Theoretische Informatik
Übungsblatt 1 (2023W)
Lösungsvorschlag

Allgemeine Hinweise:

- Die Deadline für die Abgabe der Lösungen zum Übungsblatt 1 ist der **23. Oktober 2023**.
- 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

Eine der wichtigsten Aufgaben von Datenbankmanagementsystemen (DBMS) ist es, unterschiedliche Arten von Anfragen von Benutzer_innen zu beantworten. Formal gesprochen, muss das DBMS dabei unterschiedliche Arten von *Problemen* lösen. Formulieren Sie Beispiele für die folgenden Typen von Problemen (Instanz? Frage?) im Zusammenhang mit der Anfragebeantwortung durch ein DBMS. Bedenken Sie dabei, dass Probleme für *unendlich* viele Instanzen definiert sind, d.h.: die Probleme sollten für *beliebige* Datenbanken D (Sie brauchen sich bei dieser Aufgabe nicht um ein konkretes Schema zu kümmern) und/oder *beliebige* Anfragen Q definiert werden.

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 Variante 1:
Instanz: Datenbank D, Anfrage Q.
Frage: Liefert die Anfrage Q über der Datenbank D eine nicht-leere Antwort?

Entscheidungsproblem Variante 2:
Instanz: Datenbank D, Anfrage Q, Tupel t.
Frage: Ist das Tupel t in der Antwort auf die Anfrage Q über der Datenbank D enthalten?

Bemerkung: Das Entscheidungsproblem Variante 1 entsteht in SQL z.B. bei einer geschachtelten SELECT-FROM-WHERE Anfrage mit dem Schlüsselwort "EXISTS" im WHERE-Teil der Anfrage. Variante 2 ensteht z.B. bei einer geschachtelten SELECT-FROM-WHERE Anfrage mit dem Schlüsselwort "IN" im WHERE-Teil der Anfrage.

b) Aufzählproblem:
Instanz: Datenbank D, Anfrage Q.
Frage: Ausgabe aller Tupel t in der Antwort auf die Anfrage Q über der Datenbank D.

Bemerkung: Diese Art von Problemen muss von einem DBMS in SQL z.B. für SELECT-FROM-WHERE Anfragen gelöst werden, bei denen im SELECT-Teil zumindest ein Attribut (oder "*") steht.

c) Zählproblem:
Instanz: Datenbank D, Anfrage Q.
Frage: Wie viele Tupel t sind in der Antwort auf die Anfrage Q über der Datenbank D enthalten?

Bemerkung: Diese Art von Problemen muss von einem DBMS in SQL z.B. für SELECT-FROM-WHERE Anfragen gelöst werden, bei denen das Schlüsselwort "COUNT" im SELECT-Teil verwendet wird.

d) Optimierungsproblem:
Instanz: Datenbank D, Anfrage Q, Attribut A.
Frage: Was ist der minimale (oder maximale) Wert, den das Attribut A unter allen Tupeln t annehmen kann, die in der Antwort auf die Anfrage Q über der Datenbank D enthalten sind?

Bemerkung: Diese Art von Problemen muss von einem DBMS in SQL z.B. für SELECT-FROM-WHERE Anfragen gelöst werden, bei denen eine der Aggregat-Funktionen "MIN" oder "MAX" im SELECT-Teil verwendet wird.

e) Suchproblem:
Instanz: Datenbank D, Anfrage Q.
Frage: Ausgabe eines Tupels aus der Antwort auf die Anfrage Q über der Datenbank D.

Bemerkung: Diese Art von Problemen muss von einem DBMS in SQL z.B. für SELECT-FROM-WHERE Anfragen mit dem Zusatz "LIMIT 1" gelöst werden.

Aufgabe 1.2
Beweisen Sie (mittels Diagonalargument), dass das Intervall $[0, 0.001)$ überabzählbar viele reelle Zahlen enthält.

Lösung 1.2
Indirekter Beweis: Wir nehmen an, dass die reellen Zahlen im Intervall $[0, 0.001)$ abzählbar sind. Das heißt, wir können Sie aufzählen als $z_0, z_1, z_2, \ldots$. Alle Zahlen $z_i \in [0, 0.001)$ lassen sich als Dezimalzahlen der Form $z_i = 0.000a_{i0}a_{i1}a_{i2}a_{i3}\ldots$ darstellen, wobei alle a_{ij} Ziffern aus dem Bereich $\{0, \ldots, 9\}$ sind. Mit anderen Worten, a_{ij} ist die $(j+4)$-te Nachkommastelle der i-ten Zahl in dieser Aufzählung. Die Aufzählung *aller* Zahlen $z_0, z_1, z_2, \ldots$ hat daher folgende Gestalt:

$$\begin{array}{lcl} z_0 & = & 0.000a_{00}a_{01}a_{02}a_{03}a_{04}\ldots \\ z_1 & = & 0.000a_{10}a_{11}a_{12}a_{13}a_{14}\ldots \\ z_2 & = & 0.000a_{20}a_{21}a_{22}a_{23}a_{24}\ldots \\ z_3 & = & 0.000a_{30}a_{31}a_{32}a_{33}a_{34}\ldots \\ \vdots & = & \vdots \end{array}$$

Wir definieren nun die Zahl $\hat{z} = 0.000b_0b_1b_2\ldots$ (d.h.: die ersten 3 Nachkommastellen sind 0 und die $(j+4)$-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 $(k+4)$-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 $[0, 0.001)$ enthält. Widerspruch! Das heißt, unsere ursprüngliche Annahme, dass die reellen Zahlen im Intervall $[0, 0.001)$ abzählbar sind, war falsch. Die reellen Zahlen im Intervall $[0, 0.001)$ sind also überabzählbar.

Aufgabe 1.3
Der große Fermatsche Satz besagt: *Für $n \geq 3$ gibt es keine ganzen Zahlen $a, b, c \geq 1$, die die Gleichung $a^n + b^n = c^n$ erfüllen.*

Bemerkung: Für $n = 2$ gibt es unendlich viele ganze Zahlen $a, b, c \geq 1$, für die die obige Gleichung gilt, nämlich die "Pythagoräischen Zahlen".

Pierre de Fermat hat diese Behauptung im 17. Jahrhundert aufgestellt. Es wurde aber nie ein Beweis von ihm gefunden. Der Satz wurde schließlich 1994 vom englischen Mathematiker Andrew John Wiles bewiesen, was damals als große Leistung gefeiert wurde.

Zeigen Sie, dass sich der große Fermatsche Satz prinzipiell (d.h. ohne Berücksichtigung von Komplexitätsüberlegungen) leicht beweisen ließe, wenn das Halteproblem von SIMPLE entscheidbar wäre. Sie dürfen dabei die Existenz einer library function "power(·,·)" annehmen, die bei einem Aufruf der Art "power(x,k)" den Wert x^k liefert.

Lösung 1.3

Wir können in SIMPLE folgendes Programm schreiben, das nach ganzen Zahlen a, b, c, n sucht, die den großen Fermatschen Satz widerlegen würden, d.h.: Zahlen a, b, c, n, für die gilt: $a, b, c \geq 1$, $n \geq 3$ und $a^n + b^n = c^n$. Wir gehen dabei so vor, dass wir in einer äußeren Schleife über m immer größere Werte $a, b, c, n \leq m$ mit der gewünschten Eigenschaft suchen.

```
Boolean test(Integer m)    /* checks if there exist a,b,c,n ≤ m that contradict Fermat */
   for all 1 ≤ a,b,c ≤ m and all 3 ≤ n ≤ m do {
      if ( power(a,n) + power(b,n) = power(c,n) ) then return true; }
   return false;

Void testFermat()
   m:=3;
   while test(m)=false do { m := m +1; }
```

Offensichtlich gilt: Der große Fermatsche Satz ist wahr $\Leftrightarrow$ testFermat() 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 testFermat() und dem leeren String (als Input für testFermat()) aufrufen. Wenn Π_h den Wert true liefert (d.h.: testFermat() terminiert), wissen wir, dass der große Fermatsche Satz falsch ist; falls Π_h den Wert false liefert, ist der große Fermatsche Satz wahr.

Bemerkung: Die Prozedur test() ist offensichtlich sehr ineffizient, da bei jedem Aufruf mit einem bestimmten Wert von m alle vorangeganenen Tests für kleinere Werte von m wiederholt werden. Diese Ineffizienz könnte man z.B. dadurch beseitigen, dass man in 4 Schleifen jeweils eine der Variablen a,b,c,n auf m setzt und nur die restlichen Variablen den vollen Wertebereich durchlaufen. Da es uns hier nur um die Entscheidbarkeit geht, wird auf die Effizienz zugunsten der Lesbarkeit der Prozedur verzichtet.

Aufgabe 1.4

Zeigen Sie folgende Abzählbarkeitseigenschaften:

a) Die Menge $\mathbb{G}$ der geraden Zahlen, die Menge $\mathbb{U}$ der ungeraden 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 $\mathbb{N}^3$ der Zahlentripel über den natürlichen Zahlen ist gleich groß, d.h.: sie ist abzählbar unendlich. Um diese Behauptung zu beweisen, sollen Sie eine Aufzählung der Elemente von $\mathbb{N}^3$ 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) Wir definieren folgende Abbildungen:

$f : \mathbb{N} \to \mathbb{G}$ mit $f(n) = 2n$ für beliebiges $n \in \mathbb{N}$
$g : \mathbb{N} \to \mathbb{U}$ mit $g(n) = 2n + 1$ für beliebiges $n \in \mathbb{N}$
$h : \mathbb{G} \to \mathbb{U}$ mit $h(n) = n + 1$ für beliebiges $n \in \mathbb{G}$

Die Abbildungen sind offensichtlich bijektiv, mit folgenden Umkehrfunktionen:

$f^{-1} : \mathbb{G} \to \mathbb{N}$ mit $f^{-1}(n) = \frac{n}{2}$ für beliebiges $n \in \mathbb{G}$
$g^{-1} : \mathbb{U} \to \mathbb{N}$ mit $g^{-1}(n) = \frac{n-1}{2}$ für beliebiges $n \in \mathbb{U}$
$h^{-1} : \mathbb{U} \to \mathbb{G}$ mit $h^{-1}(n) = n - 1$ für beliebiges $n \in \mathbb{U}$

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 $(a, b, c) \in \mathbb{N}^3$ aufzuzuzählen, ist, dass wir
   - die Tripel (a, b, c) mit $a, b, c \leq n$ für aufsteigendes $n \in \mathbb{N}$ aufzählen

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

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

Bemerkung: In dieser Aufgabe wurde verlangt, dass Sie die Abzählbarkeit von $\mathbb{N}^3$ durch Angabe einer Aufzählung der Tripel in $\mathbb{N}^3$ 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 $\mathbb{N}^3$ 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 $\mathbb{N}^3$.

Aufgabe 1.5

Wir betrachten folgende Variante des Halteproblems:

Konkretes Halteproblem
Instanz: (Quellcode von einem SIMPLE) Programm Π, Input String I, Integer $n \geq 1$.
Frage: Terminiert das Programm Π auf dem Input I innerhalb von n Programmschritten?

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. Geben Sie auch eine kurze Begründung, dass es sich dabei tatsächlich um eine Entscheidungsprozedur für dieses Problem handelt. Sie dürfen davon ausgehen, dass Ihnen ein Interpreter für SIMPLE Programme zur Verfügung steht, der einen Programmschritt nach dem anderen ausführen kann.

Lösung 1.5

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 Konkreten Halteproblems, d.h.: Quellcode von einem SIMPLE Programm Π, Input String I für Π 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:

(1) Falls Π_i terminiert, weil die Simulation von Π auf dem Input I terminiert, dann gibt Π_i den Wert true aus.

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

Begründung: Wir argumentieren, dass das oben skizzierte Programm Π_i eine Entscheidungsprozedur für das Konkrete Halteproblem ist. Dazu unterscheiden wir 2 Fälle:

(1) Angenommen das Programm Π_i wird mit einer positiven Instanz (Π, I, n) des Konkreten Halteproblems aufgerufen, d.h.: das Programm Π terminiert auf dem Input I innerhalb von n Programmschritten. Π_i arbeitet in der Simulation von Π einen Programmschritt von Π auf dem Input I nach dem anderen ab und wird daher (nach höchstens n Programmschritten) feststellen, dass Π terminiert. In diesem Fall gibt Π_i den Wert true aus und terminiert selbst.

(2) Angenommen das Programm Π_i wird mit einer negativen Instanz (Π, I, n) des Konkreten Halteproblems aufgerufen, d.h.: das Programm Π auf dem Input I terminiert nicht innerhalb von n Schritten. Π_i arbeitet in der Simulation von Π einen Programmschritt von Π auf dem Input I 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 wurde gerade ausgeführt) und Π bis dahin nicht terminiert, dann gibt Π_i den Wert false aus und terminiert.

In beiden Fällen ist garantiert, dass das Programm Π_i terminiert und die korrekte Antwort true/false ausgibt. Das ist das korrekte Verhalten für eine Entscheidungsprozedur.

Aufgabe 1.6

In der Vorlesung wurde folgendes Problem vorgestellt:

Selber-Output (SO)
Instanz: Programme Π_1, Π_2 und Input String I.
Frage: Haben Π_1 und Π_2 auf Input I das gleiche Verhalten?
Das heißt: entweder terminieren Π_1 und Π_2 bei Input I und liefern das gleiche Ergebnis oder beide Programme terminieren nicht bei Input I?

Außerdem wurde in der Vorlesung gezeigt, dass weder dieses Problem noch sein co-Problem semi-entscheidbar ist. Betrachten Sie nun folgendes Problem:

Dreimal-Selber-Output (3SO)
Instanz: Programme Π_1, Π_2, Π_3 und Input String I.
Frage: Haben Π_1, Π_2 und Π_3 auf Input I das gleiche Verhalten?
Das heißt: entweder alle 3 Programme terminieren bei Input I und liefern das gleiche Ergebnis oder alle 3 Programme terminieren nicht bei Input I?

Zeigen Sie mittels Reduktion von SO, dass weder 3SO noch sein co-Problem semi-entscheidbar ist. Zeigen Sie eine Richtung des Korrektheitsbeweises für die von Ihnen vorgeschlagene Reduktion R, nämlich: Wenn x eine positive Instanz von SO ist, dann ist $R(x)$ eine positive Instanz von 3SO.

Lösung 1.6

Reduktion: Sei (Π_1, Π_2, I) eine beliebige Instanz von SO. Wir definieren daraus eine Instanz $(\Pi_1', \Pi_2', \Pi_3', I')$ von 3SO mit $\Pi_1' = \Pi_1$, $\Pi_2' = \Pi_2$, $\Pi_3' = \Pi_2$ und $I' = I$. In anderen Worten, für eine Instanz $x = (\Pi_1, \Pi_2, I)$ von SO, hat die Instanz $R(x)$ von 3SO die Form $R(x) = (\Pi_1, \Pi_2, \Pi_2, I)$.

Korrektheit der Reduktion: Wir zeigen nur eine Richtung der Korrektheit der Reduktion, nämlich: Für eine beliebige Instanz x von SO gilt: wenn x eine positive Instanz von SO ist, dann ist $R(x)$ eine positive Instanz von 3SO.

Beweis: Angenommen $x = (\Pi_1, \Pi_2, I)$ ist eine positive Instanz von SO. Wir unterscheiden 2 Fälle:

(1) Π_1 und Π_2 terminieren auf dem Input I und liefern den gleichen Output. Laut Problemreduktion gelten für die Instanz $R(x) = (\Pi_1', \Pi_2', \Pi_3', I')$ die Gleichheiten $\Pi_1' = \Pi_1$, $\Pi_2' = \Pi_2$, $\Pi_3' = \Pi_2$ und $I' = I$. Das heißt, dass alle 3 Programme Π_1', Π_2', Π_3' auf dem Input I' terminieren und den gleichen Output liefern. $R(x)$ ist also eine positive Instanz von 3SO.

(2) Π_1 und Π_2 terminieren nicht auf dem Input I. Dann gilt, dass auch Π_1', Π_2', Π_3' mit $\Pi_1' = \Pi_1$, $\Pi_2' = \Pi_2$, $\Pi_3' = \Pi_2$ auf dem Input $I' = I$ nicht terminieren. Das heißt, $R(x) = (\Pi_1', \Pi_2', \Pi_3', I')$ ist eine positive Instanz von 3SO.

Aufgabe 1.7
Wir definieren folgendes Problem als leichte Abänderung des HALTEPROBLEMS:

MINDESTENS-EINS-VON-ZWEI-HALTEPROBLEM
Instanz: Programme Π_1, Π_2 und Input String I.
Frage: Hält *mindestens* eines der Programme Π_1 und Π_2 auf Input I?

Zeigen Sie, dass dieses Problem semi-entscheidbar ist, indem Sie eine Semi-Entscheidungsprozedur für das MINDESTENS-EINS-VON-ZWEI-HALTEPROBLEM skizzieren. Geben Sie auch eine kurze Begründung, dass es sich dabei tatsächlich um eine Semi-Entscheidungsprozedur für dieses Problem handelt.

Lösung 1.7
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-HALTEPROBLEMS, d.h.: Quellcode von 2 SIMPLE Programmen Π_1, Π_2 und Input I für Π_1 und Π_2;
- Π_i simuliert parallel die Ausführung von Π_1 auf dem Input I und von Π_2 auf dem Input I;
- wenn die Simulation eines der Programme Π_1 oder Π_2 terminiert, dann stoppt Π_i die Simulation des anderen Programms, gibt true aus und terminiert selbst;
- ansonsten wird mit der Simulation der Programme Π_1 und Π_2 fortgefahren.

Begründung: Wir argumentieren, dass das oben skizzierte Programm Π_i eine Semi-Entscheidungsprozedur für das MINDESTENS-EINS-VON-ZWEI-HALTEPROBLEM ist. Dazu unterscheiden wir 2 Fälle:

(1) Angenommen das Programm Π_i wird mit einer positiven Instanz (Π_1, Π_2, I) des MINDESTENS-EINS-VON-ZWEI-HALTEPROBLEMS aufgerufen, d.h.: zumindest eines der beiden Programme Π_1, Π_2 terminiert auf dem Input I. Daher wird auch die parallele Simulation der beiden Programme Π_1, Π_2 auf dem Input I irgendwann den Punkt erreichen, an dem mindestens eines der beiden Programme Π_1 und/oder Π_2 terminiert. Laut obiger Beschreibung stoppt dann Π_i die Simulation des anderen Prgramms, gibt true aus und terminiert selbst.

(1) Angenommen das Programm Π_i wird mit einer negativen Instanz (Π_1, Π_2, I) des MINDESTENS-EINS-VON-ZWEI-HALTEPROBLEMS aufgerufen, d.h.: weder Π_1 noch Π_2 terminiert auf dem Input I. Daher wird auch die parallele Simulation der beiden Programme Π_1, Π_2 auf dem Input I nie terminieren. Das Programm Π_i läuft in diesem Fall endlos.

In beiden Fällen ist das ein korrektes Verhalten für eine Semi-Entscheidungsprozedur.

Aufgabe 1.8
Wir betrachten nun folgendes Problem:

EINS-VON-ZWEI-HALTEPROBLEM
Instanz: Programme Π_1, Π_2 und Input String I.
Frage: Hält *genau* eines der Programme Π_1 und Π_2 auf Input I?
(d.h.: entweder Π_1 hält und Π_2 läuft endlos auf Input I oder umgekehrt).

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

Tipp: Wenn Sie von einer *positiven* Instanz (Π_1, Π_2, I) des EINS-VON-ZWEI-HALTEPROBLEMS mit Sicherheit wissen, dass eines der beiden Programme Π_1, Π_2 auf I *terminiert*, dann muss das andere Programm endlos laufen.

Lösung 1.8
Reduktion: Sei (Π, I) eine beliebige Instanz des co-HALTEPROBLEMS. Wir definieren daraus folgende Instanz (Π'_1, Π'_2, I') des EINS-VON-ZWEI-HALTEPROBLEMS mit $\Pi'_1 = \Pi$ und $I' = I$. Außerdem definieren wir Π'_2 als ein Programm von dem wir sicher wissen, dass es (auf jeder Instanz) terminiert, z.B.: Π'_2 ignoriert den Input und stoppt sofort, d.h.:
Void Π'_2 (String S) { }

Korrektheit der Reduktion: Wir müssen folgende Äquivalenz zeigen:
(Π, I) ist eine positive Instanz des co-HALTEPROBLEMS (d.h.: Π terminiert nicht auf Input I) $\Leftrightarrow$
(Π'_1, Π'_2, I') ist eine positive Instanz des EINS-VON-ZWEI-HALTEPROBLEMS (d.h.: genau eines der beiden Programme Π'_1, Π'_2 terminiert auf dem Input I').

"$\Rightarrow$" Angenommen (Π, I) ist eine positive Instanz des co-HALTEPROBLEMS, d.h.: Π terminiert nicht auf Input I. Dann terminiert auch $\Pi'_1 = \Pi$ nicht auf $I' = I$. Anderseits wissen wir, dass Π'_2 auf jeder Instanz, also auch auf I', terminiert. Daher ist (Π'_1, Π'_2, I') eine positive Instanz des EINS-VON-ZWEI-HALTEPROBLEMS.

"$\Leftarrow$" Angenommen (Π'_1, Π'_2, I') ist eine positive Instanz des EINS-VON-ZWEI-HALTEPROBLEMS, d.h.: genau eines der beiden Programme Π'_1, Π'_2 terminiert auf dem Input I'. Wir wissen aber, dass Π'_2 auf jeder Instanz, also auch auf I', terminiert. Es muss also gelten, dass $\Pi'_1 = \Pi$ auf dem Input $I' = I$ nicht terminiert. Das bedeutet, dass (Π, I) eine positive Instanz des co-HALTEPROBLEMS ist.

Aufgabe 1.9

Wir betrachten noch einmal das EINS-VON-ZWEI-HALTEPROBLEM von der vorigen Frage. Zeigen Sie mittels Reduktion vom co-HALTEPROBLEM, dass auch das co-EINS-VON-ZWEI-HALTEPROBLEM nicht semientscheidbar 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-HALTEPROBLEM reduzieren.

Tipp: Wenn Sie von einer *positiven* Instanz (Π_1, Π_2, I) des EINS-VON-ZWEI-HALTEPROBLEMS mit Sicherheit wissen, dass eines der beiden Programme Π_1, Π_2 auf I *nicht terminiert*, dann muss das andere Programm terminieren.

Lösung 1.9

Reduktion: Sei (Π, I) eine beliebige Instanz des HALTEPROBLEMS. Wir definieren daraus folgende Instanz (Π'_1, Π'_2, I') des EINS-VON-ZWEI-HALTEPROBLEMS mit $\Pi'_1 = \Pi$ und $I' = I$. Außerdem definieren wir Π'_2 als ein Programm von dem wir sicher wissen, dass es (auf jeder Instanz) endlos läuft, z.B.: Π'_2 ignoriert den Input und geht sofort in eine Endlosschleife, d.h.:

```
Void Π'_2 (String S) {
    while (true) do {} }
```

Korrektheit der Reduktion: Wir müssen folgende Äquivalenz zeigen:
(Π, I) ist eine positive Instanz des HALTEPROBLEMS (d.h.: Π terminiert auf Input I) $\Leftrightarrow$
(Π'_1, Π'_2, I') ist eine positive Instanz des EINS-VON-ZWEI-HALTEPROBLEMS (d.h.: genau eines der beiden Programme Π'_1, Π'_2 terminiert auf dem Input I').

"$\Rightarrow$" Angenommen (Π, I) ist eine positive Instanz des HALTEPROBLEMS, d.h.: Π terminiert auf Input I. Dann terminiert auch $\Pi'_1 = \Pi$ auf $I' = I$. Anderseits wissen wir, dass Π'_2 auf jeder Instanz, also auch auf I', endlos läuft. Daher ist (Π'_1, Π'_2, I') eine positive Instanz des EINS-VON-ZWEI-HALTEPROBLEMS.

"$\Leftarrow$" Angenommen (Π'_1, Π'_2, I') ist eine positive Instanz des EINS-VON-ZWEI-HALTEPROBLEMS, d.h.: genau eines der beiden Programme Π'_1, Π'_2 terminiert auf dem Input I'. Wir wissen aber, dass Π'_2 auf jeder Instanz, also auch auf I', endlos läuft. Es muss also gelten, dass $\Pi'_1 = \Pi$ auf dem Input $I' = I$ terminiert. Das bedeutet, dass (Π, I) eine positive Instanz des HALTEPROBLEMS ist.

Aufgabe 1.10

In der Vorlesung wurde folgende Reduktion von 2-FÄRBBARKEIT auf 2-SAT vorgestellt: Sei $G = (V, E)$ eine beliebige Instanz von 2-FÄRBBARKEIT, d.h.: G ist ein ungerichteter Graph mit Knotenmenge $V = \{v_1, \ldots, v_n\}$ und Kantenmenge E. Daraus definieren wir folgende Instanz φ_G von 2-SAT mit den aussagenlogischen Variablen $x_1, \ldots, x_n$:

$$\varphi_G = \bigwedge_{[v_i, v_j] \in E} \Big((x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j) \Big).$$

In der Vorlesung wurde bereits eine Richtung der Korrektheit dieser Reduktion gezeigt, nämlich:
G ist eine positive Instanz von 2-FÄRBBARKEIT $\Rightarrow$ φ_G ist eine positive Instanz von 2-SAT.

Zeigen Sie nun auch die andere Richtung der Korrektheit, d.h.

φ_G ist eine positive Instanz von 2-SAT $\Rightarrow$ G ist eine positive Instanz von 2-Färbbarkeit.

Lösung 1.10

Angenommen φ_G ist eine positive Instanz von 2-SAT. Dann gibt es eine Wahrheitsbelegung I auf den Variablen $x_1, \ldots, x_n$, die die Formel ϕ_G erfüllt. Die Formel ϕ_G ist eine Konjunktion von Teilformeln der Form $(x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j)$. Wenn I die Formel ϕ_G erfüllt, dann erfüllt I also auch jede Teilformel $(x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j)$ von ϕ_G.

Mit Hilfe von I **definieren wir** nun folgende Funktion $f\colon V \to \{0,1\}$:

$$f(v_i) = \begin{cases} 0 & \text{falls } I(x_i) = \text{false} \\ 1 & \text{falls } I(x_i) = \text{true} \end{cases}$$

Wir müssen noch zeigen, dass f eine gültige 2-Färbung von G ist, d.h.: *für jede* Kante $[v_i, v_j]$ im Graphen G, gilt $f(v_i) \neq f(v_j)$. Sei also $[v_i, v_j]$ eine *beliebige* Kante im Graphen G. **Laut Problemreduktion** enthält dann ϕ_G die Teilformel $(x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j)$. **Laut Annahme** erfülllt die Wahrheitsbelegung I die Formel ϕ_G und somit auch $(x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j)$, d.h.: sowohl $(x_i \vee x_j)$ als auch $(\neg x_i \vee \neg x_j)$ wird von I erfüllt.

Das bedeutet: Da $(x_i \vee x_j)$ von I erfüllt wird, muss *mindestens eine* der aussagenlogischen Variablen x_i, x_j in I true sein. Da auch $(\neg x_i \vee \neg x_j)$ von I erfüllt wird, muss mindestens eines der Literale $\neg x_i, \neg x_j$ in I true sein und somit mindestens eine der Variablen x_i, x_j in I false sein. In anderen Worten, es muss *höchstens eine* der aussagenlogischen Variablen x_i, x_j in I true sein. Zusammen genommen, bedeutet das, dass *exakt eine* der aussagenlogischen Variablen x_i, x_j in I true ist, d.h.: eine der Variablen x_i, x_j ist true in I und eine der Variablen x_i, x_j ist false in I.

Aufgrund der Definition von f ist also einer der Funktionswerte $f(v_i), f(v_j)$ gleich 1 und einer der Funktionswerte $f(v_i), f(v_j)$ gleich 0. Also insbesondere gilt $f(v_i) \neq f(v_j)$. Da wir diese Eigentschaft für eine *beliebige* Kante $[v_i, v_j]$ in G gezeigt haben, gilt die Eigenschaft für *alle* Kanten in G, d.h.: f ist eine gültige 2-Färbung von G.