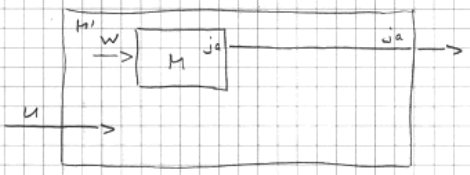
Aufgabe 1

a,) Wir beweisen die Unentscheidbarkeit des Leerheitsproblems für TM, indem wir das Wortproblem auf das Leerheitsproblem reduzieren:

Für eine TM M und eine Eingabe w kann man die TM M' konstruieren:



Die Eingabe u wird von M' ignoriert

M' akzeptiert die leere Sprache, falls $w \notin L(M)$. M' akzeptiert Σ^* falls $w \in L(M)$.

Wenn wir also für M' das Leerheitsproblem entscheiden können, können wir das Wortproblem $w \in L(M)$ entscheiden. Das Wortproblem ist aber nicht entscheidbar ↯

b,) Ebenso können wir über die TM M' das Wortproblem $\varepsilon \in L(M)$ entscheiden, wenn wir für M' das Totalitätsproblem entscheiden können.

Aufgabe 1 (Forts.)

c,) Wir zeigen, daß wir das Wortproblem für TM entscheiden können, wenn wir das Disjunktheitsproblem entscheiden können. Dazu zeigen wir, daß wir eine akzeptierenden Berechnungen TM M für Wort w als Durchschnitt zweier kontextfreier Sprachen beschreiben können. Wenn wir entscheiden könnten, ob dieser Durchschnitt leer ist (Disjunktheit), könnten wir damit das Wortproblem $w \in L(M)$ entscheiden.

Gemäß Übungsblatt 11 Aufgabe 2 gibt es für jede TM M eine kf. Grammatik G_M^1 mit $L(G_M^1) = \{ \gamma * \overleftarrow{\gamma'} \mid \gamma \vdash_M \gamma' \} = L_M^1$ und eine kf. Grammatik G_M^2 $L(G_M^2) = \{ \overleftarrow{\gamma} * \gamma' \mid \gamma \vdash_M \gamma' \} = L_M^2$

Da kontextfreie Sprachen unter Konkatenation, Kleene'schen Hüllenbildung und Vereinigung abgeschlossen sind, sind auch die Sprachen

$$L_M^g = (L_M^1 * *)^* \circ (L_M^1 \cup L_K)$$

(gerade Anzahl / ungerade Anzahl } von Konfig.)

und $L_M^u = L_K \circ (\# \circ L_M^2)^* \circ (\# \circ L_K \cup \varepsilon)$

(gerade Anzahl ungerade Anzahl von Konfig.)

wobei $L_K = \Gamma^* Q \Gamma^*$ (die Menge aller Konfigurationen der TM)

Aufgabe 1c. (Forts.)

Es gilt $\gamma_1 * \overleftarrow{\gamma_2} * \gamma_3 * \overleftarrow{\gamma_4} * \ldots \in L_M^s$

genau dann, wenn $\gamma_{2i+1} \vdash_\pi \gamma_{2i+2}$ und $(i \in \mathbb{N})$

$\gamma_1 * \overleftarrow{\gamma_2} * \gamma_3 * \overleftarrow{\gamma_4} * \ldots \in L_M^u$

genau dann, wenn $\overleftarrow{\gamma_{2i}} \vdash_\pi \gamma_{2i+1}$. $(i \in \mathbb{N}, i \geq 1)$

$L_1 = (\overbrace{q_0 w \circ (\# L_K)^*}^{\text{Anfangskonfiguration}}) \cap L_M^s$ und

$L_2 = L_M^u \cap ((L_K \#)^* \circ L_E)$ mit $L_E = \overbrace{\Gamma^* F \Gamma^*}^{\text{Menge aller akzept. Endkonfigurationen der TM}}$

sind wegen des Abschlusses von kf. Sprachen unter Schnitt mit regulären Sprachen ebenfalls kontextfrei, und es gilt

$w \in L(M)$ gdw $L_1 \cap L_2 \neq \emptyset$

(Menge aller akzeptierenden Berechnungen für w)

Da alle angewendeten Abschlußeigenschaften effektiv waren, können wir aus G_M^s und G_M^u zwei Grammatiken G_1 und G_2 mit $L(G_1) = L_1$ und $L(G_2) = L_2$ konstruieren.

Wenn wir $L(G_1) \cap L(G_2) = \emptyset$ entscheiden könnten, könnten wir das Wortproblem $w \in L(M)$ entscheiden ↯

Aufgabe 2

"⇒" Wir zeigen, daß es für jede entscheidbare Sprache L einen Generator gibt, der die Wörter der Sprache in kanonischer Reihenfolge aufzählt:
Da L entscheidbar ist, gibt es eine TM M, die auf jeder Eingabe hält und für die gilt $L(M) = L$.

Wir konstruieren aus M den Generator für L wie folgt:

- Der Generator generiert systematisch alle Wörter in der kanonischen Ordnung.
- Für jedes Wort startet der Generator die TM M (die in jedem Falle hält). Wenn M das Wort akzeptiert, gibt der Generator es aus; wenn nicht, wird nichts ausgegeben.
  Dann generiert der Generator das nächste Wort und beginnt von vorn

Offensichtlich zählt der Generator die Wörter der Sprache in der kanonischen Ordnung auf.

Aufgabe 2 (Forts.)

"⇐" Wir zeigen, daß es für jeden Generator, der die Wörter der Sprache L in der kanonischen Ordnung aufzählt, eine TM M gibt, die auf jeder Eingabe hält und für die gilt $L(M) = L$.

Wir unterscheiden zwei Fälle:
1., $|L|$ ist endlich
2., $|L|$ ist unendlich

ad., 1., Wenn $|L|$ endlich ist, gibt es offensichtlich eine TM M, die L akzeptiert (es gibt sogar einen endlichen Automaten) und auf jeder Eingabe hält.

ad., 2., Wenn $|L|$ unendlich ist, konstruieren wir die TM M mit Hilfe des Generators wie folgt:

- Wir starten den Generator und vergleichen jedes ausgegebene Wort mit der Eingabe:
  - Wenn die Wörter übereinstimmen, geben wir "ja" aus.
  - Wenn das ausgegebene Wort kleiner als die Eingabe ist, wird das nächste Wort des Generators generiert.

Aufgabe 2 (Forts.)

  - Wenn das ausgegebene Wort größer (gemäß der kan. Ordnung) ist als die Eingabe, geben wir "nein" aus.

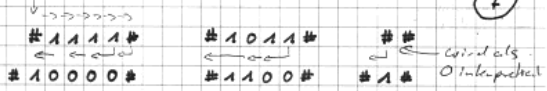
- Da L unendlich ist, kommt die konstruierte Maschine irgendwann mit ja, oder es wird vom Generator irgendwann ein Wort ausgegeben, das größer als die Eingabe ist; in diesem Fall terminiert die konstruierte TM mit "nein". → da der Generator die Wörter in kan. Reihenf. ausgibt, wird das Wort der Eingabe nie mehr kommen

Die konstruierte TM hält also auf jeder Eingabe und gibt genau dann "ja" aus, wenn der Generator das Wort der Eingabe ausgibt.

Bemerkung: Achtung unsere Konstruktion ist nicht effektiv, da wir für den Generator nicht entscheiden können, ob $|L| < \omega$ oder $|L| = \omega$ gilt!

Aufgabe 3

a) Idee:



$M_1 = (Q_1, \Sigma, \Gamma, \delta_1, q_0, B, \{q_6\})$

$Q_1 = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$ ist bei Berechnung von Funktionen nicht wirklich nötig

$\Sigma = \{0, 1, \#\}$

$\Gamma = \{0, 1, \#, B\}$

$\delta_1 = \{$

$((q_0, \#), (q_1, \#, r))$, Start

$((q_1, 0), (q_1, 0, r))$, } rechtes Ende suchen

$((q_1, 1), (q_1, 1, r))$, }

$((q_1, \#), (q_2, \#, l))$, rechtes Ende gefunden

$((q_2, 0), (q_3, 1, l))$, inkrementieren: q_3 → linkes Ende

$((q_2, 1), (q_2, 0, l))$, weiter bei höhren Bits

$((q_2, \#), (q_4, 1, l))$, linkes # mit 1 überschreiben

$((q_4, B), (q_5, \#, r))$, und links davon # ergänzen, dann Kopf positionieren: q_5

$((q_3, 0), (q_3, 0, l))$, } linkes Ende suchen

$((q_3, 1), (q_3, 1, l))$, }

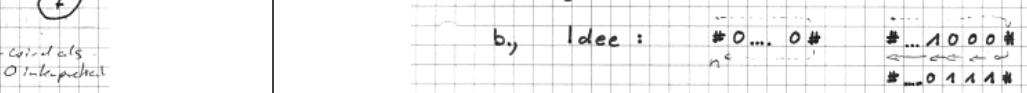
$((q_3, \#), (q_5, \#, r))$, linkes Ende gefunden

$((q_5, 0), (q_6, 0, l))$, } und Kopf auf # positionieren

$((q_5, 1), (q_6, 1, l))$ }

Aufgabe 3 (Forts.)

b) Idee:



$M_2 = (Q_2, \Sigma, \Gamma, \delta_2, p_0, B, \{p_6, n\})$

$Q_2 = \{p_0, p_1, p_2, p_3, p_4, p_5, p_6, n\}$ ← Sonderzustand für nicht dekrementierbar

$\Sigma = \{0, 1, \#\}$

$\Gamma = \{0, 1, \#, B\}$

$\delta_2 = \{$

$((p_0, \#), (p_1, \#, r))$, Start

$((p_1, 0), (p_1, 0, r))$, führende 0en überspringen

$((p_1, \#), (n, \#, r))$, Eingabe ist 0

$((p_1, 1), (p_2, 1, r))$, Eingabe ≠ 0 → p_2

$((p_2, 0), (p_2, 0, r))$, } rechtes Ende suchen

$((p_2, 1), (p_2, 1, r))$, }

$((p_2, \#), (p_3, \#, l))$, rechtes Ende gefunden

$((p_3, 1), (p_4, 0, l))$, Dekrementieren → linkes Ende gefunden

$((p_3, 0), (p_3, 1, l))$, weiter bei höhren Bits

$((p_4, 0), (p_4, 0, l))$, } linkes Ende suchen

$((p_4, 1), (p_4, 1, l))$, }

$((p_4, \#), (p_5, \#, r))$, linkes Ende gefunden

$((p_5, 0), (p_6, 0, l))$, } und Kopf positionieren

$((p_5, 1), (p_6, 1, l))$ }

Aufgabe 3 (Forts.)

c.) Idee: # 1 0 1 1 0 1 # 0 1 1 0 0 1 #

(first block: x, second block: y)

Dekrementiere y (falls möglich)

Inkrementiere x

wenn nicht möglich → lösche y

Zur Realisierung benutzen wir die Zustände und Übergangsrelationen von M_1 und M_2

$M_3 = (Q_3, \Sigma, \Gamma, \delta_3, r_0, B, \{r_7\})$ (wie gehabt: Σ, Γ)

$Q_3 = Q_1 \cup Q_2 \cup \{r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$

$\delta_3 = \delta_1 \cup \delta_2 \cup$

$\{ ((r_0, \#), (r_1, \#, r)),$ — Start

$((r_1, 0), (r_1, 0, r)),$
$((r_1, 1), (r_1, 1, r)),$ — Suche mittleres #

$((r_1, \#), (p_1, \#, r)),$ — gefunden: Starte M_2 dec.

$((p_6, \#), (r_2, \#, l)),$ — Dec erfolgreich beendet

$((r_2, 0), (r_2, 0, l)),$
$((r_2, 1), (r_2, 1, l)),$ — Suche linkes #

$((r_2, \#), (q_1, \#, r)),$ — gefunden: Starte M_1: Inc

$((q_6, \#), (r_1, \#, r)),$ — Inc erfolgreich beendet und nun wieder von vorn.

Aufgabe 3c. (Forts.)

$((n, B), (r_3, B, l)),$ — Dec: war nicht möglich ($y=0$)

$((r_3, \#), (r_4, B, l)),$
$((r_4, 0), (r_4, B, l)),$ — löschen von y

$((r_4, \#), (r_5, \#, l)),$ — löschen von y beendet

$((r_5, 0), (r_5, 0, l)),$
$((r_5, 1), (r_5, 1, l)),$ — linkes Ende suchen

$((r_5, \#), (r_6, \#, l)),$ — linkes Ende gefunden

$((r_6, B), (r_7, B, r)) \}$ — und Kopf positionieren

Bem.:
- Man hätte M_3 viel effizienter implementieren können
- Man hätte in M_3 mehr aus δ_1 und δ_2 benutzen können; dann wäre die Maschine aber unübersichtlicher gewesen.

Die Multiplikation kann man wie die Addition realisieren, indem man das erste Argument sooft auf sich selbst addiert, wie das zweite Argument angibt. Man muß dazu allerdings das erste Argument kopieren, bevor man mit der Addition beginnt.

Konvertierung kann auch mit Hilfe der Dekrementierung realisiert werden; nach jedem Dekrementieren wird ein Strich produziert.