Tentamen
ORIENTATIE INFORMATICA
5 december 2002
09.00 – 12.00 uur; tentamenhal

Opmerkingen vooraf:

- Geef bij elke Haskell-functie ook de typering.
- Geef bij elk onderdeel een duidelijke toelichting.
- In elk onderdeel mag je gebruik maken van vorige onderdelen, ook als je niet in staat bent geweest deze te maken.
- Dit tentamen bestaat uit de gedeelten A en B. Deel B dient door iedereen gemaakt te worden die een tentamenbriefje met 4 studiepunten moet hebben.
  Het staat je vrij om deel A te maken. We nemen in de berekening van het eindresultaat mee het hoogste van wat je op de toets hebt gehaald en wat je nu weet te scoren op deel A.
  Studenten Natuurkunde die een tentamenbriefje met 2 studiepunten willen, maken alleen deel A. Geef tevens op je tentamen aan dat je een 2-punts briefje wilt. **Het is niet mogelijk de 2-punts variant aan te vullen tot een 4-punts briefje.**
- Niet elke opgave telt even zwaar. Bij elk onderdeel staat aangegeven hoeveel punten er maximaal gescoord kunnen worden.
  In deel A zijn 50 punten te behalen. Het cijfer voor dit deel is het aantal behaalde punten gedeeld door 5.
  In deel B zijn 60 punten te behalen. Ook hier is het cijfer het aantal behaalde punten gedeeld door 5. Het eindcijfer is een gewogen gemiddelde van de diverse beoordelingen. Dit eindcijfer ligt tussen 1 en 10 en wordt op een half nauwkeurig vermeld.

---

lees verder

Deel A

Opgave 1

Al bijna een jaar is de euro het wettig betaalmiddel in Nederland en vele andere Europese landen. Toch verschijnen er in de pers nog geregeld uitslagen van onderzoeken, waaruit blijkt dat nog niet iedereen met de euro vertrouwd is. En als je een beetje om je heen kijkt, dan merk je het ook bij de kassa. Was het in het guldentijdperk heel gebruikelijk om bij een kassabedrag van *f* 2,05 een rijksdaalder en een stuiver aan de caissière te overhandigen, zodat je slechts twee kwartjes retour kon krijgen, tegenwoordig waagt slechts een enkeling het om kleine muntjes bij te leggen om een eenvoudig retourbedrag te krijgen.

Maar wat nu als je persé gepast wilt betalen? Dat lukt natuurlijk als je maar voldoende kleine muntjes beschikbaar hebt. Wil je bovendien zo weinig mogelijk muntjes gebruiken om het kassa-bedrag vol te maken, dan voldoet zowel bij het gulden-systeem (stuiver, dubbeltje, kwartje, ...) als bij het euro-systeem (1 cent, 2 cent, 5 cent, 10 cent , ...) een greedy algoritme.

[5 pt] ☐ 1. Leg uit wat we in het algemeen verstaan onder een greedy algoritme en beschrijf voor één van de twee genoemde muntsystemen een greedy algoritme om het gepast betalen met zo weinig mogelijk munten te realiseren.
LET OP: we vragen hier geen uitwerking in Haskell, maar een globale beschrijving van het algoritme in natuurlijke taal.

We bekijken nu een (raar) muntsysteem waarbij een greedy algoritme niet voldoet. Als muntwaarden zijn beschikbaar: 1 eenheid, 4 eenheden, 5 eenheden.

[3 pt] ☐ 2. Geef een kassa-bedrag waarbij een greedy algoritme zoals je bij de vorige vraag hebt beschreven bij dit stelsel niet het kleinste aantal munten oplevert.

Onder `(minMunten n)` verstaan we het kleinste aantal munten dat (in het systeem met 1, 4 en 5 eenheden) nodig is om het bedrag van `n` eenheden te betalen.

[3 pt] ☐ 3. Geef voor de gevallen `n = 0, 1, 2, 3, 4` de waarde van `(minMunten n)`.

[4 pt] ☐ 4. Geef een Haskell-implementatie van de functie `minMunten`.

[5 pt] ☐ 5. Beschrijf wat we in het algemeen verstaan onder *dynamisch programmeren* en leg uit waarom wellicht een implementatie van `minMunten` die gebruik maakt van dynamisch programmeren een sneller algoritme oplevert.

[4 pt] ☐ 6. Geef een implementatie van de functie `hMinMunten` met de betekenis

```
hMinMunten n = (minMunten n, minMunten (n+1), minMunten (n+2),
                minMunten (n+3), minMunten (n+4))
```

zonder daadwerkelijk gebruik te maken van `minMunten`.

[2 pt] ☐ 7. Geef een efficiëntere implementatie van de functie `minMunten` die gebruik maakt van `hMinMunten`.

Opgave 2

In deze opgave ontwikkelen we een sorteeralgoritme voor lijsten van paren die bestaan uit een hoofdletter en een geheel getal. De bedoeling is om de lijst zo te sorteren, dat de eerste elementen in alfabetische volgorde staan. We kijken voor de ordening niet naar het tweede element. Voorbeeld: de volgende lijsten zijn correct gesorteerd:

```
[('B', 17), ('T', 33), ('T', 4), ('T',18), ('V', 2), ('V', 7), ('V', 2)]
[('B', 17), ('T', 18), ('T', 33), ('T',4), ('V', 7), ('V', 2), ('V', 2)]
```

Doordat van te voren bekend is uit welke eindige verzameling de eerste elementen komen (de 26 hoofdletters), is het mogelijk om een sorteeralgoritme te maken dat efficiënter is dan de gebruikelijke methoden (insertion sort, merge sort). Daartoe maken we gebruik van een binaire zoekboom, waarin de elementen van de lijst eerst worden opgeslagen. Uit deze boom wordt vervolgens de gesorteerde lijst geconstrueerd.

In deze opgave maken we gebruik van de volgende Haskell-definitie voor bomen:

```
data Tree = Empty | Node Char [Integer] Tree Tree
```

Een knoop heeft dus een linker- en een rechter (sub-)boom en in de knoop zelf staan twee dingen geadministreerd: een karakter en een lijst van gehele getallen.

Je mag er vanuit gaan dat de karakters zo in de boom zijn geadministreerd, dat we te maken hebben met een binaire zoekboom. Bij de te construeren functies moet deze eigenschap gehandhaafd blijven.

[4 pt] ☐ 8. Geef aan wat we in dit geval verstaan onder een binaire zoekboom.

[4 pt] ☐ 9. Geef een Haskell-functie `add :: (Char, Integer) -> Tree -> Tree` die een paar verwerkt in een boom.

[4 pt] ☐ 10. Geef een Haskell-functie `list2Tree :: [(Char, Integer)] -> Tree -> Tree` die alle paren uit een lijst toevoegt aan een boom.

[4 pt] ☐ 11. Geef een Haskell-functie `tupelList` die bij een karakter en een lijst van gehele getallen de lijst oplevert bestaande uit paren met dat karakter als eerste element en de elementen van de lijst als tweede element. Voorbeeld:

```
tupelList 'G' [4,9,5] = [('G', 4), ('G', 9), ('G', 5)]
```

[4 pt] ☐ 12. Geef een Haskell-functie die bij een boom een geordende lijst oplevert van de opgeslagen gegevens.

[4 pt] ☐ 13. Geef een Haskell-functie die bij een lijst (zoals in deze opgave beschreven) een gesorteerde versie van die lijst oplevert.

Deel B

Opgave 3

Op deze dag ontkomen we natuurlijk niet aan een opgave over Sinterklaas.
Jelle is een groot liefhebber van Sinterklaas. Hij viert zelfs op drie plaatsen dit kinderfeest: met zijn huisgenoten, bij zijn familie en op de club. Bij elk van deze geledingen moet hij voor één persoon een aantal pakjes maken.

Als armlastige student heeft hij een strategie ontwikkeld om voor niet te veel geld toch een flinke hoeveelheid presentjes te kopen. Gedurende het hele jaar houdt hij nauwgezet alle folders in de gaten (Aldi, Blokker, Hema, Kruidvat, Scholtens, Zeeman, ..) en als hij aardige dingen ziet die echt goedkoop zijn, dan fietst hij onmiddelijk naar de winkel om in te slaan.

Op deze manier heeft hij in de loop van het afgelopen jaar een groot aantal presentjes verzameld. En nu komt zijn probleem: hoe moet hij deze presentjes over de drie vieringen verdelen, zonder iets over te houden. Bij elk van de drie groepen is een maximaal te besteden bedrag afgesproken, dus de stapel cadeautjes botweg in drieën verdelen is waarschijnlijk geen optie.

We helpen Jelle met een Haskell-programma. De prijzen van de diverse artikelen representeren we met een lijst van natuurlijke getallen; de drie (resterende) budgetten als een tripel natuurlijke getallen. De functie `verdeelbaar` levert op of de artikelen uit de lijst over de drie geledingen zijn te verdelen, waarbij geen van de drie budgetten wordt overschreden.

```
verdeelbaar :: [Integer] -> (Integer, Integer, Integer) -> Bool

verdeelbaar [] (a,b,c) = True
verdeelbaar (x:lst) (a,b,c)
  | (x <= a) && (verdeelbaar lst (a-x, b  , c  ))   =  True
  | (x <= b) && (verdeelbaar lst (a  , b-x, c  ))   =  True
  | (x <= c) && (verdeelbaar lst (a  , b  , c-x))   =  True
  | otherwise                                       =  False
```

We gaan op zoek naar de worst-case tijdcomplexiteit van de functie `verdeelbaar`. Als maat voor de instantie kiezen we het aantal elementen in de lijst en we tellen het aantal evaluaties van de `<=`-operator.

[4 pt] ☐ 14. Beschrijf een worst-case instantie voor `verdeelbaar` en beargumenteer je keuze.

[4 pt] ☐ 15. Laat $C(N)$ het aantal `<=`-evaluaties zijn bij een lijst van N elementen zoals je die in de vorige vraag hebt beschreven.
Geef een recurrente betrekking voor $C(N)$, dat wil zeggen druk $C(N)$ uit in $C(N-1)$.

[4 pt] ☐ 16. Geef een schatting van de worst-case tijdcomplexiteit van de functie `verdeelbaar`.
LET OP: we vragen geen formeel bewijs, geen exacte uitdrukking maar wel een toelichting!

Het is natuurlijk aardig dat Jelle nu eenvoudig te weten kan komen **of** het mogelijk is om de pakjes te verdelen, maar dat vertelt hem nog niet **hoe** hij de pakjes kan verdelen.

[4 pt] ☐ 17. Geef een Haskell-functie

```
verdeel :: [Integer] -> (Integer, Integer, Integer) ->
                                  (Bool, [Integer], [Integer], [Integer])
```

Het eerste element in het resultaat is een boolean die aangeeft of een verdeling mogelijk is. Is deze boolean `False`, dan zijn de andere drie elementen irrelevant.
Is de boolean `True`, dan geven de drie lijsten aan hoe de pakjes verdeeld kunnen worden.
NB: het is niet toegestaan in je oplossing aanroepen van `verdeelbaar` op te nemen!

De Haskell-functie `verdeelbaar` is een algoritme bij het beslissingsprobleem

(3BOX)
Parameter: een eindige lijst L van natuurlijke getallen en drie natuurlijke getallen V_1, V_2 en V_3
Gevraagd: is de lijst L te verdelen in drie deellijsten L_1, L_2 en L_3 met de voorwaarde dat (voor $i = 1, 2, 3$) de som van de elementen van L_i hoogstens V_i is?

[3 pt] ☐ 18. Leg uit wat we verstaan onder een beslissingsprobleem.
[4 pt] ☐ 19. Beargumenteer dat (**3BOX**) $\in$ **NP**.

Bekijk ook het probleem

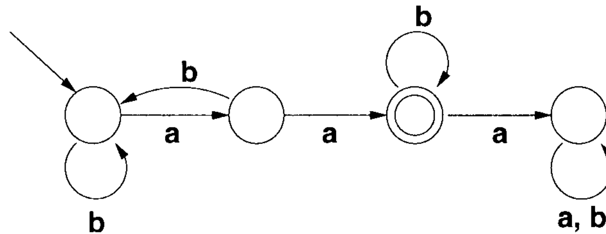
Partitieprobleem (PART)
Parameter: Een eindige verzameling A van natuurlijke getallen
Gevraagd: Is er een deelverzameling X van A zo, dat de som van de getallen in X gelijk is aan de som van de overige getallen in A?

[3 pt] ☐ 20. Geef een ja-instantie van (**PART**) en bewijs dat dit een ja-instantie is.
[3 pt] ☐ 21. Geef een nee-instantie van (**PART**) en bewijs dat dit een nee-instantie is.
[4 pt] ☐ 22. Leg uit wat we verstaan onder de klasse **NPC**.

We kunnen bewijzen dat (**3BOX**) $\in$ **NPC**.
In dat bewijs maken we gebruik van het feit dat (**PART**) $\in$ **NPC**.

[4 pt] ☐ 23. Geef aan in welke richting de reductie moet gaan en wat daarbij je bewijsverplichtingen zijn.
[4 pt] ☐ 24. Toon aan dat (**3BOX**) $\in$ **NPC**.

Opgave 4



Hierboven staat een eindige automaat M getekend.
Het invoeralfabet van deze automaat is $\{\mathtt{a},\ \mathtt{b}\}$.

[4 pt] ☐ 25. Beschrijf nauwkeurig welke strings door de automaat M worden geaccepteerd.

De taal bestaande uit alle strings die door M worden geaccepteerd noemen we L_M. We definieren nu twee talen:

$$L_1 = \{vw \mid v \in L_M \ \wedge \ w \in L_M\}$$
$$L_2 = \{xx \mid x \in L_M\}$$

[4 pt] ☐ 26. Neem onderstaand schema over en geef voor elk van de vier gevallen, voor zover mogelijk, een voorbeeld.

	wel in L_1	niet in L_1
wel in L_2		
niet in L_2		

[4 pt] ☐ 27. Teken een eindige automaat die precies de strings van de taal L_1 accepteert.

[4 pt] ☐ 28. Is er een eindige automaat die precies de strings uit de taal L_2 accepteert? Bewijs de correctheid van je antwoord.

[3 pt] ☐ 29. Is er een Turingmachine die precies de strings uit de taal L_2 accepteert? Beargumenteer je antwoord.

einde

totaal: 110 punten.

Woonplaats:

Deel B

~~14) De worst case is wanneer a=b=c, want dan moeten alledrie de aanroepen volledig doorgerekend worden. Als één van~~

wat tellen: <=

14) De worst case is als a) er geen verdeling mogelijk is, zodat alledrie de recursieve aanroepen volledig moeten worden doorgerekend en b) a=b=c zodat het zo lang mogelijk duurt tot x groter is dan a, b of c en dus de tweede expressie in de &&-operator telkens moet worden uitgerekend. Haskell evalueert boolean expressies namelijk wel lazy, in tegenstelling tot wat in het hoorcollege is verteld, want False && (1/0) levert False en geen foutmelding.

3

15) $C(N) = 3 \cdot C(N-1) + 3$

Immers, voor het doorrekenen van een lijst ter lengte N moet 3 keer een lijst ter lengte (N−1) worden doorgerekend.

$C(0) = ?$

~~16) C(N) = ... (8N)~~
~~want voor elk extra element moet~~

16) $C(1) = 3 = 3^1$

$C(N) = 3^N$ consequent maar fout

Zo wordt bereikt wat bij 15 staat: als N met 1 toeneemt wordt C(N) 3 maal zo groot. $C(1) = 3$ omdat dan in de recursieve aanroepen geen <=-expressies meer geëvalueerd hoeven worden (want $C(0) = 0$).

2

17) verdeel :: [Integer] -> (Integer, Integer, Integer) -> (Bool, [Integer], [Integer], [Integer])

~~verdaal~~ 2.0.2

4

```
17 verdeel :: [Integer] -> (Integer, Integer, Integer) ->
              (Bool, [Integer], [Integer], [Integer])
verdeel [] (a, b, c) = (True, [], [], [])
verdeel (x:lst) (a, b, c)
  | (x <= a) && (kanina = (True, x:a1, a2, a3)
  | (x <= b) && (kaninb = (True, b1, x:b2, b3)
  | (x <= c) && (kaninc = (True, c1, c2, x:c3)
  | otherwise = (False, [], [], [])
  where
    (kanina, a1, a2, a3) = verdeel lst (a-x, b, c)
    (kaninb, b1, b2, b3) = verdeel lst (a, b-x, c)
    (kaninc, c1, c2, c3) = verdeel lst (a, b, c-x)
```

3

18 Een beslissingsprobleem is een probleem dat bepaalt of een bepaalde eigenschap geldt voor de invoer, d.w.z. een probleem met slechts een boolean (ja / nee) als uitvoer.

19 P ∈ NP als een certificaat voor P in polynomiale tijd te verifiëren is. Hiervoor moeten in het geval van 3BOX twee dingen gebeuren:

1. Controleren of L_1, L_2 en L_3 daadwerkelijk deellijsten van L zijn en ze elkaar niet 'overlappen' (dit is iets anders dan disjunct zijn, omdat L dubbele elementen mag bevatten)
2. Controleren of de sommen van L_1, L_2 en L_3 V_1, V_2 resp. V_3 niet overschrijden.

3

1. Dit kunnen we doen door voor elk element van L_i dit element uit L weg te strepen. Als dit niet meer kan of als we aan het eind nog onweggestreepte elementen in L over hebben is het certificaat ongeldig. De worst case complexiteit van deze stap is $N^2 + N$ (ruwweg) want voor elk element van L_i moet in het ergste geval de hele L worden doorlopen. Doordat echter L steeds korter wordt zal de complexiteit in werkelijkheid lager zijn maar deze blijft po-

3 polynomiale tijd

nomiaal. De $+N$ is de complexiteit van de controle op onweggestreepte elementen.
2) Voor elk element L_i is één optelling nodig, dus dit is een polynomiale (lineaire) complexiteit

19 (vervolg) Het verifiëren van een certificaat voor 3BOX kan dus in polynomiale tijd, en daarom is 3BOX $\in$ NP

+1

20 Ja-instantie: $A = \emptyset$

Triviaal: $X = \emptyset$ zodat $\Sigma(X) = 0$ en $\Sigma(A \setminus X) = 0$

Flauw? [OK, nog een: [Klaar!

$A = \{1, 2, 3\}$

Bewijs: $X = \{1, 2\}$ dus $A \setminus X = \{3\}$ en $\Sigma\{1,2\} = \Sigma\{3\} = 3$

3

21 Nee-instantie: $A = \{1, 2\}$

Bewijs: X kan zijn $\{\}$, $\{1\}$, $\{2\}$ of $\{1, 2\}$.
Dan is $A \setminus X$ resp. $\{1,2\}$, $\{2\}$, $\{1\}$ of $\{\}$. Geen van deze mogelijkheden heeft de eigenschap dat $\Sigma(X) = \Sigma(A \setminus X)$ dus dit is een nee-instantie.

3

22 $P \in NPC$ wil zeggen dat het probleem $P \in NP$ de eigenschap heeft dat ieder willekeurig probleem $Q \in NP$ er in polynomiale tijd naar gereduceerd kan worden.

4

23 We moeten hiervoor PART naar 3BOX reduceren in polynomiale tijd. Immers, PART $\in$ NPC betekent dat ieder probleem $Q \in NP$ polynomiaal naar PART te reduceren is. Zo kan Q dus via PART naar 3BOX gereduceerd worden in polynomiale tijd. Bewijsverplichtingen:

1. 3BOX $\in$ NP
2. PART $\rightsquigarrow$ 3BOX : a) de reductie kan in polynomiale tijd, b) ja-instanties van PART leveren ja-inst. voor 3BOX en c) nee-inst. van PART leveren nee-inst. voor 3BOX

4

24
1. Zie vraag 19.
2. Construeer de invoer van 3BOX als volgt uit de invoer van PART:

$L = A$

$V_1 = \frac{1}{2}\Sigma(A)$ verdubbel elementen ivm delen

$V_2 = \frac{1}{2}\Sigma(A)$ door 2

$V_3 = 0$

a) Ten eerste kan dit in polynomiale tijd, want

3

sommeren kan lineair. b) Ten tweede: als we een ja-instantie van PART hebben, kan A in twee gelijkwaardige delen worden gesplitst. De waarde van deze delen is ieder de helft van de totale waarde van A. 3BOX kan de lijst dus ook verdelen in twee delen met waarde ten hoogste $\frac{1}{2}\Sigma(A)$ door precies dezelfde verdeling te gebruiken; de derde 'doos' heeft een inhoud van 0 en wordt dus niet gebruikt. c) Ten derde: hebben we een nee-instantie van PART, dan kan deze niet in twee gelijkwaardige delen worden gesplitst. Misschien kan 3BOX hem wel over twee* 'dozen' verdelen, maar deze zullen niet een gelijkwaardige inhoud hebben. Eén van deze dozen moet dus een inhoud groter dan $\frac{1}{2}\Sigma(A)$ hebben, en dat was in strijd met de voorwaarden. 3BOX zegt dus 'nee' op een nee-instantie van PART.

Aan alle bewijsverplichtingen is voldaan; 3BOX ∈ NPC

Q.E.D.

25 Precies iedere string die precies twee a's achter elkaar bevat en daarna niets of uitsluitend b's wordt geaccepteerd.

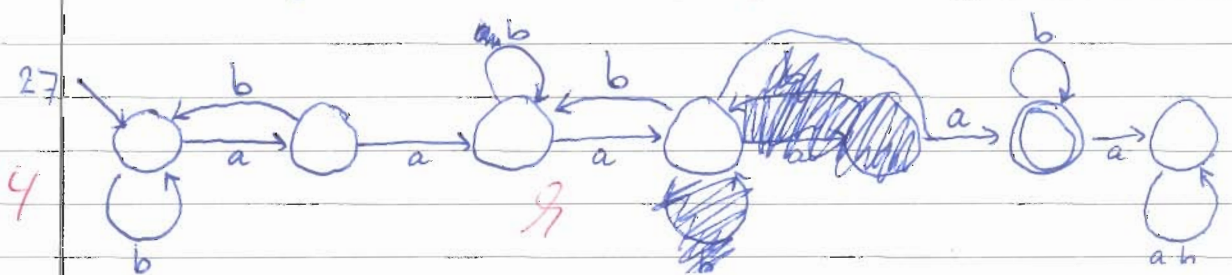
3

26

	wel in L_1	niet in L_1
wel in L_2	aaaa	onmogelijk, want L_2 is een speciaal geval van L_1
niet in L_2	aabaa	a

4

27



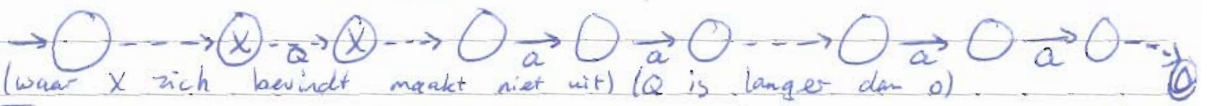
4

28.) Nee, er is geen eindige automaat die precies de strings uit L_2 accepteert. Bewijs: stel dat zo'n FSA wel zou bestaan. Dan zou deze n toestanden hebben. De executie van een string ziet er dan zo uit

Woonplaats:

4

Neem nu een string uit L_2 met een lengte groter dan n. Dan worden één of meer toestanden vaker dan een keer doorlopen:



(waar X zich bevindt maakt niet uit) (Q is langer dan 0)

Tussen X en X wordt de string Q verwerkt. Nu plaatsen we in de invoerstring nog eens Q na Q. Deze wordt nu ook geaccepteerd,* terwijl hij niet meer uit twee gelijke delen bestaat, en dus niet in L_2 zit. Door deze tegenspraak kan er geen FSA bestaan voor de taal L_2.

2g Volgens de these van Church / Turing is een Turingmachine in staat elk denkbaar algoritme uit te voeren. Het geven van een algoritme is hier dus voldoende:

1. Splits de string in twee even lange delen. Is dit niet mogelijk, lever dan NEE op.
2. Controleer of een van deze delen tot L_M behoort m.b.v. de getekende FSA. Zo nee, lever dan NEE op.
3. Controleer of beide delen van de invoerstring gelijk zijn. Zo ja, lever JA op; zo nee, lever NEE op.

3

Bewijs van correctheid:

1. Een string die uit twee gelijke delen bestaat kan nooit een oneven lengte hebben. Als dat wel zo is, is de string niet uit L_2.
2. Als $X \notin L_M$ is de string ook niet uit L_2.
3. Als beide delen wel in L_M zitten maar niet gelijk zijn zit de string ook niet in L_2. Anders is aan alle voorwaarden voldaan en zit de string dus wel in L_2. Hij bestaat dan dus uit twee gelijke, (even lange) delen die beide tot L_M behoren, en dat was de definitie van L_2.

* Doordat de FSA nog eens Q leest als hij na het verwerken van Q in X terug is, zal hij hetzelfde traject volgen en weer in X uitkomen. Daarna verloopt de executie verder gelijk.