20.30

1(a) De eerste zes elementen in de canonieke volgorde van L:

c, ab, cc, ccc, aabb, abcc

20.33

i=0	i=1	i=0	i=0	i=2	i=1
j=1	j=0	j=2	j=3	j=0	j=2

En vervolgens:

cccc, aabbc, abccc, ccccc, aaabbb, abcccc

20.387

(b) T controleert eerst of zijn invoer x begint met a of met c. In het laatste geval mag x alleen maar c's bevatten ($i=0$, $j \geq 1$). Als x met a begint, controleert T of x begint met $a^i b^i$, dus evenveel a's als b's. Dit doet T door herhaaldelijk, van buiten naar binnen de eerste a en de laatste b te vervangen door hoofdletters. Als op een gegeven moment geen a meer wordt gevonden maar een B, is inderdaad het aantal a's in x gelijk aan het aantal b's, ervanuitgaande dat de volgorde van de letters klopt.

Vervolgens vergelijkt T het aantal b's in x (i) met het aantal c's (j), door herhaaldelijk van buiten naar binnen de eerste B weer b te maken en de laatste c C te maken.

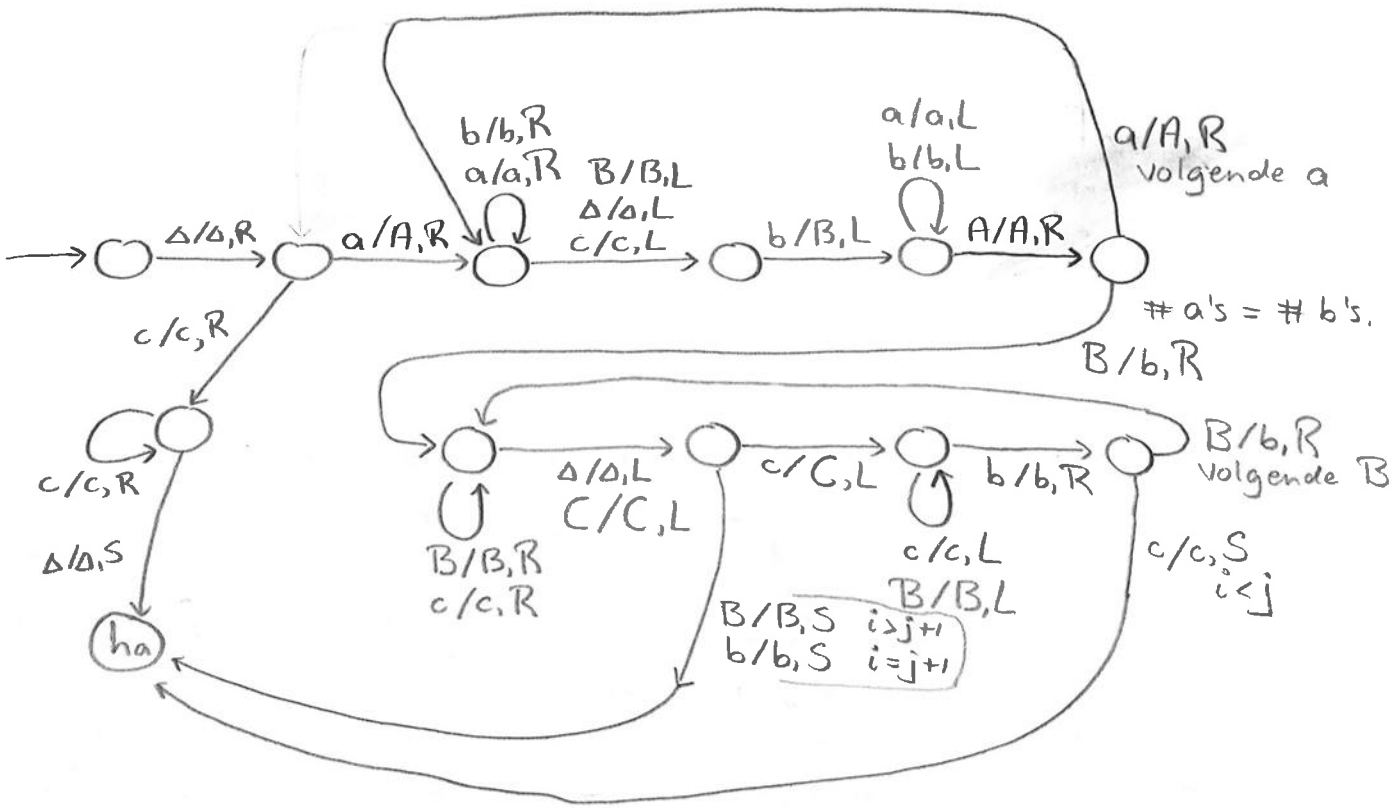
20.43/21.10

Als er geen B meer gevonden wordt maar een c, is $j > i$.

Als er geen c gevonden wordt bij een B, is $j < i$ (maar een b of B)

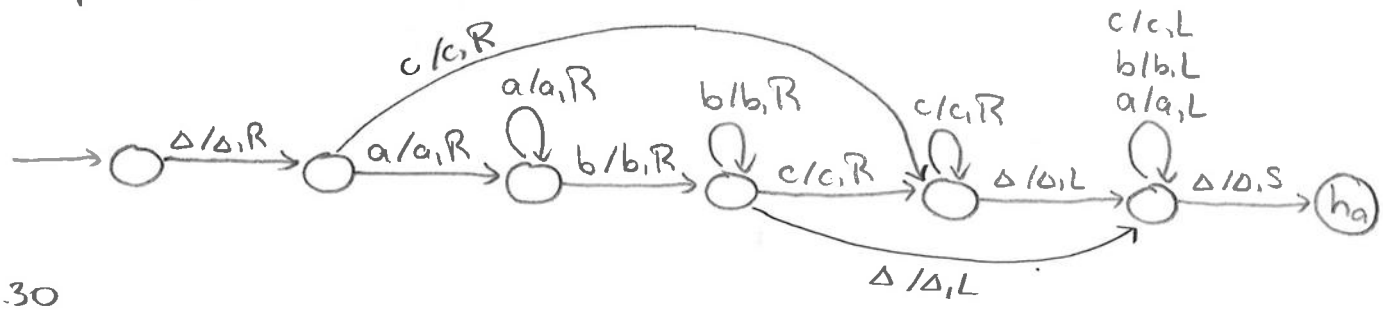
(En als er geen B meer gevonden wordt maar een C, is $j = i$. Dan verwerpen we impliciet.)

21.15



21.25

Door het van buiten naar binnen werken, wordt de volgorde impliciet gecontroleerd. Je kunt het ook expliciet controleren, met de volgende component:

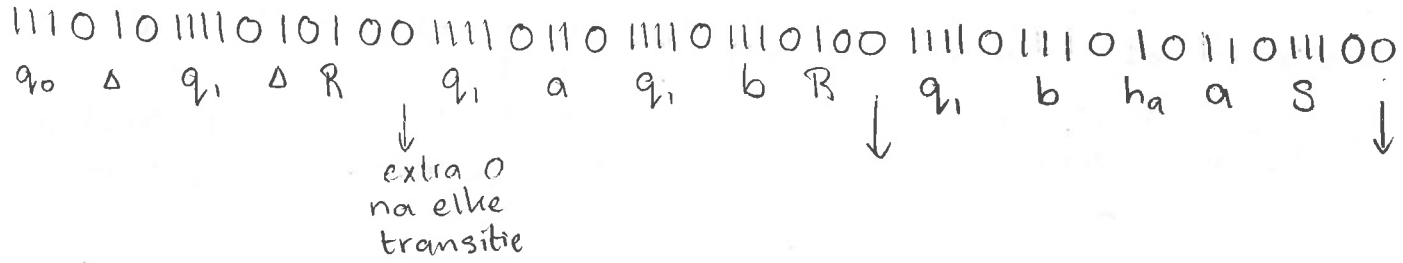


21.30

2) We coderen de transities van T als vijftallen.
Als we een transitie $(p) \xrightarrow{\sigma/\tau,D} (q)$ hebben, geeft dat de volgende bitstring $1^{n(p)}0\,1^{n(\sigma)}0\,1^{n(q)}0\,1^{n(\tau)}0\,1^{n(D)}0$
waarbij de functie n het nummer van de toestand, het tape symbool, en de richting geeft.

Met $n(q_0)=3$, $n(q_1)=4$, $n(h_a)=1$
$n(\Delta)=1$, $n(a)=2$, $n(b)=3$
$n(R)=1$, $n(L)=2$, $n(S)=3$

krijgen we de volgende codering e(T), met transities van links naar rechts gecodeerd:



21.41

3) L bestaat uit twee deeltalen: strings met $i<j$ en strings met $i>j$. Die genereren we vanuit twee verschillende variabelen: T_1, respectievelijk T_2. Startvariabele is S

$S \rightarrow T_1 \mid T_2$ — T_1 dus voor $i<j$, T_2 voor $i>j$

$T_1 \rightarrow ABCT_1 \mid CT_1 \mid c$ — voor elke A en B ook een C; extra C's toegestaan; eindig met kleine c, zodat j zeker groter dan i wordt

$T_2 \rightarrow T_2ABC \mid T_2AB \mid aB$ — voor elke C ook een A en B; extra A's en B's toegestaan; eindig met aB, zodat i zeker groter dan j wordt.

$BA \rightarrow AB$ $CA \rightarrow AC$ $CB \rightarrow BC$ hoofdletters in goede volgorde zetten

Kleine c (vanuit T_1) zorgt ervoor dat hoofdletters van rechts naar links klein worden, mits ze in goede volgorde staan:

$Cc \rightarrow cc$ $Bc \rightarrow bc$ $Bb \rightarrow bb$ $Ab \rightarrow ab$ $Aa \rightarrow aa$

Kleine a (vanuit T_2) zorgt ervoor dat hoofdletters van links naar rechts klein worden, mits ze in goede volgorde staan:

$aA \rightarrow aa$ $aB \rightarrow ab$ $bB \rightarrow bb$ $bC \rightarrow bc$ $cC \rightarrow cc$

21.56

4a) i) $\chi_L : \Sigma^* \rightarrow \{0,1\}$ wordt gedefinieerd door

$$\chi_L(x) = \begin{cases} 1 & \text{als } x \in L \\ 0 & \text{als } x \notin L \end{cases}$$

21.58

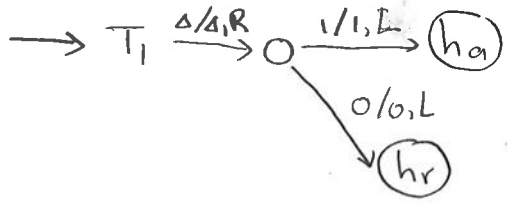
ii) Het domein van de functie χ_L is heel Σ^*. Dat betekent dat T voor elke $x \in \Sigma^*$ de toestand h_a haalt. Gezien als language acceptor accepteert T dus heel Σ^*

22.01

b) i) Stel dat L recursief is (zo'n T_1 bestaat, omdat L recursief is); en dat T_1 een Turingmachine is die χ_L berekent. We kunnen een Turingmachine T_2 construeren die L accepteert, als volgt:

Voor invoer x simuleert T_2 eerst T_1. T_1 eindigt in configuratie $ha\Delta 1$ (als $x \in L$) of configuratie $ha\Delta 0$ (als $x \notin L$).
Vervolgens kijkt T_2 welk symbool er direct rechts van zijn leeskop staat, daar neergezet door T_1. Als het 1 is, accepteert T_2; als het 0 is, verwerpt T_2.

22.07
Op niveau van transities ziet T_2 er als volgt uit:



22.09
ii) Een voorbeeld van een taal die wel recursief opsombaar maar niet recursief is:

$$SA = \{ e(T) \mid T \text{ is een Turingmachine met invoeralfabet } \{0,1\} \text{ en } T \text{ accepteert } e(T) \}$$

22.12
iii) Laat T_1 een Turingmachine zijn die L accepteert, en laat T_2 een Turingmachine zijn die L' accepteert (zulke Turingmachines bestaan, omdat L en L' allebei recursief opsombaar zijn)

Turingmachine T die de karakteristieke functie χ_L berekent kan een Turingmachine met twee tapes zijn. T kopieert eerst zijn invoer $x \in \Sigma^*$ van tape 1 (waar de invoer 'binnenkomt') naar tape 2.
Vervolgens simuleert T op tape 1 T_1 voor invoer x, en tegelijkertijd op tape 2 T_2 voor invoer x.
Hierbij voorkomt T met een speciaal symbool \$ op vakje 0 op beide tapes dat hij op een van de tapes links van de tape zou lopen, zodat T zou crashen. Ook houdt T met een speciaal symbool # op beide tapes 'het einde van de tape' bij.
T simuleert T_1 en T_2 totdat een van beide ha bereikt (als T_1 of T_2 al eerder verworpen heeft, gaat T met de andere Turingmachine verder)
Dit zal zeker gebeuren, omdat T_1 L accepteert en T_2 L' accepteert en $x \in \Sigma^* = L \cup L'$.
Als het T_1 is die x accepteert (dus $x \in L$), veegt T tape 1 schoon tussen (en inclusief) \$ en # en zet een 1 op vakje 1 van die tape.
Als het T_2 is die x accepteert (dus $x \in L'$, ofwel $x \notin L$), veegt T tape 1 ook schoon tussen (en inclusief) \$ en # en zet een 0 op vakje 1 van die tape.
In beide gevallen eindigt T in ha, met de leeskop van tape 1 op vakje 0

22.28.

23.16

5 a)

Accepts-L voldoet aan alle voorwaarden voor de stelling van Rice, want

* de instanties zijn (enkele / losse) Turingmachines
* de eigenschap of $L(T_1) = L$ is een taaleigenschap, want als $L(T_1) = L(T_2)$ voor Turingmachines T_1 en T_2, dan geldt $L(T_1) = L(T_2) = L$ óf $L(T_1) = L(T_2) \neq L$. Beide Turingmachines hebben de eigenschap dus wel (T_1 en T_2) of allebei niet.
* de eigenschap of $L(T_1) = L$ is een niet-triviale eigenschap, want
  de Turingmachine T uit opgave 1 is een ja-instantie
  de Turingmachine T_0: 
  is een nee-instantie

23.23 / 23.25

b) We gebruiken de constructie uit het bewijs van de stelling van Rice

$$\underset{\text{instanties } T_2}{\text{Accepts-}\Lambda} \leq \underset{T_1}{\text{Accepts-L}}$$

Laat T_2 een willekeurige instantie van Accepts-Λ zijn.

We construeren T_1 (een instantie van Accepts-L) als volgt:
T_1 heeft invoeralfabet $\{a,b,c\}$.
T_1 krijgt een invoer x, maar negeert die in eerste instantie.
T_1 gaat met MovePastInput namelijk achter zijn invoer staan.
Vervolgens roept T_1 Turingmachine T_2 aan. Doordat er alleen Δ's onder en rechts van de leeshop staan, simuleert T_1 feitelijk T_2 voor invoer Λ.
Als hierbij ha wordt bereikt, wordt de tape achter de oorspronkelijke invoer x schoongeveegd, en wordt de leeshop weer vóór x gezet, op vakje 0. Daarna wordt Turingmachine T van opgave 1 aangeroepen (voor invoer x dus).

Deze constructie van T_1 uit T_2 is algoritmisch uit te voeren
Als T_2 is ja-instantie van Accepts-Λ, dan accepteert T_2 Λ, dan zal T_1 na simuleren van T_2 voor invoer Λ de tape achter invoer x schoonvegen en ten slotte T uit opgave 1 uitvoeren voor invoer x.
In dat geval is $L(T_1) = L(T) = L$. Dus T_1 is ja-instantie van Accepts-L
(Omdat T een ja-instantie is van Accepts-L en het over een taaleigenschap gaat, heeft ook T_1 de eigenschap. Kortom, dan is T_1 een ja-instantie van Accepts-L. (niet nodig))

Als T_2 een nee-instantie van Accepts-Λ is, dan accepteert T_2 Λ niet.
T_1 zal bij simuleren van T_2 voor invoer Λ dus verwerpen, crashen of in een oneindige loop terechtkomen, onafhankelijk van zijn eigenlijke invoer x. T_1 accepteert dus geen enkele invoer x, ofwel $L(T_1) = \emptyset = L(T_0) \neq L$. Dus T_1 is nee-instantie van Accepts-L.

(Omdat T_0 een nee-instantie van Accepts-L is, en het over een taal eigenschap gaat, is ook T_1 in dit geval een nee-instantie van Accepts-L. (niet nodig))

Inderdaad, een ja-instantie T_2 van Accepts-Λ geeft een ja-instantie T_1 van Accepts-L, en analoog voor nee-instanties.

23.47.

23.49

Voor instanties: Accepts-Λ $\leq$ Accepts-abc
T_2 T_3

Laat T_2 een willekeurige instantie van Accepts-Λ zijn.
We construeren T_3 als instantie van Accepts-abc als volgt:

T_3 = EraseInput $\longrightarrow$ T_2 T_3 heeft ook invoeralfabet $\{a,b,c\}$.

De constructie van T_3 uit T_2 is duidelijk algoritmisch uit te voeren
Er geldt:

T_2 is ja-instantie van Accepts-Λ $\Leftrightarrow$ T_2 accepteert Λ $\Rightarrow$
T_3 (die T_2 simuleert voor invoer Λ) accepteert elke invoer $x \in \{a,b,c\}^*$ $\Rightarrow$ T_3 accepteert ook $x = abc$ $\Leftrightarrow$ T_3 is ja-instantie van Accepts-abc.

T_2 is nee-instantie van Accepts-Λ $\Leftrightarrow$ T_2 accepteert Λ niet $\Rightarrow$
T_3 accepteert geen enkele invoer $x \in \{a,b,c\}^*$ $\Rightarrow$
T_3 accepteert ook $x = abc$ niet $\Leftrightarrow$ T_3 is nee-instantie van Accepts-abc.

23.57
13.21 Alternatieve reductie voor

instanties Accepts-Λ $\leq$ Accepts-L
T_2 T_1

Laat T_2 een willekeurige instantie van Accepts-Λ zijn.
We construeren T_1 als volgt:
T_1 heeft invoeralfabet $\{a,b,c\}$
T_1 simuleert eerst T van opgave 1 op zijn invoer x.
Als T verwerpt, crasht of in oneindige loop raakt, doet T_1 dat ook.
Als T accepteert, veegt T_1 zijn tape schoon (daarvoor heeft T_1 het begin en het eind van zijn tape bijgehouden), en simuleert vervolgens T_2 vanaf vakje 0_0, ofwel T simuleert T_2 voor invoer Λ.

De rest van het bewijs gaat vergelijkbaar met het oorspronkelijke bewijs.
Ten opzichte van de oorspronkelijke reductie zijn dus T en T_2 omgedraaid.