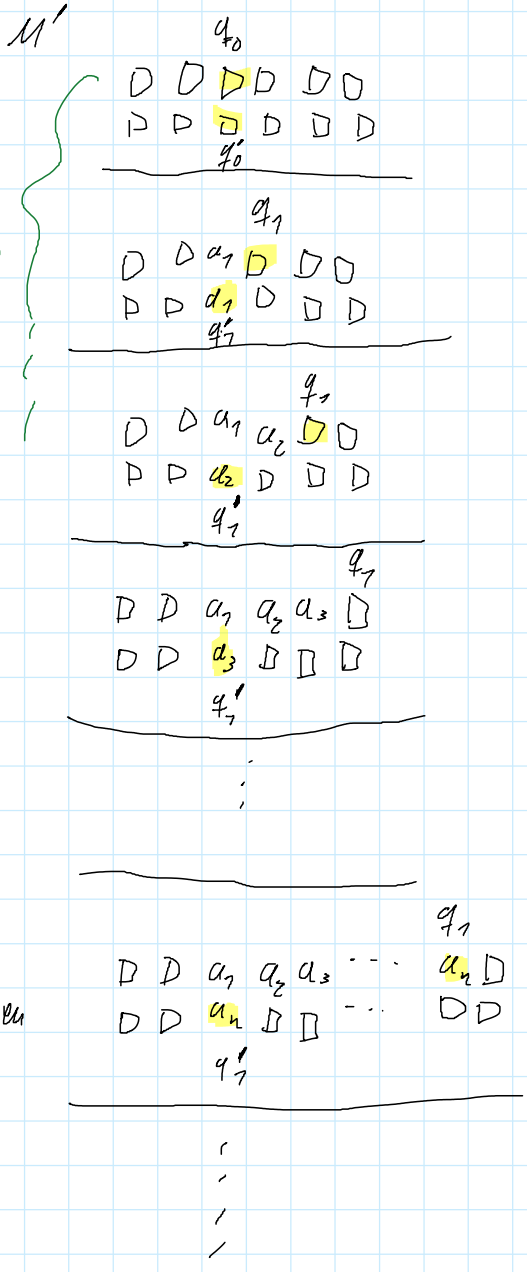
Blatt 2 w

Thursday, April 30, 2020 9:24 AM

$\Sigma = \{a_1, \dots, a_n\}$ und $\Gamma = \Sigma^2$

A2.1: Konstruiere 2-Band TM M' mit folgenden Zuständen:

(siehe das Beispiel rechts zum veranschaulichen!) →



① $(q_0, q_0') \in Q$: startet damit das unten a_1 geschrieben wird, der obere Kopf wandert nach Rechts, der untere bleibt.

② (q_1, q_1'): hier wieder oben wandert, unten bleibt.

Jetzt wird die von q_1' (unten) gelesene Zelle überschrieben mit der vorschrifft a_i lesen a_{i+1} schreiben. q_1 schreibt diesen Wert auch in die aktuelle Zelle.

Das geht so weiter, bis a_n gelesen wird. Dann schreibt q_1' □ und es wird zu (q_2, q_2') gewechselt.

③ (q_2, q_2'): Dieser Zustand lässt den oberen Kopf wieder zurück wandern bis q_2 ein □ liest und einen nach Rechts geht.

④ $\bar{q}$: Endzustand. Ausgabe ist $a_1 a_2 \dots a_n$

(4) q: Endzustand. Ausgabe ist $a_1 a_2 \dots a_n$

(5) q^*: Man kann sich auf die selbe Art des "speicherns" überlegen, wie man zwischen jedes Zeichen ein # setzen kann. Dafür wäre ein weiteren Zustand und eine leichte Modifikation von (q_2, q_2') nötig. Damit hätte man alle 5 Zustände verbraucht.

Außerdem müsste man noch die 2-Band TM M' als 1-Band TM M umschreiben. Das ist nach Theorem 7.5 ohne Probleme möglich.

A2.2:

Beweis: Sei M eine 1-Band TM nach Def 1.7 und M' eine rastlose TM. Der einzige Unterschied liegt in der Def. der Zustandsüberführungsfkt. δ:

$\delta_M: Q \setminus \{\bar{q}\} \times \Gamma \to Q \times \Gamma \times \{L, N, R\}$

$\delta_{M'}: Q \setminus \{\bar{q}\} \times \Gamma \to Q \times \Gamma \times \{L, R\}$

Wir wollen also zeigen, dass das keine echte Einschränkung ist.

Seien $q_1, q_2 \in Q$ (können gleich sein) und $x, y \in \Gamma$ (—"—), dann kann man sich klar machen, dass folgendes gilt:

Damit ist "geklgt von" gemeint

$$\delta_M(q_1, x) = (q_2, y, N) \text{ "}\Leftrightarrow\text{" } \delta_{M'}(q_1, x) = ((q', q_1), y, L)$$

$$\delta_{M'}((q', q_1), z) = (q_2, z, R)$$

Damit ist "macht das gleiche" gemeint.

mit $(q', q_1) \in Q$ und $z \in \Gamma$

Also muss man jedes mal "hin und her wandern". Es sei zu beachten, dass dann doppelt so viele Zustände nötig sind um sich beim "hin und her wandern"

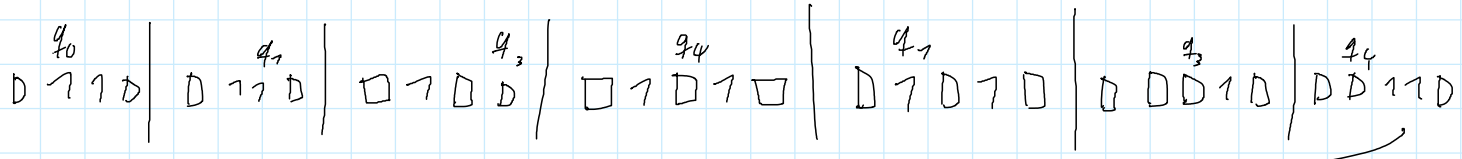
zu merken, in welchem Zustand man ursprünglich war. Also $\boxed{Q_{M'} = Q_M \times \{q'\}}$.

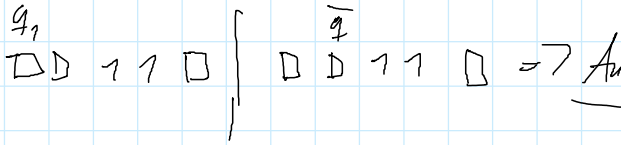
↳ Zur Rechenzeit lässt sich sagen, dass im Worst-Case (wenn jeder Schritt 'R' wäre) doppelt so viele Schritte gemacht werden müssen:

$$\underline{O(t_{M'}(n))} = O(2 \cdot t_M(n)) = \underline{O(t_M(n))}$$

AZ 3: für $w = \varepsilon$ ist Ausgabe $= \varepsilon$. (nach 2 Schritten)

Testlauf für Eingabe = 11





⇒ Ausgabe $= \varepsilon$.

Beschreibung Zustände:

q_0: überspringt Eingabe, wächselt zu q_1

q_1: terminiert für □ merkt sich 0,1 mit q_2, q_3, geht nach Rechts, schreibt □

q_2, q_3: schreiben sofort den gemerkten Wert 0,1, geht nach Links, weiter mit q_4

q_4: überspringt □ nach links, wächselt zu q_1

↳ Insgesamt 6 Einträge von δ können nicht erreicht werden.

- $\delta(q_2, 0)$ $\delta(q_2, 1)$
  $\delta(q_3, 0)$ $\delta(q_3, 1)$

  werden nie erreicht, weil man durch q_0 ganz links auf einem □ ist, einmal nach links geht, sich das Zeichen merkt (mit q_2, q_3) und dann wieder

nach rechts geht. Da kann nun wieder das □ stehen.

- Es verhält sich ähnlich mit $\delta(q_4, 0)$ und $\delta(q_4, 7)$.
  Nach q_0 geht man nach links. Danach mit q_1 nach rechts und hinterlässt ein □. Dort führt man sofort q_2 oder q_3 aus und geht wieder nach links. Also kann hier wieder nur das □ sein.

Note: Die Ausgabe ist immer ε, weil man immer auf dem □ vor der Eingabe endet. Mit der Änderung $\delta(q_4, \square) = (q_1, \square, N)$ hätte man $f_M(w) = w$ realisiert.

AZ. 4: (Ka wie die Syntax sein soll. Habs jetzt irgendwie gemacht. Ein Beispiel wäre hilfreich gewesen.)

i) Addition ist eine in LOOP-Programmen definierte Operation.

Input: x, y
Code: ;
Output: $x+y$

für $x, y \in \mathbb{R}$ ← vermutlich egal, weil wir das eh auf nem Rechner machen würden, nicht?

ii) Hier muss $x, y \in \mathbb{N}$ gegeben sein.

Input: x, y

Code: LOOP y DO $x := x + x$ END

Output: x

iii) Analog zu ii) bezeichne das LOOP Program, welches $x \cdot y$ berechnet als $P \equiv P(x, y)$

iii) Input: x, y

Code: LOOP y DO P(x, y) END

Output: x

iv) Hier nutzen wir aus das "LOOP x DO ... END" nichts macht, wenn x=0

Inp: x

Code: C := 0

LOOP x DO C := 1 END

Out: C

A2.5:

Die Idee hier ist ähnlich wie in A1.4 mit $L = \{0^n 1^n \mid n \geq 1\}$. Der einzige Unterschied ist, dass ein zusätzlicher Zustand benötigt wird der die erste 1 findet, sie mit # maskiert, nach rechts geht und in einen Zustand wechselt der eine 1 erwartet, sie ebenfalls mit # maskiert und zurück nach links geht. Dabei wird wie in A1.4 in einen Zustand gewechselt, der den nach links bis zum ersten B wandert.