Hoofdstuk 4

Talen en Automaten

4.1 Formele Talen

Dit hoofdstuk is gebaseerd op een hoofdstuk uit het dictaat Formele Talen en Automaten 1, *G. Rozenberg, H.J. Hoogeboom, en J. Engelfriet (voorjaar 2000)*

4.1.1 Woorden

▷ SCHAUM §12.2: Alphabet, Words, Free Semigroup

Een *alfabet* Σ is een eindige, niet-lege, verzameling. Elementen van Σ worden wel *letters* of *symbolen* genoemd. Een eindig (geordend) rijtje van letters uit Σ heet een *woord* of *string over* Σ. Als $x = (a_1, a_2, \ldots, a_n)$, met $n \geq 0$ en $a_i \in \Sigma$ voor $1 \leq i \leq n$, een woord is, dan is n de *lengte* van x, we noteren $|x|$. Het woord van lengte 0, dus het rijtje $()$, heet het *lege woord*. We zullen voortaan woorden noteren zonder haakjes en komma's, dus $x = a_1 a_2 \cdots a_n$. Het lege woord wordt nu genoteerd als λ.

Σ^* geeft de verzameling woorden over Σ aan, Σ^+ is de verzameling niet-lege woorden; $\Sigma^* = \Sigma^+ \cup \{\lambda\}$.

We definiëren een binaire bewerking $\cdot$ op woorden, *concatenatie* geheten. Deze bewerking voegt aan elk tweetal woorden het woord toe dat ontstaat door de twee achter elkaar te schrijven. Dus als $x = a_1 a_2 \cdots a_n$ en $y = b_1 b_2 \cdots b_m$ (met $a_i \in \Sigma$, en $b_j \in \Sigma$) dan $x \cdot y = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m$, dit is een woord van lengte $n+m$.

Concatenatie is *associatief*, dwz. $x \cdot (y \cdot z) = (x \cdot y) \cdot z$, en het lege woord is het *eenheids element* voor deze bewerking, dwz. $x \cdot \lambda = \lambda \cdot x = x$. Concatenatie is niet commutatief, dwz. $x \cdot y$ is niet altijd gelijk aan $y \cdot x$.

We laten vaak de notatie $\cdot$ weg, en schrijven dan xy ipv. $x \cdot y$. Verder gebruiken we de notatie $x^n = x \cdot \ldots \cdot x$ (n maal). Het is nu logisch om $x^0 = \lambda$ te definiëren. We hebben dan $x^{m+n} = x^m \cdot x^n$ voor alle $n, m \in \mathbb{N}$.

4.1 Voorbeeld. Voor $\Sigma = \{a,b,c\}$, zijn $x = abbcab$ en $y = cbbc$ woorden over Σ. y is bovendien een woord over $\{b,c\}$. Voor de lengtes geldt $|x| = 6$, $|y| = 4$. De concatenatie $x \cdot y = abbcab \cdot cbbc = abbcabcbbc$. $x^2 = abbcababbcab$, $x^1 = x$.

4.2 Opmerking. De gegeven definitie van de macht x^n is nogal informeel. We kunnen ook een nette inductieve definitie geven, en de gegeven eigenschap formeel (met inductie) bewijzen.

Voor een woord x definiëren we $x^0 = \lambda$, en voor $n \geq 0$, $x^{n+1} = x^n \cdot x$. Volgens deze formulering is bijvoorbeeld $(ab)^1 = \lambda \cdot ab = ab$, en is $(ab)^5$ gelijk aan $(((((ab) \cdot ab) \cdot ab) \cdot ab) \cdot ab)$. Kijken we nu naar $(ab)^2 \cdot (ab)^3$, dit is $((ab) \cdot ab) \cdot (((ab) \cdot ab) \cdot ab)$, dan staan de haakjes anders; maar de uitkomst is natuurlijk hetzelfde. Dit volgt, formeel, uit Lemma 2.8. □

x is een *deelwoord* van y als $y = vxw$ voor zekere woorden v en w. Als bovendien $v \neq \lambda$ of $w \neq \lambda$, dan heet x een *echt deelwoord* van y. We onderscheiden ook deelwoorden op speciale plaatsen. x is een *prefix* van y als $y = xw$ voor een woord w, en x is een *suffix* van y als $y = vx$ voor een woord v. Een prefix of suffix heet *echt* als het niet aan het hele woord gelijk is.

4.3 Voorbeeld. $y = aaabaab$ heeft $x = aa$ als (echt) deelwoord. We zeggen dat x verschillende *voorkomens* heeft in y (in dit geval 3). Deze voorkomens kunnen elkaar overlappen. $aaba$ is een prefix van $z = aababaab$. aab en λ (!) zijn zowel prefix als suffix van z. Natuurlijk is elk prefix en suffix ook een deelwoord.

4.4 Opmerking. De prefix-relatie op Σ^* heeft een aantal bijzondere eigenschappen. Laten we schrijven $x \preceq y$ als x een prefix van y is. Voor alle woorden x, y en z geldt:

reflexief: $x \preceq x$.
anti-symmetrisch: als $x \preceq y$ en $y \preceq x$, dan $x = y$.
transitief: als $x \preceq y$ en $y \preceq z$, dan is $x \preceq z$.

Een relatie met deze eigenschappen heet een *partiële ordening*, ▷ SCHAUM Ch. 14: Ordered Sets and Lattices. □

Voor een woord $x = a_1 a_2 \cdots a_n$ heet het woord $a_n \cdots a_2 a_1$ het *spiegelbeeld* van x en wordt genoteerd als $\mathrm{mir}(x)$. De afbeelding mir kan inductief als volgt beschreven worden: $\mathrm{mir}(\lambda) = \lambda$ en, voor $x \in \Sigma^*$ en $a \in \Sigma$, $\mathrm{mir}(xa) = a \cdot \mathrm{mir}(x)$. In het algemeen geldt dan dat $\mathrm{mir}(x \cdot y) = \mathrm{mir}(y) \cdot \mathrm{mir}(x)$ voor $x, y \in \Sigma^*$. Hoewel de eigenschap voor iedereen intuitief duidelijk is zullen we als opgave een formeel inductief bewijs vragen. Als alternatief wordt de notatie x^R voor $\mathrm{mir}(x)$ ook nog al eens gebruikt.

4.1.2 Talen en hun bewerkingen

▷ SCHAUM §12.3: Languages

▷ SCHAUM §12.4: Regular Expressions, Regular Languages

We gaan nog steeds uit van een alfabet Σ. Een deelverzameling K van Σ^* heet een *taal* (over Σ); dus is $\mathscr{P}(\Sigma^*)$ de verzameling van talen over Σ. Een verzameling van talen wordt meestal een *familie* of *klasse* genoemd. Denk er aan dat Σ^* een aftelbare verzameling is, terwijl $\mathscr{P}(\Sigma^*)$ dat niet is, zie Opgave **??**.

We bespreken een aantal bewerkingen op talen.

Verzamelings operaties. Elke taal is een verzameling, we kunnen dus spreken van de *vereniging* $K \cup L$, de *doorsnede* $K \cap L$ en het *verschil* $K - L$ van twee talen K en L. Een bijzonder geval van verschil is natuurlijk het *complement* (tov. het alfabet Σ) $\Sigma^* - K$ van de taal $K \subseteq \Sigma^*$.

Wanneer we ten opzichte van een vast alfabet werken, gelden de *regels van de Morgan*: $\Sigma^* - (K \cup L) = (\Sigma^* - K) \cap (\Sigma^* - L)$ en $\Sigma^* - (K \cap L) = (\Sigma^* - K) \cup (\Sigma^* - L)$, voor $K, L \subseteq \Sigma^*$. Ook geldt $\Sigma^* - (\Sigma^* - K) = K$, evenals de overige bekende rekenregels voor verzamelingen.

Concatenatie. Wanneer we van twee gegeven talen K en L op alle mogelijke manieren woorden van L achter woorden van K concateneren krijgen we een taal, de *concatenatie* $K \cdot L$ van K en L; formeel $K \cdot L = \{x \cdot y \mid x \in K, y \in L\}$. Net als bij concatenatie van woorden laten we de $\cdot$ vaak weg (niet als dit ten koste van de duidelijkheid gaat). Ook hier is deze bewerking associatief, maar niet commutatief.

Let op: $K \cup \emptyset = K$, en $K \cdot \emptyset = \emptyset \cdot K = \emptyset$, terwijl $K \cup \{\lambda\} = K$ desdals $\lambda \in K$, en $K \cdot \{\lambda\} = \{\lambda\} \cdot K = K$. Dus $\{\lambda\}$ is de eenheid bij concatenatie van talen.

Concatenatie distribueert over vereniging (maar niet over doorsnede!) Samengevat:

4.5 Lemma. *Laat K, L, M talen zijn over een alfabet Σ. Dan*
(i) $K \cdot \emptyset = \emptyset \cdot K = \emptyset$
(ii) $K \cdot \{\lambda\} = \{\lambda\} \cdot K = K$
(iii) $(K \cdot L) \cdot M = K \cdot (L \cdot M)$
(iv) $K \cdot (L \cup M) = K \cdot L \cup K \cdot M$,
(v) $(K \cup L) \cdot M = K \cdot M \cup L \cdot M$.

> Bewijs. In bewijzen over relaties tussen talen wordt gebruik gemaakt van het feit dat sommige bewerkingen 'monotoon' zijn: als je er meer instopt, komt er meer uit. Bijvoorbeeld $K \subseteq L$ impliceert $M \cup K \subseteq M \cup L$, en impliceert $KM \subseteq LM$ en $MK \subseteq ML$. Dit volgt rechtstreeks uit de definities.

(iii) Dit volgt tamelijk direct uit de associativiteit van concatenatie van strings. Voor willekeurige $z \in \Sigma^*$ geldt: $z \in (K \cdot L) \cdot M \iff z = xw$ met $x \in K \cdot L$ en $w \in M \iff z = xw$ met $x = uv$, met $u \in K$ en $v \in L$, en $w \in M \iff z = (uv)w = u(vw)$ met $u \in K$ en $v \in L$ en $w \in M \iff z = uy$ met $u \in K$ en $y = vw$, met $v \in L$, en $w \in M \iff z = uy$ met $u \in K$ en $y \in L \cdot M \iff z \in K \cdot (L \cdot M)$.

(iv) Eerst $K \cdot (L \cup M) \subseteq K \cdot L \cup K \cdot M$:
Neem $z \in K \cdot (L \cup M)$, dan $z = xy$ met $x \in K$ en $y \in L \cup M$. Als $y \in L$, dan $z = xy \in K \cdot L$, anders $z = xy \in K \cdot M$. Dus in beide gevallen $z \in K \cdot L \cup K \cdot M$.
Nu $K \cdot (L \cup M) \supseteq K \cdot L \cup K \cdot M$:
Omdat $L \subseteq L \cup M$, geldt $K \cdot L \subseteq K \cdot (L \cup M)$, evenzo $K \cdot M \subseteq K \cdot (L \cup M)$ waaruit het gevraagde volgt.

(v) Geheel symmetrisch aan het vorige geval. □

Machten, ster. Net als woorden kunnen ook talen met zichzelf (eventueel herhaald) geconcateneerd worden. $K^n = \{x_1 \cdot \ldots \cdot x_n \mid x_1, \ldots, x_n \in K\}$ heet de n-de *macht* van K. We kunnen dit ook inductief definiëren:

$K^0 = \{\lambda\}$, $K^{n+1} = K^n \cdot K$ voor $n \in \mathbb{N}$. De eigenschap $K^{m+n} = K^m \cdot K^n$ wordt daaruit met inductie verkregen, als in Lemma 2.8. Uit de (iteratieve) definitie volgt dat de eerste macht van een taal gelijk is aan de taal zelf, zoals we ook verwachten: $K^1 = K^0 \cdot K = \{\lambda\} \cdot K = K$.

Wanneer we verenigen over alle machten krijgen we de (*Kleene*) *ster* K^* van de taal K; dit is de taal die bestaat uit alle woorden ontstaan door nul, één, of meerdere woorden van K achter elkaar te zetten: $K^* = \bigcup_{n \in \mathbb{N}} K^n$.

We gebruiken ook wel $K^+ = \bigcup_{n \in \mathbb{N}^+} K^n$, de (*Kleene*) *plus* van K.

Dus $K^* = \{x_1 \cdot \ldots \cdot x_n \mid n \geq 0, x_1, \ldots, x_n \in K\}$, en $K^+ = \{x_1 \cdot \ldots \cdot x_n \mid n \geq 1, x_1, \ldots, x_n \in K\}$. Hier neemt binnen de definitie n alle waarden aan, terwijl in de definitie van K^n hierboven één vaste waarde geldt. Merk op dat $K^* = K^+ \cup \{\lambda\}$; voor $K = \emptyset$ geldt dat $\emptyset^+ = \emptyset$ en $\emptyset^* = \{\lambda\}$. Dat laatste is misschien een beetje vreemd.

4.6 Voorbeeld. $\{a\}^*$ is de verzameling van alle woorden bestaande uit alleen a's. We schrijven kortweg a^*. Voor talen die bestaan uit één één-letter woord laten we vaker de accolade's weg. Bijvoorbeeld:

$a^*b = \{a\}^* \cdot \{b\} = \{\lambda, a, aa, aaa, \ldots\} \cdot \{b\} = \{b, ab, aab, aaab, \ldots\}$, en

$(a^*b)^* = (\{a\}^* \cdot \{b\})^* = \{b, ab, aab, aaab, \ldots\}^* = \{\lambda, b, ab, bb, aab, abb, bab, bbb, \ldots\}$.

$\{a,b\}^*b$ is de verzameling van alle woorden over $\{a,b\}$ die eindigen op een b. Met een scherpe blik valt in te zien dat $(a^*b)^* = \{a,b\}^*b \cup \{\lambda\}$.

Zij $K = a^*b\{a,b\}^*$. Deze taal bestaat uit alle woorden over $\{a,b\}$ die minstens één symbool b bevatten. Voor het complement van K (tov. $\{a,b\}$) geldt $L =$

$\{a,b\}^* - K = \{x \in \{a,b\}^* \mid x \text{ bevat geen } b\} = a^*$. Hoewel in deze taal alleen letters a voorkomen, moeten we de taal tov. het alfabet $\{a,b\}$ beschouwen bij het nemen van complement om de oorspronkelijke taal weer terug te vinden.

$K = \{a,b\} \cdot \{a,b,0,1\}^*$ is de taal van alle Pascal identifiers (waarbij we ons voor de eenvoud tot $\{a,b,0,1\}$ beperkt hebben).

4.7 Lemma. *Laat K, L talen zijn over een alfabet Σ. Dan*
(i) $K^*K = KK^* = K^+$.
(ii) $(K^*)^n = K^*$, *voor* $n \geq 1$.
(iii) $(K^*)^* = K^*$.
(iv) $(K^*L^*)^* = (K \cup L)^* = (K^* \cup L^*)^*$.

Bewijs. Ook hier gelden algemene eigenschappen van monotoniciteit: $K \subseteq L$ impliceert $K^n \subseteq L^n$ en $K^* \subseteq L^*$.

(i) Omdat $K^* = \bigcup_{n\geq 0} K^n$, geldt $K^*K = \left(\bigcup_{n\geq 0} K^n\right)K = \bigcup_{n\geq 0}(K^nK) = \bigcup_{n\geq 0}(K^{n+1}) = \bigcup_{n\geq 1}(K^n) = K^+$.

Hoewel geheel correct, geven we hier liever een ander bewijs. Bovenstaande redenering gebruikt dat concatenatie distributief is over *oneindige* vereniging, iets wat vrijwel gelijk bewezen wordt als het geval met vereniging van twee verzamelingen, Lemma 4.5.

(ii) Voor $n = 1$ staat er gelijkheid omdat de eerste macht van een taal gelijk is aan de taal zelf.

Het geval $n = 2$ behandelen we eerst apart. Neem een willekeurige $x \in \Sigma^*$. Er geldt: $x \in K^*K^* \iff x = yz$ voor zekere $x \in K^*$ en $y \in K^* \iff x = yz$ voor zekere $x \in K^i$ en $y \in K^j$ en $i, j \in \mathbb{N} \iff x \in K^i \cdot K^j$ voor zekere $i, j \in \mathbb{N} \iff x \in K^{i+j}$ voor zekere $i, j \in \mathbb{N} \iff x \in K^m$ voor zekere $m \in \mathbb{N} \iff x \in K^*$.

Nu bewijzen we de bewering met inductie naar n. De basis hebben we al gehad.

Voor $n \geq 1$ geldt $(K^*)^{n+1} = (K^*)^n \cdot K^* = K^* \cdot K^* = K^*$. Omdat we hier de gelijkheid $K^* \cdot K^* = K^*$, dwz. de bewering voor $n = 2$ steeds expliciet gebruiken is deze apart bewezen.

(iii) Uit (ii) hierboven. □

In het voorbeeld hierboven staat dat 'met een scherpe blik valt in te zien'. De gelijkheid blijkt een gevolg van een algemene regel: $(K^*L)^* = (K \cup L)^*L \cap \{\lambda\}$.

Een familie van talen $\mathscr{F}$ heet *gesloten* onder een bewerking op talen, als die bewerking toegepast op talen uit de familie $\mathscr{F}$ weer altijd een taal uit $\mathscr{F}$ oplevert.

De talen die gemaakt kunnen worden met de bewerkingen vereniging, concatenatie en ster, heten de *reguliere* talen. Formeel is het de kleinste familie van talen

die de eindige talen bevat, en gesloten is onder deze bewerkingen. Anders gezegd, met een inductieve definitie:

4.8 Definitie. De familie van *reguliere talen* REG over een alfabet Σ is gedefinieerd door

basis. $K \in \mathsf{REG}$ voor elke eindige taal K over Σ.

inductie. Als K en L talen in REG zijn, dan ook $K \cup L$, $K \cdot L$ en K^* in REG. □

Een *reguliere expressie* is een string met een speciale syntax en een bijbehorende semantiek. De string bevat symbolen die de reuliere operaties voorstellen. De semantiek van een expressie is de bijbehorende reguliere taal. Zie SCHAUM.

Spiegelbeeld. Voor een taal L definiëren we het *spiegelbeeld* van L door $\mathrm{mir}(L) = \{\, \mathrm{mir}(x) \mid x \in L \,\}$. Merk op dat $\mathrm{mir}(K \cdot L) = \mathrm{mir}(L) \cdot \mathrm{mir}(K)$, analoog aan de gelijkheid $\mathrm{mir}(x \cdot y) = \mathrm{mir}(y) \cdot \mathrm{mir}(x)$ voor woorden x en y.

4.9 Stelling.
(i) $\mathrm{mir}(K \cup L) = \mathrm{mir}(K) \cup \mathrm{mir}(L)$,
(ii) $\mathrm{mir}(K \cdot L) = \mathrm{mir}(L) \cdot \mathrm{mir}(K)$,
(iii) $\mathrm{mir}(K^*) = (\mathrm{mir}(K))^*$. □

De reguliere talen (zie Definitie 4.8) zijn *per definitie* gesloten onder vereniging, concatenatie en ster. Ze zijn ook gesloten onder mir.

4.10 Stelling. *De reguliere talen zijn gesloten onder de bewerking* mir.

> Bewijs. Inductie naar de definitie van de reguliere talen.
> *basis.* De bewerking toegepast op een eindige taal levert weer een eindige taal; die is regulier per definitie.
>
> *inductie.* Neem aan dat voor de reguliere talen K en L de spiegelbeelden $\mathrm{mir}(K)$ en $\mathrm{mir}(L)$ ook regulier zijn (inductie-aanname). Te bewijzen is dat ook $\mathrm{mir}(K \cup L)$, $\mathrm{mir}(K \cdot L)$ en $\mathrm{mir}(K^*)$ regulier zijn. Dat volgt uit de inductie-aanname en de voorgaande stelling. Immers, als $\mathrm{mir}(K)$ en $\mathrm{mir}(L)$ regulier zijn, dan zijn ook $\mathrm{mir}(K \cup L) = \mathrm{mir}(K) \cup \mathrm{mir}(L)$, $\mathrm{mir}(K \cdot L) = \mathrm{mir}(L) \cdot \mathrm{mir}(K)$, en $\mathrm{mir}(K^*) = (\mathrm{mir}(K))^*$ regulier, omdat zij met toegestane bewerkingen uit reguliere talen geconstrueerd zijn. □

Het vervolg hieronder, over (inverse) homomorfismen, valt buiten de tentamenstof.

Homomorfismen. Laat Σ en Δ twee alfabetten zijn. Een *homomorfisme* is een afbeelding $h : \Sigma \to \Delta^*$ die in woorden over Σ elke letter op gelijke wijze vervangt door een woord uit Δ^*. We zullen dit stapsgewijs definiëren.

$h : \Sigma \to \Delta^*$ wordt uitgebreid tot woorden ($h : \Sigma^* \to \Delta^*$) volgens

$h(\lambda) = \lambda$, $h(xa) = h(x) \cdot h(a)$ voor $x \in \Sigma^*$, $a \in \Sigma$.

Dit betekent dat $h(a_1 \cdots a_n) = h(a_1) \cdot \ldots \cdot h(a_n)$ voor $a_i \in \Sigma$, en in het algemeen voor $x, y \in \Sigma^*$ geldt $h(x \cdot y) = h(x) \cdot h(y)$.

We breiden h uit tot talen door woord voor woord te vervangen: $h(K) = \{h(x) \mid x \in K\}$.

4.11 Voorbeeld. $\Sigma = \{a, b, c\}$ en $\Delta = \{0, 1\}$. Laat $h : \Sigma \to \Delta^*$ gegeven zijn door: $h(a) = 10, h(b) = 100, h(c) = 010$. Dan
$h(bbacaa) = 100 \cdot 100 \cdot 10 \cdot 010 \cdot 10 \cdot 10 = (100)^3(10)^3$.
Evenzo geldt dat $h(acccaa) = 10 \cdot 010 \cdot 010 \cdot 010 \cdot 10 \cdot 10 = (100)^3(10)^3$.
Laat $K \subseteq \Sigma^*$ de taal $\{bac^n a^n \mid n \in \mathbb{N}\}$ zijn. Dan $h(K) = \{h(bac^n a^n) \mid n \in \mathbb{N}\} = \{100 \cdot 10 \cdot (010)^n \cdot (10)^n \mid n \in \mathbb{N}\} = \{(100)^{n+1} \cdot (10)^{n+1} \mid n \in \mathbb{N}\}$.

Inverse homomorfismen. Net als bij elke functie kunnen we ook kijken naar de *inverse* van een homomorfisme, dwz. naar de verzameling van woorden die op een gegeven verzameling afgebeeld worden: voor $h : \Sigma^* \to \Delta^*$, en $L \subseteq \Delta^*$ definiëren we $h^{-1}(L) = \{x \in \Sigma^* \mid h(x) \in L\}$. In het algemeeen is h^{-1} geen functie. We vervolgen het voorbeeld.

4.12 Voorbeeld. Laat $L = \{(100)^n(10)^n \mid n \in \mathbb{N}\}$. We hebben al gezien dat de taal $K = \{bac^n a^n \mid n \in \mathbb{N}\}$ afgebeeld wordt in L: $h(K) \subseteq L$. Er geldt daarom dat $K \subseteq h^{-1}(L)$. $h^{-1}(L)$ bestaat echter uit meer woorden, ook *bbacaa* en *acccaa* behoren ertoe. Om $h^{-1}(L)$ volledig te vinden moeten we woorden 'ontleden'; we doen dit van links naar rechts. Als voorbeeld kijken we naar het woord $(100)^3(10)^3$.

```
100.100.100.10.10.10  ←  b.b.b.a.a.a
100.100.10.010.10.10  ←  b.b.a.c.a.a
100.10.010.010.10.10  ←  b.a.c.c.a.a
10.010.010.010.10.10  ←  a.c.c.c.a.a
```

Aan de hand hiervan vermoeden we dat

$$h^{-1}(L) = \{\lambda\} \cup \{\, b^k a c^{n-k} a^{n-1} \mid n \geq 1, 0 \leq k \leq n \,\}.$$

4.2 Eindige Automaten

▷ SCHAUM §12.5: Finite State Automata

De automaten in SCHAUM zijn volgens de gegeven definitie altijd *deterministisch*. We geven hier een wat algemenere formulering. Het *pomplemma* SCHAUM p. 309 behandelen we niet.

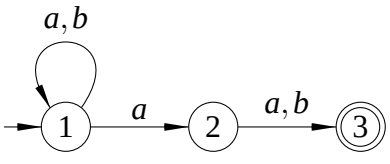
4.2.1 Inleiding

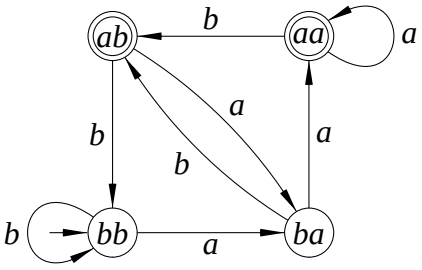
Gerichte grafen zijn een goed middel om de werking van bepaalde systemen weer te geven. De toestanden van het systeem corresponderen dan met de punten uit de graaf, de overgangen van een toestand naar een andere met de pijlen uit de graaf. De graaf dient als model van het systeem, en kan gebruikt worden om het systeem te analyseren of te simuleren. De graaf kan nog verder versierd worden: de takken kunnen namen krijgen om de acties van het systeem weer te geven, en evenzo kunnen we de punten voorzien van nadere informatie over de toestand. We kunnen bijvoorbeeld de aanvangstoestand van het systeem markeren, of toestand waarin het systeem stabiel is (wat dat ook moge betekenen).

4.13 Voorbeeld. We modelleren een systeem, met acties a en b dat als een-na-laatste actie (voor het stoppen) een actie a doet. In de initiële toestand mag elke actie gedaan worden, alvorens terug te keren in die toestand. Voor terminatie van het systeem moet de actie a gedaan worden, daarna een willekeurige actie, en we stoppen.

Het plaatje hieronder links *beschrijft* dit gedrag. Het geeft geen *algoritme* om van een string over $\{a,b\}$, 'on the fly' van links naar rechts lezend, te bepalen of die string als een-na-laatste actie een a had. De gegeven graaf laat ons in toestand 1 de keuze om bij actie a de tak $(1,a,1)$ te kiezen of de tak $(1,a,2)$. Dit werkt alleen als we van te voren weten wanneer we aan de een-na-laatste actie zijn toegekomen.

Voor een algoritmische methode moeten we steeds de twee laatste letters onthouden, zodat we op elk moment kunnen besluiten of de reeds gelezen string een juiste actie-reeks beschrijft. Dit geeft het tweede plaatje.





4.2.2 Definities

Een eindige automaat is niets anders dan een gerichte *multi*graaf met labels op de pijlen; de gelabelde gerichte graaf SCHAUM p.202. Hier zijn de reële gewichten echter vervangen door symbolen, een abstractie van de 'acties' van de gemodelleerde systemen. De gerichte graaf is bovendien voorzien van een speciaal beginpunt en een verzameling eindpunten.

4.14 Definitie. Een *eindige automaat* is een vijf-tupel $\mathscr{A} = (Q, \Sigma, E, q_{\text{in}}, F)$, waarbij

- Q een eindige, niet-lege, verzameling *toestanden*,
- Σ een alfabet,
- $E \subseteq Q \times \Sigma \times Q$ een verzameling *takken*,
- $q_{\text{in}} \in Q$ de *begintoestand*,
- $F \subseteq Q$ de verzameling *eindtoestanden*. □

4.15 Voorbeeld. De automaten uit het vorige voorbeeld worden als volgt gespecificeerd:

$Q = \{\, 1,2,3 \,\}$, $\Sigma = \{\, 0,1 \,\}$,
$E = \{\, (1,0,1),(1,1,1),(1,0,2),(2,0,3),(2,1,3) \,\}$,
$q_{\text{in}} = 1$, $F = \{\, 3 \,\}$,

respectievelijk:

$Q = \{\, 11,10,01,00 \,\}$, $\Sigma = \{\, 0,1 \,\}$,
$E = \{\, (ij,k,jk) \mid i,j,k \in \{0,1\} \,\}$
$q_{\text{in}} = 11$, $F = \{\, 01,00 \,\}$.

De *graaf-representatie* die we gebruiken is duidelijk. Toestanden corresponderen met punten, en takken (toestandsovergangen) corresponderen met pijlen voorzien van een label. De begintoestand heeft een pijltje ↖○ ter onderscheiding, de eindtoestanden een dubbele cirkel ◎.

De tak (p,a,q) uit E loopt van p naar q met *label* a. De begrippen *wandeling* en *cykel* zijn voor de hand liggende generalisaties van de begrippen voor gerichte grafen. Het zijn reeksen aaneengeschakelde pijlen (takken, toestandsovergangen) hier voorzien van een label.

De wandeling $\pi = (p_0,a_1,p_1)(p_1,a_2,p_2)\ldots(p_{n-1},a_n,p_n)$ noteren we ook wel als $\pi = p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \ldots p_{n-1} \xrightarrow{a_n} p_n$. We zeggen dat deze wandeling *loopt van* p_0 *naar* p_n, met *label* $a_1a_2\ldots a_n$; zij heeft *lengte* n.

De *taal* bij een automaat bestaat uit de labels afgelezen langs wandelingen van begin- naar eindtoestand.

4.16 Definitie. Zij $\mathscr{A} = (Q, \Sigma, E, q_{\text{in}}, F)$ een eindige automaat. De taal van $\mathscr{A}$, genoteerd $L(\mathscr{A})$ is de verzameling

$$\{\, w \in \Sigma^* \mid \text{er is een wandeling van } q_{\text{in}} \text{ naar een toestand in } F \text{ met label } w \,\}$$

□

Het begrip wandeling kunnen we ook op een inductieve manier definieren, net als de afsluiting van een relatie, net als de ster van een taal (bladzijde 32).

Zij $\mathscr{A} = (Q, \Sigma, E, q_{\text{in}}, F)$ een eindige automaat. Gebaseerd op de éénstaps-relatie $E \subseteq Q \times \Sigma \times Q$ ('van p naar q met label a') maken we daar als volgt wandelingen van:

(i) $(q, \lambda, q) \in E^*$ voor elke $q \in Q$,
(ii) als $(p, x, q) \in E^*$ en $(q, a, r) \in E$, dan $(p, xa, r) \in E^*$
(iii) E^* bevat geen drietallen anders dan die met (i) en (ii) gevonden.

Nu geldt $(p, x, q) \in E^*$ desdals er een wandeling is van p naar q met label x. Als alternatief kunnen ook de machten E^n, n-staps verbindingen, genomen worden en dan $E^* = \bigcup_{n \geq 0} E^n$. De *drie* begrippen, wandeling, E^* inductief, E^* als oneindige vereniging, zijn allemaal equivalent. Dat zou (net als vergelijkbare definities voor de ster K^* van een taal K, of de afsluiting R^* van een binaire relatie, Definitie 1.5) formeel bewezen kunnen worden met inductie (natuurlijk).

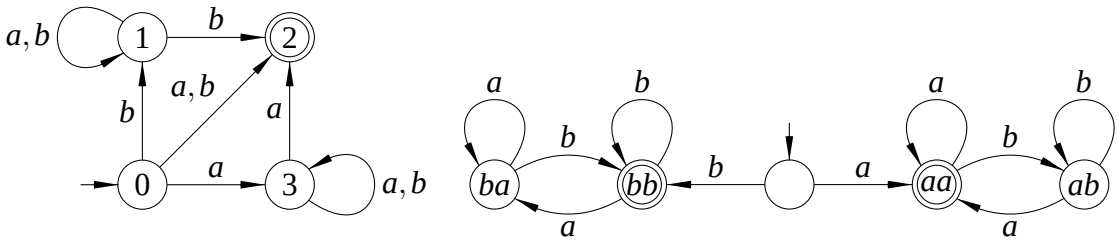
4.17 Opmerking. De notatie hier is dubbelzinnig. De takken van de automaat E vormen enerzijds de eenstaps-relatie $E \subseteq Q \times \Sigma \times Q$ en bepalen de relatie E^* die het effect van wandelingen samenvat. In Voorbeeld 4.13, rechter automaat, vinden we bijvoorbeeld $(bb, aaaba, ba) \in E^*$ omdat er een wandeling $bb \xrightarrow{a} ba \xrightarrow{a} aa \xrightarrow{a} aa \xrightarrow{b} ab \xrightarrow{a} ba$ is van toestand bb naar toestand ba met label $aaaba$.

Anderzijds vormen de takken een eindige, niet-lege, verzameling, een alfabet. Daarmee is E^* de verzameling van alle strings over E. In het voorbeeld is (ba, a, aa) $(aa, a, aa)(bb, a, ba)(ab, b, bb)$ zo'n string. Let wel, deze string stelt geen wandeling voor!

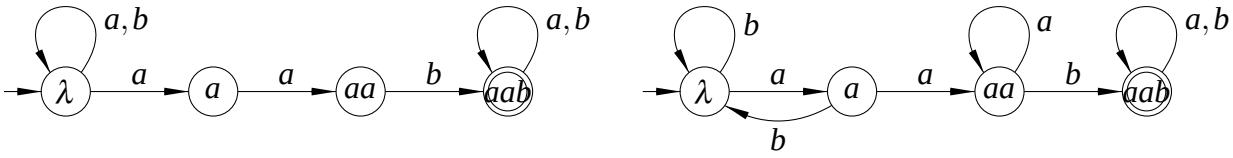
Dit is wat informatici *overloading* noemen, dubbelzinnige notatie waarvan uit de context moet blijken welke betekenis zij heeft. □

4.2.3 Voorbeelden

Laatste en eerste letter. Automaat voor de taal
$\{\, x \in \{a, b\}^* \mid$ de eerste letter van x is gelijk aan de laatste letter van $x \,\}$. De automaat onthoudt als het ware de eerste letter die langs de wandeling gelezen wordt, en mag daarna alleen eindigen wanneer de laatste letter op de wandeling daaraan gelijk is. Twee versies, beschrijvend en algoritmisch.

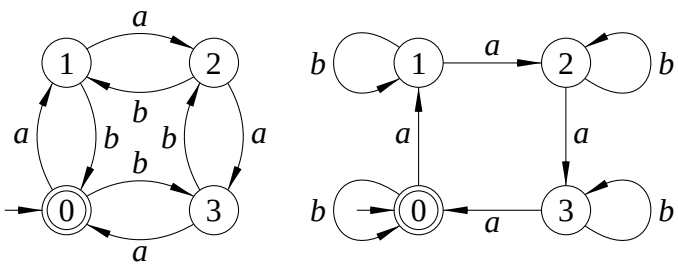


Deelwoord. De taal van alle woorden over $\{a,b\}$ met deelwoord *aab*, $\{a,b\}^*\{\ aab\ \}\{a,b\}^*$, beschrijvend en algoritmisch.



Tellen. Met een eindige automaat kunnen we niet onbeperkt tellen, zoals we in Stellling 4.21 zullen zien, we hebben tenslotte maar een eindig aantal toestanden tot onze beschikking.

Tellen modulo een constante (Hoofdstuk 3.1) lukt wel. We geven automaten voor $\{\ w \in \{a,b\}^* \mid \#_a(w) - \#_b(w) = 0 \pmod 4\ \}$, en
$\{\ w \in \{a,b\}^* \mid \#_a(w) = 0 \pmod 4\ \}$.

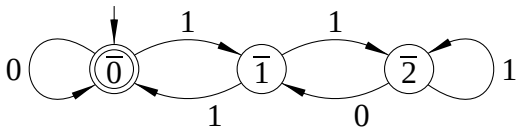


Echt modulo-rekenen met de representatie van een getal gebeurt in het volgende voorbeeld.

Modulo rekenen. Voor $w \in \{0,1\}^*$ bepalen we $\mathrm{val}(w)$ als de waarde die de string heeft, gelezen als binair getal. Dus, $\mathrm{val} : \{0,1\}^* \to \mathbb{N}$ is een functie, inductief gedefinieerd door

(i) $\mathrm{val}(\lambda) = 0$,
(ii) $\mathrm{val}(x0) = 2\mathrm{val}(x)$,
$\mathrm{val}(x1) = 2\mathrm{val}(x)+1$, voor $x \in \{0,1\}^*$.

We bepalen een automaat voor de taal $\{ x \in \{0,1\}^* \mid \mathrm{val}(x) = 0 \pmod 3 \}$.
De toestanden van de automaat zijn de restklassen $\overline{0}, \overline{1}, \overline{2}$. Een langs een wandeling gelezen woord eindigt in de juiste restklasse. Dat kan door te zorgen dat elke tak de juiste berekening volgt, nl. $(\bar{i}, 0, \overline{2i})$ en $(\bar{i}, 1, \overline{2i+1})$



4.2.4 Determinisme

Een eindige automaat bepaalt een algoritme dat voor een woord na gaat of dit tot de taal behoort als er vanuit elke toestand voor elke letter precies één uitgaande tak met die letter is. We kunnen dan de unieke wandeling bij het woord volgen om zo aan de laatste toestand het antwoord af te lezen.

We formaliseren dat als volgt.

4.18 Definitie. Een eindige automaat $\mathscr{A} = (Q, \Sigma, E, q_{\mathrm{in}}, F)$ heet (totaal) *deterministisch* als E een functie van $Q \times \Sigma$ naar Q is. □

Als voorheen kunnen we E uitbreiden tot een relatie E^* in $Q \times \Sigma^* \times Q$. Bij een deterministische automaat is dit een functie (van $Q \times \Sigma^*$ naar Q). Voor elke $(q,w) \in Q \times \Sigma^*$ geeft $E^*(q,w)$ de eindtoestand van de *unieke* wandeling met label w beginnend in q. De taal van de automaat kunnen we nu schrijven als $\{ w \in \Sigma^* \mid E^*(q_{\mathrm{in}}, w) \in F \}$.

4.19 Voorbeeld. De rechter automaat uit Voorbeeld 4.13 is deterministisch. Als functie geschreven hebben we de volgende takken: $E(ij,k) = jk$ voor alle $ij \in Q$ en alle $k \in \Sigma$. In tabelvorm vinden we bij elke toestand ij en elke letter k een uitgaande tak naar de toestand volgens:

ij:	11	10	01	00
k: 0	10	00	10	00
1	11	01	11	01

Correctheid. Een automaat legt een taal vast. Wanneer we bij een gegeven taal een automaat construeren, moeten we er ons van vergewissen dat deze automaat

inderdaad de gevraagde taal heeft. Een methode om dit te doen is om een geschikte eigenschap van E^* (de relatie tussen woorden en eindpunt van bijbehorende wandelingen) inductief te bewijzen.

Als we het idee kennen waarmee de automaat ontworpen is, dan is het bewijs vaak een eenvoudige oefening. We geven dan een kenmerkende eigenschap voor de woorden die in elke toestand eindigen via een wandeling vanuit de begintoestand en laten dan zien dat die eigenschap behouden blijft door het volgen van de takken.

Voor deterministische automaten is die aanpak het eenvoudigst: elk woord eindigt in precies één toestand. Bij beschrijvende automaten kunnen er alternatieven voor de wandeling zijn, en ook kunnnen er woorden zijn waar geen wandeling voor bestaat (wegens het ontbreken van geschikte takken).

Het is natuurlijk niet altijd nodig om een compleet bewijs met volledige inductie te geven. Een korte uitleg, en een beroep op het gezonde verstand doen wonderen.

4.20 Voorbeeld. Hierboven werd een (deterministische) automaat voor de taal $L = \{\, x \in \{0,1\}^* \mid \mathrm{val}(x) = 0 \pmod 3 \,\}$ gegeven door de volgende takken:

$\bar{i}:$	$\overline{0}$	$\overline{1}$	$\overline{2}$
$a:0$	$\overline{0}$	$\overline{2}$	$\overline{1}$
1	$\overline{1}$	$\overline{0}$	$\overline{2}$

We tonen correctheid van de automaat aan, door de volgende eigenschap inductief te bewijzen.

Bewering. Voor $x \in \{\, 0,1 \,\}^*$, $E^*(\overline{0},x) = \bar{i}$ desdals $\mathrm{val}(x) = i \pmod 3$.

Bewijs. Inductie naar x.

basis. Voor $x = \lambda$, dan enerzijds $E^*(\overline{0},\lambda) = \overline{0}$, per definitie (van E^*), terwijl anderzijds $\mathrm{val}(\lambda) = 0 \pmod 3$, alweer per definitie (van val ditmaal).

inductiestap. We nemen aan dat de bewering klopt voor x. We tonen met behulp hiervan de bewering voor $x0$ en $x1$ aan.

Voor letter 0 bevat de automaat de takken $(\overline{0},0,\overline{0})$, $(\overline{1},0,\overline{2})$, en $(\overline{2},0,\overline{1})$. Anderzijds geldt voor $x0$ de gelijkheid $\mathrm{val}(x0) = 2\mathrm{val}(x)$.

Door de drie mogelijkheden voor $\mathrm{val}(x) \pmod 3$ te onderscheiden, zien we dat de bewering voor $x0$ ook geldt.

$\mathrm{val}(x)$	$E^*(\overline{0},x)$	tak	$E^*(\overline{0},x0)$	$\mathrm{val}(x0)$
$0 \pmod 3$	$\overline{0}$	$(\overline{0},0,\overline{0})$	$\overline{0}$	$2\cdot 0 = 0 \pmod 3$
$1 \pmod 3$	$\overline{1}$	$(\overline{1},0,\overline{2})$	$\overline{2}$	$2\cdot 1 = 2 \pmod 3$
$2 \pmod 3$	$\overline{2}$	$(\overline{2},0,\overline{1})$	$\overline{1}$	$2\cdot 2 = 1 \pmod 3$

De midelste regel van de tabel, bijvoorbeeld, wordt als volgt gelezen: Als $\mathrm{val}(x) = 1 \pmod 3$ dan is $E^*(\bar{,}x) = \{\, \overline{1} \,\}$ volgens de inductieveronderstelling. De enige tak met label 0 die deze toestand $\overline{1}$ verlaat is $(\overline{1},0,\overline{2})$, zodat

$E^*(,x0) = \{\ \overline{2}\ \}$. Anderzijds is $\mathrm{val}(x0)$ volgens de inductieve definitie gelijk aan $2\mathrm{val}(x) = 2$.

Een vergelijkbare tabel kan opgesteld worden voor $x1$, dwz. de verlenging van x met letter 1.

$\mathrm{val}(x)$	$E^*(\overline{0},x)$	tak	$E^*(\overline{0},x1)$	$\mathrm{val}(x1)$
$0 \pmod 3$	$\overline{0}$	$(\overline{0},1,\overline{1})$	$\overline{1}$	$2\cdot 0+1=1 \pmod 3$
$1 \pmod 3$	$\overline{1}$	$(\overline{1},1,\overline{0})$	$\overline{0}$	$2\cdot 1+1=0 \pmod 3$
$2 \pmod 3$	$\overline{2}$	$(\overline{2},1,\overline{2})$	$\overline{2}$	$2\cdot 2+1=2 \pmod 3$

□

4.2.5 Representeerbare talen

De talen die vastgelegd worden door een eindige automaat heten *representeerbaar*. Ze vormen een verzameling, of liever gezegd, een familie talen, genoteerd met REP. Over de familie REP is vrij veel bekend: afsluitingseigenschappen (operaties die op talen in REP toegepast mogen worden en dan tot een andere taal uit REP leiden) en alternatieve karakterisaties van talen uit REP.

Niet élke taal behoort tot REP, dwz. niet elke taal heeft een passende eindige automaat. (Er geldt zelfs REP is aftelbaar, terwijl $\mathscr{P}(\Sigma^*)$ dat niet is.) Intuitief kunnen we met een eindig aantal toestanden niet een oneindig aantal essentieel verschillende strings uit elkaar houden. We bewijzen nu SCHAUM Example 12.8 zónder het pomplemma te gebruiken.

4.21 Stelling. $\{\ a^n b^n \mid n \in \mathbb{N}\ \} \notin$ REP.

Bewijs. Bewijs door middel van tegenspraak. Neem aan dat $K = \{\ a^n b^n \mid n \in \mathbb{N}\ \}$ wél door een eindige automaat gerepresenteerd kan worden, en laat N het aantal toestanden van zo'n automaat zijn.

Voor elk woord $a^n b^n$ ($n \in \mathbb{N}$) is er een wandeling van q_{in} naar een toestand in F. Neem zo'n wandeling (het kunnen er meer zijn) voor $a^N b^N$ en bekijk de toestanden op de eerste helft van de wandeling $q_{\text{in}} = q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \ldots \xrightarrow{a} q_N$. Er zijn hier $N+1$ toestanden, terwijl de automaat er N heeft. Er moet dus tenminste één toestand dubbel voorkomen; de wandeling bevat een gesloten wandeling. We kunnen dit gesloten deel uit de wandeling voor $a^N b^N$ snijden (vgl. SCHAUMThm 8.2) en we houden zo een wandeling over van q_{in} naar een toestand in F voor een woord $a^m b^N$ met $m < N$.

Dit betekent dat dit woord ook tot de taal van de automaat behoort terwijl $a^m b^N \notin K$.

Tegenspraak. □

Zonder bewijs geven we belangrijke resultaten over eindige automaten, zie Fundamentele Informatica 2. Allereerst blijkt een willekeurige 'beschrijvende' automaat in een deterministische 'algoritmische' automaat omgezet te kunnen worden. Verder zijn de eindige automaten equivalent aan de reguliere expressies: ze specificeren dezelfde familie van talen.

4.22 Stelling.

a. *Gegeven een eindige automaat kan een deterministische eindige automaat geconstrueerd worden voor dezelfde taal.*

b. *De familie van eindige automaattalen is gelijk aan de familie van talen die met de operaties vereniging, concatenatie en ster uit eindige talen verkregen worden,* $\mathsf{REP} = \mathsf{REG}$. □

Als praktische oefening voor **b.** zullen we op het werkcollege een aantal keren de taal van een automaat (REP) schrijven met de operaties vereniging, concatenatie en ster (REG). Een systematische aanpak wordt behandeld in Fundamentele Informatica 2.

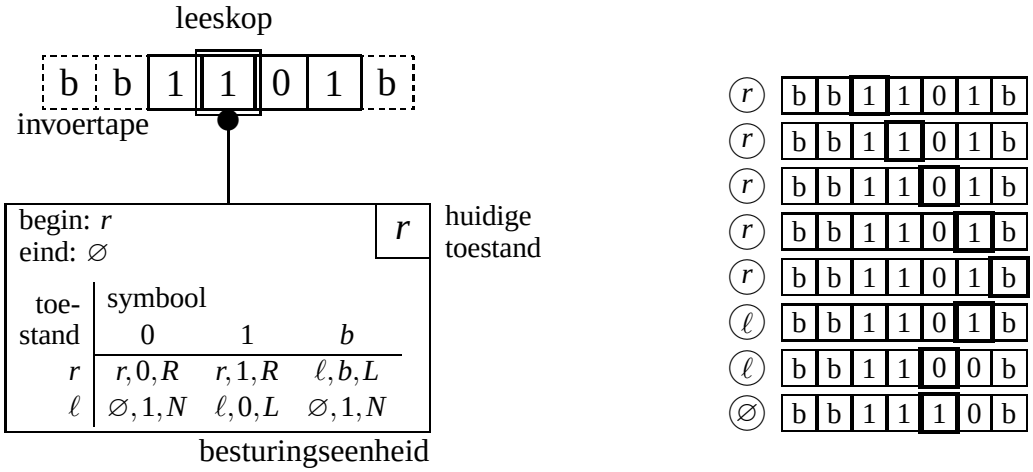
4.3 ⁎Turing Machine

Tekst ooit geschreven voor een ander doel, alleen voor liefhebbers.

Om te kunnen vergelijken hoe efficiënt algoritmen zijn, moeten we ze uitvoeren onder vergelijkbare omstandigheden, en in ieder geval op dezelfde computer. Het zou mooi zijn als er een algemeen aanvaarde standaard architectuur gekozen kon worden, zodat nieuwe algoritmen vergeleken kunnen worden met oude resultaten in de literatuur verschenen. Zo'n architectuur bestaat, en is in feite geen electronisch rekenapparaat, maar een wiskundig model. Dit model, de Turing Machine, werd opgesteld door de Britse wiskundige Alan Turing omdat deze wilde redeneren over mogelijkheden en onmogelijkheden van de Algoritmiek, het oplossen van problemen aan de hand van vaste recepten. Turing deed een fundamentele ontdekking: er zijn simpel te formuleren problemen waarvoor geen algoritmisch recept bestaat. Een wonderlijk resultaat, vooral als we ons realizeren dat deze ontdekking stamt van vóór de tweede wereldoorlog, dus van voor de bouw van de eerste computers. Turings machine is daarmee niet alleen een standaard machine om de complexiteit van algoritmen aan te ijken, maar is allereerst een fundamenteel model dat ons inzicht geeft in de mogelijkheden van de computer. Hoewel de machine heel simpel van structuur is, is iedereen er van overtuigd dat elk computer algoritme tot een programma voor de Turing Machine valt over te zetten. Dit kan niet bewezen worden, daarvoor ontbreekt een waterdichte definitie van het begrip computer. Totdat anders aangetoond is blijft de Turing Machine de maat voor wat berekenbaar is met behulp van programmeerbare computers.

De Turing Machine is gebaseerd op het idee dat een algoritme door een (mechanische) klerk uitgevoerd moet kunnen worden. De klerk heeft een onuitputtelijke voorraad papier waarop aantekeningen gemaakt kunnen worden om tijdens de verdere berekening te kunnen raadplegen. De rekenstappen die de klerk maakt moeten van te voren strikt vastgelegd worden in een *programma*, bestaande uit een eindig aantal rekenregels, *instructies*. De klerk onthoudt waar hij mee bezig is, maar heeft slechts een beperkte, eindige, geheugen capaciteit. Hij raadpleegt steeds de instructies om zijn volgende aktie te bepalen, gebaseerd op zijn aantekeningen en zijn geheugentoestand.

Hieronder een plaatje van de machine zoals door Turing voorgesteld. We zien een *besturingseenheid* die een *leeskop* langs een *invoertape* kan bewegen. De *toestand* die de besturingseenheid onthoudt, komt overeen met het beperkte beschikbare geheugen. Die toestand kan slechts een van te voren vastgelegd aantal waarden aannemen (en dat is dus slechts een eindig aantal, dat echter per Turing Machine kan variëren). De invoertape bestaat uit een aaneengeregen aantal *vakjes* of cellen, die elk een symbool bevatten. Op deze tape leest de Turing Machine zijn invoer, maar de tape dient ook als (extern) geheugen van de machine: er kan tijdens het rekenen informatie op geschreven worden. In principe is de opslagcapaciteit van



Figuur 4.1: Links een Turing Machine met drie symbolen (0,1 en *b*) en drie toestanden (*r*,ℓ en eindtoestand $\varnothing$). Rechts een berekening van deze Turing Machine op invoer 1101, getekend is steeds de tape, de positie van de leeskop (met een kader om de tape positie), en de geldende toestand (omcirkeld).

de tape onbegrensd, en loopt de tape naar twee zijden oneindig door. Lege vakjes, nog onbeschreven, worden aangegeven door het symbool *b* (voor *blank*). Hoewel er oneindig veel vakjes ter beschikking van de machine staan, is op elk moment het aantal beschreven vakjes eindig. De akties van de machine worden vastgelegd in een programma, dat uit een aantal instructies bestaat. Elk van de instructies is van de vorm (p,A,q,B,d), met de volgende interpretatie: als de toestand p is, en het symbool onder de kop is A, dan gaat de machine over in toestand q, schrijft B op de tape op de huidige positie, en beweegt de kop in de richting aangegeven door d. Hierbij kan d drie waarden aannemen: L linksaf, R rechtsaf, en N neutraal (de kop wordt niet verplaatst). De besturingseenheid bevat het eigenlijke Turing Machine programma, hier gegeven als een tabel. Onderdeel van dit programma zijn de begin- en eindtoestand. De berekening begint in de begintoestand, en stopt zodra de eindtoestand wordt bereikt. Er zijn geen instructies vanuit de eindtoestand.

Voor de Turing Machine uit Figuur 4.1 zijn er drie mogelijke toestanden, r, ℓ en de eindtoestand $\varnothing$, en drie mogelijke symbolen op de tape, 0, 1 en blank b. Het programma bevat zes instructies. Voor toestand r en tape symbool 1 vinden we in rij r en kolom 1 van de instuctietabel "$r,1,R$". Dat wil zeggen, voor r en 1 is er de instructie $(r,1,r,1,R)$; in woorden gezegd: in toestand r bij invoer 1 wordt de kop naar rechts bewogen (terwijl toestand en tape symbool gelijk blijven). In het geval van de illustratie zou na deze aktie de kop op de cel gevuld met 0 komen te staan; verder blijft alles gelijk.

4.23 Voorbeeld. We ontwerpen een Turing Machine voor een eenvoudige taak. het herkennen van het getal nul. Op de tape staat een reeks 0-en en 1-en (omgeven door blanko vakjes, dus met inhoud *b*). Deze reeks wordt gelezen en uitgeveegd (dus vervangen door *b*'s). Als er tenminste één 1 gelezen is, dan schrijft de machine 1. Anders (er stonden alleen 0-en op de tape) schrijft de machine 0.

Omdat we moeten onthouden of we een 1 zijn tegengekomen voeren we in ieder geval twee toestanden in: *n* (voor 'nul') en *e* (voor 'een'). We beginnen in toestand *n*. Bij het lezen van een 0 blijven we in toestand *n*, overschrijven we het symbool met *b* en gaan we naar rechts, volgens de instructie $(n,0,n,b,R)$. Bij het lezen van 1 doen we hetzelfde, maar onthouden we dat we dit symbool gezien hebben, en gaan we over in toestand *e* volgens de instructie $(n,0,e,b,R)$.

In toestand *e* hebben we reeds een 1 gelezen en wissen we elk symbool dat we tegenkomen, met de instructies $(e,0,e,b,R)$ en $(e,1,e,b,R)$.

Wanneer we een *b* lezen zijn we aan het eind van de invoer gekomen. Zijn we op dat moment in toestand *n* dan schrijven we 0 en stoppen we door naar de eindtoestand $\varnothing$ te gaan; in toestand *e* hebben we eerder een 1 gelezen en schrijven we 1. In deze laatste stap bewegen we de kop niet. Dit levert instructies $(n,b,\varnothing,0,N)$ en $(e,b,\varnothing,1,N)$.

De uiteindelijke instructietabel is daarmee:

toe-	symbool		
stand	0	1	*b*
n	n,b,R	e,b,R	$\varnothing,0,N$
e	e,b,R	e,b,R	$\varnothing,1,N$

Terug naar de eerste Turing Machine zoals gegeven in Figuur 4.1. Deze eenvoudige Turing Machine zal, geplaatst op een invoer string die uit nullen en enen bestaat, in toestand *r* naar rechts over de tape bewegen totdat een eerste *b* gevonden wordt. Op dat moment staat de leeskop net ná de invoer. De toestand wordt dan ℓ en de kop gaat een vakje terug naar links. Nu loopt de machine naar links, zolang een 1 gelezen wordt. Deze 1 wordt dan telkens door een 0 vervangen. Wanneer de machine, zo doorgaand in toestand ℓ, een 0 of een *b* vindt, wordt deze door een 1 overschreven, de machine gaat over in toestand $\varnothing$ en stopt.

Het is misschien niet direct duidelijk, maar dit programma doet wel degelijk iets nuttigs; het telt bij de binaire invoer één op.