22.12

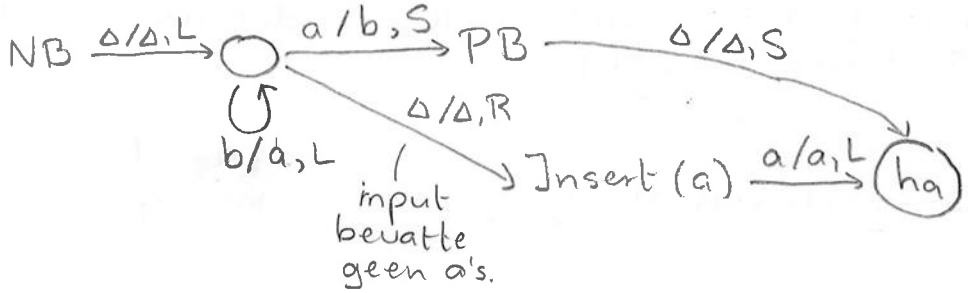
1(a) De eerste zes strings in de canonieke volgorde van $\{a,b\}^*$:
Λ, a, b, aa, ab, ba

22.14

(b)

T_1 loopt naar achteren en vervolgens terug naar voren over de input. De achterste a van de input wordt een b, en alle b's daarachter wordt een a. Als de input geen a bevat wordt aan de voorkant een b ge-insert:

22.17

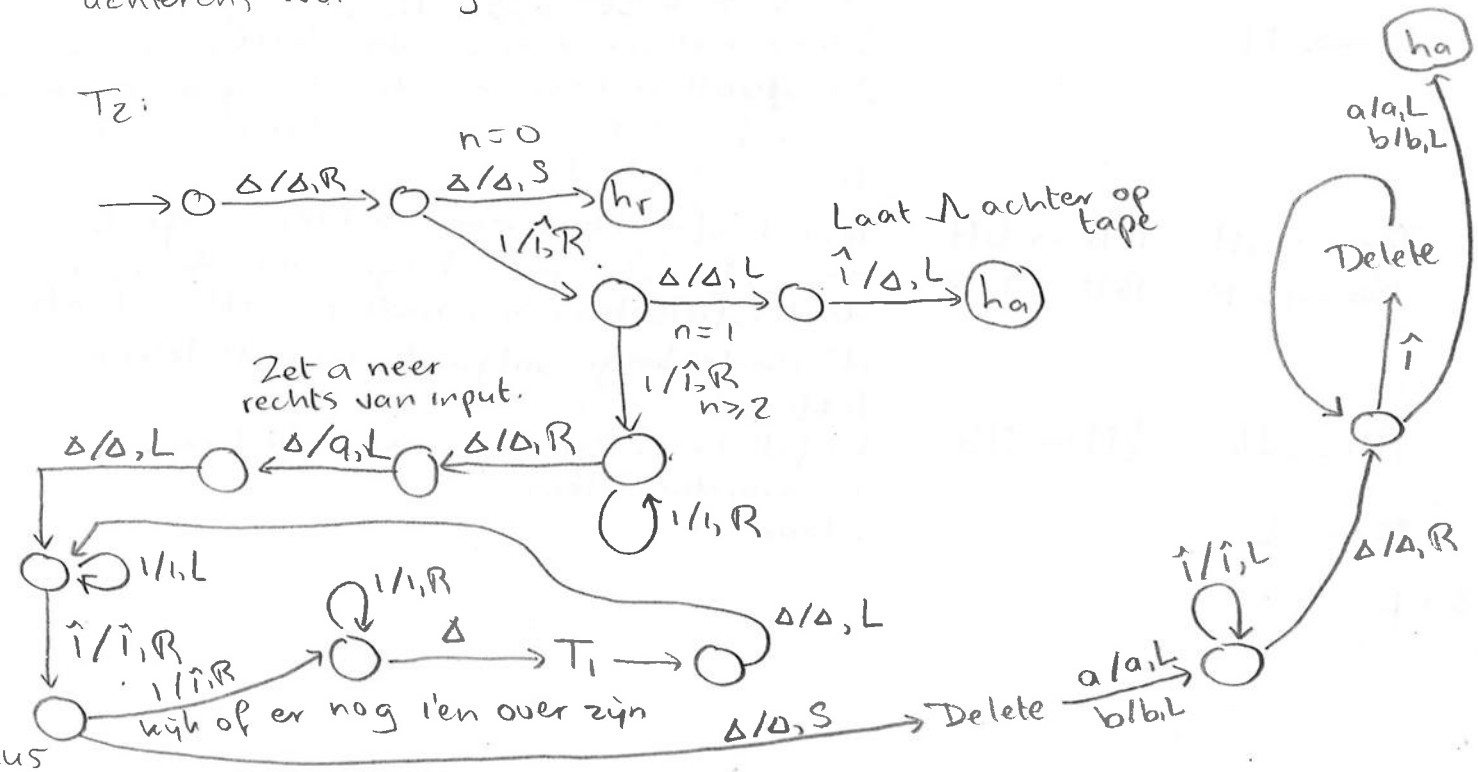


22.21

(c) T_2 controleert eerst of de invoer soms Λ is ($n=0$). In dat geval wordt direct verworpen.
Als de invoer $\hat{1}$ is, wordt de lege string op de tape achtergelaten.
Anders ($n \geq 2$) wordt achter de invoer een a neergezet (het tweede element in de canonieke volgorde van $\{a,b\}^*$). De twee eerste 1'en van n worden gemarkeerd als $\hat{1}$.
Zolang er nog 1'en in de invoerstring over zijn, wordt de eerste 1 gemarkeerd als $\hat{1}$, en wordt T_1 aangeroepen voor de string achter de invoer
Als elke $\hat{1}$ een $\hat{1}$ is geworden, wordt de Δ tussen invoer en het resultaat ge-Dele-d, waarna alle $\hat{1}$'s van de invoer, van voren naar achteren, worden gedelete:

van de invoer gescheiden door een Δ

T_2:



22.45

2(a)

$P(L_1) = \{\Lambda, a, ab, b, ba, bab, babb\}$

22.47

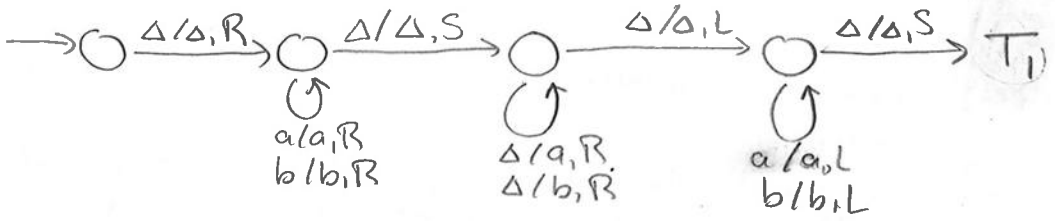
(b)

Dit algoritme werkt niet per se, omdat het mogelijk is dat het een element x van $P(L)$ niet accepteert. Stel immers dat $x \in P(L)$ en dat y_1 de eerste string in de canonieke volgorde van Σ^* is, zódat $xy_1 \in L$. Het is mogelijk dat T_1 voor een eerdere string $y \in \Sigma^*$ in een oneindige lus raakt voor invoer xy. Dan wordt y niet verder verhoogd, zodat ook y_1 niet bereikt wordt. In dat geval wordt x niet geaccepteerd, terwijl $x \in P(L)$

22.58

(c)

T_2 plakt niet-deterministisch een string y achter invoer x, en roept vervolgens T_1 aan:



23.08

3(a)

$S \longrightarrow aAS \mid bBS$	S is startvariabele genereer letters a en b in tweevoud: de kleine letters zijn bedoeld voor de eerste string s, de hoofdletters voor de tweede string s
$S \longrightarrow M$	klaar met genereren van letters voor s. M wordt achteraan de string gezet, maar gaat functioneren als midden van de te genereren string
$Aa \rightarrow aA$ $Ab \rightarrow bA$ $Ba \rightarrow aB$ $Bb \rightarrow bB$	laat hoofdletters naar achteren lopen, naar M. De onderlinge volgorde van de hoofdletters verandert niet, net als de onderlinge volgorde van de kleine letters.
$AM \rightarrow Ma$ $BM \rightarrow Mb$	hoofdletters springen over M heen, en worden klein
$M \rightarrow \Lambda$	klaar.

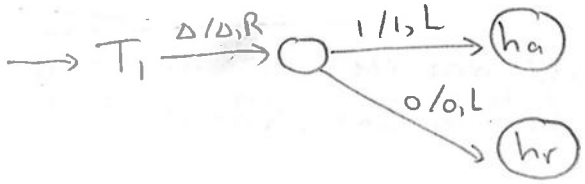
23.17

23.19
(b) $L(G_3)$ bestaat uit de suffixen van de strings uit L.
Immers, in de eerste stap van een afleiding vanuit startvariabele S wordt ET gegenereerd. Vervolgens kan vanuit T een willekeurige string $x \in L(G_2) = L$ gegenereerd worden, achter E.
Daarna kan E met producties $Ea \to E$ en $Eb \to E$ een willekeurig aantal letters aan de voorkant van x opeten, waarna E met productie $E \to \Lambda$ verdwijnt. Dan blijft een suffix van x over.

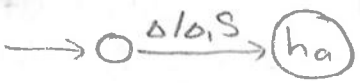
23.27
4 (a)
We noemen een taal L recursief als er een Turingmachine bestaat die de karakteristieke functie van L berekent.

23.29
(b)
Stel dat L recursief opsombaar is. Dan bestaat er een Turingmachine T_1 die de karakteristieke functie van L berekent. De volgende Turingmachine T_2 accepteert L:

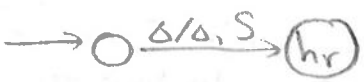
T_1 bereikt voor elke string $x \in \Sigma^*$ de toestand ha



23.33
5(a)
* Accepts $= \Lambda$ heeft als instanties (enkele) Turingmachines
* Accepts $= \Lambda$ gaat over een taaleigenschap van Turingmachines. Immers als $\Lambda \in L(T_1)$ en $L(T_1) = L(T_2)$, dan is ook $\Lambda \in L(T_2)$
* Accepts $= \Lambda$ gaat over een niet-triviale eigenschap. Immers, de volgende Turingmachine is een ja-instantie van Accepts $= \Lambda$



En de volgende Turingmachine is een nee-instantie van Accepts $= \Lambda$



23.42
(b) Aan te tonen:

	Accepts $= \Lambda$	$\leq$	ExistsXReachesState
instanties:	T_1		(T_2, q)

Laat T_1 een willekeurige instantie van Accepts $= \Lambda$ zijn.
Turingmachine
We construeren vanuit T_1 een instantie (T_2, q) van ExistsXReachesState, als volgt.

q is een nieuwe toestand die nog niet in T_1 voorkomt.
We voegen q toe aan de verzameling toestanden van T_1.
We vervangen vervolgens elke transitie van de vorm

Hiermee hebben we er al voor gezorgd dat een ja-instantie T_1 van Accepts-Λ is omgezet in een ja-instantie (T_2, q) van ExistsXReachesState.

Immers:

* T_1 is ja-instantie van Accepts-Λ $\Leftrightarrow$ T_1 accepteert Λ $\Leftrightarrow$ T_2 bereikt toestand q voor invoer $x_2 = \Lambda$ $\Rightarrow$ er bestaat een invoer x_2 (namelijk $x_2 = \Lambda$) waarvoor T_2 toestand q haalt $\Leftrightarrow$ (T_2, q) is een ja-instantie van ExistsXReachesState.

Echter, als T_1 Λ niet accepteert, maar er is wel een andere string x_1 die door T_1 geaccepteerd wordt, dan levert een nee-instantie T_1 van Accepts-Λ een ja-instantie van ExistsXReachesState op. Om dat te voorkomen begint T_2 met EraseInput, en gaat T_2 daarna verder zoals hierboven beschreven. T_2 simuleert dus altijd T_1 voor invoer $x_2 = \Lambda$, met de aanpassing naar toestand q bij accepteren.

Nu geldt: De constructie van (T_2, q) uit T_1 is algoritmisch uit te voeren.

* T_1 is een ja-instantie van Accepts-Λ $\Leftrightarrow$ T_1 accepteert Λ $\Leftrightarrow$ T_2 bereikt toestand q voor elke invoer x_2 $\Rightarrow$ er bestaat een invoer x_2 (bijvoorbeeld $x_2 = \Lambda$) waarvoor T_2 toestand q bereikt $\Leftrightarrow$ (T_2, q) is een ja-instantie van ExistsXReachesState.
* T_1 is een nee-instantie van Accepts-Λ $\Leftrightarrow$ T_1 accepteert Λ niet $\Rightarrow$ T_2 bereikt toestand q voor geen enkele invoer x_2 $\Leftrightarrow$ (T_2, q) is een nee-instantie van ExistsXReachesState.

Inderdaad geldt dus Accepts-$\Lambda \leq$ ExistsXReachesState.
Omdat Accepts-Λ niet beslisbaar is, is ExistsXReachesState dat zeker niet.