Theoretische Informatik
Übungsblatt 1 (2025W)
Lösungsvorschlag

Allgemeine Hinweise:

- Die Deadline für die Abgabe der Lösungen zum Übungsblatt 1 ist der **26. Oktober 2025**.
- 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 sowie offensichtlich mit ChatGPT (oder ähnlichen Tools) generierte Lösungen werden mit 0 Punkten bewertet. Die Verwendung von Latex wird nachdrücklich empfohlen (ein Latex template wurde auf TUWEL bereitgestellt). 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

Bei vielen Problemen im Operational Research (OR) gibt es den Begriff einer "Lösung". Ein klassisches OR Problem ist das Handelsreisenden Problem (engl. Traveling Salesperson Problem, TSP). Dabei sind gegeben:

- eine Anzahl n von Städten;
- für alle $i, j \in \{1, \ldots, n\}$ mit $i \neq j$: die Entfernung $d_{i,j}$ zwischen der Stadt i und der Stadt j. Die Entfernung soll symmetrisch sein, d.h. es gilt $d_{i,j} = d_{j,i}$ für alle $i \neq j$; außerdem wird üblicherweise angenommen, dass alle $d_{i,j}$ natürliche Zahlen sind;
- eine natürliche Zahl B als obere Schranke für die erlaubte Gesamtlänge einer "Tour" der/des Handelsreisenden: eine Tour beginnt bei irgendeiner Stadt, besucht jede Stadt genau einmal und führt am Ende zum Ausgangspunkt zurück.

Eine Lösung des Handelsreisenden Problems besteht also aus einer Reihenfolge, in der die Städte besucht werden, so dass die Gesamtlänge der Tour B nicht überschreitet. Mathematisch gesprochen ist eine Lösung also eine Permutation $(i_1, \ldots, i_n)$ der Zahlen $\{1, \ldots, n\}$, so dass $d_{i_1,i_2} + d_{i_2,i_3} + \cdots + d_{i_{n-1},i_n} + d_{i_n,i_1} \leq B$.

In seiner ursprünglichen Form ist das Handelsreisenden Problem ein Entscheidungsproblem. Es lassen sich aber auch alle anderen in der Vorlesung vorgestellten Problemtypen rund um das Handelsreisenden Problem formulieren. Formulieren Sie Beispiele für die folgenden Typen von Problemen (Instanz? Frage?) im Zusammenhang mit dem Handelsreisenden Problem.

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: natürliche Zahl n; für alle $i, j \in \{1, \ldots, n\}$ mit $i \neq j$: natürliche Zahlen $d_{i,j}$ mit $d_{i,j} = d_{j,i}$; natürliche Zahl B.
Frage: Gibt es eine Permutation $(i_1, \ldots, i_n)$ der Zahlen $\{1, \ldots, n\}$, so dass $d_{i_1,i_2} + d_{i_2,i_3} + \cdots + d_{i_{n-1},i_n} + d_{i_n,i_1} \leq B$ gilt?

b) Aufzählproblem:
Instanz: natürliche Zahl n; für alle $i, j \in \{1, \ldots, n\}$ mit $i \neq j$: natürliche Zahlen $d_{i,j}$ mit $d_{i,j} = d_{j,i}$; natürliche Zahl B.
Frage: Ausgabe aller Permutationen $(i_1, \ldots, i_n)$ der Zahlen $\{1, \ldots, n\}$, so dass $d_{i_1,i_2} + d_{i_2,i_3} + \cdots + d_{i_{n-1},i_n} + d_{i_n,i_1} \leq B$ gilt?

Bemerkung: In der Praxis wird man 2 Lösungen, die sich nur im Startpunkt unterscheiden aber die Städte in der selben Reihenfolge durchlaufen, als redundant betrachten. Ohne Einschränkung der Allgemeinheit könnte man also verlangen, dass jede Tour bei der Stadt 1 beginnt, d.h. $i_1 = 1$.

c) Zählproblem:
Instanz: natürliche Zahl n; für alle $i, j \in \{1, \ldots, n\}$ mit $i \neq j$: natürliche Zahlen $d_{i,j}$ mit $d_{i,j} = d_{j,i}$; natürliche Zahl B.
Frage: Wie viele Permutationen $(i_1, \ldots, i_n)$ der Zahlen $\{1, \ldots, n\}$ gibt es, so dass $d_{i_1,i_2} + d_{i_2,i_3} + \cdots + d_{i_{n-1},i_n} + d_{i_n,i_1} \leq B$ gilt?

Bemerkung: Wie schon beim obigen Aufzählproblem wird man sich in der Praxis vermutlich nur für die Anzahl der Permutationen mit $i_1 = 1$ interessieren.

d) Optimierungsproblem (in diesem Fall natürlicherweise ein Minimierungsproblem):
Instanz: natürliche Zahl n; für alle $i, j \in \{1, \ldots, n\}$ mit $i \neq j$: natürliche Zahlen $d_{i,j}$ mit $d_{i,j} = d_{j,i}$.
Frage: Was ist die kleinste Zahl D, für die es eine Permutation $(i_1, \ldots, i_n)$ der Zahlen $\{1, \ldots, n\}$ gibt, so dass $D = d_{i_1,i_2} + d_{i_2,i_3} + \cdots + d_{i_{n-1},i_n} + d_{i_n,i_1}$?

e) Suchproblem:
Instanz: natürliche Zahl n; für alle $i, j \in \{1, \ldots, n\}$ mit $i \neq j$: natürliche Zahlen $d_{i,j}$ mit $d_{i,j} = d_{j,i}$; natürliche Zahl B.
Frage: Ausgabe einer Permutation $(i_1, \ldots, i_n)$ der Zahlen $\{1, \ldots, n\}$, so dass $d_{i_1,i_2} + d_{i_2,i_3} + \cdots + d_{i_{n-1},i_n} + d_{i_n,i_1} \leq B$ gilt.

Aufgabe 1.2

Beweisen Sie mittels Diagonalargument, dass das Intervall $[0.5, 0.51)$ überabzählbar viele reelle Zahlen enthält.

Lösung 1.2

Indirekter Beweis: Wir nehmen an, dass die reellen Zahlen im Intervall $[0.5, 0.51)$ abzählbar sind. Das heißt, wir können sie aufzählen als $z_0, z_1, z_2, \ldots$. Alle Zahlen $z_i \in [0.5, 0.51)$ lassen sich als Dezimalzahlen der Form $z_i = 0.50a_{i0}a_{i1}a_{i2}a_{i3}\ldots$ 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:

$$\begin{array}{lcl} z_0 & = & 0.50a_{00}a_{01}a_{02}a_{03}a_{04}\ldots \\ z_1 & = & 0.50a_{10}a_{11}a_{12}a_{13}a_{14}\ldots \\ z_2 & = & 0.50a_{20}a_{21}a_{22}a_{23}a_{24}\ldots \\ z_3 & = & 0.50a_{30}a_{31}a_{32}a_{33}a_{34}\ldots \\ \vdots & = & \vdots \end{array}$$

Wir definieren nun die Zahl $\hat{z} = 0.50b_0b_1b_2\ldots$ (d.h.: die ersten 2 Nachkommastellen sind 50 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 $[0.5, 0.51)$ enthält. Widerspruch! Das heißt, unsere ursprüngliche Annahme, dass die reellen Zahlen im Intervall $[0.5, 0.51)$ abzählbar sind, war falsch. Die reellen Zahlen im Intervall $[0.5, 0.51)$ sind also überabzählbar.

Aufgabe 1.3

In der Vorlesung wurde die Goldbach'sche Vermutung (die bis heute unbewiesen ist) vorgestellt. Eine ähnliche (ebenfalls unbewiesene) Vermutung ist die sogenannte Lemoine'sche Vermutung (in der Literatur auch Levy'sche Vermutung genannt):
Jede ungerade Zahl $n \geq 7$ lässt sich in der Form $n = 2 \cdot p + q$ für 2 (nicht notwendigerweise verschiedene) Primzahlen darstellen.

Beispiele: $13 = 2 \cdot 3 + 7$, $19 = 2 \cdot 3 + 13$, $47 = 2 \cdot 2 + 43$, etc.

Zeigen Sie, dass sich die Vermutung *prinzipiell* (d.h. ohne Berücksichtigung von Komplexitätsüberlegungen) leicht beweisen bzw. widerlegen ließe, wenn das Halteproblem von SIMPLE Programmen entscheidbar wäre. Sie dürfen dabei die Existenz einer library function "isPrime(·)" 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 in der Form $n = 2 \cdot p + q$ für zwei Primzahlen p und q darstellen lässt.

```
Boolean test(Integer n)    /* checks if n = 2p + q for two primes p, q */
   for all i, j with 2 ≤ i, j ≤ n do {
      if (isPrime(i) and isPrime(j) and 2 * i + j = n) then return true; }
   return false;

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

Offensichtlich gilt: Die Lemoine'sche Vermutung gilt $\Leftrightarrow$ testConjecture() terminiert nicht. Das heißt: wenn wir ein Programm Π_h hätten, das das Halteproblem von SIMPLE entscheidet, 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

Beweis der Abzählbarkeit der rationalen Zahlen.

a) Zeigen Sie mit Hilfe einer Aufzählung der Elemente von $\mathbb{Q}$, dass die Menge $\mathbb{Q}$ abzählbar ist. Zwecks Nachvollziehbarkeit der Aufzählung, sollten Sie bei dieser Aufzählung die rationalen Zahlen 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!

   Bemerkung: Der Einfachheit halber brauchen Sie Brüche, die eigentlich (durch entsprechendes Kürzen) die selbe Zahl darstellen, nicht zu unterscheiden. Zum Beispiel können Sie ruhig 1/2, 2/4, 3/6, 4/8, etc. in der Aufzählung wie unterschiedliche Zahlen behandeln. Klarerweise ändert das nichts an der Gültigkeit des Abzählbarkeitsbeweises. Aus der Aufzählung folgt ja trotzdem, dass es eine injektive Abbildung von $\mathbb{Q}$ nach $\mathbb{N}$ gibt. Gehen Sie bei der Aufzählung ähnlich vor wie bei der Aufzählung von Σ^* bzw. $\mathbb{N}^2$ in der Vorlesung und wählen Sie eine geeignete Abwechslung zwischen positiven und negativen Zahlen.

b) Betrachten Sie nun die Dezimaldarstellung der rationalen Zahlen, z.B.: 0.5 (= 1/2) 0.333... (= 1/3), 0.2 (= 1/5), 0.1666... (= 1/6), 0.142857142857... (= 1/7), etc. Offensichtlich haben unendlich viele rationale Zahlen eine unendliche Dezimaldarstellung (nämlich immer, wenn der gekürzte Bruch im Nenner nicht nur die Primfaktoren 2 und 5 hat). In Aufgabe 1.2 mussten Sie beweisen, dass es selbst in dem kleinen Intervall $[0.5, 0.51)$ *überabzählbar viele reelle Zahlen* gibt. Der Grund dafür war die unendliche Dezimaldarstellung dieser Zahlen. Wieso können dann die rationalen Zahlen (von denen unendliche viele ebenfalls eine unendliche Dezimaldarstellung haben) *abzählbar* sein? Geben Sie eine Begründung für diesen scheinbaren Widerspruch! Überlegen Sie sich auch, wieso ein Überabzählbarkeitsbeweis mittels Diagonal-Argument (wie er in Aufgabe 1.2 funktioniert) bei den rationalen Zahlen scheitern würde!

Lösung 1.4

a) Eine Möglichkeit, die rationalen Zahlen aufzuzuzählen, ist, dass wir
   - die Brüche p/q mit $p, q \leq n$ für aufsteigendes $n \in \mathbb{N}$ aufzählen,
   - innerhalb der Brüche, die noch nicht vorher vorgekommen sind, (p, q) lexikographisch ordnen,
   - und nach jedem Block positiver Brüche die entsprechenden negativen Brüche ausgeben.

Bruch	Anzahl	Positionen
0	1	0
1, -1	2	1,2
1/2, 2/1, 2/2	3 $(= 2^2 - 1)$	3, ..., 5 $(= 2^2 + 1)$
-1/2, -2/1, -2/2	3	6, ..., 8 $(= 2 \cdot 2^2)$
1/3, 2/3, 3/1, 3/2, 3/3	5 $(= 3^2 - 2^2)$	9, ..., 13 $(= 3^2 + 2^2)$
-1/3, -2/3, -3/1, -3/2, -3/3	5	14, ..., 18 $(= 2 \cdot 3^2)$
1/4, 2/4, 3/4, 4/1, 4/2, 4/3, 4/4	7 $(= 4^2 - 3^2)$	19, ..., 25 $(= 4^2 + 3^2)$
-1/4, -2/4, -3/4, -4/1, -4/2, -4/3, -4/4	7	26, ..., 32 $(= 2 \cdot 4^2)$
etc.		

b) Es haben zwar unendlich viele rationale Zahlen eine unendliche Dezimaldarstellung. Aber diese Dezimaldarstellung hat eine (endliche!) Periode, d.h.: auch in der Dezimaldarstellung (aber eben unter Angabe der Periode) lassen sich die rationalen Zahlen als *endliche* Strings über einem endlichen Alphabet Σ darstellen, nämlich: $\Sigma = \{0, \ldots, 9\} \cup \{\dot{0}, \ldots, \dot{9}\} \cup \{-, .\}$, z.B.: $0.\dot{3}$ $(= 1/3)$, $0.1\dot{6}$ $(= 1/6)$, $0.\dot{1}\dot{4}\dot{2}\dot{8}\dot{5}\dot{7}$ $(= 1/7)$, etc.

   Im Prinzip könnte man natürlich versuchen, die Überabzählbarkeit von $\mathbb{Q}$ im Stil von Aufgabe 1.2 zu beweisen, indem man (wie im indirekten Beweis für Aufgabe 1.2) mit einer Aufzählung von $\mathbb{Q}$ startet und eine neue Zahl $\hat{z}$ mittels Änderung der "Diagonale" konstruiert. Der Beweis würde aber daran scheitern, dass die dabei konstruierte Zahl $\hat{z}$ keine rationale Zahl ist. Man hätte also nicht den gewünschten Widerspruch, dass sozusagen eine Zahl in der Aufzählung der rationalen Zahlen fehlt.

Aufgabe 1.5

Wir betrachten folgende Variante des Erreichbarer-Code Problems:

Einfach-Erreichbarer-Code
Instanz: (Quellcode von einem SIMPLE) Programm Π, String I, String L, Integer $n \geq 1$.
Frage: Führt das Programm Π auf dem Input I innerhalb von maximal n Programmschritten die Programmzeile mit dem Label L aus?

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 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 Einfach-Erreichbarer-Code Problem.

b) Begründung für die Korrektheit der von Ihnen skizzierten Entscheidungsprozedur auf positiven Instanzen des Einfach-Erreichbarer-Code Problems.

c) Begründung für die Korrektheit der von Ihnen skizzierten Entscheidungsprozedur auf negativen Instanzen des Einfach-Erreichbarer-Code 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 Erreichbarer-Code Problem in der Vorlesung (Folie 33) war nur eine *Semi-Entscheidungsprozedur* möglich, weil weder ein konkreter Input String I noch die Anzahl n der Programmschritte vorgegeben war. In Aufgabe 1.5 ist eine

Entscheidungsprozedur möglich, weil sowohl der Input String I als auch die Anzahl n der untersuchten Programmschritte als Teil der Instanz gegeben 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 Einfach-Erreichbarer-Code Problems, d.h.: Quellcode von einem SIMPLE Programm Π, Input String I für Π, Label L und Integer $n \geq 1$;
- Π_i analysiert Π und simuliert die Ausführung von Π auf dem Input I;
- Π_i führt einen Zähler k, der mit 0 initialisiert wird und nach jedem Programmschritt von Π um 1 erhöht wird.
- Π_i terminiert, wenn die Simulation von Π auf dem Input I terminiert oder wenn der Zähler k den Wert n erreicht oder wenn bei der Simulation von Π die Programmzeile mit dem Label L ausgeführt wird:
  (1) Falls Π_i terminiert, weil die Simulation von Π auf dem Input I die Programmzeile mit dem Label L ausführt, gibt Π_i den Wert true aus und terminiert selbst.
  (2) Falls Π_i terminiert, weil die Simulation von Π auf dem Input I terminiert oder wenn der Zähler k den Wert n erreicht, aber die Programmzeile mit dem Label L nicht ausgeführt wurde, 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 Einfach-Erreichbarer-Code 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, L, n) des Einfach-Erreichbarer-Code Problems aufgerufen, d.h.: das Programm Π führt auf dem Input I innerhalb von n Programmschritten die Programmzeile mit dem Label L aus. Π_i arbeitet in der Simulation von Π einen Programmschritt von Π auf dem Input I nach dem anderen ab und wird daher nach der Simulation von höchstens n Programmschritten die Programmzeile mit dem Label L des Programms Π ausführen. 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, L, n) des Einfach-Erreichbarer-Code Problems aufgerufen, d.h.: innerhalb von maximal n Programmschritten führt das Programm Π bei Aufruf mit Input I die Programmzeile mit dem Label L *nicht* aus – egal ob das Programm in weniger als n Schritten terminiert oder nicht. Π_i arbeitet in der Simulation von Π einen Programmschritt von Π auf dem Input I nach dem anderen ab. Und zwar werden maximal n Programmschritte simuliert. Da es sich bei (Π, I, L, n) um eine negative Instanz handelt, wird in der Simulation innerhalb von n Schritten die Programmzeile mit dem Label L nicht erreicht. Laut Fall (2) der obigen Beschreibung gibt Π_i in diesem Fall den Wert false aus und terminiert selbst. Das ist das korrekte Verhalten für eine Entscheidungsprozedur auf einer negativen Instanz.

Aufgabe 1.6

Beim Einfach-Erreichbarer-Code Problem in Aufgabe 1.5 wird das Erreichbarer-Code Problem von der Vorlesung in doppelter Hinsicht vereinfacht: eine Instanz enthält sowohl einen konkreten Input I als auch eine maximal erlaubte Anzahl n an Programmschritten, innerhalb derer eine bestimmte Programmzeile erreicht werden soll. Wir betrachten nun folgende Variante des Erreichbarer-Code Problems, die sozusagen zwischen den beiden Extremen liegt: dabei ist zwar der konkrete Input I aber nicht die maximal erlaubte Anzahl n an Programmschritten gegeben.

Variante2-Erreichbarer-Code
Instanz: (Quellcode von einem SIMPLE) Programm Π, String I, Label (= String) L.
Frage: Führt das Programm Π auf dem Input I den Code auf der Programmzeile mit dem Label L aus?

Beweisen Sie die Unentscheidbarkeit von Variante2-Erreichbarer-Code mittels Reduktion vom Halteproblem, d.h.: definieren Sie eine Reduktion vom Halteproblem auf das Variante2-Erreichbarer-Code Problem und beweisen Sie die Korrektheit der Reduktion.

Es sind also folgende Teilaufgaben zu lösen:

a) Reduktion vom HALTEPROBLEM auf das VARIANTE2-ERREICHBARER-CODE Problem.

b) "⇒"-Richtung des Korrektheitsbeweises: (Π, I) ist eine positive Instanz des HALTEPROBLEMS $\Rightarrow$ (Π', I', L) ist eine positive Instanz des VARIANTE2-ERREICHBARER-CODE Problems.

c) "⇐"-Richtung des Korrektheitsbeweises: (Π', I', L) ist eine positive Instanz des VARIANTE2-ERREICHBARER-CODE Problems $\Rightarrow$ (Π, 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', I', L) des VARIANTE2-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', I', L) von VARIANTE2-ERREICHBARER-CODE, indem wir für L einen beliebigen String wählen, z.B.: $L =$ "abcd", $I' = I$ setzen und Π' folgendermaßen definieren:

```
String Program Π' (String S)
    call Π(S);
    abcd: return "123";
```

d.h.: Π' nimmt einen String als Input und ruft mit diesem String sofort das Programm Π auf. Wenn die Kontrolle vom Aufruf von Π zu Π' zurückkommt, dann führt Π' nur noch das return-statment aus und terminiert. Dieses return-Statement hat das Label "abcd".

Korrektheit der Reduktion: Wir müssen folgende Äquivalenz zeigen:
(Π, I) ist eine positive Instanz des HALTEPROBLEMS ⇔
(Π', I', L) ist eine positive Instanz des VARIANTE2-ERREICHBARER-CODE Problems.
Wir zeigen die zwei Richtungen der Äquivalenz getrennt:

b) "⇒": Angenommen (Π, I) ist eine positive Instanz des HALTEPROBLEMS. Das heißt, bei Aufruf von Π mit dem Input I terminiert Π. Wir betrachten nun das Verhalten von Π' beim Aufruf mit dem Input $I' = I$: Das Programm Π' ruft sofort Π mit dem Input $I' = I$ auf. Laut Annahme terminiert Π auf dem Input I, d.h.: irgendwann kommt die Kontrolle wieder zu Π' zurück. Nach der Rückkehr vom Aufruf von Π führt Π' die Programmzeile mit dem Label $L =$ "abcd" aus. Also ist (Π', I', L) eine positive Instanz des VARIANTE2-ERREICHBARER-CODE Problems.

c) "⇐": Angenommen (Π', I', L) ist eine positive Instanz des VARIANTE2-ERREICHBARER-CODE Problems. Das heißt: Bei Ausführung von Π' mit dem Input I' wird der Code auf der Programmzeile mit dem Label $L =$ "abcd" ausgeführt. Laut obiger Definition von Π' ruft Π' zunächst das Programm Π mit dem Input I $(= I')$ auf. Damit Π' den Code auf der Programmzeile mit label $L =$ "abcd" ausführen kann, muss die Kontrolle vom Aufruf von Π mit Input I zu Π' zurückkommen. Das bedeutet, dass Π auf dem Input I terminiert Das heißt: (Π, I) ist eine positive Instanz des HALTEPROBLEMS.

Aufgabe 1.7

Wir definieren eine weitere Variante des ERREICHBARER-CODE Problems:

MINDESTENS-EINS-VON-ZWEI-ERREICHBARER-CODE (kurz MINDESTENS-EVZEC)
Instanz: (Quellcode von einem SIMPLE) Programm Π, Strings I_1, I_2, Label (= String) L.
Frage: Führt das Programm Π bei *mindestens* einem der Aufrufe mit Input I_1 oder mit Input I_2 die Programmzeile mit dem Label L aus?

Zeigen Sie, dass dieses Problem semi-entscheidbar ist, indem Sie eine Semi-Entscheidungsprozedur für das MINDESTENS-EVZEC 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-EVZEC Problem.

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

c) Begründung für die Korrektheit der von Ihnen skizzierten Semi-Entscheidungsprozedur auf negativen Instanzen des MINDESTENS-EVZEC 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-EVZEC Problems, d.h.: Quellcode von einem SIMPLE Programm Π, Input Strings I_1, I_2 und Label L.
- Π_i simuliert abwechselnd einen Schritt von Π auf dem Input I_1 und einen Schritt von Π auf dem Input I_2.
- Wenn eine der Simulationen die Programmzeile mit dem Label L ausführt, dann gibt Π_i den Wert true aus und terminiert selbst.
- Wenn die Simulation auf einem der Inputs terminiert, ohne dass die Programmzeile mit dem Label L ausgeführt wurde, wird nur noch die Simulation mit dem anderen Input fortgesetzt. Wenn auch diese Simulation terminiert, ohne dass die Programmzeile mit dem Label L ausgeführt wurde, dann gibt Π_i den Wert false aus und terminiert selbst.

Korrektheit der Semi-Entscheidungsprozedur: Wir argumentieren, dass das oben skizzierte Programm Π_i eine Semi-Entscheidungsprozedur für das MINDESTENS-EVZEC 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 mit einer positiven Instanz (Π, I_1, I_2, L) des MINDESTENS-EVZEC Problems aufgerufen wird, d.h.: zumindest auf einem der beiden Input Strings I_1 und/oder I_2 führt das Programm Π die Programmzeile mit dem Label L aus. Sei I_j mit $j \in \{1, 2\}$ der Input auf dem das Programm Π die Programmzeile mit dem Label L als erstes ausführt, d.h.: das Programm Π ereicht bei Ausführung mit Input I_j die Programmzeile mit dem Label L nach einer gewissen Anzahl k von Schritten; bei Ausführung mit Input I_{3-j} erreicht Π die Programmzeile mit Label L entweder nie oder erst nach k' Schritten mit $k' \geq k$.

   Das Programm Π_i simuliert jeweils abwechselnd einen Schritt von Π auf dem Input I_1 und auf dem Input I_2. Nach der Simulation von k Schritten wird Π_i "entdecken", dass Π auf dem Input I_j die Programmzeile mit dem Label L erreicht. Laut obiger Beschreibung gibt Π_i dann den Wert true aus und terminiert. Das ist 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, I_1, I_2, L) des MINDESTENS-EVZEC Problems aufgerufen wird, d.h.: weder beim Aufruf mit Input String I_1 noch beim Aufruf mit dem Input String I_2 führt das Programm Π die Programmzeile mit dem Label L aus.

   Das Programm Π_i simuliert jeweils abwechselnd einen Schritt von Π auf dem Input I_1 und auf dem Input I_2. Laut Annahme wird keine der Simulationen jemals die Programmzeile mit dem Label L erreichen. Wir unterscheiden 2 Fälle:

   - Das Programm Π terminiert auf beiden Inputs I_1 und I_2: Laut Annahme wird bei beiden Inputs nie die Programmzeile mit Label L erreicht. Laut obiger Beschreibung gibt Π_i dann den Wert false aus und terminiert.
   - Auf mindestens einem der Inputs I_1 und/oder I_2 läuft das Programm Π endlos. Laut Annahme wird bei beiden Inputs nie die Programmzeile mit Label L erreicht. Laut obiger Beschreibung läuft dann das Programm Π_i endlos.

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

Aufgabe 1.8

In der Vorlesung wurde gezeigt, dass das HALTEPROBLEM unentscheidbar und semi-entscheidbar ist. Daraus haben wir geschlossen, dass das co-HALTEPROBLEM nicht semi-entscheidbar ist. Es ist leicht zu zeigen, dass auch folgende Variante des HALTEPROBLEMs unentscheidbar und semi-entscheidbar ist (wir schreiben ϵ für den leeren String, d.h.: $\epsilon =$ “”).

HALTEPROBLEM-NICHTLEERERSTRING (kurz HALTEPROBLEM-NLS):
Instanz: (Quellcode von einem SIMPLE) Programm Π, *nicht-leerer* String I (d.h.: $I \neq \epsilon$).
Frage: Terminiert das Programm Π bei Aufruf mit Input I?

Wir definieren nun noch eine weitere Variante des ERREICHBARER-CODE Problems:

EXAKT-EINS-VON-ZWEI-ERREICHBARER-CODE (kurz EXAKT-EVZEC)
Instanz: (Quellcode von einem SIMPLE) Programm Π, Strings I_1, I_2, Label (= String) L.
Frage: Führt das Programm Π bei *exakt* einem der Aufrufe mit Input I_1 bzw. I_2 die Programmzeile mit dem Label L aus? Das heißt: bei Ausführung mit einem der beiden Input Strings erreicht Π die Programmzeile mit Label L, während bei Ausführung mit dem anderen Input String die Programmzeile mit dem Label L nicht erreicht wird.

Zeigen Sie mittels Reduktion vom co-HALTEPROBLEM-NLS, dass das EXAKT-EVZEC Problem 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-NLS auf das EXAKT-EVZEC Problem.

b) “$\Rightarrow$”-Richtung des Korrektheitsbeweises: (Π, I) ist eine positive Instanz des co-HALTEPROBLEM-NLS $\Rightarrow$ (Π', I_1, I_2, L) ist eine positive Instanz des EXAKT-EVZEC Problems.

c) “$\Leftarrow$”-Richtung des Korrektheitsbeweises: (Π', I_1, I_2, L) ist eine positive Instanz des EXAKT-EVZEC Problems $\Rightarrow$ (Π, I) ist eine positive Instanz des co-HALTEPROBLEM-NLS.

Tipp: Für eine gegebene Instanz (Π, I) des co-HALTEPROBLEM-NLS wählen Sie den Input String $I_1 = \epsilon$ (d.h.: leerer String) und definieren Sie das Programm Π' so, dass das Programm Π' bei Input I_1 die Programmzeile mit Label L *garantiert erreicht*. Mit dem Verhalten von Π' auf dem Input I_2 müssen Sie in geeigneter Weise das Verhalten von Π auf I nachbauen.

Lösung 1.8

a) *Reduktion:* Sei (Π, I) eine beliebige Instanz des co-HALTEPROBLEM-NLS. Wir definieren daraus folgende Instanz (Π', I_1, I_2, L) des EXAKT-EVZEC Problems mit $I_1 =$ “”, $I_2 = I$ und $L =$ “abcd” (d.h.: wir wählen einen beliebigen String). Außerdem definieren wir Π' wie folgt:

```
String Program Π' (String S)
    if S != “” then call Π(S);
    abcd: return “123”;
```

Korrektheit der Reduktion: Wir müssen folgende Äquivalenz zeigen:
(Π, I) ist eine positive Instanz des co-HALTEPROBLEM-NLS (d.h.: Π terminiert nicht auf dem Input I) $\Leftrightarrow$ (Π', I_1, I_2, L) ist eine positive Instanz des EXAKT-EVZEC Problems (d.h.: das Programm Π' erreicht bei Ausführung mit exakt einem der Input Strings I_1, I_2 die Programmzeile mit Label L, und beim anderen Input String wird die Programmzeile mit Label L nicht erreicht).

Wir zeigen die zwei Richtungen der Äquivalenz getrennt:

b) “$\Rightarrow$”: Angenommen (Π, I) ist eine positive Instanz des co-HALTEPROBLEM-NLS, d.h.: bei Aufruf von Π mit dem Input I terminiert Π nicht. Laut Definition des HALTEPROBLEM-NLS (und somit auch des co-HALTEPROBLEM-NLS) gilt, dass I ein nicht-leerer String ist.

Wir betrachten zuerst das Verhalten von Π' beim Aufruf mit dem Input $I_1 =$ “”. Da in diesem Fall die if-Bedingung im Programm Π' nicht erfüllt ist, wird die if-Anweisung nicht ausgeführt. Das Programm geht sofort weiter zur Programmzeile mit dem Label “abcd”.

Wir betrachten nun das Verhalten von Π' beim Aufruf mit dem Input $I_2 = I$. Laut Definition des HALTEPROBLEM-NLS ist $I_2 = I$ ein nicht-leerer String, d.h.: in diesem Fall ist die if-Bedingung im Programm Π' erfüllt und die if-Anweisung wird daher ausgeführt. Also wird das Programm Π mit dem Input String $I_2 = I$ aufgerufen.

Laut Annahme terminiert Π auf dem Input I nicht, d.h.: das Programm Π' bekommt nie mehr die Kontrolle zurück. Daher führt Π' in diesem Fall die Programmzeile mit dem Label $L =$ "abcd" niemals aus. Insgesamt gilt also: Das Programm Π' erreicht bei Aufruf mit dem Input I_1 die Programmzeile mit Label L, und bei Aufruf mit Input I_2 erreicht Π' die Programmzeile mit Label L nicht. Also ist (Π', I_1, I_2, L) eine positive Instanz des EXAKT-EVZEC Problems.

c) "$\Leftarrow$": Angenommen (Π', I_1, I_2, L) ist eine positive Instanz des EXAKT-EVZEC Problems. Das heißt: Bei Ausführung von Π' mit einem der beiden Inputs I_1 oder I_2 wird die Programmzeile mit dem Label L ausgeführt und beim anderen Input nicht.

Wir betrachten zuerst das Verhalten von Π' beim Aufruf mit dem Input $I_1 =$ "". Da in diesem Fall die if-Bedingung im Programm Π' nicht erfüllt ist, wird die if-Anweisung nicht ausgeführt. Das Programm geht daher sofort weiter zur Programmzeile mit dem Label "abcd".

Laut Annahme erreicht Π' bei exakt einem der beiden Aufrufe mit I_1 oder I_2 die Programmzeile mit dem Label "abcd". Daraus folgt, dass Π' beim Aufruf mit dem Input String I_2 die Programmzeile mit dem Label "abcd" nicht erreicht. Das ist aber nur möglich, wenn Π' die if-Anweisung ausführt und die Kontrolle vom Aufruf des Programms Π mit dem Input I_2 $(= I)$ nie mehr zu Π' zurückkommt. Das bedeutet, dass Π auf dem Input I nicht terminiert, d.h.: (Π, I) ist eine positive Instanz des co-HALTEPROBLEM-NLS.

Aufgabe 1.9

Wir betrachten noch einmal das EXAKT-EVZEC Problem und das HALTEPROBLEM-NLS von Aufgabe 1.8.

Zeigen Sie mittels Reduktion vom co-HALTEPROBLEM-NLS, dass auch das co-EXAKT-EVZEC 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-NLS auf das EXAKT-EVZEC Problem reduzieren.

Es sind also folgende Teilaufgaben zu lösen:

a) Reduktion vom HALTEPROBLEM-NLS auf das EXAKT-EVZEC Problem.

b) "$\Rightarrow$"-Richtung des Korrektheitsbeweises: (Π, I) ist eine positive Instanz des HALTEPROBLEM-NLS $\Rightarrow$ (Π', I_1, I_2, L) ist eine positive Instanz des EXAKT-EVZEC Problems.

c) "$\Leftarrow$"-Richtung des Korrektheitsbeweises: (Π', I_1, I_2, L) ist eine positive Instanz des EXAKT-EVZEC Problems $\Rightarrow$ (Π, I) ist eine positive Instanz des HALTEPROBLEM-NLS.

Tipp: Für eine gegebene Instanz (Π, I) des HALTEPROBLEM-NLS wählen Sie den Input String $I_1 = \epsilon$ (d.h.: leerer String) und definieren Sie das Programm Π' so, dass das Programm Π' bei Input I_1 die Programmzeile mit Label L *garantiert nicht erreicht*. Mit dem Verhalten von Π' auf dem Input I_2 müssen Sie in geeigneter Weise das Verhalten von Π auf I nachbauen.

Lösung 1.9

a) *Reduktion:* Sei (Π, I) eine beliebige Instanz des HALTEPROBLEM-NLS. Wir definieren daraus folgende Instanz (Π', I_1, I_2, L) des EXAKT-EVZEC Problems mit $I_1 =$ "", $I_2 = I$ und $L =$ "abcd" (d.h.: wir wählen einen beliebigen String). Außerdem definieren wir Π' wie folgt:

String Program Π' (String S)
  if $S =$ "" then return "123";
  call $\Pi(S)$;
  abcd: return "456";

Korrektheit der Reduktion: Wir müssen folgende Äquivalenz zeigen:
(Π, I) ist eine positive Instanz des HALTEPROBLEM-NLS (d.h.: Π terminiert auf dem Input I) $\Leftrightarrow$ (Π', I_1, I_2, L) ist eine positive Instanz des EXAKT-EVZEC Problems (d.h.: das Programm Π' erreicht bei Ausführung mit exakt einem der Input Strings I_1, I_2 die Programmzeile mit Label L, und beim anderen Input String wird die Programmzeile mit Label L nicht erreicht.

Wir zeigen die zwei Richtungen der Äquivalenz getrennt:

b) "$\Rightarrow$": Angenommen (Π, I) ist eine positive Instanz des HALTEPROBLEM-NLS, d.h.: bei Aufruf von Π mit dem Input I terminiert Π. Laut Definition des HALTEPROBLEM-NLS gilt, dass I ein nicht-leerer String ist.

Wir betrachten zuerst das Verhalten von Π' beim Aufruf mit dem Input $I_1 =$ "". Da in diesem Fall die if-Bedingung im Programm Π' erfüllt ist, wird die if-Anweisung ausgeführt, d.h.: das Programm liefert den Wert "123" und terminiert. Insbesondere bedeutet das, dass die Programmzeile mit dem Label "abcd" nie erreicht wird. (Bemerkung: natürlich hätten wir Π' auch so definieren können, dass in der if-Anweisung eine Endlos-Schleife steht; auch dann würde die Programmzeile mit dem Label "'abcd" nie erreicht werden).

Wir betrachten nun das Verhalten von Π' beim Aufruf mit dem Input $I_2 = I$. Laut Definition des HALTEPROBLEM-NLS ist $I_2 = I$ ein nicht-leerer String, d.h.: in diesem Fall ist die if-Bedingung im Programm Π' nicht erfüllt und die if-Anweisung wird daher nicht ausgeführt. Also geht die Programmausführung von Π' beim Aufruf des Programms Π mit dem Input String $I_2 = I$ weiter.

Laut Annahme terminiert Π auf dem Input I, d.h.: das Programm Π' bekommt irgendwann die Kontrolle zurück und führt als nächstes die Programmzeile mit dem Label $L =$ "abcd" aus. Insgesamt gilt also: Das Programm Π' erreicht bei Aufruf mit dem Input I_2 die Programmzeile mit Label L, und bei Aufruf mit Input I_1 erreicht Π' die Programmzeile mit Label L nicht. Also ist (Π', I_1, I_2, L) eine positive Instanz des EXAKT-EVZEC Problems.

c) "$\Leftarrow$": Angenommen (Π', I_1, I_2, L) ist eine positive Instanz des EXAKT-EVZEC Problems. Das heißt: Bei Ausführung von Π' mit einem der beiden Inputs I_1 oder I_2 wird die Programmzeile mit dem Label L ausgeführt und beim anderen Input nicht.

Wir betrachten zuerst das Verhalten von Π' beim Aufruf mit dem Input $I_1 =$ "". Da in diesem Fall die if-Bedingung im Programm Π' erfüllt ist, wird die if-Anweisung ausgeführt, d.h.: das Programm liefert den Wert "123" und terminiert. Insbesondere bedeutet das, dass die Programmzeile mit dem Label "abcd" nie erreicht wird.

Laut Annahme erreicht Π' bei exakt einem der beiden Aufrufe mit I_1 oder I_2 die Programmzeile mit dem Label "abcd". Daraus folgt, dass Π' beim Aufruf mit dem Input String I_2 die Programmzeile mit dem Label "abcd" erreicht. Das ist aber nur möglich, wenn zuerst Π mit Input $I_2 = I$ aufgerufen wird und die Kontrolle von diesem Aufruf irgendwann zu Π' zurückkommt. Das bedeutet, dass Π auf dem Input I terminiert, d.h.: (Π, I) ist eine positive Instanz des HALTEPROBLEM-NLS.

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 unentscheidbar (und somit auch co-$\mathcal{P}_1$),
- $\mathcal{P}_2$ ist semi-entscheidbar (während über co-$\mathcal{P}_2$ nichts bekannt ist),
- $\mathcal{P}_4$ ist entscheidbar (und somit auch co-$\mathcal{P}_4$).

Über das Problem $\mathcal{P}_3$ 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 $\mathcal{P}_3 \leq \mathcal{P}_2$ 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 $\mathcal{P}_4 \leq \mathcal{P}_3$ 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-$\mathcal{P}_3$ *zusätzlich* zum bereits bekannten Wissen aussagen?

d) Angenommen, es gelten die Reduktionen $\mathcal{P}_2 \leq \mathcal{P}_4$ 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?

Lösung 1.10

a) Dann gilt:
   - $\mathcal{P}_1$ ist semi-entscheidbar (wegen $\mathcal{P}_1 \leq \mathcal{P}_2$ und Semi-Entscheidbarkeit von $\mathcal{P}_2$);
   - co-$\mathcal{P}_1$ ist nicht einmal semi-entscheidbar (wegen Unentscheidbarkeit und Semi-Entscheidbarkeit von $\mathcal{P}_1$).
   - $\mathcal{P}_2$ ist unentscheidbar (wegen $\mathcal{P}_1 \leq \mathcal{P}_2$ und Unentscheidbarkeit von $\mathcal{P}_1$);
   - co-$\mathcal{P}_2$ ist nicht einmal semi-entscheidbar (wegen Unentscheidbarkeit und Semi-Entscheidbarkeit von $\mathcal{P}_2$);
   - $\mathcal{P}_3$ ist unentscheidbar (wegen $\mathcal{P}_2 \leq \mathcal{P}_3$ und Unentscheidbarkeit von $\mathcal{P}_2$);
   - co-$\mathcal{P}_3$ ist nicht einmal semi-entscheidbar (weil $\mathcal{P}_2 \leq \mathcal{P}_3$ äquivalent zu co-$\mathcal{P}_2 \leq$ co-$\mathcal{P}_3$ ist und co-$\mathcal{P}_2$ nicht einmal semi-entscheidbar ist);

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

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

d) Dann gilt:
   - $\mathcal{P}_3$ ist entscheidbar (wegen $\mathcal{P}_3 \leq \mathcal{P}_4$ und Entscheidbarkeit von $\mathcal{P}_4$) und somit auch co-$\mathcal{P}_3$;
   - $\mathcal{P}_2$ ist entscheidbar (wegen $\mathcal{P}_2 \leq \mathcal{P}_4$ und Entscheidbarkeit von $\mathcal{P}_4$) und somit auch co-$\mathcal{P}_2$.