Voornaam:	Achternaam:

AB: gequoteerde oefenzitting 2
20 dec 2022 08u30 tot 10u30

Richtlijnen Elke vraag wordt gevolgd door een kader waarin je antwoord moet komen: er is meer dan genoeg ruimte, dus wat je buiten dat kader schrijft wordt niet beschouwd als deel van je antwoord en je schrijft er best niks. Kladbladen worden ook ingediend, maar zullen niet bekeken worden tijdens het verbeteren. Structureer je antwoord en schrijf duidelijk: voor klad dienen de kladbladen die je kan krijgen.

Behalve de kladbladen en de opgavenbladen ligt op je werkplek enkel nog je cursustekst waarin geen opgeloste oefeningen (behalve voor de zelf-doens) mogen staan. Zet je gsm uit en steek deze in je boekentas. Leg je studentenkaart klaar met foto en naam naar boven.

Elke vraag volgt het concept van een 'watervalvraag'. Hierbij krijg je per onderdeel een gamma aan vragen gerankschikt op moeilijkheid. Bij iedere vraag staat een gewicht. Voor ieder onderdeel kies je er één vraag uit, deze los je op. Op dit voorblad duid je aan welke vraag je gekozen hebt door de corresponderende letter in het juiste vakje in de tabel hieronder in te vullen. Er wordt maximaal één vraag verbeterd per onderdeel, dus het heeft geen zin om meerdere vragen te kiezen. Voor ieder onderdeel kan je maximaal het aantal punten verdienen dat overeenkomt met het gewicht van je gekozen vraag. In totaal kan je maximaal 20 punten verdienen. Kies je vragen dus wijs!

Veel succes en prettige feestdagen!

1.	
2.	
3.	
4.	

1 Beslisbaar, Herkenbaar of Coherkenbaar? (6p)

Geef voor je gekozen taal of die taal (I) beslisbaar, (II) herkenbaar en niet beslisbaar, of (III) co-herkenbaar en niet beslisbaar is. Bij die drie gevallen geef je ook (I) een beschrijving van een beslisser, (II) een bewijs van niet-beslisbaarheid en een beschrijving van een herkenner, of (III) een bewijs van niet-beslisbaarheid en een beschrijving van een co-herkenner

A (4p) $\{\langle M\rangle | M$ is een Turing Machine die geen enkele string met een oneven aantal 1'en aanvaardt.$\}$

B (5p) $\{\langle M\rangle | M$ is een DFA die geen enkele string met een oneven aantal 1'en aanvaardt.$\}$

C (6p) $\{\langle M, N, s\rangle | M, N$ zijn Turing Machines en bij input s voert M minder stappen uit dan $N\}$

A De taal (in wat volgt aangeduid met L_A) is co-herkenbaar en niet beslisbaar.

- Niet beslisbaar: stelling van Rice.
  Eigenschap van Turing Machines
  Niet-triviaal: sommige TM's behoren wel tot L_A, andere niet.
  Taal-invariant: gegeven meerdere TM's die dezelfde taal bepalen, dan zitten ze ofwel allemaal in L_A, of geen enkele.
  Dus de stelling van Rice is toe te passen en L_A is niet beslisbaar.
- Co-herkenbaar: we bouwen een Herkenner H_A voor het complement $\overline{L_A}$:
  Voor input $\langle M\rangle$ gaat H_A M stapsgewijs uitvoeren op verschillende strings. In stap i overlopen we alle strings s over het alfabet met een oneven aantal 1'en en maximaal lengte i. We voeren dan i stappen van M uit op s.
  - Als M s aanvaard heeft in die i stappen, dan aanvaardt M een string met een oneven aantal 1'en, dus $\langle M\rangle \in \overline{L_A}$, dus H zal $\langle M\rangle$ aanvaarden.
  - Als M s niet aanvaard heeft, dan gaan we naar de volgende string tot alle strings van maximaal lengte i getest zijn. Als geen enkele s aanvaard werd, verhogen we i en herhalen we de procedure.

  Aangezien we een herkenner kunnen bouwen, is $\overline{L_A}$ herkenbaar, dus L_A is co-herkenbaar.

Vaak gemaakte fouten:

- Het acceptance probleem voor TM's is niet beslisbaar. Een naïeve herkenner waarbij je alle strings overloopt en de machine gewoon laat lopen op de strings tot de machine eindigt, is onvoldoende! Dan kan de machine vastlopen in een lus voor een bepaalde string, terwijl een latere string wel aanvaard zou worden. Daarom laten we de machine telkens maar een eindig aantal stappen uitvoeren.
- $L_A \neq \{w | w$ heeft geen oneven aantal 1'en$\}$, dus een DFA die die laatste taal bepaalt geldt niet als beslisser of herkenner voor L_A.

B De taal (in wat volgt aangeduid met L_B) is beslisbaar.
We bouwen een beslisser B_B voor L_B:
Een DFA met q toestanden aanvaardt geen enkele string met een oneven aantal 1'en als er geen enkel pad bestaat naar een eindtoestand dat een oneven aantal 1'en op de labels heeft.

- Stel de DFA heeft geen lussen: alle strings in de taal bepaald door de DFA hebben maximaal lengte q. Al deze strings moeten getest worden.
- Stel de DFA heeft wel lussen: nu moeten we kunnen testen of de lussen een even of oneven aantal 1'en bevatten.
  - Lussen met een even aantal 1'en mogen weggedacht worden, die hebben geen invloed op de evenheid van het aantal 1'en in aanvaarde strings. Hier moeten we dus opnieuw alle strings van maximaal lengte q testen.

- Als er een lus bestaat met een oneven aantal 1'en met een pad naar een eindtoestand, zullen er zeker strings aanvaard worden door de DFA met een oneven aantal 1'en. Stel een string s, die aanvaard wordt door de DFA, wordt gesplitst in delen x, y en z met x de substring die voor de lus komt, y het deel in de lus en z het deel na de lus. Als y een oneven aantal 1'en bevat, zal ofwel xz een oneven aantal 1'en tellen, ofwel zal xyz een oneven aantal 1'en tellen. Dus is het voldoende om alle strings te testen die maximaal één lus doorlopen hebben. Deze worden begrensd door lengte $2q$.

Dit betekent dat het voor eender welke DFA voldoende is om alle strings te overlopen van maximaal lengte $2q$, los van hoeveel en welke lussen de DFA heeft. Dus voor input $\langle M \rangle$ zal onze beslisser B_B systematisch alle strings s met een oneven aantal 1'en over het alfabet aflopen tot en met lengte $2q$. Dan gaan we na of M s aanvaardt, dit komt overeen met het beslisbare acceptance probleem voor DFA's, dus we kunnen hiervoor de correcte beslisser gebruiken.

- Als een s aanvaard wordt, dan zal B_B $\langle M \rangle$ weigeren.
- Als s niet aanvaard wordt, gaat B_B verder met de volgende string. Als er geen strings meer overblijven met lengte kleiner of gelijk aan $2q$, aanvaardt B_B de input $\langle M \rangle$.

Alternatieve oplossing: bouw een beslisser door de doorsnede te nemen met het complement van de DFA voor de reguliere expressie $((a|b|\ldots)(11)^*)^*1((a|b|\ldots)(11)^*)^*$ met $a, b, \ldots \in \Sigma \backslash \{1\}$, dit is opnieuw een DFA want de reguliere talen zijn gesloten onder complement en doorsnede. Dan kan de beslisser voor E_{DFA} gebruikt worden om te beslissen.
Vaak gemaakte fouten:

- $L_B \neq \{w | w$ heeft geen oneven aantal 1'en$\}$, dus een DFA die die laatste taal bepaalt geldt niet als beslisser of herkenner voor L_B.
- Stelling van Rice is enkel toepasselijk op eigenschappen van **Turing Machines**. DFA's zijn ook Turing Machines, maar niet alle Turing Machines zijn DFA's. Soms kan je van een bepaalde eigenschap wel beslissen of een DFA de eigenschap heeft door gebruik te maken van de specifieke eigenschappen van een DFA. Dit wil niet zeggen dat de eigenschap ook beslist kan worden voor eender welke Turing Machine.
- Enkel testen tot lengte q is onvoldoende door het mogelijke voorkomen van lussen.

C De taal (in wat volgt aangeduid met L_C) is herkenbaar en niet-beslisbaar:

- Herkenbaar: We bouwen een herkenner H_C voor L_C:
  Voor input $\langle M, N, s \rangle$ doet H_C: laat M en N parallel lopen op s.
  - Als M stopt en N loopt nog, dan aanvaardt H_C zijn input $\langle M, N, s \rangle$.
  - Als N stopt (voordat M stopt of op hetzelfde moment als M) dan weigert H_C zijn input.

  Als M minder stappen uitvoert dan N, dan moet dit een eindig aantal stappen zijn, dus dan zal de uitvoering van M stoppen. We hebben dus een herkenner gebouwd, dus L_C is herkenbaar.
- Niet beslisbaar: We reduceren het halting-probleem H_{TM} naar L_C:
  Stel L_C is beslisbaar, dan hebben we een beslisser B_C voor L_C, hiermee bouwen we een beslisser B_H voor H_{TM}. We bouwen eerst een Turing Machine N die voor elke string in een lus gaat. Bijvoorbeeld de Turingmachine $(\{q_s, q_a, q_r\}, \Sigma, \Gamma, \delta, q_s, q_a, q_r)$ met δ zodat

  $$\forall q \in \{q_s, q_a, q_r\}, \forall \gamma \in \Gamma : \delta(q, \gamma) = (q_s, \gamma, S)$$

  Nu zal B_H voor input $\langle M, s \rangle$ de beslisser B_C uitvoeren met input $\langle M, N, s \rangle$.
  - Als B_C zijn input aanvaardt, dan voert M minder dan oneindig stappen uit voor s, dus M stopt bij input s. Dan aanvaardt B_H zijn input $\langle M, s \rangle$.

– Als B_C de input weigert, dan voert M minstens evenveel stappen uit als N, dus oneindig veel, dus M stopt niet bij input s. Dan weigert B_H zijn input $\langle M, s \rangle$.

Nu hebben we een beslisser gebouwd voor H_{TM}, wat een onbeslisbare taal is. De assumptie dat er een beslisser bestaat voor L_C is dus verkeerd. Dus L_C is onbeslisbaar.

Vaak gemaakte fouten:

- Er moet bewezen worden dat de taal niet beslisbaar is, een redenering is onvoldoende. Stelling van Rice werkt hier niet, want de eigenschap is niet taal-invariant. Bewijs kan enkel door het probleem te reduceren naar een onbeslisbaar probleem.

2 Reduceren (6p)

A (4p) Toon aan dat $\{\langle M, N\rangle | M, N$ zijn Turing Machines en $L(M) \cap L(N) \subseteq \{1\}^*\}$ turingreduceerbaar is naar A_{TM}.

B (6p) Toon aan adhv een reductie dat $\{\langle M\rangle | M$ is een Turing Machine die de string 001110 aanvaardt.$\}$ niet beslisbaar is.

A $L_A \leqslant_T A_{TM}$.

- $L_A \leqslant_m E_{TM}$:
  We transformeren de input $\langle M, N\rangle$ naar $\langle M'\rangle$ waarbij M' een Turing Machine is die de doorsnede herkent van $L(M)$, $L(N)$ en het complement van de reguliere taal gegeven door de reguliere expressie 1*. Dit is een Turing-berekenbare transformatie. De transformatie is analoog aan hoe de doorsnede gedefinieerd werd voor de DFA's bij reguliere talen.
  - Als $\langle M, N\rangle \in L_A$, dan is de doorsnede van de talen met het complement van de reguliere taal leeg, dus $M' \in E_{TM}$.
  - Als $\langle M, N\rangle \notin L_A$, dan is de doorsnede van de talen met het complement van de reguliere taal niet leeg, dus $M' \notin E_{TM}$.

  Dus we vinden dat $L_A \leqslant_m E_{TM}$
- Als $L_A \leqslant_m E_{TM}$ dan $L_A \leqslant_T E_{TM}$.
- We weten dat $E_{TM} \leqslant_T A_{TM}$ (zie cursus p.130), dus $L_A \leqslant_T E_{TM} \leqslant_T A_{TM}$, dus $L_A \leqslant_T A_{TM}$.

Vaak gemaakte fouten:

- Je mag geen concrete M en N kiezen, dus ook de doorsnede kan eender wat zijn. De doorsnede hoeft niet de reguliere taal gegeven door de reguliere expresie 1* te zijn. Ook weten we van geen enkele string uit die taal dat de string in M en N moet zitten om aan de voorwaarde te voldoen. Testen of een concrete string in je taal zit, volstaat dus niet.
- Als je alle strings die niet in 1* afloopt en ervan test met het orakel of de string geaccepteerd wordt door M en N, zal je een coherkenner construeren, geen beslisser. Er zijn oneindig veel strings om te testen, dus als er geen enkele in de doorsnede zit, zal de machine nieuwe strings blijven genereren. Om aan te tonen dat de taal turingreduceerbaar is naar A_{TM}, moet je een **beslisser** kunnen bouwen aan de hand van een orakel voor A_{TM}.
- Een orakelmachine is een machine die een taal beslist aan de hand van een orakel, dit is dus niet hetzelfde als een orakel.

B $A_{TM} \leqslant_m L_B$:
Stel er bestaat een beslisser B_B voor L_B, we construeren een beslisser B_A voor A_{TM} met input $\langle M, s\rangle$:

- Constueer een machine M' die bij input w M laat lopen op s. Als M aanvaardt, dan M' ook, als M weigert, dan M' ook. Deze machine herkent alle strings over het alfabet als M s accepteert, anders herkent M' de lege taal. Dit is een Turingberekenbare functie.
- Laat B_B lopen op $\langle M'\rangle$.
  - Als B_B aanvaardt dan kan M' niet de lege taal herkennen, dus aanvaardt B_A zijn input.
  - Als B_B weigert, dan weigert B_A zijn input.

  Dus $A_{TM} \leqslant_m L_B$. Aangezien A_{TM} niet beslisbaar is, kan L_B ook niet beslisbaar zijn.

Vaak gemaakte fouten:

- Je mag eender welke reductie kiezen, zolang de reductie aantoont dat de taal niet beslisbaar is. Om aan te tonen dat een taal L_1 niet beslisbaar is via reductie, moet je een gekende onbeslisbare taal L_2 reduceren naar L_1. L_1 reduceren naar L_2 is onvoldoende. Moest dit laatste onbeslisbaarheid betekenen, dan was geen enkele taal beslisbaar. Want als L_1 beslisbaar is, dan is L_1 zeker ook beslisbaar als we een orakel hebben voor eender welke L_2, dus is L_1 turingreduceerbaar naar eender welke taal, dus ook naar de niet beslisbare talen. Dit zou dan een contradictie geven.

3 Primitieve functies (6p)

Toon aan dat de functie een primitief recursieve functie is. Je definieert best hulpfuncties die je kan gebruiken binnen in je primitief recursief voorschrift zodat het voorschrift overzichtelijk blijft. Enerzijds maak je dan minder kans op overschrijffouten, anderzijds vergemakkelijkt dit het werk van de verbeteraar. Bewijs van die hulpfuncties ook (correct) dat ze primitief recursief zijn om een maximaal punt te kunnen behalen.

A (3p) $Som_{tot} : \mathbb{N} \to \mathbb{N}$ waarbij $Som_{tot}(x) = \Sigma_{i=0}^{x} i$

B (4p) $Kl : \mathbb{N}^2 \to \mathbb{N}$ waarbij

$$\begin{cases} Kl(a,b) = 1 & \text{als } a \leqslant b \\ Kl(a,b) = 0 & \text{anders} \end{cases}$$

C (6p) Gegeven een paar natuurlijke getallen (a, b), dan geeft de functie $Rest : \mathbb{N}^2 \to \mathbb{N}$ de restwaarde terug bij de deling a/b.

De kleuren worden enkel gebruikt als hulpmiddel om in te zien van waar een bepaalde term komt. De infix- en prefixnotatie worden door elkaar gebruikt om de overzichtelijkheid te begunstigen.

A $Som_{tot} : \mathbb{N} \to \mathbb{N}$ waarbij $Som_{tot}(x) = \Sigma_{i=0}^{x} i$

- Hulpfunctie $+ : \mathbb{N}^2 \to \mathbb{N} : +(x, y) = x + y$ wordt gegeven door

$$Pr[p_1^1, Cn[succ, p_3^3]] \text{ (zie cursus p.136)}$$

- Som_{tot}

$$\begin{cases} Som_{tot}(0) = 0 \\ Som_{tot}(x+1) = Som_{tot}(x) + (x+1) \end{cases}$$

wordt gegeven door

$$Pr[nul, Cn[+, p_2^2, Cn[succ, p_1^2]]]$$

Vaak gemaakte fouten:

- Een primitieve recursie Pr heeft een functie f nodig wanneer het laatste argument $= 0$ en g gedefinieerd in termen van y als het laatste argument gelijk is aan $y + 1$. We gebruiken dus de regel $Som_{tot}(x + 1) = Som_{tot}(x) + (x + 1)$ en niet de regel $Som_{tot}(x) = Som_{tot}(x - 1) + x$. De predecessor-functie is hier dus onnodig.

B $Kl : \mathbb{N}^2 \to \mathbb{N}$

- Hulpfunctie uit oefenzitting $pred : \mathbb{N} \to \mathbb{N}$:

$$\begin{cases} pred(0) = 0 \\ pred(x+1) = x \end{cases}$$

wordt gegeven door

$$Pr[nul, p_1^2]$$

- Hulpfunctie uit oefenzitting $\dot{-} : \mathbb{N}^2 \to \mathbb{N}$:

$$\begin{cases} \dot{-}(x,y) = x - y & (x \geqslant y) \\ \dot{-}(x,y) = 0 & (x \leqslant y) \end{cases}$$

Dit kan geschreven worden als

$$\begin{cases} \dot{-}(x,0) = 0 \\ \dot{-}(x,y+1) = pred(\dot{-}(x,y)) \end{cases}$$

En dit wordt dan

$$Pr[p_1^1, Cn[pred, p_1^3]]$$

- $Kl : \mathbb{N}^2 \to \mathbb{N}$

$$\begin{cases} Kl(a,b) = 1 & (a \leqslant b) \\ Kl(a,b) = 0 & (\text{anders}) \end{cases}$$

Dit kan geschreven worden als

$$Kl(a,b) = 1 \dot{-} (a \dot{-} b)$$

En dit wordt dan

$$Cn[\dot{-}, Cn[succ, nul], Cn[\dot{-}, p_1^2, p_2^2]]$$

Vaak gemaakte fouten:

- Kijk goed naar je aantal argumenten, waarop pas je een deelfunctie eigenlijk toe. Daarom helpt het om je functie te definiëren aan de hand van hulpfuncties. Zo blijft het overzichtelijk en zie je makkelijker op welke variabelen je functie toegepast wordt.
- Als je een functie f toepast op een andere functie g dan doe je dit aan de hand van $Cn[f, g]$, iets schrijven in het genre van $f(g(x))$ is geen correcte notatie.

C $Rest : \mathbb{N}^2 \to \mathbb{N}$

- Hulpfunctie uit oefenzitting $pred : \mathbb{N} \to \mathbb{N}$: zie hierboven
- Hulpfunctie uit oefenzitting $\dot{-} : \mathbb{N}^2 \to \mathbb{N}$: zie hierboven
- Hulpfunctie $>_0$: $\mathbb{N} \to \mathbb{N}$:

$$\begin{cases} >_0 (x) = 1 & (x > 0) \\ >_0 (x) = 0 & (\text{anders}) \end{cases}$$

Dit kan gedefinieerd worden als

$$\begin{cases} >_0 (0) = 0 \\ >_0 (x+1) = 1 \end{cases}$$

Dit wordt gegeven door

$$Pr[nul, Cn[succ, nul]]$$

- Hulpfunctie uit cursus $+ : \mathbb{N}^2 \to \mathbb{N}$: zie hierboven
- Hulpfunctie uit cursus $* : \mathbb{N}^2 \to \mathbb{N} : *(x,y) = x * y$ wordt gegeven door

$$Pr[nul, Cn[+, p_1^3, p_3^3]] \text{ (zie cursus p.136)}$$

- Hulpfunctie $\neq$: $\mathbb{N}^2 \to \mathbb{N}$:

$$\begin{cases} \neq (a,b) = 1 & (a \neq b) \\ \neq (a,b) = 0 & (a = b) \end{cases}$$

Dit kan ook geschreven worden als

$$\neq (a,b) = >_0 ((a \dot{-} b) + (b \dot{-} a))$$

Dit wordt gegeven door

$$Cn[>_0, Cn[+, Cn[\dot{-}, p_1^2, p_2^2], Cn[\dot{-}, p_2^2, p_1^2]]]$$

- De restfunctie kan gedefinieerd worden als $Rest : \mathbb{N}^2 \to \mathbb{N}$

$$\begin{cases} Rest(0,b) = 0 & \\ Rest(a+1,b) = Rest(a,b) + 1 & (Rest(a,b) \neq b-1) \\ Rest(a+1,b) = 0 & (Rest(a,b) = b-1) \end{cases}$$

Het gedrag van de functie bij 0 werd niet gespecifieerd, dus hier moet je niet speciaal rekening mee houden. Deze functie kan geschreven worden als

$$\begin{cases} Rest(0,x) = 0 \\ Rest(y+1,x) = (Rest(y,x)+1) * (\neq (Rest(y,x), x-1)) \end{cases}$$

Let op, de argumenten moeten van plaats gewisseld worden! Primitieve recursie werkt met 0 en $(y+1)$ in het laatste argument! Als we $Rest'(a,b)$ definiëren als $Rest(b,a)$ dan wordt $Rest'(a,b)$ gegeven door de primitief recursieve functie

$$Pr[nul, Cn[*, Cn[succ, p_3^3], Cn[\neq, p_3^3, Cn[pred, p_1^3]]]]$$

Nu kunnen we $Rest(b,a)$ opstellen als

$$Cn[Rest', p_2^2, p_1^2]$$

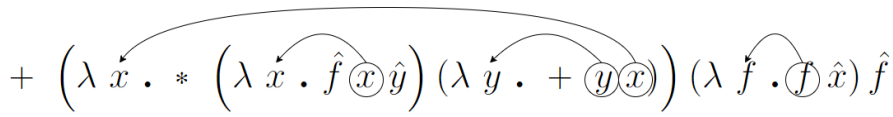
Vaak gemaakte fouten:

- Een haskell-achtige definitie is onvoldoende. We willen een definitie voor de functie in termen van de primitief recursieve functies die gezien werden in de cursus.

4 Lambda calculus (3p)

A (2p) In lambda-calculus zijn de voorkomens van variabelen vrij of gebonden. Omcirkel bij onderstaande expressie de gebonden variabelen en wijs met pijl de overeenkomstige lambda aan, zoals in:

Ongebonden variabelen duid je aan met een hoedje ($\hat{x}$). Doe het eerst in het klad en schrijf het dan netjes over: kladwerk hieronder wordt niet bekeken.



B (3p) Beschrijf in natuurlijke en beknopte taal welke functie of waarde voorgesteld wordt aan de hand van onderstaande lambda-expressie

$$(\lambda\ v\ .\ (\lambda\ f\ .\ f\ v)\,(\lambda\ f\ .\ exp\ f\ v))$$

A Zie hierboven
Vaak gemaakte fouten:

- Hoedjes niet vergeten op de f (2 keer)
- $+, *$ zijn geen variabelen hier, maar specifieke functies.

B $(\lambda\ v\ .\ (\lambda\ f\ .\ f\ v)\,(\lambda\ f\ .\ exp\ f\ v))$
Deze functie neemt een variabele v en past dan een functie f toe op v. De functie f wordt gegeven door $(\lambda\ x\ .\ exp\ x\ v)$, voor de duidelijkheid vervangen we de variabele f in het laatste deel door een nieuwe variabele x die nergens anders gebruikt wordt. Dus in standaard wiskundige notatie wordt f gegeven door $f(x) = x^v$. Zoals al gezegd wordt deze functie toegepast op v, dus we krijgen als gedefinieerde functie een functie g die een element v mapt op $g(v) = f(v) = v^v$.

Merk op dat $(\lambda\ v\ .\ exp\ v\ v)$ een simpeler voorschrift is voor dezelfde functie.
Vaak gemaakte fouten:

- Het is niet omdat we een variabele f noemen, dat deze ook een functie voorstelt. De 2de f in deze lambda-expressie is geen functie. De eerste en tweede f stellen dan ook niet dezelfde variabele voor, ze zijn gebonden aan andere λf.