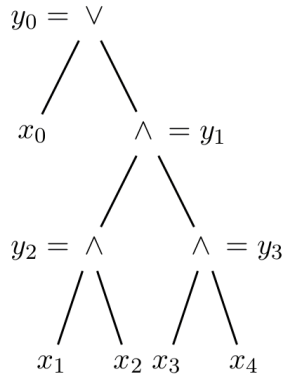
$$\begin{aligned} \neg(A \wedge B) &= \neg A \vee \neg B \\ \neg(A \vee B) &= \neg A \wedge \neg B \\ \neg\neg A &= A \end{aligned}$$

von links nach rechts.

Beispiel: $F_1 = x_0 \vee ((x_1 \wedge x_2) \wedge (x_3 \wedge x_4))$



Beispiel: $F_1 = x_0 \vee ((x_1 \wedge x_2) \wedge (x_3 \wedge x_4))$



$F_2 =$

$y_0 \wedge (y_0 \leftrightarrow (x_0 \vee y_1))$

Beweis (Forts.):

F_1 erf. $\implies$ F_2 erf.: y_i bekommt Wert seines Teilbaums.

MENGENÜBERDECKUNG (MÜ)

Gegeben: Teilmengen $T_1, \ldots, T_n \subseteq M$ einer endlichen Menge M und eine Zahl $k \leq n$.

Problem: Gibt es $i_1, \ldots, i_k \in \{1, \ldots, n\}$ mit $M = T_{i_1} \cup \cdots \cup T_{i_k}$?

MENGENÜBERDECKUNG (MÜ)

Gegeben: Teilmengen $T_1, \ldots, T_n \subseteq M$ einer endlichen Menge M und eine Zahl $k \leq n$.

Problem: Gibt es $i_1, \ldots, i_k \in \{1, \ldots, n\}$ mit $M = T_{i_1} \cup \cdots \cup T_{i_k}$?

Beispiel 6.25

$$\begin{array}{llll} T_1 & = & \{1,2\} & T_2 & = & \{1,3\} \\ T_3 & = & \{3,4\} & T_4 & = & \{3,5\} \\ M & = & \{1,2,3,4,5\} & k & = & 3 \end{array}$$

Satz 6.27

MÜ ist NP-vollständig.

Beweis: KNF-SAT $\leq_p$ MÜ.

Sei $F = K_1 \wedge \cdots \wedge K_m$ in KNF, mit Variablen $x_1, \ldots, x_l$.

Wir konstruieren:

Satz 6.27

MÜ ist NP-vollständig.

Beweis: KNF-SAT $\leq_p$ MÜ.

Sei $F = K_1 \wedge \cdots \wedge K_m$ in KNF, mit Variablen $x_1, \ldots, x_l$.

Wir konstruieren:

$$\begin{array}{rcl} M & := & \{1, \ldots, m\} \cup \{m+1, \ldots, m+l\} \\ T_i & := & \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\} \\ T_i' & := & \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\} \\ k & := & l \end{array}$$

F ist erfülbar
gdw
M wird durch l der Teilmengen $T_1, \ldots, T_l, T_1', \ldots, T_l'$ überdeckt

Beweis (Forts.):

Sei $F = K_1 \wedge \cdots \wedge K_m$ in KNF, mit Variablen $x_1, \ldots, x_l$.

$$\begin{array}{rcl} M & := & \{1, \ldots, m\} \cup \{m+1, \ldots, m+l\} \\ T_i & := & \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\} \\ T_i' & := & \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\} \end{array}$$

„$\Rightarrow$“ Gegeben σ mit $\sigma(F) = 1$.

$$U_i := \textbf{if } \sigma(x_i) = 1 \textbf{ then } T_i \textbf{ else } T_i'$$

Beh.: Die U_i überdecken M. Sei $j \in M$.

1. $1 \leq j \leq m$: Sei l ($= x_i$ oder $\neg x_i$) ein Literal in K_j mit $\sigma(l) = 1$.

Die Berechnung einer **Lösung** von

Gegeben: Teilmengen $\vec{T} := T_1, \ldots, T_n \subseteq M$ einer endlichen Menge M, und eine Zahl $k \leq n$.

Problem: Finde eine Überdeckung von M durch k der Teilmengen.

Die Berechnung einer **Lösung** von

Gegeben: Teilmengen $\vec{T} := T_1, \ldots, T_n \subseteq M$ einer endlichen Menge M, und eine Zahl $k \leq n$.

Problem: Finde eine Überdeckung von M durch k der Teilmengen.

kann auf das Entscheidungsproblem reduziert werden:

if $(\vec{T}, M, k) \notin$ MÜ **then** output("nicht lösbar")

CLIQUE

Gegeben: Ungerichteter Graph $G = (V, E)$ und Zahl $k \in \mathbb{N}$.

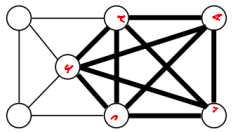
Problem: Besitzt G eine „Clique" der Größe mindestens k?
Dh eine Teilmenge $V' \subseteq V$ mit $|V'| \geq k$
und alle $u, v \in V'$ mit $u \neq v$ sind benachbart.

CLIQUE

Gegeben: Ungerichteter Graph $G = (V, E)$ und Zahl $k \in \mathbb{N}$.

Problem: Besitzt G eine „Clique" der Größe mindestens k?
Dh eine Teilmenge $V' \subseteq V$ mit $|V'| \geq k$
und alle $u, v \in V'$ mit $u \neq v$ sind benachbart.

Beispiel mit 5-Clique:

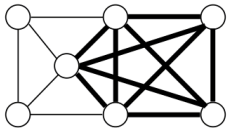


CLIQUE

Gegeben: Ungerichteter Graph $G = (V, E)$ und Zahl $k \in \mathbb{N}$.

Problem: Besitzt G eine „Clique“ der Größe mindestens k?
Dh eine Teilmenge $V' \subseteq V$ mit $|V'| \geq k$
und alle $u, v \in V'$ mit $u \neq v$ sind benachbart.

Beispiel mit 5-Clique:



Anwendung von Clique-Berechnung:

Knoten	=	Prozess
Kante	=	Zwei Prozesse sind parallel ausführbar

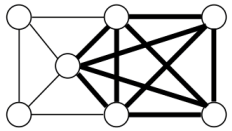
$\Longrightarrow$ Clique ist Gruppe von parallelisierbaren Prozessen

CLIQUE

Gegeben: Ungerichteter Graph $G = (V, E)$ und Zahl $k \in \mathbb{N}$.

Problem: Besitzt G eine „Clique“ der Größe mindestens k?
Dh eine Teilmenge $V' \subseteq V$ mit $|V'| \geq k$
und alle $u, v \in V'$ mit $u \neq v$ sind benachbart.

Beispiel mit 5-Clique:



Anwendung von Clique-Berechnung:

Knoten	=	Prozess
Kante	=	Zwei Prozesse sind parallel ausführbar

$\Longrightarrow$ Clique ist Gruppe von parallelisierbaren Prozessen

Fakt 6.28
CLIQUE $\in$ *NP*

414

Satz 6.29
CLIQUE ist NP-vollständig.

Satz 6.29
CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT $\leq_p$ CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \ldots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

415

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT $\leq_p$ CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \ldots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$V = \{(1,1),(1,2),(1,3),\ldots,(k,1),(k,2),(k,3)\}$$

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT $\leq_p$ CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \ldots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$\begin{aligned} V &= \{(1,1),(1,2),(1,3),\ldots,(k,1),(k,2),(k,3)\} \\ E &= \{\{(i,j),(p,q)\} \mid i \neq p \text{ und } z_{ij} \neq \neg z_{pq}\} \end{aligned}$$

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT $\leq_p$ CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \ldots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$\begin{aligned} V &= \{(1,1),(1,2),(1,3),\ldots,(k,1),(k,2),(k,3)\} \\ E &= \{\{(i,j),(p,q)\} \mid i \neq p \text{ und } z_{ij} \neq \neg z_{pq}\} \end{aligned}$$

F ist erfüllbar durch eine Belegung σ

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT $\leq_p$ CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \ldots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$\begin{aligned} V &= \{(1,1),(1,2),(1,3),\ldots,(k,1),(k,2),(k,3)\} \\ E &= \{\{(i,j),(p,q)\} \mid i \neq p \text{ und } z_{ij} \neq \neg z_{pq}\} \end{aligned}$$

F ist erfüllbar durch eine Belegung σ

$\iff$ Es gibt in jeder Klausel i ein Literal z_{ij_i} mit $\sigma(z_{ij_i}) = 1$

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT $\leq_p$ CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \ldots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$\begin{aligned} V &= \{(1,1),(1,2),(1,3),\ldots,(k,1),(k,2),(k,3)\} \\ E &= \{\{(i,j),(p,q)\} \mid i \neq p \text{ und } z_{ij} \neq \neg z_{pq}\} \end{aligned}$$

F ist erfüllbar durch eine Belegung σ
$\iff$ Es gibt in jeder Klausel i ein Literal z_{ij_i} mit $\sigma(z_{ij_i}) = 1$
$\iff$ Die Literale $z_{1j_1}, \ldots, z_{kj_k}$ sind paarweise nicht komplementär

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT $\leq_p$ CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \ldots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$\begin{aligned} V &= \{(1,1),(1,2),(1,3),\ldots,(k,1),(k,2),(k,3)\} \\ E &= \{\{(i,j),(p,q)\} \mid i \neq p \text{ und } z_{ij} \neq \neg z_{pq}\} \end{aligned}$$

F ist erfüllbar durch eine Belegung σ
$\iff$ Es gibt in jeder Klausel i ein Literal z_{ij_i} mit $\sigma(z_{ij_i}) = 1$
$\iff$ Die Literale $z_{1j_1}, \ldots, z_{kj_k}$ sind paarweise nicht komplementär
$\iff$ Die Knoten $(1,j_1), \ldots, (k,j_k)$ sind paarweise benachbart

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT $\leq_p$ CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \ldots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$\begin{aligned} V &= \{(1,1),(1,2),(1,3),\ldots,(k,1),(k,2),(k,3)\} \\ E &= \{\{(i,j),(p,q)\} \mid i \neq p \text{ und } z_{ij} \neq \neg z_{pq}\} \end{aligned}$$

F ist erfüllbar durch eine Belegung σ
$\iff$ Es gibt in jeder Klausel i ein Literal z_{ij_i} mit $\sigma(z_{ij_i}) = 1$
$\iff$ Die Literale $z_{1j_1}, \ldots, z_{kj_k}$ sind paarweise nicht komplementär
$\iff$ Die Knoten $(1,j_1), \ldots, (k,j_k)$ sind paarweise benachbart
$\iff$ G hat eine Clique der Größe k.

□

RUCKSACK

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$ und $b \in \mathbb{N}$.

Problem: Gibt es $R \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in R} a_i = b$?

RUCKSACK

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$ und $b \in \mathbb{N}$.

Problem: Gibt es $R \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in R} a_i = b$?

Satz 6.30

RUCKSACK ist NP-vollständig.

Beweis (Forts.):

Wir geben Zahlen v_{i0}, v_{i1} für jede variable x_i, Zahlen k_{j1}, k_{j2} für jede Klausel K_j, und eine Zahl b an.

PARTITION

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

PARTITION

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK $\leq_p$ PARTITION

Sei $a_1, a_2, \ldots, a_k, b$ Instanz von RUCKSACK und $M := \sum_{i=1}^{k} a_i$.

PARTITION

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK $\leq_p$ PARTITION

Sei $a_1, a_2, \ldots, a_k, b$ Instanz von RUCKSACK und $M := \sum_{i=1}^{k} a_i$.

$$a_1, a_2, \ldots, a_k, b \;\mapsto\; a_1, a_2, \ldots, a_k, M - b + 1, b + 1$$

PARTITION

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK $\leq_p$ PARTITION

Sei $a_1, a_2, \ldots, a_k, b$ Instanz von RUCKSACK und $M := \sum_{i=1}^{k} a_i$.

$$a_1, a_2, \ldots, a_k, b \;\mapsto\; a_1, a_2, \ldots, a_k, M - b + 1, b + 1$$

Beispiel: $a_1, \ldots, a_7 = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

PARTITION

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK $\leq_p$ PARTITION

Sei $a_1, a_2, \ldots, a_k, b$ Instanz von RUCKSACK und $M := \sum_{i=1}^{k} a_i$.

$$a_1, a_2, \ldots, a_k, b \;\mapsto\; a_1, a_2, \ldots, a_k, M - b + 1, b + 1$$

Beispiel: $a_1, \ldots, a_7 = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

Lösung: Die Zahlen $7, 9, 7, 15$, dh $R = \{2, 4, 5, 7\}$

PARTITION

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK $\leq_p$ PARTITION

Sei $a_1, a_2, \ldots, a_k, b$ Instanz von RUCKSACK und $M := \sum_{i=1}^{k} a_i$.

$$a_1, a_2, \ldots, a_k, b \;\mapsto\; a_1, a_2, \ldots, a_k, M - b + 1, b + 1$$

Beispiel: $a_1, \ldots, a_7 = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

Lösung: Die Zahlen $7, 9, 7, 15$, dh $R = \{2, 4, 5, 7\}$

Es gilt: $M = 57, \quad M - b + 1 = 20, \quad b + 1 = 39$.

PARTITION

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK $\leq_p$ PARTITION

Sei $a_1, a_2, \ldots, a_k, b$ Instanz von RUCKSACK und $M := \sum_{i=1}^{k} a_i$.

$$a_1, a_2, \ldots, a_k, b \;\mapsto\; a_1, a_2, \ldots, a_k, M - b + 1, b + 1$$

Beispiel: $a_1, \ldots, a_7 = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

Lösung: Die Zahlen $7, 9, 7, 15$, dh $R = \{2, 4, 5, 7\}$

Es gilt: $M = 57$, $M - b + 1 = 20$, $b + 1 = 39$.

Resultierendes PARTITIONS-Problem: $12, 7, 4, 9, 7, 3, 15, 20, 39$

PARTITION

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK $\leq_p$ PARTITION

Sei $a_1, a_2, \ldots, a_k, b$ Instanz von RUCKSACK und $M := \sum_{i=1}^{k} a_i$.

$$a_1, a_2, \ldots, a_k, b \;\mapsto\; a_1, a_2, \ldots, a_k, M - b + 1, b + 1$$

Beispiel: $a_1, \ldots, a_7 = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

Lösung: Die Zahlen $7, 9, 7, 15$, dh $R = \{2, 4, 5, 7\}$

Es gilt: $M = 57$, $M - b + 1 = 20$, $b + 1 = 39$.

Resultierendes PARTITIONS-Problem: $12, 7, 4, 9, 7, 3, 15, 20, 39$

Lösung: $7 + 9 + 7 + 15 + 20 = 12 + 4 + 3 + 39$. □

PARTITION

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK $\leq_p$ PARTITION

Sei $a_1, a_2, \ldots, a_k, b$ Instanz von RUCKSACK und $M := \sum_{i=1}^{k} a_i$.

$$a_1, a_2, \ldots, a_k, b \;\mapsto\; a_1, a_2, \ldots, a_k, M - b + 1, b + 1$$

Beispiel: $a_1, \ldots, a_7 = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

Lösung: Die Zahlen $7, 9, 7, 15$, dh $R = \{2, 4, 5, 7\}$

Es gilt: $M = 57$, $M - b + 1 = 20$, $b + 1 = 39$.

PARTITION

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK $\leq_p$ PARTITION

Sei $a_1, a_2, \ldots, a_k, b$ Instanz von RUCKSACK und $M := \sum_{i=1}^{k} a_i$.

$$a_1, a_2, \ldots, a_k, b \;\mapsto\; a_1, a_2, \ldots, a_k, M - b + 1, b + 1$$

Beispiel: $a_1, \ldots, a_7 = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

BIN PACKING

Gegeben: Eine „Behältergröße“ $b \in \mathbb{N}$, die Anzahl der Behälter $k \in \mathbb{N}$ und „Objekte“ $a_1, a_2, \ldots a_n$.

Problem: Können die Objekte so auf die k Behälter verteilt werden, dass kein Behälter überläuft?

BIN PACKING

Gegeben: Eine „Behältergröße“ $b \in \mathbb{N}$, die Anzahl der Behälter $k \in \mathbb{N}$ und „Objekte“ $a_1, a_2, \ldots a_n$.

Problem: Können die Objekte so auf die k Behälter verteilt werden, dass kein Behälter überläuft?

Satz 6.32

BIN PACKING ist NP-vollständig.

420

HAMILTONKREIS

Gegeben: Ungerichteter Graph G

Problem: Enthält G einen Hamilton-Kreis, dh einen geschlossenen Pfad, der jeden Knoten genau einmal enthält?

HAMILTONKREIS

Gegeben: Ungerichteter Graph G

Problem: Enthält G einen Hamilton-Kreis, dh einen geschlossenen Pfad, der jeden Knoten genau einmal enthält?

Satz 6.33

HAMILTON ist NP-vollständig.

421

TRAVELLING SALESMAN (TSP)

Gegeben: Eine $n \times n$ Matrix $M_{ij} \in \mathbb{N}$ von „Entfernungen“ und eine Zahl $k \in \mathbb{N}$

Problem: Gibt es eine „Rundreise“ (Hamilton-Kreis) der Länge $\leq k$?

TRAVELLING SALESMAN (TSP)

Gegeben: Eine $n \times n$ Matrix $M_{ij} \in \mathbb{N}$ von „Entfernungen“ und eine Zahl $k \in \mathbb{N}$

Problem: Gibt es eine „Rundreise“ (Hamilton-Kreis) der Länge $\leq k$?

Satz 6.34

TSP ist NP-vollständig.

FÄRBBARKEIT (COL)

Gegeben: Ein ungerichteter Graph (V, E) und eine Zahl k.

Problem: Gibt es eine Färbung der Knoten V mit k Farben, so dass keine zwei benachbarten Knoten die gleiche Farbe haben?

FÄRBBARKEIT (COL)

Gegeben: Ein ungerichteter Graph (V, E) und eine Zahl k.

Problem: Gibt es eine Färbung der Knoten V mit k Farben, so dass keine zwei benachbarten Knoten die gleiche Farbe haben?

Satz 6.35

FÄRBBARKEIT ist NP-vollständig für $k \geq 3$.

FÄRBBARKEIT (COL)

Gegeben: Ein ungerichteter Graph (V, E) und eine Zahl k.

Problem: Gibt es eine Färbung der Knoten V mit k Farben, so dass keine zwei benachbarten Knoten die gleiche Farbe haben?

Satz 6.35
FÄRBBARKEIT ist NP-vollständig für $k \geq 3$.

Beweis:
3KNF-SAT $\leq_p$ 3FÄRBBARKEIT □

Die NP-Bibel, der NP-Klassiker:


Michael Garey and David Johnson.
Computers and Intractability: A Guide to the Theory of NP-Completeness. 1979.

Despite the 23 years that have passed since its publication, I consider Garey and Johnson the single most important book on my office bookshelf.

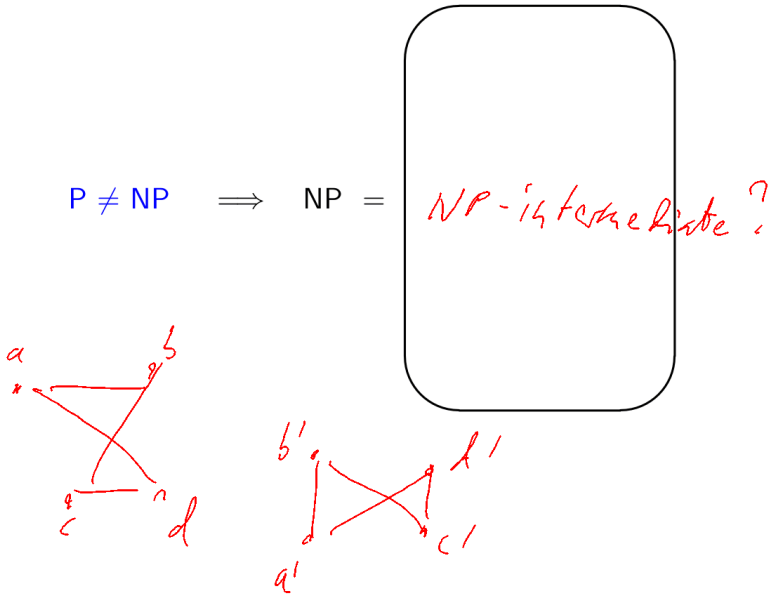
Lance Fortnow, 2002.

Die NP-Bibel, der NP-Klassiker:


Michael Garey and David Johnson.
Computers and Intractability: A Guide to the Theory of NP-Completeness. 1979.

Man weiß nicht ob P = NP.

Man weiß nicht ob P = NP. Aber man weiß (Ladner 1975)



6.5 Approximation Algorithms

What to do with NP-complete problems?

6.5 Approximation Algorithms

What to do with NP-complete problems?

- more computational power?
- encode into SAT
- approximation algorithms

Definition 6.37

A d-approximation algorithm ($d \in \mathbb{R}$) for an optimization problem is an algorithm that computes in polynomial time a solution to the problem that is at most d times worse than the optimal solution.

Travelling Salesman Problem

Definition 6.38 (TSP)

Given a *complete*, weighted, undirected graph $G = (V, E)$ with non-negative weights $c\colon V \to \mathbb{N}$, find a cycle that visits exactly all nodes and does so with *minimal length*.

Travelling Salesman Problem

Definition 6.38 (TSP)

Given a *complete*, weighted, undirected graph $G = (V, E)$ with non-negative weights c: $V \to \mathbb{N}$, find a cycle that visits exactly all nodes and does so with *minimal length*.

Properties

- We can assume *triangle inequality*:

$$\forall u, v, w \in V. c(u, v) \leq c(u, w) + c(w, v)$$

- NP-complete
- We show a 2-approximation

Travelling Salesman Problem

Definition 6.38 (TSP)

Given a *complete*, weighted, undirected graph $G = (V, E)$ with non-negative weights c: $V \to \mathbb{N}$, find a cycle that visits exactly all nodes and does so with *minimal length*.

Properties

- We can assume *triangle inequality*:

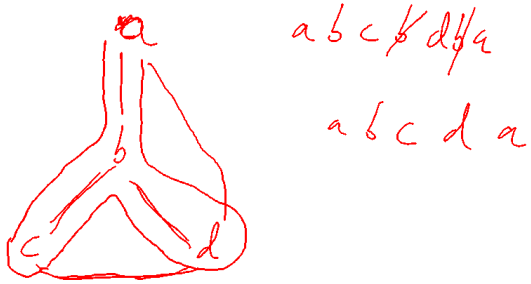
$$\forall u, v, w \in V. c(u, v) \leq c(u, w) + c(w, v)$$

- NP-complete
- We show a 2-approximation
- There is a 1.5-approximation

2-Approximation Algorithm for TSP

Algorithm

1. T := a minimum spanning tree
2. cycle := traverse along depth-first search of T, jumping over visited nodes



2-Approximation Algorithm for TSP

Algorithm

1. T := a minimum spanning tree
2. cycle := traverse along depth-first search of T, jumping over visited nodes

Algorithm is

- *polynomial*

2-Approximation Algorithm for TSP

Algorithm

1. T := a minimum spanning tree
2. cycle := traverse along depth-first search of T, jumping over visited nodes

Algorithm is

- *polynomial*
- *2-approximation*

2-Approximation Algorithm for TSP

Algorithm

1. T := a minimum spanning tree
2. cycle := traverse along depth-first search of T, jumping over visited nodes

Algorithm is

- *polynomial*
- *2-approximation*
  - $c(T) \leq$ minimal cycle

2-Approximation Algorithm for TSP

Algorithm

1. T := a minimum spanning tree
2. cycle := traverse along depth-first search of T, jumping over visited nodes

Algorithm is

- *polynomial*
- *2-approximation*
  - $c(T) \leq$ minimal cycle
  - traversal costs $2 \cdot c(T)$ since jumping over costs at most the sum of traversed edges

Knapsack

Definition 6.39 (Knapsack)

Given weight W of knapsack and *weights* and *values* of n items: $w_1, \ldots, w_n, v_1, \ldots, v_n$, pick $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} w_i \leq W$ and $\sum_{i \in I} v_i$ is maximal.

Knapsack

Definition 6.39 (Knapsack)

Given weight W of knapsack and *weights* and *values* of n items: $w_1, \ldots, w_n, v_1, \ldots, v_n$, pick $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} w_i \leq W$ and $\sum_{i \in I} v_i$ is maximal.

Greedy Algorithm

- take items in the order $v_1/w_1 \geq v_2/w_2 \cdots \geq v_n/w_n$

Knapsack

Definition 6.39 (Knapsack)

Given weight W of knapsack and *weights* and *values* of n items: $w_1, \ldots, w_n, v_1, \ldots, v_n$, pick $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} w_i \leq W$ and $\sum_{i \in I} v_i$ is maximal.

Greedy Algorithm

- take items in the order $v_1/w_1 \geq v_2/w_2 \cdots \geq v_n/w_n$

Properties

- optimal for "fractional" knapsack problem

Knapsack

Definition 6.39 (Knapsack)

Given weight W of knapsack and *weights* and *values* of n items: $w_1, \ldots, w_n, v_1, \ldots, v_n$, pick $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} w_i \leq W$ and $\sum_{i \in I} v_i$ is maximal.

Greedy Algorithm

- take items in the order $v_1/w_1 \geq v_2/w_2 \cdots \geq v_n/w_n$

Properties

- optimal for "fractional" knapsack problem
- for $v_1 = 1.001, w_1 = 1, v_2 = W, w_2 = W$ no better than a W-approximation.

$v_1/w_1 > v_2/w_2$

2-Approximation of Knapsack

Modified Greedy Algorithm (ModGreedy):

- $S_1 :=$ solution by Greedy

2-Approximation of Knapsack

Modified Greedy Algorithm (ModGreedy):

- $S_1 :=$ solution by Greedy
- $S_2 :=$ item with the largest value

2-Approximation of Knapsack

Modified Greedy Algorithm (ModGreedy):

- $S_1 :=$ solution by Greedy
- $S_2 :=$ item with the largest value
- Return whichever of S_1, S_2 that has more value

2-Approximation of Knapsack

Modified Greedy Algorithm (ModGreedy):

- $S_1 :=$ solution by Greedy
- $S_2 :=$ item with the largest value
- Return whichever of S_1, S_2 that has more value

Lemma 6.40
ModGreedy is a 2-approximation.

PTAS for Knapsack

- Polynomial-time approximation scheme (PTAS): any approximation ratio possible

PTAS for Knapsack

- Polynomial-time approximation scheme (PTAS): any approximation ratio possible
- Idea: brute-force a part of the solution and then use Greedy Algorithm to finish up the rest

PTAS for Knapsack

- Polynomial-time approximation scheme (PTAS): any approximation ratio possible
- Idea: brute-force a part of the solution and then use Greedy Algorithm to finish up the rest

Algorithm, k fixed constant

- for all possible subsets of objects that have up to k objects: use the greedy algorithm to fill up the rest of the knapsack

PTAS for Knapsack

- Polynomial-time approximation scheme (PTAS): any approximation ratio possible
- Idea: brute-force a part of the solution and then use Greedy Algorithm to finish up the rest

Algorithm, k fixed constant

- for all possible subsets of objects that have up to k objects: use the greedy algorithm to fill up the rest of the knapsack
- return the most profitable subset

PTAS for Knapsack

- Polynomial-time approximation scheme (PTAS): any approximation ratio possible
- Idea: brute-force a part of the solution and then use Greedy Algorithm to finish up the rest

Algorithm, k fixed constant

- for all possible subsets of objects that have up to k objects: use the greedy algorithm to fill up the rest of the knapsack
- return the most profitable subset

Properties

- runtime $\mathcal{O}(kn^k)$ subsets, filling up in $\mathcal{O}(n)$
- thus total running time $\mathcal{O}(kn^{k+1})$

PTAS for Knapsack

- Polynomial-time approximation scheme (PTAS): any approximation ratio possible
- Idea: brute-force a part of the solution and then use Greedy Algorithm to finish up the rest

Algorithm, k fixed constant

- for all possible subsets of objects that have up to k objects: use the greedy algorithm to fill up the rest of the knapsack
- return the most profitable subset

Properties

- runtime $\mathcal{O}(kn^k)$ subsets, filling up in $\mathcal{O}(n)$
- thus total running time $\mathcal{O}(kn^{(k+1)})$
- $(1+\frac{1}{k})-$approximation

7. Automaten, Berechenbarkeit, und Logik

Vorlesungen

- 25. April
- 29. April
- 2. Mai
- 6. Mai
- 9. Mai
- 13. Mai
- 16. Mai
- 20. Mai
- 23. Mai
- 27. Mai
- 3. Juni
- 6. Juni
- 13. Juni
- 17. Juni
- 24. Juni
- 27. Juni
- 1. Juli
- 4. Juli
- 8. Juli
- 11. Juli

LOOP–Berechenbarkeit

Statt `while`-Schleifen betrachten wir nun `for`-Schleifen.

LOOP–Berechenbarkeit

Statt `while`-Schleifen betrachten wir nun `for`-Schleifen.

LOOP-Programme haben die gleiche Syntax WHILE-Programme aber statt der WHILE-Schleifen gibt es nur LOOP-Schleifen mit folgender Syntax:

`LOOP` X `DO` P `END`

LOOP–Berechenbarkeit

Statt `while`-Schleifen betrachten wir nun `for`-Schleifen.

LOOP-Programme haben die gleiche Syntax WHILE-Programme aber statt der WHILE-Schleifen gibt es nur LOOP-Schleifen mit folgender Syntax:

`LOOP` X `DO` P `END`

Semantik:
Führe P genau n mal aus, wobei n der *Anfangswert* von X ist.

LOOP–Berechenbarkeit

Statt `while`-Schleifen betrachten wir nun `for`-Schleifen.

LOOP-Programme haben die gleiche Syntax WHILE-Programme aber statt der WHILE-Schleifen gibt es nur LOOP-Schleifen mit folgender Syntax:

`LOOP` X `DO` P `END`

Semantik:
Führe P genau n mal aus, wobei n der *Anfangswert* von X ist.
Zuweisungen an X in P ändern die Anzahl n der Schleifendurchläufe nicht.

Beispiel 5.26

`LOOP` x_0 `DO` x_1 `:=` x_1`+1` `END` simuliert x_1 `:=` x_1 `+` x_0

Definition 5.27
Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist LOOP-berechenbar gdw es ein LOOP-Programm P gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$:

P, gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 in den anderen Var.) terminiert mit $f(n_1, \ldots, n_k)$ in x_0.

Definition 5.27
Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist LOOP-berechenbar gdw es ein LOOP-Programm P gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$:

P, gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 in den anderen Var.) terminiert mit $f(n_1, \ldots, n_k)$ in x_0.

Lemma 5.28
Alle LOOP-berechenbaren Funktionen sind total.

Definition 5.27
Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist LOOP-berechenbar gdw es ein LOOP-Programm P gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$:

P, gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 in den anderen Var.) terminiert mit $f(n_1, \ldots, n_k)$ in x_0.

Lemma 5.28
Alle LOOP-berechenbaren Funktionen sind total.

Beweis mit Induktion über die Syntax der LOOP-Programme.

267

Definition 5.27
Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist LOOP-berechenbar gdw es ein LOOP-Programm P gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$:

P, gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 in den anderen Var.) terminiert mit $f(n_1, \ldots, n_k)$ in x_0.

Lemma 5.28
Alle LOOP-berechenbaren Funktionen sind total.

Beweis mit Induktion über die Syntax der LOOP-Programme.

Sind alle totalen Funktionen LOOP-berechenbar?

Fakt 5.29
WHILE-Schleifen können LOOP-Schleifen simulieren.

267

Definition 5.27
Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist LOOP-berechenbar gdw es ein LOOP-Programm P gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$:

P, gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 in den anderen Var.) terminiert mit $f(n_1, \ldots, n_k)$ in x_0.

Lemma 5.28
Alle LOOP-berechenbaren Funktionen sind total.

Beweis mit Induktion über die Syntax der LOOP-Programme.

Sind alle totalen Funktionen LOOP-berechenbar?

Fakt 5.29
WHILE-Schleifen können LOOP-Schleifen simulieren.

Fakt 5.30
LOOP-Schleifen können WHILE-Schleifen nicht simulieren.

267

5.5 Primitiv rekursive Funktionen

5.5 Primitiv rekursive Funktionen

Klassen von Programmiersprachen

Imperativ	Funktional
C, Java,... TM, WHILE, GOTO	OCaml, Lisp, ... μ-rekursiv
LOOP	Primitiv rekursiv

Wir betrachten Funktionen $\mathbb{N}^k \to \mathbb{N}$, $k \geq 0$.

5.5 Primitiv rekursive Funktionen

Klassen von Programmiersprachen

Imperativ	Funktional
C, Java,... TM, WHILE, GOTO	OCaml, Lisp, ... μ-rekursiv
LOOP	Primitiv rekursiv

Wir betrachten Funktionen $\mathbb{N}^k \to \mathbb{N}$, $k \geq 0$.
Wir identifizieren $\mathbb{N}^0 \to \mathbb{N}$ mit $\mathbb{N}$ und $c()$ mit c.

5.5 Primitiv rekursive Funktionen

Klassen von Programmiersprachen

Imperativ	Funktional
C, Java,... TM, WHILE, GOTO	OCaml, Lisp, ... μ-rekursiv
LOOP	Primitiv rekursiv

Wir betrachten Funktionen $\mathbb{N}^k \to \mathbb{N}$, $k \geq 0$.
Wir identifizieren $\mathbb{N}^0 \to \mathbb{N}$ mit $\mathbb{N}$ und $c()$ mit c.

Definition der primitiv rekursiven Funktionen:

Klassen von Programmiersprachen

Imperativ	Funktional
C, Java,...	OCaml, Lisp, ...
TM, WHILE, GOTO	μ-rekursiv
LOOP	Primitiv rekursiv

Wir betrachten Funktionen $\mathbb{N}^k \to \mathbb{N}$, $k \geq 0$.
Wir identifizieren $\mathbb{N}^0 \to \mathbb{N}$ mit $\mathbb{N}$ und $c()$ mit c.

Definition der primitiv rekursiven Funktionen:

Fixe Basisfunktionen zB $s(x) = x + 1$

Funktionskomposition zB $f(x, y) = g(x, h(x, y))$

Klassen von Programmiersprachen

Imperativ	Funktional
C, Java,...	OCaml, Lisp, ...
TM, WHILE, GOTO	μ-rekursiv
LOOP	Primitiv rekursiv

Wir betrachten Funktionen $\mathbb{N}^k \to \mathbb{N}$, $k \geq 0$.
Wir identifizieren $\mathbb{N}^0 \to \mathbb{N}$ mit $\mathbb{N}$ und $c()$ mit c.

Definition der primitiv rekursiven Funktionen:

Fixe Basisfunktionen zB $s(x) = x + 1$

Funktionskomposition zB $f(x, y) = g(x, h(x, y))$

Fixe Art der Rekursion zB
$$\begin{aligned} f(0) &= 1 \\ f(n+1) &= n * f(n) \end{aligned}$$

Definition 5.31 (Basisfunktionen)

- Die konstante Funktion 0

Definition 5.31 (Basisfunktionen)

- Die konstante Funktion 0
- Die Nachfolgerfunktion $s(n) = n + 1$

Definition 5.31 (Basisfunktionen)

- Die konstante Funktion 0
- Die Nachfolgerfunktion $s(n) = n + 1$
- Die Projektionsfunktionen $\pi_i^k : \mathbb{N}^k \to \mathbb{N}$, $1 \leq i \leq k$:

$$\pi_i^k(x_1, \ldots, x_k) = x_i$$

Definition 5.31 (Basisfunktionen)

- Die konstante Funktion 0
- Die Nachfolgerfunktion $s(n) = n + 1$
- Die Projektionsfunktionen $\pi_i^k : \mathbb{N}^k \to \mathbb{N}$, $1 \leq i \leq k$:

$$\pi_i^k(x_1, \ldots, x_k) = x_i$$

Definition 5.32

Die Komposition von g und $h_1, \ldots, h_k$ erzeugt die Funktion f

$$f(\overline{x}) \;=\; g(h_1(\overline{x}), \ldots, h_k(\overline{x}))$$

Definition 5.31 (Basisfunktionen)

- Die konstante Funktion 0
- Die Nachfolgerfunktion $s(n) = n + 1$
- Die Projektionsfunktionen $\pi_i^k : \mathbb{N}^k \to \mathbb{N}$, $1 \leq i \leq k$:

$$\pi_i^k(x_1, \ldots, x_k) = x_i$$

Definition 5.32

Die Komposition von g und $h_1, \ldots, h_k$ erzeugt die Funktion f

$$f(\overline{x}) \;=\; g(h_1(\overline{x}), \ldots, h_k(\overline{x}))$$

wobei $\overline{x} = (x_1, \ldots, x_n)$ und

$$\begin{aligned} f &: \mathbb{N}^n \to \mathbb{N} \\ g &: \mathbb{N}^k \to \mathbb{N} \\ h_i &: \mathbb{N}^n \to \mathbb{N} \qquad (i = 1, \ldots, k) \end{aligned}$$

Definition 5.33

Das Schema der primitiven Rekursion erzeugt aus g und h die Funktion f

$$\begin{aligned} f(0, \overline{x}) &= g(\overline{x}) \\ f(m+1, \overline{x}) &= h(f(m, \overline{x}), m, \overline{x}) \end{aligned}$$

Definition 5.33

Das Schema der primitiven Rekursion erzeugt aus g und h die Funktion f

$$\begin{aligned} f(0, \overline{x}) &= g(\overline{x}) \\ f(m+1, \overline{x}) &= h(f(m, \overline{x}), m, \overline{x}) \end{aligned}$$

wobei $\overline{x} = (x_1, \ldots, x_n)$ und

$$\begin{aligned} f &: \mathbb{N}^{n+1} \to \mathbb{N} \\ g &: \mathbb{N}^{n} \to \mathbb{N} \\ h &: \mathbb{N}^{n+2} \to \mathbb{N} \end{aligned}$$

Definition 5.34 (PR)

Die Menge PR der primitiv rekursiven Funktionen ist die folgende induktiv definierte Teilmenge aller Funktionen $\mathbb{N}^k \to \mathbb{N}$, $k \geq 0$:

Definition 5.34 (PR)

Die Menge PR der primitiv rekursiven Funktionen ist die folgende induktiv definierte Teilmenge aller Funktionen $\mathbb{N}^k \to \mathbb{N}$, $k \geq 0$:

Definition 5.34 (PR)

Die Menge PR der primitiv rekursiven Funktionen ist die folgende induktiv definierte Teilmenge aller Funktionen $\mathbb{N}^k \to \mathbb{N}$, $k \geq 0$:

- Die Basisfunktionen 0, s, und π_i^k sind primitiv rekursiv.
- Sind g und h_i primitiv rekursiv, dann auch ihre Komposition

$$f(\overline{x}) = g(h_1(\overline{x}), \ldots, h_k(\overline{x}))$$

- Sind g und h primitiv rekursiv, dann auch die mit primitiver Rekursion definierte Funktion

$$\begin{aligned} f(0, \overline{x}) &= g(\overline{x}) \\ f(m+1, \overline{x}) &= h(f(m, \overline{x}), m, \overline{x}) \end{aligned}$$

Definition 5.34 (PR)

Die Menge PR der primitiv rekursiven Funktionen ist die folgende induktiv definierte Teilmenge aller Funktionen $\mathbb{N}^k \to \mathbb{N}$, $k \geq 0$:

- Die Basisfunktionen 0, s, und π_i^k sind primitiv rekursiv.
- Sind g und h_i primitiv rekursiv, dann auch ihre Komposition

$$f(\overline{x}) = g(h_1(\overline{x}), \ldots, h_k(\overline{x}))$$

- Sind g und h primitiv rekursiv, dann auch die mit primitiver Rekursion definierte Funktion

$$\begin{aligned} f(0, \overline{x}) &= g(\overline{x}) \\ f(m+1, \overline{x}) &= h(f(m, \overline{x}), m, \overline{x}) \end{aligned}$$

Lemma 5.35

Jede primitiv-rekursive Funktion ist total.

Definition 5.34 (PR)

Die Menge PR der primitiv rekursiven Funktionen ist die folgende induktiv definierte Teilmenge aller Funktionen $\mathbb{N}^k \to \mathbb{N}$, $k \geq 0$:

- Die Basisfunktionen 0, s, und π_i^k sind primitiv rekursiv.
- Sind g und h_i primitiv rekursiv, dann auch ihre Komposition

$$f(\overline{x}) = g(h_1(\overline{x}), \ldots, h_k(\overline{x}))$$

- Sind g und h primitiv rekursiv, dann auch die mit primitiver Rekursion definierte Funktion

$$\begin{aligned} f(0, \overline{x}) &= g(\overline{x}) \\ f(m+1, \overline{x}) &= h(f(m, \overline{x}), m, \overline{x}) \end{aligned}$$

Lemma 5.35

Jede primitiv-rekursive Funktion ist total.

Beweis:

Induktion über den Aufbau der primitiv-rekursiven Funktionen. □

Beispiel 5.36

Alle Konstanten $n \in \mathbb{N}$ sind PR: $1 = s(0)$, $2 = s(1)$, ...

Definition 5.34 (PR)

Die Menge PR der primitiv rekursiven Funktionen ist die folgende induktiv definierte Teilmenge aller Funktionen $\mathbb{N}^k \to \mathbb{N}$, $k \geq 0$:

- Die Basisfunktionen 0, s, und π_i^k sind primitiv rekursiv.
- Sind g und h_i primitiv rekursiv, dann auch ihre Komposition

$$f(\overline{x}) = g(h_1(\overline{x}), \ldots, h_k(\overline{x}))$$

- Sind g und h primitiv rekursiv, dann auch die mit primitiver Rekursion definierte Funktion

$$\begin{aligned} f(0, \overline{x}) &= g(\overline{x}) \\ f(m+1, \overline{x}) &= h(f(m, \overline{x}), m, \overline{x}) \end{aligned}$$

Lemma 5.35

Jede primitiv-rekursive Funktion ist total.

Beispiel 5.36

Alle Konstanten $n \in \mathbb{N}$ sind PR: $1 = s(0)$, $2 = s(1)$, $\ldots$

Beispiel 5.36

Alle Konstanten $n \in \mathbb{N}$ sind PR: $1 = s(0)$, $2 = s(1)$, $\ldots$

Beispiel 5.36

Alle Konstanten $n \in \mathbb{N}$ sind PR: $1 = s(0)$, $2 = s(1)$, $\ldots$

Addition ist PR:

Beispiel 5.36

Alle Konstanten $n \in \mathbb{N}$ sind PR: $1 = s(0)$, $2 = s(1)$, $\ldots$

Addition ist PR:

$$\begin{aligned} h(r,x,y) &= s(\pi_1^3(r,x,y)) \\ add(0,y) &= \pi_1^1(y) \\ add(x+1,y) &= h(add(x,y),x,y) \end{aligned}$$

Beispiel 5.36

Alle Konstanten $n \in \mathbb{N}$ sind PR: $1 = s(0)$, $2 = s(1)$, ...

Addition ist PR:

Beispiel 5.36

Alle Konstanten $n \in \mathbb{N}$ sind PR: $1 = s(0)$, $2 = s(1)$, ...

Beispiel 5.36

Alle Konstanten $n \in \mathbb{N}$ sind PR: $1 = s(0)$, $2 = s(1)$, ...

Addition ist PR:

$$\begin{aligned} h(r,x,y) &= s(\pi_1^3(r,x,y)) \\ add(0,y) &= \pi_1^1(y) \\ add(x+1,y) &= h(add(x,y),x,y) \end{aligned}$$

Lesbarer, aber nicht dem syntaktischen PR-Format entsprechend:

$$\begin{aligned} add(0,y) &= y \\ add(x+1,y) &= s(add(x,y)) \end{aligned}$$