Theoretische Informatik
Übungsblatt 1 (2024W)
Lösungsvorschlag

Allgemeine Hinweise:

- Die Deadline für die Abgabe der Lösungen zum Übungsblatt 1 ist der **25. Oktober 2024**.
- Wiederholung von der Vorbesprechung: Eine Abgabe besteht aus *einer* PDF-Datei. Punkte gibt es für jeden ernsthaften Lösungsversuch, auch wenn die Lösung falsch ist. Nur abgeschriebene Lösungen werden mit 0 Punkten bewertet. Die Verwendung von Latex wird nachdrücklich empfohlen (ein Latex template wurde auf TUWEL bereitgetstellt). Handschriftliche Lösungen sind okay, solange sie *gut* lesbar sind.
- Das Übungsblatt 1 enthält 10 Fragen. Jede Frage ist max. 10 Punkte wert, sodass auf das Übungsblatt 1 max. 100 Punkte erzielt werden können. Gemeinsam mit den kommenden 3 Übungsblättern werden insgesamt max. 400 Punkte erreichbar sein. Das Gesamtergebnis für den Übungsteil der VU wird dann durch Division mit 10 und Aufrunden berechnet (also max. 40 Punkte auf den Übungsteil). Auf diese Weise werden Bruchteile von Punkten sowie mehrmaliges Runden vermieden.

Aufgabe 1.1

Ein klassisches Problem in der Graphentheorie ist das k-FÄRBBARKEIT Problem. (In Teil 1 der Vorlesung wurde der Spezialfall des 2-FÄRBBARKEIT Problems betrachtet. In Teil 4 der Vorlesung wird das Problem für weitere Werte von k betrachtet, insbesondere 3-FÄRBBARKEIT). Dabei besteht die Aufgabe darin, in einem gegebenen Graphen jedem Knoten jeweils eine von k möglichen Farben zuzuweisen, so dass benachbarte Knoten nie die selbe Farbe haben. In seiner ursprünglichen Form ist das k-FÄRBBARKEIT Problem ein Entscheidungsproblem. Es lassen sich aber auch alle anderen in der Vorlesung vorgestellten Problemtypen rund um k-FÄRBBARKEIT formulieren. Formulieren Sie Beispiele für die folgenden Typen von Problemen (Instanz? Frage?) im Zusammenhang mit dem k-FÄRBBARKEIT Problem.

Tipp: Mathematisch gesprochen, ist eine k-Färbung der Knoten eine Funktion $f\colon V \to \{1,\ldots,k\}$, wobei V die Knotenmenge ist.

a) Beispiel für ein Entscheidungsproblem

b) Beispiel für ein Aufzählproblem

c) Beispiel für ein Zählproblem

d) Beispiel für ein Optimierungsproblem

e) Beispiel für ein Suchproblem

Lösung 1.1

a) Entscheidungsproblem:
Instanz: $G = (V,E)$ mit Knotenmenge V, Kantenmenge E und Integer k;
Frage: Besitzt der Graph G eine gültige k-Färbung? D.h.: Existiert eine Funktion $f\colon V \to \{1,\ldots,k\}$, so dass für alle Kanten $e = \{u,v\}$ aus E die Eigenschaft $f(u) \neq f(v)$ gilt?

b) Aufzählproblem:
Instanz: $G = (V,E)$ mit Knotenmenge V, Kantenmenge E und Integer k;
Frage: Ausgabe aller gültigen k-Färbungen. D.h.: Alle Funktionen $f\colon V \to \{1,\ldots,k\}$, so dass für alle Kanten $e = \{u,v\}$ aus E die Eigenschaft $f(u) \neq f(v)$ gilt.

Bemerkung: In der Praxis interessiert man sich möglicherweise nur für alle k-Färbungen modulo Permutation der Farben, d.h.: Von 2 Lösungen, die sich nur durch ein "Vertauschen" der Farben unterscheiden, möchte man nur eine ausgeben. Mathematisch gesprochen, gilt für 2 solche Lösungen f_1 und f_2, dass eine Permutation π der Zahlen $\{1,\ldots,k\}$ mit der Eigenschaft $f_2(\cdot) = \pi(f_1(\cdot))$ existiert.

c) Zählproblem:
Instanz: $G = (V, E)$ mit Knotenmenge V, Kantenmenge E und Integer k;
Frage: Wie viele gültige k-Färbungen hat der Graph G? D.h.: Wie viele Funktionen $f : V \to \{1, \ldots, k\}$ gibt es mit der Eigenschaft, dass für alle Kanten $e = \{u, v\}$ aus E die Eigenschaft $f(u) \neq f(v)$ gilt?

Bemerkung: Wie schon beim obigen Aufzählproblem wird man sich in der Praxis möglicherweise nur für die Anzahl der k-Färbungen modulo Permutation der Farben interessieren.

d) Optimierungsproblem:
Instanz: $G = (V, E)$ mit Knotenmenge V, Kantenmenge E;
Frage: Kleinste Zahl k, so dass der Graph G eine gültige k-Färbung besitzt. D.h.: Minimaler Wert k, für den eine Funktion $f : V \to \{1, \ldots, k\}$ existiert, so dass für alle Kanten $e = \{u, v\}$ aus E die Eigenschaft $f(u) \neq f(v)$ gilt?

Bemerkung: Dieser minimale Wert k wird als "chromatische Zahl" des Graphen G bezeichnet.

e) Suchproblem:
Instanz: $G = (V, E)$ mit Knotenmenge V, Kantenmenge E und Integer k;
Frage: Ausgabe einer gültigen k-Färbung des Graphen G. D.h.: Ausgabe einer Funktion $f : V \to \{1, \ldots, k\}$ mit der Eigenschaft, dass für alle Kanten $e = \{u, v\}$ aus E die Eigenschaft $f(u) \neq f(v)$ gilt.

Aufgabe 1.2
Beweisen Sie mittels Diagonalargument, dass das Intervall $[3.2, 3.21)$ überabzählbar viele reelle Zahlen enthält.

Lösung 1.2
Indirekter Beweis: Wir nehmen an, dass die reellen Zahlen im Intervall $[3.2, 3.21)$ abzählbar sind. Das heißt, wir können Sie aufzählen als $z_0, z_1, z_2, \ldots$. Alle Zahlen $z_i \in [3.2, 3.21)$ lassen sich als Dezimalzahlen der Form $z_i = 3.20a_{i0}a_{i1}a_{i2}a_{i3}\ldots$ darstellen, wobei alle a_{ij} Ziffern aus dem Bereich $\{0, \ldots, 9\}$ sind. Mit anderen Worten, a_{ij} ist die $(j + 3)$-te Nachkommastelle der i-ten Zahl in dieser Aufzählung. Die Aufzählung *aller* Zahlen $z_0, z_1, z_2, \ldots$ hat daher folgende Gestalt:

$$\begin{array}{lcl} z_0 & = & 3.20a_{00}a_{01}a_{02}a_{03}a_{04}\ \ldots \\ z_1 & = & 3.20a_{10}a_{11}a_{12}a_{13}a_{14}\ \ldots \\ z_2 & = & 3.20a_{20}a_{21}a_{22}a_{23}a_{24}\ \ldots \\ z_3 & = & 3.20a_{30}a_{31}a_{32}a_{33}a_{34}\ \ldots \\ \vdots & = & \vdots \end{array}$$

Wir definieren nun die Zahl $\hat{z} = 3.20b_0b_1b_2\ldots$ (d.h.: die ersten 2 Nachkommastellen sind 20 und die $(j+3)$-te Nachkommastelle ist b_j für alle $j \in \mathbb{N}$) mit

$$b_j = \begin{cases} 0 & \text{falls } a_{jj} > 0 \\ 1 & \text{falls } a_{jj} = 0 \end{cases}$$

Wir zeigen nun, dass diese Zahl nicht in der Aufzählung $z_0, z_1, z_2, \ldots$ enthalten ist. Denn angenommen $\hat{z}$ ist in der Aufzählung enthalten, dann muss $\hat{z} = z_k$ für ein $k \in \mathbb{N}$ gelten. Aber laut Definition von $\hat{z}$ hat $\hat{z}$ an der $(k + 3)$-ten Stelle den Wert $b_k = 0$, falls z_k hier einen Wert > 0 hat bzw. den Wert $b_k = 1$, falls z_k hier den Wert 0 hat. Daraus folgt, dass die Aufzählung $z_0, z_1, z_2, \ldots$ *nicht* alle Zahlen aus dem Intervall $[3.2, 3.21)$ enthält. Widerspruch! Das heißt, unsere ursprüngliche Annahme, dass die reellen Zahlen im Intervall $[3.2, 3.21)$ abzählbar sind, war falsch. Die reellen Zahlen im Intervall $[3.2, 3.21)$ sind also überabzählbar.

Aufgabe 1.3
In der Vorlesung wurde die Goldbachsche Vermutung (die bis heute unbewiesen ist) vorgestellt. Eng verwandt mit der Goldbachschen Vermutung ist folgende Vermutung für ungerade Zahlen:
Jede ungerade Zahl $n \geq 7$ ist die Summe von 3 Primzahlen.

Bemerkung: Es ist leicht zu überprüfen, dass diese Vermutung unmittelbar aus der Goldbachschen Vermutung folgt: Jede ungerade Zahl $n \geq 7$ lässt sich in der Form $n = m + 3$ darstellen, wobei m eine gerade Zahl ≥ 4 ist. Die Goldbachsche Vermutung besagt, dass sich m in der Form $m = p_1 + p_2$ für 2 Primzahlen p_1, p_2 darstellen lässt. Also gilt $n = p_1 + p_2 + p_3$ mit $p_3 = 3$, d.h.: n ist die Summe von 3 Primzahlen.

Zeigen Sie, dass sich die Vermutung für ungerade Zahlen $n \geq 7$ *prinzipiell* (d.h. ohne Berücksichtigung von Komplexitätsüberlegungen) leicht beweisen bzw. widerlegen ließe, wenn das Halteproblem von SIMPLE entscheidbar wäre. Sie dürfen dabei die Existenz einer library function "isPrime($\cdot$)" annehmen, die bei einem Aufruf der Art "isPrime(n)" testet, ob n eine Primzahl ist und entsprechend den Wert true bzw. false liefert.

Lösung 1.3

Wir können in SIMPLE folgendes Programm schreiben, das alle ungeraden Zahlen $n \geq 7$ durchläuft und überprüft, ob sich n als Summe von drei Primzahlen darstellen lässt.

```
Boolean test(Integer n)    /* checks if n is the sum of three primes */
    for all i ≤ n, j ≤ n, k ≤ n do {
        if (isPrime(i) and isPrime(j) and isPrime(k) and i + j + k = n) then return true; }
    return false;

Void testConjecture()
    n:=7;
    while test(n)=true do { n := n + 2; }
```

Offensichtlich gilt: Die Vermutung für ungerade Zahlen $n \geq 7$ gilt $\Leftrightarrow$ testConjecture() terminiert nicht. Das heißt: wenn wir ein Programm Π_h hätten, das das Halteproblem von SIMPLE löst, könnten wir Π_h mit dem Quellcode von testConjecture() (inklusive der Unterprozedur test()) und dem leeren String (als Input für testConjecture()) aufrufen. Wenn Π_h den Wert true liefert (d.h.: testConjecture() terminiert), wissen wir, dass die Vermutung für ungerade Zahlen $n \geq 7$ *nicht* gilt; falls Π_h den Wert false liefert, ist die Vermutung für ungerade Zahlen $n \geq 7$ wahr.

Aufgabe 1.4

Für $k \geq 1$, sei N_k die Menge aller durch k teilbaren natürlichen Zahlen, d.h.: $N_k = \{0, k, 2k, 3k, \dots\}$. Zeigen Sie folgende Abzählbarkeitseigenschaften:

a) Die Mengen N_3 der durch 3 teilbaren Zahlen und N_5 der durch 5 teilbaren Zahlen und die Menge $\mathbb{N}$ der natürlichen Zahlen sind alle "gleich groß", d.h.: es gibt jeweils bijektive Abbildungen zwischen diesen Mengen.

b) Auch die Menge $(N_3)^3$ der Zahlentripel über den durch 3 teilbaren Zahlen ist gleich groß, d.h.: sie ist abzählbar unendlich. Um diese Behauptung zu beweisen, sollen Sie eine Aufzählung der Elemente von N_3^3 skizzieren. Zwecks Nachvollziehbarkeit der Aufzählung, sollten Sie bei dieser Aufzählung die Zahlentripel in Gruppen gliedern. Geben Sie für jede Gruppe auch die Anzahl der Elemente in dieser Gruppe sowie die Positionen, die die Elemente der Gruppe in der Aufzählung einnehmen, an!

Lösung 1.4

a) Folgende Abbildungen sind offensichtlich bijektiv:

   – $f \colon \mathbb{N} \to N_3$ mit $f(n) = 3n$ für beliebiges $n \in \mathbb{N}$
     und Umkehrfunktion $f^{-1} \colon N_3 \to \mathbb{N}$ mit $f^{-1}(n) = n/3$ für beliebiges $n \in N_3$.

   – $g : \mathbb{N} \to N_5$ mit $g(n) = 5n$ für beliebiges $n \in \mathbb{N}$
     und Umkehrfunktion $g^{-1} \colon N_5 \to \mathbb{N}$ mit $g^{-1}(n) = n/5$ für beliebiges $n \in N_5$.

   – $h : N_3 \to N_5$ mit $h(n) = (n/3) * 5$ für beliebiges $n \in N_3$
     und Umkehrfunktion $h^{-1} \colon N_5 \to N_3$ mit $h^{-1}(n) = (n/5) * 3$ für beliebiges $n \in N_5$.

   Bemerkung: es würde reichen, 2 dieser bijektiven Abbildungen anzugeben. Daraus folgt unmittelbar die Existenz der dritten, z.B.: $h(\cdot) = g(f^{-1}(\cdot))$.

b) Eine Möglichkeit, die Tripel $(a, b, c) \in (N_3)^3$ aufzuzuzählen, ist, dass wir

   – die Tripel (a, b, c) mit $a, b, c \leq n$ für aufsteigendes $n \in N_3 = \{0, 3, 6, 9, \dots\}$ aufzählen

   – und innerhalb der Tripel, die noch nicht vorher vorgekommen sind, in lexikographischer Reihenfolge vorgehen.

Tripel	Anzahl	Positionen
(0,0,0)	1	0
(0,0,3), (0,3,0), (0,3,3), (3,0,0), (3,0,3), (3,3,0), (3,3,3)	7 ($= 2^3 - 1$)	1, ..., 7
(0,0,6), (0,3,6), (0,6,0), (0,6,3), (0,6,6), (3,0,6), (3,3,6), (3,6,0), (3,6,3), (3,6,6), (6,0,0), (6,0,3), (6,0,6), ..., (6,6,6)	19 ($= 3^3 - 2^3$)	8, ..., 26
(0,0,9), (0,3,9), (0,6,9), (0,9,0), (0,9,3),(0,9,6),(0,9,9), ..., (9,9,0), (9,9,3),(9,9,6),(9,9,9)	37 ($= 4^3 - 3^3$)	27, ..., 63
etc.		

Bemerkung: In dieser Aufgabe wurde verlangt, dass Sie die Abzählbarkeit von $(N_3)^3$ durch Angabe einer Aufzählung der Tripel in $(N_3)^3$ beweisen. Wenn nur "irgendein" Beweis verlangt worden wäre, könnten Sie auch folgendermaßen argumentieren:

- In der Vorlesung wurde gezeigt, dass folgende Eigenschaft gilt: für jedes beliebige (endliche) Alphabet Σ ist die Menge Σ^* abzählbar.
- Jede natürliche Zahl lässt sich als endlicher String über dem Alphabet der erlaubten Ziffern darstellen. Im dekadischen System haben wir also das Alphabet $\Sigma = \{0, \ldots, 9\}$. (Sie können natürlich auch die Binärdarstellung mit Alphabet $\Sigma = \{0, 1\}$ oder irgendeine andere Darstellung für die natürlichen Zahlen mit entsprechendem Alphabet Σ wählen).
- Jedes Zahlentripel aus $(N_3)^3$ lässt sich dann als endlicher String über dem erweiterten Alphabet $\Sigma' = \{0, \ldots, 9, \text{"("}, \text{","}, \text{")"}\}$ darstellen.
- Es gibt also nur abzählbar viele endliche Strings über dem Alphabet Σ' und somit auch nur abzählbar viele Zahlentripel in $(N_3)^3$.

Aufgabe 1.5
Wir betrachten folgende Variante des KORREKTHEIT Problems:

ENDLICHE-KORREKTHEIT
Instanz: (Quellcode von einem SIMPLE) Programm Π, Strings I_1 und I_2, Integer $n \geq 1$.
Frage: Terminiert das Programm Π auf dem Input I_1 innerhalb von n Programmschritten und liefert den Output I_2?

Als "Programmschritt" könnte z.B. die Abarbeitung eines Maschinenbefehls des übersetzten Programms genommen werden. Zeigen Sie, dass dieses Problem entscheidbar ist, indem Sie dafür eine Entscheidungsprozedur *skizzieren.* Sie dürfen davon ausgehen, dass Ihnen ein Interpreter für SIMPLE Programme zur Verfügung steht, der einen Programmschritt nach dem anderen ausführen kann. Geben Sie auch eine kurze Begründung, dass es sich dabei tatsächlich um eine Entscheidungsprozedur für dieses Problem handelt, d.h.: überprüfen Sie, dass sich die von Ihnen skizzierte Entscheidungsprozedur sowohl auf positiven Instanzen als auch auf negativen korrekt verhält

Es sind also folgende Teilaufgaben zu lösen:

a) Skizzierung einer Entscheidungsprozedur für das ENDLICHE-KORREKTHEIT Problem.

b) Begründung für die Korrektheit der von Ihnen skizzierten Entscheidungsprozedur auf positiven Instanzen des ENDLICHE-KORREKTHEIT Problems.

c) Begründung für die Korrektheit der von Ihnen skizzierten Entscheidungsprozedur auf negativen Instanzen des ENDLICHE-KORREKTHEIT Problems.

Bemerkung: Die Betonung bei dieser Frage liegt auf "skizzieren". Es wird nur die Skizze eines Lösungsverfahrens verlangt – im Stil der Semi-Entscheidungsprozeduren für das Halteproblem und das Korrektheit Problem auf den Folien 31/32 der Vorlesung. Was sind die wesentlichen Schritte des Lösungsverfahrens (in diesem Fall: einer Entscheidungsprozedur) als bullet list? Beim Halteproblem war nicht mehr als eine Semi-Entscheidungsprozedur möglich. In Aufgabe 1.5 ist eine Entscheidungsprozedur möglich, weil die Anzahl der Programmschritte durch n beschränkt ist.

Lösung 1.5

a) *Skizzierung einer Entscheidungsprozedur*:
Mit Hilfe eines Interpreters für SIMPLE Programme können wir ein Programm Π_i mit folgenden Eigenschaften schreiben:

– Π_i nimmt als Input eine beliebige Instanz des ENDLICHE-KORREKTHEIT Problems, d.h.: Quellcode von einem SIMPLE Programm Π, Input String I_1 und Output String I_2 für Π und Integer $n \geq 1$;

– Π_i analysiert Π und simuliert die Ausführung von Π auf dem Input I_1;

– Π_i führt einen Zähler k, der mit 0 initialisiert wird und nach jedem Programmschritt von Π um 1 erhöht wird.

– Wenn Π ein Ergebnis (= den Output) liefert, wird dieses von Π_i abgespeichert.

– Π_i terminiert, wenn die Simulation von Π auf dem Input I_1 terminiert oder wenn der Zähler k den Wert n erreicht:

(1) Falls Π_i terminiert, weil die Simulation von Π auf dem Input I_1 terminiert, wird der von Π gelieferte (und von Π_i abgespeicherte) Output überprüft. Falls der Output I_2 ist, dann gibt Π_i den Wert true aus und terminiert selbst; bei einem anderen Output gibt Π_i den Wert false aus und terminiert selbst.

(2) Falls Π_i terminiert, weil der Zähler k den Wert n erreicht, ohne dass die Simulation von Π auf dem Input I_1 terminiert, dann gibt Π_i den Wert false aus und terminiert selbst.

Korrektheit der Entscheidungsprozedur: Wir argumentieren, dass das oben skizzierte Programm Π_i eine Entscheidungsprozedur für das ENDLICHE-KORREKTHEIT Problem ist, indem wir das Verhalten von Π_i auf positiven bzw. auf negativen Instanzen überprüfen:

b) Angenommen das Programm Π_i wird mit einer positiven Instanz (Π, I_1, I_2, n) des ENDLICHE-KORREKTHEIT Problems aufgerufen, d.h.: das Programm Π terminiert auf dem Input I_1 innerhalb von n Programmschritten und liefert den Output I_2. Π_i arbeitet in der Simulation von Π einen Programmschritt von Π auf dem Input I_1 nach dem anderen ab und wird daher (nach höchstens n Programmschritten) feststellen, dass Π terminiert. Außerdem speichert Π_i den vom Programm Π bei Input I_1 erzeugten Output ab. Laut Annahme ist (Π, I_1, I_2, n) eine positive Instanz des ENDLICHE-KORREKTHEIT Problems, d.h.: der von Π bei Input I_1 erzeugte Output ist I_2. Laut Fall (1) der obigen Beschreibung gibt Π_i in diesem Fall den Wert true aus und terminiert selbst. Das ist das korrekte Verhalten für eine Entscheidungsprozedur auf einer positiven Instanz.

c) Angenommen das Programm Π_i wird mit einer negativen Instanz (Π, I_1, I_2, n) des ENDLICHE-KORREKTHEIT Problems aufgerufen. Dann unterscheiden wir wieder 2 Fälle:

i) Angenommen, das Programm Π terminiert auf dem Input I_1 nicht innerhalb von n Schritten. Π_i arbeitet in der Simulation von Π einen Programmschritt von Π auf dem Input I_1 nach dem anderen ab und zählt dabei die Programmschritte im Zähler k mit. Wenn $k = n$ gilt (d.h.: der n-te Programmschritt von Π auf Input I_1 wurde gerade ausgeführt) und Π bis dahin nicht terminiert, dann gibt Π_i laut Fall (2) der obigen Beschreibung den Wert false aus und terminiert.

ii) Angenommen, das Programm Π terminiert auf dem Input I_1 innerhalb von n Schritten aber der von Π produzierte Output stimmt nicht mit I_2 überein. Π_i arbeitet in der Simulation von Π einen Programmschritt von Π auf dem Input I_1 nach dem anderen ab und zählt dabei die Programmschritte im Zähler k mit. Außerdem speichert Π_i jeglichen von Π produzierten Output ab. Da Π innerhalb von n Schritten terminiert, wird die Ausführung von Π auf dem Inptut I_1 von Π_i zur Gänze simuliert. Laut Fall (1) der obigen Beschreibung analysiert Programm Π den abgespeicherten Output von Π und stellt fest, dass der tatsächliche Output vom "gewünschten" Output String I_2 abweicht. Das Programm Π_i gibt in diesem Fall den Wert false aus und terminiert.

In beiden Fällen i) und ii) ist das das korrekte Verhalten für eine Entscheidungsprozedur auf einer negativen Instanz.

Aufgabe 1.6
In der Vorlesung wurde das ERREICHBARER-CODE Problem vorgestellt und intuitiv erklärt, dass das Problem unentscheidbar ist. Wir definieren eine leicht unterschiedliche Version von ERREICHBARER-CODE (die aber für die Unentscheidbarkeit völlig unerheblich ist):

ERREICHBARER-CODE
Instanz: (Quellcode von einem SIMPLE) Programm Π, und ein Label (= String) L.
Frage: Gibt es einen Input I für das Programm Π, so dass Π bei Ausführung mit Input I den Code auf der Programmzeile mit dem Label L ausführt?

Außerdem wurde in der Vorlesung intuitiv argumentiert, dass dieses Problem unentscheidbar ist.

Beweisen Sie die Unentscheidbarkeit von ERREICHBARER-CODE mittels Reduktion vom HALTEPROBLEM, d.h.: definieren Sie eine Reduktion vom HALTEPROBLEM auf das ERREICHBARER-CODE Problem und beweisen Sie die Korrektheit der Reduktion.

Es sind also folgende Teilaufgaben zu lösen:

a) Reduktion vom HALTEPROBLEM auf das ERREICHBARER-CODE Problem.

b) "$\Rightarrow$"-Richtung des Korrektheitsbeweises: (Π, I) ist eine positive Instanz des HALTEPROBLEMS $\Rightarrow (\Pi', L)$ ist eine positive Instanz des ERREICHBARER-CODE Problems.

c) "$\Leftarrow$"-Richtung des Korrektheitsbeweises: (Π', L) ist eine positive Instanz des ERREICHBARER-CODE Problem $\Rightarrow (\Pi, I)$ ist eine positive Instanz des HALTEPROBLEMS.

Bemerkung: Erinnern Sie sich an die Vorlesung, wie wir bei (Many-One) Reduktionen von einem Problem A auf ein Problem B vorgegangen sind: Wir starten mit einer beliebigen Instanz von Problem A und definieren dazu eine entsprechende Instanz von Problem B. In diesem Beispiel starten wir also mit einer beliebigen Instanz (Π, I) des HALTEPROBLEMS und definieren eine entsprechende Instanz (Π', L) des ERREICHBARER-CODE Problems. Auf keinen Fall sollen Sie versuchen, ein Programm zu schreiben, das eines der beiden Probleme "löst".

Lösung 1.6

a) *Reduktion:* Sei (Π, I) eine beliebige Instanz des HALTEPROBLEMS. Wir definieren daraus eine Instanz (Π', L) von ERREICHBARER-CODE, indem wir für L einen beliebigen String wählen, z.B.: $L =$ "abcd" und Π' folgendermaßen definieren:

Program Π' (String S)
    call $\Pi(I)$;
    abcd: return;

d.h.: Π' ignoriert den Input String und ruft sofort das Programm Π mit dem Input I auf. Wenn die Kontrolle vom Aufruf von Π zu Π' zurückkommt, dann führt Π' nur noch das return-statment aus, sprich: es wird beendet. Dieses return-Statement hat das Label "abcd".

Korrektheit der Reduktion: Wir müssen zeigen, dass (Π, I) eine positive Instanz des *Halteproblems* ist $\Leftrightarrow$ (Π', L) laut obiger Definition ist eine positive Instanz des ERREICHBARER-CODE Problems.
Wir zeigen die zwei Richtungen der Äquivalenz getrennt:

b) "$\Rightarrow$": Angenommen (Π, I) ist eine positive Instanz des *Halteproblems*. Das heißt, bei Aufruf von Π mit dem Input I terminiert Π. Für das Programm Π' bedeutet das, das nach dem Aufruf von Π mit dem Input I irgendwann die Kontrolle wieder zu Π' zurückkommt – und zwar egal mit welchem Input String Π' aufgerufen wurde. Nach der Rückkehr vom Aufruf von Π führt Π' die Programmzeile mit dem Label $L =$ "abcd" aus. Zusammengefasst heißt das: Es gibt einen Input S (z.B. "xyz") für das Programm Π', so dass Π' bei Ausführung mit Input S den Code auf der Programmzeile mit dem Label L ausführt. Also ist (Π', L) eine positive Instanz des ERREICHBARER-CODE Problems.

c) "$\Leftarrow$": Angenommen (Π', L) ist eine positive Instanz des ERREICHBARER-CODE Problems. Das heißt: Es gibt einen Input S für das Programm Π', so dass Π' bei Ausführung mit Input S den Code auf der Programmzeile mit dem Label L ausführt. Laut obiger Definition von Π' ruft Π' unabhängig vom konkreten Wert von S sofort das Programm Π mit dem Input I auf. Laut Annahme ist (Π', L) eine positive Instanz, d.h.: die Programmzeile mit Label $L =$ "abcd" wird irgendwann ausgeführt. Das geht aber nur, wenn die Kontrolle vom Aufruf von Π zu Π' zurückkommt. Das bedeutet, dass Π auf dem Input I terminert. Das heißt: (Π, I) ist eine positive Instanz des *Halteproblems*.

Aufgabe 1.7
Wir definieren folgendes Problem als leichte Abänderung des KORREKTHEIT Problems:

MINDESTENS-EINS-VON-ZWEI-KORREKTHEIT
Instanz: (Quellcode der) Programme Π_1, Π_2, Strings I_1, I_2.
Frage: Terminiert *mindestens* eines der Programme Π_1 und Π_2 auf dem Input I_1 und liefert es als Output I_2?

Zeigen Sie, dass dieses Problem semi-entscheidbar ist, indem Sie eine Semi-Entscheidungsprozedur für das MINDESTENS-EINS-VON-ZWEI-KORREKTHEIT Problem *skizzieren.* Geben Sie auch eine kurze Begründung, dass es sich dabei tatsächlich um eine Semi-Entscheidungsprozedur für dieses Problem handelt, d.h.: überprüfen Sie, dass sich die von Ihnen skizzierte Semi-Entscheidungsprozedur sowohl auf positiven Instanzen als auch auf negativen Instanzen korrekt verhält.

Es sind also folgende Teilaufgaben zu lösen:

a) Skizzierung einer Semi-Entscheidungsprozedur für das MINDESTENS-EINS-VON-ZWEI-KORREKTHEIT Problem.

b) Begründung für die Korrektheit der von Ihnen skizzierten Semi-Entscheidungsprozedur auf positiven Instanzen des MINDESTENS-EINS-VON-ZWEI-KORREKTHEIT Problems.

c) Begründung für die Korrektheit der von Ihnen skizzierten Semi-Entscheidungsprozedur auf negativen Instanzen des MINDESTENS-EINS-VON-ZWEI-KORREKTHEIT Problems.

Bemerkung: Wie schon in Aufgabe 1.5 liegt auch bei dieser Frage die Betonung auf "skizzieren". Es wird nur die Skizze einer Semi-Entscheidungsprozedur verlangt – im Stil der Semi-Entscheidungsprozeduren für das Halteproblem und das Korrektheit Problem auf den Folien 31/32 der Vorlesung. Was sind die wesentlichen Schritte des Lösungsverfahrens als bullet list?

Lösung 1.7

a) *Skizzierung einer Semi-Entscheidungsprozedur*:
Mit Hilfe eines Interpreters für SIMPLE Programme können wir ein Programm Π_i mit folgenden Eigenschaften schreiben:
   - Π_i nimmt als Input eine beliebige Instanz des MINDESTENS-EINS-VON-ZWEI-KORREKTHEIT Problems, d.h.: Quellcode von 2 SIMPLE Programmen Π_1, Π_2 und Strings I_1, I_2.
   - Π_i simuliert abwechselnd einen Schritt von Π_1 und von Π_2 auf dem Input I_1.
   - Wenn die Simulation eines der Programme Π_1 oder Π_2 terminiert, dann überprüft Π_i, ob das Programm, das gerade terminiert, den Output I_2 geliefert hat.
     - Wenn ja, dann stoppt Π_i die Simulation beider Programme und gibt true zurück.
     - Wenn nein, dann fährt Π_i mit der Simulation des anderen Programms fort.
   - Wenn die Simulation des anderen Programms terminiert, dann überprüft Π_i, ob das simulierte Programm den Output I_2 geliefert hat.
     - Wenn ja, dann stoppt Π_i die Simulation des Programmes und gibt true zurück.
     - Wenn nein, dann stoppt Π_i ebenfalls die Simulation des Programmes und gibt false zurück.

Korrektheit der Semi-Entscheidungsprozedur: Wir argumentieren, dass das oben skizzierte Programm Π_i eine Semi-Entscheidungsprozedur für das MINDESTENS-EINS-VON-ZWEI-KORREKTHEIT Problem ist, indem wir das Verhalten von Π_i auf positiven bzw. auf negativen Instanzen überprüfen:

b) Wir betrachten zuerst den Fall, dass das Programm Π_i wird mit einer positiven Instanz (Π_1, Π_2, I_1, I_2) des MINDESTENS-EINS-VON-ZWEI-KORREKTHEIT Problems aufgerufen wird, d.h.: zumindest eines der beiden Programme Π_1, Π_2 terminiert auf dem Input I_1 und liefert den korrekten Output I_2. Wir unterscheiden folgende Fälle:
   - Angenommen, das Programm, das den korrekten Output I_2 liefert, terminiert als erstes. Dann wird auch die abwechselnde Simulation von Programmschritten der beiden Programme Π_1, Π_2 auf dem Input I_1 irgendwann den Punkt erreichen, an dem dieses Programm mit dem korrekten Output I_2 terminiert. Laut obiger Beschreibung stoppt dann Π_i die Simulation beider Programme, gibt true aus und terminiert selbst.

- Angenommen es terminiert zuerst das andere Programm und liefert ebenfalls den korrekten Output I_2 zurück. Dann verhält sich Π_i genau wie im vorigen Fall, d.h.: es stoppt die Simulation beider Programme, gibt true aus und terminiert selbst.
- Angenommen es terminiert zuerst das andere Programm und liefert einen Output ungleich I_2 zurück. Laut obiger Beschreibung setzt dann Π_i die Simulation des korrekten Programms fort. Das korrekte Programm wird irgendwann terminieren und den Output I_2 liefern. Laut obiger Beschreibung wird dann Π_i den Wert true liefern und selbst terminieren.

In allen Fällen ist das das korrekte Verhalten für eine Semi-Entscheidungsprozedur auf einer positiven Instanz.

c) Wir betrachten nun den Fall, dass das Programm Π_i mit einer negativen Instanz (Π_1, Π_2, I_1, I_2) des Mindestens-Eins-von-Zwei-Korrektheit Problems aufgerufen wird. Dazu unterscheiden wir folgende Fälle;

- Angenommen, keines der beiden Programme Π_1, Π_2 terminiert auf dem Input I_1. Dann wird auch die abwechselnde Simulation von Programmschritten der beiden Programme Π_1, Π_2 auf dem Input I_1 nicht terminieren. Das Programm Π_i läuft in diesem Fall endlos.
- Angenommen, genau eines der beiden Programme Π_1, Π_2 terminiert auf dem Input I_1, liefert aber einen Output ungleich I_2, während das andere Programm nicht terminiert. Dann wird auch die abwechselnde Simulation von Programmschritten der beiden Programme Π_1, Π_2 auf dem Input I_1 irgendwann den Punkt erreichen, an dem das eine Programm terminiert. Da es einen Output ungleich I_2 liefert, wird laut obiger Beschreibung die Simulation des anderen Programms fortgesetzt. Da das andere Programm nicht terminiert, wird auch die Simulation durch Π_i nicht terminieren. Π_i läuft in diesem Fall endlos.
- Angenommen, beide Programme Π_1, Π_2 terminieren auf dem Input I_1, liefern aber einen Output ungleich I_2. Dann wird auch die abwechselnde Simulation von Programmschritten der beiden Programme Π_1, Π_2 auf dem Input I_1 irgendwann den Punkt erreichen, an dem das erste der beiden Programme terminiert. Da es einen Output ungleich I_2 liefert, wird laut obiger Beschreibung die Simulation des anderen Programms fortgesetzt. Da auch das andere Programm terminiert, wird auch seine Simulation durch Π_i irgendwann terminieren. Da es einen Output ungleich I_2 liefert, wird laut obiger Beschreibung das Programm Π_i den Wert false liefern und selbst terminieren.

In allen Fällen ist das ein korrektes Verhalten für eine Semi-Entscheidungsprozedur auf einer negativen Instanz.

Aufgabe 1.8
Wir betrachten nun folgendes Problem:

Eins-von-Zwei-Korrektheit
Instanz: Programme Π_1, Π_2 und Strings I_1, I_2.
Frage: Verhält sich *genau* eines der Programme Π_1 und Π_2 auf Input I_1 korrekt?
d.h.: es hält und liefert den korrekten Output I_2 (während sich das andere Programm inkorrekt verhält, d.h.: entweder es liefert einen Output ungleich I_2 oder es terminiert nicht).

Zeigen Sie mittels Reduktion vom co-Halteproblem, dass das Eins-von-Zwei-Korrektheit nicht semi-entscheidbar ist. Geben Sie auch einen Beweis für die Korrektheit Ihrer Reduktion.

Es sind also folgende Teilaufgaben zu lösen:

a) Reduktion vom co-Halteproblem auf das Eins-von-Zwei-Korrektheit Problem.

b) "$\Rightarrow$"-Richtung des Korrektheitsbeweises: (Π, I) ist eine positive Instanz des co-Halteproblems $\Rightarrow$ (Π_1, Π_2, I_1, I_2) ist eine positive Instanz des Eins-von-Zwei-Korrektheit Problems.

c) "$\Leftarrow$"-Richtung des Korrektheitsbeweises: (Π_1, Π_2, I_1, I_2) ist eine positive Instanz des Eins-von-Zwei-Korrektheit Problems $\Rightarrow$ (Π, I) ist eine positive Instanz des co-Halteproblems.

Tipp: Definieren Sie eines der Programme (sagen wir Π_1) so, dass es sicher korrekt ist. Und das andere Programm sollte einen Aufruf der Instanz des co-Halteproblems (d.h.: (Π, I)) enthalten und ebenfalls das korrekte Ergebnis liefern, falls Π auf I terminiert, d.h.: damit wir eine positive Instanz von EINS-VON-ZWEI-KORREKTHEIT erhalten, darf die Kontrolle vom Aufruf von Π mit Input I nie an das aufrufende Programm zurückkommen. Mit anderen Worten, (Π, I) muss eine positive Instanz des co-HALTEPROBLEMS sein.

Lösung 1.8

a) *Reduktion:* Sei (Π, I) eine beliebige Instanz des co-HALTEPROBLEMS. Wir definieren daraus folgende Instanz (Π_1, Π_2, I_1, I_2) des EINS-VON-ZWEI-KORREKTHEIT Problems mit $I_1 = I$ und I_2 beliebig, z.B. $I_2 =$ "123". Außerdem definieren wir Π_1 und Π_2 wie folgt:

```
Program Π1 (String S)
    return "123";

Program Π2 (String S)
    call Π(S);
    return "123";
```

Korrektheit der Reduktion: Wir müssen zeigen, dass (Π, I) eine positive Instanz des co-HALTEPROBLEMS ist (d.h.: Π terminiert nicht auf Input I) $\Leftrightarrow$ (Π_1, Π_2, I_1, I_2) ist eine positive Instanz des EINS-VON-ZWEI-KORREKTHEIT Problems (d.h.: genau eines der beiden Programme Π_1, Π_2 terminiert und liefert den korrekten Output I_2 auf dem Input I_1).

Wir zeigen die zwei Richtungen der Äquivalenz getrennt:

b) "$\Rightarrow$" Angenommen (Π, I) ist eine positive Instanz des co-HALTEPROBLEMS, d.h.: Π terminiert nicht auf Input I. Dann terminiert auch Π_2 auf dem Input $I_1 = I$ nicht, da vom Aufruf von Π mit dem Input I nie die Kontrolle zum aufrufenden Programm Π_2 zurückkommt. Das heißt, dass Π_2 auf dem Input I_1 nicht korrekt ist. Anderseits ist Π_1 offensichtlich korrekt, da es unabhängig vom konkreten Wert von I_1 terminiert und den gewünschten Wert $I_2 =$ "123" zurückliefert. Somit ist (Π_1, Π_2, I_1, I_2) eine positive Instanz des EINS-VON-ZWEI-KORREKTHEIT Problems

c) "$\Leftarrow$" Angenommen (Π_1, Π_2, I_1, I_2) eine positive Instanz des EINS-VON-ZWEI-KORREKTHEIT Problems, d.h.: genau eines der Programme Π'_1, Π'_2 terminiert und liefert den korrekten Output auf dem Input I_1. Das Programm Π_1 ist offensichtlich korrekt, da es unabhängig vom konkreten Wert von I_1 terminiert und den gewünschten Wert $I_2 =$ "123" zurückliefert. Laut Annahme ist (Π_1, Π_2, I_1, I_2) eine positive Instanz des EINS-VON-ZWEI-KORREKTHEIT Problems. Da Π_1 korrekt ist, muss also Π_2 beim Aufruf mit $I_1 = I$ inkorrekt sein. Der Aufruf Π_2 mit Input $I_1 = I$ führt sofort zu einem Aufruf von Π mit dem Input String I. Falls Π auf I terminiert, würde Π_2 die Kontrolle zurückbekommen und den Output "123" liefern und somit ebenfalls korrekt sein. Damit Π_2 auf dem Input I_2 inkorrekt ist, darf also Π auf dem Input I nicht terminieren. Das heißt, dass (Π, I) eine positive Instanz des co-HALTEPROBLEMS ist.

Aufgabe 1.9

Wir betrachten noch einmal das EINS-VON-ZWEI-KORREKTHEIT Problem von der vorigen Frage.

Zeigen Sie mittels Reduktion vom co-HALTEPROBLEM, dass auch das co-EINS-VON-ZWEI-KORREKTHEIT Problem nicht semi-entscheidbar ist. Geben Sie auch einen Beweis für die Korrektheit Ihrer Reduktion.

Bemerkung: Wie in der Vorlesung argumentiert wurde, gilt für 2 beliebige Probleme $\mathcal{P}_1$ und $\mathcal{P}_2$:

$$\mathcal{P}_1 \leq \mathcal{P}_2 \Leftrightarrow \text{co-}\mathcal{P}_1 \leq \text{co-}\mathcal{P}_2$$

Sie können diese Aufgabe also lösen, indem Sie das HALTEPROBLEM auf das EINS-VON-ZWEI-KORREKTHEIT Problem reduzieren.

Es sind also folgende Teilaufgaben zu lösen:

a) Reduktion vom HALTEPROBLEM auf das EINS-VON-ZWEI-KORREKTHEIT Problem.

b) "$\Rightarrow$"-Richtung des Korrektheitsbeweises: (Π, I) ist eine positive Instanz des HALTEPROBLEMS $\Rightarrow$ (Π_1, Π_2, I_1, I_2) ist eine positive Instanz des EINS-VON-ZWEI-KORREKTHEIT Problems.

c) "$\Leftarrow$"-Richtung des Korrektheitsbeweises: (Π_1, Π_2, I_1, I_2) ist eine positive Instanz des EINS-VON-ZWEI-KORREKTHEIT Problem $\Rightarrow$ (Π, I) ist eine positive Instanz des HALTEPROBLEMS.

Tipp: Definieren Sie eines der Programme (sagen wir Π_1) so, dass es sicher inkorrekt ist. Und das andere Programm sollte einen Aufruf der Instanz des HALTEPROBLEMs (d.h.: (Π, I)) enthalten und das korrekte Ergebnis liefern, falls Π auf I terminiert, d.h.: damit wir eine positive Instanz von EINS-VON-ZWEI-KORREKTHEIT erhalten, muss also die Kontrolle vom Aufruf von Π mit Input I an das aufrufende Programm zurückkommen. Mit anderen Worten, (Π, I) muss eine positive Instanz des HALTEPROBLEMs sein.

Lösung 1.9

a) *Reduktion:* Sei (Π, I) eine beliebige Instanz des HALTEPROBLEMS. Wir definieren daraus folgende Instanz (Π_1, Π_2, I_1, I_2) des EINS-VON-ZWEI-KORREKTHEIT Problems mit $I_1 = I$ und I_2 beliebig, z.B.: $I_2 =$ "abc". Außerdem definieren wir Π_1 und Π_2 wie folgt:

Program Π_1 (String S)
  return "xyz";

Program Π_2 (String S)
  call $\Pi(S)$;
  return "abc";

Korrektheit der Reduktion: Wir müssen zeigen, dass (Π, I) eine positive Instanz des HALTEPROBLEMS ist (d.h.: Π terminiert auf Input I) $\Leftrightarrow$ (Π_1, Π_2, I_1, I_2) ist eine positive Instanz des EINS-VON-ZWEI-KORREKTHEIT Problems (d.h.: genau eines der beiden Programme Π_1, Π_2 terminiert und liefert den korrekten Output I_2 auf dem Input I_1).

Wir zeigen die zwei Richtungen der Äquivalenz getrennt:

b) "$\Rightarrow$" Angenommen (Π, I) ist eine positive Instanz des HALTEPROBLEMS, d.h.: Π terminiert auf Input I. Wir betrachten das Verhalten des Programms Π_2: Unabhängig vom konkreten Wert von I_1 ruft Π_2 sofort das Programm Π mit dem Input $I_1 = I$ auf. Laut Annahme terminiert Π auf dem Input I. Also kehrt die Kontrolle zum aufrufenden Progreamm Π_2 zurück. Dieses liefert dann den Output "abc" und terminiert. Das heißt, Π_2' ist bei Input I_1 korrekt. Anderseits liefert Π_1 unabhängig vom Input den Output "xyz". Das heißt, Π_1 ist bei Input I_1 inkorrekt. Das bedeutet, dass (Π_1, Π_2, I_1, I_2) eine positive Instanz des EINS-VON-ZWEI-KORREKTHEIT Problems ist.

c) Angenommen (Π_1, Π_2, I_1, I_2) ist eine positive Instanz des EINS-VON-ZWEI-KORREKTHEIT Problems, d.h.: genau eines der beiden Programme Π_1, Π_2 terminiert und liefert den korrekten Output auf dem Input I_1. Das Programm Π_1 liefert unabhängig vom Input den Output "xyz". Das heißt, Π_1 ist bei Input I_1 inkorrekt. Damit also (Π_1, Π_2, I_1, I_2) eine positive Instanz des EINS-VON-ZWEI-KORREKTHEIT Problems ist, muss Π_2 bei Input I_1 korrekt sein. Wir betrachten das Verhalten des Programms Π_2: Unabhängig vom konkreten Wert von I_1 ruft Π_2 sofort das Programm Π mit dem Input $I_1 = I$ auf. Damit Π_2 das return Statement mit dem Output String "abc" ausführen kann, muss es die Kontrolle vom Aufruf von Π zurückbekommen. Das heißt, dass Π auf dem Input $I = I_1$ terminiert. Das bedeutet, dass (Π, I) eine positive Instanz des HALTEPROBLEMS ist.

Aufgabe 1.10

Wir haben vier Entscheidungsprobleme gegeben: $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$, $\mathcal{P}_4$. Von diesen ist folgendes bekannt:

– $\mathcal{P}_1$ ist entscheidbar (und somit auch co-$\mathcal{P}_1$),

– $\mathcal{P}_3$ ist semi-entscheidbar (während über co-$\mathcal{P}_3$ nichts bekannt ist),

– $\mathcal{P}_4$ ist unentscheidbar (und somit auch co-$\mathcal{P}_4$).

Über das Problem $\mathcal{P}_2$ und sein co-Problem ist nichts bekannt. Welche *zusätzlichen* Aussagen können wir treffen, wenn jeweils folgende (berechenbare) Reduktionen gelten.

a) Angenommen, es gelten die Reduktionen $\mathcal{P}_1 \leq \mathcal{P}_2$ und $\mathcal{P}_2 \leq \mathcal{P}_3$; was können wir dann über die Probleme $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$ sowie co-$\mathcal{P}_1$, co-$\mathcal{P}_2$ und co-$\mathcal{P}_3$ *zusätzlich* zum bereits bekannten Wissen aussagen?

b) Angenommen, es gelten die Reduktionen $\mathcal{P}_3 \leq \mathcal{P}_1$ und $\mathcal{P}_2 \leq \mathcal{P}_1$; was können wir dann über die Probleme $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$ sowie co-$\mathcal{P}_1$, co-$\mathcal{P}_2$ und co-$\mathcal{P}_3$ *zusätzlich* zum bereits bekannten Wissen aussagen?

c) Angenommen, es gelten die Reduktionen $\mathcal{P}_2 \leq \mathcal{P}_3$ und $\mathcal{P}_3 \leq \mathcal{P}_4$; was können wir dann über die Probleme $\mathcal{P}_2$, $\mathcal{P}_3$, $\mathcal{P}_4$ sowie co-$\mathcal{P}_2$, co-$\mathcal{P}_3$ und co-$\mathcal{P}_4$ *zusätzlich* zum bereits bekannten Wissen aussagen?

d) Angenommen, es gelten die Reduktionen $\mathcal{P}_4 \leq \mathcal{P}_3$ und $\mathcal{P}_3 \leq \mathcal{P}_2$; was können wir dann über die Probleme $\mathcal{P}_2$, $\mathcal{P}_3$, $\mathcal{P}_4$ sowie co-$\mathcal{P}_2$, co-$\mathcal{P}_3$ und co-$\mathcal{P}_4$ *zusätzlich* zum bereits bekannten Wissen aussagen?

Lösung 1.10

a) Dann gilt:
   - $\mathcal{P}_2$ ist semi-entscheidbar (wegen $\mathcal{P}_2 \leq \mathcal{P}_3$ und Semi-Entscheidbarkeit von $\mathcal{P}_3$).

b) Dann gilt:
   - $\mathcal{P}_2$ ist entscheidbar (wegen $\mathcal{P}_2 \leq \mathcal{P}_1$ und Entscheidbarkeit von $\mathcal{P}_1$) und somit auch co-$\mathcal{P}_2$;
   - $\mathcal{P}_3$ ist entscheidbar (wegen $\mathcal{P}_3 \leq \mathcal{P}_1$ und Entscheidbarkeit von $\mathcal{P}_1$) und somit auch co-$\mathcal{P}_3$.

c) Dann gilt:
   - $\mathcal{P}_2$ ist semi-entscheidbar (wegen $\mathcal{P}_2 \leq \mathcal{P}_3$ und Semi-Entscheidbarkeit von $\mathcal{P}_3$).

d) Dann gilt:
   - $\mathcal{P}_3$ ist unentscheidbar (wegen $\mathcal{P}_4 \leq \mathcal{P}_3$ und Unentscheidbarkeit von $\mathcal{P}_4$);
   - $\mathcal{P}_2$ ist unentscheidbar (wegen $\mathcal{P}_3 \leq \mathcal{P}_2$ und Unentscheidbarkeit von $\mathcal{P}_3$);
   - co-$\mathcal{P}_3$ ist nicht einmal semi-entscheidbar (wegen Unentscheidbarkeit und Semi-Entscheidbarkeit von $\mathcal{P}_3$);
   - co-$\mathcal{P}_2$ ist nicht einmal semi-entscheidbar (weil $\mathcal{P}_3 \leq \mathcal{P}_2$ äquivalent zu co-$\mathcal{P}_3 \leq$ co-$\mathcal{P}_2$ ist und co-$\mathcal{P}_3$ nicht einmal semi-entscheidbar ist);
   - $\mathcal{P}_4$ ist semi-entscheidbar (wegen $\mathcal{P}_4 \leq \mathcal{P}_3$ und Semi-Entscheidbarkeit von $\mathcal{P}_3$);
   - co-$\mathcal{P}_4$ ist nicht einmal semi-entscheidbar (wegen Unentscheidbarkeit und Semi-Entscheidbarkeit von $\mathcal{P}_4$).