Prof. J. Esparza
S. Sickert, J. Krämer

Technische Universität München
KEINE ABGABE

Einführung in die theoretische Informatik

Sommersemester 2017 – Übungsblatt Lösungsskizze 11

Übungsblatt

Wir unterscheiden zwischen Übungs- und Abgabeblättern. Auf diesem *Übungsblatt* finden Sie eine Übersicht über die Kernaspekte, die Sie in Kalenderwoche 28 in den Tutorien diskutieren, üben und vertiefen. Die Aufgaben auf diesem Blatt dienen dem Üben und Verstehen des Vorlesungsstoffes, sowie dem *eigenständigen Erarbeiten* der Kernaspekte. Außerdem sollen Ihnen diese Aufgaben auch helfen, ein Gefühl dafür zu bekommen, was Sie inhaltlich in der Klausur erwartet. Klausuraufgaben können jedoch deutlich von den hier gestellten Aufgaben abweichen. Abschreiben und Auswendiglernen von Lösungen wird Ihnen daher keinen dauerhaften Erfolg in der Vorlesung bringen. Fragen zu den Übungsblättern können Sie montags bis donnerstags von 12 Uhr bis 14 Uhr in der *THEO-Sprechstunde* in Raum 03.11.034 stellen.

Kernaspekte

K11.1 korrektes Wiedergeben der folgenden Definitionen und Algorithmen

- WHILE-Programme
- GOTO-Programme
- enscheidbar
- charakteristische Funktion
- spezielles Halteproblem
- allgemeines Halteproblem
- reduzierbar
- semi-entscheidbar
- rekursiv-aufzählbar

K11.2 Turing-Maschinen konstruieren, die mit einem bestimmten Bandinhalt terminieren
K11.3 Reduktionen zwischen Mengen aufstellen und ihre Korrektheit und Berechenbarkeit begründen
K11.4 begründet entscheiden, ob Aussagen zur Entscheidbarkeit bzw. Semi-Entscheidbarkeit und Berechenbarkeit korrekt sind
K11.5 begründet entscheiden, ob gegebene Beispiele neu eingeführte Definitionen erfüllen
K11.6 Aussagen mit neu eingeführten Definitionen beweisen oder widerlegen

AUFGABE 11.1. (*Entscheidbarkeit vs. Berechenbarkeit*) Stufe B

Ordnen Sie die folgenden Satzanfänge den Satzenden so zu, dass richtige Aussagen entstehen. Sei dazu $\mathsf{A}, \mathsf{B} \subseteq \{0,1\}^*$:

(a) Die Funktion χ_{A} ist berechenbar,
(b) Die Funktion χ'_{A} ist berechenbar,
(c) A ist entscheidbar,
(d) B ist nicht entscheidbar,

(i) wenn $\mathsf{A} \leq \mathsf{B}$ gilt und B entscheidbar ist.
(ii) wenn A rekursiv aufzählbar ist.
(iii) wenn A entscheidbar ist.
(iv) wenn $\mathsf{A} \leq \mathsf{B}$ gilt und A nicht entscheidbar ist.

Lösungsskizze
(a) → (iii), (b) → (ii), (c) → (i), (d) → (iv).

AUFGABE 11.2. Stufe B/C

Entscheiden Sie, ob die folgenden Behauptungen korrekt oder inkorrekt sind. Begründen Sie dann Ihre Antworten wie folgt: Wenn L entscheidbar bzw. semi-entscheidbar ist, beschreiben Sie einen Algorithmus, der die charakteristische Funktion χ_{L} bzw. die Funktion χ'_{L} berechnet. Wenn L unentscheidbar ist, leiten Sie einen Widerspruch zu einem Ergebnis der Vorlesung ab.

(a) Wenn A und B entscheidbare Sprachen sind, dann ist $\mathsf{A} \cap \mathsf{B}$ entscheidbar.
(b) Wenn A und $\mathsf{A} \cup \mathsf{B}$ entscheidbar sind, dann ist B entscheidbar.
(c) Das Problem, ob $\mathsf{L}_H(\mathsf{M}) \neq \emptyset$ für eine gegebene Turingmaschine M gilt, ist semi-entscheidbar.
(d) Das Problem, ob $\mathsf{L}_H(\mathsf{M}) = \emptyset$ für eine gegebene Turingmaschine M gilt, ist semi-entscheidbar.

Lösungsskizze

(a) Korrekt.
Sei T_A DTM, die A entscheidet, T_B DTM, die B entscheidet.
DTM zu $A \cap B$: Gegeben x, berechne $T_A(x)$. Falls $T_A(x) = 0$, lehne x ab, ansonsten berechne $T_B(x)$. Gilt $T_B(x) = 0$, lehne x ab, sonst akzeptiere x. Da T_A, T_B die jeweiligen Mengen entscheiden, terminieren beide DTM immer, womit auch die DTM zu $A \cap B$ stets mit dem korrekten Ergebnis terminiert, womit $A \cap B$ entscheidbar ist.

(b) Inkorrekt.
Sei $B = H_0 \subseteq \{0,1\}^*$ das Halteproblem auf leere Eingabe und $A = \{0,1\}^*$. Dann sind A und $A \cup B$ entscheidbar, aber B nicht.

(c) Korrekt.
Eine NTM, kann einfach ein Wort aus $\mathsf{L}(M)$ raten, soweit es existiert die TM ausführen und dann halten. Diese NTM kann von einer DTM simuliert werden und terminiert gdw. $\mathsf{L}(M)$ nicht leer ist. Andernfalls terminiert die Simulation nie.
Direkte DTM Konstruktion: Verwende einen wachsenden Zähler $i = 0, 1, \ldots$ und simuliere $\mathsf{M}[w]$ für i Schritte und für alle $w \in \Sigma^*$ mit $|w| \leq i$. Falls irgendeine $\mathsf{M}[w]$ hält, dann stoppe Simulation und gebe "1" aus.

(d) Inkorrekt.
Kann nicht semi-entscheidbar sein, sonst wäre $\mathsf{L}_F(M) \neq \emptyset$ entscheidbar, womit das Halteproblem auf leere Eingabe entscheidbar wäre mittels der Reduktion: Bilde TM-Codierung w auf Codierung w' der TM:

> "Falls $x \neq \varepsilon$, lehne x ab, sonst simuliere $M_w(\varepsilon)$ bis sie terminiert hat, und akzeptiere dann ε"

Somit akzeptiert $M_{w'}$ höchstens ε und das genau dann, wenn M_w auf ε terminiert.

AUFGABE 11.3. (*Church-Turing-These*)
Stufe B

Wenn es darum geht zu beweisen, dass etwas entscheidbar oder unentscheidbar ist, verlangen wir von Ihnen – außer es ist explizit anders angegeben – nie die formale Definition einer Turing-Maschine unter Angabe des Zustandsraumes und der Transitionsrelation, sondern die Angabe eines Algorithmus, aus dem man eine Turing-Maschine konstruieren könnte. Ihre Beschreibung soll dabei so konkret sein, dass es kein Missverständnis über die Funktionsweise der Turing-Maschine geben kann. *Betrachten Sie die Beispiele in Aufgabe 11.7 und 11.8 sowie die Aufgabenstellung von Aufgabe 11.2. Diskutieren Sie, welche Besonderheiten Ihnen in unseren Algorithmen auffallen, insbesondere wie wir sicherstellen, dass die Funktionsweise der beschriebenen Turing-Maschine klar wird. Erklären Sie dann mithilfe der Church-Turing-These, warum wir davon ausgehen, dass wenn man so einen Algorithmus beschreiben kann, es auch eine passende Turing-Maschine gibt.*

AUFGABE 11.4. (*Länge von Wörtern*)
Stufe B

Diskutieren Sie, wie viele Schritte eine Turing-Maschine mindestens machen muss, um zu entscheiden, ob für eine Eingabe w gilt: $|w| \geq 314$.

AUFGABE 11.5.
Stufe C

Geben Sie eine nichtdeterministische 1-Band-TM mit $\Sigma = \{\mathsf{a}, \mathsf{b}\}$ an, die nichtdeterministisch die Eingabe permutiert. Ist w die Eingabe, dann terminiert die TM auf *jedem* Rechenpfad, und zu jeder Permutation w' von w gibt es mindestens eine Rechnung der TM, an deren Ende der Bandinhalt gerade w' ist.
Beispiel: Auf Eingabe $w = \mathsf{aabb}$ soll die TM für jede der möglichen Permutation $w' \in \{\mathsf{abab}, \mathsf{abba}, \mathsf{baba}, \mathsf{bbaa}, \mathsf{baab}, \mathsf{aabb}\}$ jeweils mindestens einen Rechenpfad besitzen, an deren Ende genau w' auf dem Band steht.

Lösungsskizze

- Merke ersten Buchstaben in Kontrolle
$$p \xrightarrow[x \in \Sigma]{x/x,N} p_x$$
- Laufe nicht deterministich zu einem beliebigen Buchstaben (immer nach rechts, nicht deterministisch Stopp samt Zustandswechsel), falls rechtes Ende erreicht wird, Terminieren samt Löschen von Markierungen
$$p_x \xrightarrow[y \in \Sigma]{y/y,R} p_x \qquad p_x \xrightarrow[y \in \Sigma]{y'/y',R} p_x \qquad p_x \xrightarrow[y \in \Sigma]{y/y,N} q_x \qquad p_x \xrightarrow{\square/\square,L} s$$
$$s \xrightarrow[y \in \Sigma]{y'/y,L} s \qquad s \xrightarrow[y \in \Sigma]{y/y,L} s \qquad s \xrightarrow{\square/\square,R} s$$
- Solange der aktuelle Buchstabe nicht markiert ist:
Schreibe gespeicherten Buchstabe an aktuelle Position samt Markierung und merke überschriebenen Buchstaben in Kontrolle
$$q_x \xrightarrow[y \in \Sigma]{y/x',L} r_y$$

laufe nach ganz links, suche nicht deterministisch nach einem noch nicht markierten Buchstaben

$$r_x \xrightarrow[y\in\Sigma]{\substack{y'/y',L\\ y/y,L}} r_x \qquad r_x \xrightarrow{\square/\square,R} p_x$$

und wiederhole Vorgang.

AUFGABE 11.6. Stufe C

Geben Sie für jede der folgenden Funktionen sowohl ein WHILE-Programm als auch ein GOTO-Programm an, das die jeweilige Funktion implementiert:

(a) $f(x_1, x_2) = x_1 \bmod x_2$

(b) $g(x_1) = \begin{cases} \max\{k \in \mathbb{N}_0 \mid \frac{x_1}{2^k} \in \mathbb{N}_0\} & \text{falls } x_1 > 0 \\ \bot & \text{sonst} \end{cases}$

(c) $h(x_1, x_2) = (2x_1 + 1) \cdot 2^{x_2}$

Hinweise:

- Verwenden Sie für die (a) nur das Grundschema für WHILE- und GOTO-Programme und folgende Abkürzungen: $x_i := x_j + x_k$, $x_i := x_j - x_k$, $x_i := n$, $x_i := x_j$.
- Halten Sie sich an die Konventionen aus der Vorlesung: Soll ein WHILE-Programm eine Funktion $F\colon \mathbb{N}_0^k \to \mathbb{N}_0$ berechnen, so werden die Variablen $x_1, \ldots, x_k$ entsprechend mit den Eingabewerten $n_1, \ldots, n_k \in \mathbb{N}_0$ initialisiert, während alle anderen Variablen zu Beginn auf 0 initialisiert werden. Nach Terminierung speichert die Variable x_0 den Funktionswert $F(n_1, \ldots, n_k)$.

Lösungsskizze

(a) Zur besseren Lesbarkeit verwenden wir folgende Abkürzungen: x für x_1, y für x_2 und z für x_0.

- WHILE-Programm:

```
z := x - y;
WHILE z ≠ 0 DO
  x := z + 0;
  z := z - y
END;
z := x + 1;
z := z - y;
IF z = 0 DO
  z := x
ELSE
  z := 0
END
```

- GOTO-Programm:

```
      z := x - y;
M1:   IF z = 0 GOTO M2;
      x := z + 0;
      z := z - y;
      GOTO M1;
M2:   z := x + 1;
      z := z - y;
      IF z = 0 GOTO M3;
      z := 0;
      HALT;
M3:   z := x
      HALT;
```

(b) Ab jetzt verwenden wird MOD als zusätzlichen Befehl, sowie den syntaktischen Zucker der Folien. Wieder verwenden wir zu besseren Lesbarkeit Buchstaben als Variablenbezeichner wie in (a) und weitere Kleinbuchstaben für sonstige Hilfsvariaben (nach Konvention zu Beginn mit 0 initialisiert), denen dann den Bezeichnern x_i zu geordnet werden.
Da ein GOTO-Programm durch einfache Substitution aus einem WHILE-Programmen abgeleitet werden kann, geben wir nur noch die WHILE-Programme an.

- WHILE-Programm:

```
z := 1;
y := 2;
b := 1;
WHILE b ≠ 0 DO
  u := x MOD y;
  IF u = 0 DO //y = 2^z mit x/2^z ∈ N, teste, ob auch x/2^(z+1) ∈ N
    z := z + 1;
    u := y; // Konvention aus Folien: x_i := x_j + x_k nur fuer i ≠ k
        gestattet.
    y := u + u;
  ELSE //x/2^z ∉ N, aber x/2^(z-1) ∈ N
    b := 0; //beende While-Schleife
    z := z - 1;
  END
END
```

(c) Kommentare wie bei (b).

- WHILE-Programm:

```
1  z := y + y;
2  z := z + 1;
3  y := 1;
4  WHILE x ≠ 0 DO
5     u := y;
6     y := u + u;
7     x := x - 1
8  END;
9  u := z;
10 z := y * u;
```

AUFGABE 11.7. (*Entscheidbarkeit*) Stufe C

In dieser Aufgabe sollen Sie zeigen, dass die angegebenen Probleme entscheidbar bzw. semi-entscheidbar sind. Dazu sollen Sie jeweils eine Turing-Maschine beschreiben, die das Problem löst.

Beispiel:

Es ist entscheidbar, ob eine deterministische Turing-Maschine für irgendeine Eingabe mehr als 314 Schritte macht.

Wir führen die Turing-Maschine nacheinander auf allen Wörtern $w \in \Sigma^*$ mit $|w| \leq 315$ für höchstens 315 Schritte aus. Wenn wir 315 Schritte simuliert haben, halten wir und geben "ja" aus, ansonsten halten wir, sobald wir die Turing-Maschine auf allen Wörtern w wie oben angegeben simuliert haben, und geben "nein" aus.

Korrektheit des Algorithmus: Es gibt nur endlich viele Wörter bis Länge maximal 315. Wir begründen zunächst durch einen Widerspruchsbeweis, dass es genügt, alle Wörter bis Länge 315 zu betrachten. Angenommen, wir entscheiden uns falsch, d.h. es gibt eine Eingabe $w \in \Sigma^*$ mit $|w| > 315$, auf die die Turing-Maschine mindestens 315 Schritte macht, aber für alle w' mit $|w'| \leq 315$ macht sie maximal 314 Schritte. Betrachte v mit $|v| = 315$ und es gibt v', so dass $vv' = w$. Die Turing-Maschine hält für v nach maximal 314 Schritten. Da sie in 314 Schritten maximal 314 Felder lesen kann, kann sie nicht die gesamte Eingabe v lesen. Da die Turing-Maschine deterministisch ist, wird sie alles nach 314 Feldern ignorieren und so insbesondere v und w gleich behandeln. Widerspruch.

(a) Es ist entscheidbar, ob eine Turing-Maschine mehr als 314 Zustände hat.
(b) Es ist entscheidbar, ob eine Turing-Maschine bei Eingabe ε mehr als 314 Schritte macht.
(c) Es ist entscheidbar, ob eine Turing-Maschine auf Eingabe ε ihren Kopf mehr als 314 Felder von der Startposition entfernen kann.

Lösungsskizze

(a)
- *Algorithmus:* Decodiere bzw. lese die TM ein und teste $|\mathsf{Q}| > 314$ unter Verwendung eines Zählers. Gebe "Ja" aus falls dies der Fall ist, sonst "Nein".
- *Korrektheit:* Unter der Annahme der Korrektheit der Dekodierung hat die TM mehr als 314 Zustände.

(b)
- *Algorithmus:* Decodiere und simuliere die TM M auf der Eingabe ε für 315 Schritte. Falls sie bis Schritt 314 nicht hält, gebe "Ja" aus, sonst "Nein".
- *Korrektheit:* Der Algorithmus terminiert spätestens nach 315 simulierten Schritten. Falls die Simulation vorher terminiert, dann terminiert auch die gegebene TM nach maximal 314 Schritten. Falls die Simulation Schritt 315 erreicht, dann läuft auch die TM M mehr als 314 Schritte auf der Eingabe ε.

(c)
- *Algorithmus:* Decodiere und simuliere die TM M auf der Eingabe ε auf einem endlichen Band, das in beide Richtungen 315 Felder hat. Falls M den Kopf auf einen der Randfelder des simulierten Bands bewegt, breche ab und gebe "Ja" aus. Falls sich eine Konfiguration wiederholt bzw. M hält, breche ab und gebe "Nein" aus.
- *Korrektheit:* Die Simulation terminiert immer, da es nur endlich viele erreichbare Konfigurationen auf dem endlichen Band gibt. Falls die Simulation "Ja" antwortet, dann bewegt die TM auch den Kopf soweit von der Startposition weg. Falls die Simulation mit "Nein" antwortet, hat der Algorithmus eine Schleife in den Konfigurationen entdeckt, die immer wieder wiederholt wird, oder die TM hat gehalten. In beiden Fälle hat aber die TM das endliche Band nicht verlassen.
- *Hinweis:* Diese Idee lässt sich auch auf NTMs anwenden, in dem alle erreichbaren Konfiguration auf dem endlichen Band berechnet werden. Dies ist möglich, da diese Menge endlich ist.

AUFGABE 11.8. (*Reduktionen und Unentscheidbarkeit*) Stufe C

In dieser Aufgabe sollen Sie zeigen, dass das angegebene Problem unentscheidbar ist, indem Sie eine passende Reduktion auf ein unentscheidbares Problem angeben.

Beispiel:

Es ist unentscheidbar, ob eine Turing-Maschine für alle Eingaben 0 *ausgibt.* Formal heißt dies, dass wir zeigen wollen, dass $V_0 \coloneqq \{\mathsf{enc}(\mathsf{M}) \mid \forall \mathsf{x} \in \Sigma^*.\ \varphi_{\mathsf{enc}(\mathsf{M})}(\mathsf{x}) = 0\}$ unentscheidbar ist.

Reduktion von $\overline{\mathsf{K}}$*:* Wir konstruieren für ein gegebenes Encoding $\mathsf{enc}(\mathsf{M})$ eine Turing-Maschine N_M mit Eingabe $\mathsf{x} \in \{0,1\}^*$ wie folgt: Wir simulieren die Turing-Maschine M auf $\mathsf{enc}(\mathsf{M})$ für $|\mathsf{x}|$ viele Schritte. Falls M nicht innerhalb von $|\mathsf{x}|$ vielen Schritten hält, dann geben wir 0 aus, ansonsten 1.

Die Reduktion ist berechenbar, weil wir das Encoding von Turing-Maschinen berechnen können und Turing-Maschinen anhand ihres Encodings für eine begrenzte Anzahl an Schritten simulieren können.

Die Reduktion ist korrekt, weil... Angenommen, $\mathsf{enc}(\mathsf{M}) \in \overline{\mathsf{K}}$. Dann hält die Simulation der Turing-Maschine M auf Eingabe $\mathsf{enc}(\mathsf{M})$ während der Ausführung der Turing-Maschine N_M für keine Eingabe x. Das heißt, die Turing-Maschine N_M gibt immer 0 aus.

Angenommen, $\mathsf{enc}(\mathsf{M}) \in \mathsf{K}$. Dann gibt es eine natürliche Zahl t, so dass die Simulation von M in t Schritten auf $\mathsf{enc}(\mathsf{M})$ hält. Dann gibt die Turing-Maschine N_M den Wert 1 für alle Eingaben der Länge größer oder gleich t aus.

Damit ist die folgende Funktion eine Reduktion von $\overline{\mathsf{K}}$ auf V_o: Sei $y \in \overline{V_0}$.

$$i \mapsto \begin{cases} \mathsf{enc}(\mathsf{N}_\mathsf{M}) & \text{falls } i = \mathsf{enc}(\mathsf{M}) \text{ für eine Turing-Maschine } \mathsf{M} \\ y & \text{ansonsten} \end{cases}$$

Zeigen Sie: Es ist unentscheidbar zu prüfen, ob für zwei als Eingabe gegebene Turing-Maschinen die eine auf allen Eingaben genau dann hält, wenn die andere nicht hält.
Zeigen Sie auch, dass dieses Problem nicht semi-entscheidbar ist.

Lösungsskizze
Wir bezeichnen die Menge des Entscheidungsproblems: $\mathsf{H}_{\mathsf{NEQ}}$

Reduktion von H_0*:* Sei $\mathsf{M}_\perp$ die Turingmaschine, die auf keiner Eingaben hält. Sei nun M eine beliebige TM. Wir konstruieren nun folgende Maschine M': Ignoriere Eingabe w und lösche das Band. Führe dann $\mathsf{M}[\varepsilon]$ aus.

Die Reduktion ist berechenbar, weil wir das Encoding von Turing-Maschinen decodieren und eine Turing-Maschine dann simulieren. können. Außerdem können wir alle Zeichen auf dem Band, die nicht dem Leerzeichen entsprechen, überschreiben.

Die Reduktion ist korrekt: Angenommen, $\mathsf{enc}(\mathsf{M}) \in \mathsf{H}_0$. Dann hält M' auf alle Eingaben und da $\mathsf{M}_\perp$ auf alle Eingaben divergiert, ist $\mathsf{enc}(\mathsf{M}')\#\mathsf{enc}(\mathsf{M}_\perp) \in \mathsf{H}_{\mathsf{NEQ}}$.

Umgekehrt: Angenommen $\mathsf{enc}(\mathsf{M}) \notin \mathsf{H}_0$. Dann divergiert die Turing-Maschine für alle Eingaben, genauso auch die Turing-Maschine $\mathsf{M}_\perp$, also ist $\mathsf{enc}(\mathsf{M}')\#\mathsf{enc}(\mathsf{M}_\perp) \notin \mathsf{H}_{\mathsf{NEQ}}$.

Damit ist die folgende Funktion eine Reduktion von H_0 auf $\mathsf{H}_{\mathsf{NEQ}}$. Sei dafür $y \in \overline{\mathsf{H}_{\mathsf{NEQ}}}$ beliebig:

$$i \mapsto \begin{cases} \mathsf{enc}(\mathsf{M}')\#\mathsf{enc}(\mathsf{M}_\perp) & \text{falls } i = \mathsf{enc}(\mathsf{M}) \text{ für eine Turing-Maschine } \mathsf{M} \\ y & \text{ansonsten} \end{cases}$$

Bis zu diesem Punkt haben wir nur gezeigt, dass $\mathsf{H}_{\mathsf{NEQ}}$ nicht entscheidbar (= unentscheidbar ist). Mit dieser Reduktion von H_0 können wir aber nichts darüber aussagen, ob das Problem semi-entscheidbar ist oder nicht. Die folgende Reduktion zeigt nun, dass das Problem *nicht* semi-entscheidbar ist.

Reduktion von $\overline{\mathsf{H}_0}$*:* Sei $\mathsf{M}_\top$ die Turingmaschine, die auf allen Eingaben hält. Sei nun M eine beliebige TM. Wir konstruieren nun folgende Maschine M': Ignoriere Eingabe w und lösche das Band. Führe dann $\mathsf{M}[\varepsilon]$ aus.

Die Reduktion ist berechenbar, weil wir das Encoding von Turing-Maschinen decodieren und eine Turing-Maschine dann simulieren. können. Außerdem können wir alle Zeichen auf dem Band, die nicht dem Leerzeichen entsprechen, überschreiben.

Die Reduktion ist korrekt: Angenommen, $\mathsf{enc}(\mathsf{M}) \in \overline{\mathsf{H}_0}$. Dann divergiert M' auf alle Eingaben und da $\mathsf{M}_\top$ auf alle Eingaben hält, ist $\mathsf{enc}(\mathsf{M}')\#\mathsf{enc}(\mathsf{M}_\top) \in \mathsf{H}_{\mathsf{NEQ}}$.

Umgekehrt: Angenommen $\mathsf{enc}(\mathsf{M}) \notin \overline{\mathsf{H}_0}$. Dann hält die Turing-Maschine für alle Eingaben, genauso auch die Turing-Maschine $\mathsf{M}_\top$, also ist $\mathsf{enc}(\mathsf{M}')\#\mathsf{enc}(\mathsf{M}_\top) \notin \mathsf{H}_{\mathsf{NEQ}}$.

Damit ist die folgende Funktion eine Reduktion von $\overline{\mathsf{H}_0}$ auf $\mathsf{H}_{\mathsf{NEQ}}$. Sei dafür $y \in \overline{\mathsf{H}_{\mathsf{NEQ}}}$ beliebig:

$$i \mapsto \begin{cases} \mathsf{enc}(\mathsf{M}')\#\mathsf{enc}(\mathsf{M}_\top) & \text{falls } i = \mathsf{enc}(\mathsf{M}) \text{ für eine Turing-Maschine } \mathsf{M} \\ y & \text{ansonsten} \end{cases}$$

Stufe D

AUFGABE 11.9.
Wir erweitern GOTO-Programme um die Möglichkeit, Variablenwerte auf einem globalen Stack zwischenzuspeichern. Mittels der Anweisung **PUSH** x_i wird der aktuelle Wert der Variable x_i auf den Stack gelegt, mittels $x_i :=$ **POP** wird der aktuell oberste Wert vom Stack genommen und in der Variable x_i gespeichert – sollte der Stack aktuell keine Werte enthalten, wird x_i auf 0 gesetzt. Ob der Stack aktuell einen Wert enthält, kann mittels $x_i :=$ **EMPTY** erfragt werden, wobei x_i auf 1 gesetzt wird, falls der Stack leer ist, ansonsten wird x_i auf 0 gesetzt.
Beispiel:

```
1  M1: IF x1 = 0 GOTO M2;
2      IF x1 = 1 GOTO M3;
3      x1 := x1 - 1;
4      PUSH x1;
5      x1 := x1 - 1;
6      GOTO M1;
7  M2:  x0 := x0 + 0;
8      GOTO M4;
9  M3:  x0 := x0 + 1;
10     GOTO M4;
11 M4:  x1 := EMPTY;
12     IF x1 = 1 GOTO M5;
13     x1 := POP;
14     GOTO M1;
15 M5: HALT
```

(a) Bestimmen Sie die Funktion $\mathbb{N}_0 \to \mathbb{N}_0$, die von Programm aus dem Beispiel berechnet wird.
(b) Skizzieren Sie, wie sich jedes GOTO-Programm mit Stack in ein GOTO-Programm ohne Stack übersetzen lässt.

Lösungsskizze

(a) Fibonacci-Zahlen mit $F_0 = 0, F_1 = 1, F_{n+2} = F_{n+1} + F_n$
Beweisskizze (Nicht gefordert von der Aufgabe, nur zum Verständnis!):
- Induktionsanfang:
  Rechenpfad zu $x_1 = 0$: $M_1 \to M_2 \to M_4 \to M_5$ mit $x_0 = 0$ (nach Konvention alle Variablen außer Eingabevariablen auf 0 initialisiert).
  Rechenpfad zu $x_1 = 1$: $M_1 \to M_3 \to M_4 \to M_5$ mit $x_0 = 1$
- Induktionsschritt: $n \geq 2$ beliebig fixiert.
  Annahme: Programm berechnet F_k für alle $k < n$.
  Erster Durchlauf $M_1 \to M_1$: $n-1$ wird auf Stack gepushed, es gilt $x_1 = n-2$ bei zweitem Besuch von M_1.
  Betrachte Berechnung β_{n-2} von Programm auf Eingabe $n-2$. Nach Annahme terminiert Programm, d.h. erreicht M_5 mit $x_0 = F_{n-2}$. Sei β'_{n-2} der Präfix von β_{n-2} bis einschließlich dem letzten Test, ob der Stack leer ist.
  In der Berechnung zu n wird nach einmaliger Schleife $M_1 \to M_1$ die Berechnung β'_{n-2} ausgeführt. Im Gegensatz zu β_{n-2} ist der Stack nicht mehr leer, weswegen statt dem Sprung zu M_5 der Stack gepushed und damit x_1 auf $n-1$ gesetzt wird, bevor wieder zu M_1 gesprungen wird. Danach ist man in der Situation einer Berechnung zur Eingabe $n-1$ bei anfänglich leerem Stack mit dem einzigen Unterschied, dass x_0 bereits den Wert F_{n-2} hat.
  Da das Programm stets nur 0 oder 1 zu dem Wert von x_0 hinzuaddiert, folgt, dass der Durchlauf zu $n-1$ dann den Wert von x_0 noch zusätzlich um F_{n-1} erhöht, womit dann bei Terminierung x_0 den Wert $F_n = F_{n-1} + F_{n-2}$ enthält.

(b) Simulieren von Stack in einer Variablen z.B. mittels Iterieren von Cantorscher Paarungsfunktion oder mit Funktionen aus der vorhergehende Aufgaben. Der leere Stack wird mit 0 encodiert und wir nehmen an, dass er in der Variable s gespeichert ist. Weiterhin bezeichnen wir mit i die Eingabe und mit o die Ausgabe, falls vorhanden. Die Operation werden dann wie folgt implementiert:
- EMPTY: `IF s = 0 DO o := 1; ELSE o := 0; END`
- PUSH: `s:= h(i,s)`
- (TOP: $o := \frac{s}{2^{g(s)}}$)
- POP: $o := \frac{s}{2^{g(s)}}$; $s := g(s)$

Beispiel:
- $s = 0$ zu Beginn
- Push 3: $s = 7 \cdot 2^0 = 7$
- Push 2: $s = 5 \cdot 2^7 = 640$
- Top: $\frac{640}{2^7} = 5$
- Pop: $s = 7$

- Pop: $s = 0$