Finite automata

Leereenheid 2

Finite automata

INTRODUCTIE

In leereenheid 1 hebt u kennisgemaakt met drie begrippen die heel belangrijk zijn voor deze cursus: talen, grammatica's en automaten.
U weet nu dat een (formele) taal een verzameling is, bestaande uit strings over een zeker alfabet. Zowel een grammatica als een automaat kan worden gebruikt om exact aan te geven welke strings over een alfabet tot een bepaalde taal behoren, en – vaak impliciet – welke niet. Vaak wordt een grammatica gezien als een mechanisme waarmee de strings van een taal kunnen worden gegenereerd (geproduceerd, voortgebracht, gemaakt), terwijl een automaat wordt gezien als een machine waarmee alle strings van de taal kunnen worden herkend. De automaat accepteert alleen strings uit die taal – vandaar de Engelse term accepter – en wel precies alle woorden uit die taal.
In deze leereenheid bestuderen we de *eindige automaat*; dat is de eenvoudigste automaat die met eindige middelen toch talen met oneindig veel strings kan herkennen. Die eindige middelen bestaan uit: een eindig aantal *toestanden*, een eindig *alfabet* en een (daardoor ook eindig) aantal *toestandsovergangen*. Verder heeft een eindige automaat een begintoestand en een aantal eindtoestanden.
Ondanks het feit dat eindige automaten zo eenvoudig zijn, zijn er toch nog verschillende varianten van. De verschillen zitten in de structuur van de onderliggende graaf. Een eindige automaat is *deterministisch* als er maar één manier is waarop die string door die automaat herkend kan worden. Er bestaan ook niet-deterministische eindige automaten. Het blijkt dat het verschil in structuur tussen beide soorten groot is, maar dat ze toch precies even krachtig zijn.
Deze leereenheid is ook onze eerste kennismaking met het begrip *familie* van talen. We zullen zien dat alle eindige automaten samen de familie van reguliere talen herkennen.

LEERDOELEN

Na het bestuderen van deze leereenheid wordt verwacht dat u

- een definitie van een deterministische eindige automaat kunt geven
- een definitie van een niet-deterministische eindige automaat kunt geven
- een definitie van equivalentie van eindige automaten kunt geven
- bij een gegeven reguliere taal een (niet-)deterministische eindige automaat kunt construeren die die taal accepteert
- de overeenkomsten en verschillen tussen deterministische en niet-deterministische eindige automaten kunt uitleggen
- het verband tussen de begrippen totaliteit en (niet-)determinisme, en de aan- of afwezigheid van λ-overgangen kunt uitleggen
- een gegeven niet-deterministische automaat kunt omzetten in een equivalente deterministische automaat

Studeeraanwijzingen
Bij deze leereenheid hoort chapter 2 van het tekstboek van Linz. Section 2.4 daaruit is facultatief.

De studielast van deze leereenheid bedraagt circa 10 uur.

LEERKERN

Studeeraanwijzing Lees de introductie op chapter 2 in het tekstboek van Linz.

1 Deterministic finite accepters

Studeeraanwijzing Bestudeer in section 2.1 van Linz de subsection Deterministic accepters and transition graphs.

Laten we de definitie van de overgangsfunctie in definition 2.1 nader bekijken. De overgangsfunctie heet δ en is gedefinieerd als een totale functie van $Q \times \Sigma$ naar Q. Dat wil zeggen dat aan iedere waarde (p, a) uit $Q \times \Sigma$ precies één waarde q uit Q wordt toegekend – *precies* één omdat de functie totaal is. In termen van de eindige automaat betekent dat: als de automaat in toestand p is en het symbool a leest, dan moet hij zichzelf in toestand q zetten en naar het volgende invoersymbool gaan. In de onderliggende graaf is er dan een pijl met label a van toestand p naar toestand q.

OPGAVE 2.1
Maak exercise 1 van section 2.1. Neem in plaats van 00001101 de string 0000110.

We gaan nu verder met de *uitgebreide* overgangsfunctie (extended transition function) $\delta : Q \times \Sigma^* \to Q$. We herhalen de recursieve definitie uit Linz:

$$\delta^*(q, \lambda) = q \quad (2.1)$$
$$\delta^*(q, wa) = \delta(\delta^*(q, w), a) \quad (2.2)$$

Probeer in uw eigen woorden uit te leggen wat de uitgebreide overgangsfunctie doet, en in het bijzonder wat regel (2.2) zegt.

De uitgebreide overgangsfunctie legt vast wat het resultaat is van het toepassen van een aantal overgangen (volgens de 'gewone' overgangsfunctie) achter elkaar. Met andere woorden, de uitgebreide overgangsfunctie geeft aan in welke toestand de automaat komt als hij vanuit een gegeven toestand een gegeven *string* leest in plaats van een *symbool*, zoals bij de gewone overgangsfunctie. Overigens is δ^* vooral een hulpmiddel om bij het redeneren over de werking van automaten wat grotere stappen te kunnen nemen dan we met δ kunnen. We hebben δ nog steeds nodig om de eindige automaat te specificeren.
Regel (2.2) van de recursieve definitie zegt dat het lezen van wa vanuit toestand q precies hetzelfde is als: eerst w lezen vanuit q, en vervolgens a lezen vanuit de toestand waarin de automaat na het lezen van w terechtgekomen is. Die laatstgenoemde toestand is precies $\delta^*(q, w)$.
Natuurlijk werkt zo'n recursieve definitie alleen als er een 'bodem' in de recursie zit; daartoe dient regel (2.1).

OPGAVE 2.2
Bepaal $\delta^*(q_0, 111)$ en $\delta^*(q_2, 100010)$ voor de automaat van figure 2.1. U hoeft hiervoor de stappen van de recursieve definitie niet helemaal uit te schrijven (dat mag wel natuurlijk); het gaat erom dat u begrijpt wat δ^* doet.

Studeeraanwijzing Bestudeer in section 2.1 van Linz de subsection Languages and dfa's.

Trap state Een *trap state* is een toestand waar de eindige automaat niet meer uitkomt. Alle uitgaande overgangen zijn een lus naar de trap state. Trap states kunnen final zijn of niet. Een trap state is final als de toestand een eindtoestand is. Nonfinal trap states worden vaak gebruikt om ervoor te zorgen dat de overgangsfunctie totaal is.

We beginnen de bespreking van deze subsection met een paar opgaven over talen en bijbehorende eindige automaten.

OPGAVE 2.3
Laat $\Sigma = \{a, b\}$. Construeer dfa's die de verzamelingen accepteren die bestaan uit:
a alle strings met precies één *a*
b alle strings met minstens één *a*
c alle strings met hoogstens drie *a*'s
d alle strings met ten minste één *a* en precies twee *b*'s

OPGAVE 2.4
Construeer dfa's voor de volgende talen:
a $L_1 = \{x \in \{a, b\}^* : x$ bevat de substring $bab\}$
b $L_2 = \{x \in \{a, b\}^* : x$ bevat geen substring $bab\}$

OPGAVE 2.5
Maak exercise 9 van section 2.1.

OPGAVE 2.6
Maak exercise 10 van section 2.1.

In opgave 2.5 en 2.6 hebben we een constructie gezien waarmee we bij iedere gegeven dfa M een dfa M' kunnen maken zodat M' precies het complement van M accepteert. In leereenheid 4 zien we nog meer van zulke constructies.
Uit de uitwerking bij opgave 2.5 kunnen we nog iets anders afleiden: het totaal zijn van de overgangsfunctie van een dfa en het deterministisch zijn van die dfa zijn twee aparte begrippen. Het determinisme volgt immers uit het definiëren van de verzameling overgangen met behulp van een *functie* van $Q \times \Sigma$ naar Q, terwijl de totaliteit een extra eis aan die functie is. Een dfa zoals gedefinieerd in definition 2.1 is totaal en deterministisch.

Stel dat we de eis weglaten dat de overgangsfunctie van een dfa totaal is. Is er dan nog een manier om vast te stellen dat een bepaalde invoerstring *niet* wordt geaccepteerd?

Ja, maar we moeten dan twee gevallen onderscheiden: voor invoerstrings die niet geaccepteerd moeten worden, geldt óf dat de dfa na het lezen van de gehele string niet in een eindtoestand is, óf dat de dfa blijft 'hangen' (in een eindtoestand of in een niet-eindtoestand) voordat het einde van de invoerstring bereikt is. We kunnen afspreken dat in beide gevallen de string niet geaccepteerd wordt. Voor strings die wel geaccepteerd moeten worden, verandert er niets.
In de literatuur zien we dan ook vaak definities van eindige automaten die niet totaal hoeven te zijn, onder andere omdat we ons dan niet druk hoeven te maken om een nonfinal trap state, en we daardoor kleinere, overzichtelijkere automaten krijgen. Aan de andere kant is het voor sommige resultaten over eindige automaten handig of zelfs noodzakelijk om aan te kunnen nemen dat de overgangsfunctie totaal is (zoals voor het nemen van het complement – zie opgave 2.5 en 2.6). Gelukkig zijn totale eindige automaten een *normaalvorm* voor eindige automaten: voor iedere eindige automaat kunnen we eenvoudig een eindige automaat met een totale overgangsfunctie construeren die dezelfde taal accepteert.

Normaalvorm

OPGAVE 2.7
Laat zien dat (en hoe) voor iedere niet-totale eindige automaat een totale eindige automaat geconstrueerd kan worden die precies dezelfde taal accepteert.

Afspraak

Als u in deze cursus gevraagd wordt een dfa voor een gegeven taal te construeren, dan hoeft u deze niet totaal te maken, tenzij daar expliciet om gevraagd wordt. Dit geldt ook voor opdrachten in JFLAP, want JFLAP accepteert ook niet-totale automaten.

OPDRACHT 2.8
a Maak exercise 11a van section 2.1.
b Teken uw dfa in JFLAP en controleer de gegeven voorbeeldstrings.

Studeeraanwijzing

Bestudeer van section 2.1 in Linz de subsection Regular languages.

Een belangrijk punt in deze subsection is het verschil tussen een taal en een familie van talen. Een *taal* is een verzameling strings, en een *familie van talen* is een verzameling talen. De talen in een familie hebben qua structuur iets met elkaar te maken: voor iedere taal in de familie van *reguliere talen* bijvoorbeeld kan een dfa geconstrueerd worden die die taal accepteert. We leren later nog andere families van talen kennen, en het verband tussen die families en de familie van reguliere talen.

OPGAVE 2.9
Maak exercise 15 van section 2.1.

2 Nondeterministic finite accepters

Studeeraanwijzing

Bestudeer section 2.2 van Linz.

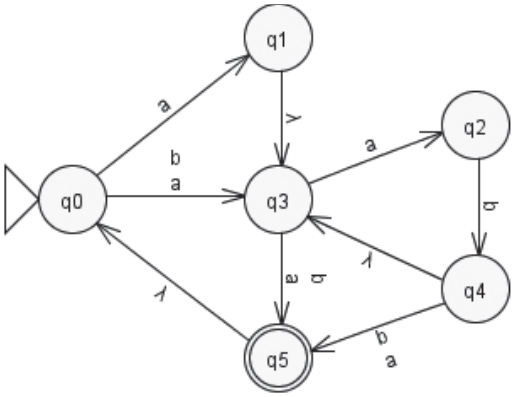
Het belangrijkste verschil tussen een dfa en een nfa is dat in een nfa een toestand meer dan één uitgaande pijl met hetzelfde label mag hebben – zie bijvoorbeeld figure 2.8. Let hier ook op het woord *mag*: een nfa kan toevallig ook een dfa zijn. Met andere woorden: iedere dfa is een nfa, maar niet andersom.

OPGAVE 2.10
Wat zijn de verschillen en overeenkomsten tussen de manier waarop een dfa strings accepteert en de manier waarop een nfa dat doet?

OPGAVE 2.11
Construeer een nfa voor de taal $L = \{01, 010\}^*$. Probeer ook (heel even) een dfa voor L te vinden.

OPGAVE 2.12
Construeer een nfa voor de taal
$L = \{w \in \{a, b\}^* : w$ bevat de substring $bab\}$.

OPGAVE 2.13
Bekijk onderstaande nfa.



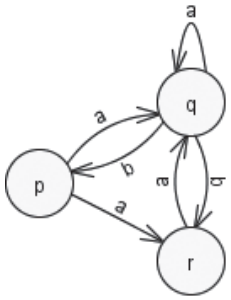
Wat is volgens u de taal die door deze nfa wordt geaccepteerd? Motiveer uw antwoord. U kunt uw antwoord controleren met JFLAP voordat u de uitwerking achteraan deze leereenheid bekijkt: de nfa is beschikbaar in het bestand Jexercise2.2.jff in de map JFLAP Activities | JFLAP Files | JFLAP Exercises .

Zoals in de subsection Definition of a nondeterministic accepter in het tekstboek wordt opgemerkt, zijn er drie belangrijke verschillen tussen de definities van de overgangsfunctie in een dfa en in een nfa. We herhalen beide definities hier en bespreken ze dan:

$\delta : Q \times \Sigma \rightarrow Q$ (dfa, definition 2.1)
$\delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow 2^Q$ (nfa, definition 2.4)

Merk om te beginnen op dat er een vierde verschil tussen beide definities is: de δ van een nfa lijkt qua structuur veel op de δ van een dfa, maar betekent toch iets heel anders. Als een dfa de overgang $\delta(p, a) = q$ heeft, dan kunnen we q weer 'in δ stoppen' om zo nog een stap te doen: bijvoorbeeld $\delta(q, b) = r$. De dfa heeft dan een pijl met label a van toestand p naar toestand q, en vanuit q gaat er een pijl met label b naar toestand r. Als daarentegen een nfa een overgang $\delta(p, a) = \{q, r\}$ heeft, dan kunnen we het resultaat (de verzameling $\{q, r\}$) *niet* weer 'in δ stoppen', want het eerste argument van δ moet een toestand zijn, en niet een verzameling toestanden. Om uit de δ van een nfa te kunnen afleiden hoe de automaat

er uit ziet, is de uitleg onder definition 2.4 onontbeerlijk: als we weten dat $\delta(p, a) = \{q, r\}$, dan zoeken we daarna $\delta(q, \ldots)$ en $\delta(r, \ldots)$ op (waarbij we op de puntjes ieder invoersymbool mogen invullen). Als we dan bijvoorbeeld vinden dat $\delta(q, a) = \{q\}$, $\delta(q, b) = \{p, r\}$ en $\delta(r, a) = \{q\}$, dan weten we dat de automaat in ieder geval de volgende overgangen heeft:



Waaraan kunt u in bovenstaand plaatje zien dat de (onvolledige) automaat die daar getekend is niet-deterministisch is?

De automaat heeft een toestand van waaruit twee pijlen met hetzelfde label vertrekken, en daarom is de automaat niet-deterministisch. Deze automaat heeft zelfs twee van zulke toestanden: vanuit p vertrekken twee pijlen met een a, en vanuit q vertrekken er twee met een b.
Laten we nu deze observatie eens vergelijken met definition 2.4, in het bijzonder met de definitie van δ daar, namelijk $\delta : Q \times (\Sigma \cup \{\lambda\}) \to 2^Q$. Linz heeft er dus voor gekozen om niet-deterministische eindige automaten te definiëren als eindige automaten die niet deterministisch hoeven te zijn, èn die λ-overgangen mogen hebben.

Vindt u dat een logische keuze?

Aan de ene kant is dat een heel logische keuze: iedere automaat die een λ-overgang bevat, is niet-deterministisch (zie opgave 2.14). Aan de andere kant kun je best niet-determinisme hebben zonder λ-overgangen, zoals we net gezien hebben. Net als determinisme en totaliteit in de vorige paragraaf hoeven niet-determinisme en λ-overgangen dus niet per se tegelijkertijd voor te komen. Overigens maakt het niet zoveel uit of je wel of geen λ-overgangen toelaat: eindige automaten zonder λ-overgangen zijn een normaalvorm, dat wil zeggen, voor iedere eindige automaat kun je een eindige automaat maken die geen λ-overgangen bevat maar die wel precies dezelfde taal accepteert. We bespreken die constructie hier echter niet.

OPGAVE 2.14
Waarom is de eindige automaat van figure 2.10 in Linz, met overgangen (q_0, a, q_1), (q_1, λ, q_2) en (q_2, λ, q_0), en eindtoestand q_1, niet-deterministisch?

OPGAVE 2.15
Maak exercise 13 van section 2.2.

OPGAVE 2.16
a Construeer een nfa voor de taal $L_a = \{b\}^*\{a\} \cup \{b\}\{a\}^*\{b\}^*$.
b Construeer een nfa voor de taal $L_b = \{01\}^* \cup \{10\}^*$.

We begonnen deze deelparagraaf met de opmerking dat er drie verschillen zijn tussen de definities van δ in definition 2.1 en definition 2.4. Als eerste noemt Linz het feit dat bij de nfa rechts van de pijl 2^Q staat, terwijl dat bij de dfa Q was. We hebben al gezien dat dit verschil zorgt voor het mogelijke niet-determinisme in een nfa: de waarde van $\delta(q, a)$ is nu een *verzameling* toestanden (met mogelijk meer dan één element), terwijl dat eerst altijd één toestand was.
Als tweede noemt Linz het mogelijke voorkomen van λ-overgangen in een nfa; ook dat verschil hebben we besproken.
Het derde verschil is dat in de overgangsfunctie van een nfa de verzameling $\delta(q, a)$ leeg mag zijn. Dit betekent dan dat er geen overgang is gespecificeerd voor de combinatie (q, a). Deze uitleg is een logisch vervolg op wat we hiervoor besproken hebben:
– als $\delta(q, a) = \{p, r\}$, dan zijn er twee overgangen voor de combinatie (q, a),
– als $\delta(q, a) = \{p\}$, dan is er maar één overgang voor die combinatie, en
– als $\delta(q, a) = \varnothing$, dan is er geen overgang voor die combinatie.

OPGAVE 2.17
Gegeven is de taal $L = \{vwv : v, w \in \{a, b\}^* \text{ en } |v| = 2\}$.
a Is *aabaa* een element van *L*? En *ababba*, *bab* en *baba*?
b Construeer een nfa zonder λ-overgangen die *L* accepteert.
c Construeer nu een dfa voor *L*.

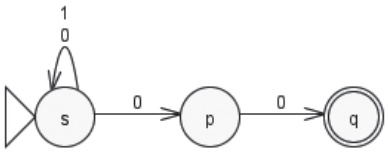
OPGAVE 2.18
Maak exercise 19 van section 2.2.

3 Equivalence of deterministic and nondeterministic finite accepters

Studeeraanwijzing Bestudeer section 2.3 uit het tekstboek van Linz.

In de vorige paragraaf hebben we gemerkt dat het soms makkelijker is een nfa voor een taal te construeren dan een dfa. Maar we hadden daarvoor, bij de complementconstructie, al gezien dat het in sommige gevallen heel handig is als we kunnen uitgaan van een deterministische automaat. Dat is ook vaak zo als we iets willen bewijzen over een specifieke automaat of over automaten in het algemeen.
Gelukkig is de deterministische eindige automaat een normaalvorm: voor *iedere* willekeurige eindige automaat kunnen we een *equivalente* deterministische eindige automaat construeren.
Het grote verschil tussen een nfa en een dfa is dat je in een nfa voor één invoerstring verschillende paden vanuit de begintoestand kunt hebben (zie bijvoorbeeld de uitwerking van opgave 2.15). Het idee achter de constructie om van een willekeurige nfa een equivalente dfa te maken (de procedure nfa-to-dfa uit Linz) is om die verschillende paden als het ware samen te vatten in één pad in de dfa. Iedere toestand op dat pad in de dfa houdt dan bij in welke toestanden de nfa op dat moment (op die positie van de invoerstring) allemaal kan zijn. Het bijhouden van meerdere nfa-toestanden in één dfa-toestand gaat het makkelijkst met een verzameling, en iedere dfa-toestand heeft dan als label een deelverzameling van de verzameling toestanden van de nfa.
De constructie verandert niet als de nfa λ-overgangen bevat, maar wordt er wel wat lastiger door uit te voeren. We kijken dus eerst naar een voorbeeld zonder λ-overgangen (opgave 2.19).

OPGAVE 2.19
Gegeven is de nfa M:



Construeer een equivalente dfa.

In de uitwerking van opgave 2.19 merkten we op dat de resulterende dfa totaal was, terwijl de oorspronkelijke nfa dat niet was. Als we de procedure nfa-to-dfa heel precies uitvoeren, is het resultaat *altijd* een totale deterministische automaat, want de procedure zegt expliciet dat we daarvoor moeten zorgen. We vinden het echter ook goed genoeg om tijdens het uitvoeren van nfa-to-dfa alleen die toestanden en overgangen te introduceren waar we op natuurlijke wijze tegenaan lopen. Dat wil zeggen, als de dfa-onder-constructie een toestand $\{p, q\}$ heeft, en de oorspronkelijke nfa heeft noch vanuit p noch vanuit q een overgang met label a, dan geven we de dfa ook geen overgang met label a vanuit toestand $\{p, q\}$. Mocht het later wenselijk zijn om toch een *totale* deterministische automaat te hebben, dan is een nonfinal trap state met bijbehorende overgangen zo toegevoegd.

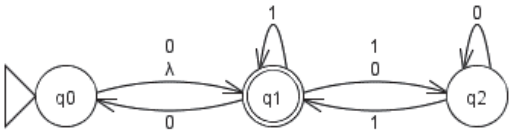
Verder zegt de procedure nfa-to-dfa niets over het aanhouden van een bepaalde volgorde bij het afwerken van de nieuw gevonden toestanden en de invoersymbolen. Dat hoeft natuurlijk ook niet, maar enige systematiek geeft wel meer overzicht. We raden u dan ook aan om, net als in de procedure, altijd te beginnen met $\{q_0\}$, en om dan eerst voor alle $a \in \Sigma$ te bepalen in welke (eventueel nieuwe) toestand(en) u met die a vanuit $\{q_0\}$ kunt komen. Vervolgens werkt u dan de nieuwe toestanden af, bijvoorbeeld in de volgorde waarin u ze gevonden hebt. Zorg steeds dat u alle mogelijke overgangen vanuit een toestand bekeken hebt voordat u met de volgende toestand verder gaat. Op deze manier krijgt u niets meer en niets minder dan alle toestanden en overgangen die u nodig hebt.

We gaan nu ook opgaven maken waarin een nfa *die λ-overgangen bevat* 'deterministisch gemaakt moet worden'. Dat is duidelijk een stuk moeilijker dan wanneer de nfa *geen* λ-overgangen bevat. Het kan enorm helpen om eerst een tabel te maken waarin u voor iedere toestand q uit de nfa en voor ieder invoersymbool a aangeeft welke elementen de verzameling $\delta^*(q, a)$ bevat. Let op de *: die geeft aan dat die ene a gelezen kan worden door meer dan één overgang te volgen! In het uiterste geval namelijk eerst een aantal λ-overgangen, dan een a-overgang en dan weer een aantal λ-overgangen.

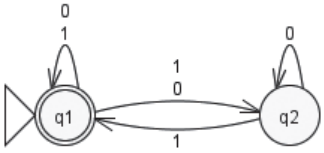
OPGAVE 2.20
Maak exercise 1 van section 2.3.

OPGAVE 2.21
Maak exercise 2 van section 2.3. Daar wordt gevraagd om de nfa van exercise 13 van section 2.2 te converteren naar een equivalente dfa.

OPDRACHT 2.22
a Construeer een equivalente dfa bij de volgende nfa :



b Gegeven is de volgende nfa uit bouwsteen opdracht02.22b.jff. Is deze nfa equivalent aan die van onderdeel a (bouwsteen opdracht02.22a.jff)? Gebruik JFLAP om dit te controleren.



OPGAVE 2.23
Maak exercise 9 van section 2.3.

OPGAVE 2.24
Als we een gegeven nfa met behulp van de procedure nfa-to-dfa omschrijven naar een dfa, kan die dfa dan λ-overgangen bevatten?

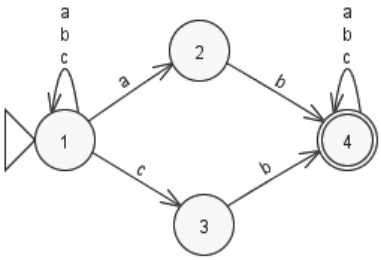
JFLAP heeft ook een mogelijkheid om een gegeven nfa te converteren naar een dfa: kies Convert | Convert to DFA. U kunt via internet in de JFLAP Tutorial opzoeken hoe dit precies werkt. Er zijn twee mogelijkheden: u laat JFLAP per toestand uitzoeken welke overgangen en nieuwe toestanden de dfa nodig heeft (met de knop State Expander), of u geeft dit zelf aan per toestand (met de knop Expand Group on Terminal). De eerste manier kan heel handig zijn om de procedure nfa-to-dfa te leren (met JFLAP in de rol van docent die laat zien hoe de constructie werkt), maar het moge duidelijk zijn dat het minstens zo belangrijk is om de procedure helemaal zelf te oefenen. De tweede manier gebruikt een wat onhandige notatie, met een naam èn een label voor iedere toestand; de naam is daarbij wat verwarrend, want die komt meestal ook voor in de nfa maar heeft daar niets mee te maken: in het *label* wordt bijgehouden in welke verzameling toestanden de nfa op dat moment kan zijn.

OPDRACHT 2.25
Open bouwsteen opdracht02.22a.jff van opdracht 2.22. Converteer deze nfa met behulp van JFLAP naar een dfa. Probeer beide hierboven genoemde methoden uit.

ZELFTOETS

1 a Construeer een dfa voor de taal $K = \{w \in \{a, b\}^* :$ als w een a bevat, dan bevat w precies twee b's$\}$.
b Construeer nu een nfa voor K.

2 Gegeven is de eindige automaat M:



a Is M totaal? Is M deterministisch? Motiveer uw antwoord.
b Bepaal $\delta^*(q, s)$ voor alle $q \in Q$ en alle $s \in \Sigma$.
c Construeer een dfa die equivalent is met M.

3 Laat zien dat alle eindige talen regulier zijn.

4 Kunt u beredeneren waarom de taal $L = \{a^n b^n : n \geq 1\}$ niet regulier is?

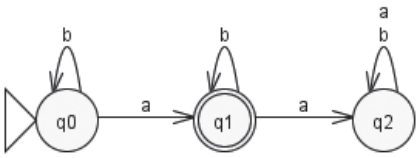
TERUGKOPPELING

1 Uitwerking van de opgaven

2.1 Voor 0001 komt de automaat na de begintoestand q_0 achtereenvolgens in toestand q_0, q_0, q_0 en q_1, en dan is de hele string gelezen. De laatste toestand is een eindtoestand, dus 0001 wordt geaccepteerd.
Voor 01101 komt de automaat na de begintoestand q_0 achtereenvolgens in toestand q_0, q_1, q_2, q_2 en q_1, dus 01101 wordt geaccepteerd.
Voor 0000110 komt de automaat na de begintoestand q_0 achtereenvolgens in q_0, q_0, q_0, q_0, q_1, q_2 en q_2. De hele string is gelezen, maar q_2 is geen eindtoestand, dus 0000110 wordt niet geaccepteerd.

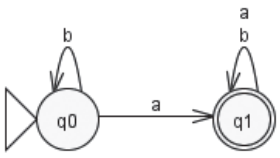
2.2 $\delta^*(q_0, 111) = q_1$, dat wil zeggen: als de automaat van figure 2.1 in de begintoestand begint, dan is hij na het lezen van 111 in toestand q_1.
$\delta^*(q_2, 100010) = q_0$, dus 100010 lezen vanuit q_2 brengt ons weer terug in de begintoestand.

2.3 a Een dfa voor de taal $L_a = \{x \in \{a, b\}^* : x$ bevat precies één $a\}$ is:



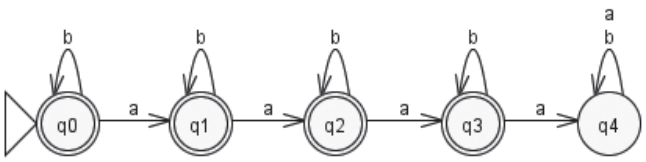
We gebruiken (de indices van) de namen van de toestanden om te onthouden hoeveel a's er al gelezen zijn: in q_0 zijn dat er geen, in q_1 is dat er één, en in q_2 zijn dat er twee of meer.
Ter controle proberen we een aantal goed gekozen strings uit; sommige moeten geaccepteerd worden (bijvoorbeeld a, ab, ba, $bbbbabbbb$) en andere juist niet (bijvoorbeeld λ, b, $bbbb$, $abab$, aa). In al deze gevallen geeft de gegeven dfa het goede antwoord. Ga dit na!

b Een dfa voor de taal $L_b = \{x \in \{a, b\}^* : x$ bevat minstens één $a\}$ is:

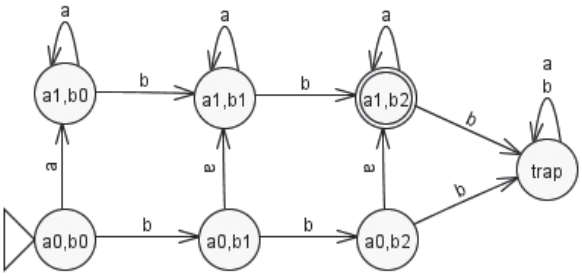


Ter controle: λ, b, $bbbbb$ moeten afgekeurd worden, en a, aa, $bababababab$ moeten geaccepteerd worden. Dat gebeurt ook inderdaad.

c Een dfa voor de taal $L_c = \{x \in \{a, b\}^* : x$ bevat hoogstens drie a's$\}$ is:

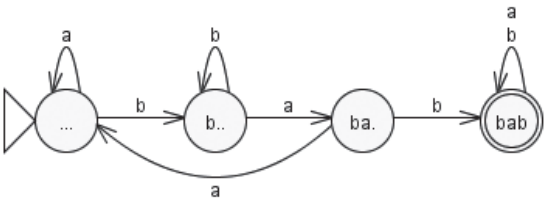


d Een dfa voor de taal $L_d = \{x \in \{a, b\}^* : x$ bevat minstens één a en precies twee b's$\}$ is:



We hebben de toestanden in deze dfa namen gegeven die aangeven welke informatie in die toestand bekend is. In dit geval gebruiken we bijvoorbeeld a0,b1 om aan te geven dat er nog geen a's gelezen zijn (van de minstens één die we uiteindelijk moeten hebben), maar wel al één b (van de precies twee die we moeten hebben). Door het kiezen van de juiste eindtoestand zorgen we er dan voor dat precies L_d geaccepteerd wordt.

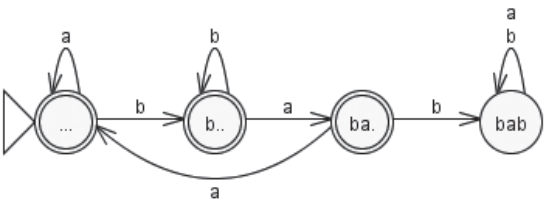
2.4 a Het idee achter onze oplossing is om in de toestanden bij te houden welk deel van *bab* al herkend is. De toestand ba. staat bijvoorbeeld voor de situatie dat de automaat net de (sub)string *ba* heeft gelezen. Als hij in die situatie een *a* tegenkomt, dan is het laatstgelezen stuk van de invoer blijkbaar *baa* en kan hij weer opnieuw beginnen met zoeken naar *bab*: in de substring *baa* zit geen beginstuk (prefix) van *bab*. Hij gaat dus weer naar de begintoestand … en leest het volgende symbool van de invoer. Hij accepteert alleen strings waarvoor hij eindigt in de toestand bab; vanwege de lus met label *a,b* bij die toestand accepteert hij niet alleen strings die op *bab* eindigen, maar ook strings die *bab* als substring hebben. De dfa voor $L_1 = \{x \in \{a, b\}^* : x$ bevat de substring *bab*$\}$ die we zo krijgen is:



In opgave 2.12 zullen we een makkelijkere manier zien om een eindige automaat voor deze taal te maken.

b Voor $L_2 = \{x \in \{a, b\}^* : x$ bevat geen substring *bab*$\}$ gebruiken we precies dezelfde strategie, en we krijgen ook precies dezelfde onderliggende graaf. Het enige verschil is de eindtoestand: we moeten nu alleen strings accepteren die niet de volledige substring *bab* bevatten.

We krijgen dan de volgende dfa voor L_2:

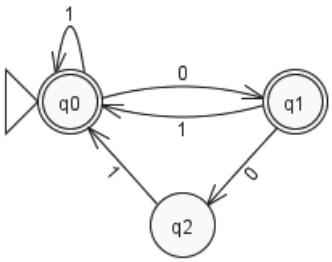


2.5 (Eigenlijk hebben we hetzelfde net al gedaan in opgave 2.4b, maar dan voor een andere automaat, en zonder uit te leggen waarom het 'verwisselen' van eind- en gewone toestanden het gewenste resultaat oplevert.)
De overgangsfunctie van de eindige automaat in figure 2.6 is *totaal* (want dat moet volgens de definitie van dfa), dus voor iedere toestand en ieder invoersymbool is er precies één pijl vanuit die toestand naar een (eventueel andere) toestand, met dat invoersymbool als label van de pijl. Dat betekent dat deze automaat *alle* strings over $\{a, b\}^*$ helemaal uitleest. Bovendien is de dfa *deterministisch* (omdat de overgangsfunctie een functie is). Dat wil zeggen dat er voor iedere invoerstring hoogstens één pad door de automaat is. Deze twee aspecten samen (totaal en deterministisch) zorgen ervoor dat er voor *iedere* invoerstring *precies één* pad door de automaat is. Als dat pad eindigt in een eindtoestand wordt de string geaccepteerd, als het pad eindigt in een niet-eindtoestand wordt de string niet geaccepteerd. En als we dus eindtoestanden en niet-eindtoestanden 'verwisselen', zoals gevraagd in de opgave, dan worden door de nieuwe automaat precies die strings geaccepteerd die door de oorspronkelijke niet werden geaccepteerd, en andersom. Inderdaad is de taal die door de nieuwe automaat wordt geaccepteerd dan het complement van de taal van de oude automaat.
Merk op dat dit alles niets te maken heeft met de taal in kwestie; alleen de structuur van de oorspronkelijke eindige automaat is van belang.

2.6 Laat $M = (Q, \Sigma, \delta, q_0, F)$ en $M_1 = (Q, \Sigma, \delta, q_0, Q - F)$ twee totale deterministische eindige automaten zijn. Dan geldt:
$L(M_1)$
$= \{w \in \Sigma^* : \delta^*(q_0, w) \in Q - F\}$ (definition 2.2 toegepast)
$= \{w \in \Sigma^* : \delta^*(q_0, w) \notin F\}$ (dat klopt alleen omdat beide automaten totaal en deterministisch zijn)
$= L(M)$

2.7 Laten we uitgaan van een eindige automaat $M = (Q, \Sigma, \delta, q_0, F)$ die wel deterministisch is, maar niet totaal (we laten dus de kwalificatie 'totaal' weg uit de beschrijving van δ in definition 2.1). We voegen nu een nieuwe toestand q_{trap} aan Q toe, die dienst gaat doen als trap state: we voegen ook de overgangen $\delta(q_{\text{trap}}, a) = q_{\text{trap}}$ toe voor alle $a \in \Sigma$. Verder voegen we voor iedere combinatie van een toestand $q \in Q - \{q_{\text{trap}}\}$ en een $a \in \Sigma$ waarvoor de oorspronkelijke automaat geen overgang heeft, de overgang $\delta(q, a) = q_{\text{trap}}$ toe. Kortom: we vullen de oorspronkelijke automaat gewoon aan met de ontbrekende overgangen, en laten die allemaal 'doodlopen' in een nieuwe trap state. Verder komt er voor ieder invoersymbool een overgang van de trap state naar zichzelf.

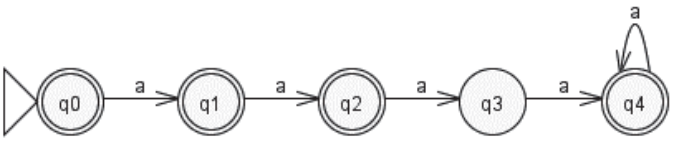
2.8 a Een oplossing is de volgende automaat, die als volgt is geconstrueerd:
– in toestand q_0 is nog niets van 00 gelezen, en zolang we ook niets van 00 lezen (dus alleen 1'en zien) kunnen we daar blijven en ook meteen alles accepteren
– in toestand q_1 is de eerste 0 van 00 gelezen, maar zolang we de tweede 0 niet zien hoeven we nergens op te letten, dus we kunnen weer accepteren. Als we nu een 1 lezen weten we dat het niet ging om 'de eerste 0 van 00', dus kunnen we terug naar toestand q_0 ('niets van 00 gelezen'). Als we echter na de 0 nog een 0 zien hebben we wel 00 gelezen, en moeten we gaan opletten dat er nu een 1 volgt. We gaan dus naar een nieuwe toestand q_2, waarin we niet mogen accepteren.
– vanuit q_2 moeten we een 1 lezen, en daarna kunnen we eventueel doorlezen en op zoek gaan naar een volgende 00; we gaan dus naar toestand q_0.



Merk op dat we (volgens onze afspraak) de nonfinal trap state achterwege hebben gelaten. De automaat is dus niet totaal maar wel deterministisch.

b Het controleren van de voorbeeldstrings gaat eenvoudig door Input| Multiple Run te kiezen. Aan de rechterkant kunt u dan onder Input de voorbeeldstrings invoeren (geef steeds een enter na iedere invoerstring). Als u alle gewenste strings hebt ingevoerd klikt u op Run Inputs. Naast iedere geaccepteerde string verschijnt dan Accept, naast iedere niet geaccepteerde string verschijnt Reject.
Met de andere drie opties uit het menu Input (Step with Closure…, Step by State… en Fast Run…) kunt u de strings één voor één testen, waarbij u dan ook nog kunt zien welk pad door de automaat gekozen wordt.

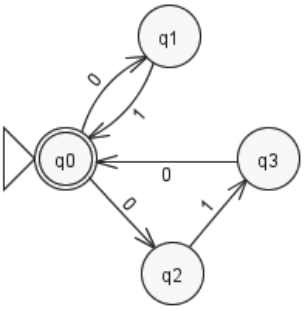
2.9 De voor de hand liggende oplossing is de volgende automaat:



2.10 Een string wordt geaccepteerd door een dfa als de string correspondeert met een wandeling van de begintoestand naar een eindtoestand. In een dfa is deze wandeling uniek: er is maar één manier om met die string van de begintoestand in een eindtoestand te komen.

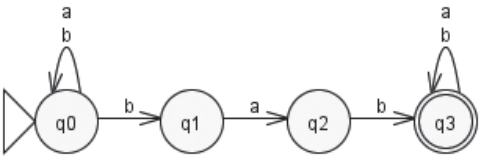
Voor een nfa geldt hetzelfde: een string wordt geaccepteerd door een nfa als de string correspondeert met een wandeling van de begintoestand naar een eindtoestand. Het verschil is dat in een nfa deze wandeling niet uniek hoeft te zijn: er kan meer dan één manier zijn om met die string van de begintoestand in een eindtoestand te komen.

2.11 Een nfa voor L is snel gemaakt:



Een dfa voor dezelfde taal is veel lastiger (direct) te vinden; we gaan ons daar dan ook niet in verdiepen. In de volgende paragraaf bespreken we een manier om bij een gegeven nfa een dfa te construeren die precies dezelfde taal accepteert.

2.12 Een nfa voor L is veel eenvoudiger te maken dan een dfa (zie daarvoor opgave 2.4 a):



De tactiek van de dfa van opgave 2.4 a is om het eerste voorkomen van de substring *bab* in de invoerstring te herkennen. De nfa daarentegen herkent eender welk voorkomen van de substring *bab*.

2.13 Het enige wat toestand q_1 doet is een extra manier bieden om van q_0 met een a in q_3 te komen; dat kan ook al rechtstreeks. We kunnen toestand q_1 dus veilig weglaten, inclusief de twee pijlen die eraan vast zitten.
Het driehoekje $q_0 - q_3 - q_5$ (en weer terug naar q_0) herkent de taal $\{aa, ab, ba, bb\}^+$. De vraag is nu of de toestanden q_2 en q_4 met de bijbehorende overgangen daar nog iets aan toevoegen. Het antwoord is nee: er zijn twee zinvolle paden die gebruik maken van q_2 en q_4, en beide kunnen worden 'nagedaan' binnen de driehoek $q_0 - q_3 - q_5$. Een alternatief voor het pad $q_3 - q_2 - q_4 - q_3$ met label ab is het pad $q_3 - q_5 - q_0 - q_3$. Een alternatief voor het pad $q_3 - q_2 - q_4 - q_5$ met label aba of abb is $q_3 - q_5 - q_0 - q_3 - q_5$. We kunnen toestanden q_2 en q_4 met de bijbehorende overgangen dus ook verwijderen zonder de geaccepteerde taal te veranderen. De automaat die dan overblijft, accepteert duidelijk precies $\{aa, ab, ba, bb\}^+$.
U kunt dit in JFLAP controleren door Jexercise2.2.jff op te slaan onder een andere naam, in dat tweede bestand toestanden q_1, q_2 en q_4 te verwijderen, en dan (zorg dat beide bestanden geopend zijn!) in een van beide vensters te kiezen voor Test | Compare Equivalence.

2.14 De automaat van figure 2.10, met overgangen (q_0, a, q_1), (q_1, λ, q_2) en (q_2, λ, q_0), is niet-deterministisch omdat er een λ-overgang in zit: als de automaat in toestand q_1 is, dan kan hij kiezen of hij daar blijft of dat hij naar toestand q_2 gaat, in beide gevallen zonder een symbool van de invoerstring te lezen. Er zijn weliswaar niet twee uitgaande pijlen met hetzelfde label vanuit toestand q_1, maar op basis van dezelfde actie op de invoerstring (namelijk het *niet* lezen van het volgende symbool), kan de automaat wel kiezen uit twee verschillende 'bestemmingen', namelijk q_1 en q_2. Omdat er ook een λ-overgang is vanuit q_2 naar q_0, kan hij er zelfs ook nog voor kiezen geen invoersymbool te lezen en naar q_0 te gaan.
Het is dus wel zo dat een automaat met (minstens) een λ-overgang niet-deterministisch is, maar een niet-deterministische automaat hoeft geen λ-overgangen te hebben.

2.15 We introduceren even een notatie voor paden door een automaat:
$p\ ab \rightarrow q\ b$ betekent dat de automaat in toestand p begint met het lezen van de invoerstring ab; na het lezen van het symbool a is de automaat in toestand q en moet dan nog b lezen.

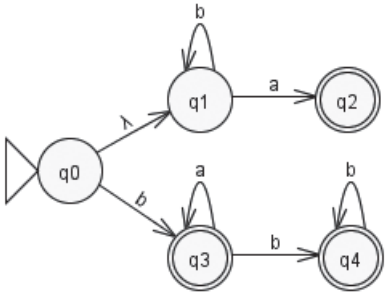
00 wordt niet geaccepteerd door de automaat van exercise 13, want alle mogelijke paden vanuit de begintoestand met label 00 zijn:
$q_0\ 00 \rightarrow q_1\ 0 \rightarrow q_0$ (de string is helemaal gelezen, maar q_0 is geen eindtoestand)
$q_0\ 00 \rightarrow q_1\ 0 \rightarrow q_2$ (ook niet geëindigd in eindtoestand)
$q_0\ 00 \rightarrow q_1\ 0 \rightarrow q_2\ 0 \rightarrow$?? (loopt dood)

01001 wordt wel geaccepteerd, want er is een pad van de begintoestand naar de eindtoestand met label 01001:
$q_0\ 01001 \rightarrow q_1\ 1001 \rightarrow q_1\ 001 \rightarrow q_0\ 01 \rightarrow q_1\ 1 \rightarrow q_1$, en dat is de eindtoestand.

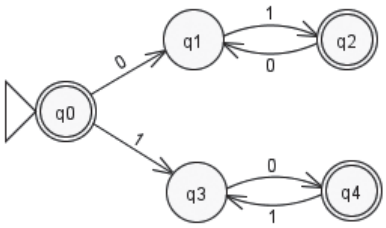
10010 wordt niet geaccepteerd:
$q_0\ 10010 \rightarrow q_1\ 0010 \rightarrow q_0\ 010 \rightarrow q_1\ 10 \rightarrow q_1\ 0 \rightarrow q_0q_0\ 10010 \rightarrow q_1\ 0010 \rightarrow q_0\ 010 \rightarrow q_1\ 10 \rightarrow q_2\ 10 \rightarrow q_1\ 0 \rightarrow q_0$
$q_0\ 10010 \rightarrow q_1\ 0010 \rightarrow q_0\ 010 \rightarrow q_1\ 10 \rightarrow q_2\ 10 \rightarrow q_1\ 0 \rightarrow q_2$
$q_0\ 10010 \rightarrow q_1\ 0010 \rightarrow q_0\ 010 \rightarrow q_1\ 10 \rightarrow q_2\ 10 \rightarrow q_1\ 0 \rightarrow q_2\ 0 \rightarrow$??
$q_0\ 10010 \rightarrow q_1\ 0010 \rightarrow q_2\ 010 \rightarrow$??
$q_0\ 10010 \rightarrow q_1\ 0010 \rightarrow q_2\ 0010 \rightarrow$??
Geen van deze mogelijkheden leidt naar een eindtoestand.

000 wordt wel geaccepteerd, maar 0000 weer niet.

2.16 a De taal $L_a = \{b\}^*\{a\} \cup \{b\}\{a\}^*\{b\}^*$ is ook een van die talen waarvoor veel makkelijker een nfa dan een dfa gemaakt kan worden, en waarbij λ-overgangen ook goed van pas komen. Een oplossing is de volgende:

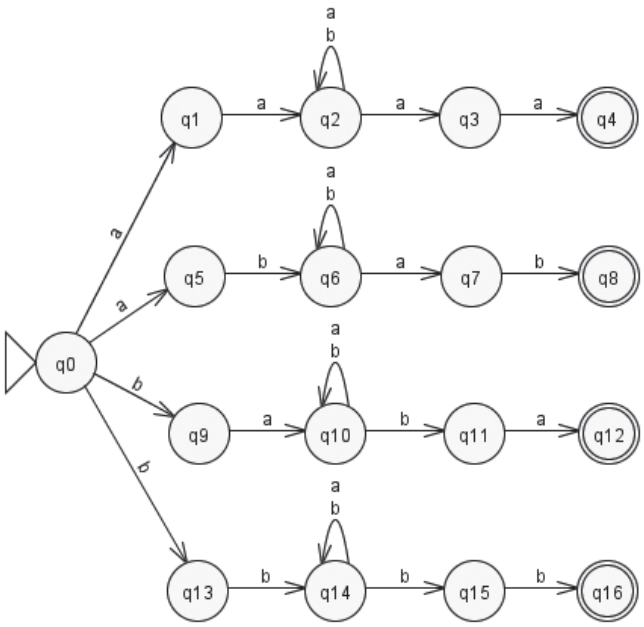


b Een nfa (die toevallig wel deterministisch, maar niet totaal is) voor L_b is:

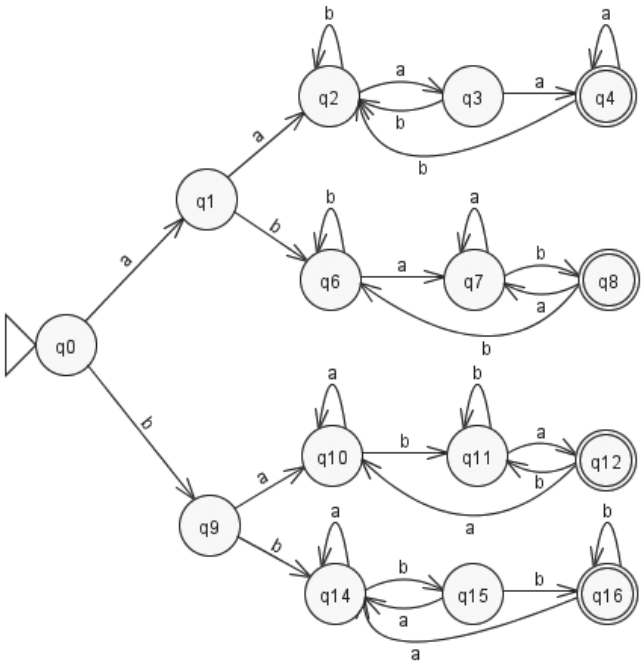


2.17 a Voor de eerste string geldt $aabaa \in L$, want die begint en eindigt met dezelfde string van lengte 2 (alles over $\{a, b\}^*$). De tweede string doet dat niet, dus $ababba \notin L$. De string bab is te kort; alle strings in L hebben immers ten minste lengte 4. Dus $bab \notin L$. De laatste string, $baba$, is wel een element van L, met $w = \lambda$ en $v = ba$.

b De meest voor de hand liggende eindige automaat voor L is niet-deterministisch en ziet er als volgt uit:

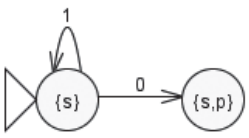


c Uitgaande van de nfa van onderdeel b kunnen we de toestanden q_1 en q_5 samennemen, evenals q_9 en q_{13}. Dan moeten we de dubbele a-overgangen vanuit q_2 en die vanuit q_6 nog oplossen, en de dubbele b-overgangen vanuit q_{10} en die vanuit q_{14}. Bij q_2 komt dat neer op het verzinnen van een deterministische eindige automaat voor "eindigt op *aa*". Het resultaat wordt dan als volgt:

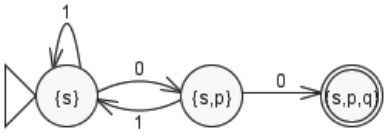


2.18 Zie de uitwerking op pagina 405 in Linz. Het idee is om een nieuwe toestand aan de automaat toe te voegen, met daarvandaan een λ-overgang naar iedere begintoestand van de oorspronkelijke automaat. Vervolgens worden alle begintoestanden weer gewone toestanden (ze blijven wel eindtoestand als ze dat al waren), en de nieuw toegevoegde toestand wordt de enige begintoestand.

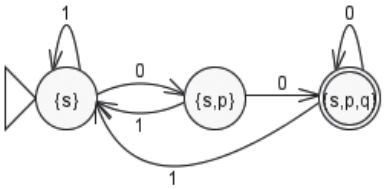
2.19 De enige toestand waarin een (zinnig) pad door de nfa kan beginnen is toestand s, dus de begintoestand van de dfa wordt $\{s\}$. Vanuit s kunnen we in de nfa met een 0 naar s èn naar p. In de dfa vatten we deze twee paden (in dit geval zijn het gewoon overgangen) samen in het pad (de overgang) $(\{s\}, 0, \{s, p\})$, naar een nieuwe toestand $\{s, p\}$. Er is nog een overgang vanuit toestand s in de nfa, met label 1. Deze overgang eindigt in toestand s. We hoeven dus niet nóg een nieuwe toestand aan de dfa toe te voegen: we maken alleen een overgang $(\{s\}, 1, \{s\})$. We zijn nu klaar met toestand $\{s\}$. Het plaatje is nu als volgt:



We gaan verder met de enige nog niet afgewerkte toestand van de dfa, dus met $\{s, p\}$. In de nfa kunnen we vanuit s met een 0 naar s en naar p (dat hadden we net ook al gezien, maar het levert nu een bijdrage aan een andere overgang in de dfa!), en vanuit p met een 0 naar q. Samengevat: vanuit $\{s, p\}$ kunnen we met een 0 naar $\{s, p, q\}$, en deze laatste toestand bestond nog niet in de dfa, dus die voegen we toe, met de bijbehorende overgang. In de nfa is q een eindtoestand, dus wordt $\{s, p, q\}$ ook een eindtoestand.
We kijken ook meteen waar we in de nfa vanuit $\{s, p\}$ met een 1 kunnen komen: vanuit s is dat s zelf weer, vanuit p nergens. De dfa krijgt dus ook een overgang $(\{s, p\}, 1, \{s\})$. Het plaatje wordt dan:

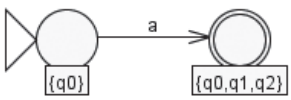


Weer is er maar één toestand bij gekomen: $\{s, p, q\}$. Met een 0 kunnen we in de nfa vanuit s in s en p komen, vanuit p in q en vanuit q nergens; de dfa krijgt dus een overgang $(\{s, p, q\}, 0, \{s, p, q\})$. Met een 1 kunnen we in de nfa vanuit dit drietal toestanden alleen maar in s komen; de dfa krijgt dus een overgang $(\{s, p, q\}, 1, \{s\})$. Er zijn geen nieuwe toestanden bij gekomen, dus het eindresultaat is:

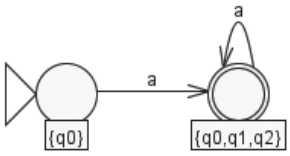


Merk op dat deze automaat totaal is, terwijl de oorspronkelijke nfa dat niet is. Als u de procedure nfa-to-dfa precies volgt is het resultaat altijd een totale deterministische automaat; als u de pragmatische benadering volgt die we in deze opgave beschreven hebben, kan het resultaat ook een niet-totale deterministische automaat zijn.

2.20 We beginnen met de dfa een begintoestand $\{q_0\}$ te geven. Hiervandaan loopt natuurlijk nog geen overgang met label a, dus die gaan we nu toevoegen. Maar waar moet die heen? Vanuit q_0 kunnen we in de nfa met een a naar q_1. Verder kunnen we vanuit q_1 via de λ-overgang 'doorlopen' naar q_2, en daarvandaan zelfs weer doorlopen naar q_0. Dan kunnen we niet meer verder zonder het *volgende* invoersymbool te lezen (en bovendien hebben we alle toestanden van de nfa al gehad), dus de dfa krijgt een overgang
$(\{q_0\}, a, \{q_0, q_1, q_2\})$. Omdat $q_1 \in \{q_0, q_1, q_2\}$ wordt de nieuwe toestand meteen eindtoestand. De voorlopige dfa is:



Volgens de procedure nfa-to-dfa moeten we nu $\delta^*(q_0, a) \cup \delta^*(q_1, a) \cup \delta^*(q_2, a)$ berekenen. We beginnen vooraan, met $\delta^*(q_0, a)$, en zien dat we die in de eerste stap al berekend hebben: $\delta^*(q_0, a) = \{q_0, q_1, q_2\}$. En omdat $\{q_0, q_1, q_2\}$ alle toestanden van de nfa bevat, kan deze verzameling niet uitgebreid worden door vereniging met $\delta^*(q_1, a)$ en $\delta^*(q_2, a)$. We zijn dus klaar, en het resultaat is:

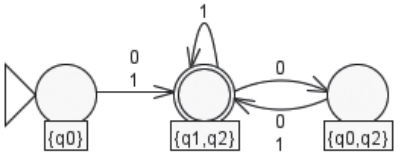


We hadden dit resultaat ook direct kunnen bereiken, dus zonder de procedure nfa-to-dfa uit te voeren, door in te zien dat de taal die door de nfa van figure 2.10 wordt geaccepteerd gelijk is aan $\{a^n : n > 0\}$. De voor de hand liggende dfa voor die taal is bovenstaande automaat.

2.21 Dit is zo'n opgave waarbij veel verwarring en vergissingen voorkomen kunnen worden door eerst een tabel met de waarden van δ^* voor de nfa te maken. Die tabel ziet er als volgt uit:

δ^*	0	1
q_0	$\{q_1, q_2\}$	$\{q_1, q_2\}$
q_1	$\{q_0, q_2\}$	$\{q_1, q_2\}$
q_2	$\varnothing$	$\{q_1, q_2\}$

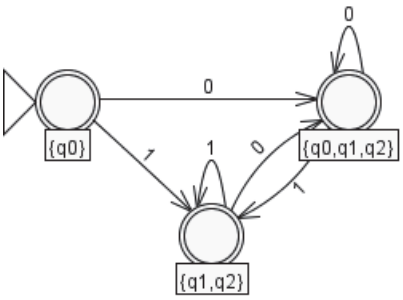
We hebben hier niets anders gedaan dan de mogelijkheden in de nfa opsommen, bijvoorbeeld: vanuit q_0 kunnen we met een 1 naar q_1 (via de 1-overgang) of naar q_2 (via de 1-overgang gevolgd door de λ-overgang). We hebben dus alvast wat voorwerk gedaan voor de procedure nfa-to-dfa, want daar hebben we deze informatie ook nodig. Merk op dat we nog steeds wel goed moeten opletten dat we de tabel vullen met de juiste waarden, vanwege de λ-overgangen!
Het 'echte' werk is nu eenvoudig: we beginnen de dfa met een begintoestand $\{q_0\}$, en geven die een 0-overgang naar $\{q_1, q_2\}$ en ook een 1-overgang naar $\{q_1, q_2\}$. Nu komt het moment waarop we voordeel hebben van de tabel: om te bepalen naar welke toestanden we overgangen vanuit $\{q_1, q_2\}$ moeten toevoegen, nemen we
voor de 0-overgang $\delta^*(q_1, 0) \cup \delta^*(q_2, 0) = \{q_0, q_2\} \cup \varnothing = \{q_0, q_2\}$, en
voor de 1-overgang $\delta^*(q_1, 1) \cup \delta^*(q_2, 1) = \{q_1, q_2\} \cup \{q_1, q_2\} = \{q_1, q_2\}$.
Zo gaan we verder tot we uiteindelijk de volgende dfa krijgen:



2.22 a We maken eerst weer de tabel voor δ*:

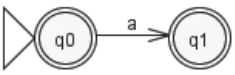
δ*	0	1
q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$
q_1	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$
q_2	$\{q_2\}$	$\{q_1\}$

Daarna is het construeren van de dfa eenvoudig:



b De nfa's zijn equivalent. Dit kunnen we in JFLAP controleren door beide bestanden te openen en dan vanuit één van beide de optie Test|Compare Equivalence te gebruiken.

2.23 Voeg een nieuwe eindtoestand p_f toe, en voeg voor iedere $q \in F$ een λ-overgang toe van q naar deze p_f. Maak dan alle 'oude' eindtoestanden gewone toestanden, zodat p_f de enige eindtoestand is.
Het is dan eenvoudig om aan te tonen dat als eerst $\delta^*(q_0, w) \cap F \neq \varnothing$ gold, nu $p_f \in \delta^*(q_0, w)$ geldt, en andersom.
Deze constructie werkt niet voor dfa's, omdat daarin geen λ-overgangen mogen voorkomen. Er zou natuurlijk een andere constructie te verzinnen kunnen zijn die wel werkt, maar dat blijkt niet zo te zijn. Het is namelijk zo dat een dfa voor de taal $\{\lambda, a\}$ alleen met twee eindtoestanden gemaakt kan worden:



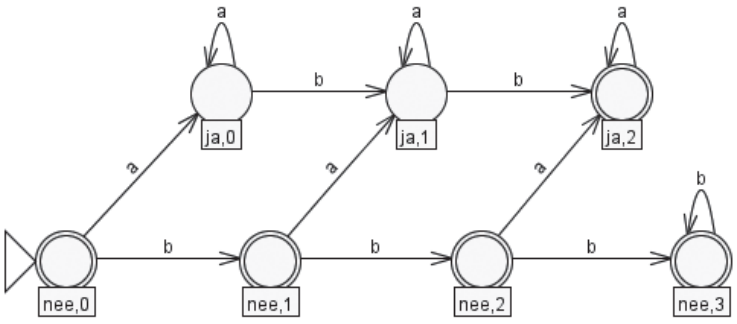
Het is dus inderdaad zo dat zo'n constructie voor dfa's niet bestaat. Dat ligt overigens geheel aan het feit dat we rekening moeten houden met talen die λ bevatten, maar daar gaan we verder niet op in.

2.24 Nee, want de procedure nfa-to-dfa (en ook onze praktische versie daarvan) zegt dat we overgangen moeten blijven toevoegen zolang er toestanden zijn die nog niet voor alle $a \in \Sigma$ een overgang hebben. Er komen dus alleen overgangen voor invoersymbolen, en λ is geen invoersymbool.

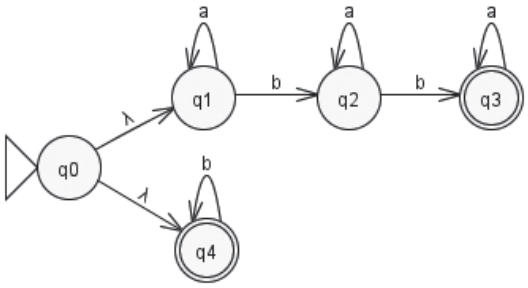
2.25 Als u op de knop State Expander hebt geklikt (voor een 'demonstratie' per toestand), kunt u daarna steeds een toestand in de dfa-in-aanbouw aanklikken die u afgehandeld wilt zien. We raden u aan om steeds voor uzelf na te gaan wat er gebeurd is en waarom. Denk eraan dat in de dfa de labels (dus niet de namen) van de toestanden aangeven wat de corresponderende toestanden in de nfa zijn: label 0,1 staat voor $\{q_0, q_1\}$. Met de knop Complete kunt u de dfa in één keer af laten maken. Met de knop Done? Kunt u een aanwijzing krijgen over wat er nog moet gebeuren.
Als u niet de knop State Expander kiest maar de knop Expand Group on Terminal kunt u daarna per toestand zelf aangeven voor welk invoersymbool er een overgang naar welke toestand moet komen. Als u in een toestand klikt, neemt JFLAP aan dat u een lus bij die toestand wilt maken. Als u een overgang naar een nieuwe toestand wilt maken moet u daarom in een toestand klikken, de muis ingedrukt houden en de muiswijzer een eindje uit de toestand bewegen. Als u de muis dan loslaat, vraagt JFLAP u eerst welk label de overgang moet hebben, en vervolgens wat het label van de nieuwe toestand moet zijn. U kunt dan bijvoorbeeld 1,2 intypen als u $\{q_1, q_2\}$ bedoelt. De knoppen Complete en Done? werken zoals eerder beschreven.

2 Uitwerking van de zelftoets

1 a We moeten eerst bepalen welke strings de taal bevat. De omschrijving zegt "als w een a bevat, …". Dus K bevat ook strings zonder a's, met andere woorden, K bevat alle strings die uit 0 of meer b's bestaan (want a en b zijn hier de enige invoersymbolen). Verder zegt de omschrijving dat alle strings waar één of meer a's in zitten precies twee b's moeten bevatten. Dit deel van de taal is dus ook te omschrijven als $\{a\}^*\{b\}\{a\}^*\{b\}\{a\}^*$. Kortom, een andere definitie van K is $\{b\}^* \cup \{a\}^*\{b\}\{a\}^*\{b\}\{a\}^*$. Deze definitie suggereert een eenvoudige nfa, maar helaas wordt in onderdeel a om een dfa gevraagd. Een mogelijke oplossing is om toch eerst de nfa (zie onderdeel b) te maken, en die dan met de procedure nfa-to-dfa om te zetten naar een dfa. Wij kiezen nu echter voor een directe aanpak.
We gebruiken het inmiddels bekende recept: we houden in de namen van de toestanden bij of we al a's hebben gelezen (ja/nee), en hoeveel b's we hebben gelezen (0, 1, 2, 3). Vervolgens zetten we de toestanden in een soort matrixvorm, en bepalen voor iedere toestand waar we met een a heen moeten kunnen en waar met een b. Dat levert de volgende automaat:



b Een nfa voor K is veel sneller gemaakt: aangezien de taal bestaat uit twee duidelijk te onderscheiden delen, geven we de automaat ook twee onafhankelijke delen.

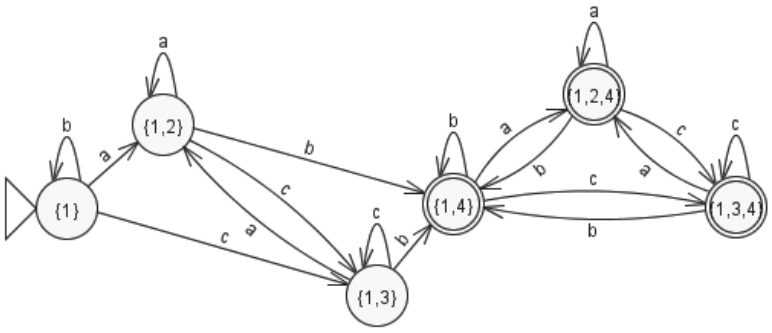


2 a M is niet totaal, want bijvoorbeeld $\delta(3, a)$ is niet gedefinieerd (niet voor iedere combinatie toestand-invoersymbool is er een overgang). M is ook niet deterministisch, want bijvoorbeeld $\delta(1, c) = \{1, 3\}$ (er zijn combinaties toestand-invoersymbool waarvoor er een keuze is).

b We geven weer een tabel voor δ^*:

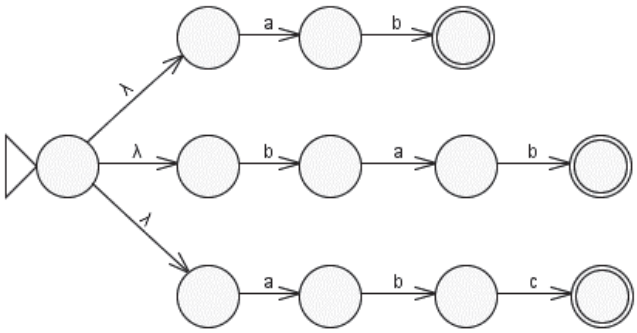
δ^*	a	b	c
1	{1, 2}	{1}	{1, 3}
2	∅	{4}	∅
3	∅	{4}	∅
4	{4}	{4}	{4}

c Als we de procedure systematisch aanpakken (we lopen de invoersymbolen op alfabet af, en de nieuwe toestanden in de volgorde waarin we ze tegenkomen), vinden we achtereenvolgens de toestanden {1}, {1, 2}, {1, 3}, {1, 4}, {1, 2, 4} en {1, 3, 4}. De toestanden die 4 bevatten worden eindtoestanden. De dfa ziet er als volgt uit:



Als u goed naar het cluster van eindtoestanden in de dfa kijkt, ziet u dat de constructie om een nfa deterministisch te maken niet altijd de meest efficiënte automaten oplevert: we kunnen toestand {1, 4} een extra lus met label a en een met label c geven, en de toestanden {1, 2, 4} en {1, 3, 4} weglaten.

3 De eenvoudigste manier om dit te laten zien is aan de hand van een voorbeeld. Een nfa voor de eindige taal $L = \{ ab, bab, abc \}$ is bijvoorbeeld:



In het algemeen komt het erop neer dat je gewoon voor iedere string uit de taal zijn eigen pad van de begintoestand naar een eindtoestand maakt; in een nfa gaat dat heel eenvoudig (we gebruiken hier λ-overgangen, maar het kan ook zonder). Omdat er maar eindig veel strings in de taal zitten, wordt de automaat ook eindig.

4 De taal $L = \{a^n b^n : n \geq 1\}$ kan niet worden herkend door een eindige automaat (van welke soort dan ook – deterministisch of niet-deterministisch, totaal of niet, met λ-overgangen of niet), omdat de informatie die tijdens het doorlopen van een invoerstring moet worden bijgehouden niet eindig is. Voor iedere string moet eerst het aantal *a*'s geteld worden, zodat daarna kan worden gecontroleerd dat er precies evenveel *b*'s volgen. Omdat het aantal *a*'s willekeurig groot kan worden zijn daarvoor oneindig veel toestanden nodig.
NB Het is wel mogelijk een eindige automaat te maken voor de taal $\{a\}^*\{b\}^*$, omdat we dan niet hoeven te tellen: we hoeven alleen maar te zorgen dat we eerst alle *a*'s krijgen en dan alle *b*'s. Het is ook mogelijk een eindige automaat te maken voor de taal $\{a^n b^n : 1 \leq n \leq m\}$, voor een of andere vaste $m \geq 1$, omdat die laatste taal eindig is.
In leereenheid 4 bewijzen we formeel dat *L* inderdaad niet regulier is.