Logica voor informatica

Matt ter Steege
matttersteege@gmail.com
Universiteit Utrecht
Utrecht, Nederland

Contents

Contents 1
1 Propositielogica 2
1.1 Proposities 2
1.2 Semantiek van propositie operatoren 2
1.3 Voorbij thruth tables 2
2 sets 3
2.1 leden en gelijkheid 3
2.2 subsets and supersets 3
2.3 Operaties op sets 3
2.4 Partities 4
2.5 Naive set theory 4
3 Boolean algebra 4
3.1 Bools algebra regels 4
3.2 monoïde 4
3.3 Bool's algebra 4
3.4 waarheidstabel naar formule/circuit 4
3.5 dualiteit 5
4 Predikatenlogica 5
4.1 Predicaten en vrije variabelen 5
4.2 Kwantificatoren en gebonden variabelen 5
4.3 Universele kwantificator 5
4.4 Existentiële kwantificator 5
4.5 Gebonden kwantificatoren 6
4.6 predikatenlogica lezen 6
5 Proof strategies 6
5.1 Bewijzen voor logische operatoren 6
5.2 Bewijzen voor kwantificatoren 7
5.3 Afgeleide bewijzen 7
6 functies 8
6.1 terminologie 8
6.2 afbeelding en pre-afbeelding 8
6.3 grafieken en identiteit functies 8
6.4 injectief, surjectief en bijectief 9
6.5 compositie van functies 9
7 Relaties 10
7.1 Eigenschappen van relaties 10
7.2 equivalentie klassen 10
8 Inductie 10
8.1 Backus-Naur Form (BNF) 10
8.2 inductief gedefineerde functies 11
8.3 Recursieve functies 11
9 Games 12
10 Natural deduction 12
10.1 Inference rules 12
10.2 natuurlijke deductie in propositielogica (Derivation) 13
11 Redenering over progamma's 14
12 Hoare logica 14

1 Propositielogica

Propositielogica is de studie van bewijzen. Een propositie is te beantwoorden met waar of niet waar Een propositie is $3 < 17$, dit kan je checken op waarheid. Een propositie is $x \implies y$.

Propositielogica komt ook met een hele hoop termen, zo heb je **negatie** wat (gezien in een waarheidstabel) het omdraaien van waarde betekend, waar wordt onwaar en vice versa. Je hebt ook **conjunctie**, wat een chique woord is voor *en*, $a \wedge b$, b EN a. Een andere variant is **disjunctie**, wat dan staat voor *of*, $a \vee b$, a OF b. **Implicatie** is de er ook een, dat is de $x \implies y$, a IMPLCIEERT b. De allerlaatste is **equivalentie**, $a \Leftrightarrow b$, a is equivalent aan b. LET OP! Dit betekend niet "gelijk aan" 2 rode canta's zijn equivalent, maar niet hetzelfde, het is namelijk niet letterlijk dezelfde auto, maar ze zijn wel identiek aan elkaar.

1.1 Proposities

We gebruiken in logica variabelen zoals P en Q, dit zijn *atomisch* proposities, deze zijn altijd geschreven in HOOFDLETTERS. Deze zijn niet verder op te delen, dus niet opgebouwd van kleinere delen zoals implicaties etc. deze waardes zijn Waar/True/1 of Onwaar/False/0.

Je hebt ook kleine letters p, q, …, dit zijn niet-atomische proposities, het zijn geen propositionele formules, maar eerder *metavariabelen*.

Elke propositionele formule bestaat uit:

- Atomitsche proposities (P, Q, R, …)
- true, (T, Waar, 1)
- false, (F, Onwaar, 0)

Deze kunnen ook kleiner opgedeeld zijn, dan zijn dit ook propositionele formules:

- $P \implies Q$ - **implicatie** als P, dan Q
- $P \wedge Q$ - **conjunctie** P én Q
- $P \vee Q$ - **disjunctie** P of Q
- $\neg P$ - **negatie** (niet P / P houd geen stand)

Zoals in de wiskunde ook is heeft propositielogica óók een volgorde, deze is:

(1) Haakjes ()
(2) Negatie ¬
(3) Conjunctie ∧
(4) Disjunctie ∨
(5) Implicatie $\implies$

$\neg P \vee Q \implies Q \wedge P$ moet gelezen worden als $((\neg P) \vee Q) \implies (Q \wedge P)$

Om dit te onthouden kan je deze zij onthouden: "Hoe Navigeert Connie De Ijssel"

1.2 Semantiek van propositie operatoren

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \oplus Q$	$P \implies Q$	$P \equiv Q$
0	0	1	0	0	0	1	1
0	1		0	1	1	1	0
1	0	0	0	1	1	0	0
1	1		1	1	0	1	1

Tabel 1: Truthtable van basisoperaties

Maar stel, je wilt iets moeilijks bewijzen zoals $\neg P \vee Q \implies Q \wedge P$, dan kan je dat op de volgende manier doen:

P	Q	$\neg P$	$\neg P \vee Q$	$Q \wedge P$	$\neg P \vee Q \implies Q \wedge P$
0	0	1	1	0	0
0	1	1	1	0	0
1	0	0	0	0	1
1	1	0	1	1	1

Tabel 2: Truthtable van moeilijkere propositie formule

Je kan ook met verschillende kleuren pennen een kleine waarheidstabel schrijven, maar dit is een hele nette manier om het ook te doen.

1.3 Voorbij thruth tables

Je kan d.m.v een truth table bewijzen of een formule wel of niet houd, maar dat kan ook anders, bijvoorbeeld door het versimpelen van de formules.

	Expression	**Law**
	$(p \implies q) \vee (q \implies p)$	original
$\Leftrightarrow$	$(\neg p \vee q) \vee (\neg q \vee p)$	implication
$\Leftrightarrow$	$\neg p \vee ((q \vee \neg q) \vee p)$	associativity / rearrangement
$\Leftrightarrow$	$\neg p \vee (T \vee p)$	tertium non datur
$\Leftrightarrow$	$\neg p \vee T$	absorbing property of T
$\Leftrightarrow$	T	absorbing property of T

Mocht het nou gebeuren dat elke rij waar is, ofwel de combinatie van input waardes maakt niet uit voor de uitkomst; deze is altijd waar. Dan heet dat een **tautologie**. Voor het omgekeerde (altijd onwaar), geld dat dit een **contradictie** heet.

Deze afleiding kan je maken door de regels te gebruiken die hieronder stan geschreven (er zijn er nog meer).

Commutativity:

- $p \wedge q \Leftrightarrow q \wedge p$
- $p \vee q \Leftrightarrow q \vee p$

Associativity:

- $p \wedge (q \wedge r) \Leftrightarrow (p \wedge q) \wedge r$
- $p \vee (q \vee r) \Leftrightarrow (p \vee q) \vee r$

Tertium non datur:

- $p \vee \neg p \Leftrightarrow T$

Idempotence:

- $p \wedge p \Leftrightarrow p$
- $p \vee p \Leftrightarrow p$

De Morgan:

- $\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$
- $\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$

Double negation (niet niet is wel):

- $p \Leftrightarrow \neg(\neg p)$

Properties of T en F:

- $p \vee F \Leftrightarrow p$
- $p \wedge F \Leftrightarrow F$
- $q \vee T \Leftrightarrow T$
- $q \wedge T \Leftrightarrow q$

Implication:

- $(p \implies q) \Leftrightarrow (\neg p \vee q)$

Contraposition:

- $(p \implies q) \Leftrightarrow (\neg q \implies \neg p)$

2 sets

Propositielogica heeft ook een manier om over een groep dingen te redeneren, ***sets***. Deze schrijf tussen accolades met comma's tussen de items: $\{false, true\}$, $\{3, 7, 14\}$, $\{red, blue, yellow\}$. Dit kan voor kleine sets met de hand, maar als je grote sets wilt schrijven, dan kost dat heel veel tijd, daarom kan je ook: 1, 3, 5, ..., 99 om alle oneven cijfers van 1 t/m 99 als set te hebben. Of als 1, 2, 3, ... om alle cijfers vanaf 1 te hebben (oneindig). De lengte van een sets is op te schrijven door |A| of A, waar A een set is met een lengte, de **cardinality** hier tel je alleen het aantal unieke elementen, maar dit werkt alleen voor eindige sets, dus $|1, 2, 3, 4| = 4$. Als er maar 1 element in een set zit, dan heet dat een **singleton**.

Het **cartesisch product** van twee sets A en B is de set van alle mogelijke paren (a, b) waarbij a een element is van A en b een element is van B. Dit schrijf je als $A \times B$, en stel A = 1, 2 en B = x, y, dan is $A \times B = \{(1, x), (1, y), (2, x), (2, y)\}$.

Ook kan je de ***set-builder notation*** gebruiken, hierbij schrijf je het soort element dat je wilt en wat het element kan zijn: {x : x has the property P}

Voorbeeld **(Set-builder notation)**:

Een collectie van alle prime numbers: {x : x is een priemgetal}
Of {s : s is een strand in Nederland}.

Er zijn ook een paar basis sets, die al van tevoren zijn gedefinieerd:

- $\emptyset = \{\}$ (the empty set)
- $\mathbb{B} = \{0, 1\}$ (the binaire set)
- $\mathbb{N} = \{0, 1, 2, 3, ...\}$ (De natuurlijke nummers)
- $\mathbb{Z} = \{..., -2, -1, 0, 1, 2, ...\}$ (De integers)
- $\mathbb{Q} = \{\frac{m}{n} : \text{m, n} \in \mathbb{Z}, \text{n}\neg 0\}$ (De rationele nummers)
- $\mathbb{R} = \{x : \text{x is a real number}\}$ (De real nummers)

$\emptyset$ en { $\emptyset$ } zijn twee verschillende dingen, de een is namelijk: { }, terwijl de ander { { } } is!

2.1 leden en gelijkheid

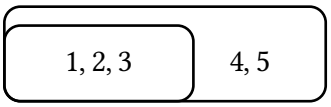
Je kan bekijken of een set een bepaalde member heeft door $7 \in \{3, 7, 14\}$ te doen. Hier is $\in$ te lezen als *"is element onderdeel van"* Je hebt ook de negatie vorm, $\notin$, wat *"is element niet onderdeel van"*, oftwel $6 \notin \{3, 7, 14\}$ betekend.

Ook is goed om te weten dat gelijkheid op basis van unieke element rust, dus $\{1, 2, 3, 2, 1\}\{1, 3, 2\}$. Ongelijkheid is dus te bepalen door te kijken of een van de twee sets een waarde heeft die niet in de ander voorkomt. Dus volgorde maakt niet uit, en hoeveelheid elementen die dan welniet hetzelfde zijn maakt ook niet uit.

2.2 subsets and supersets

Een ***subset*** is een set waarin minstens een deel van de waardes van de superset in voorkomen.

Dus $A = \{1, 2, 3\}$ is een subset van $B = \{1, 2, 3, 4\}$, hierin is B een ***superset*** van A en A een ***subset*** van B Dit schrijf je als volgt: $A \subseteq B$ *(A is included or contained in B)* en vice versa $B \supseteq A$ *(B includes or contains A).*



2.3 Operaties op sets

Union $A \cup B = \{x : x \in A \vee x \in B\}$ Combineert A en B met elkaar
Intersection $A \cap B = \{x : x \in A \wedge x \in B\}$ Het deel wat zowel in A als in B zit
Complement $\overline{A} = \{x : x \notin A\}$, alles wat niet in A zit.
Difference $A = \{x : x \in A \wedge X \notin B\}$ Alles wat in A zit, maar niet in B

Zo heb je ook de *powerset P(A)*, dit is een set waarin alle mogelijke subsets van A zitten: $P(A) = \{X : X \subseteq A\}$. Deze heeft n^2 elementen, waar n het aantal elementen van set A is.

Voorbeeld **(powerset/matchsset)**:

Stel je hebt een computer scherm, van 1680 x 1050, je wilt elke mogelijke configuratie van zwart/witte pixels opschrijven. Een set geeft aan welke pixels wit zijn, dus je wilt elke set met elke mogelijke manier van pixelcoordinaten, dat kan je zo schrijven:

$W = \{0, 1, ..., 1697\}$ voor elke X coordinaat
$H = \{0, 1, ..., 1049\}$ Voor elke Y coordinaat
$W \times H = \{\{0, 0\}, \{0, 1\}, \{0, 2\}, ..., \{1, 0\}, ...\{1679, 1049\}\}$ dit is een volledig wit scherm, elke pixel is gerepresentateerd in deze set
$P(W \times H)$ = elke configuratie van witte pixels

2.4 Partities

Je kan een set A opdelen in meerdere sets, zoals A_1 en A_2, hier zijn de regels: $A_1 \cup A_2 = A$ en $A_1 \cap A_2 = \emptyset$ Zo kan je A = $\mathbb{N}$ opsplitsen in A_1 en A_2 waar A_1 de even getallen zijn en A_2 de oneven getallen zijn.

2.5 Naive set theory

3 Boolean algebra

korte notitie* in Bolean algebra is de + hetzelfde als de OR operator, $\cdot$ hetzelfde als de AND operator, $^{-1}$ hetzelfde als NOT

3.1 Bools algebra regels

Commutativity:
- $x + y = y + x$
- $x \cdot y = y \cdot x$

Associativity:
- $x + (y + z) = (x + y) + z$
- $x \cdot (y \cdot z) = (x \cdot y) \cdot z$

Distributivity:
- $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$
- $x + (y \cdot z) = (x + y) \cdot (x + z)$

Identity:
- $x + 0 = x$
- $x \cdot 1 = x$

Complement:
- $x + \overline{x} = 1$
- $x \cdot \overline{x} = 0$

3.2 monoïde

Monoide is een verzameling A, is niet ledig en heeft minstens 1 element e, en een binaire operator $\oplus$ Een monoide heeft een nul-element, dit is een waarde die niks doet in de operatie. $x + 0 = x$, hier doet de 0 helemenaal niks, dus 0 is het 0-element in de "+" operatie. $x \times 1 = x$, hier is 1 het nul-element in de monoide $\times$ 1; het doet niks in de operatie. het is dus een neutrale waarde.

3.3 Bool's algebra

Boolean algebra heeft een paar regels:
- een set B
- Twee elementen $0 \in B$ en $1 \in B$, genaamd **zero** en **unit**
- Twee binaire operatoren + en $\cdot$ som/OR en product/AND respectievelijk.
- Een Unaire operator $^{-1}$ de inverse genoemd ofwel NOT (kan ook als x' zijn genoteerd).

Voorbeeld **(Bewijs dat $x + x = x$)**:

We willen bewijzen dat voor elke x, $x + x = x$

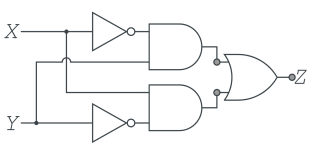
$x + x$	$= (x + x) \cdot 1$	Ident
	$= (x + x) \cdot (x \cdot x^{-1})$	Complement
	$= x + (x \cdot x^{-1})$	Distrobution
	$= x + ()$	Complement
	$= x$	Ident

3.4 waarheidstabel naar formule/circuit

Stel je krijgt een waarheidstabel zoals deze:

x	y	z
0	0	0
0	1	1
1	0	1
1	1	0

Als je dit omzet naar een propositie, dan moet je kijken naar alle rijen waar de uitkomst 1 is. Voor elke 0 waarde schrijf je de variabele (hier x of y), op als $\overline{x}$ of $\overline{y}$. Elke 1 waarde schrijf je alleen de variabele op, dus x of y. Als je dat hier zou doen dan krijg je hetvolgende antwoord: $(\overline{x} \wedge y) \vee (x \wedge \overline{y})$. Dit kan je dan vervolgens omzetten naar een circuit:



3.5 dualiteit

Je kan voor elke vergelijking die je opschrijft een soort duale tweede vergelijking maken door hetvoglende te doen:
- 0 met 1 vervangen
- 1 met 0 te vervangen
- + met · te vervangen
- · met + te vervangen

4 Predikatenlogica

4.1 Predicaten en vrije variabelen

We zagen eerder de *set-builder*-notatie

$$\{x \mid x \text{ heeft eigenschap } P\},$$

waarbij P een voorwaarde is. Een voorbeeld van zo'n voorwaarde is

$$P(x) = Prime(x) = \text{“}x \text{ is een priemgetal”}.$$

Het predicaat geeft voor elke waarde x waar of onwaar terug, en op basis daarvan bepalen we of x tot de verzameling behoort. De bijbehorende *truth set* bestaat uit alle waarden die voldoen aan het predicaat.

Voorbeeld **(Eendjes)**:

We hebben zeven eendjes:

$$Eendjes = \{Kwik, Kwek, Kwak, Dagobert, Donald, Katrien, Dumbella\}.$$

We definiëren het predicaat

$$Female(x) = \text{“}x \text{ is vrouwelijk”}.$$

- $Female(Katrien) \land Female(Dumbella)$ is waar.
- $Female(Kwik) \land Female(Kwek) \land \cdots \land Female(Donald)$ is niet waar.
- De truth set van $Female(x)$ is dus $\{Katrien, Dumbella\}$.

4.2 Kwantificatoren en gebonden variabelen

Stel dat we willen controleren of elke waarde in een lijst aan een voorwaarde voldoet. Het boek geeft het volgende voorbeeld:
Er zijn vier kinderen:

$$Kinderen = \{Joel, Felix, Oskar, Amanda\}.$$

Eén van hen moet voorin op de passagiersstoel zitten, omdat er achterin maar drie plekken zijn. Het predicaat $Front(x)$ betekent dat kind x voorin zit. We weten dus zeker dat voor één waarde van x het predicaat waar moet zijn.
De samengestelde propositie

$$Front(Joel) \lor Front(Felix) \lor Front(Oskar) \lor Front(Amanda)$$

moet waar zijn. Dat levert in principe een grote lijst aan mogelijke combinaties op, wat snel onoverzichtelijk wordt.

Als je een predikaat schrijft zoals $P(x) = \ldots x \ldots$, dan is x een gebonden variabele. Als je een predikaat schrijft zoals $P(x) = \ldots y \ldots$, dan is y een vrije variabele.
Bij gebonden variabelen kan je de x vervangen door een waarde, $P(x) = x > 1337$. $P(10000)$ wordt dan $10000 > 1337$

4.3 Universele kwantificator

Om dit compacter te schrijven gebruiken we de universele kwantor $\forall$. De notatie $\forall x\, P(x)$ betekent: “voor elke x geldt $P(x)$”. De propositie is alleen waar wanneer alle waarden voldoen aan P.
Het voorbeeld

$$Happy(Joel) \land Happy(Felix) \land Happy(Oskar) \land Happy(Amanda)$$

kun je schrijven als

$$\forall x\, Happy(x),$$

waarbij de vraag is: zijn Joel, Felix, Oskar én Amanda allemaal blij?

4.4 Existentiële kwantificator

Als je wilt uitdrukken dat er minstens één waarde is waarvoor het predicaat waar is, gebruik je de existentiële kwantor $\exists$.
De notatie $\exists x\, P(x)$ betekent: “er bestaat een x waarvoor $P(x)$ geldt”. Deze propositie is waar wanneer één of meer waarden voldoen aan P.
Het voorbeeld

$$Happy(Joel) \lor Happy(Felix) \lor Happy(Oskar) \lor Happy(Amanda)$$

kan worden geschreven als

$$\exists x\, Happy(x).$$

Daarnaast bestaat er de vorm $\exists! x\, P(x)$: “Er bestaat precies één x waarvoor $P(x)$ geldt.”

4.5 Gebonden kwantificatoren

Je kunt kwantoren beperken tot een verzameling.

De universele, beperkte vorm:

$$\forall x \in A\ P(x) \quad \text{(alle waarden in } A \text{ voldoen aan } P\text{)}.$$

De existentiële, beperkte vorm:

$$\exists x \in A\ P(x) \quad \text{(er bestaat minstens één waarde in } A \text{ die voldoet aan } P\text{)}.$$

4.6 predikatenlogica lezen

- $\forall x\, P(x) \quad \rightarrow$ "Voor alle x geldt: $P(x)$". "Elke [type van x] heeft de eigenschap ...".
- $\exists x\, P(x) \quad \rightarrow$ "Er bestaat een x zodanig dat $P(x)$". "Er is ten minste één [type] waarvoor ...".
- $\exists! x\, P(x) \quad \rightarrow$ "Er bestaat precies één x zodanig dat $P(x)$". In goed Nederlands: "Er is precies één ... die/het ...".
- $P(x) \wedge Q(x) \quad \rightarrow$ "$P(x)$ en $Q(x)$".
- $P(x) \vee Q(x) \quad \rightarrow$ "$P(x)$ en/of $Q(x)$".
- $P(x) \rightarrow Q(x) \quad \rightarrow$ "Als $P(x)$, dan $Q(x)$".
- $\neg P(x) \quad \rightarrow$ "niet $P(x)$".
- $x = y \quad \rightarrow$ "x en y zijn hetzelfde object".
- Meerdere kwantoren: volg strikt de volgorde $\forall x \exists y \rightarrow$ "Voor elke x bestaat een y ...", $\exists y \forall x \rightarrow$ "Er bestaat een y die voor alle x ..." (volgorde verandert betekenis!).

Voorbeeld **(Een Predikatenlogica formule lezen)**:

$$\forall x\big(Student(x) \rightarrow \exists y(Course(y) \wedge Enrolled(x, y) \wedge \neg Failed(x, y))\big).$$

Stap 1: Kies domein = "personen en vakken": variabele x = persoon, y = vak.

Stap 2:

- Buitenste kwantor: $\forall x$ — "Voor alle personen x geldt: ..."
- Binnen: $Student(x) \rightarrow \exists y(\dots)$ — een implicatie: "als x student is, dan bestaat er een y zodanig dat ..."
- Binnenste existentie: $\exists y(Course(y) \wedge Enrolled(x, y) \wedge \neg Failed(x, y))$.

Stap 3:

> Voor elke persoon x: als x een student is, dan bestaat er een vak y zodanig dat y een vak is, x staat ingeschreven voor y, en het is niet zo dat x voor y gezakt is.

Stap 4: Herschrijf de zin:

> Elke student is ingeschreven voor minstens één vak waarin die niet gezakt is.

---

5 Proof strategies

5.1 Bewijzen voor logische operatoren

Implicatie-Introductie ($\implies -I$)

Deze gaat als volgt: **Neem aan dat P waar is, bewijs Q, daarom houdt** $P \implies Q$. Dit werkt omdat de waarheidstabel van inplicatie altijd waar is, behalve als P waar is en Q niet. Dus als we ervanuit gaan dat P waar is en we bewijzen dat Q ook waar is, dan klopt de stelling. Namelijk als P niet waar blijkt te zijn, dan maakt Q niet meer uit, want de uitkomst is dan (volgens de waarheidstabel) toch waar. En Q kan onwaar zijn, want we hebben het tegendeel bewezen.

Implicatie-Eliminatie ($\implies -E$)

Deze gaat als volgt: **Bewijs** $P \implies Q$**, bewijs P, daarom houdt Q**. Dit werkt, omdat we de volledige stelling bewijzen, daardoor weten we dat in een waarheidstabel alleen de 1e,2e en 4e rijen nog opties zijn. Als we dan bewijzen dat P waar is, dan weten we met 100% zekerheid dat Q dus ook waar moet zijn, want er is maar 1 optie waar P waar is en de stelling in zijn geheel waar is (4e rij).

Conjunctie-Introductie ($\wedge - I$)

Deze is heel simple, als je $P \wedge Q$ wilt bewijzen, dan moet je deze natuurlijk los bewijzen. Ofwel, **Bewijs P, Bewijs Q, daarom houdt** $P \wedge Q$.

Conjunctie-Eliminatie ($\wedge - E$)

Hier zijn twee varianten van. De eerste is **Bewijs** $P \wedge Q$**, bewijs P, daarom houdt Q**, de ander is **Bewijs** $P \wedge Q$**, bewijs Q, daarom houdt P**. Je bewijst namelijk al dat de gehele stelling moet waar zijn, en dat kan alleen als allebei de delen waar zijn. Als je dus bewijst dat de linker- of rechterzijde waar is, dan weet je dat de andere zijde ook waar moet zijn.

Negatie-Introdcutie ($\neg - I$)

Deze valt nog mee: **Neem aan dat P waar is, bewijs het tegendeel (**$\neg P$**), daarom houdt P niet.** Want P is altijd waar, totdat je bewijst dat het niet zo is.

Negatie-Eliminatie ($\neg - E$)

Bewijs dat P waar is én bewijs dat P niet waar is ($\neg P$), dan krijg je dus $P \wedge \neg P$, die stelling komt altijd uit op $\neg P$, want waar én onwaar is onwaar.

equivalentie-Introdcutie ($\Leftrightarrow -I$)

Als je $P \Leftrightarrow Q$ wilt bewijzen, dan moet je dus de 2 onderdelen hiervan bewijzen. Namelijk: **Bewijs** $P \implies Q$**, bewijs** $Q \implies P$**, daarom houdt** $P \Leftrightarrow Q$. De definitie van equivalentie is namelijk dat beide onderdelen elkaar impliceren, dus daarom moet je ze apart van elkaar bewijzen.

equivalentie-Eliminatie ($\Leftrightarrow -E$)

Zoals elke Eliminatie strategie moet je eerst bewijzen dat de gehele stelling houdt en daarna kan je conclusies trekken. Hier is dat zo: **Bewijs** $P \Leftrightarrow Q$**, daarom houdt** $P \implies Q$ óf **Bewijs** $P \Leftrightarrow Q$**, daarom houdt** $Q \implies P$ Omdat (nogmaals de definitie van equivalentie) is een equivalent op te splitsen in twee implicaties.

Disjunctie-Introdcutie ($\vee - I$)

Deze is heel simpel en je hebt hier 2 vormen van als je $P \vee Q$ wilt bewijzen. Namelijk: **bewijs P, daarom houdt** $P \vee Q$, de ander is **bewijs Q, daarom houdt** $P \vee Q$. In een disjunctie (OR) is namelijk of links waar of rechts waar (of beide) dus als je er één bewijst, dan maakt het niet meer uit wat de ander is.

Disjunctie-Eliminatie ($\vee - E$)

Hoe simpel de introductie was, zo lastig is de eliminatie. Je moet namelijk heel veel stappen doorlopen om $P \vee Q$ te bewijzen. Namelijk: **Bewijs** $P \vee Q$**, 1. neem aan dat P waar is, bewijs R; 2. neem aan dat Q waar is, bewijs R; Daarom is R waar onafhankelijk of P of Q waar is.** Deze is wat lastiger om te bedenken, maar bekijk het zo: 1. Als ik binnen ben, dan heb ik mijn telefoon bij me. 2. Als ik buiten ben, dan heb ik mijn telefoon bij me. Ik ben binnen óf buiten, dus ik heb mijn telefoon bij me.

5.2 Bewijzen voor kwantificatoren

Universele kwantificering–Introductie ($\forall - I$)

Het is hier de bedoeling dat je bewijst dat een stelling voor elke waar de houdt. Dat doe je dus door te zeggen aan het begin van je bewijs "let a be arbitrairy". Dit geeft aan dat je het bewijs gaat geven met een waarde die niet vast staat (misschien wel alleen onderdeel is van de natuurlijke nummers e.d. maar je gaat niet in op een eigenschap van één specifiek element).

Stappen:

(1) Neem een element a, waarbij je expliciet zegt dat het willekeurig is.
(2) Bewijs dat $P(a)$ geldt zonder enig speciaal kenmerk van a te gebruiken.
(3) Concludeer vervolgens dat $P(x)$ voor alle x geldt, dus $\forall x\, P(x)$.

Universele kwantificering–Elimination ($\forall - E$)

Deze is makkelijker, namelijk dat $\forall x.P(x)$, als dat gelukt is, dan weet je al dat voor elke a, P(a) houdt.

Existentiële kwantificering–Introductie ($\exists - I$)

Deze is een stuk makkelijker dan de introductie van Universele kwantificering, want je hoeft maar één element te kunnen benoemen waarvoor de stelling houdt. En dan ben je al klaar, want de $\exists x$ geeft aan dat er minimaal één element x is waarvoor de stelling houdt.

Stappen:

(1) Neem een concreet object t.
(2) Bewijs dat $P(t)$ geldt.
(3) Concludeer dat er minstens één object bestaat waarvoor P geldt: $\exists x\, P(x)$.

Existentiële kwantificering–Introductie ($\exists - E$)

Deze gaat als volgt: **Bewijs dat** $\exists x.P(x)$ **houdt**, dan noem je het/een element waardoor de stelling houdt de naam R (o.i.d.). Je bewijst dat R een van/de rede is dat de propositie houdt, dan heb je bewezen dat R houdt.

5.3 Afgeleide bewijzen

Contrapositie

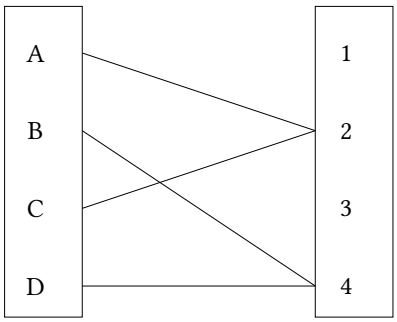
Je neemt hier aan: **Neem aan dat** $\neg Q$**, bewijs** $\neg P$**, daarom** $P \implies Q$ Dit kan je terugzien in de een waarheidstabel:

P	Q	$P \implies Q$	$\neg Q$	$\neg P$	$\neg Q \implies \neg P$
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	0	0
1	1	1	0	0	1

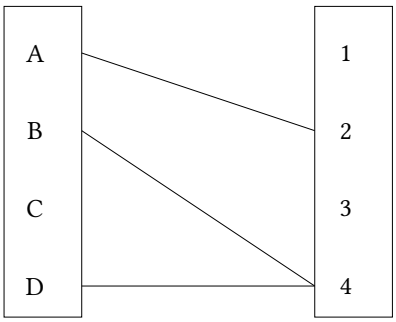
6 functies

Als je op elk element in eens set een verandering wilt doorvoeren, dan kan je daar een functie voor gebruiken. $f : A \longrightarrow B$ betekend dat je op elke element in A een functie uitvoerd en deze in de set B zet.

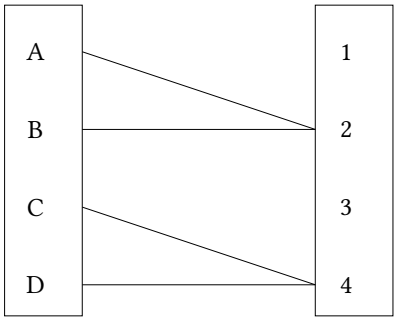
Het is mogelijk dat er verschillende inputs zijn, die dezelfde output geven. Zo kan er functie $even(x)$ bestaan die de set $\{1, 2, 3, 4, 5, 6\} \rightarrow \{0, 1, 0, 1, 0, 1\}$ (0 betekend oneven, 1 betekend even). Je kan dus schrijven: $even(1) = even(3) = even(5) = 0$. Het is wel zo dat de functie elke input om moet zetten en een output, hieronder staan mogelijkheden en of deze wel of te niet correcte functies zijn.



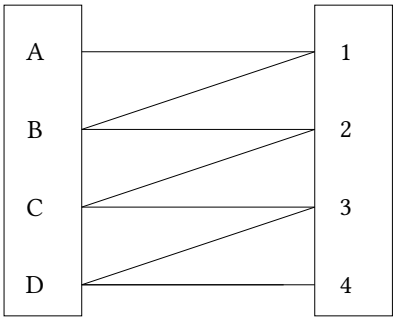
Dit is een functie: elke input geeft een output.



Dit is geen functie, want C heeft geen output, dat moet wel



Ook dit is een functie, want elke input geeft een output.



Dit is geen functie, input mogen min- en maximaal 1 output hebben

6.1 terminologie

A noemen we het **domein**, en B noemen we het **codomein**. Je kan ook meerdere argumenten meegeven: $f : A \times B \times C \longrightarrow D$ De hoeveelheid argumenten noem je de **ariteit**.

Als je een functie defineerd die 2 variabelen hebt, een **binaire functie** dan kan je die opeen speciale manier schrijven. Namelijk de functie "+" kan je hem tussen de variablene zetten, namelijk $A + B$ wat hetzelfde betekend als $+(A, B)$.

6.2 afbeelding en pre-afbeelding

de image van een set is je begint met een set en die zet je om naar de na-functie set. Deze is gedefinieerd als: $f(S) = \{f(a) : a \in S\}$. Te lezen als: "De set S wordt door functie f omgezet naar een nieuwe set, die bestaat uit alle outputs van f voor elk element a in S".

De pre-image is het resultaat en die wil je dan omzetten naar het origineel. Deze is gedefinieerd als: $f^{-1}(T) = \{a : a \in A \wedge f(a) \in T\}$. Te lezen als: "De set T wordt door de inverse van functie f omgezet naar een nieuwe set, die bestaat uit alle inputs a in A waarvoor de output van f in T zit".

Voorbeeld **(image en pre-image)**:

Dit voorbeeld met de functie: $sqrt(x)$ die inhoudt dat de output $x \times x$ bedraagt. We nemen ook de set $S = \{1, 4, 9, 16\}$.
Dan is de image van de set $sqrt(S)$ gelijk aan $\{1, 16, 81, 256\}$, dit is ná de functie is uitgevoerd op de input set.
De Pre-image is het tegenovergestelde, die is namelijk $\{1, 2, 3, 4\}$. Hier kijk je naar de input set en ga je ervanuit dat de functie al is uitgevoerd en moet je dus kijken welke zet heeft dit als resultaat.

6.3 grafieken en identiteit functies

Zoals we kennen in de wiskunde van $y = 2x$, kan je een grafiek maken van de functie. In de functie $f : A \rightarrow B$ defineren we de subset $A \times B$ als de grafiek van f. Dit houdt in dat elk element in A gekoppeld is aan zijn output in B. (zoals in een xy-grafiek). Een graph functie is dus een verzameling van paren (a, b) waarbij a in A zit en b de output is van f(a) is. Zo krijg je dus de grafiek van f als: $G = \{(a, f(a)) : a \in A\}$. Te lezen als: "De grafiek G is de verzameling van alle paren (a, f(a)) waarbij a in A zit".

De identiteit functie is een speciale functie die elk element in A koppelt aan zichzelf in B. Dit is dus $id : A \rightarrow A$ met $id(a) = a$ voor alle a in A. Dit houdt in dat de output van de functie altijd hetzelfde is als de input.

Voor een Subset S van A, kunnen we een characteristieke functie definiëren, deze is meestal genoteerd als $\chi : A \to \{0, 1\}$. Deze functie werkt als volgt:

$$\chi(a) = \begin{cases} 1 & \text{als } a \in S \\ 0 & \text{als } a \notin S \end{cases}$$

Dit betekent dat de functie 1 teruggeeft als het element a in de subset S zit, en 0 als het element a niet in S zit.

6.4 injectief, surjectief en bijectief

Een functie kan op verschillende manieren de elementen in de sets koppelen.

- **Injectief**: Elk element in A wordt gekoppeld aan een uniek element in B. Dit betekent dat er geen twee verschillende elementen in A zijn die naar hetzelfde element in B wijzen. Dus als $f(a_1) = f(a_2)$, dan moet $a_1 = a_2$ zijn.
  De functie $f : \mathbb{R} \to \mathbb{R}$ gedefinieerd door $f(x) = 2x$ is injectief, omdat verschillende waarden van x verschillende waarden van $f(x)$ opleveren, maar niet alle waarden in B worden bereikt (want, $x = 0.5$ is geen reëel getal).
  Als je een injectieve functie hebt tussen twee sets, dan kan je dat noteren als $A \preceq B$. Te lezen als: “Er is een injectieve functie van A naar B.”
- **Surjectief**: Elk element in B wordt bereikt door ten minste één element in A. Dit betekent dat de functie alle elementen in B "bedient". Dus voor elke b in B, is er minstens één a in A zodat $f(a) = b$.
  De functie $g : \mathbb{R} \to \mathbb{R}$ gedefinieerd door $g(x) = x^2$ is niet surjectief, omdat negatieve waarden in B niet worden bereikt door enige waarde van x in A.
  Als je een surjectieve functie hebt tussen twee sets, dan kan je dat noteren als $A \succeq B$. Te lezen als: “Er is een surjectieve functie van A naar B.”
- **Bijectief**: De functie is zowel injectief als surjectief. Dit betekent dat er een één-op-één correspondentie is tussen de elementen in A en B. Dus elk element in A wijst naar een uniek element in B, en elk element in B wordt bereikt door precies één element in A.
  De functie $h : \mathbb{R} \to \mathbb{R}$ gedefinieerd door $h(x) = x + 1$ is bijectief, omdat elke waarde van x een unieke waarde van $h(x)$ oplevert, en elke waarde in B wordt bereikt door precies één waarde van x.

Bij Bijectieve functies zijn belangrijk omdat ze een perfecte "match" tussen de twee sets creëren, wat betekent dat de lengte van beide sets gelijk is als ze eindig zijn. Maar dat wil je wel kunnen noteren, daarvoor is de $\simeq$ notatie. Dus als er een bijectie is tussen set A en set B, dan noteren we dit als $A \simeq B$. Dit betekent dat de cardinaliteit (grootte) van A gelijk is aan die van B. Dit werkt voor eindige sets, maar ook voor oneindige sets.

Dit is niet alleen bij bijectieve functies zo, dit geldt altijd:

- De set van dagen in de week $\{Ma, Di, Wo, Do, Vr, Za, Zo\} \simeq \{1, 2, 3, 4, 5, 6, 7\}$
- De set van natuurlijke getallen groter dan nul $\mathbb{N}_{>0} \simeq \mathbb{N}$

6.5 compositie van functies

Stel je hebt twee functies: $f : A \to B$ en $g : B \to C$. De compositie van deze twee functies, genoteerd als $g \circ f$, is een nieuwe functie die elementen van A naar C brengt door eerst f toe te passen en daarna g. Je leest dit dus als "g na f". De compositie wordt gedefinieerd als:

$$(g \circ f)(a) = g(f(a))$$

voor alle a in A. Dit betekent dat je eerst de functie f toepast op het element a om een element in B te krijgen, en vervolgens pas je de functie g toe op dat resultaat om een element in C te krijgen.

Het is goed om te weten dat als beide functies injectief zijn, dan is hun compositie ook injectief. Als beide functies surjectief zijn, dan is hun compositie ook surjectief. En als beide functies bijectief zijn, dan is hun compositie ook bijectief.

Voorbeeld **(Bijectieve functies en compositie)**:

Stel je hebt de functie $f(x) = 2x + 3$, deze is bijectief omdat elke waarde van x een unieke waarde van f(x) oplevert, en elke waarde in de codomein wordt bereikt door precies één waarde van x (ervan uitgaande dat de codomein alle reële getallen bevat). Als je wilt bewijzen dat f bijectief is, kun je de volgende stappen volgen: Je splitst de functie op in twee delen: eerst $g(x) = 2x$ en daarna $h(x) = x + 3$. Beide functies zijn bijectief:

- $g(x) = 2x$ is injectief omdat verschillende waarden van x verschillende waarden van g(x) opleveren, en het is surjectief omdat elke waarde in de codomein wordt bereikt door precies één waarde van x.
- $h(x) = x + 3$ is ook injectief en surjectief om dezelfde redenen.

Omdat zowel g als h bijectief zijn, is hun compositie $f(x) = h(g(x)) = 2x + 3$ ook bijectief.

---

7 Relaties

Een relatie is een manier om elementen van een set met elkaar te verbinden. Formeel is een relatie R op de sets A en B een subset van het Cartesisch product $A \times B$. Een relatie R tussen A en B schrijf je als ARB, zo ziet het er slecht te lezen uit, maar ">" is ook een relatie die je als $X > Y$ schrijft, dat is logischer te lezen. Je kan de relatie R ook schrijven als $R(A, B)$ of $(A, B) \in R$.

Het verschil tussen functies en relaties, is dat een functie elke input koppelt aan precies één output, terwijl een relatie meerdere outputs kan hebben voor een enkele input, of zelfs geen output. Bijvoorbeeld, de relatie "is ouder dan" tussen mensen kan meerdere mensen omvatten die ouder zijn dan een bepaalde persoon, terwijl een functie zoals "geef de leeftijd van" elke persoon koppelt aan precies één leeftijd.

De relatie R heeft ook een domein en een bereik. Het domein van R is de set van alle eerste elementen in de paren van R, en het bereik van R is de set van alle tweede elementen in de paren van R. Ofwel:
$domain(R) = \{a \in A : \exists b \in B, (a, b) \in R\}$
$range(R) = \{b \in B : \exists a \in A, (a, b) \in R\}$.

7.1 Eigenschappen van relaties

Relaties kunnen verschillende eigenschappen hebben, afhankelijk van hoe de elementen in de sets met elkaar verbonden zijn. Hier zijn enkele belangrijke eigenschappen van relaties:

- **Reflexief**: Een relatie R op een set A is reflexief als voor alle elementen a in A, het paar (a, a) in R zit. Bijvoorbeeld, de relatie "is gelijk aan" is reflexief omdat elke persoon gelijk is aan zichzelf. Als dit nooit het geval is, dan noem je het **irreflexief**.
- **Symmetrisch**: Een relatie R op een set A is symmetrisch als voor alle elementen a en b in A, als (a, b) in R zit, dan ook (b, a) in R zit. Bijvoorbeeld, de relatie "is getrouwd met" is symmetrisch omdat als persoon A getrouwd is met persoon B, dan is persoon B ook getrouwd met persoon A.
- **Antisymmetrisch**: Een relatie R op een set A is antisymmetrisch als voor alle elementen a en b in A, als (a, b) en (b, a) in R zitten, dan moet a gelijk zijn aan b. Bijvoorbeeld, de relatie "is ouder dan" is antisymmetrisch omdat als persoon A ouder is dan persoon B, en persoon B ouder is dan persoon A, dan moeten ze dezelfde leeftijd hebben (wat onmogelijk is).
- **Transitief**: Een relatie R op een set A is transitief als voor alle elementen a, b, en c in A, als (a, b) en (b, c) in R zitten, dan ook (a, c) in R zit. Bijvoorbeeld, de relatie "is een voorouder van" is transitief omdat als persoon A een voorouder is van persoon B, en persoon B een voorouder is van persoon C, dan is persoon A ook een voorouder van persoon C.

7.2 equivalentie klassen

Je kan een equivalentie relatie gebruiken om een set op te delen in verschillende subsets, die we equivalentie klassen noemen. Elke equivalentie klasse bevat elementen die allemaal met elkaar gerelateerd zijn volgens de equivalentie relatie. Bijvoorbeeld, als we de set van auto's hebben en de relatie "heeft dezelfde kleur als", dan kunnen we de auto's opdelen in equivalentie klassen op basis van hun kleur. Dus alle rode auto's vormen één equivalentie klasse, alle blauwe auto's vormen een andere klasse, enzovoort.

Deze kan je vervolgens opdelen in partities, waarbij elke element in de set precies in één subset zit. Dus in het bovenstaande voorbeeld, zou elke auto precies in één kleurklasse zitten, en deze zouden samen de hele set van auto's vormen zonder overlap. De definitie hiervan is $A_1 \cap A_2 = \emptyset$ voor alle A_1, A_2 in de partitie, waarbij $A_1 \neq A_2$. En $A_1 \cup A_2 \cup ... \cup A_n = A$.

Alles hierboven is stof voor de eerste midterm

8 Inductie

Elke inductie is omschreven met 3 stappen:

(1) De **base case**, die verteld dat er sommige elementen in een set zit
(2) De **inductieove case**, die verteld dat als een bepaalde waarde in de set zit, dat die waarde er ook bij moet. Bijv. je hebt een set met 1 iemand, en dan voeg je alle mensen toe die hij/zij kent, en die kennen ook weer mensen die je dan toe voegt etc.
(3) De **external clause**, die geeft aan dat elementen die niet toegevoegd zijn door case 1 en 2 niet worden toegevoegd.

Voor de set $\mathbb{N}$ natuurlijke nummers, gaat het als volgt:

- $0 \in \mathbb{N}$.
- voor elke $n \in \mathbb{N}$, voeg $(n + 1) \in \mathbb{N}$.
- Er zijn geen andere getallen uit $\mathbb{N}$.

Voor de set even natuurlijke nummers, gaat het als volgt:

- $0 \in \mathbb{N}$.
- voor elke $n \in \mathbb{N}$, voeg $(n + 2) \in \mathbb{N}$, wanneer $(n - 2) \in \mathbb{N}$, waar $n \neq 0$ bestaat
- Er zijn geen andere getallen uit $\mathbb{N}$.

8.1 Backus-Naur Form (BNF)

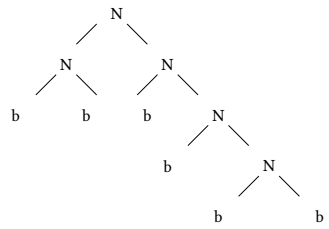
BNF is een manier om formeel de syntax van talen te definiëren. Het bestaat uit regels die aangeven hoe symbolen kunnen worden gecombineerd om geldige zinnen of expressies te vormen. Elke regel heeft een linkerkant (non-terminal symbool) en een rechterkant (combinatie van terminal en non-terminal symbolen). De notatie gebruikt meestal de volgende symbolen:

- **Non-terminal symbolen**: Dit zijn symbolen die verder kunnen worden uitgebreid of vervangen door andere symbolen. Ze worden meestal geschreven in hoekige haken, zoals <expression>, <term>, etc.
- **Terminal symbolen**: Dit zijn de basis symbolen die niet verder kunnen worden uitgebreid. Ze vertegenwoordigen de daadwerkelijke tekens of woorden in de taal, zoals cijfers, letters, of specifieke keywords.

- **Productie regels**: Dit zijn de regels die aangeven hoe non-terminal symbolen kunnen worden vervangen door combinaties van terminal en non-terminal symbolen. Ze worden meestal geschreven in de vorm: <non-terminal> ::= <combinatie van symbolen>.

Een goed voorbeeld is een binary tree:

```
<Node> ::= b | N(<Node>, <Node>)
```



Dit betekent dat een <Node> ofwel een blad (b) is, of een knoop (N) met twee sub-nodes.

Een ander voorbeeld is een eenvoudige BNF-definitie voor een taal die bestaat uit getallen en optellingen:

```
<expression> ::= <term> | <term> + <expression>
<term> ::= <digit> | <digit> <term>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Deze regels geven aan dat een <expression> kan bestaan uit een <term>, of een <term> gevolgd door een '+' en nog een <expression>. Een <term> kan bestaan uit een <digit> of een <digit> gevolgd door nog een <term>. Een <digit> kan een van de cijfers 0 tot 9 zijn.

Zo kan je een hele simpele programmeertaal definiëren met BNF.

```
<program> ::= x:=<e> | <program>; <program> | if <e> then <program> else <program> | while <b> do <program>
<e> ::= n | x | e + e | e - e | e * e | e / e
<b> ::= true | false | e = e | e <= e | not b | b and b | b or b
```

Hierboven is een simpele programmeertaal gedefinieerd met toewijzingen, sequenties, conditionals en loops. Expressies kunnen getallen, variabelen en basis rekenkundige operaties bevatten. Booleaanse expressies kunnen waarheidswaarden, vergelijkingen en logische operaties bevatten

8.2 inductief gedefineerde functies

Net zoals je inductief een set kan definiëren, kan je ook inductief een functie definiëren. Dit doe je door een basis geval te definiëren en een inductie stap. Bijvoorbeeld, de factorial functie kan je zo definiëren:

$$0! = 1; \quad \text{(basis geval)}$$
$$n! = n \times (n-1)! \quad \text{voor } n > 0 \quad \text{(inductie stap)}$$

Bijvoorbeeld, om 5! te berekenen, gebruik je de inductieve definitie:

$$\begin{aligned} 5! &= 5 \times 4! \\ &= 5 \times (4 \times 3!) \\ &= 5 \times (4 \times (3 \times 2!)) \\ &= 5 \times (4 \times (3 \times (2 \times 1!))) \\ &= 5 \times (4 \times (3 \times (2 \times (1 \times 0!)))) \\ &= 5 \times (4 \times (3 \times (2 \times (1 \times 1)))) \\ &= 120 \end{aligned}$$

8.3 Recursieve functies

Recursieve functies zijn functies die zichzelf aanroepen om een probleem op te lossen. Je schrijft het als volgt:

$$f(n) = \begin{cases} 1, & \text{als } n \leq 0; \\ f(n/2), & \text{als } n > 1 \text{ even}; \\ f(3n+1), & \text{als } n > 1 \text{ oneven}. \end{cases}$$

In dit voorbeeld is de functie f gedefinieerd met drie gevallen:

- Als n kleiner dan of gelijk aan 0 is, retourneert f de waarde 1.
- Als n groter dan 1 is en even, roept f zichzelf aan met de waarde n gedeeld door 2.
- Als n groter dan 1 is en oneven, roept f zichzelf aan met de waarde 3n + 1.

Recursieve functies zijn krachtig omdat ze complexe problemen kunnen oplossen door ze op te splitsen in kleinere, beter beheersbare subproblemen. Het is echter belangrijk om ervoor te zorgen dat recursieve functies een basisgeval hebben om oneindige lussen te voorkomen.

9 Games

Een paar definitie's:

- **Game of chance**, Zoals roulette of dobbelstenen, waarbij de uitkomst volledig afhankelijk is van toeval.
- **Game of no chance**, Zoals boter-kaas-en-eieren of schaken, waarbij de uitkomst volledig afhankelijk is van de beslissingen van de spelers.
- **Perfect information**, Beide spelers weten alles wat er te weten valt over de staat van het spel op elk moment.
- **Winning strategy**, Een strategie die ervoor zorgt dat een speler altijd wint, ongeacht wat de tegenstander doet.
- **Drawing strategy**, Een strategie die ervoor zorgt dat een speler minstens gelijk speelt, ongeacht wat de tegenstander doet.
- **lossing strategy**, Een strategie die ervoor zorgt dat een speler altijd verliest, ongeacht wat de tegenstander doet.

De rest van de stof is te vinden in hoofdstuk 10 van het boek, of slide 12.

10 Natural deduction

10.1 Inference rules

Deze relaties kun je door middel van een formule weergeven. Bijvoorbeeld: $\frac{}{0 \leq n} \leq -Base$ en $\frac{n \leq m}{s(n) \leq s(m)} \leq -Step$ Als je hier wilt bewijzen dat $2 \leq 5$, dan kan je dat zo doen:

$$\frac{\frac{\frac{}{0 \leq s(s(s(0)))} \leq -Base}{s(0) \leq s(s(s(s(0))))} \leq -Step}{s(s(0)) \leq s(s(s(s(0))))} \leq -Step$$

Je zet als het ware elke stap in een breuk, waarbij de bovenkant de premissen zijn en de onderkant de conclusie. En dit Kan je "nesten" totdat je bij de stelling komt die je wilt bewijzen.

Je kan dit ook in een tabel zetten:

Stelling	Reden
$0 \leq s(s(s(0)))$	$\leq -Base$
$s(0) \leq s(s(s(s(0))))$	$\leq -Step$ op 1
$s(s(0)) \leq s(s(s(s(s(0)))))$	$\leq -Step$ op 2

Hoe werkt dit nou? Het is de bedoeling dat je met de regels die je hebt, een stelling kunt bewijzen. Elke regel heeft een aantal premissen (boven de streep) en een conclusie (onder de streep). Als je alle premissen kunt bewijzen, dan mag je de conclusie aannemen. Dit doe je net zolang totdat je de stelling die je wilt bewijzen hebt bereikt. De rede dat ik gelijk $0 \leq s(s(s(0)))$ mag aannemen is omdat dit de basis is van de inductie. $s(0)$ is (per definitie) wat komt ná 0, dus $s(0) = 1$, daarom is $0 \leq s(0)$ altijd waar en mag je dat aannemen.

Voorbeeld **(Even en Oneven getallen)**:

Stel we defineren de Base en Step regels voor Even en Oneven getallen als volgt:

$$\frac{}{\text{isEven}(0)}\text{isEven-Base}$$
$$\frac{isEven(n)}{isEven(s(s(n)))}\text{isEven-Step}$$

Hieruit kan je bijvoorbeeld opmaken dat $s(s(s(0)))$ even is (makkelijker in tabelvorm):

Stelling	Reden
$isEven(0)$	$isEven - Base$
$isEven(s(s(0)))$	$isEven - Step$ op 1
$isEven(s(s(s(s(0)))))$	$isEven - Step$ op 2

Nog een voorbeeld

Voorbeeld **(isSorted)**:

Stel we defineren 3 regels voor isSorted (isSorted-empty, isSorted-singleton, isSorted-step) als volgt:

$$\frac{}{\text{isSorted}([])}\text{isSorted-empty}$$
$$\frac{}{\text{isSorted}(n : [])}\text{isSorted-singleton}$$
$$\frac{n \leq m \quad \text{isSorted}(m : w)}{\text{isSorted}(n : m : w)}\text{isSorted-step}$$

Hieruit kan je bijvoorbeeld opmaken dat de lijst $[0, 2, 5, 7]$ gesorteerd is (makkelijker in tabelvorm):

$$\dfrac{\dfrac{\dfrac{}{isSorted([])}\,isSorted-empty}{isSorted(7:[])}\,isSorted-singleton}{isSorted(5:7:[])}\,isSorted-step$$
$$isSorted(2:5:7:[])$$

Stelling	Reden
$isSorted([])$	$isSorted-empty$
$isSorted(7:[])$	$isSorted-singleton$
$isSorted(5:7:[])$	$isSorted-step$ op 2, omdat $5 \leq 7$
$isSorted(2:5:7:[])$	$isSorted-step$ op 3, omdat $2 \leq 5$
$isSorted(0:2:5:7:[])$	$isSorted-step$ op 4, omdat $0 \leq 2$

Maar wat nou als n **niet** kleiner is dan m (ofwel de lijst is niet gesorteerd)? Dan kan je dat ook bewijzen (op de lijst $[0, 2, 7, 5]$):

Stelling	Reden
$isSorted([])$	$isSorted-empty$
$isSorted(5:[])$	$isSorted-singleton$
(geen regel toepasbaar op $isSorted(7:5:[])$)	$7 \not\leq 5$

10.2 natuurlijke deductie in propositielogica (Derivation)

Als we $P \implies Q$ wilt bewijzen, dan neem je aan dat P waar is, en dan bewijs je Q. Maar P is alleen een geldige aannname binnen de context van dit specifieke bewijs en niet daarbuiten.

Om dit te bewijzen moeten we dus ook buiten de context van P kunnen bewijzen dat P waar is. Dit wordt vervolgens bijgehouden in een Set, een set met alle aannames die we hebben gemaakt. Zo kan je dus in de set zetten: $\Gamma \vdash P$ wat in woorden "Uit de aannames in de set Γ volgt dat P waar is" betekend. Je kan tijdens het bewijzen ook nieuwe aannames maken, deze worden dan toegevoegd aan de set.

Als je die breuk-achtige dingen maakt, dan zijn het zegmaar herschrijf regels. Je hebt dan allemaal regels die al kant en klaar zijn en die moet je dan invullen met de data die je hebt dus bijvoorbeeld:
$\frac{PQ}{P \wedge Q}$, P moet waar zijn en Q moet waar zijn en dan is P en Q waar. Maar je kan voor P bijvoorbeeld ook $P = A \vee B$ invullen, en dan moet je dus bewijzen dat $A \vee B$ waar is. dan krijg je een nieuwe regel: $\frac{A \vee BQ}{(A \vee B) \wedge Q}$ Die A of B is dan op zichzelf ook weer een breuk en zo gaat dat door.

Een lijst van alle regels van natuurlijke deductie in propositielogica:

$$\frac{P \in \Gamma}{\Gamma \vdash P}\text{-Assumption}$$
"Als P in de aannameset zit, dan volgt P"

$$\frac{}{\Gamma \vdash \top}\text{T-Introduction}$$
"Waarheid is altijd waar"

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash P}\text{F-Elimination}$$
"Als je een onwaarheid hebt, kan je alles bewijzen"

$$\frac{\Gamma, P \vdash Q}{\Gamma \vdash P \implies Q}\text{Implication-Introduction}$$
"Als je vanuit P kan bewijzen dat Q waar is, dan volgt dat P impliceert Q"

$$\frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q}\text{And-Introduction}$$
"Als P waar is en Q waar is, dan is P en Q waar"

$$\frac{\Gamma \vdash P \implies Q \quad \Gamma \vdash P}{\Gamma \vdash Q}\text{Implication-Elimination}$$
"Als P Q impliceert, en P waar is, dan volgt Q"

$$\frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash P}\text{And-Elimination-1}$$
"Als P en Q waar is, dan is P waar"

$$\frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash Q}\text{And-Elimination-2}$$
"Als P en Q waar is, dan is Q waar"

$$\frac{\Gamma \vdash P}{\Gamma \vdash P \vee Q}\text{Or-Introduction-1}$$
"Als P waar is, dan is P of Q waar"

$$\frac{\Gamma \vdash Q}{\Gamma \vdash P \vee Q}\text{Or-Introduction-2}$$
"Als Q waar is, dan is P of Q waar"

$$\frac{\Gamma, P \vdash R \quad \Gamma, Q \vdash R}{\Gamma \vdash P \vee Q \implies R}\text{Or-Elimination}$$
"Als vanuit P kan bewijzen dat R waar is, en vanuit Q kan bewijzen dat R waar is, dan volgt dat P of Q R impliceert"

$$\frac{\Gamma \vdash P}{\Gamma \vdash \neg\neg P}\text{Negation-Introduction}$$

"Als P waar is, dan is niet niet P waar"

$$\frac{\Gamma \vdash \neg\neg P}{\Gamma \vdash P}\text{Negation-Elimination}$$

"Als niet niet P waar is, dan is P waar"

$$\frac{\Gamma, P \vdash Q \quad \Gamma, Q \vdash \bot}{\Gamma \vdash P \Leftrightarrow Q}\text{Biconditional-Introduction}$$

"Als vanuit P kan bewijzen dat Q waar is, en vanuit Q kan bewijzen dat onwaarheid waar is, dan volgt dat P als en slechts als Q"

$$\frac{\Gamma \vdash P \Leftrightarrow Q}{\Gamma \vdash P \implies Q}\text{Biconditional-Elimination-1}$$

"Als P als en slechts als Q waar is, dan volgt dat P Q impliceert"

$$\frac{\Gamma \vdash P \Leftrightarrow Q}{\Gamma \vdash Q \implies P}\text{Biconditional-Elimination-2}$$

"Als P als en slechts als Q waar is, dan volgt dat Q P impliceert"

$$\frac{\Gamma, \neg P \vdash \bot}{\Gamma \vdash P}\text{Proof by Contradiction}$$

"Als vanuit niet P kan bewijzen dat onwaarheid waar is, dan volgt dat P waar is"

Dit zijn alle natuurlijke deductie regels die je nodig hebt om propositielogica te bewijzen. Deze kan je vervolgens samenvoegen om complexere stellingen te bewijzen.

Voorbeeld **($P \wedge Q$)**:

---

11 Redenering over progamma's

In dit gedeelte behandelen we hoe we formeel kunnen redeneren over computerprogramma's.

Hier heb je verschillende soorten semantiek:

- σ: een staat, die de waarden van alle variabelen op een bepaald moment in het programma vertegenwoordigt. Je schrijft dit als een functie die variabelen naar hun waarden mappt.
- $\langle p, \sigma\rangle \rightarrow \sigma'$: Dit is de operationele semantiek, die beschrijft hoe een programma p wordt uitgevoerd in een staat σ en resulteert in een nieuwe staat σ'.
- $[[p]](\sigma) = \sigma'$: Dit is de denotationale semantiek, die een programma p beschrijft als een functie die een staat σ naar een nieuwe staat σ' mapt.

Zoals je hebt gezien bij natuurlijke deductie, kan je ook regels opstellen voor programmeertalen.

$$\frac{[\![e]\!](\sigma) = n}{\langle x := e, \sigma\rangle \rightarrow \sigma[x \mapsto n]}\text{Assign}$$

Deze regel zegt dat als de expressie e in de staat σ evalueert naar de waarde n, dan zal het toewijzingscommando $x := e$ in de staat σ resulteren in een nieuwe staat σ' waarin de variabele x de waarde n heeft.

$$\frac{[\![b]\!](\sigma) = true \quad \langle p, \sigma\rangle \rightarrow \sigma'}{\langle \text{if } b \text{ then } p \text{ else } q, \sigma\rangle \rightarrow \sigma'}\text{If-True}$$

Deze regel zegt dat als de booleaanse expressie b in de staat σ evalueert naar waar, en het commando p in de staat σ resulteert in een nieuwe staat σ', dan zal het if-commando in de staat σ ook resulteren in de staat σ'.

$$\frac{[\![b]\!](\sigma) = false \quad \langle q, \sigma\rangle \rightarrow \sigma'}{\langle \text{if } b \text{ then } p \text{ else } q, \sigma\rangle \rightarrow \sigma'}\text{If-False}$$

Deze regel zegt dat als de booleaanse expressie b in de staat σ evalueert naar onwaar, en het commando q in de staat σ resulteert in een nieuwe staat σ', dan zal het if-commando in de staat σ ook resulteren in de staat σ'.

Voorbeeld **(een conditional uitvoeren)**:

Stel dat we de volgende staat hebben: $\sigma(x) = 3, \quad \sigma(y) = 10$ En het volgende programma: `if x < y then z := x + 1 else z := y + 1`
Als we hier de derivatie regels op toepassen, krijgen we het volgende:

$$\frac{[\![x < y]\!](\sigma) = true \quad \frac{[\![x]\!](\sigma)=3}{\langle r:=x,\sigma\rangle \rightarrow \sigma[r\mapsto 3]}}{\langle \text{if } x < y \text{ then } r := x \text{ else } r := y, \sigma\rangle \rightarrow \sigma[r \mapsto 3]}$$

---

12 Hoare logica

Hoare logica is bedoeld om formeel te redeneren over de correctheid van computerprogramma's. Het maakt gebruik van zogenaamde Hoare triples, die de vorm hebben: $\{P\}C\{Q\}$. Hierin is P de preconditie, C het commando (of programma), en Q de postconditie. De betekenis van een Hoare triple is dat als de preconditie P waar is voordat het commando C wordt uitgevoerd, dan zal de postconditie Q waar zijn nadat C is uitgevoerd, op voorwaarde dat C correct wordt uitgevoerd (d.w.z. zonder fouten of uitzonderingen).
$\{P\}C\{Q\} \leftarrow$ "Voor elke staat σ die voldoet aan P, als we $\langle p, \sigma\rangle$ uitvoeren (het commando p in de staat σ) en dit leidt tot een nieuwe staat τ, dan voldoet τ' aan Q."

Voorbeeld **(Hoare triple)**:

Stel we hebben de volgende Hoare triple:

$$\{x = 3\} \quad y := x + 1 \quad \{y = 4\}$$

Deze triple zegt dat als de preconditie $x = 3$ waar is voordat we het commando $y := x + 1$ uitvoeren, dan zal de postconditie $y = 4$ waar zijn nadat het commando is uitgevoerd. Laten we dit stap voor stap bekijken:

- Preconditie: We beginnen met de aanname dat $x = 3$.
- Commando: We voeren het commando $y := x + 1$ uit. Dit betekent dat we de waarde van x (die 3 is) nemen en er 1 bij optellen, waardoor y de waarde 4 krijgt.
- Postconditie: Na het uitvoeren van het commando, hebben we $y = 4$. Dit voldoet aan de postconditie die we wilden bewijzen.

Dus de Hoare triple is correct, omdat de preconditie leidt tot de postconditie na het uitvoeren van het commando.
Een ander voorbeeld is de volgende Hoare triple:

$$\{x = A \wedge y = B\} \quad z := x; x := y; y := z \quad \{x = B \wedge y = A\}$$

Deze triple beschrijft een commando dat de waarden van x en y verwisselt. Laten we dit stap voor stap bekijken:

- Preconditie: We beginnen met de aanname dat $x = A$ en $y = B$.
- Commando: We voeren het commando $z := x; x := y; y := z$ uit. Dit betekent dat we eerst de waarde van x opslaan in een tijdelijke variabele z, vervolgens de waarde van y toewijzen aan x, en ten slotte de waarde van z (die oorspronkelijk x was) toewijzen aan y.
- Postconditie: Na het uitvoeren van het commando, hebben we $x = B$ en $y = A$. Dit voldoet aan de postconditie die we wilden bewijzen.

Dus de Hoare triple is correct, omdat de preconditie leidt tot de postconditie na het uitvoeren van het commando.
Een laatste voorbeeld is de volgende Hoare triple:

$$\{true\}\text{while true do p := 0 od}\{p = 500\}$$

Deze triple beschrijft een oneindige lus die altijd het commando $p := 0$ uitvoert. Laten we dit stap voor stap bekijken:

- Preconditie: We beginnen met de aanname dat de preconditie altijd waar is (aangeduid door $true$).
- Commando: We voeren het commando $whiletruedop := 0$ od uit. Dit betekent dat we een oneindige lus hebben die altijd het commando $p := 0$ uitvoert.
- Postconditie: Na het uitvoeren van het commando, hebben we $p = 500$. Dit voldoet niet aan de postconditie die we wilden bewijzen, omdat p altijd 0 zal zijn na het uitvoeren van het commando.

Dus de Hoare triple is niet correct, omdat de preconditie niet leidt tot de postconditie na het uitvoeren van het commando.