Formeel Denken

Herman Geuvers
Deels gebaseerd op het herfst 2002 dictaat van Henk Barendregt en Bas Spitters,
met dank aan het Discrete Wiskunde dictaat van Wim Gielen

December 21, 2006

Contents

1. Propositielogica

In dit blok behandelen we de propositielogica, ook wel uitspraakrekening genoemd.

1.1. Formele taal en natuurlijke taal

Natuurlijke talen (Nederlands, Engels, Duits,...) zijn niet erg precies. Kijk bijvoorbeeld maar naar de volgende voorbeelden:

- Socrates is een mens. Een mens is sterfelijk. Dus Socrates is sterfelijk.
- Ik ben iemand. Iemand schilderde de Mona Lisa. Dus ik ben de schilder van de Mona Lisa.

De eerste redenering klopt, de tweede niet, maar heeft wel een soortgelijke vorm.

Is de zin

Deze zin is niet waar.

nu waar of niet?

Om dit soort problemen te voorkomen gebruiken we een formele taal. Dit is een soort laboratorium-model voor de natuurlijke taal. We zullen zien dat we met een heel eenvoudige kunst-taal, toch heel veel uitspraken en redeneringen heel precies kunnen maken. Dit is bijvoorbeeld erg belangrijk voor het specificeren van programma's, over de specificatie mag geen misverstand bestaan.

We zullen nu laten zien hoe we van het Nederlands naar een formele taal gaan. Om te redeneren combineren we vaak een aantal eenvoudige uitspraken, bijvoorbeeld: '*als* het regent *en* ik ben buiten, *dan* word ik nat'. Deze uitspraak is opgebouwd uit de eenvoudige uitspraken 'het regent', 'ik ben buiten' en 'ik word nat'.

1.2. Woordenboek

We kunnen dit ook formeler opschrijven met behulp van een woordenboekje.

R	het regent
Z	de zon schijnt
RB	er is een regenboog
N	ik word nat
D	ik blijf droog
Bui	ik ben buiten
Bin	ik ben binnen

De zin hierboven wordt dan 'als R en Bui, dan N'. Net zo, kunnen we de volgende zinnen maken: 'als RB, dan Z', 'Z en RB', 'als R en Bin, dan D'.

1.3. Verbindingswoorden

We kunnen ook de verbindingswoorden formeel opschrijven.

Dat doen we op de volgende manier:

Formele taal	Nederlands
$f \wedge g$	f en g
$f \vee g$	f of g
$f \to g$	als f, dan g
$f \leftrightarrow g$	f dan en slechts dan als g
$\neg f$	niet f

De zinnetjes hierboven worden dan 'RB→Z', 'Z∧RB' en '(R∧Bin)→D'.

Nederlands	semi-formeel	Formeel
als het regent *en* ik ben buiten, *dan* word ik nat	als R en Bui, dan N	$(R \wedge \text{Bui}) \to N$
als er een regenboog is, *dan* schijnt de zon	als RB, dan Z	$RB \to Z$
ik ben binnen *of* buiten	Bin of Bui	$\text{Bin} \vee \text{Bui}$

1.1. Opgave. Vind zinnen uit de formele taal die hetzelfde betekenen als de volgende zinnen.

1. Het regent niet, noch schijnt de zon.
2. De zon schijnt, tenzij het regent
3. Of de zon schijnt, of het regent. (We bedoelen dus niet allebei)
4. Er is alleen een regenboog als de zon schijnt en het regent.
5. Als ik buiten ben, dan word ik nat, mits het regent.

We denken bij de voegtekens (zoals $\vee, \wedge$) natuurlijk aan de bekende Nederlandse verbindingswoorden, Ook 'of' is zo'n voegwoord, dat van twee beweringen een nieuwe bewering maakt. De betekenis van 'of' is in ons taalgebruik echter een beetje onduidelijk: sommige mensen vinden de bewering '$1 + 1 = 2$ of $2 + 3 = 5$' waar, en anderen onwaar. Wat vind jij ervan? Als je erover nadenkt merk je, dat de betekenis van het woordje 'of' in ons taalgebruik dubbelzinnig is. Maar informaticastudenten haten dubbelzinnige woordjes, dus wij zullen van de twee gangbare betekenissen van 'of' er één kiezen: we spreken af dat 'A of B' waar is in het geval A en B beide waar zijn. Ons voegwoord 'of', afgekort tot '$\vee$', heeft dus de volgende waarheidstabel:

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

1.2. Opgave. Kun je $\leftrightarrow$ ook uitdrukken met behulp van de andere symbolen?

1.3. Opgave. Vertaal de volgende zinnen naar het Nederlands:

1. $R \leftrightarrow Z$
2. $\text{RB} \to (R \wedge Z)$
3. $\text{Bui} \to \neg\,\text{Bin}$

4. Bui $\vee$ Bin

1.4. Definitie. De taal van de uitspraakrekening (of propositielogica) is de volgende formele taal. Het alfabet bestaat uit een oneindige verzameling

$$A := \{a, b, c, d, a_1, a_2, a_3, \ldots\}$$

van Atomen, de verzameling

$$V := \{\wedge, \vee, \rightarrow, \leftrightarrow, \neg\}$$

van Voegtekens en de verzameling

$$H := \{(,)\}$$

van Haakjes.
Woorden uit deze taal worden als volgt opgebouwd:

1. Een atoom is een woord.
2. Als f en g woorden zijn, dan zijn $(f \wedge g), (f \vee g), (f \rightarrow g), (f \leftrightarrow g)$ en $\neg f$ woorden.
3. Alle woorden worden op deze manier gevormd.

De woorden van deze taal noemen we *proposities*.

1.5. Conventie. Meestal laten we de buitenste haakjes weg, dus we schrijven bijvoorbeeld $a \wedge b$ in plaats van $(a \wedge b)$. De binnenste haakjes mogelijk we natuurlijk niet weg laten: vergelijk $(R \wedge Z) \rightarrow RB$ en $R \wedge (Z \rightarrow RB)$. We kunnen wel een notatie *afspreken* waarbij we *sommige* haakjes weglaten en dat doen we ook door middel van de volgende afspraak:

- $\neg$ bindt sterker dan $\wedge$
- $\wedge$ bindt sterker dan $\vee$
- $\vee$ bindt sterker dan $\rightarrow$
- $\rightarrow$ bindt sterker dan $\leftrightarrow$

Dit betekent dat we Bin $\vee$ RB $\rightarrow$ Bui $\leftrightarrow \neg$ Z $\wedge$ R moeten lezen als ((Bin $\vee$ RB) $\rightarrow$ Bui) $\leftrightarrow$ ($\neg$ Z $\wedge$ R)

1.6. Opmerking. (Deze opmerking heeft pas betekenis na het blok "Talen"; bekijk hem dan eens goed.) De formele definitie m.b.v. een contextvrije grammatica is als volgt: $\Sigma = A \cup V \cup H$, ofwel

$$\Sigma = \{a, b, c, \ldots., \wedge, \vee, \rightarrow, \leftrightarrow, \neg, (,), \}.$$

($\cup$ geeft de vereniging van twee verzamelingen aan).

$$\boxed{\begin{array}{l} S \rightarrow a \mid (SvS) \mid \neg S \\ v \rightarrow \vee \mid \wedge \mid \rightarrow \mid \leftrightarrow \end{array}}$$

hier mag a ieder element van A zijn.

1.4. Betekenis en waarheidstabellen

De zin 'als a en b, dan a' is waar, wat je ook voor a en b invult. We zouden nu ook willen zeggen dat de zin '$a \wedge b \to a$' waar is [1], maar we hebben nog niet gedefinieerd wat dat betekent: '$a \wedge b \to a$' is zomaar een zin in een taal. We gaan nu definiëren wat de betekenis van een zin is, in het bijzonder, wanneer een zin in de taal van de logica *waar* is.

Voor de atomen kunnen we aan eenvoudige uitspraken denken zoals '2=3', 'Jolly Jumper is een paard' of 'het regent'. In de klassieke logica, waar we ons in dit college toe beperken, nemen we aan dat alle atomen of waar zijn of niet waar zijn. We maken ons geen zorgen om het feit dat we soms niet weten of de zin waar of niet waar is, zoals bij de zin, 'op 1 januari 2050 regent het in Nijmegen'.

De waarheid van de atomen wordt bepaald door hun interpretatie in een *model*. Bijvoorbeeld '2=3' is niet waar in het model van de natuurlijke getallen. 'Jolly Jumper is een paard' is waar in het model van het stripboek Lucky Luke. De zin 'het regent' was niet waar in Nijmegen op 17 september 2002.

Bekijk nu eens de zin '$a \wedge b$'. We willen dat deze zin alleen waar is in een model als zowel a als b waar zijn in dat model. Als we nu een lijstje maken met alle mogelijke waarden voor a en voor b, dan kunnen we ook bepalen wat de mogelijke waarden zijn voor $a \wedge b$.

In de informatica schrijven we vaak 1 voor *waar* en 0 voor *onwaar*. De logische operaties zijn de elementaire operaties op bits.

We krijgen dan de volgende tabel:

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

'Als $\cdots$, dan $\cdots$' is eveneens een methode om van twee beweringen een nieuwe bewering te maken. Ook hier geeft ons gangbare taalgebruik geen duidelijk antwoord op de vraag, wanneer 'als A, dan B' waar is. Laten we bijvoorbeeld eens kijken naar het geval, dat A onwaar is en ook B onwaar is. Is 'als A, dan B' in dat geval waar? Wat vind je ervan? Denk bijvoorbeeld eens aan:

'als $1 + 1 = 3$, dan $2 + 2 = 6$'
'als ik van het Erasmusgebouw spring, dan verander ik in een vogeltje'
'als ik hier iets van snap, dan heet ik Alpje'

We zullen aan alle onduidelijkheid een einde maken door een waarheidstabel af te spreken (we schrijven '$A \to B$' in plaats van 'als A, dan B'). Dit staat allemaal in de volgende definitie.

1.7. Definitie. De *waarheidstabellen* voor de logische voegtekens zijn als volgt

x	$\neg x$
0	1
1	0

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

[1] Let op: $a \wedge b \to a$ moeten we dus lezen als $(a \wedge b) \to a$

x	y	$x \to y$
0	0	1
0	1	1
1	0	0
1	1	1

x	y	$x \leftrightarrow y$
0	0	1
0	1	0
1	0	0
1	1	1

Met behulp van de waarheidstabellen kunnen we de waarde van een complexe propositie uitrekenen als we de waarde van de atomen kennen. Zo kunnen we ook voor een complexe propositie een waarheidstabel maken die alle mogelijkheden weergeeft.

1.8. VOORBEELD. De waarheidstabel van $a \vee b \to a$.

a	b	$a \vee b$	a	$(a \vee b) \to a$
0	0	0	0	1
0	1	1	0	0
1	0	1	1	1
1	1	1	1	1

1.9. OPGAVE. Schrijf de waarheidstabellen op van: $a \vee \neg a$, $(a \to b) \to a$, $a \to (b \to a)$, $a \wedge b \to a$, $a \wedge (b \to a)$, $\neg a \to \neg b$.

1.5. Modellen en waarheid

In de introductie spraken we over *modellen* en *waarheid in een model*. In de waarheidstabellen hebben we gezien dat de 'waarheid' van een propositie volledig bepaald wordt door de waarden die we aan de atomen toekennen. Een model in de propositielogica is dan ook eenvoudigweg een toekenning van waarden ($\{0, 1\}$) aan de atomen.

1.10. DEFINITIE. Een *model* in de propositielogica is een *waardetoekenning* of *valuatie* van de atomen: een functie $v : A \to \{0, 1\}$.

Om de waarde van een propositie f te bepalen hoeven we niet de waarden van alle atomen te weten, maar alleen van de atomen die in f voorkomen. We zullen daarom een model vaak gelijkstellen aan een eindige waardentoekenning.

1.11. VOORBEELD. In het model v met $v(a) = 0$ en $v(b) = 1$ heeft de propositie $a \vee b \to a$ de waarde 0.

1.12. CONVENTIE. Als v een model is, dan schrijven we ook wel $v(f)$ voor de waarde van f. Als deze waarde 1 is zeggen we dat f *waar is in het model* v.

Dus $a \vee b \to a$ is niet waar in v als $v(a) = 0$ en $v(b) = 1$, maar $a \to (b \to a)$ is wel waar in zo'n model.

1.13. DEFINITIE. Als een propositie f waar is in ieder model (dat wil zeggen dat er in de waarheidstabel van f alleen maar 1-en staan), dan noemen we de propositie *logisch waar*. Notatie $\vDash f$. Een logisch ware propositie heet ook wel een *tautologie*.

1.14. OPGAVE. Welke van de volgende proposities zijn logisch waar? $a \vee \neg a$, $a \to (a \to a)$, $a \to a$, $(a \to b) \to a$, $a \to (b \to a)$, $a \wedge b \to a$, $a \vee (b \to a)$,

1.15. Opgave. Laat f en g proposities zijn. Ga na of de volgende uitspraken kloppen:

1. Als $\vDash f$ en $\vDash g$, dan $\vDash f \wedge g$.
2. Als niet $\vDash f$, dan $\vDash \neg f$.
3. Als $\vDash f$ of $\vDash g$, dan $\vDash f \vee g$.
4. Als (als $\vDash f$, dan $\vDash g$), dan $\vDash f \to g$.
5. Als $\vDash \neg f$, dan niet $\vDash f$.
6. Als $\vDash f \vee g$, dan $\vDash f$ of $\vDash g$.
7. Als $\vDash f \to g$, dan (als $\vDash f$, dan $\vDash g$).
8. Als $\vDash f \leftrightarrow g$, dan ($\vDash f$ dan en slechts dan als $\vDash g$).
9. Als ($\vDash f$ dan en slechts dan als $\vDash g$), dan $\vDash f \leftrightarrow g$.

1.6. Logisch Equivalent

1.16. Definitie. Twee proposities zijn *logisch equivalent* als f waar is in een model, dan en slechts dan als g waar is in dat model.
Iets preciezer geformuleerd: f en g zijn logisch eqivalent als voor ieder model v geldt dat $v(f) = 1$ dan en slechts dan als $v(g) = 1$. Dat wil zeggen dat f en g dezelfde waarheidstabel hebben.
Als f en g equivalente proposities zijn schrijven we ook wel $f \equiv g$.

Vaak kunnen we een propositie vervangen door een eenvoudigere equivalente propositie. Bijvoorbeeld $a \wedge a$ is equivalent met a.

1.17. Opgave. Laat zien dat de twee proposities $(a \wedge b) \wedge c$ en $a \wedge (b \wedge c)$ logisch equivalent zijn.

Laat zien dat de twee proposities $(a \vee b) \vee c$ en $a \vee (b \vee c)$ logische equivalent zijn.

In dit geval doen de haakjes er dus eigenlijk niet toe. Soms laten we in zo'n geval de haakjes ook wel weg.

Let op: de proposities: $a \wedge b$ en $b \wedge a$ zijn logisch equivalent. In het Nederlands is dit niet altijd eenduidig, met de zin 'ze trouwde en kreeg een kind' bedoelen we waarschijnlijk wat anders dan met de zin 'ze kreeg een kind en trouwde'.

De eerste twee van de volgende equivalenties heten de wetten van *De Morgan*.

1. $\neg(f \wedge g) \equiv \neg f \vee \neg g$.
2. $\neg(f \vee g) \equiv \neg f \wedge \neg g$.
3. $f \wedge (g \vee h) \equiv f \wedge g \vee f \wedge h$.[2]
4. $f \vee g \wedge h \equiv (f \vee g) \wedge (f \vee h)$.

1.18. Opgave. Laat f en g proposities zijn. Is de volgende uitspraak waar? $f \equiv g$ dan en slechts dan als $\models f \leftrightarrow g$. waar is.

[2]NB: Volgens de haakjesconventie is dit $(f \wedge g) \vee (f \wedge h)$.

1.7. Logisch gevolg

In het Nederlands volgt uit 'het regent en de zon schijnt' dat 'de zon schijnt'. Net zo willen we dat uit $a \wedge b$ volgt dat a. Dat maken we nu precies.

1.19. Definitie. Een propositie g is een *logisch gevolg* van de propositie f als g waar is in ieder model waar f waar is. Anders gezegd: als er op alle plaatsen waar in de waarheidstabel van f een 1 staat er in de waarheidstabel van g ook een 1 staat. Notatie $f \models g$.

1.20. Opgave. Zijn de volgende uitspraken waar? $a \wedge b \vDash a$, $a \vee b \vDash a$, $a \vDash a \vee b$, $a \wedge \neg a \vDash b$.

1.21. Stelling. Laat f en g proposities zijn.

$$\vDash f \to g, \text{ dan en slechts dan als } f \vDash g.$$

Voor meer informatie over logica kun je bijvoorbeeld het boek [vBe] bekijken.

2. Predicatenlogica

"Een vrouw is gelukkig als zij van iemand houdt
en een man is gelukkig als iemand van hem houdt."

We gaan het niet hebben over de juistheid van deze zin, maar over hoe we deze formeel kunnen opschrijven. Dat zal als voordeel hebben dat we beter kunnen analyseren of dit soort zinnen inderdaad waar zijn.

We springen meteen in het diepe en geven de formalisering van bovenstaande zin.

Woordenboek

V	verzameling van (alle) vrouwen
M	verzameling van (alle) mannen
$H(x, y)$	x houdt van y
$G(x)$	x is gelukkig

Vertaling

$$\forall v{\in}V[(\exists x{\in}(M \cup V)\ H(v, x)){\rightarrow}G(v)] \wedge \forall m{\in}M\ [(\exists x{\in}(M \cup V)\ H(x, m)) \rightarrow G(m)]. \qquad (1)$$

De $\forall$ staat hier voor "voor alle" en de $\exists$ staat voor "er is een". Als we deze formule (bijna) letterlijk terugvertalen naar natuurlijke taal staat er

> Voor iedere vrouw geldt dat als er een persoon is van wie zij houdt dan is zij gelukkig en voor iedere man geldt dat als er een persoon is die van hem houdt dan is hij gelukkig.

Ga zelf na dat dit hetzelfde zegt als de zin waar we mee begonnen en ga ook na dat je dit uit de formule kunt aflezen. Je ziet dat we ook andere haakjes [_] gebruiken. Dat is alleen voor de leesbaarheid, zodat je beter kunt zien welk begin-haakje bij welk eind-haakje hoort.
Omdat een woordenboek de mogelijkheid geeft om een uitspraak te vertalen, noemen we het ook wel een *interpretatie*.

2.1. Predicaten, relaties en individuen

Toen we nog propositielogica deden konden we de zin

Als Sharon gelukkig is, dan is Koos niet gelukkig

als volgt formaliseren.

Woordenboek

SG	Sharon is gelukkig
KG	Koos is gelukkig

Vertaling

$$SG \rightarrow \neg KG$$

Maar als er nu ook nog Joris en Maud zijn, dan krijgen we wel een lang woordenboek met de extra proposities JG en MG.

We zetten daarom een vergrootglas op de uitspraak

Sharon is gelukkig

en analyseren deze als bestaande uit het predicaat

$G(\ )$ toegepast op het *subject* s (Sharon)

en dat schrijven we als

$$G(s).$$

Het voordeel is dat we dan meteen kunnen schrijven

$$G(k),\ G(j),\ \text{en } G(m).$$

Ook voor de subjectnamen gebruiken we het woordenboek om aan te geven welk subject er bij welke naam hoort. Bovendien geven we hier aan tot welk domein het subject behoort.

s	Sharon $\in$ "vrouwen"
k	Koos $\in$ "mannen"
j	Joris $\in$ "mannen"
m	Maud $\in$ "vrouwen"

Het voordeel is nu nog niet duidelijk. In plaats van vier proposities $SG, KG,\ JG, MG$ hebben we nu één predicaat G en vier subjecten s, k, j, m dat is dus in totaal vijf symbolen. Maar onze vier spelers kunnen nog meer eigenschappen hebben

$L(x)$	x is lang
$K(x)$	x is knap
$A(x)$	x is aardig
$I(x)$	x is intelligent

Bovendien zijn er naast de predicaten (eigenschappen) ook nog (binaire) relaties. Bijvoorbeeld:

$$H(x, y):\ x \text{ houdt van } y.$$

We kunnen nu formaliseren

Sharon is een intelligente knappe vrouw; er is een aardige lange man die van zo'n vrouw houdt

$I(s) \wedge K(s) \quad \wedge$
$\exists x{\in}M(A(x) \wedge L(x) \wedge \exists y{\in}V\ [H(x, y) \wedge K(y) \wedge I(y)]).$

Of bedoelen we

$I(s) \wedge K(s) \quad \wedge$
$\exists x{\in}M\ [L(x) \wedge A(x) \wedge H(x, s)]?$

Merk op dat we de twee beweringen in de twee zinnen tot één formule maken door er een $\wedge$ tussen te zetten. Dat Sharon een vrouw is ligt besloten in het woordenboek, dus die informatie hoeven we niet meer toe te voegen (d.m.v. "$s \in V$" o.i.d.).

Of we de eerste of de tweede formalisatie verkiezen hangt ervan af hoe we de verwijzing "zo'n vrouw" vertalen. (Het kan slaan op "Sharon", maar ook op "intelligente knappe

vrouw".) Hier gaat de logica zelf niet over, maar de logica maakt wel expliciet dat er hier een keuze gemaakt moet worden; de ambiguïteit in de natuurlijke taal wordt expliciet gemaakt.

2.1. Opgave. Geef twee mogelijke vertalingen voor de volgende zinnen.

Sharon houdt van Maud; een aardige man houdt van zo'n knappe vrouw.

We gaan nu de andere kant op, een formele uitspraak ontcijferen.

2.2. Opgave. Vertaal de onderstaande formules naar een zin in het Nederlands.
(i) $\exists x{\in}M[L(x) \wedge \exists v{\in}V\ [K(v) \wedge I(v) \wedge H(x,v)]]$
(ii) $\exists x{\in}M\ [L(x) \wedge \exists v{\in}V\ [K(v) \wedge \neg I(v) \wedge H(x,v)] \wedge \exists w{\in}V[I(w) \wedge H(w,x)]]$

2.2. De taal van de predicatenlogica

We definiëren precies wat de taal van de predicatenlogica is.

2.3. Definitie. De taal van de predicaten logica bevat de volgende ingrediënten.

1. *Variabelen*, meestal $x, y, z, x_0, x_1, \ldots$, en soms ook $v, w, \ldots$
2. *Individu constanten*, of ook wel 'namen', meestal $a, b, c, \ldots$,
3. *Domeinen*, zoals $M, V, E, \ldots$,
4. *Relatiesymbolen*, met een vaste "ariteit", die we er vaak expliciet bij schrijven, zoals P^1, R^2, T^3, ... Dit betekent dat P^1 één argument heeft, R^2 twee argumenten enz.
5. De *atomen* (of 'atomaire formules') zijn $P^1(t)$, $R^2(t_1, t_2)$, $T^3(t_1, t_2, t_3)$ enz., waarbij t, t_1, t_2 en t_3 variabelen of individu constanten zijn.
6. De *formules* worden als volgt gevormd.
   - de atomaire formules
   - als f en g formules zijn dan zijn $(f \wedge g), (f \vee g), (f \to g), (f \leftrightarrow g)$ en $\neg f$ formules.
   - als f een formule is, x een variabele en D een domein, dan zijn $(\forall x \in D\ f)$ en $(\exists x \in D\ f)$ ook formules.

2.4. Conventie. Net als bij de propositielogica laten we meestal de buitenste haakjes weg. Om ook wat op de binnenste haakjes te bezuinigen breiden we de haakjesconventie van de propositie logica (1.5) uit:

- $\forall$ en $\exists$ binden sterker dan alle andere voegtekens

Dit betekent dus dat we onze eerste formule uit de predicatenlogica (1) dus kunnen schrijven als

$$\forall v{\in}V[\exists x{\in}(M \cup V)\ H(v,x) \to G(v)] \wedge \forall m{\in}M\ [\exists x{\in}(M \cup V)\ H(x,m) \to G(m)].$$

2.5. Opmerking. Grammatica van de predicatenlogica.

Net als bij de propositie kunnen we de taal van de predicatenlogica definiëren als een formele taal. Dat gaat als volgt. Bekijk deze opmerking nog eens na het stuk over Talen.

Individu	:=	variabele | naam
Variabele	:=	$x, y, z, x_1, y_1, z_1, \ldots$
Naam	:=	$a, b, c, d, e, \ldots$
Domein	:=	$D, E, \ldots$
Atoom	:=	P^1(Individu) | R^2(Individu,Individu)
		| T^3(Individu,Individu,Individu) | ...
Formule	:=	Atoom
		| ¬ Formule
		| (Formule→Formule) | (Formule ∧ Formule)
		| (Formule ∨ Formule) | (Formule ↔ Formule)
		| (∀ variabele∈Domein Formule)
		| (∃ variabele∈Domein Formule)

Bekijk de volgende twee formules

$$\begin{aligned} F_1 &= \forall x{\in}D \exists y{\in}D\ K(x,y) \\ F_2 &= \exists x{\in}D \forall y{\in}D\ K(x,y) \end{aligned}$$

(Let op hoe we op haakjes bezuinigen.) Wanneer is een formule *waar*? Waarheid is relatief en hangt af van een *interpretatie* en een *model*. We geven hier 3 voorbeelden waarin we de waarheid van F_1 resp. F_2 kunnen nagaan.

1. Model 1

D	alle studenten in het lokaal
$K(x,y)$	x heeft een lager studentnummer dan y

2. Model 2

D	alle studenten in het lokaal
$K(x,y)$	x is niet ouder dan y

3. Model 3

D	alle studenten in het lokaal
$K(x,y)$	x zit naast y

We kunnen de waarheid van F_1, resp. F_2, in deze modellen vast stellen door te kijken of de eigenschap geldt, die door de formule tot uitdrukking wordt gebracht.

2.6. Opgave. 1. Ga na dat in het eerste model F_2 niet geldt. Geldt F_1?

2. Ga na dat in het tweede model F_1 geldt. Geldt F_2 ook?

3. Kijk om je heen en ga na of F_1 in het derde model geldt. Ga ook na of F_2 geldt.

2.7. Definitie. • Een *interpretatie* wordt gegeven door het woordenboek, waarin staat

1. Welke verzamelingen er horen bij de domeinen,

2. Welke subjecten er horen bij de namen (en in welke domeinverzamelingen deze zitten),
3. Welke predicaten en relaties er horen bij de predicaat- en relatiesymbolen.

- Een *model* is het stukje van de 'echte' wereld waarin de formules een betekenis krijgen via de interpretatie.

2.8. VOORBEELD. In het geval van Sharon, Koos, Joris en Maud is de interpretatie al gegeven in het woordenboek: we weten nu wat we met $H(s,k)$ bedoelen en wat met $\exists m{\in}M\ H(m,s)$. Het model is de feitelijke situatie omtrent deze mensen, waar we kunnen nagaan of $H(s,k)$ waar is (i.e. of Sharon van Koos houdt) en of $\exists m{\in}M\ H(m,s)$ waar is (i.e. of er een man is die van Sharon houdt).

We bekijken de formules G_1 en G_2 en we bekijken twee modellen waarin we ze kunnen interpreteren. (Merk het verschil op met F_1 en F_2 boven.)

$$\begin{aligned} G_1 &= \forall x{\in}D \exists y{\in}D\ K(x,y) \\ G_2 &= \forall x{\in}D \exists y{\in}D\ K(y,x) \end{aligned}$$

2.9. VOORBEELD. Interpretatie$_1$.

D	verzameling van natuurlijke getallen $\mathbb{N} = \{0,1,2,3,\ldots\}$;
$K(x,y)$	x is kleiner dan y (ook: $x < y$).

Interpretatie$_2$.

D	verzameling van rationale getallen $\mathbb{Q}$ (positieve en negatieve breuken, bijvoorbeeld $-\frac{1}{2}, 3(=\frac{3}{1}), 0, \ldots$);
$K(x,y)$	x is kleiner dan y (ook: $x < y$).

Onder de interpretatie$_1$ is de formule G_1 waar. Immers, voor ieder getal $x \in \mathbb{N}$ is er een getal $y \in \mathbb{N}$ (bijvoorbeeld $x+1$) zodat $x < y$. Onder de interpretatie$_2$ is de formule G_1 waar. Immers, voor ieder getal $x{\in}\mathbb{Q}$ is er een getal $y{\in}\mathbb{Q}$ (bijvoorbeeld $x+1$) zodat $x < y$.

2.10. CONVENTIE. Als een formule f waar is in een model M onder interpretatie I, schrijven we

$$(M, I) \models f$$

Dus we hebben in het voorbeeld de volgende zaken ingezien:

$$\begin{aligned} (\mathbb{N}, <, \text{interpretatie}_1) &\models G_1 \\ (\mathbb{Q}, <, \text{interpretatie}_2) &\models G_1 \end{aligned}$$

2.11. OPGAVE. Ga na dat G_2 waar is in interpretatie$_2$, maar niet in interpretatie$_1$. Ofwel: $(\mathbb{Q}, <, \text{interpretatie}_2) \models G_2$, en $(\mathbb{N}, <, \text{interpretatie}_1) \not\models G_2$

2.12. OPGAVE. Definieer interpretatie$_3$ als volgt.

D	$\mathbb{N}$;
$K(x,y)$	$x = 2y$.

Welke van de formules G_1, G_2 zijn waar?

2.13. Opgave. Definieer interpretatie$_4$ als volgt.

D	$\mathbb{Q}$;
$K(x,y)$	$x = 2y$.

Welke van de formules G_1, G_2 zijn waar?

2.14. Opgave. We bekijken als model de landen van Europa met de volgende interpretatie. Interpretatie

L	verzameling van landen van Europa
n	Nederland
d	Duitsland
i	Ierland
$G(x,y)$	x grenst aan y
$D(x,y,z)$	x, y en z hebben een drielandenpunt met elkaar

1. Formaliseer de zin "Nederland en Duitsland delen een drielandenpunt"
2. Welke van de volgende formules is waar in dit model:
   $G_3 := \forall x{\in}L \exists y{\in}L[G(x,y)]$,
   $G_4 := \forall x,y{\in}L[(\exists z{\in}L\ D(x,y,z)) \to G(x,y)]$,
   $G_5 := \forall x{\in}L[G(i,x) \to \exists y \in L[D(i,x,y)]]$.

2.15. Opgave. Bedenk zelf een model M en een interpretatie I zodat

$$(M,I) \models \forall x \in D \exists y \in E[R(x,y) \wedge \neg R(y,x) \wedge \neg R(y,y)]$$

2.16. Opgave. Formaliseer de volgende zin.

Sharon is knap; er is een man die lekker in zijn vel zit van wie zij houdt

Hierbij wordt gedefinieerd dat iemand lekker in zijn vel zit als hij of zij van zich zelf houdt.

2.17. Opgave. Formaliseer de volgende zinnen.

Voor iedere x en y geldt: x houdt van y alleen als x lekker in zijn vel zit.

Voor iedere x en y geldt: x houdt van alle mensen y die lekker in hun vel zitten.

Voor iedere x en y geldt: x houdt precies dan van y als y lekker in zijn of haar vel zit.

Er is iemand die van alle mensen houdt.

2.3. De taal van de predicatenlogica met gelijkheid

Soms wordt een formalisering van het volgende gevraagd.

Sharon is knap; er is een man die alleen aan haar aandacht geeft

Om dit te formaliseren hebben we een *gelijkheidsrelatie* nodig. De formalisering wordt dan als volgt.

$$K(s) \wedge \exists x{\in}M\ (A(x,s) \wedge \forall y{\in}V(A(x,y) \rightarrow y = s))$$

Om dit als een formule van de predicatenlogica te kunnen zien moeten we de taal van de predicatenlogica uitbreiden.

2.18. Definitie. predicatenlogica met gelijkheid wordt gedefinieerd door aan de predicatenlogica standaard een binair predicaat "=" toe te voegen.
De interpretatie van dit predicaat in een model is altijd "gelijk aan".

2.19. Opgave. Formaliseer de volgende zinnen in de predicatenlogica met gelijkheid.

Iedereen heeft precies één moeder.

Iedereen heeft precies twee oma's.

Iedere getrouwde man heeft precies één echtgenote.

2.20. Opgave. Gegeven de interpretatie

M	domein van de mensen;
$V(x)$	x is vrouw;
$O(x,y)$	x is ouder van y.

Formaliseer de volgende eigenschappen.

- $S(x,y)$: x en y hebben samen een kind gemaakt.
- $B(x,y)$: x is broer van y (pas op: zie ook volgende item).
- $H(x,y)$: x is halfzus van y.

Vertaal terug naar de omgangstaal.

- $\exists x{\in}M \forall y{\in}M\ O(x,y)$. Geldt dit?
- $\forall z_1{\in}M \forall z_2{\in}M[(\exists x{\in}M \exists y_1{\in}M \exists y_2{\in}M\ O(x,y_1) \wedge O(y_1,z_1) \wedge O(x,y_2) \wedge O(y_2,z_2)) \rightarrow \neg(\exists w{\in}M(O(z_1,w) \wedge O(z_2,w)))$.] Geldt dit?

2.21. Opgave. Gegeven de interpretatie

$\mathbb{N}$	domein van de natuurlijke getallen;
$P(x,y,z)$	$x + y = z$;
$M(x,y,z)$	$x.y = z$.

Formaliseer

- $x < y$.
- $x \mid y$ (x is deler van y).
- x is priemgetal.

2.4. Waarheid en gevolgtrekking

Laat A een formule van de predicatenlogica (met of zonder gelijkheid) zijn. A heet waar in een model M onder een interpretatie (woordenboek) I als na vertaling de formule klopt, notatie: $(M, I) \models A$.

2.22. DEFINITIE. A heet *waar* zonder meer, notatie $\models A$, als voor *ieder* model en interpretatie de vertaling klopt.

2.23. VOORBEELD. (i) $\models \forall x{\in}D\ (P(x) \to P(x))$.
(ii) $\models [\exists x{\in}D \forall y{\in}D\ P(x,y)] \to [\forall y{\in}D \exists x{\in}D\ P(x,y)]$.
(iii) $\not\models [\forall y{\in}D \exists x{\in}D\ P(x,y)] \to [\exists x{\in}D \forall y{\in}D\ P(x,y)]$.

Dat formule in (iii) niet waar is zien we door de volgende interpretatie te nemen. $D := \mathbb{N}$ en $P(x,y) := x{>}y$. Onder deze interpretatie geldt $\forall y{\in}D \exists x{\in}D\ P(x,y)$ (voor ieder getal $y{\in}\mathbb{N}$ is er een groter getal $x{\in}\mathbb{N}$), maar niet $\exists x{\in}D \forall y{\in}D\ P(x,y)$ (want dan zou er in $\mathbb{N}$ een grootste getal moeten bestaan en dat is niet zo).

2.24. DEFINITIE. Laten A en B formules uit de predicatenlogica zijn. Dan zeggen we dat B uit A volgt, notatie $A \models B$, als $\models A \to B$. Dat houdt in dat in iedere situatie waarin A geldt, ook B geldt.

Uit (ii) in het vorige voorbeeld volgt dat $\forall y{\in}D \exists x{\in}D\ P(x,y)$ uit $\exists x{\in}D \forall y{\in}D\ P(x,y)$ volgt.

2.5. Waarheid als spel met twee spelers

Gegeven een formule met de kwantoren $(\forall, \exists)$ voorop. Als voorbeeld nemen we de formule

$$\forall x, y{\in}D \exists z{\in}D\ [K(x,y) \to (K(x,z) \wedge K(z,y))] \qquad (+)$$

en de interpretatie $D := \mathbb{Q}$, $K(x,y) := (x{<}y)$. Eén speler speelt voor de advocaat en de andere speler speelt voor de aanklager. De aanklager mag bij ieder (rijtje van) $\forall$ kwantor(en) een element van het domein keizen; de advocaat bij iedere $\exists$. Dit gebeurt in de volgorde waarin ze voorkomen (van links naar rechts). Op het laatst houd je een formule over zonder kwantoren en met de variabelen vervangen door namen.

2.25. VOORBEELD. De aanklager kiest $x := 3$ en $y := 4$. Omdat de advocaat nu $z := 3\frac{1}{2}$ kan kiezen en het klopt dat $3 < 4 \to (3 < 3\frac{1}{2} \wedge 3\frac{1}{2} < 4)$ wint de advocaat en wordt daarmee de formule waar onder de interpretatie.

Onder de interpretatie $D := \mathbb{N}$, $K(x,y) := (x{<}y)$. is de formule $(+)$ niet waar. We zien dat de aanklager wint als hij deze keer weer $x := 3$ en $y := 4$ kiest. Want wat de advocaat ook zal kiezen voor z, dat getal zal niet tussen 3 en 4 kunnen liggen.

Wat gebeurt er als de aanklager $x := 4$ en $y := 3$ kiest? Dan heeft de advocaat een makkie: $4{<}3$ is onwaar en daar volgt alles uit, zodat hij voor z bijvoorbeeld 200 kan kiezen:

$$4{<}3 \to (4{<}200 \wedge 200{<}3).$$

Dus een slimme aanklager kiest x en y zo dat $x{<}y$ gaat kloppen; alleen dan wordt de advocaat in het nauw gedreven.

2.26. Stelling. Laat de formule $A =$

$$\forall x, y{\in}D \exists x_1, y_1{\in}D \ldots B(x, y, x_1, y_1, \ldots).$$

gegeven zijn. Dan is A waar (in een bepaalde interpretatie of algemeen) als bij iedere keuze van de aanklager (voor de $\forall$ variabelen) de advocaat zodanig kan kiezen (voor de $\exists$ variabelen) dat de uiteindelijke formule B zonder kwantoren klopt. Als de aanklager slim kan kiezen zodat P uiteindelijk niet klopt dan is A onwaar (in de gegeven interpretatie of algemeen).

2.27. Opgave. Ga na of de volgende uitspraken waar zijn of niet. [Hint. Gebruik de spelthe-oretische methode.]

(i) $\exists x{\in}M \forall y{\in}M O(x, y)$.
(ii) $\forall x{\in}D \exists y{\in}D \neg(x = y)$.
(iii) $\exists x, y{\in}D \forall z{\in}D \exists w{\in}D[x \neq y \rightarrow z \neq w]$.

2.6. Waarheid en de correctheid van een ontwerp

Als we een product ontwerpen doen we dit vaak door het samen te stellen uit kleinere componenten waarvan we de werking al kennen. Dit zien we bijvoorbeeld bij een stereo installatie, die wordt samengesteld uit een versterker, een CD speler, een tuner, boxen en kabels. Deze componenten zelf zijn ook weer samengesteld uit kleinere componenten: transistoren, bedrading, het aandrijfmotortje van de CD speler etc.

Als we een ontwerp maken willen we graag het volgende weten

- Is het ontwerp op hoog niveau correct? Ofwel: is het waar dat het apparaat "het" doet, onder aanname van de goede werking van de componenten?
- Wat doet het apparaat dan precies? Ofwel: wat bedoelen we eigenlijk met het "correct" zijn van het ontwerp?
- Wat worden de componenten verondersteld precies te doen?

We illustreren dit aan de hand van de stereo installatie, waarbij we logica (propositielogica en predicatenlogica) gebruiken. De algemene werkwijze is als volgt.

Bij een *ontwerp* hoort een *formule*
Een ontwerp is *correct* als de bijbehorende formuled *waar (bewijsbaar)* is.

Wat is de formule die we willen bewijzen? Uiteindelijk willen we bewijzen **CD in speler** $\wedge$ **Start** $\rightarrow$ **Muziek uit box** want dit is de formule waar het ontwerp aan moet voldoen. Dit noemen we ook wel de **specificatie** van het product dat we willen maken. Deze formule kunnen we uiteraard alleen bewijzen als we allerlei aannames doen over de componenten, de versterker, de CD-speler, de boxen en de kabels.

De aannames over de componenten formuleren we ook als specificaties, maar nu zijn dat de specificaties waar de componenten aan moeten voldoen. We houden het in eerste instantie simpel en kijken straks verder wat we misschien nog nodig hebben.

CD speler	**CD in speler** $\wedge$ **Start** $\rightarrow$ **Signaal uit speler**
kabel-1	**Signaal uit speler** $\rightarrow$ **Signaal in versterker**
Versterker	**Signaal in versterker** $\rightarrow$ **Signaal uit versterker**
kabel-2	**Signaal uit versterker** $\rightarrow$ **Signaal in box**
Box	**Signaal in box** $\rightarrow$ **Muziek uit box**

Om de correctheid van het ontwerp te bewijzen moeten we nu de volgende formule aantonen.

$F :=$**CD speler** $\wedge$ **kabel-1** $\wedge$ **Versterker** $\wedge$ **kabel-2** $\wedge$ **Box** $\rightarrow$ (**CD in speler** $\wedge$ **Start** $\rightarrow$ **Muziek uit box**)

We zien dat een correctheidsformule er in het algemeen zo uit ziet:

$$(a_1 \rightarrow c_1) \wedge \ldots \wedge (a_n \rightarrow c_n) \rightarrow (A \rightarrow C)$$

waarbij $A \rightarrow C$ de hoog niveau specificatie is van het product en de $a_i \rightarrow c_i$ de specificaties van de componenten zijn. Feitelijk is dit een versimpelde situatie, zoals we straks zullen zien. In het algemeen hebben we predicatenlogica nodig om een echte specificatie op te schrijven.

Wat we nu willen is het volgende

1. Nagaan of de formule F waar (bewijsbaar) is.
2. Nagaan of de componenten wel echt aan de specificaties voldoen.
3. Nagaan of de specificatie eigenlijk wel genoeg zegt.

(1) De formule F is waar. Dit is eenvoudig na te gaan met een waarheidstabel, maar ook gewoon door middel van een bewijs (redenering): van de verschillende componenten wordt de aanname (a_i) steeds waargemaakt door de conclusie (c_j) van een eerdere component of door de aanname in de globale specificatie (A). Waarheid van F impliceert dat, als het apparaat niet werkt, dat alleen kan komen doordat één van de componenten niet aan zijn specificatie voldoet.

(2) Voldoen de componenten aan de gegeven specificaties? Als ze niet voldoen kunnen er twee dingen aan de hand zijn: (i) de component is stuk en moet vervangen worden (ii) we hebben iets geëist dat te sterk is. Omdat we de specificatie zo expliciet gemaakt hebben kunnen we kijken of de eisen die we aan de componenten stellen wel kloppen.

Bij de kabels is duidelijk dat, als ze niet aan de specificatie voldoen, we de kabel moeten vervangen. Maar bij de versterker hebben we onze eisen wel erg "hoog" gesteld: dit is niet wat een versterker doet, want om een signaal naar de boxen te krijgen moet je ook nog de volume-knop hoog genoeg zetten en je moet het apparaat ook aan zetten. Dat laatste geldt overigens ook voor de CD speler. Dus we krijgen een verfijning van de specificaties die noodzakelijkerwijs ook leidt tot een verfijning van de specificatie van het hele ontwerp (want anders is hij niet meer bewijsbaar).

CD speler	**Speler aan** $\wedge$ **CD in speler** $\wedge$ **Start** $\rightarrow$ **Signaal uit speler**
kabel-1	**Signaal uit speler** $\rightarrow$ **Signaal in versterker**
Versterker	**Versterker aan** $\wedge$ **Signaal in versterker** $\wedge$ **Volume hoog** $\rightarrow$ **Signaal uit versterker**
kabel-2	**Signaal uit versterker** $\rightarrow$ **Signaal in box**
Box	**Signaal in box** $\rightarrow$ **Muziek uit box**

$$F' := \textbf{Versterker aan} \wedge \textbf{Volume hoog} \wedge \textbf{Speler aan} \wedge \textbf{CD in speler} \wedge \textbf{Start} \rightarrow \textbf{Muziek uit box}$$

(3) Zegt deze specificatie wel genoeg? Hij zegt niet dat de muziek die uit de box komt ook daadwerkelijk van de CD is! Er zijn vele producten mogelijk die aan deze specificatie voldoen, maar waar we toch niet tevreden mee zijn (bijvoorbeeld een versterker die continu één lied herhaalt, zonder acht te slaan op het signaal uit de CD speler). Om de specificatie nog preciezer te maken hebben we predicatenlogica nodig.

CD speler	$\forall x \in \mathrm{CD}($**Speler aan**$\wedge\ x$ **in speler** $\wedge$ **Start**
	$\rightarrow$ **Signaal**(x) **uit speler**$)$
kabel-1	$\forall x \in \mathrm{CD}($**Signaal**$(x)$ **uit speler** $\rightarrow$ **Signaal**(x) **in versterker**$)$
Versterker	$\forall x \in \mathrm{CD}($**Versterker aan**$\wedge$**Signaal**$(x)$ **in versterker**$\wedge$
	Volume hoog $\wedge$**Versterker op CD** $\rightarrow$ **Signaal**(x) **uit versterker**$)$
kabel-2	$\forall x \in \mathrm{CD}($**Signaal**$(x)$ **uit versterker** $\rightarrow$ **Signaal**(x) **in box**$)$
Box	$\forall x \in \mathrm{CD}($**Signaal**$(x)$ **in box** $\rightarrow$ **Muziek**(x) **uit box**$)$

De hoog-niveau specificatie wordt nu als volgt:

$$\begin{aligned} F'' \ := \ & \forall x \in \mathrm{CD}(\textbf{Versterker aan} \wedge \textbf{Volume hoog} \wedge \textbf{Speler aan} \wedge \\ & x \textbf{ in speler} \wedge \textbf{Start} \rightarrow \textbf{Muziek}(x) \textbf{ uit box}) \end{aligned}$$

2.28. Opgave. 1. Ga na of de correctheidsformule die bij F' hoort waar is.

2. Ga na of de correctheidsformule die bij F'' hoort waar is.

3. Talen

Het genereren en beschrijven van talen is een belangrijke toepassing van computers. Een computer kan alleen goed overweg met talen die een precieze "formele" definitie hebben, zoals bijvoorbeeld een programmeertaal. Maar ook kan men proberen een natuurlijke taal aan de computer te leren door de regels van de natuurlijke taal precies (formeel) te definiëren. Voor nu houden we ons alleen bezig met formeel gedefinieerde talen. Een *taal* vatten we op als een verzameling *woorden*. Een *woord* is een string symbolen uit een van tevoren vastgelegd *alfabet*.

Met deze definities kunnen we al een heel aantal interessante vragen over talen precies maken en beantwoorden, zoals

- Zit het woord w in de taal L?
- Zijn de talen L en L' gelijk?

Allerlei informatica-problemen die op het eerste gezicht misschien niets met talen te maken hebben kunnen als een taalprobleem beschouwd worden. Met behulp van (formele) talen kunnen puzzels en belangrijkere zaken opgelost worden.

3.1. Alfabet, woord, taal

3.1. Definitie. (i) Een *alfabet* is een verzameling Σ van symbolen.
(ii) Een *woord* over Σ is een eindig rijtje symbolen uit dit alfabet.
(iii) λ is het woord dat bestaat uit 0 symbolen, ook wel genaamd het *lege woord*.
(iv) Σ^* is de verzameling van alle woorden over Σ.
(v) Een *taal* L over Σ is een deelverzameling van Σ^*.

3.2. Voorbeeld. (i) $\Sigma = \{a, b\}$ is een alfabet.
(ii) *abba* zit in Σ^* (notatie: $abba \in \Sigma^*$).
(iii) $abracadabra \notin \Sigma^*$.
(iv) $abracadabra \in \Sigma_0^*$, met $\Sigma_0 = \{a, b, c, d, r\}$.
(v) λ het *lege woord* zit in Σ^* voor iedere Σ.

We kunnen talen op allerlei manieren beschrijven. Twee manieren die we in dit college zullen tegenkomen zijn *reguliere expressies* en *grammatica's*. Maar we kunnen talen ook gewoon als verzamelingen beschrijven. We introduceren even wat handige notatie.

3.3. Notatie. We schrijven a^n voor $a \ldots a$ met n keer een a. Preciezer gezegd: $a^0 = \lambda$ en $a^{n+1} = a^n a$.
De *lengte* van een woord w geven we weer als $|w|$.
Als w een woord is, dan is w^R het *omgekeerde* ("reverse") woord, dus bijvoorbeeld $(abaabb)^R = bbaaba$.

3.4. Voorbeeld. 1. $L_1 := \{w \in \{a, b\}^* \mid w \text{ bevat een even aantal } a\text{'s}\}$.

2. $L_2 := \{a^n b^n \mid n \in \mathbb{N}\}$.

3. $L_3 := \{wcv \in \{a, b, c\}^* \mid w \text{ bevat geen } b,\ v \text{ bevat geen } a \text{ en } |w| = |v|\}$.

4. $L_4 := \{w \in \{a,b\}^* \mid w = w^R\}$.

Bedenk bij ieder tweetal talen uit dit voorbeeld een woord w dat wel in de ene, maar niet in de andere taal zit.

Als we twee talen L_1 en L_2 hebben kunnen we met de bekende verzamelingsoperaties nieuwe talen definiëren. We brengen wat notatie in herinnering.

3.5. NOTATIE. Laat L en L' talen over het alfabet Σ zijn.

1. $\overline{L}$ is het *complement* van L: de taal bestaande uit de woorden w die *niet* in L zitten. (Dus de $w \in \Sigma^*$ waarvoor geldt $w \notin L$.)

2. $L \cap L'$ is de *doorsnijding* van L en L': de taal bestaande uit de woorden w die zowel in L als in L' zitten.

3. $L \cup L'$ is de *vereniging* van L en L': de taal bestaande uit de woorden w die in L of in L' zitten.

3.6. OPGAVE. 1. Beschrijf $L_1 \cap L_2$.

2. Beschrijf $L_2 \cap L_4$.

3. Beschrijf $L_3 \cap L_4$.

Er zijn ook operaties die specifiek voor talen zijn. Die definiëren we nu.

3.7. NOTATIE. Laat L en L' talen over het alfabet Σ zijn.

1. LL' is de *concatenatie* van L en L': de taal bestaande uit de woorden van de vorm wv met $w \in L$ en $v \in L'$,

2. L^R is de taal van *omgekeerde* woorden van L en bestaat uit de woorden w^R met $w \in L$,

3. L^* is de taal bestaande uit *eindige concatenaties* van woorden uit L en bestaat uit de woorden van de vorm $w_1 w_2 \dots w_k$ met $k \geq 0$ en $w_1, w_2, \dots, w_k \in L$.

De taal L^* heet ook wel de *Kleene afsluiting* van L. De taal L^* bestaat uit concatenaties van 0 of meer woorden uit L, dus het is altijd zo dat $\lambda \in L^*$. Verder is het zo dat $L \subset L^*$ voor iedere taal L. Ga voor jezelf na wat L^* is als $L = \emptyset$ of als $L = \{\lambda\}$.

3.8. OPGAVE. (Zie de definities van L_1 en L_2 in voorbeeld 3.4.)

1. Bewijs dat $L_1 = L_1^*$

2. Geldt $L_2 L_2 = L_2$? Geef een bewijs of een tegenvoorbeeld.

3. Geldt $\overline{L_1} = \overline{L_1}^*$? Geef een bewijs of een tegenvoorbeeld.

4. Voor welke talen uit 3.4 geldt $L = L^R$? (Alleen antwoord, geen bewijs.)

3.2. Reguliere Talen

Een populaire manier om talen te beschrijven is door middel van *reguliere expressies*. Een taal die door een reguliere expressie beschreven kan worden noemen we een *reguliere taal*. In de informatica worden reguliere talen gezien als (relatief) eenvoudig: als l een reguliere taal is, dan is het niet moeilijk om een programmaatje te schrijven dat bepaalt of een gegeven woord w in L zit. Zo'n programma heet een *parser* en het oplossen van de vraag "$w \in L$" heet ook wel *parseren*. Parsers voor reguliere talen zijn dus eenvoudig te maken: er zijn zelfs (vele) programma's die parsers voor je genereren (als je er een reguliere expressie in stopt)! De door dit soort "parser generators" gegenereerde parsers zijn zelfs bijzonder *efficiënt*: ze beslissen heel snel of w al dan niet in L zit. We gaan het in dit college verder niet hebben over parseren, maar het zal duidelijk zijn dat reguliere talen een belangrijke klasse van talen vormen. Dus die gaan we nader bekijken.

3.9. Definitie. Laat Σ een alfabet zijn. De *reguliere expressies* over Σ zijn als volgt gedefinieerd.

1. $\emptyset$ is een reguliere expressie,
2. λ is een reguliere expressie,
3. x is een reguliere expressie voor iedere $x \in \Sigma$,
4. als r_1 en r_2 reguliere expressies zijn, dan zijn $(r_1 \cup r_2)$ en $(r_1\, r_2)$ ook reguliere expressies,
5. als r een reguliere expressie is, dan is r^* ook een reguliere expressie.

Dus bijvoorbeeld aba, $ab^*(a \cup \lambda)$, $(a \cup b)^*$, $(a \cup \emptyset)b^*$ en $(ab^* \cup b^*a)^*$ zijn reguliere expressies. Een reguliere expressie beschrijft een taal op de volgende manier.

3.10. Definitie. Bij iedere reguliere expressie r definiëren we de *taal van* r, $\mathcal{L}(r)$ als volgt.

1. $\mathcal{L}(\emptyset) := \emptyset$,
2. $\mathcal{L}(\lambda) := \{\lambda\}$,
3. $\mathcal{L}(x) := \{x\}$ voor iedere $x \in \Sigma$,
4. $\mathcal{L}(r_1 \cup r_2) := \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$,
5. $\mathcal{L}(r_1\, r_2) := \mathcal{L}(r_1)\mathcal{L}(r_2)$,
6. $\mathcal{L}(r^*) := \mathcal{L}(r)^*$.

Als we nagaan wat dat betekent voor de reguliere expressies die we boven als voorbeeld gaven, dan zien we:

- $\mathcal{L}(aba) = \{aba\}$,
- $\mathcal{L}(ab^*(a \cup \lambda)) = \{a\}\{b\}^*\{a, \lambda\}$ en bestaat uit de woorden die beginnen met een a, dan een willekeurig aantal b's en dan niets of weer een a,
- $\mathcal{L}((a \cup b)^*) = \{a, b\}^*$, dus dat zijn alle woorden over het alfabet $\{a, b\}$,

- $\mathcal{L}((a \cup \emptyset)b^*) = \{a\}\{b\}^*$, en dat is dus dezelfde taal als van ab^*,
- $\mathcal{L}((ab^* \cup b^*a)^*) = (\{a\}\{b\}^* \cup \{b\}^*\{a\})^*$. Deze taal lijkt wat moeilijker in woorden te beschrijven.

In reguliere expressies wordt soms ook de operator + gebruikt: de reguliere expressie a^+ staat dan voor "1 of meer keer een a". We hebben de + echter niet nodig, want in plaats van r^+ kunnen we gewoon rr^* schrijven, want dat betekent precies hetzelfde (eerst een r en dan nog 0 of meer keer een r).

3.11. Opgave. 1. Laat zien dat we de operator ?, waarbij $a^?$ staat voor 0 of 1 keer a niet hoeven toe te voegen aan de reguliere expressies, omdat we hem al kunnen maken.

2. Wat is $\mathcal{L}(\emptyset ab^*)$?

3. Geef een reguliere expressie die de taal $L_5 := \{w \in \{a,b\}^* \mid w \text{ bevat minstens één } a\}$ beschrijft.

4. Geef een reguliere expressie die de taal L_1 uit Voorbeeld 3.4 beschrijft.

5. Laat zien dat $\mathcal{L}(ab(ab)^*) = \mathcal{L}(a(ba)^*b)$.

We zien dat de reguliere expressie $\emptyset$ niet zo nuttig is: in plaats van $r \cup \emptyset$ kunnen we net zo goed r schrijven en $\emptyset r$ beschrijft $\emptyset$. De reguliere expressie $\emptyset$ wordt alleen gebruik om de lege taal $\emptyset$ te kunnen beschrijven en we zullen hem verder niet tegenkomen.

3.12. Definitie. Laat Σ een alfabet zijn. We noemen een taal L over Σ *regulier* als er een reguliere expressie is die hem beschrijft. (Ofwel: als er een reguliere expressie r is zodat $\mathcal{L}(r) = L$.)

De taal L_1 uit Voorbeeld 3.4 is regulier, zoals we in Opgave 3.11 gezien hebben. De talen L_2, L_3 en L_4 uit Voorbeeld 3.4 zijn niet regulier. Het bewijs van de niet-regulariteit van deze talen voert te ver voor dit college.

3.13. Opgave. Laat zien dat de volgende talen regulier zijn.

1. $L_6 := \{w \in \{a,b\}^* \mid \text{iedere } a \text{ in } w \text{ wordt direct gevolgd door een } b\}$,
2. $L_7 :=$ de taal van alle goedgevormde integer-expressies. Deze bestaan uit de symbolen $0,1,2,3,4,5,6,7,8,9,$ maar ze beginnen nooit met een 0 en mogen eventueel worden voorafgegaan door een + of een −.
3. $L_8 :=$ de taal van alle goedgevormde rekenkundige expressies zonder haakjes. Deze bestaan uit natuurlijke getallen met daartussen de operatoren +, − of ×, bijvoorbeeld $7+3\times 29-78$. (Hint: maak eerst een reg. expr. voor de natuurlijke getallen; zo'n getal begint nooit met een 0.)

Als een taal L regulier is dan is de taal L^R ook regulier, want als L wordt beschreven door de reguliere expressie e ($L = \mathcal{L}(e)$) wordt L^R beschreven door de reguliere expressie e^R ($L^R = \mathcal{L}(e^R)$). Er zijn nog meer mooie eigenschappen van de klasse van reguliere talen: Als L_1 en L_2 regulier zijn, dan zijn $L_1 \cup L_2$ en $L_1 \cap L_2$ ook regulier en $\overline{L_1}$ is ook regulier. Van

$L_1 \cup L_2$ is dat eenvoudig in te zien (ga het voor jezelf na!), maar van $L_1 \cap L_2$ en $\overline{L_1}$ niet. Dat komt aan de orde in het onderdeel Automaten.

Men identificeert soms een reguliere taal met een reguliere expressie die hem beschrijft. Men heeft het dan bijvoorbeeld over "de taal $b^*(aab)^*$", terwijl de taal $\mathcal{L}(b^*(aab)^*)$ bedoeld wordt. We zullen dat in dit college niet doen.

3.14. Opgave. Welke van de volgende reguliere expressies beschrijven dezelfde taal? Toon het aan of geef een string die wel in de ene en niet in de andere taal zit.

- $b^*(aab)^*$.
- $b^*(baa)^*b$.
- $bb^*(aab)^*$.

3.3. Contextvrije Grammatica's

Er is ook een andere manier om talen te definiëren. Daarbij proberen we niet de taal in één keer te *beschrijven* maar we geven een voorschrift hoe de woorden uit de taal *gegenereerd* (of *geproduceerd*) kunnen worden. Zo'n voorschrift heet een "grammatica".

3.15. Voorbeeld. $\Sigma = \{a, b\}$. De taal $L_9 \subseteq \Sigma^*$ wordt gedefinieerd door *producties* vanuit S (start).

$$\begin{aligned} S &\rightarrow aAb \\ A &\rightarrow aAb \\ A &\rightarrow \lambda \end{aligned}$$

De manier om nu woorden te genereren is als volgt. We beginnen bij S (start). Dit is het *startsymbool*. S en A zijn de *hulpsymbolen*. We volgen de pijl (één mogelijkheid). Indien er nog een hulpsymbool staat volgen we nog een pijl. Totdat er geen hulpsymbool meer staat en dan hebben we een woord geproduceerd. Voorbeelden van producties:

$$\begin{aligned} &S \rightarrow aAb \rightarrow aaAbb \rightarrow aabb; \\ &S \rightarrow aAb \rightarrow aaAbb \rightarrow aaaAbbb \rightarrow aaabbb. \end{aligned}$$

Deze manier van weergeven van een taal noemen we een *grammatica*. Bovenstaande grammatica wordt ook wel weergegeven als

$$\begin{aligned} S &\rightarrow aAb \\ A &\rightarrow aAb \mid \lambda \end{aligned}$$

We bedoelen dan dat er vanuit A twee producties mogelijk zijn: $A \rightarrow aAb$ en $A \rightarrow \lambda$.

Dit levert op als taal $L_9 = \{ab, aabb, aaabbb, a^4b^4, \ldots, a^nb^n, \ldots\}$. Anders opgeschreven:

$$L_9 = \{a^nb^n \mid n \in \mathbb{N} \text{ en } n > 0\}$$

3.16. Definitie. Een *contextvrije grammatica* G is een drietal $< \Sigma, V, R >$ bestaande uit
(i) Een alfabet Σ

(ii) Een verzameling *hulpsymbolen* V, waarvan er één speciaal is, S, het *startsymbool*.
(iii) Een verzameling *productieregels* R van de vorm

$$X \to w$$

waarbij X een hulpsymbool is en w een woord bestaande uit letters uit het alfabet en hulpsymbolen. (Anders gezegd: $w \in (\Sigma \cup V)^*$.)

We zullen voor de hulpsymbolen altijd hoofdletters gebruiken, dus S, A, B etc en voor de letters van het alfabet kleine letters (a, b, c etc.)

3.17. VOORBEELD. (i) De taal L_9 uit 3.15 wordt geproduceerd door een contextvrije grammatica.
(ii) De taal L_{10} met $\Sigma = \{a\}$, $V = \{S\}$ en productieregels $S \to aaS \mid a$. Deze grammatica produceert alle woorden bestaande uit een oneven aantal a'tjes.
(iii) De taal L_{11} met $\Sigma = \{a, b\}$, $V = \{S, A, B\}$ en productieregels $S \to AB$, $A \to Aa \mid \lambda$, $B \to Bb \mid \lambda$. Deze grammatica produceert alle woorden bestaande uit eerst een rij a'tjes en dan een rij b'tjes.

3.18. DEFINITIE. Talen geproduceerd door contextvrije grammatica's heten *contextvrije talen*. Voor de taal die wordt geproduceerd door de grammatica G schrijven we ook we $\mathcal{L}(G)$.

Contextvrije talen worden systematisch behandeld in het college T1a voor de studierichting Informatica.

3.19. VOORBEELD. De taal van goedgevormde rekenkundige expressies met haakjes is contextvrij. (Deze taal is *niet* regulier!) Een grammatica voor deze taal is de volgende.

$$\begin{array}{lcl} S & \to & B\,S\,O\,S\,E \mid G \\ B & \to & (\\ E & \to &) \\ O & \to & + \mid \times \mid - \\ G & \to & P\,C \\ P & \to & 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\ C & \to & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid \lambda \end{array}$$

Laat eens zien hoe je $(33 + (20 * 5))$ maakt en $((33 + 20) * 5)$.

3.20. OPGAVE.
1. Laat zien dat de taal van *gebalanceerde haakjesexpressies* contextvrij is. Dat zijn expressies over $\{(,)\}$ waarbij ieder openingshaakje altijd een sluithaakje heeft, dus bijv. $(()(()))$ en $(())()$ zijn gebalanceerd, maar $(()())))$ niet.
2. Laat zien dat L_1 uit Voorbeeld 3.4 contextvrij is.
3. Laat zien dat L_2 uit Voorbeeld 3.4 contextvrij is. (Kijk naar Voorbeeld 3.15.)
4. Laat zien dat L_3 uit Voorbeeld 3.4 contextvrij is.
5. Laat zien dat L_4 uit Voorbeeld 3.4 contextvrij is.

3.21. Opgave. Gegeven is de volgende grammatica voor de taal L_{12}.

$$\begin{aligned} S &\rightarrow aSb \mid A \mid \lambda \\ A &\rightarrow aAbb \mid abb \end{aligned}$$

De hulpsymbolen zijn dit keer S en A en $\Sigma = \{a, b\}$.

1. Geef de productie van abb en van $aabb$.
2. Welke woorden zitten in L_{12}?

Met enige ervaring is het wel in te zien welke woorden in de taal L_{12} zitten. Maar het is niet eenvoudig om ook echt te *bewijzen* dat dit ze ook precies allemaal zijn. Dit kan (meestal) wel, afhankelijk van de moeilijkheid van de grammatica, maar dat voert hier te ver. Een iets eenvoudiger vraag is de volgende.

> Laat zien dat aab niet in L_{12} zit.

Hoe zou je dat aanpakken? Om te laten zien dat een woord *wel* geproduceerd kan worden door een grammatica moet je gewoon op zoek naar een productie, en als die er is vind je die (meestal) wel. Maar hoe laat je zien dat een woord niet geproduceerd kan worden?

In de informatica is daarvoor het begrip *invariant* geïntroduceerd. Een invariant is een eigenschap P die geldt voor alle woorden die geproduceerd kunnen worden door de grammatica G. Dat P geldt voor alle $w \in \mathcal{L}(G)$ bewijs je door te laten zien dat P voor S geldt en dat P *invariant* is onder de productieregels, dwz:

> als P geldt voor een woord v en ik kan v' produceren uit v, dan geldt P ook voor v'.

Als een woord w niet aan P voldoet, kun je concluderen dat w niet geproduceerd kan worden. Dus om te werken met invarianten moet je het volgende doen.

- Een "goede" eigenschap P verzinnen,
- Laten zien dat P invariant is onder de productieregels,
- Laten zien dat w niet aan P voldoet.

En nu de praktijk: we willen aantonen dat $aab \notin L_{12}$. Wat is een goede invariant? We nemen $P(w) :=$ het aantal b's in $w \geq$ het aantal a's in w. Dit is een invariant, want

- P geldt voor S
- Als $P(v)$ en $v \longrightarrow v'$, dan ook $P(v')$. (Ga dit na voor iedere productieregel: of er komt een a en een b bij, of een a en 2 b's, of geen a en geen b.)

Dus: $P(w)$ geldt voor alle $w \in L_{12}$. Conclusie: $aab \notin L_{12}$.

3.22. Opgave. 1. Laat op dezelfde manier als hierboven (mbv een invariant) zien dat $bba \notin L_{12}$.

2. Laat mbv een invariant zien dat *aabbb* niet geproduceerd kan worden mbv de grammatica voor L_3 die je in opgave 3.20 gemaakt hebt.

3. Laat mbv een invariant zien dat *aabbb* niet geproduceerd kan worden mbv de grammatica voor L_4 die je in opgave 3.20 gemaakt hebt.

Contextvrije grammatica's heten *contextvrij* omdat in de productieregels aan de linker kant alleen maar hulpsymbolen mogen staan. (De regel $Sa \to Sab$ is bijvoorbeeld niet toegestaan.) Bij contextvrije grammatica's hoef je, om een productieregel toe te passen nooit naar de *context* (wat er om het hulpsymbool heen staat) te kijken.

3.4. Rechts-lineaire grammatica's

Een bekende beperking van contextvrije grammatica's zijn de *rechts-lineaire* grammatica's.

3.23. Definitie. Een *rechts-lineaire grammatica* is een contextvrije grammatica waarbij de productie regels de volgende vorm hebben

$$\begin{aligned} X &\to wY \\ X &\to w \end{aligned}$$

waarbij X en Y hulpsymbolen zijn en $w \in \Sigma^*$.

Dus in een rechts-lineaire grammatica mogen hulpsymbolen in de rechterkant alleen aan het einde staan.

3.24. Voorbeeld. (i) In Voorbeeld 3.17 is alleen L_{10} een rechts-lineaire grammatica.
(ii) Soms kunnen we bij een context-vrije grammatica een equivalente rechts-lineaire grammatica maken. De volgende rechtslineaire grammatica (over $\Sigma = \{a, b\}$) produceert de taal L_{11} uit voorbeeld 3.17 (iii). $S \to aS \mid B$, $B \to bB \mid \lambda$.

Een diepzinnige stelling zegt dat de klasse van talen die geproduceerd kunnen worden met een rechtslineaire grammatica precies de klasse van reguliere talen is.

3.25. Stelling. Een taal L is regulier dan en slechts dan als er een rechts-lineaire grammatica is die hem beschrijft.
Een reguliere taal is dus vanzelf ook contextvrij.

De stelling bewijzen we niet. Om haar te bewijzen moet je laten zien hoe je bij een reguliere expressie een rechts-lineaire grammatica maakt die dezelfde taal produceert. Omgekeerd moet je bij iedere rechts-lineaire grammatica een reguliere expressie opschrijven die dezelfde taal beschrijft. Het eerste illustreren we met een voorbeeld.

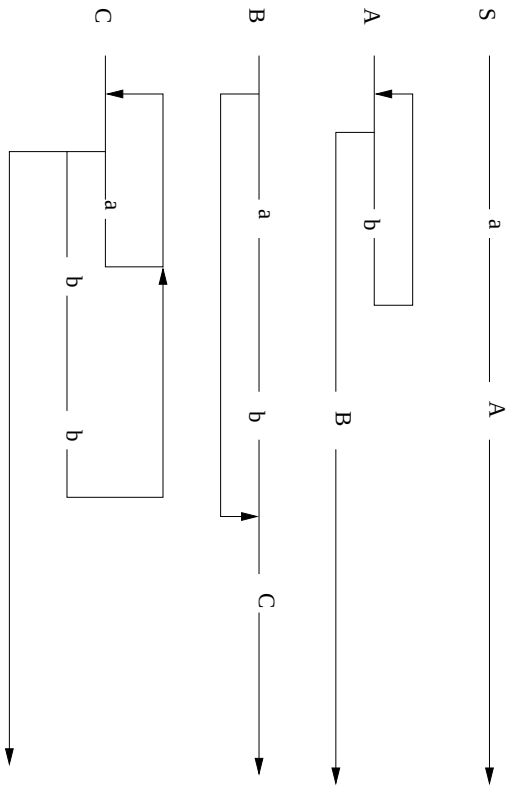
3.26. Voorbeeld. Beschouw de reguliere expressie

$$ab^*(ab \cup \lambda)(a \cup bb)^*$$

Een rechts-lineaire grammatica die dezelfde taal genereert is

$$\begin{aligned} S &\to aA \\ A &\to bA \mid B \\ B &\to abC \mid C \\ C &\to aC \mid bbC \mid \lambda \end{aligned}$$

Soms worden rechts-lineaire grammatica's ook weergegeven met zogenaamde *syntax diagrammen*. We laten daarvan hier alleen een voorbeeld zien. Dit is het diagram dat hoort bij de grammatica uit voorbeeld 3.26.



Een direct gevolg van stelling 3.25 is dat de volgende twee talen (over het alfabet $\{a, b\}$) regulier zijn (zie Voorbeeld 3.17): $L_{10} = \{a^n \mid n \text{ is oneven}\}$ en $L_{11} = \{wv \mid w \text{ bevat alleen } a\text{'s}, v \text{ bevat alleen } b\text{'s}\}$. Probeer voor jezelf een reguliere expressie op te schrijven voor L_{10} en ook voor L_{11}.

3.27. Opgave. (i) Bekijk de volgende grammatica over alfabet $\{a, b, c\}$: $S \to A \mid B$, $A \to abS \mid \lambda$, $B \to bcS \mid \lambda$.
Ga na of je de volgende woorden kunt produceren met deze grammatica: *abab*, *bcabbc*, *abba*. Omschrijf de reguliere taal L_{12} die deze grammatica produceert.

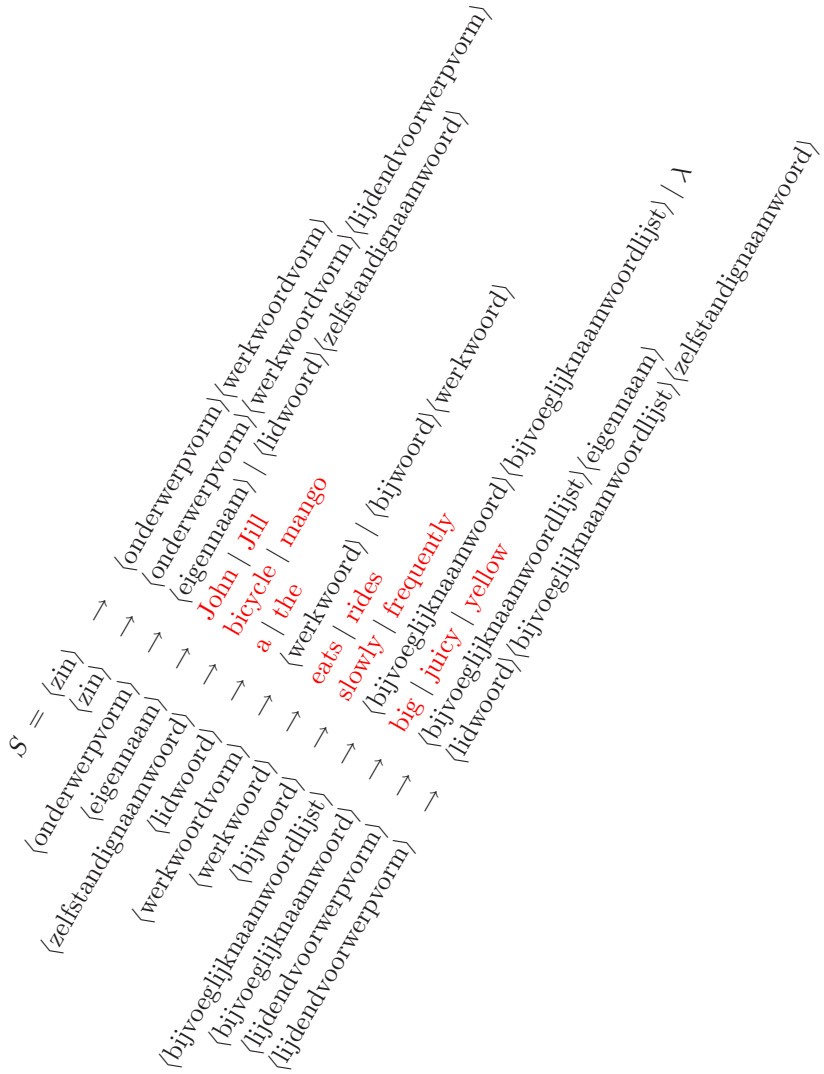
(ii) Maak een rechtslineaire grammatica voor de taal L_{13} bestaande uit woorden van de vorm $ab \ldots aba$ (dus a's en b's om en om met vooraan en achteraan een a; zorg dat je ook a krijgt).

3.28. Opgave. Geef rechts-lineaire grammatica's voor de talen uit opgave 3.13:

1. $L_6 := \{w \in \{a, b\}^* \mid \text{iedere } a \text{ in } w \text{ wordt direct gevolgd door een } b\}$,
2. $L_7 :=$ de taal van alle goedgevormde integer-expressies. Deze bestaan uit de symbolen $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$, maar ze beginnen nooit met een 0 en mogen eventueel worden voorafgegaan door een $+$ of een $-$.

3. L_8 := de taal van alle goedgevormde rekenkundige expressies zonder haakjes. Deze bestaan uit natuurlijke getallen met daartussen de operatoren $+$, $-$ of $\times$, bijvoorbeeld $7+3\times 29-78$. (Hint: maak eerst een reg. expr. voor de natuurlijke getallen; zo'n getal begint nooit met een 0.)

We geven een grammatica voor een klein gedeelte van het Engels.



3.29. Opgave. Is deze grammatica rechts-lineair?

Laat zien hoe je de volgende zin produceert.

Jill frequently eats a big juicy yellow mango.

Maak zelf andere zinnetjes.

3.5. Algemene Grammatica's [Geen examenstof]

We kunnen veel algemenere grammatica's toestaan dan alleen contextvrije of rechts-lineaire. Op die manier kunnen we talen definiëren die niet contextvrij zijn en die worden ook wel bestudeerd, hoewel ze lastiger zijn.

3.30. DEFINITIE. Een *grammatica* G is een drietal $< \Sigma, V, R >$ bestaande uit
(i) Een alfabet Σ
(ii) Een verzameling *hulpsymbolen* V, waarvan er één speciaal is, S, het *startsymbool*.
(iii) Een verzameling *productieregels* R van de vorm

$$v \to w$$

waarbij v en w bestaan uit letters uit het alfabet en hulpsymbolen, met de restrictie dat v minstens één hulpsymbool bevat. (Anders gezegd: $w \in (\Sigma \cup V)^* V (\Sigma \cup V)^*$ en $w \in (\Sigma \cup V)^*$.)

Als voorbeeld beschouwen we de volgende grammatica G, met $\Sigma := \{a, b, c\}$ en $V = \{S, B, T, U\}$.

$$\begin{array}{rcl} S & \to & BT \\ T & \to & abTc \mid U \\ bUc & \to & Ubc \\ aUb & \to & Uab \\ bUa & \to & abU \\ bUb & \to & Ubb \\ aUa & \to & Uaa \\ BU & \to & \lambda \end{array}$$

We laten zien hoe je abc kunt produceren.

$$S \to BT \to BabTc \to BabUc \to BaUbc \to BUabc \to abc.$$

Laat zelf zien hoe je $aabbcc$ produceert.
De taal van G is $\{a^n b^n c^n \mid n \geq 0\}$. Deze taal is *niet* contextvrij. (Probeer maar eens een contextvrije grammatica voor deze taal te maken; het zal je niet lukken en je kunt ook bewijzen dat het onmogelijk is.)
Het is duidelijk uit de regels (dit is een invariant) dat in iedere w die je kunt produceren het aantal a's, het aantal b's en het aantal c's gelijk zijn. Dat ze uiteindelijk ook in de goede volgorde terecht komen kun je (intuïtief) inzien omdat:

- Eerst wordt $B(ab)^n U c^n$ gegenereerd.
- Dan zorgen de regels voor U ervoor dat alle b's achter alle a's komen.
- Dit eindigt in $BUa^n b^n c^n$ en dan naar $a^n b^n c^n$.

4. Combinatoriek

Graphen

Dit hoofdstuk is een korte inleiding in de graphentheorie. Graphen kom je op allerlei plaatsen tegen bijvoorbeeld bij talen, netwerken, datastructuren, elektrische circuits, transport problemen en stroomschema's.

Intuïtief bestaat een graph uit een verzameling P van punten en een verzameling L van lijnen tussen punten. Twee voorbeelden:

Natuurlijk zijn er nu allerlei knagende onzekerheden, zoals: 'Wanneer zijn twee graphen gelijk?', 'Moeten die punten in een plat vlak liggen?', 'Mogen de lijnen elkaar snijden?'. Al je twijfels verdwijnen als sneeuw voor de zon, dankzij de volgende formele definitie:

4.1. Definitie. Een *graph* is een paar $\langle P, L\rangle$ waarbij P een verzameling is, en L een verzameling van uit twee elementen bestaande deelverzamelingen van P. (De elementen van P noemen we 'punten', de elementen van L noemen we 'lijnen'; een lijn noteren we niet als $\{a, b\}$ maar als (a, b). De lijn (a, b) is dus gelijk aan de lijn (b, a).)
De graph G_1 uit ons eerste voorbeeld is dus $\langle P, L\rangle$ met $P = \{1, 2, 3, 4\}$ en $L = \{(1, 4), (2, 3), (2, 4)\}$.
De graph G_2 is het paar $\langle P, L\rangle$ met $P = \{a, b, c, d\}$ en $L = \{(a, c), (a, d), (c, d)\}$

NB. Merk op dat de lijnen in onze graphen geen *richting* hebben: het zijn geen pijlen. Verder is het niet mogelijk dat er tussen twee punten p en q meerdere lijnen zijn. Je zou de definitie van 'graph' kunnen veranderen en dit wel toestaan, maar het blijkt dat we met Definitie 4.1 al genoeg stof tot nadenken hebben en al veel interessante eigenschappen kunnen beschrijven en bestuderen.

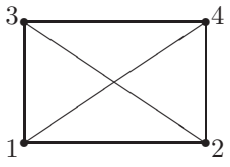
4.2. Definitie. (Hierin is G de graph $\langle P, L\rangle$, en zijn p en q punten)

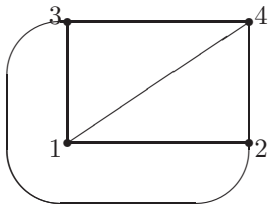
- Een *buur* van p is een punt x met $(p, x) \in L$.
- De *graad* (of: *valentie*) van p is het aantal buren van p.
- Een *pad* van p naar q is een rijtje van verschillende lijnen $(x_0, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n)$ met $n > 0$, $x_0 = p$ en $x_n = q$. Een kortere notatie voor dit pad: $x_0 \to x_1 \to x_2 \to \cdots \to x_n$.
- Met 'G is *samenhangend*' bedoelen we: tussen ieder tweetal punten bestaat een pad.
- Een *component* van G is een zo groot mogelijk samenhangend deel van een graph.
- Een *cykel* is een pad van een punt naar zichzelf.

- Met 'G is *planair*' bedoelen we: je kunt G in het platte vlak zó tekenen dat de (gebogen) lijnen elkaar niet snijden.
- Met 'G is een *boom*' bedoelen we: G is samenhangend en bevat geen cykels.

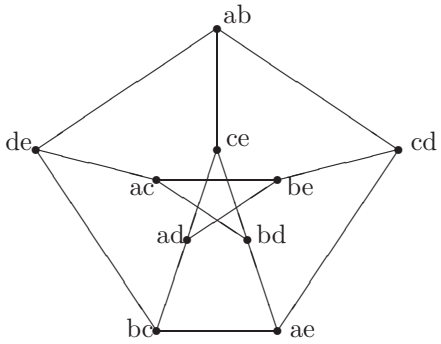
De vraag of een graph planair is, is bijvoorbeeld belangrijk als de graph een elektrisch circuit voorstelt dat we op een chip willen branden.

4.3. VOORBEELD.
- De 'landgraph' is $\langle P, L\rangle$ waarbij P de verzameling van alle landen van de wereld is, en L de relatie 'grenst aan'. De buren van Nederland zijn dan Duitsland en België. De graad van Nederland is 2. Een pad van Nederland naar Spanje is bijvoorbeeld Nederland $\rightarrow$ Duitsland $\rightarrow$ Frankrijk $\rightarrow$ Spanje. De landgraph is niet samenhangend. $\{\text{Engeland}, \text{Schotland}\}$ is een component. Nederland $\rightarrow$ Duitsland $\rightarrow$ België $\rightarrow$ Nederland is een cykel. De landgraph is planair.
- $K_4 = \langle \{1,2,3,4\}\ ,\ \{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\}\rangle$ (de volledige graph met vier punten). Algemeen: K_n is de graph $\langle \{1,\ldots,n\}\,,\, \{(p,q)\,|\,1 \le p < q \le n\}\rangle$. De graph K_4 is planair. Als je dat wilt inzien, moet je niet naar de linker maar naar de rechter tekening van K_4 kijken:





- De Petersen-graph is $\langle P, L\rangle$ met $\begin{cases} P = \{\text{ab}, \text{ac}, \text{ad}, \text{ae}, \text{bc}, \text{bd}, \text{be}, \text{cd}, \text{ce}, \text{de}\} \\ L = \{(p,q)\,|\,p \text{ en } q \text{ hebben geen letter gemeen}\} \end{cases}$



Je kunt laten zien dat de Petersen graph niet planair is.

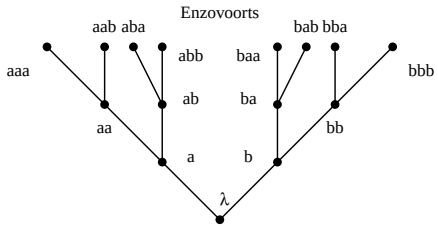
- Bij een taal die gegeven is d.m.v. een inductieve taaldefinitie, kunnen we een graph maken. Bekijk bijvoorbeeld eens de taal die als volgt geproduceerd wordt:

axioma	λ
regel	$x \Rightarrow xa$ $y \Rightarrow yb$

Bij deze taal kunnen we een graph maken door

– eerst de woorden op te schrijven die er in zitten op grond van het axioma (dat zijn de eerste punten van de graph),
– dan stap voor stap woorden (en dus punten) toe te voegen die er op grond van de toepassing van een regel inzitten, en een lijn te maken naar het woord op grond waarvan we dit woord gekregen hebben.

We krijgen dan de volgende graph:

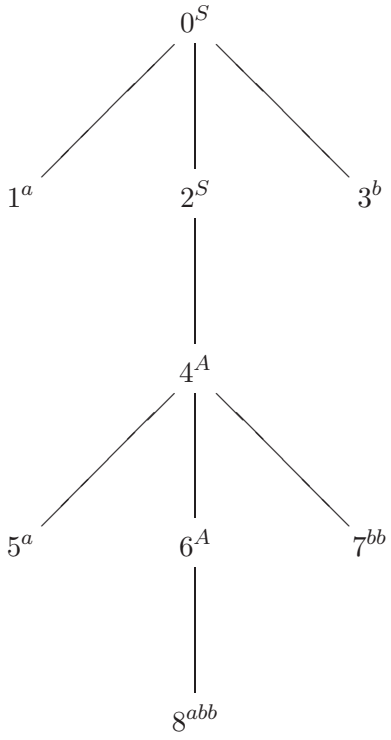


Deze graph is een boom.

- Bij een context-vrije grammatica en een woord dat door deze grammatica geproduceerd wordt, kun je een *parseerboom* maken. Een parseerboom geeft grafisch weer hoe het woord geproduceerd wordt. Bekijk de volgende context-vrije grammatica uit Opgave 3.21.

$$\begin{aligned} S &\rightarrow aSb \mid A \mid \lambda \\ A &\rightarrow aAbb \mid abb \end{aligned}$$

Het woord *aaabbbbb* wordt geproduceerd door deze grammatica, op de volgende manier: $S \rightarrow aSb \rightarrow aAb \rightarrow aaAbbb \rightarrow aaabbbbb$. Hieronder staat de parseerboom van deze productie.

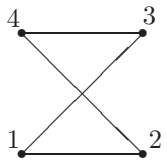


Dus als we $S \to aSb$ doen, geven we het punt met label S drie onderburen met labels a, S en b en als we $A \to aAbb$ doen, geven we het punt met label A drie onderburen met labels a, A en bb. Als je de (labels van de) bladeren van de boom van links naar rechts achter elkaar zet krijg je het woord dat bij deze parseerboom hoort.
NB. Informatici tekenen een boom bijna altijd 'op de kop', dus met de 'wortel' boven en de takken naar beneden gericht. Wiskundigen tekenen een boom altijd met de wortel onder (zie vorige voorbeeld). Bij een parseerboom worden doorgaans alleen de labels opgeschreven bij de punten (en laat men de punten dus ongenummerd).

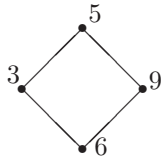
4.4. Definitie (Bijectie). Een afbeelding f van een verzameling A naar een verzameling B heet een *bijectie* als ieder element in B precies één origineel heeft.

4.5. Definitie (Isomorfie van graphen.). We noemen twee graphen $\langle P, L\rangle$ en $\langle P', L'\rangle$ *isomorf* als er een bijectie $\varphi : P \to P'$ bestaat zodat voor alle x, y in P, $(x, y) \in L$ desda $(\varphi(x), \varphi(y)) \in L'$. Anders gezegd: twee graphen zijn isomorf als ze, afgezien van de namen van de punten, hetzelfde zijn.

Bijvoorbeeld:



is isomorf met



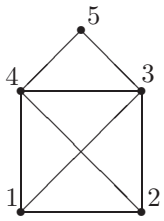
Een isomorfisme φ is:

$$1 \mapsto 6$$
$$2 \mapsto 9$$
$$3 \mapsto 3$$
$$4 \mapsto 5$$

Isomorfe graphen zijn 'hetzelfde' als we alleen geïnteresseerd zijn in graph-theoretische eigenschappen. Bijvoorbeeld: Als G en G' isomorph zijn en G is samenhangend, dan is G' dat ook.

4.6. Definitie (Euler-circuits.). Een *Euler-pad* in een graph $\langle P, L\rangle$ is een pad waarin iedere lijn uit L precies één keer voorkomt. Een *Euler-circuit* of *Euler-cykel* is een cykel waarin iedere lijn uit L precies één keer voorkomt.

Bijvoorbeeld:



$1 \to 4 \to 5 \to 3 \to 1 \to 2 \to 4 \to 3 \to 2$ is een Euler-pad. Omdat punt 1 graad 3 heeft, is er geen Euler-circuit. Dat kun je inzien door te kijken naar het aantal keren dat je bij een cykel door het punt 1 loopt: is dit aantal 1, dan mis je één van de drie bij 1 samenkomende lijntjes, en is aantal 2, dan doorloop je minstens een van deze lijntjes dubbel. Als je deze redenering iets algemener opschrijft, staat er een bewijs van de volgende eenvoudige stelling:

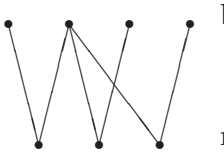
4.7. Stelling (Euler). In een samenhangende graph geldt:

1. Er bestaat een Euler-circuit dan en slechts dan als ieder punt een even graad heeft.
2. Er bestaat een Euler-pad dan en slechts dan als er hoogstens twee punten van oneven graad zijn.

Euler-circuits zijn bijvoorbeeld van belang voor de krantenbezorger of de wijkagent, die iedere straat van zijn wijk precies één keer wil doorlopen en bij zijn uitgangspunt wil terugkeren. Een heel ander probleem heeft de 'travelling salesman', die iedere klant (of iedere stad) precies één keer wil bezoeken en daarna weer naar huis wil. Hij is gebaat bij een 'Hamilton-circuit':

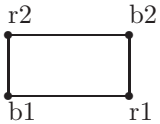
4.8. Definitie (Hamilton-circuits.). Een *Hamilton-pad* in een graph $\langle P, L\rangle$ is een pad waarin je ieder punt van P precies één keer ontmoet. Een *Hamilton-circuit* of *Hamilton cykel* is een cykel waarin ieder punt precies één keer optreedt (afgezien van beginpunt = eindpunt natuurlijk)

4.9. Definitie (Bipartite graphen.). Een graph $\langle P, L\rangle$ heet *bipartite* als P te schrijven is als $P_1 \cup P_2$ zó dat iedere lijn loopt van een punt uit P_1 naar een punt uit P_2. Anders gezegd: je kunt de punten zó blauw en rood kleuren dat iedere lijn een blauw en een rood uiteinde heeft.



blauwe punten

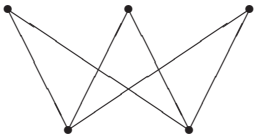
rode punten



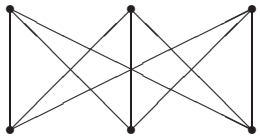
Twee voorbeelden:

De *volledige bipartite graph* met m rode en n blauwe punten, waarbij ieder rood punt met ieder blauw punt verbonden is, wordt $K_{m,n}$ genoemd

$K_{2,3}$



$K_{3,3}$



4.10. Definitie (Kleuring van graphen.). Een *punt-kleuring* van een graph $\langle P, L\rangle$ is een functie $f : P \to \{1, \ldots, n\}$ zodat als (p, q) een lijn is, dan $f(p) \neq f(q)$ (elk punt krijgt één der n kleuren, buren krijgen niet dezelfde kleur).
Het *kleurgetal* (of: *chromatisch getal*) van de graph is de kleinste n waarvoor dit mogelijk is.

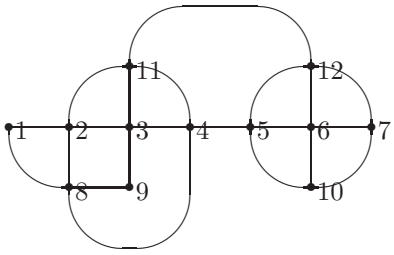
Bipartite graphen zijn dus de graphen met kleurgetal 1 of 2. Een interessante stelling van Appel en Haken, waarvan het in 1975 gevonden bewijs het niveau van dit college helaas ontstijgt:

4.11. Stelling (Vierkleurenstelling.). Het kleurgetal van een planaire graph is hoogstens 4. Anders gezegd: Iedere landkaart kan met hooguit 4 kleuren worden ingekleurd, zodanig dat twee aangrenzende landen een verschillende kleur hebben.

Opgaven

1. Bewijs dat er in een boom van een punt p naar een ander punt q precies één pad bestaat.

2. Hier is een plattegrond G van een dorpje, waarbij de straten aangegeven worden door lijntjes. Op ieder hoekpunt bevindt zich een kroeg. De kroegen zijn aangegeven door punten, genummerd van 1 t/m 12:



   Formuleer de volgende vragen in termen van Hamilton- of Euler-circuits/paden, en beantwoord ze:

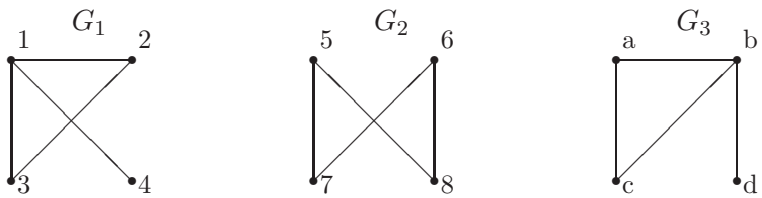
   (a) Is het mogelijk, een wandeling te maken waarbij je iedere straat precies één keer doorloopt?

   (b) Is het mogelijk, een wandeling te maken waarbij je iedere straat precies één keer doorloopt en bovendien begint en eindigt bij kroeg 3?

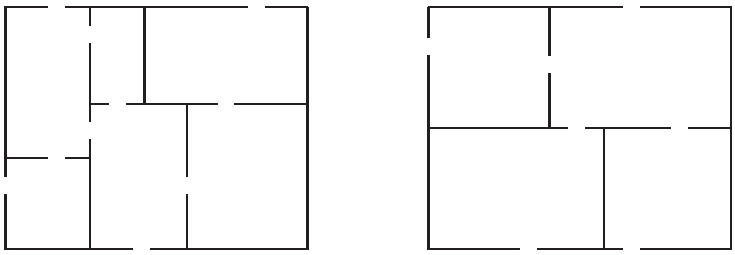
   (c) Bestaat er een kroegentocht waarin iedere kroeg precies één keer voorkomt?

3. Een *brug* in een graph G is een lijn l waarvoor geldt: als je l uit G weglaat, wordt het aantal componenten verhoogd.
   Bewijs dat in een boom iedere lijn een brug is.

4. Ga na welke van de onderstaande graphen isomorf zijn:



5. Hieronder vind je twee plattegronden van huizen.



   (a) Teken bij elke plattegrond de bijbehorende graph, waarbij je de vertrekken (inclusief het buitengebied) door punten voorstelt, en de deuren door lijnen.

   (b) Zoek voor elk huis uit of het mogelijk is een rondwandelingetje te maken waarin je iedere deur precies één keer gebruikt en bij je uitgangspunt terugkeert.

   (c) Zoek voor elk huis uit of het mogelijk is een rondwandelingetje te maken waarin je ieder vertrek (en de tuin) precies één keer bezoekt en bij je uitgangspunt terugkeert.

6. Welke van de volgende graphen hebben een Hamilton-circuit?

7. Zij G de kubus-graph, met als P de verzameling van de acht hoekpunten, en als L de verzameling van de twaalf ribben

   (a) Is G planair?

   (b) Heeft G een Hamilton-circuit?

   (c) Heeft G een Euler-circuit?

8. Laat zien dat de Petersen graph een Hamilton-pad heeft, maar geen Hamilton-cykel.

9. Laat zien dat als een bipartite graph een Hamilton-pad toelaat het aantal rode en het aantal blauwe punten hoogstens één verschilt.

Veel van de stof uit dit blok is afkomstig uit [Gie]. Een ander goed naslagwerk is [Tru91].

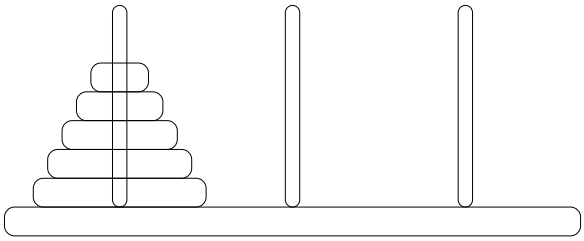
Recursie

Torens van Hanoi

De puzzel de 'torens van Hanoi' bestaat uit 3 pinnen en een aantal schijven van verschillend formaat. De bedoeling is om alle schijven naar één ander pin te verschuiven. Iedere zet mogen we een schijf verzetten, maar er mag nooit een grote schijf op een kleinere staan.

In het plaatje hebben we vijf schijven. Kun je deze puzzel oplossen?

Na wat puzzelen lukt het je misschien wel, maar dan weet je vaak niet meer hoe je het gedaan hebt. Verder wil je natuurlijk ook weten hoe je het met zeven schijven doet.



Het belangrijke idee bij het oplossen deze puzzel is om het probleem te *generaliseren*. Kunnen we het probleem oplossen bij een willekeurig aantal schijven? Daarnaast moeten we opmerken dat het probleem voor vijf schijven lijkt op het probleem van vier schijven, wat weer lijkt op het probleem voor drie schijven, etc.

Als ik het probleem voor vier schijven op kan lossen kan ik het ook voor vijf schijven. Want laat de grootste schijf maar even liggen. Verplaats de bovenste vier schijven naar pin 2 (We hebben aangenomen dat we dat kunnen). Pak nu de grootste schijf op en verplaats hem naar pin 3. Verplaats nu de andere schijven van pin 2 naar pin 3.

Nu zeg je misschien wat heb ik hieraan? Het probleem voor vier schijven kan ik ook niet oplossen. Nou ja, dan maken we het nog een beetje makkelijker. Als ik het probleem voor drie schijven kan oplossen, dan kan ik het ook voor vier. En als ik het probleem voor twee schijven kan oplossen, dan kan ik het ook voor drie. Tenslotte als ik het probleem voor een schijf kan oplossen, dan kan ik het ook voor twee. Dus *als* ik het probleem voor een schijf kan oplossen, dan kan ik het ook voor vijf. Maar het probleem voor een schijf is erg eenvoudig! Dus we kunnen het nu ook voor vijf schijven.

Tenslotte kunnen we nog opmerken dat er niet bijzonders is aan vijf. Het probleem kunnen we nu ook oplossen voor tien schijven, of voor ieder ander aantal.

We hebben het vorige probleem *recursief* opgelost. Dat wil zeggen, we hebben het probleem zo ingedeeld dat het uit elkaar valt in een aantal deelproblemen die heel veel op elkaar lijken en zodat moeilijkere problemen altijd zijn terug te brengen tot eenvoudigere problemen. Tenslotte moeten we het allereenvoudigste geval natuurlijk wel op kunnen lossen.

Recursie is ook een belangrijke programmeertechniek.

Vermenigvuldigen

Stel je hebt een programmeertaal waarin je wel kunt optellen, maar nog niet kunt vermenigvuldigen. Hoe definieer je de vermenigvuldiging?

Dat kunnen we recursief doen: $n * 1 := n$ en $n * (m + 1) := n * m + n$.

Nu we kunnen vermenigvuldigen kunnen we ook machtsverheffen: $n^1 := n$ en $n^{(m+1)} := n^m * n$.

Hoeveel zetten?

Hoeveel zetten hebben we nu nodig met onze strategie voor de torens van Hanoi? Laat a_n het aantal zetten zijn dat we nodig hebben om de puzzel met n schijven op te lossen. Dus $a_1 = 1$ en $a_2 = 3$. Maar wat is a_5? Het is lastig om dit in een keer in te zien. Maar in ieder geval weten we dat we eerst het probleem met vier schijven moeten oplossen, dan de grote schijf verschuiven en weer het probleem met vier schijven oplossen. Dus $a_5 = a_4 + 1 + a_4 = 2a_4 + 1$.

Net zo, $a_4 = 2a_3 + 1$ en $a_3 = 2a_2 + 1$. Maar a_2 kennen we! Dus $a_3 = 7$, $a_4 = 15$ en $a_5 = 31$. Op dezelfde manier kunnen we iedere a_n uitrekenen.

Als we nog eens naar de rij van oplossingen 1,3,7,15,31,... kijken valt je misschien op dat de rij lijkt op de rij 2,4,8,16,32,... De rij van de machten van twee. De eerste rij is steeds eentje minder. Is dit toeval of klopt het altijd? Stel even dat $a_{37} = 2^{37} - 1$, dan is

$$a_{38} = 2a_{37} + 1 = 2 \cdot (2^{37} - 1) + 1 = 2^{38} - 2 + 1 = 2^{38} - 1.$$

Dus als het voor het 37ste element klopt, dan ook voor het 38ste. Nu is 37 natuurlijk helemaal niet bijzonder. Dus op de zelfde manier zien we dat als het voor n klopt dan ook voor $n + 1$. Verder wisten we al dat $a_1 = 2^1 - 1$. Dus het klopt voor 2, dus het klopt voor 3, dus het klopt voor 4, dus ..., dus het klopt voor 37, dus het klopt voor 38, dus ... We zien dus dat voor alle getallen n geldt: $a_n = 2^n - 1$.

Inductie

De bewijsmethode die we net hebben gebruikt heet *inductie*. Inductie lijkt heel veel op de recursieve methode die we gebruiken om functies te definiëren.

Inductie kunnen we gebruiken als we voor alle natuurlijke getallen (de getallen 1, 2, 3, 4, 5, 6, 7, ...) een bepaalde uitspraak $P(n)$ willen bewijzen. Een bewijs met inductie gaat op de volgende manier:

1. Bewijs eerst $P(1)$.
2. Bewijs dan: Als $P(n)$, dan geldt ook $P(n + 1)$.

Dat is genoeg.

Als we nu willen laten zien dat bijvoorbeeld $P(37)$ geldt, dan weten we dat $P(1)$ waar is, dus (met 2.) ook $P(2)$, dus (weer met 2.) ook $P(3)$, dus ook $P(4)$,..., dus ook $P(36)$, dus ook $P(37)$!

In het vorige voorbeeld was $P(n)$ de uitspraak $a_n = 2^n - 1$.

Inductie kun je zien als het omgekeerde van recursie. Met inductie ligt de nadruk op steeds moeilijkere gevallen, bij recursie is dat omgekeerd.

Optellen

Hoeveel is $1+2+3+\ldots+99$? Dat kun je natuurlijk op papier doen. Je kunt je rekenmachine gebruiken. In beide gevallen ben je lang bezig. Tenzij je het slim doet. Je kunt bijvoorbeeld definiëren $s(n) := 1+2+\cdots+n$. Dan kun je een recursief programmaatje schrijven wat $s(99)$ voor je uitrekent, want $s(n + 1) = s(n) + n$.

Als je wat waarden uitrekent krijg je het vermoeden dat $s(n) = (n^2 + n)/2$.

Klopt dit nu ook? We gaan het bewijzen met inductie. Neem voor $P(n)$ de uitspraak $s(n) = (n^2 + n)/2$.

1. $P(1) : s(1) = 1 = (1^2 + 1)/2$.
2. Stel dat $P(n)$, d.w.z. $s(n) = (n^2 + n)/2$. Dan

$$s(n + 1) = s(n) + n + 1 = \frac{n^2 + n}{2} + n + 1 = \frac{n^2 + n + 2n + 2}{2} = \frac{(n + 1)^2 + (n + 1)}{2}.$$

Dus $P(n+1)$ geldt.

Dus geldt voor alle getallen n, $s(n) = (n^2 + n)/2$.

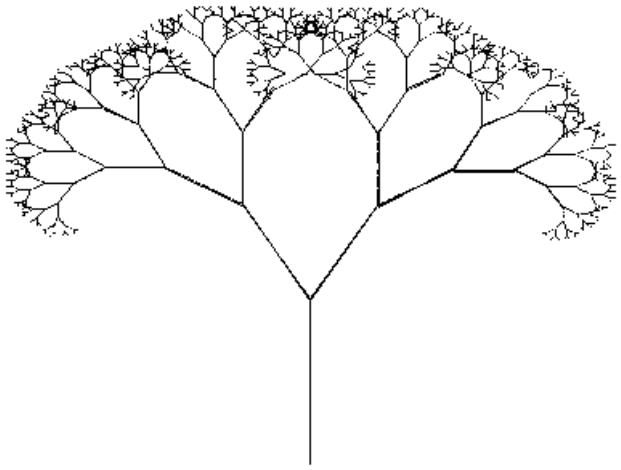
Rangschikkingen

Op hoeveel manieren kunnen we de cijfers 1,2,3,4,5,6,7,8,9 rangschikken? Het is lastig alle mogelijkheden op te schrijven. Maar gelukkig hoeft dat niet. Laat a_n het aantal rangschikkingen zijn van een verzameling van n elementen. We willen dus a_9 weten. Gelukkig weten we dat $a_1 = 1$.Verder is $a_{n+1} = (n+1)a_n$. Want we kiezen eerst een van de $(n+1)$ elementen, dat zetten we vooraan in de rij. Daarna hebben we nog n elementen en die kunnen we op a_n manieren rangschikken.

Hoe rekenen we a_9 nu uit? De definitie die we boven hebben gegeven is precies de definitie van de faculteit-functie, die wordt genoteerd met $n!$. Dus $a_9 = 9!$ en algemeen $a_n = n!$ Deze functie zit standaard op de meeste rekenmachines.

Binaire bomen

Beschouw het volgende plaatje:



Hoewel deze boom er misschien complex uit ziet, is deze getekend met een simpele recursieve procedure. Het basisinzicht hierbij is dat een boom bestaat uit een stam met daarboven twee andere bomen, links en rechts, die net iets kleiner zijn (en iets gedraaid zijn).

Een recursief recept (functie, procedure) is eenvoudig te geven:

1. teken een stam
2. teken de linker (sub)boom
3. teken de rechter (sub)boom

Dit recept kunnen we preciezer maken met de volgende functie/procedure:

$$f(n,x,y,\alpha,l) = \begin{cases} \textit{doe niets} & \textit{als } n = 0 \\ 1)\,\textit{teken stam}\,(\textit{positie } x,y;\ \textit{hoek } \alpha;\ \textit{lengte } l) & \\ 2)\,f(n-1,x_1,y_1,\alpha-30,l/2) & \textit{als } n > 0 \\ 3)\,f(n-1,x_2,y_2,\alpha+30,l/2) & \end{cases}$$

Hierbij is n de hoogte van de boom; x en y de positie van de boom (startpunt van de stam); α is de hoek van de boom en l is de lengte van de stam.

De hoogte van de boom, n, is belangrijk in deze recursieve procedure; de andere variabelen laten we even buiten beschouwing.

Het tekenen van een boom met hoogte n kan dus terug gebracht worden tot een simpeler probleem: het tekenen van twee bomen met hoogte $n-1$. Dit recursieve patroon is duidelijk te herkennen wanneer je de stappen volgt van een computer volgt die de procedure uitvoert:

```
[1] teken boom: f(3)
[1.1] teken stam
[1.2] teken linker boom: f(2)
[1.2.1] teken stam
[1.2.2] teken linker boom: f(1)
[1.2.2.1] teken stam
[1.2.2.2] teken linker boom: f(0)
[1.2.2.2.1] doe niets
[1.2.2.3] teken rechter boom: f(0)
[1.2.2.3.1] doe niets
[1.2.3] teken rechter boom: f(1)
[1.2.3.1] teken stam
[1.2.3.2] teken linker boom: f(0)
[1.2.3.2.1] doe niets
[1.2.3.3] teken rechter boom: f(0)
[1.2.3.3.1] doe niets
[1.3] teken rechter boom: f(2)
[1.3.1] teken stam
[1.3.2] teken linker boom: f(1)
[1.3.2.1] teken stam
[1.3.2.2] teken linker boom: f(0)
[1.3.2.2.1] doe niets
[1.3.2.3] teken rechter boom: f(0)
[1.3.2.3.1] doe niets
[1.3.3] teken rechter boom: f(1)
[1.3.3.1] teken stam
[1.3.3.2] teken linker boom: f(0)
[1.3.3.2.1] doe niets
[1.3.3.3] teken rechter boom: f(0)
[1.3.3.3.1] doe niets
```

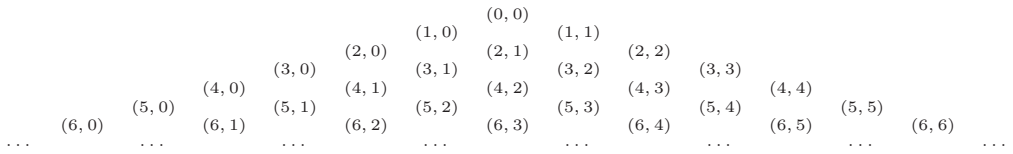
Merk op dat een recursieve *definitie* vaak erg eenvoudig is, maar de *uitvoering* van een recursieve functie veel werk met zich mee brengt: er zijn veel stappen, en je moet een goede

administratie bijhouden om steeds te weten met welk subprobleem je bezig bent. Typisch werk voor een computer dus.

Driehoek van Pascal

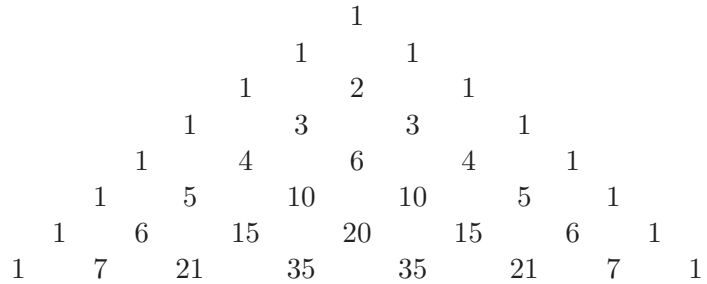
Binomiaalcoëfficiënten. We definiëren, voor natuurlijke getallen n en k met $k \leq n$, $\binom{n}{k}$ als het aantal manieren om k objecten uit een verzameling van n elementen te pakken.

We bekijken nu de roosterpunten (n, k) waarvoor $k \leq n$. Deze verzameling van roosterpunten tekenen we zó, dat het punt $(0, 0)$ de top wordt:



We gaan nu ieder van deze roosterpunten voorzien van een getal:

Eerste Driehoek van Pascal. Zet aan de rand eentjes, en vul de rest in door 'schuin optellen'. Anders gezegd: voorzie de roosterpunten $(n, 0)$ en (n, n) van het getal 1, en schrijf bij elk ander roosterpunt (n, k) de som van de getallen, die je bij de roosterpunten $(n-1, k)$ en $(n-1, k-1)$ moet schrijven. De Eerste Driehoek van Pascal (1e $\triangle$vP) is dus het volgende schema van getallen (het gaat naar onder oneindig ver door, ik heb alleen maar een klein beginstuk opgeschreven):



Tweede Driehoek van Pascal. Zet op plaats (n, k) het getal $\binom{n}{k}$

Opgave: bewijs dat er in 2e $\triangle$vP aan de rand allemaal eentjes staan.

We beweren dat als $k < n$, dan $\binom{n+1}{k+1} = \binom{n}{k+1} + \binom{n}{k}$. Kiezen we namelijk een deelverzameling A van k elementen uit $\{1, 2, 3, 4, \ldots, n+1\}$ dan zijn er twee mogelijkheden $n+1 \in A$ of $n \notin A$. In het eerste geval is $A - \{n+1\}$ dus een deelverzameling van k elementen uit $\{1, 2, 3, 4, \ldots, n\}$, in het tweede geval is A een deelverzameling van $k+1$ elementen uit $\{1, 2, 3, 4, \ldots, n\}$. Dus ook in de 2e $\triangle$vP het principe van 'schuin optellen' geldt: ieder inwendig getal is de som van de schuin erboven staande getallen.

Derde Driehoek van Pascal. Zet op plaats (n, k) het aantal wegen via roosterpunten van $(0, 0)$ naar (n, k), waarbij we iedere keer een stap naar links-onder of rechts-onder zetten.

Je kunt nu inzien:

$$\begin{cases} \text{in de 3e } \triangle \text{ vP staan aan de rand allemaal eentjes} \\ \text{in de 3e } \triangle \text{ vP geldt ook het principe van 'schuin optellen'} \end{cases}$$

Gevolg: de 3e $\triangle$vP is precies gelijk aan de 1e $\triangle$vP

Vierde Driehoek van Pascal. Zet op plaats (n, k) de coëfficiënt van x^k in de veelterm $(1 + x)^n$.

Ga zelf na, dat ook deze 4e $\triangle$vP weer precies gelijk is aan de 1e $\triangle$vP. De vier behandelde versies van de $\triangle$vP leiden dus allemaal tot hetzelfde schema van getallen

Binomium van Newton.

$$(1+x)^n = \binom{n}{0} + \binom{n}{1}x + \binom{n}{2}x^2 + \cdots + \binom{n}{n}x^n$$

Dit Binomium van Newton is niets anders dan de stelling: 2e $\triangle$vP = 4e $\triangle$vP.

Het meer algemene geval

$$(y+z)^n = \binom{n}{0}y^n + \binom{n}{1}y^{n-1}z + \binom{n}{2}y^{n-2}z^2 + \cdots + \binom{n}{n}z^n$$

volgt hier uit: $(y + z)^n = y^n(1 + (\frac{z}{y})^n)$ als $y \neq 0$. Het geval $y = 0$ is eenvoudig.

Opgaven

10. De uitvinder van het schaakbord mocht van de koning van Perzië een beloning uitzoeken. Hij mocht vragen wat hij maar wilde. Hij koos voor de volgende beloning. Hij wilde op het eerste vakje van het schaakbord 1 rijstkorrel hebben, op de tweede 2, op de derde 4, etc. Dus op ieder vakje twee keer zoveel rijstkorrels. De koning vond hem erg bescheiden.
    Laten we eens kijken hoeveel hij vroeg: hij wilde dus

    $$1 + 2 + 2^2 + 2^3 + 2^4 + \cdots + 2^{63}$$

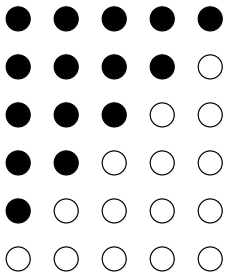
    korrels hebben. Kun je een eenvoudige formule vinden voor de uitkomst van deze som? [Hint: tel eerst eens wat termen uit het begin van de rij $1, 2, 2^2, 2^3, 2^4, \ldots$ op en bekijk het verband, dus bekijk 1 en $1+2$ en $1+2+2^2$ enz. Herken je iets? Wat is je vermoeden van de uitkomst? Kun je dit met inductie bewijzen?]

11. [Deze opgave past niet echt in dit hoofdstuk.] Neem aan dat een rijstkorrel breedte en dikte van 1mm heeft en een lengte van 5mm. Spreid alle rijstkorrels van vraag 1 gelijkmatig uit over Nederland (400km$\times$200km). Hoe hoog wordt deze berg?

12. Bewijs met inductie dat voor alle $n \in \mathbb{N}$, $2^n \geq n$.

13. We hebben al een mooie formule voor de som van de rij

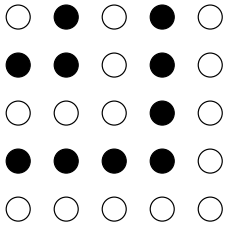
    $$1 + 2 + 3 + 4 + 5 + \cdots + n = \frac{(n+1)n}{2}.$$



Wat is het verband met het volgende plaatje?

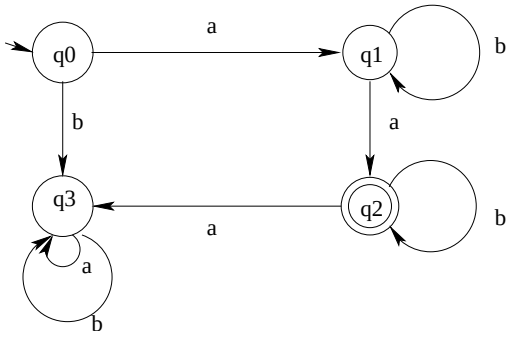
14. **Rijtjes van lengte n.** Hoeveel rijtjes kunnen we maken van lengte n, met daarin alleen de getallen 1 tot en met 5? Dit kunnen we ook weer recursief oplossen. Noem het aantal rijtjes van lengte n a_n. Een rijtje van lengte 1 kunnen we op 5 manieren maken, dus $a_1 = 5$. Een rijtje van lengte $n+1$ kunnen we maken door eerst een rijtje van lengte n te maken en daar een element achter te zetten. Dus $a_{n+1} := 5a_n$. Herinner je nu even de definitie van machtsverheffen. Bewijs met inductie dat voor alle n, $a_n = 5^n$.

15. Bewijs met inductie dat $1 + 3 + 5 + 7 + \cdots + (2n - 1) = n^2$.
Wat is het verband met het volgende plaatje?



5. Automaten

In de informatica worden ook machines bestudeerd. Dat gebeurt door naar de architectuur van computers zelf te kijken, maar ook door geïdealiseerde, abstracte machines te bestuderen. Het voordeel van het bestuderen van zo'n geïdealiseerd, abstract machine *model* is dat het eenvoudiger is om aan zo'n model allerlei eigenschappen te bestuderen. Een bekend abstract machine model zijn de *eindige automaten*. Deze eindige automaten hebben veel meer toepassingen dan alleen als model voor simpele berekeningen. Zo worden ook processen met eindige automaten (of kleine uitbreidingen daarvan) gemodelleerd. Dit zullen we verderop zien. Laten we eerst kijken wat een eindige automaat is en hoe je ermee 'rekent'. Hier is een voorbeeld van een eindige automaat.



(M0)

Een automaat is een soort graph, alleen zijn de lijnen nu *pijlen* en deze pijlen hebben ook een *label*. De punten in deze graph noemen we de *toestanden* van de automaat: $q0, q1, q2$ en $q3$. Het is gebruikelijk om deze toestanden als een rondje te schrijven met de naam van de toestand erin. Er zijn 2 toestanden van een speciale soort:

1. De *begintoestand*. Deze wordt aangegeven door er een pijl uit het niets naar toe te tekenen. Een automaat heeft altijd *precies één* begintoestand, hierboven dus $q0$.

2. De *eindtoestanden*. Deze worden aangegeven door een dubbele cirkel. Een automaat heeft altijd *minimaal één* eindtoestand, hierboven is er (toevallig) precies één, $q2$. (De begintoestand kan rustig ook een eindtoestand zijn!)

We kunnen een automaat zien als een (heel simpel) computertje dat voor ons kan rekenen. Dat rekenen gaat heel simpel: we kunnen er een woord instoppen (in bovenstaande automaat woorden over het alfabet $\{a, b\}$) en na een aantal stappen *eindigt* de automaat. Dan zijn we óf in een eindtoestand óf in een niet-eindtoestand. In het eerste geval zeggen we dat het woord *geaccepteerd* wordt, in het tweede geval wordt het woord *niet geaccepteerd* of ook wel *verworpen*.

5.1. Voorbeeld. In de bovenstaande automaat wordt het woord *aa* geaccepteerd: begin in $q0$ en lees het eerste teken a. Dan komen we in toestand $q1$ en we hebben nog a over. Lees ook deze a en we komen in $q2$. Er is nu geen input meer: de berekening stopt in een eindtoestand, dus *aa* is *geaccepteerd*.
Ga zelf na dat *abbba* en *ababb* geaccepteerd worden en dat *abababab* en *bab* niet geaccepteerd worden.

Voor we verder gaan geven we een precieze definitie van het begrip eindige automaat.

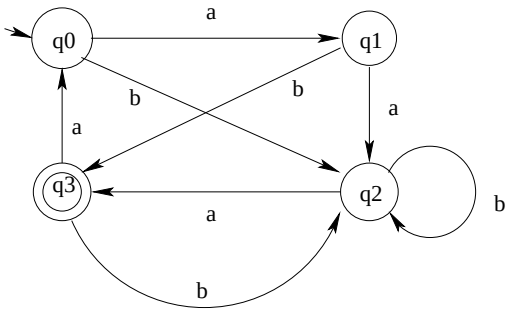
5.2. Definitie. Een *eindige automaat* bestaat uit de volgende 5 componenten

1. Een eindige verzameling Σ, het *input alfabet* of de verzameling van *atomaire acties*,
2. Een eindige verzameling Q, de *toestanden*,
3. Een speciale toestand $q_0 \in Q$, de *begintoestand*,
4. Een niet-lege verzameling speciale toestanden $F \subseteq Q$, de *eindtoestanden*,
5. Een *overgangsfunctie* δ, die bij iedere toestand q en actie a aangeeft wat de nieuwe toestand q' is. (Dit zijn de gelabelde pijlen in de automaat boven).

Een eindige automaat noemt men ook wel een *DFA*, naar het Engels': "Deterministic Finite Automaton". Een eindige automaat wordt soms geschreven als $M := <\Sigma, Q, q_0, F, \delta>$.

We zullen, als we een automaat moeten definiëren, gewoon een diagram tekenen, net als we in het voorbeeld boven gedaan hebben.

5.3. Opgave. Bekijk de volgende automaat $M1$.



(M1)

In deze automaat is $Q = \{q0, q1, q2, q3\}$, $F = \{q3\}$ en $\Sigma = \{a, b\}$.

1. Ga voor de volgende woorden na of ze geaccepteerd worden: *abaab*, *aaaba*, *bab*, λ en *aabbab*.
2. Gelden de volgende uitspraken? (Geef steeds een tegenvoorbeeld of een bewijs.)
   - Als w geaccepteerd wordt, dan wordt $wabba$ ook geaccepteerd.
   - Als w geaccepteerd wordt, dan wordt wab niet geaccepteerd.
   - Als w niet geaccepteerd wordt, dan wordt waa ook niet geaccepteerd.
   - Als w niet geaccepteerd wordt, dan wordt wbb ook niet geaccepteerd.

5.1. Automaten en Talen

We kunnen de Σ uit de eindige automaat opvatten als de verzameling van "atomaire acties" (die ons van één toestand overvoeren in de volgende), maar ook als de symbolen van een alfabet, zoals we boven al gedaan hebben. Als we aan Σ denken als een alfabet, kunnen we een automaat zien als een *taalherkenner*. Op deze manier hoort er bij iedere eindige automaat een taal, nl. de taal bestaande uit precies die woorden die de automaat accepteert.

5.4. Definitie. Bij een eindige automaat $M := <\Sigma, Q, q_0, F, \delta>$ definiëren we de *taal van* M, $L(M)$ als volgt

$$L(M) := \{w \in \Sigma^* \mid w \text{ wordt geaccepteerd door } M\}.$$

Dus: $w \in L(M)$ desda de automaat M stopt in een eindtoestand op invoer w.

Laten we eens kijken welke taal de automaat $M0$ uit het voorbeeld accepteert. De volgende woorden worden geaccepteerd:

- aa wordt geaccepteerd,
- awa wordt geaccepteerd waarbij w een willekeurig lange rij b-tjes is,
- $awav$ wordt geaccepteerd waarbij w en v beide willekeurig lange rijen b-tjes zijn.

Als we toestand $q2$ "voorbij zijn" komen we nooit meer in een eindtoestand, dus wat hierboven staat is alles. We kunnen dit samenvatten als

$$L(M0) = \{ab^n ab^m \mid n, m \geq 0\}$$

Op basis van onze kennis van talen, opgedaan in hoofdstuk 3, zien we meteen de reguliere expressie die bij deze taal hoort (en we kunnen dus concluderen dat $L(M0)$ regulier is):

$$L(M0) = \mathcal{L}(ab^*ab^*).$$

We kunnen hetzelfde met de taal van $M1$ proberen, maar dat is lastiger:

- ab wordt geaccepteerd,
- ba wordt geaccepteerd,
- aaa wordt geaccepteerd,
- aab^ka wordt geaccepteerd waarbij $k \geq 0$,
- bb^ka wordt geaccepteerd waarbij $k \geq 0$,
- $aab^ka(ba)^l$ wordt geaccepteerd waarbij $k \geq 0$, $l \geq 0$,
- $bb^ka(ba)^l$ wordt geaccepteerd waarbij $k \geq 0$, $l \geq 0$,

maar we hebben nog steeds niet alles, want als we $q2$ "voorbij zijn" kunnen we via een lus weer in $q2$ terechtkomen. Kunnen we toch meer grip krijgen op de taal van deze automaat? Ja dat kunnen we door bij deze automaat een *grammatica* te maken die dezelfde taal *genereert*. Dit gaat als volgt

1. Maak bij iedere toestand q_i een hulpsymbool X_i, zodat bij q_0 het startsymbool S hoort.
2. Als $q_i \xrightarrow{a} q_j$ in de automaat, voeg dan aan de grammatica de regel $X_i \to aX_j$ toe.
3. Als $q_i \in F$, voeg dan aan de grammatica de regel $X_i \to \lambda$ toe.

Voor de automaat $M1$ krijgen we dan de volgende grammatica G_1, waarbij $\Sigma = \{a, b\}$.

$$\begin{array}{rcl} S & \to & bB \mid aA \\ A & \to & aB \mid bC \\ B & \to & bB \mid aC \\ C & \to & bB \mid aS \mid \lambda \end{array}$$

Merk op dat deze grammatica *rechtslineair* is en dat ook de taal $L(M1)$ dus *regulier* is.

Deze schrijfwijze van de taal $L(M1)$ m.b.v. een grammatica is interessant, al was het alleen al omdat het een andere beschrijving van dezelfde taal oplevert. Maar het kan nog meer opleveren, namelijk als we in de grammatica symbolen gaan substitueren: vul eerst voor A in het rechterlid alle mogelijkheden voor A in: aB en bC. Doe dan hetzelfde met C: vul in de rechterleden voor C overal in bB, aS en λ. Dit levert de volgende grammatica G_2 op, die dezelfde taal als boven genereert. Dit is weer een rechtslineaire grammatica (zie de definitie).

$$\begin{array}{rcl} S & \to & bB \mid aaB \mid abbB \mid abaS \mid ab \\ B & \to & bB \mid abB \mid aaS \mid a \end{array}$$

5.5. Opgave. Concludeer uit de grammatica G_2 dat

1. $(aba)^k ab \in L(M1)$ voor $k \geq 0$,
2. $aab^k a \in L(M1)$ voor $k \geq 0$,
3. als $w \in L(M1)$, dan ook $abaw \in L(M1)$,
4. als $w \in L(M1)$, dan ook $aaaaw \in L(M1)$,

Dat we van een eindige automaat een rechtslineaire grammatica kunnen maken, op de mainier zoals we boven voor $M1$ gedaan hebben is een algemeen feit.

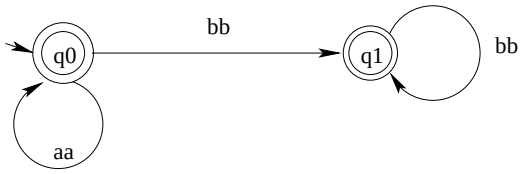
5.6. Stelling. Bij iedere eindige automaat M kunnen we een rechtslineaire grammatica G maken zodat $L(G) = L(M)$.
(De taal die G genereert is dezelfde als de taal die M accepteert.) Bij gevolg is $L(M)$ regulier voor iedere eindige automaat M.

5.7. Opgave. Maak een rechtslineaire grammatica bij de eindige automaat $M0$, zoals boven gedaan voor $M1$. Verklein deze grammatica door overbodige productieregels en hulpsymbolen weg te laten.

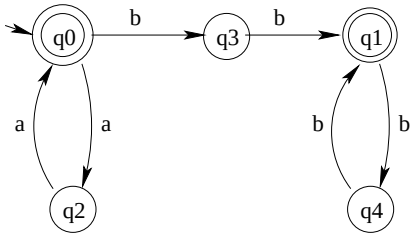
We kunnen ook de andere kant op: bij een rechtslineaire grammatica een eindige automaat maken. Bekijk de volgende rechtslineaire grammatica G_3.

$$
\begin{aligned}
S &\rightarrow aaS \mid bbB \mid \lambda \\
B &\rightarrow bbB \mid \lambda
\end{aligned}
$$

We creëren nu eerst voor ieder hulpsymbool een toestand en maken transities (pijlen) gelabeld met woorden (in plaats van alleen letters). De hulpsymbolen die naar λ kunnen gaan worden eindtoestand en S wordt uiteraard begintoestand.

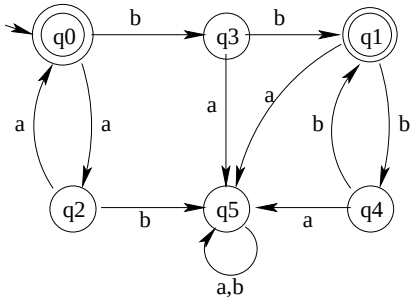


Vervolgens voegen we extra toestanden toe waarmee we de woorden bij de pijlen 'ophakken" in letters. We krijgen dan



(M2)

Dit is nog (net) geen eindige automaat, want bij een eindige automaat eisen we dat er *vanuit iedere toestand voor iedere letter* een stap mogelijk is, en dat is nu niet zo. Om er een eindige automaat van te maken moeten we een soort "put" toevoegen waar alle nog niet opgegeven transities naar toe gaan en waar je niet meer uitkomt. Dat wordt $q5$ en het levert uiteindelijk de volgende eindige automaat op.



(M2)

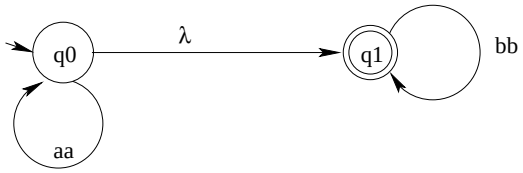
5.8. Opmerking. In de bovenstaande automaat hebben we één pijl (van $q5$ naar zichzelf twee labels gegeven. Dit is hetzelfde als 2 gelabelde pijlen, maar de pijlen-spaghetti wordt zo iets minder.
We zouden ook toe kunnen staan dat we niet alle pijlen hoeven toe te voegen en dan zouden

we de "put" dus weg kunnen laten. Dat doen we toch niet, want bijvoorbeeld een woord *bba* moet *niet* door $M2$ geaccepteerd worden: dat is duidelijk in de 2de versie van $M2$, terwijl in de 1ste versie de automaat stopt in toestand $q1$, een eindtoestand ...

Deze procedure om een automaat uit een rechtslineaire grammatica te maken werkt heel algemeen, behalve wanneer we een productieregel van de vorm $S \rightarrow B$ hebben, dus waar het woord dat vòòr het hulpsymbool staat λ is. Bekijk de volgende grammatica G_4.

$$\begin{aligned} S &\rightarrow aaS \mid B \\ B &\rightarrow bbB \mid \lambda \end{aligned}$$

Als we hier de eerste stap zetten om een automaat te maken (op de manier die we boven gevolgd hebben) krijgen we hetvolgende.



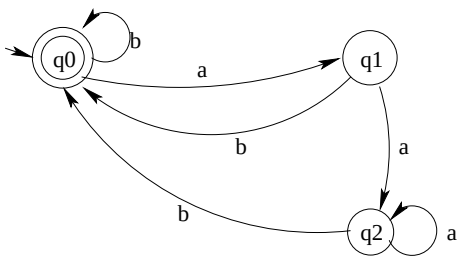
We hebben dus λ, het lege woord, bij de pijl en dit woord kunnen we uiteraard niet "ophakken", dus hier werkt onze techniek niet. De oplossing is om eerst een *equivalente* rechtslineaire grammatica te verzinnen waar producties van de vorm $S \rightarrow B$ niet voorkomen. Dat kan (altijd), maar we laten hier niet zien hoe je het in zijn algemeen doet. Voor G_4 zou je uitkomen op de grammatica G_3, dus de automaat $M2$ accepteert dezelfde taal als grammatica G_4 genereert. (En G_3 en G_4 genereren dezelfde taal: ga dat voor jezelf na!)

5.9. Stelling. Bij iedere rechtslineaire grammatica G kunnen we een eindige automaat M maken zodat $L(M) = L(G)$.

5.10. Opgave. Maak een eindige automaat die de taal van de volgende grammatica accepteert.

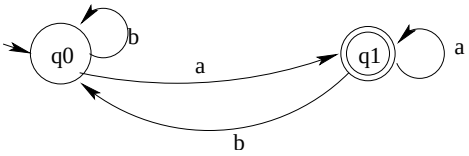
$$\begin{aligned} S &\rightarrow abS \mid aA \mid bB \\ A &\rightarrow aA \mid \lambda \\ B &\rightarrow bB \mid \lambda \end{aligned}$$

5.11. Opgave. Bekijk de eindige automaat $M3$ (volgende pagina).



(M3)

Maak een rechtslineaire grammatica die $L(M3)$ genereert.



(M4)

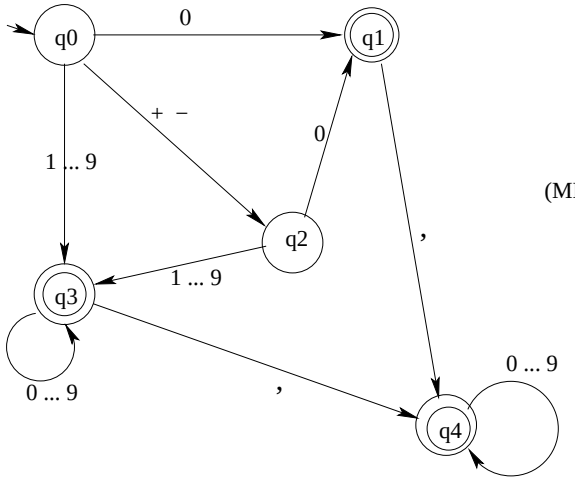
5.12. Opgave. Bekijk de eindige automaat $M4$ (volgende pagina).

1. Maak een rechtslineaire grammatica die $L(M4)$ genereert.
2. Geef een (eenvoudige) beschrijving van $L(M4)$.
3. Als we alle toestanden eindtoestand maken, welke taal accepteert $M4$ dan?
4. Als we de eindtoestanden en de niet-eindtoestanden omwisselen, welke taal accepteert $M4$ dan?

5.13. Opgave. Maak een eindige automaat die de volgende taal over $\Sigma = \{a, b\}$ accepteert: $L := \{(ab)^k(aba)^l \mid k, l \geq 0\}$

5.14. Opgave. Maak een eindige automaat die de volgende taal over $\Sigma = \{a, b\}$ accepteert: $L := \{(ab)^k x(ab)^l \mid x \in \{a, b\}, k, l \geq 0\}$

5.15. Opgave. Bekijk de automaat MD die (input) getallen in 'normale' decimale notatie accepteert. Zo'n getal mag evt. beginnen met een + of een −, maar niet met een 0, behalve natuurlijk als het van de vorm $0,5$ is, want dan is het weer wel goed. In het diagram staat $0 \ldots 9$ voor één van de getallen uit 0 t/m 9 enz.



(MD)

1. Vind je de automaat MD goed? Worden alle 'juiste' getallen geaccepteerd en alle 'foute' verworpen?
2. Pas MD aan als je het er niet mee eens bent.
3. Geef een rechtslineaire grammatica die dezelfde taal als MD genereert.

5.2. Nondeterministische Automaten

In de definitie van automaat hebben we een speciale eis gesteld

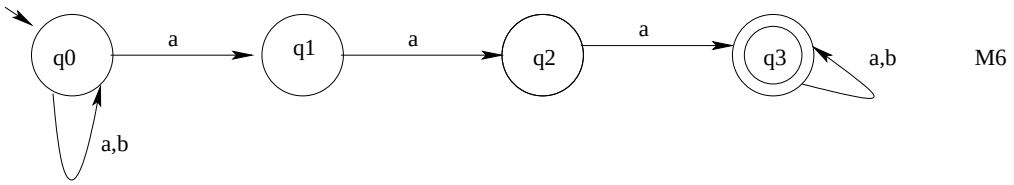
> Bij iedere toestand q en iedere letter a uit het alfabet is er *precies één pijl* vanuit q met label a.

We kunnen deze eis verzwakken als volgt

- Vanuit iedere toestand q zijn er *eindig veel* pijlen met label a voor iedere $a \in \Sigma$ (dus 0, 1 of meer)
- Vanuit iedere toestand q zijn er *eindig veel* pijlen met label λ (dus 0, 1 of meer)

Een automaat waarin we dit toestaan noemen we een *non-determinstische eindige automaat*.

5.16. VOORBEELD. Bekijk de volgende automaat



1. Als we het woord *baaa* als invoer nemen, zijn er 2 berekeningen mogelijk: we kunnen eindigen in q_0 of in q_3. De laatste is een eindtoestand, de eerste niet. Dit fenomeeen noemen we *non-determinisme*: de automaat voert *non-deterministisch* (niet van te voren te bepalen) één van de mogelijke berekeningen uit.
2. Als we het woord *bbb* als invoer nemen is er maar één berekening mogelijk, die eindigt in q_0.
3. Op invoer *baa* zijn er 2 berekeningen mogelijk die beide eindigen in een niet-eindtoestand.
4. Op invoer *aba* zijn er ook twee berekeningen mogelijk. Eén eindigt in q_0 en de ander "eindigt" in q_1 *terwijl nog niet het hele woord gelezen is* (alleen de letter a is gelezen). Deze situatie noemen we *deadlock*: de berekening kan niet verder, hoewel nog niet alle input gelezen is.

Wanneer *accepteert* een non-determinstische eindige automaat een woord w?

5.17. DEFINITIE. De non-deterministische automaat M *accepteert* het woord w als op invoer w *er een berekening is* die stopt in een eindtoestand waarbij *hete hele woord gelezen* is.

Een berekening die de hele invoer "consumeert" en eindigt in een eindtoestand noemen we ook wel een *succesvolle* of *accepterende* bereking. Dus: als de berekening stopt in een *deadlock*, is dit geen succesvolle berekening.

Laten we ook even precies definiëren wat een non-deterministische eindige automaat is en wat de taal is die zo'n automaat accepteert.

5.18. DEFINITIE. Een *non-deterministische eindige automaat* bestaat uit de volgende 5 componenten

1. Een eindige verzameling Σ, het *input alfabet* of de verzameling van *atomaire acties*,
2. Een eindige verzameling Q, de *toestanden*,
3. Een speciale toestand $q_0 \in Q$, de *begintoestand*,
4. Een verzameling speciale toestanden $F \subseteq Q$, de *eindtoestanden*,
5. Een *overgangsrelatie* δ, die bij iedere toestand q en $d \in \Sigma \cup \{\lambda\}$ een verzameling van toestanden X geeft. (Als $q' \in \delta(q, d)$, dan is er een pijl $q \xrightarrow{d} q'$, dus dit zijn de gelabelde pijlen in het diagram van de automaat).

Een non-deterministische eindige automaat wordt soms geschreven als $M := <\Sigma, Q, q_0, F, \delta>$. De *taal van* M, $L(M)$, is als volgt gedefinieerd

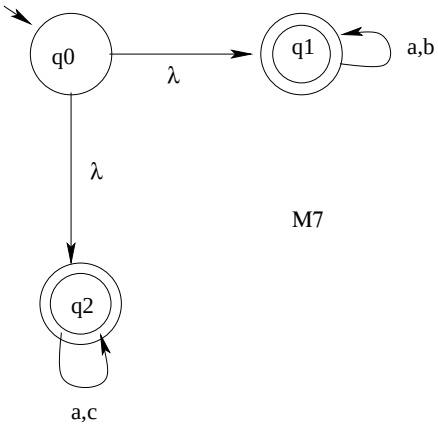
$$L(M) := \{w \in \Sigma^* \mid w \text{ wordt geaccepteerd door } M\}.$$

Dus: $w \in L(M)$ desda er is een berekening van de automaat M op invoer w die stopt in een eindtoestand waarbij w geheel gelezen is.

5.19. OPGAVE. 1. Ga in de bovenstaande non-deterministische automaat M_6 na welke berekeningen er zijn met invoer *abaaa*, *ababa*, *ab* en *baaab*.

2. Welke van bovenstaande woorden worden geaccepteerd?
3. Beschrijf de taal die $M6$ accepteert.
4. Pas M_6 aan zodat hij $\{w \mid w \text{ eindigt met } aaa\}$ accepteert.

5.20. OPGAVE. Bekijk de automaat M_7.



1. Welke berekeningen zijn er op invoer *aba*, *cac*, *abc* en λ?
2. Welke van deze woorden wordt geaccepteerd?

3. Beschrijf de taal die M_7 accepteert.

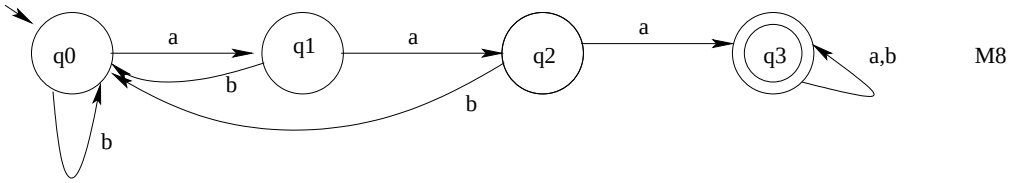
Kunnen we met non-deterministische automaten meer dan met deterministische automaten? Ja, we kunnen non-deterministische berekeingen modelleren. Maar kunnen we ook talen accepteren die we eerst niet konden accepteren? Oftwel:

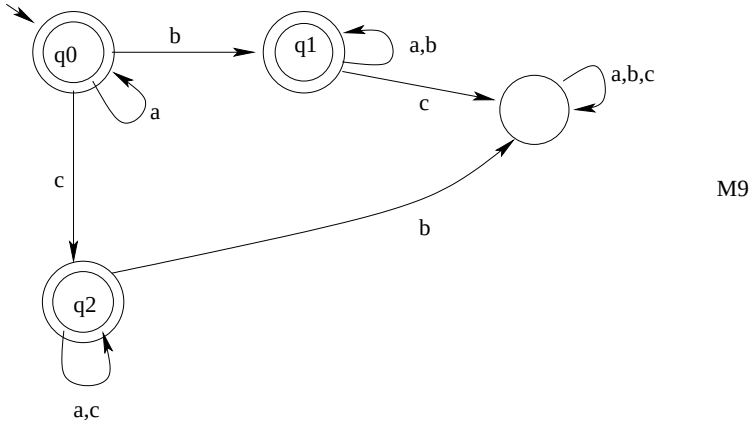
> Is er een taal L waarvoor we *wel* een non-deterministische automaat M hebben die L accepteert ($L = L(M)$), maar waarvoor er *geen* deterministische automaat M' is die L accepteert ($L = L(M')$)

Het antwoord is nee:

5.21. Stelling. Bij iedere non-deterministische automaat M kunnen we een deterministische automaat M' maken zodat $L(M) = L(M')$.

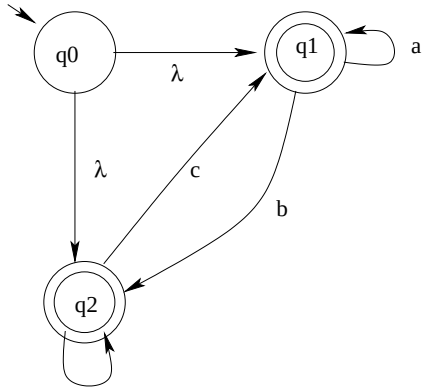
De constructie is niet vreselijk moeilijk, maar doen we hier toch niet. We illustreren het met twee voorbeelden en dan zien we meteen waarom non-deterministische automaten soms handig zijn (want veel kleiner). Bekijk daartoe de 2 eindige automaten die corresponderen met M_6 en M_7: M_8 en M_9. Verifieer zelf dat deze automaten inderdaad dezelfde talen accepteren.





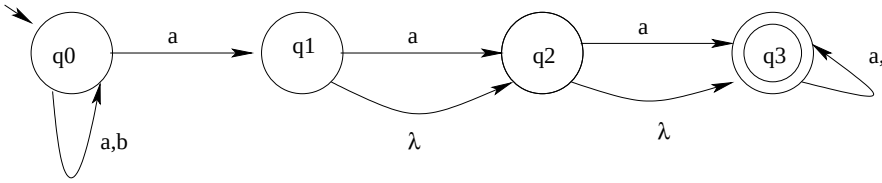
5.22. Opgave. 1. Maak een non-deterministische automaat voor de taal $L = \{w \in \{a,b,c\}^* \mid w \text{ eindigt met } aab \text{ of } w \text{ eindigt met } ccb\}$.

2. Maak een non-deterministische automaat voor de taal $L' = \{w \in \{a,b,c\}^* \mid w = vu \text{ en } v \text{ bevat } aa \text{ en } u \text{ bevat } bb\}$.



M11

5.23. Opgave.



M12

1. Maak een deterministische automaat die de taal van M_{11} accepteert.
2. Beschrijf de taal die M_{11} accepteert.
3. Maak een deterministische automaat die de taal van M_{12} accepteert.
4. Beschrijf de taal die M_{12} accepteert.

5.24. Opgave.

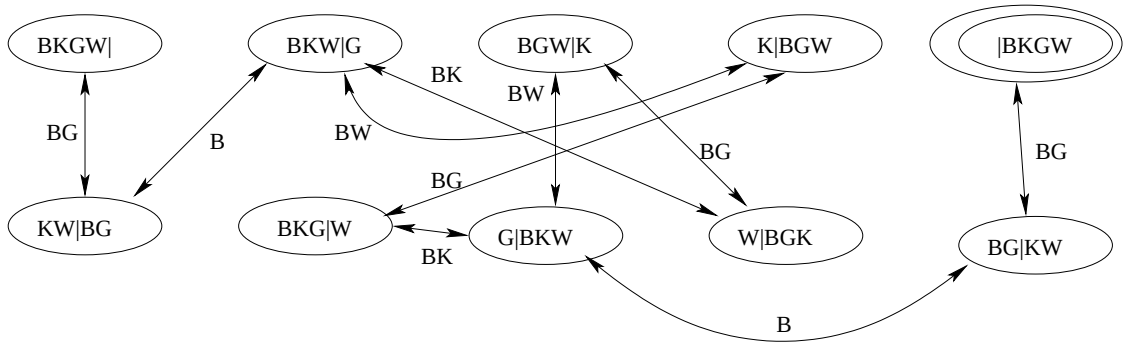
1. Stel dat M_1 een automaat is die L_1 accepteert en dat M_2 een automaat is die L_2 accepteert. Maak een non-determinitische automaat die $L_1 \cup L_2$ accepteert. (Hint kijk naar voorbeeld M_7 boven.)
2. Bewijs dat de klasse van reguliere talen gesloten is onder $\cup$, d.w.z. als L_1 en L_2 regulier zijn, dan is $L_1 \cup L_2$ ook regulier.
3. Stel dat M_1 een automaat is die L_1 accepteert en dat M_2 een automaat is die L_2 accepteert. Maak een non-determinitische automaat die L_1L_2 accepteert. (L_1L_2 is de taal bestaande uit eerst een woord uit L_1 en dan een woor uit L_2, dus $L_1L_2 = \{vw \mid v \in L_1, w \in L_2\}$.)
4. Bewijs dat de klasse van reguliere talen gesloten is onder *concatenatie* d.w.z. als L_1 en L_2 regulier zijn, dan is L_1L_2 ook regulier.

5.3. Automaten en processen

We gaan nu met (non-deterministische) automaten processen modelleren.

De Boer B staat met een kool K, een geit G en een wolf W aan één zijde van de rivier en wil naar de overkant. Er is een roeiboot, maar daar kan hij maar met hooguit één "medepassagier" in. Hoe komt hij naar de overkant zonder dat de geit de kool op eet of de wolf de geit opeet?

5.25. VOORBEELD. We modelleren dit door alle "goede toestanden" (waarbij niets misgaat) te tekenen en alle mogelijke overgangen aan te geven met een pijl die aangeeft wie er in de boot overgaan. (Alle pijlen kunnen dus twee kanten op, daarom heb ik ze hier als een dubbele pijl getekend.). Je kunt ook *alle* toestanden tekenen , maar dan wordt de automaat nog groter.



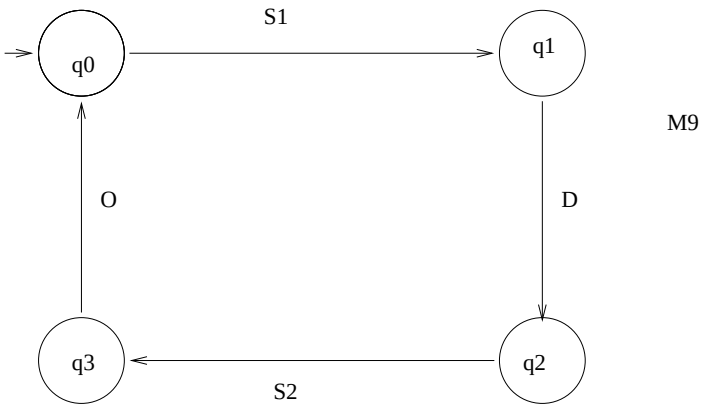
De vraag is nu niet welke "woorden geaccepteerd" worden, maar of er een pad van begin naar eindtoestand is. In termen van talen zou je dit kunnen formuleren als: is de taal van deze automaat leeg? (Dan is het probleem van de boer onoplosbaar) Of anders: is er een woord dat geaccepteerd wordt door deze automaat?
Ga na dat er oneindig veel oplossingen zijn. Ga na dat er twee essentieel verschillende 'optimale' oplossingen zijn voor de boer.

We zien dat we bovenstaand probleem kunnen oplossen door het als een automaat te modelleren. We waren op zoek naar een 'proces' en dat lezen we af uit de automaat. Een iets andere situatie ontstaat als we een proces willen besturen. Deze besturing kunnen we ook modelleren m.b.v. een automaat en dan kunnen we daaraan vaak al allerlei problemen aflezen.

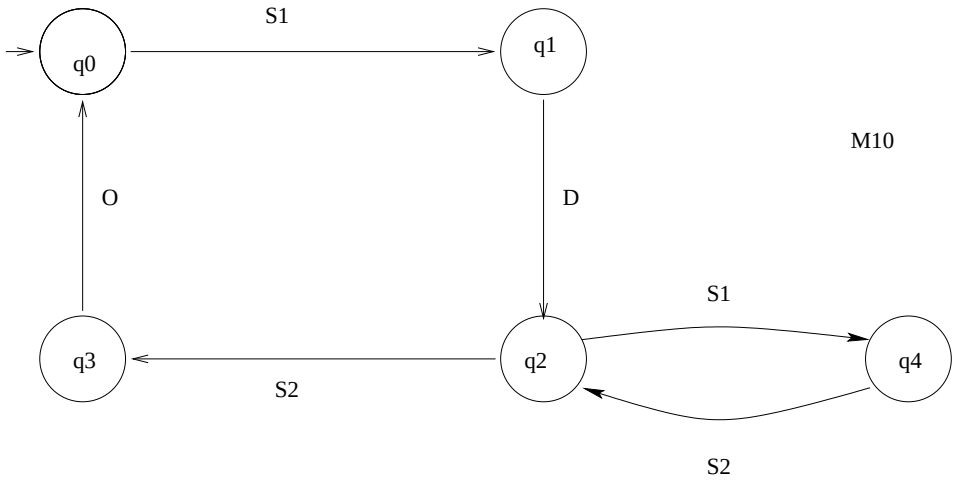
5.26. VOORBEELD. Gegeven een spoor met een spoorwegovergang met automatische overwegbomen. We nemen aan dat treinen altijd van links naar rechts rijden. Links zit een eind voor de overgang een sensor S_1 die een voorbijkomende trein detecteert. Rechts zit na de overgang sensor S_2 die hetzelfde doet. De besturing van de overgang m.b.v. de sensoren zou als volget gemodelleerd kunnen worden met automaat M_9. (D betekent: bomen gaan dicht; O betekent: bomen gaan open.)

Merk op dat er geen "eindtoestand" is: de automaat is nooit klaar, want we modelleren een (oneindig) proces.

De vraag is nu of dit een goede besturing is, aannemende dat de sensoren en overwegbomen goed functioneren (en dat er geen treinen van rechts naar links rijden). Dit is niet het geval, want het kan fout gaan als er 2 treinen in hetzelfde "baanvak" zitten: als er na de eerste trein



t_1 een tweede trein t_2 voorbij S_1 komt, nog voor dat t_1 langs S_2 komt, dan gaat de spoorboom open terwijl t_2 nog tussen de 2 sensoren is (en misschien wel bij de spoorwegovergang). Dus de besturing moet verfijnder, om precies te zijn als volgt:



Pas zelf de automaat aan voor het geval er ook 3 treinen kort na elkaar kunnen rijden. (In het algemene geval is het handig om hiervoor een *push-down automaat*, ook wel *stapelautomaat* genoemd, te gebruiken.)

5.27. Opgave. Modelleer het volgende probleem (op de manier van het $BKGW$ probleem).

- Er staan 4 soldaten, A, B, C en D voor een gammele brug in het donker en moeten zo snel mogelijk naar de overkant.
- Ze hebben slechts één lamp en de brug kan maximaal 2 personen tegelijk dragen.
- De soldaten zijn gewond en lopen niet snel meer: ze hebben het volgende aantal minuten nodig om de brug te passeren: A 5, B 10, C 20 en D 25 minuten.

1. Modelleer de toestanden in de automaat.

2. Modelleer de toestandsovergangen in de automaat. (Houdt bij een pijl ook de tijd bij die zo'n overgang kost.)

3. Lees uit de automaat af wat de tijd is die de soldaten minimaal nodig hebben om de overkant te bereiken.

5.28. Opgave. Modelleer de besturing van een spoorwegovergang met 2 sporen, waar over de voorste de treinen van links naar rechts rijden (met sensoren S_1 en S_2) en over de achterste van rechts naar links (met sensoren S_3 en S_4).
Doe het eerst voor het simpele geval (waar je nooit 2 treinen in hetzelfde baanvak hebt) en dan voor het moeilijkere geval.

References

[Gie] Wim Gielen. Discrete wiskunde voor informatici. Katholieke Universiteit Nijmegen - Subfaculteit Wiskunde 2002.

[Tru91] J.K. Truss. *Discrete Mathematics for computer scientists.* Addison-Wesley, 1991. ISBN 0-201-17564-9.

[vBe] van Benthem *et. al. Logica voor informatici.* Addison-Wesley, Nederland. ISBN 90-6789-484-2 (dit boek wordt ook gebruikt in het college Beweren en Bewijzen).