Opgave 3 [11]

Splits lst in kop & staart.
kijk of x gelijk is aan kop.
zo nee, tel de freq van x in de staart
zo ja, doe 't zelfde, maar tel er 1 bij op.
Een lege lijst heeft altijd freq 0 van x

```
freq :: Integer -> [Integer] -> Integer

freq x [] = 0
freq x (a:lst)
  | x == a    = 1 + rest
  | otherwise = rest
    where rest = freq x lst
```

[12] Het idee is dat we steeds het i^e element van de i^e niet-lege lijst in een nieuwe lijst plaatsen. Als de grote lijst alleen (alleen lege lijsten bevat) moet er een lege lijst teruggegeven worden.

```
maakLijst :: [[Integer]] -> [Integer]

maakLijst [[]] = []
maakLijst [([]:lstn)] = maakLijst [lstn]
maakLijst [((a:lst):lstn)]
    = a : (maakLijst [(lst : lstn)])
```

[13] We tellen voor ieder element van lst 1 de freq ervan in lst1 en lst2 en die moeten gelijk zijn van lst1 moet gelijk zijn van die van lst2

```
uniek :: [Integer] -> [Integer] -> Bool
uniek [] lst2 = True
uniek (a:lst1) [] = False
uniek (a:lst1) lst2
```

Om deze freq te kunnen tellen, dienen de oorspronkelijke lijsten 1 en 2 compleet te blijven
We

hoofd (en correct): { maakLijst [] = []
maakLijst (a:bt) = a ++ maakLijst bt

~~unieke~~

[13] NB: elk elem. v lst1 moet 1 keer voorkomen in lst2. lst 2 mag dus wel 'unieke' elem. hebben.
[1] en [1, 2] => True!

Duplicaten mogen niet voorkomen (gegeven), dus voor ieder element in lst1 moet de freq ervan in lst2 gelijk aan 1 zijn

geef type: `uniek :: [Integer] -> [Integer] -> Bool`

```
uniek [] lst2    = True
uniek lst1 []    = False        (lst1 is dus niet leeg)
uniek (a:lst1) lst2
   = (freq a lst2 == 1)
     'and' (uniek lst1 lst2)
```

overbodig

[14] Te controleren of elk element van V in precies één elem v C voorkomt.
We koppelen nu de deellijsten van C aan elkaar, en (met 'maakLijst'), en moeten nu dus controleren "of elk element van V precies één keer voorkomt in die gekoppelde verzameling van C".
En dat is precies wat 'uniek' voor ons doet.

```
exOverd :: [Integer] -> [[Integer]] -> Bool
exOverd V C = (uniek V (maakLijst C))
        lst1 lst2      lst1           lst2
```

[15] Er zijn 2 verschillende vgln:
* in freq: "x == a" Deze vgl wordt evenvaak uitgevoerd als het aantal elem.n. dat de meegegeven lijst bevat. (hier: maakLijst lst2)
* in 'uniek': "freq a lst == 1" (+ recursief)
Ook deze vgl wordt evenvaak uitgevoerd als het aantal vd elem dat lst1 bezit
NB. in mijn 'maakLijst' wordt niet (expliciet) een vgl uitgevoerd. 'm' heeft dus geen invloed op de tijdscomplexiteit (Tc)

Uit bovenstaande valt ~~af~~ te halen, dat de Tc gelijk is aan $\sum_{i=0}^{n} i = \frac{n}{2}(n+1) \Rightarrow O(n^2)$

~~O(n^2)~~ merk op: exOverd lst1 lst2 = uniek lst1 (maakLijst lst2)
lengte n   in worst case:

Woonplaats: Delft | Jaar van eerste inschrijving: | Naam docent:

16 $V = [12, 5, 4, 7, 9, 1, 3]$

We kiezen: $X = [[2,7,1], [1,4], [3,9]]$

Inderdaad: elk element van V komt precies één element van C voor

En C is is een deelverz v M

$\Rightarrow$ deelverz C van M is exacte overdekking v V

$\Rightarrow$ ja-instantie

17 Te bewijzen: $(XC) \in NP$

$\Rightarrow$ te bewijzen dat een certificaat van XC "in polynomiale tijd" (='pol.') te verifiëren valt.

V, M en C zijn dus gegeven.

Te controleren:

(*) C is deelverz. v. M

(**) C is exacte overd. v. V

Input: (V, M)

certificaat: C

In vraag 15 reeds gezien dat (**) in 'pol.' uit te voeren valt.

Voor (*) valstaat het de elementen van C met de elem. van M te vgl. Dit kan uiteraard ook 'pol.'

De complexiteit v verifiëren v/h certificaat is dus de som v deze 2 compl.n. $\Rightarrow$ 'pol' $\Rightarrow XC \in NP$

18 Een probleem P heet NP-volledig als $P \in NP$ en ieder probleem $Q \in NP$ 'pol' te reduceren valt tot P ($Q \rightsquigarrow P$) (en $P \in NP$)

19 Dus te bewijzen: $SAT \rightsquigarrow XC$

Deze reductie R zal dus zodanig moeten zijn, dat precies iedere ja-instantie van SAT leidt tot een ja-instantie van XC via de Reductie R.

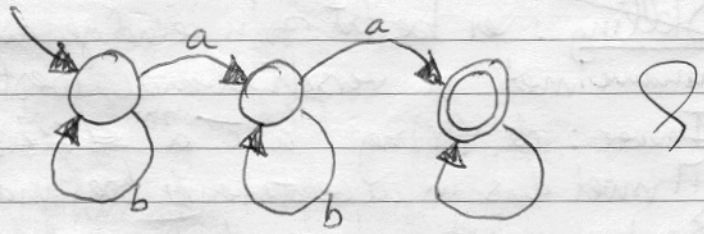
Verder verplichting: te controleren dat R in polynomiale tijd te berekenen is.

Opgave 4

(3) [10] $\langle A \rangle = b^m a b^n$, met $m, n \in \{0, 1, 2, \ldots\}$
Dus: 1 a, met evt. een aantal b's ervoor en erachter ✓

[11] $\langle B \rangle = \langle A \rangle \langle A \rangle = b^l a b^m a b^n$
met $l, m, n \in \{0, 1, 2, \ldots\}$
Dus eindige automaat (EA):

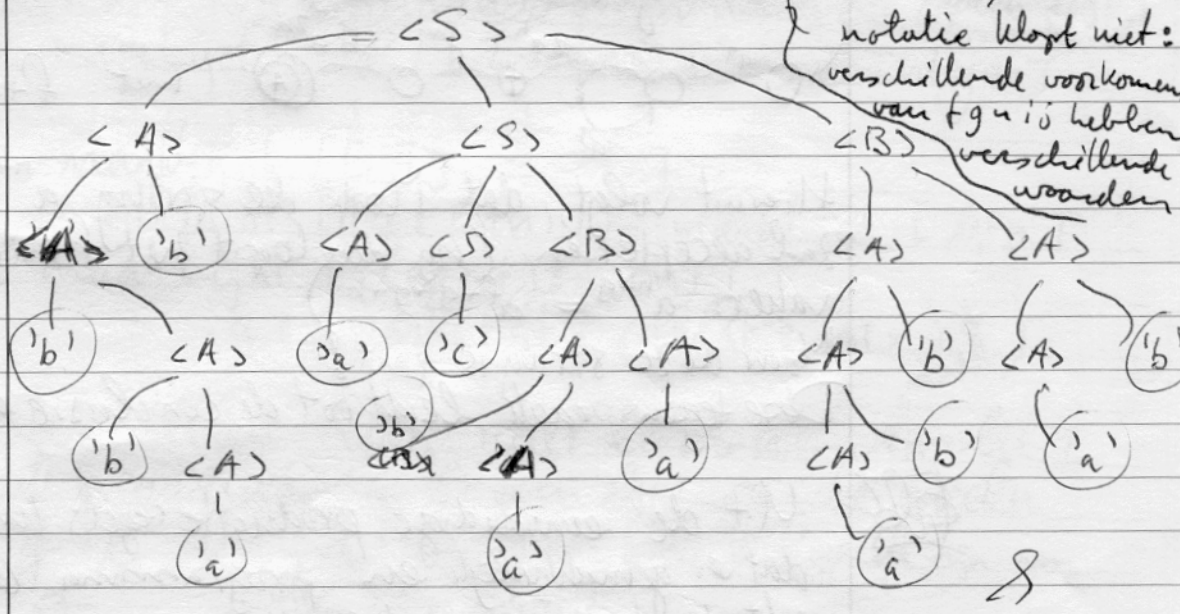
(4)



✓

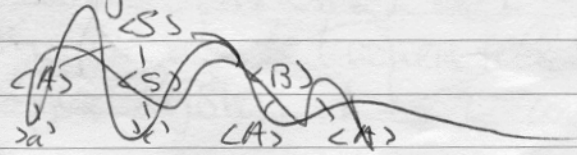
[11] $\langle S \rangle = \langle A \rangle \langle S \rangle \langle B \rangle = \langle A \rangle \langle S \rangle \langle A \rangle \langle A \rangle$
$\Rightarrow \langle S \rangle = (b^f a b^g)^m c (b^h a b^i a b^j)^n$ met $f, g, h, i, j, m, n \in \{0, 1, 2, 3, \ldots\}$
"bbabacbaaaabbab"

idee OK, notatie klopt niet: verschillende voorkomens van f g h i j hebben verschillende waarden

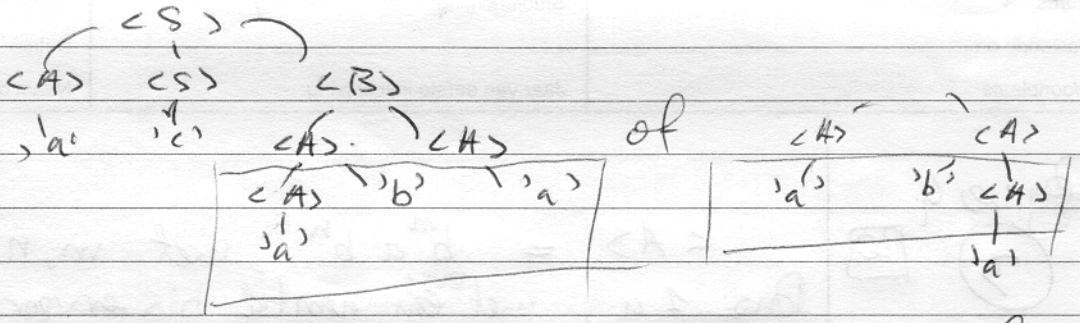


(4) ✓

[23] Ja, er is een string te bedenken die op meerdere manieren uit de productie-regels v/d gram ontstaan kan zijn: bijv de string "acabbā"



(3)

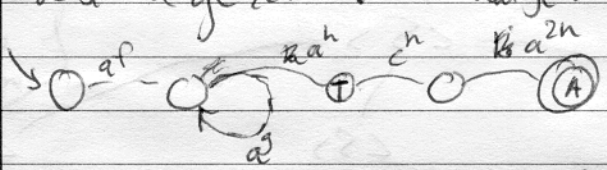


⇒ Ambigue.

(2) [24] Reeds gedaan in [20] – [22] : ⇒ zie [22]

[25] Stelling : er bestaat zo'n eindige automaat M, aanname: met n verschillende toestanden.

(4) Invoer : de string $a^n c^n a^{2n}$ ~~a^n c^n a^n~~ ⇒ correct
M moet dus in accepterende toestand (A) eindigen.
Beschouw de tussentoestand T nadat M het 'n' aantal a's heeft gelezen.
M heeft nu 'n' toestandsovergangen ondergaan, maar mag nooit in A zijn geweest. waarom niet?
Dus M heeft minimaal 2 keer in minstens 1 toestand gezeten. Plaatje:



met $f+g+h=n$

Hieruit volgt, dat M ook de string $a^{n+g} c^n a^{2n}$ zal accepteren (hij doorloopt het 'rondje' een keer vaker. $a^{n+g} = a^{f+g+g+h}$)
Maar deze string is fout!
Deze tegenspraak leidt tot de conclusie dat M niet bestaat

[26] Uit de eenvoudige productie regels kunnen we zien dat er gemakkelijk een programma (algoritme) valt te bedenken die de invoer controleert op de grammatica.

(7) Uit de these v Turing volgt dan dat er ook een Turing-machine voor bestaat.