Deadline zondag 6 oktober 2019, 23:59. Instructies volgen.

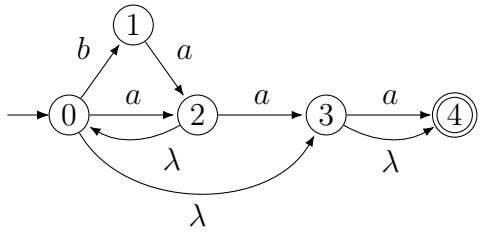
1a. Construeer een DFA M_1 voor de taal $K_1 = \{w \in \{a,b\}^* \mid w \text{ eindigt op } ab\}$.

b. Construeer een DFA M_2 voor de taal $K_2 = \{w \in \{a,b\}^* \mid w \text{ eindigt op } bb\}$.

c. Gebruik de productconstructie om een DFA te krijgen voor de taal $L = K_1 \cup K_2$.

d. Laat zien dat elke deterministische automaat voor L, dus strings die eindigen op ab of bb, tenminste drie toestanden moet hebben.

2. Gebruik eerst **a.** eliminatie van λ-takken, en daarna **b.** de subsetconstructie, om te komen tot een deterministische eindige automaat die dezelfde taal accepteert als onderstaande automaat.



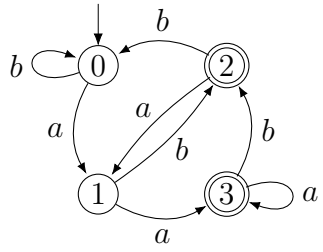
3. De operatie *mirror* of reversal van een string over Σ keert de string w om. Dus als $x = a_1 a_2 \dots a_n$ dan is $x^R = a_n \dots a_2 a_1$, waarbij $a_i \in \Sigma$. De mirror L^R van een taal $L \subseteq \Sigma^*$ keert alle strings in de taal om.

Als bijvoorbeeld $L = \{w \in \{a,b\}^* \mid w \text{ begint met } abb\}$, dan is $L^R = \{w \in \{a,b\}^* \mid w \text{ eindigt op } bba\}$.

a. Beschrijf nauwkeuring hoe een DFA voor de taal L omgezet kan worden in een DFA voor de taal L^R.

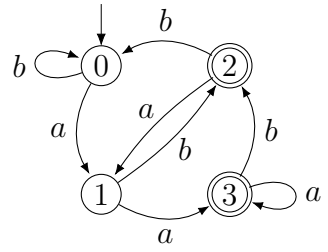
Hierbij mag je aannemen dat de productconstructie, het verwijderen van λ-takken, en de subsetconstructie, bekend zijn.

b. Contrueer een DFA voor de taal L^R, waarbij de oorspronkelijke taal L gespecificeerd wordt door onderstaande DFA.



Deadline donderdag 31 oktober 2019, 23:59. Instructies, zie website.

4. Gebruik een algoritme (*state elimination* van Brzozowski *et* McCluskey, of het algebraïsch oplossen van taal-vergelijkingen) om een reguliere expressie te vinden voor de taal van onderstaande automaat.



5. De Myhill-Nerode relatie, $x \equiv_L y$ desdals voor alle z geldt $xz \in L \Leftrightarrow yz \in L$, verdeelt de taal L in equivalentieklassen van *indistinguishable* strings.

Bekijk de taal $K = a^* + b^*$.

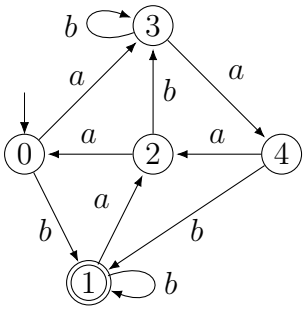
a. Beschrijf elk van de equivalentieklassen van $\equiv_K$.

b. Geef voor elk van de equivalentieklassen de taal $K/x = \{\ z \in \{a,b\}^* \mid xz \in K\ \}$ (voor x in de klasse). (In het college heet deze taal de 'toekomst' van x in K.)

c. Leg uit hoe we met behulp van de equivalentieklassen een eindige automaat voor K kunnen maken (en doe dat dan ook).

6. Minimaliseer de volgende deterministische eindige automaat.

Geef voldoende uitleg, niet alleen het eindresultaat.



Deadline donderdag 21 november 2019, 23:59. Instructies, zie website.

7. Geef context-vrije grammatica's voor de volgende talen (met wat uitleg natuurlijk).
a. $K = \{\ a^{i_1}b^{j_1}a^{i_2}b^{j_2} \mid (i_1 = j_1 \text{ en } i_2 = j_2) \text{ of } j_1 = j_2\ \}$
b. $L = \{\ a^n b^m \mid 0 \leq m \leq 2n\ \}$

8. Bekijk de CFG $G = (\{S, X, Y\}, \{a, b\}, P, S)$, met producties

$$S \to XY \qquad X \to a \mid aX \qquad Y \to ab \mid bY \mid X$$

a. Laat zien dat G dubbelzinnig (*ambiguous*) is.
b. Laat zien dat $L(G)$ regulier is.
c. Geef een rechts-lineaire grammatica voor $L(G)$.

9. Gegeven is de grammatica met producties $S \to SS \mid SaSb \mid \lambda$.
Bepaal een equivalente grammatica in Chomsky normaalvorm.
Volg netjes de vier stappen uit het algoritme, leg uit wat je doet, en geef de tussenresultaten.

Deadline donderdag 12 december 2019, 23:59. Instructies, zie website.

10. Construeer een deterministische stapelautomaat voor de taal

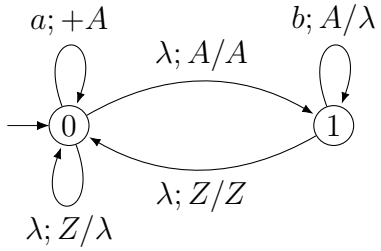
$L = \{\ a^n b^m a^{n+m} \mid m, n \geq 0\ \}$

Let op dat ook $m = 0$ en $n = 0$ zijn toegestaan.

(Ook een niet-deterministische automaat levert al punten op.)

11. Onderstaand het diagram van een stapelautomaat M. Zoals gebruikelijk is Z het initiële stapelsymbool.

a. Construeer, volgens de standaard constructie, een CFG G zó dat $L(G) = L_e(M)$, de lege stapel-taal van M.



Een variabele A heet *live* als $A \Rightarrow^* w$ voor een terminale string w.

b. Bepaal de live variabelen van de CFG G die je in het vorige onderdeel hebt geconstrueerd.

12. Laat $L \subseteq \Sigma^*$, en $\sigma \in \Sigma$. De operatie $:\sigma$ geeft alle strings uit L waarvan het laatste symbool σ verwijderd is, dus $L{:}\sigma = \{\ x \mid x\sigma \in L\ \}$.

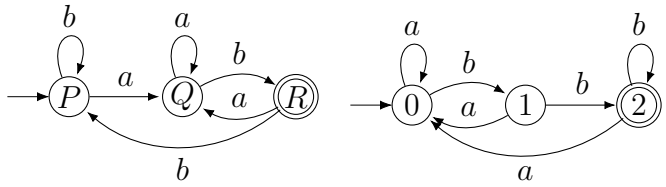
Let op dat alleen strings die eindigen op de gekozen letter σ meedoen.

Bijvoorbeeld, als $K = 0^*1^* + (10)^*$, dan is $K{:}0 = 0^* + (10)^*1$ en $K{:}1 = 0^*1^*$.

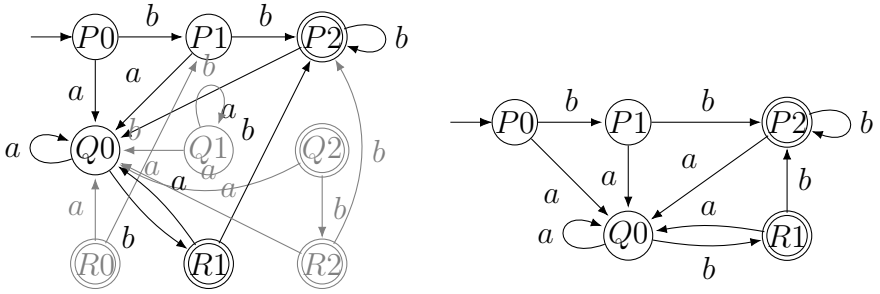
Gebruik stapelautomaten om te laten zien dat de context-vrije talen gesloten zijn onder de operatie $:\sigma$. Dus, geef een constructie die, gegeven een stapelautomaat M voor de taal L, een nieuwe stapelautomaat M_1 maakt voor de taal $L{:}\sigma$.

Leg uit wat je aan het doen bent, zoals bij alle opgaven.

1ab. De twee automaten voor eindigen op ab resp bb. Toestanden komen overeen met de prefixen van de string.



c. Als we de volledige productconstructie uitvoeren op de twee automaten krijgen we $3{\cdot}3 = 9$ toestanden, zie hieronder links. Een gedeelte van die toestanden wordt niet bereikt vanuit de begintoestand en kan weggelaten worden. Omdat er vanuit de weggelaten verzameling toestanden alleen uitgaande pijlen zijn is de automaat na verwijdering nog steeds deterministisch (rechts).



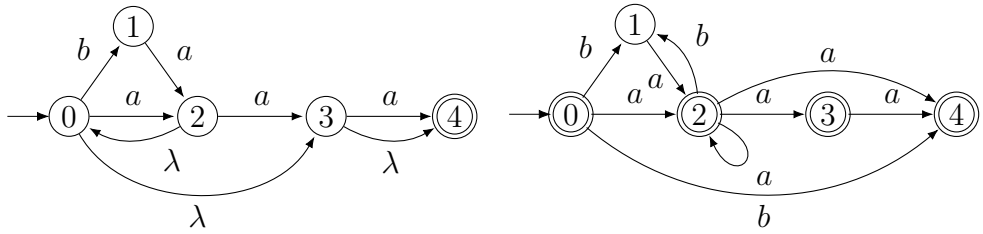
Eindigen op aa of op bb, dus eindigen op b, en tenminste twee letters lang.

d. We laten van drie strings zien dat ze nooit in dezelfde toestand (van een DFA) kunnen eindigen omdat ze verschillende 'toekomst' hebben. Namelijk, als $xz \in L$ terwijl $yz \notin L$, dan $\delta(q_{in}, x) \neq \delta(q_{in}, y)$.

De tabel geeft aan of $xz \in L$ voor de gegeven waardes van x, z. Als twee kolommen verschillen kunnen de bijbehorende strings niet in dezelde toestand eindigen.

$x =$	λ	a	bb
$z = \lambda$	×	×	✓
b	×	✓	✓

2a. Alle toestanden van waaruit een accepterende toestand is te bereiken via alleen λ-takken worden zelf ook accepterend. Als q te bereiken is vanuit p met alleen λ-takken, dan worden alle takken vanuit q naar p gekopieerd. Oorspronkelijke λ-takken worden verwijderd.



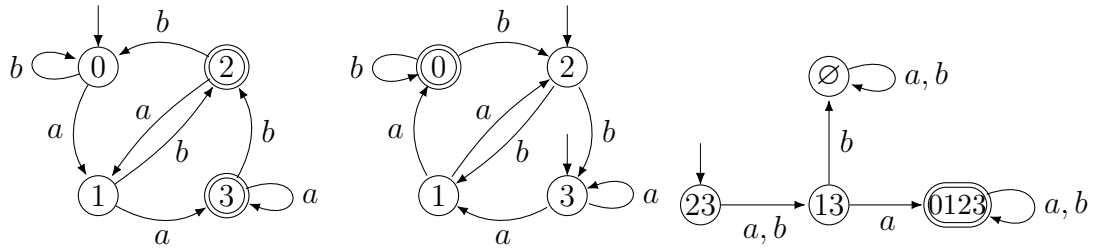
3a. Een pad in de automaat voor een string in de taal L begint in een begintoestand, volgt de takken, en eindigt in een accepterende toestand. Voor het spiegelbeeld moeten we dus de takken in omgekeerde volgorde volgen, van accepterende toestand naar begintoestand.

Als we in de oorspronkelijke automaat M de takken omkeren, en begin- en accepterende toestanden omwisselen hebben we in principe de juiste automaat M^R, maar zijn er twee

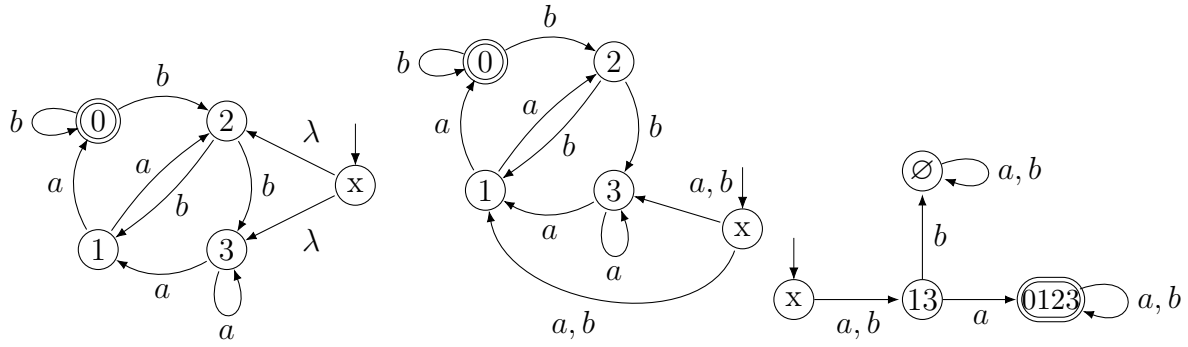
technische problemen: (1) de automaat M^R is waarschijnlijk niet langer deterministisch, (2) zou meerdere begintoestanden kunnen krijgen.

Beide problemen worden verholpen door de subsetconstructie uit te voeren: normaal begint de subsetconstructie met (een verzameling van) één toestand, maar we kunnen goed met een andere verzameling toestanden beginnen.

3b. Het uitwerken van een voorbeeld is een goede manier om je ideeen te testen.

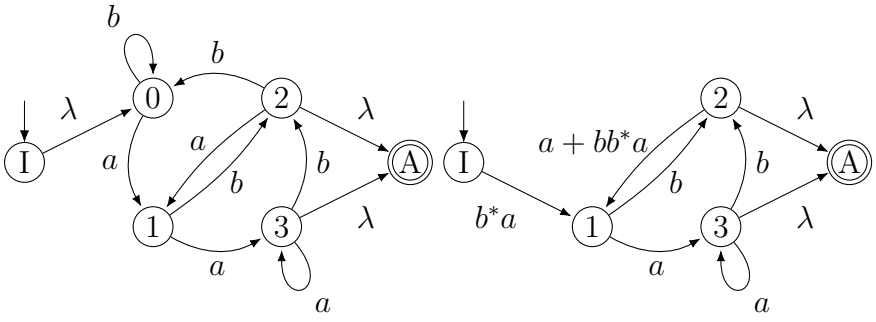


Mocht je je ongemakkelijk voelen met (tijdelijk) meerdere begintoestanden, dan is er een *alternatief*. Kies een nieuwe begintoestand, en verbind deze met de oude eindtoestanden door middel van λ-takken. Daarna kunnen we de λ-takken verwijderen met de standaard constructie.

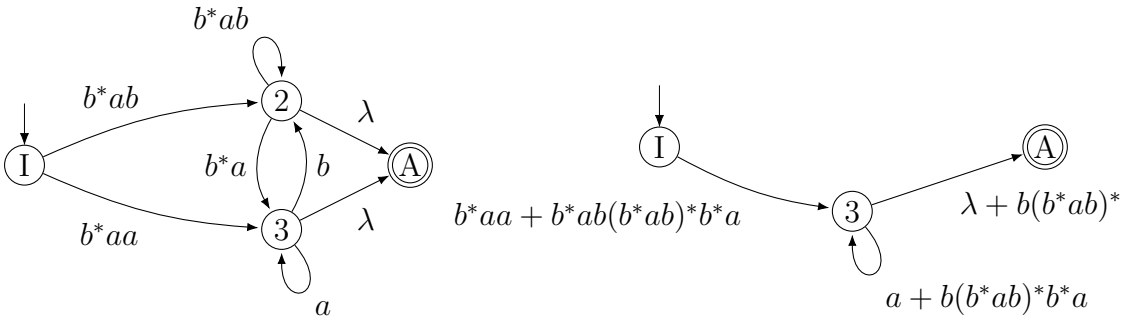


4. Brzozowski *et* McCluskey. Begin met het toevoegen van speciale begin en acceptatie toestanden. Reduceer nu een voor een alle oorspronkelijke toestanden. Dat mag in elke volgorde. Dat kan soms schelen wat betreft de hoeveelheid werk, maar hier nemen we de gewone volgorde.

Reduceer toestand 0. Dat geeft een label $a+bb^*a$ dat ik daarna vervang door het equivalent b^*a.



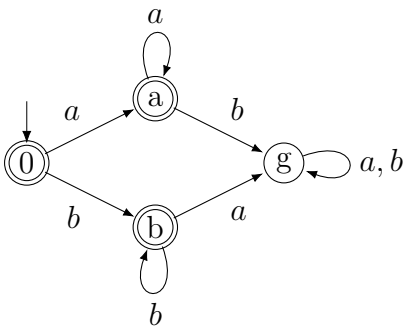
Daarna toestand 1 en 2.



Als we tenslotte toestand 3 reduceren krijgen we de gruwelijke expressie $(b^*aa+b^*ab(b^*ab)^*b^*a)(a+b(b^*ab)^*b^*a)^*(\lambda + b(b^*ab)^*)$

Als je kijkt naar de uitwerking van **3** dan zie je dat het spiegelbeeld van de taal bestaat uit alle strings met tweede letter a, met expressie $(a+b)a(a+b)^*$. Voor de oorspronkelijke taal, die uit deze opgave, zou ook de expressie $(a+b)^*a(a+b)$ voldoen. Maar helaas levert het algoritme niet zoiets fraais op.

5. $K = a^* + b^*$. De equivalentieklassen van K komen overeen met de toestanden in een minimale deterministische automaat voor K. Er bestaat een DFA met vier toestanden voor K. (We moeten nog laten zien dat dit het minimale aantal is, daar komen we op terug.)



ab. De *equivalentieklassen* van $\equiv_K$ komen overeen met de strings die in elke toestand eindigen. (Elke klasse is dus een taal, die we krijgen door die toestand als accepterende toestand te kiezen.) De klasse van g (de garbage toestand) bestaat uit alle strings die zowel een a als een b bevatten.

De *toekomst* van elke klasse (toestand) zijn alle strings waarmee we kunnen vervolgen zodat de totale string in K zit. (De taal door die toestand als begin te kiezen.)

toestand	klasse		toekomst
0	$\{\lambda\}$	$[\lambda]$	$a^* + b^*$
a	aa^*	$[a]$	a^*
b	bb^*	$[b]$	b^*
g	$(aa^*b + bb^*a)(a + b)^*$	$[ab]$	$\varnothing$

Aan de toekomst kun je zien dat de automaat minimaal is: die is namelijk voor elke toestand anders. Equivalente toestanden zouden gelijke toekomst hebben.

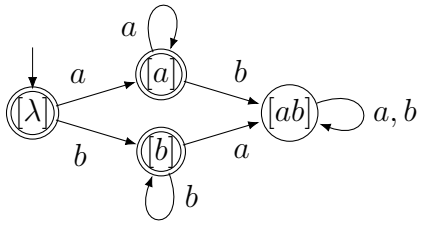
c. De klassen partitioneren $\{a, b\}^*$ in disjuncte verzamelingen. We noteren $[u]$ voor de klasse waar string u toe behoort. De transities van de automaat zijn van de vorm $[u], \sigma, [u\sigma])$.

De vier toestanden kunnen gerepresenteerd worden door de klassen, of door uit elk van die klassen een vertegenwoordiger te kiezen: $[\lambda]$, $[a]$, $[b]$ en $[ab]$.

De begintoestand is $[\lambda]$, de klasse van de lege string. Eindtoestand zijn klassen die binnen de taal K liggen, dus $[\lambda]$, $[a]$ en $[b]$. (Dit zijn ook precies de klassen zie de string λ in hun toekomst hebben.)

Hier is de transitietabel. Er staat bijvoorbeeld $[aa] = [a]$ omdat zowel aa als a element zijn van de klasse aa^*. We kennen het plaatje.

	a	b
$[\lambda]$	$[a]$	$[b]$
$[a]$	$[aa] = [a]$	$[ab]$
$[b]$	$[ba] = [ab]$	$[bb] = [b]$
$[ab]$	$[aba] = [ab]$	$[abb] = [ab]$



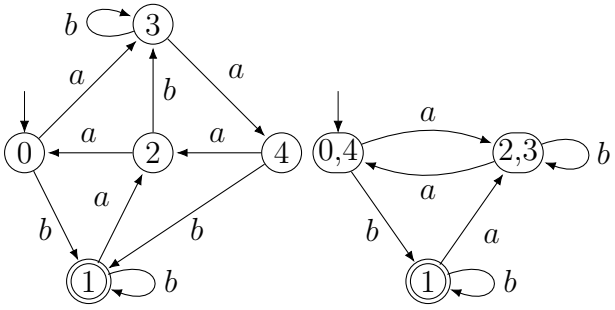
6. We markeren paren toestanden (p, q) die niet equivelent zijn, stap 0, te beginnen met accepterende vs. niet-accepterende toestanden.

Stap 1. Markeer $(0, 2)$ want $\delta(0, b) = 1$ en $\delta(2, b) = 3$, en het paar $(1, 3)$ is zelf gemarkeerd. Idem $(0, 3)$. Idem $(2, 4)$. Idem $(3, 4)$.

Stap 2. Geen nieuwe paren gevonden. Bijvoorbeeld $(\delta(0, a), \delta(4, a)) = (3, 2)$, en $(\delta(0, b), \delta(4, b)) = (1, 1)$ zijn ongemarkeerd. Idem $(2, 3)$.

1	0			
2	1	0		
3	1	0	.	
4	.	0	1	1
	0	1	2	3

De niet gemarkeerde paren zijn equivalent. Dit zijn $(0, 4)$ en $(2, 3)$. Deze paren toestanden worden samengevoegd.



7a. Splits de taal op in twee delen, via de 'of', en gebruik dat context-vrije talen gesloten zijn onder vereniging.

$L_1 = \{a^n b^n \mid n \geq 0\}^2$. Axioma S_1. Producties $S_1 \to XX \quad X \to aXb \mid \lambda$

$L_2 = a^{i_1} b^n a^{i_2} b^n \mid i_1, i_2, n \geq 0\}$. Axioma S_2

Producties $S_2 \to aS_2 \mid Y \quad Y \to bYb \mid Z \quad Z \to aZ \mid \lambda$

Tenslotte is het uiteindelijke axioma S, met de extra producties $S \to S_1 \mid S_2$ (voor de vereniging).

b. Voor elke a maximaal twee b's. Axioma S met regels $S \to aS \mid aSb \mid aSbb \mid \lambda$

8. Kijk even wát de taal is die G genereert. (Dat zou anders onderdeel zijn van **b.**)

Definieer $L_Z = \{x \in \{a, b\}^* \mid Z \Rightarrow_G^* w\}$, voor $Z \in \{S, X, Y\}$. Dus $L(G) = L_S$.

Allereerst zien we snel in dat $L_X = \{a^n \mid n \geq 1\} = aa^*$.

Met de productie $Y \to bY$ genereert Y de strings b^*Y, dan afhankelijk van de gekozen andere productie krijgen we b^*ab of b^*X. In dat laatste geval leiden we een string af in $b^*L_X = b^*aa^*$.

Tenslotte $L_S = L_X L_Y$, en die twee talen kennen we al.

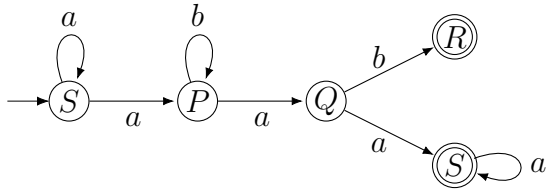
$L(G) = aa^*(b^*ab + b^*aa^*) = aa^*b^*ab + aa^*b^*aa^*$.

a. Er is een string in de taal met twee linkspreferente (*leftmost*) afleidingen (oftewel, met twee verschillende afleidingsbomen).

$S \overset{\ell}{\Rightarrow} XY \overset{\ell}{\Rightarrow} aY \overset{\ell}{\Rightarrow} aX \overset{\ell}{\Rightarrow} aaX \overset{\ell}{\Rightarrow} aaa$ en $S \overset{\ell}{\Rightarrow} XY \overset{\ell}{\Rightarrow} aXY \overset{\ell}{\Rightarrow} aaY \overset{\ell}{\Rightarrow} aaX \overset{\ell}{\Rightarrow} aaa$.

b. Hierboven werd al duidelijk dat de taal regulier is.

c. Teken eventueel een eindige automaat voor de reguliere taal, en noem de toestanden als de variabelen van een grammatica. (Die automaat hoeft niet deterministisch te zijn.)



Omgezet volgens recept wordt dit (met axioma S)

$S \to aS \mid aP \quad P \to bP \mid aQ \quad Q \to bR \mid aS \quad R \to \lambda \quad S \to aS \mid \lambda$.

9. Mijn algoritme heeft onderstaande stappen. Andere *gestructureerde* aanpak zou ook mogen, zolang deze maar niet ad-hoc is, en dus algemeen toepasbaar is.

stap 1. Bepaal *nullable* variabelen. Dat is duidelijk alleen S. We voegen alle producties toe waar S eventueel kan verdwijnen.

$S \to SS \mid SaSb \mid \lambda \mid S \mid aSb \mid Sab \mid ab$.

Nu kunnen we λ-producties weglaten. (We houden dan dezelfde taal, op λ na.)

$S \to SS \mid SaSb \mid S \mid aSb \mid Sab \mid ab$.

stap 2. Verwijder keten-producties (*unit productions*). Dat is hier eenvoudig omdat variabelen niet elkaar kunnen afleiden via ketens.

$S \to SS \mid SaSb \mid aSb \mid Sab \mid ab$.

stap 3. Hernoem terminalen aan de rechterzijde van producties. Voer de variabelen A, B in (in het boek heten die X_a, X_b).

$S \to SS \mid SASB \mid ASB \mid SAB \mid AB \quad A \to a \quad B \to b$.

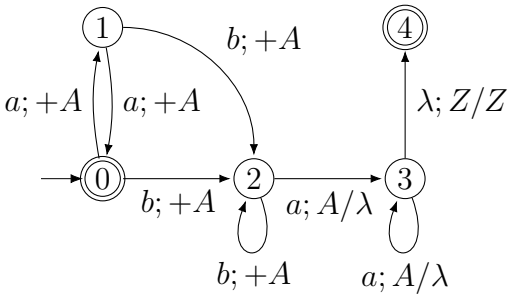
stap 4. Vervang te lange rechterkanten, en voer nieuwe variabelen in voor de suffixen (in dit geval dus voor ASB, SB, AB). Opgelet, we kunnen SB hergebruiken.

$S \to SS \mid SX \mid AY \mid SZ \mid AB \quad A \to a \quad B \to b$.

$X \to AY \quad Y \to SB \quad Z \to AB$.

10. We stapelen A's zolang we eerst a's en dan b's lezen. Als er dan een a komt beginnen we met het poppen van A's.

Als $m = 0$ dan bestaat het woord uit een even aantal a's. Deze woorden accepteren we terwijl we toch blijven stapelen, omdat we niet weten of er nog een b gaat volgen.



11a. De variabelen zijn drie-tupels $[p, X, q]$ met $p \in \{0, 1\}$ en $X \in \{A, Z\}$, dat zijn er acht. Daarnaast een nieuw axioma S.

Als volgt de instructies.

	$S \to [0, Z, 0] \mid [0, Z, 1]$
$(0, a, Z, 0, AZ)$	$[0, Z, 0] \to a\,[0, A, 0]\,[0, Z, 0]$
	$[0, Z, 0] \to a\,[0, A, 1]\,[1, Z, 0]$
	$[0, Z, 1] \to a\,[0, A, 0]\,[0, Z, 1]$
	$[0, Z, 1] \to a\,[0, A, 1]\,[1, Z, 1]$
$(0, a, A, 0, AA)$	$[0, A, 0] \to a\,[0, A, 0]\,[0, A, 0]$
	$[0, A, 0] \to a\,[0, A, 1]\,[1, A, 0]$
	$[0, A, 1] \to a\,[0, A, 0]\,[0, A, 1]$
	$[0, A, 1] \to a\,[0, A, 1]\,[1, A, 1]$
$(0, \lambda, Z, 0, \lambda)$	$[0, Z, 0] \to \lambda$
$(0, \lambda, A, 1, A)$	$[0, A, 0] \to [1, A, 0]$
	$[0, A, 1] \to [1, A, 1]$
$(1, b, A, 1, \lambda)$	$[1, A, 1] \to b$
$(1, \lambda, Z, 0, Z)$	$[1, Z, 0] \to [0, Z, 0]$
	$[1, Z, 1] \to [0, Z, 1]$

b. Werk van achter naar voren.

Eerst: (0) alle variabelen die terminale strings genereren zijn *live*. Dus $[0, Z, 0]$ en $[1, A, 1]$ (vanwege $[0, Z, 0] \to \lambda$ en $[1, A, 1] \to b$).

Dan, herhaald, alle variabelen die strings met eerder gevonden live variabelen (en evt. terminalen) produceren.

(1) S (vanwege $S \to [0, Z, 0]$), $[0, A, 1]$ (want $[0, A, 1] \to [1, A, 1]$), $[1, Z, 0]$ (want $[1, Z, 0] \to [0, Z, 0]$).

(2) Geen nieuwe variabelen.

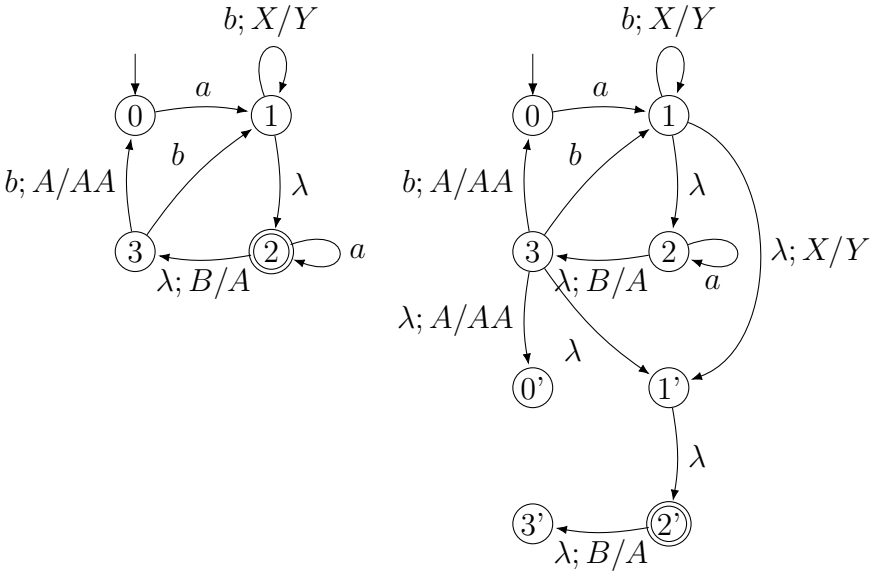
Klaar (en dat lukte me niet in één keer). Dit is als antwoord voldoende, maar we kunnen nu de grammatica reduceren door alle niet-live variabelen te verwijderen (evenals producties die deze variabelen gebruiken). Immers, deze variabelen worden niet gebruikt in afleidingen via terminale strings.

$S \to [0, Z, 0]$

$[0, Z, 0] \to a\,[0, A, 1]\,[1, Z, 0] \mid \lambda$

$[0, A, 1] \to a\,[0, A, 1]\,[1, A, 1] \mid [1, A, 1]$
$[1, A, 1] \to b$
$[1, Z, 0] \to [0, Z, 0]$
Omdat er slechts één productie is voor zowel $[1, A, 1]$ als $[1, Z, 0]$ kunnen we deze variabelen in de overige regels vervangen door de rechterzijde van hun productie. We houden
$S \to [0, Z, 0]$
$[0, Z, 0] \to a\,[0, A, 1]\,[0, Z, 0] \mid \lambda$
$[0, A, 1] \to a\,[0, A, 1]\,b \mid b$
Nu kunnen $[0, Z, 0]$, $[0, A, 1]$ hernoemen naar Z en A, en gewoon Z als axioma nemen. We vinden
$Z \to aAZ \mid \lambda \quad A \to aAb \mid b$

12. Allereerst *het idee*, dat is de helft van het werk.
De PDA M_1 voor $L{:}\sigma$ is grotendeels gelijk aan de PDA M voor L. Extra is dat M_1 op niet-deterministische wijze gokt wanneer σ als laatste letter wordt gelezen. Die σ wordt vervangen door een λ-transitie, en daarna wordt de oorspronkelijke automaat gesimuleerd met allen λ-instructies om te zien of we een accepterende toestand tegenkomen.
Hier *een voorbeeld*, met een paar pushdown instructies voor de eenvoud. Links de automaat voor L, rechts die voor $L{:}b$. Daarin hebben alle toestanden een kopie om na de gok een accepterende toestand te kunnen vinden. Elke b-instructie heeft een kopie die naar de nieuwe toestanden springt, nu met λ ipv b, bv $(3, b, A, 0, AA)$ dan ook $(3, \lambda, A, 0', AA)$. Elke λ-instructie heeft een kopie in de nieuwe toestanden, bv $(2, \lambda, B, 3, A)$ dan ook $(2', \lambda, B, 3', A)$.



Tenslotte *de constructie*.
Zij $M = (Q, \Sigma, \Gamma, \delta, q_{in}, Z_{in}, A)$. Dan is $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, q_{in}, Z_{in}, A_1)$, waarbij
$Q_1 = Q \cup Q'$, met $Q' = \{q' \mid q \in Q\}$ (kopie van de oorspronkelijke toestanden)
$A_1 = \{q' \mid q \in A\}$ (accepterende toestanden verplaatst naar de kopie)
$\delta_1 = \delta \cup \delta' \cup \delta''$, waarbij de nieuwe instructies $\delta' = \{(p, \lambda, A, q', \alpha) \mid (p, \sigma, A, q, \alpha) \in \delta\}$ (gok, laatste σ, ga naar Q'), $\delta'' = \{(p', \lambda, A, q', \alpha) \mid (p, \lambda, A, q, \alpha) \in \delta\}$ (simuleer de λ-instructies in de kopie).