ALCO serie 5

Samy Boussaa, Micha de Groot (10434410), Caroline Azeau (10334858)

October 2016

1. Stel de machine wil 4 en 5 optellen. In binaire is dit 00100 en 00101. Bij elkaar moet dit 9 worden, in binair 01001. De band begint als volgt: $\#00100\#00101B$

   - Turingmachine:

$$
\begin{aligned}
(q_0, \#) &=< q_1, \#, R > \\
(q_1, 0) &=< q_1, 0, R > \\
(q_1, 1) &=< q_2, 1, R > \\
(q_1, \#) &=< q_F, \#, R > \\
(q_2, 0) &=< q_2, 0, R > \\
(q_2, 1) &=< q_2, 1, R > \\
(q_2, \#) &=< q_3, \#, L > \\
(q_3, 0) &=< q_3, 1, L > \\
(q_3, 1) &=< q_4, 0, R > \\
(q_4, 0) &=< q_4, 0, R > \\
(q_4, 1) &=< q_4, 1, R > \\
(q_4, \#) &=< q_4, \#, R > \\
(q_4, B) &=< q_5, B, L > \\
(q_5, 0) &=< q_6, 1, L > \\
(q_5, 1) &=< q_5, 0, L > \\
(q_6, 0) &=< q_6, 0, L > \\
(q_6, 1) &=< q_6, 1, L > \\
(q_6, \#) &=< q_7, \#, L > \\
(q_7, 0) &=< q_7, 0, L > \\
(q_7, 1) &=< q_7, 1, L > \\
(q_7, \#) &=< q_1, \#, R >
\end{aligned}
$$

   - RAM: In de RAM zien de eerste 11 registers er als volgt uit: 100#101. Dat is de input. Register 11 en verder gebruiken we voor administratie. De waardes daarvan zijn: 0. In totaal staan er de volgende

waardes in de registers. We gaan er ook van uit dat # een waarde groter dan 1 heeft en dat x en y even veel bits lang zijn. We gaan er ook van uit dat er constanten gebruikt mogen worden die niet in registers staan. Zo niet kan elke keer dat er een constante gerbuikt wordt dat vervangen worden door het laden en gebruiken van één van de oneindig veel registers die die constante bevat. Moch die constante nog niet bestaan kan die gemaakt worden door steeds 1 bij een al bestaande constante op te tellen.

1: LOAD 7# (Laad het eerste bit)
2: SUB ($const_1$) (we halen 1 van de waarde af)
3: JGTZ 7 (we hebben # bereikt)
4: LOAD 7 (haal register 7 op)
5: ADD ($const_1$) (voeg er 1 aan toe)
6: STORE 7 (sla het op in r7)
7: JUMP 0 (jump weer naar het begin)
8: LOAD 7 (laad het adres van #)
9: SUB ($const_1$) (Maak er het adres van x[0] van)
10: STORE 8 (sla het op in r8)
11: LOAD #8 (haal x[0] op)
12: STORE 9 (sla het op in r9)
13: LOAD 8 (Haal het adres van x[0] op)
14: ADD 7 (maak er y[0] van)
15: STORE 8 (Sla het op in r8)
16: LOAD #8 (haal y[0] op)
17: ADD 9 (tel er x[0] bij op)
18: STORE 9 (sla x[0] + y[0] op in r9)
19: LOAD 7 (laad het adres van #)
20: SUB ($const_2$) (Maak er het adres van x[1] van)
21: STORE 8 (sla het op in r8)
22: LOAD #8 (haal x[1] op)
23: STORE 10 (sla het op in r10)
24: LOAD 8 (Haal het adres van x[1] op)
25: ADD 7 (maak er y[1] van)
26: STORE 8 (Sla het op in r8)
27: LOAD #8 (haal y[1] op)
28: ADD 10 (tel er x[1] bij op)
29: STORE 10 (sla x[1] + y[1] op in r10)
30: LOAD 7 (laad het adres van #)
31: SUB ($const_3$) (Maak er het adres van x[1] van)
32: STORE 8 (sla het op in r8)

33: LOAD #8 (haal x[2] op)
34: STORE 11 (sla het op in r11)
35: LOAD 8 (Haal het adres van x[2] op)
36: ADD 7 (maak er y[2] van)
37: STORE 8 (Sla het op in r8)
38: LOAD #8 (haal y[2] op)
39: ADD 11 (tel er x[2] bij op)
40: STORE 11 (sla x[2] + y[2] op in r11)
41: LOAD 10 (laad x[1]+y[1])
42: ADD 10 (verdubbel x[1] en y[1])
43: STORE 10 (sla het weer op)
44: LOAD 11 (laad x[2]+y[2])
45: ADD 11 (verdubbel het)
46: STORE 11 (sla het op)
47: ADD 11 (verdubbel het nog een keer)
48: ADD 10 (tel xy[1] er bij op)
49: ADD 9 (tel xy[0] er bij op)
50: STORE 0 (sla het op in 0)

2. Stel M_E bestaat. Dus er bestaat een machine die beslist of een machine M stopt of niet bij een gegeven invoer. Het is gegeven dat M stopt bij een even getal als invoer, en niet stopt bij een andere invoer. Dus als M_E de machine M en een even getal als invoer krijgt, dan geeft zij 'STOPT' als uitvoer. Stel dat M een machine is die stopt als hij een even getal als invoer krijgt, dan zou M_E een machine zijn die kan bepalen of zijn input een machine is die stopt. Dat zou betekenen dat M_E een M_H is voor bepaalde invoer. We weten dat M_H niet bestaat. Hieruit kunnen we afleiden dat M_E dus ook niet betstaat.

3. (a) Voor een machine met 9 toestanden die niet de accepterende toestand zijn (dus in totaal 10) en 35 bandcellen zijn er $9 * 2^{35}$ stappen nodig voordat je zeker weet dat de accepterende toestand niet bereikt zal worden.

(b) Zie tabel:

	e	f	g
A	1	1	7
B	9	8	7
C	3	2	4

Dit is een n bij n (stel n = 3) tabel waarbij A, B en C een turingmachines zijn met n^2 bandcellen de invoer e, f en g hebben (dit zijn ook turingmachines). Na diagonalisatie krijg je altijd een antwoord met de lengte die groter is dan $\log(n)$. Dus kan de taal niet worden geaccepteerd.

4. Alles kan gecheckt worden in polynomiaal tijd voor alle talen L die in NP zitten en omdat er maar één geheugencel per tijdseenheid kan wordt aangesproken is het ook in polynomiaal geheugen

   (a) Als een inputsignaal door m poorten moet voordat het bij de uitgang is en er maximaal n ingangen zijn hoeven er minder dan $(m * n^2)$ poorten doorgerekend te worden om zeker te weten dat de output voldoet.

   (b) Een aantal weken geleden hebben we gezien dat dit probleem opgelost kan worden door het Ford Fulkerson algoritme. Aangezien dit algoritme in polynomiale tijd kan worden uitgevoerd, zit dit probleem sowieso in NP. De tijd die het kost om een oplossing te controleren is $knopen^3$

   (c) Vertaal dit probleem allereerst naar een graaf. De personen zijn vertexen, en alle personen die in elkaars bereik liggen worden met elkaar verbonden. Nu, als twee mensen aan het bellen zijn kunnen de mensen die verbonden zijn met deze mensen niet bellen. Het antwoord is een graaf met alle mensen die aan het bellen zijn met elkaar verbonden.
   Het is duidelijk dat het gegeven antwoord in polynomiale tijd gecheckt kan worden omdat er hoogstens n^2 kanten zijn. Het kost minder dan n^5 stappen om te controleren of het aantal verbonden punten groter is dan K. Ook kan in polynomiale tijd worden gecontroleerd of aan ieder punt maar één kant zit.