Logica voor Informatici

Marc Denecker

22 september 2023

Bij de samenstelling van deze cursus werd gebruik gemaakt van verschillende andere cursussen: in de eerste plaats van de cursus van Prof. Jan Denef, die dit vak gaf tot in 2010, en ook van de cursus *CSC330 Logical Specifications* van Prof. Hector Levesque van de universiteit van Toronto.

De software LogicPalet die gebruikt wordt in deze cursus werd ontwikkeld door Jan Denef. Eén van de componenten van dit systeem is het tool Geo-Worlds, dat geïnspireerd is op het tool Tarski-Worlds dat hoort bij het boek van Barwise en Etchemendy: Language, Proof and Logic, CSLI publications (2002). LogicPalet biedt wel veel meer mogelijkheden dan Tarksi-Worlds. In LogicPalet wordt ook gebruik gemaakt van twee logische redeneersystemen: de theoremprover Spass van het Max Planck Institute for Computer Science, en de modelgenerator IDP die ontwikkeld wordt in de KRR-groep van Marc Denecker.

Aanbevolen literatuur

- J. Barwise, J. Etchemendy, *Language, Proof and Logic,* CSLI publications (2002)
- J. van Benthem, H. van Ditmarsch, J. Ketting, W. Meyer-Viol, *Logica voor informatici,* Addison Wesley Nederland (1994).
- Jeremy Avigad, Robert Y. Lewis, Floris van Doorn, Logic and Proof, `https://avigad.github.io/logic_and_proof/`, Cursus aan Vrije Universiteit Amsterdam.

Inhoudsopgave

1 Inleiding: logica en zijn rol in informatica **6**

1.1 De rol van logica in de informatica 6

1.2 Korte geschiedenis van de logica 7

1.3 Toepassingen van logica in de Informatica 8

1.4 Waarom vertrekken van predicatenlogica? 11

1.5 Notaties en basisbegrippen 12

1.6 Verzamelingen en relaties 13

1.7 Inductieve definities, inductief bewijs 16

2 De predikatenlogica **24**

2.1 Inleiding tot de predikatenlogica 24

2.2 Syntax van predikatenlogica 31

2.3 Formele semantiek van predikatenlogica 40

2.4 De informele semantiek van Predicatenlogica 51

2.5 Berekenen en bewijzen van waarheid in een structuur 55

2.6 Construeren van modellen van een zin 61

2.7 Geo-werelden en Decawerelden 63

2.8 Definities 65

2.9 Pragmatiek van predikatenlogica 68

2.9.1 Functiesymbolen stellen totale functies voor 70

2.9.2 Kwantificeren over een klasse 71

2.9.3 Veralgemeende kwantoren in natuurlijke taal 72

2.9.4 Implicaturen van definities 75

2.9.5 Implicaturen van namen 75

2.9.6 Het domein uitdrukken 76

2.9.7 De materiële implicatie. 76
2.10 Isomorfisme 77

3 Logisch gevolg en redeneren 81
3.1 Logisch waar, logisch gevolg 81
3.2 Propositielogica 86
3.3 Wetten van het denken van propositionele logica 91
3.3.1 Normaalvormen van propositionele logica 95
3.4 Wetten van het denken van predikatenlogica 97
3.4.1 Normaalvormen van predicatenlogica 109
3.5 Een formele bewijsmethode voor predikatenlogica 110
3.6 Samenvatting 123

4 Modelleren en redeneren in informatica-toepassingen 124
4.1 Modelleren in logica: inleiding 125
4.2 Oplossen van informaticaproblemen met behulp van inferentie 128
4.3 Databanken revisited 130
4.3.1 Zinvolle en zinloze databanken en Integriteitsbeperkingen. 131
4.3.2 Queries modelleren en oplossen 132
4.3.3 Een databank als een logische theorie 139
4.3.4 SQL 144
4.4 De oudste logische theorie: de Peano axioma's 147
4.5 Zoekproblemen en modelgeneratie 152
4.6 Automated theoremproving (ATP) 156
4.7 Formele methoden in Software Engineering 157
4.8 Conclusie 164

5 Algoritmes, Berekenbaarheid, het Halting problem, Onbeslisbaarheid en de Onvolledigheidsstelling van Gödel 165
5.1 Registermachines, Berekenbaarheid en de hypothese van Church 166
5.2 Onbeslisbaarheid van het stop-probleem 170
5.3 Berekenbaarheid van inferentieproblemen 173
5.3.1 Waarheid van formules berekenen in een eindige structuur 175

5.3.2 Waarheid van formules over $\mathbb{N}$: onbeslisbaarheid van $\mathbb{N}$ 178
5.3.3 Berekenbaarheid van modelgeneratie en modelexpansie 179
5.3.4 Semi-onbeslisbaarheid van deductie in predikatenlogica. 180
5.4 De Onvolledigheidsstelling van Gödel 183
5.5 Conclusie 185

A Appendix **187**
A.1 Meer over SQL en logica 187

Hoofdstuk 1

Inleiding: logica en zijn rol in informatica

1.1 De rol van logica in de informatica

Deze cursus gaat over logica en het gebruik ervan in de informatica.

Het woord *logica* heeft verschillende betekenissen afhankelijk van de context waarin het gebruikt wordt. "Een logica" is een kunstmatige *formele taal*, t.t.z.[1] een taal waarvan de syntactische vorm en de betekenis van de uitdrukkingen op wiskundige manier vastgelegd wordt. In een andere context betekent logica ook *de kunst van het redeneren*.

Een wiskundige taal versus redeneerkunst: dat is wel erg verschillend! Het verband is echter: om te redeneren heb je *informatie* nodig. Normaal gezien drukken we informatie uit in natuurlijke taal (het Nederlands bv.). Maar om de wetten van het correct redeneren te bestuderen op een formeel wetenschappelijke manier, voldoet natuurlijke taal niet. Immers natuurlijke taal is soms vaag, onvolledig of dubbelzinnig. Dus is een wiskundige taal nodig om informatie in voor te stellen: een taal waarvan de syntax en de semantiek op wiskundige wijze beschreven wordt. Op die manier ging de studie van de redeneerkunst dus hand in hand met de ontwikkeling van formele talen (zie sectie 1.2 voor een kort overzicht van de geschiedenis). In deze cursus zullen we vertrekken van twee van de belangrijkste en de oudste formele talen, namelijk propositielogica en predicatenlogica.

Logica omvat dus niet alleen de wetenschappelijke studie van het redeneren maar ook van "informatie". En informatie is waar alles om draait in de *Informatica*. Een centraal thema van de Informatica is: informatie, en hoe je met informatie in computersystemen omgaat. Informatie is een soort *grondstof* die mensen gebruiken om problemen op te lossen en taken uit te voeren. Informatici schrijven programma's om zo'n taken te laten uitvoeren door een computer. Om die programma's te kunnen schrijven moeten zij beschikken over voldoende basisinformatie.

Bijvoorbeeld, veronderstel dat we een softwaresysteem willen bouwen om uurroosters aan de KU Leuven op te slaan, te corrigeren, te berekenen, op correctheid te testen. Een programmeur die hieraan meewerkt moet het volgende weten: *(a) noch studenten, noch docenten kunnen twee hoorcolleges tergelijkertijd bijwonen, (b) in 1 auditorium kunnen geen twee hoorcolleges tergelijkertijd plaats vinden , (c) de capaciteit van het auditorium moet groter zijn dan het aantal*

[1] t.t.z. : afkorting van "t is te zeggen, dat wil zeggen".

studenten die het hoorcollege volgen, enz. De items (a), (b), (c), enz. zijn stukjes informatie waarmee de programmeur rekening moet houden, en die hij in zo'n uurroosterprogramma's moet verwerken, bv. in een programma dat de correctheid van uurroosters test, of dat correcte uurroosters genereert.

Aan het eind van deze cursus zal het duidelijk zijn hoe deze informatie in logica kan uitgedrukt worden. We zullen zien welke vormen van redeneren vereist zijn om dergelijke problemen op te lossen. Verder, dat er logische redeneersystemen zijn die dit soort redeneren kunnen uitvoeren en dus in staat zijn dergelijke problemen op te lossen op basis van de in logica uitgedrukte informatie.

Wat we hier niet zullen bestuderen is hoe een programmeur deze informatie gebruikt om een programma op te stellen (bv. om uurroosters op te stellen). Op dit punt staat de wetenschap nog niet ver. Hoe programmeurs precies informatie omzetten in een programma dat de gestelde taak oplost wordt nog altijd niet goed begrepen. Daarom zegt men soms:"programmeren is een kunst, geen ingenieursdiscipline".

Logica bestudeert dus informatie en welke vormen van redeneren nodig zijn om problemen en taken op te lossen. Op die manier is logica één van de fundamentele theoretische pijlers van de informatica.

1.2 Korte geschiedenis van de logica

Er gebeurt zeer veel onderzoek naar logica, niet alleen in de Informatica maar ook in filosofie en in wiskunde. Dat is waar de oorsprong van logica ligt.

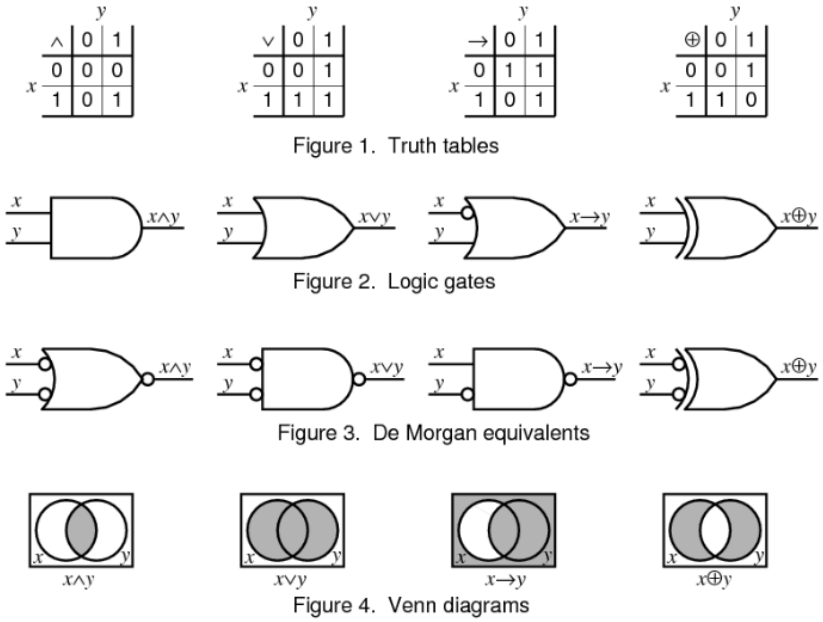
De Griekse wijsgeer *Aristoteles* (384-322 v.C.) was de eerste die de regels van het correct redeneren ontleedde. Hij heeft geen formele taal ontwikkeld maar is bekend omwille van zijn redeneerregels, *syllogismen* genaamd, zoals bv.

$$\frac{\begin{array}{c}\text{Alle grieken zijn sterfelijk}\\ \text{Socrates is een griek}\end{array}}{\text{Socrates is sterfelijk}}$$

Leibniz (Duits wiskundige en wijsgeer (1646-1716)) stelde voor de regels van het redeneren met wiskundige middelen te bestuderen. Hij was van oordeel dat complexe "gedachten" (waarmee ongeveer hetzelfde bedoeld wordt als "informatie" in de vorige sectie) konden samengesteld worden uit eenvoudiger gedachten met behulp een soort wiskundige operatoren gelijkaardig aan rekenkundige operatoren $+$, $\times$. Zelf heeft hij dat idee nooit volledig uitgewerkt.

Op het einde van de 19^{de} eeuw begonnen *Boole, Peano, Frege, Russell*, en anderen met de *wiskundige* studie van de redeneerregels (*symbolische* of *mathematische logica*) (voordien was de logica het domein van wijsgeren). Hun doel was de wiskunde op meer exacte wijze te funderen, om zo een aantal *paradoxen* weg te werken die te wijten waren aan een te slordige fundering (*Grondslagen* onderzoek in de wiskunde).

In 1879 publiceerde Frege het *"Begriffschrift"*, wat nu gekend is als de eerste versie van *predicatenlogica*. In predicatenlogica beschikt men over een zeer beperkte set van operatoren: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \forall, \exists$. Ze laten toe om complexe "gedachten" te construeren uit eenvoudiger "gedachten", zoals Leibniz in gedachten had. Met deze kleine verzameling van operatoren kunnen verbazingwekkend veel "gedachten" uitgedrukt worden, alhoewel dit niet altijd zo eenvoudig is,



Figuur 1.1: Booleaanse Algebra en circuits

en hoewel er ook wel nuttige gedachten zijn die niet met deze operatoren uitgedrukt kunnen worden.

Sinds *Gödel* (1930) is de "wiskundige logica" *meer* dan de studie der redeneerregels en grondslagenonderzoek (zie Hoofdstuk 5). De "wiskundige logica" heeft implicaties voor de filosofie, zuivere wiskunde, computerwetenschappen, kunstmatige intelligentie en de taalkunde.

Van meet af aan stond logica aan de wieg van de informatica. *Alan Turing*, één van de uitvinders van de computer, was een logicus. *John McCarthy*, vader van de artificiële intelligentie, stelde voor logica als basistaal van intelligente machines te gebruiken. *Edgar Codd* ontwikkelde de relationele databanken op basis van logica. *Hoare* gebruikte logica om over de correctheid van programma's te redeneren, enz. In vrijwel alle domeinen van de informatica speelt logica een basisrol, op zijn minst bij de ontwikkeling van de theoretische grondslagen ervan maar ook steeds meer bij concrete toepassingen, zoals we zonet gezien hebben.

Belangrijke toepassingen in de informatica zijn zoals gezegd computers zelf (gebaseerd op boolse algebra), databanken om data op te slaan en op te vragen, en in het algemeen vele softwaretools die het mogelijk maken complexe informatie op te slaan en computers ermee te laten redeneren, problemen op te lossen, of de correctheid van programma's te bewijzen.

1.3 Toepassingen van logica in de Informatica

Hieronder volgt een (onvolledig) overzicht van de belangrijkste gebieden waar logica effectief gebruikt wordt in de informatica.

Booleaanse algebra is een andere benaming voor propositielogica en wordt intensief gebruikt in de electronica, om circuits en processoren te ontwerpen. De ontwikkeling van computers en andere electronische hardware is ondenkbaar zonder booleaanse algebra.

Logici waren dan ook nauw betrokken bij de ontwikkeling van de eerste computers. In de tweede wereldoordlog ontsluierden Poolse logici de principes van Enigma, de Duitse geheime code en droegen hun kennis over aan de engelse geheime dienst. Een spilfiguur in het Engelse team dat de eerste electronische machine (de "Colosus") bouwde om geheime berichten te decoderen was Alan Turing, een logicus. Alan Turing legde ook de basis voor een andere fundamentele pijler van de informatica, namelijk de theorie van berekenbaarheid. De A.M. Turing Award wordt beschouwd als de nobelprijs van de informatica. Ook von Neumann, naar wie de hedendaagse von Neumann computerarchitectuur genoemd wordt, was een logicus en wiskundige.

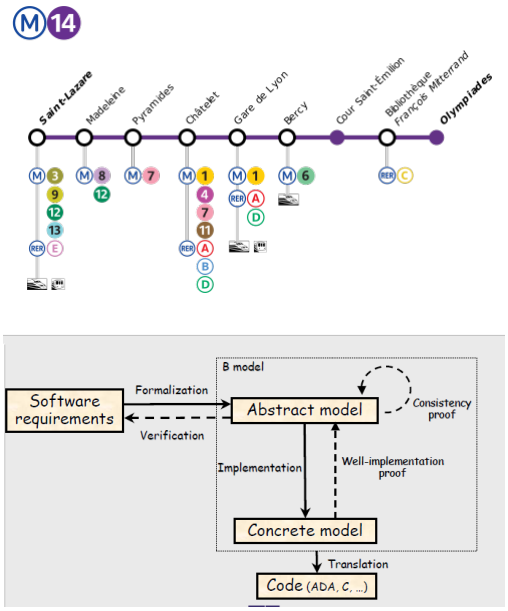
Relationele databanken zijn van de meest gebruikte informaticasystemen en kunnen beschouwd worden als logische redeneersystemen. De ontwikkeling ervan begon met het werk van Codd in 1970 dat gebaseerd was op logica. Zoals we zullen zien in deze cursus zijn queries aan hedendaagse databanken te beschouwen als logische uitdrukkingen (weliswaar niet in de predikatenlogica). Databanken implementeren een eenvoudige maar belangrijke vorm van redeneren. Op die manier zijn het tegenwoordig de meest courante softwaretoepassingen van logica.

Het semantische web is het idee om het internet te gebruiken als een gedistribueerde kennisbank waarin data en complexe kennis kunnen worden gedeeld, geïntegreerd en hergebruikt, en waarover complexe logische vragen en problemen kunnen worden opgelost.

Formele modellering en specificatie, verificatie. Een eerste stap in het ontwikkelen van software is het opstellen van *specificaties*. Verschillende aspecten kunnen gemodelleerd worden: belangrijke achtergrondkennis, concepten en informatie uit het probleemdomein, of gedetailleerde beschrijvingen van *wat* de software moet doen, en alle informatie die daarvoor van belang is. Meestal in natuurlijke taal, maar in toenemende mate worden deze specificaties in een logica geformuleerd waardoor redeneersystemen gebruikt kunnen worden om de eigenschappen van het systeem na te gaan (verificatie), of om het te simuleren. Een speciaal geval is de **automatische programmaverificatie.** In dit domein worden de eigenschappen van de input en output van een programma beschreven door middel van logische formules: de eigenschappen van de input noemt men precondities, die van de output postcondities. Men ontwikkelt automatische bewijssystemen om de correctheid van programma's bewijzen: als de preconditie voldaan is voor oproep, dan zal de postconditie voldaan zijn na de oproep. Een voorbeeld hiervan is de taal spec# van Microsoft of het Verifast tool dat hier aan het departement Computerwetenschappen van de KUL ontwikkeld wordt.

(Semi-)automatische programmageneratie genereert software uit formele specificaties, weliswaar vooralsnog met intense menselijke interactie. Een voorbeeld is de software van de automatische metro-lijn 14 in Parijs die op deze manier door Siemens werd ontwikkeld (zie Figuur 1.2). Siemens manier van werken bestond erin om (manueel) een serie van steeds concretere wordende formele specificaties te bouwen. De correctheid van elke verfijnde specificatie ten opzichte van de vorige werd bewezen door een automatisch systeem. Tenslotte bekwam men een specificatie die kon vertaald worden tot een programma. Dit was een zeer arbeidsintensieve manier van werken (meer lijnen logica dan het uiteindelijke programma), maar een dergelijk systeem moest dan ook bug-vrij zijn. Sinds de indienststelling van de metro in 1998 werd geen enkele softwarefout vastgesteld!

Het is vooralsnog een grote investering om deze logische methodes toe te passen. In de huidige stand van zaken is de toepassing ervan voorlopig beperkt tot *kritische* software-applicaties waar bugs een dramatische impact zouden kunnen hebben op mensen of bedrijven en dus ten alle prijze vermeden moeten worden (bv. het metro-systeem, software voor medische instrumenten, besturingssoftware voor raketten, besturingssystemen, enz). Maar de redeneersystemen worden krachtiger en het gebruik ervan eenvoudiger, en het gebruik neemt toe.



- Parijse Metro 14: volautomatisch, in dienst sinds 1998
- Ontwikkeld door Siemens met formele methode: B-method
- Interactief proces: een serie van steeds concreter formele specificaties
- De correctheid van elke volgende specificatie werd automatisch bewezen ten opzichte van de vorige.
- Het (meest concrete) resultaat wordt automatisch omgezet naar een programma (Ada/C/Java).
- Arbeidsintensief: 110.000 lijnen in de B taal → 86.000 lijnen Ada.
- Er werd tot nog toe geen enkele softwarefout vastgesteld. De software Metro14.1.0 was nog steeds operationeel bij het verschijnen van deze paper: `http://deploy-eprints.ecs.soton.ac.uk/8/1/fm_sc_rs_v2.pdf`

Figuur 1.2: Semi-automatische code-generatie

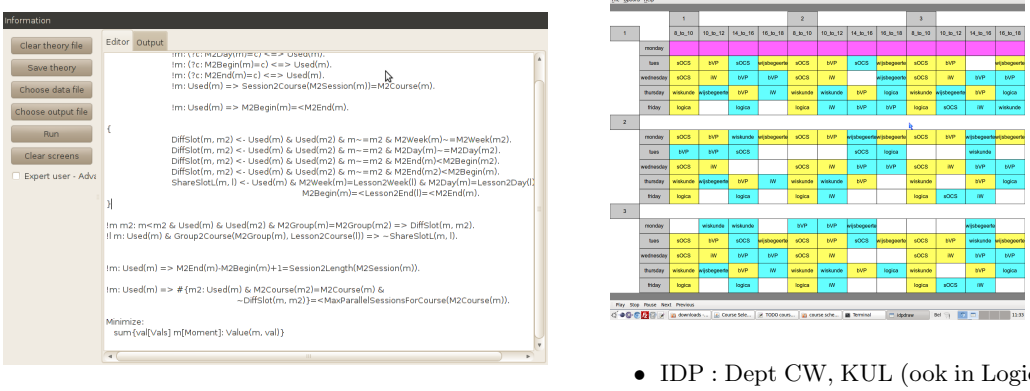
Declaratief programmeren (of problem solving). Een logisch redeneersysteem lost "het probleem" op enkel gebruik makend van de logische specificatie. Programma's schrijven hoeft niet meer. Dergelijke systemen worden ontwikkeld in het gebied van de kunstmatige intelligentie en de computationele logica. Er gebeurt veel onderzoek naar de ontwikkeling van formele talen en solvers ervoor, voor een snel toenemende scala van toepassingen: scheduling, planning, natuurlijke taal verwerking, diagnose, modellering van architectuur, bestuderen van privacy, toegangsrechten, enz. Vele toepassingen zitten nog in de onderzoeksfase. Een logisch kennisbanksysteem dat diverse types van inferentie ondersteunt is het IDP systeem dat aan het departement Computerwetenschappen ontwikkeld wordt in de onderzoeksgroep van Prof. Denecker. Een toepassing van IDP voor het berekenen van uurroosters wordt geïllustreerd in Figuur 1.3. Dit systeem zal gebruikt worden in de oefenzittingen van deze cursus.

Microsoft is een voorbeeld van een bedrijf dat veel onderzoek doet in dit domein. Over declaratief programmeren gaat het in een recent interview van Bill Gates
`http://www.infoworld.com/d/developer-world/gates-talks-declarative-modeling-language-effort-386` (2011)

> *With the [Microsoft] declarative language project, the goal is to make programming declarative rather than procedural. "Most code that's written today is procedural code. And there's been this holy grail of development forever, which is that you shouldn't have to write so much [procedural] code," Gates said. "We're investing very heavily to say that customization of applications, the dream, the quest, we call it, should take a tenth as much code as it takes today."*

Conclusie Logica is van fundamenteel belang voor de Informatica, omdat het de wetenschap is van informatie, de grondstof van informatici, en van het gebruik van informatie om taken op te lossen.

De toepassing van logica en logische systemen is sterk in opmars. In databanksystemen worden logica en bepaalde vormen van redeneren dagdagelijks toegepast in de vorm van querytalen



- Theorie Uurrooster.idp
- IDP berekent (optimale) structuur S die voldoet aan theorie Uurrooster.idp

- IDP : Dept CW, KUL (ook in Logic-Palet)
- Een generator van eindige modellen.
- Taal van IDP: een uitbreiding van predikatenlogica

Figuur 1.3: IDP voor (1) berekenen en (2) visualiseren van uurroosters

zoals SQL. Toch kun je de vraag stellen waarom logica en logische systemen niet meer aandacht krijgen in de informatica en waarom er niet méér toepassingen van logische systemen bestaan. Er zijn daar verschillende redenen voor:

- Ten eerste is het ontwikkelen van domeinonafhankelijke solvers die problemen kunnen oplossen door te redeneren over logische theorieën uiterst complex, en de vooruitgang is traag. Voor veel toepassingen kunnen logische systemen nog niet toegepast worden, eenvoudigweg omdat er niet voldoende efficiënte tools zijn.

- Ten tweede heeft de wetenschappelijke wereld nog geen goed idee van welke soorten informatie relevant zijn in informatica-toepassingen, in welke formele taal deze informatie kan voorgesteld worden, en welke vormen van redeneren nuttig zouden zijn om het probleem op te lossen. Er is veel dat we nog niet begrijpen.

De belangrijkste doelstellingen van deze cursus zijn enerzijds, om te leren hoe gedachten/informatie op een klare en eenduidige manier uit te drukken, door gebruik te maken van logica, en anderzijds om een idee te krijgen van welke vormen van redeneren nuttig zijn in de context van informaticatoepassingen.

1.4 Waarom vertrekken van predicatenlogica?

Er worden veel logica's en formele talen gebruikt in de Informatica, en er worden er steeds nieuwe geïntroduceerd. De taal die wij in deze cursus gebruiken is gebaseerd op de taal die Frege zo'n 130 jaar geleden heeft geïntroduceerd, namelijk de predikatenlogica.

Predikatenlogica, in de vorm zoals in deze cursus voorgesteld wordt, wordt niet vaak gebruikt in softwaresystemen. Bv. in databanksystemen gebruikt men meestal de taal SQL, niet predicatenlogica. In feite worden in de meeste toepassingsdomeinen van logica telkens nieuwe logica's geïntroduceerd, die er meestal heel anders *uitzien* dan predicatenlogica maar die bij nader toezien er toch uit werden afgeleid en dus toch voortbouwen op de basisconcepten van predicatenlogica.

De reden is dat predicatenlogica bestaat uit een minimale set van elementaire taalconstructies die absoluut noodzakelijk zijn om informatie in te beschrijven. Deze taalconstructies, hun formele notatie en hun betekenis zijn gegeven in de volgende lijst:

$\ldots \wedge \ldots$	... en ...
$\ldots \vee \ldots$	... of ... (of allebei)
$\neg \ldots$	het is niet waar dat ...
$\ldots \Rightarrow \ldots$	als ... dan ...
$\ldots \Leftrightarrow \ldots$	... als en slechts als ...
$\forall \ldots\ldots$	voor alle ... geldt dat ...
$\exists \ldots\ldots$	er bestaat ... zodat ...
$\ldots = \ldots$	... is gelijk aan ...

Deze symbolen is men later ook vaak gaan gebruiken in wiskundige teksten, als afkortingen voor de uitdrukkingen rechts in de tabel. Dit zijn het soort connectoren en kwantoren die noodzakelijk zijn om complexe gedachten/uitspraken uit te drukken zoals Leibniz bedacht had. Vandaar dat ze op één of ander manier in vele logica's en formele talen voorkomen, alhoewel vaak onder andere syntactische vorm. Die variatie in notaties of in *syntactische suiker* zoals men ook zegt, maakt het soms moeilijk om van het bos de bomen te zien. We vermijden dit in deze cursus door ons te houden aan de syntax van predicatenlogica.

1.5 Notaties en basisbegrippen

We gebruiken de volgende afkortingen :

- Geg. Gegeven
- T.B. Te Bewijzen
- Bew. Bewijs
- Q.E.D. Quod Erat Demonstrandum : einde bewijs
- V.T.B. Voldoende Te Bewijzen.
- asa als en slechts als
- t.t.z. 't is te zeggen
- m.a.w. met andere woorden
- i.p.v. in plaats van

We gebruiken ook een aantal griekse letters:

- α alpha
- β beta
- θ theta
- Σ Sigma

1.6 Verzamelingen en relaties

We gebruiken een aantal basisconcepten uit verzamelingentheorie.

Logica is de taal der wiskundige relaties en functies: logica laat toe eigenschappen van relaties en functies te beschrijven. Om te begrijpen waar logica om draait, is het nodig te begrijpen wat relaties en functies zijn. Deze sectie introduceert enkele elementaire begrippen, waarvan we eigenlijk veronderstellen dat elke beginnende student informatica deze reeds kent en geoefend heeft. Indien je dit niet 100% snapt, is het belangrijk dit zelf aan te pakken.

Meer informatie kan gevonden worden in elk basisboek wiskunde of op het internet:
`http://en.wikipedia.org/wiki/Set_(mathematics)`
`http://en.wikipedia.org/wiki/Finitary_relation`
`http://en.wikipedia.org/wiki/Theory_of_relations`
`http://en.wikipedia.org/wiki/Binary_relation`
`http://en.wikipedia.org/wiki/Cartesian_product`
`http://en.wikipedia.org/wiki/Function_(mathematics)`

Definitie 1.6.1. Een *verzameling* is een ongeordend geheel van elementen. Twee verzamelingen zijn gelijk asa ze dezelfde elementen hebben.

Verzamelingen zijn potentieel oneindig.

Voorbeelden van verzamelingen zijn:

- $\emptyset$, de lege verzameling. Ook genoteerd als $\{\}$
- $\{0\}$, een singleton (bevat 1 element)
- $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$: de verzamelingen van natuurlijke, gehele, rationale, reële en complexe getallen.
- De verzameling van alle mensen op deze wereld
- $\{jan, piet, joris, corneel\}$
- $\{\bigstar, \clubsuit, \blacksquare, \heartsuit, \triangle\}$
- De verzameling van alle veeltermen
- $\{x \mid x = 2n + 1, n \in \mathbb{N}\}$ is bekend als de verzameling oneven getallen

We noteren dat a een element van S is als $a \in S$.

A is een deelverzameling van B als alle elementen van A elementen van B zijn. Notatie $A \subseteq B$. Dit sluit niet uit dat $A = B$. $A \subsetneq B$ betekent dat A een strikte subset is van B en dus verschillend.

Zij A, B verzamelingen.

- De unie $A \cup B$ is de verzameling van alle elementen die behoren tot A of tot B.
- De doorsnede $A \cap B$ is de verzameling van alle elementen die behoren tot A en tot B.

- Het verschil $A \setminus B$ is de verzameling van alle elementen die behoren tot A en niet tot B.
- De kardinaliteit $|A|$ van een verzameling A is het aantal elementen. Dit kan oneindig zijn.
- De delenverzameling van een verzameling A is de verzameling $\{B \mid B \subseteq A\}$. Notatie 2^A.

Definitie 1.6.2. Een *tupel* of een *n-tal* of een *koppel* is een eindige rij $(d_1, \ldots, d_n)$ van objecten, waarbij dubbele objecten mogen voorkomen in de lijst, en de volgorde van de lijst belang heeft.

Voorbeelden van tupels zijn:

- $(0, 0)$
- $(\bigstar, \clubsuit, \blacksquare, \blacksquare)$
- elk punt uit het cartesisch vlak kan voorgesteld worden door een tupel van twee getallen
- () is het lege tupel

Definitie 1.6.3. Gegeven een rij $(S_1, \ldots, S_n)$ van verzamelingen (waarin dezelfde verzameling meerdere keer mag voorkomen), dan is het *cartesisch product* van deze rij de verzameling van alle mogelijke koppels $(d_1, \ldots, d_n)$ waarbij $d_i \in S_i$. We noteren deze verzameling als $S_1 \times \ldots \times S_n$. Meer formeel: $S_1 \times \ldots \times S_n = \{(d_1, \ldots, d_n) \mid d_1 \in S_1, \ldots, d_n \in S_n\}$. Een cartesisch product over n verzamelingen is *n-voudig* of is *n-dimensioneel*. Het k-voudige cartesisch product van een reeks gelijke verzamelingen S korten we af als S^k.

Voorbeelden:

- Het cartesisch vlak wordt uitgedrukt als $\mathbb{R} \times \mathbb{R}$, ofwel $\mathbb{R}^2$
- Als $S_1 = \{\clubsuit, \heartsuit\}$, en $S_2 = \{0, 1\}$, dan is $S_1 \times S_2 = \{(\clubsuit, 1), (\clubsuit, 0), (\heartsuit, 1), (\heartsuit, 0)\}$.
- Als A de verzameling is van alle letters in het romeins alfabet, dan is A^k de verzameling van alle mogelijke k-tupels van letters. Dit komt overeen met strings van lengte k.
- Het 0-voudige cartesisch product is het singleton $\{()\}$ (de verzameling met als enig element het lege tupel).

Definitie 1.6.4. Een *relatie* R is een deelverzameling van een cartesisch product C. We zeggen dat R *over* C gaat of dat C het domein van R is. R heeft *ariteit* n indien C n-voudig is. Een relatie met ariteit 1 noemen we een *unaire* relatie, een relatie met ariteit 2 *binair*, en een relatie met ariteit 3 *ternair*.

Noot: we laten toe om bij het neerschrijven van unaire tupels de haakjes weg te laten. $\{(3), (4), (5)\}$ en $\{3, 4, 5\}$ stellen dezelfde verzameling voor.

Voorbeelden:

- De relatie "is-deler-van" is een binaire relatie bestaande uit koppels (n, m) van gehele getallen zodat n een delers is van m. Het is een deelverzameling van $\mathbb{Z}^2$, en bevat o.a. de tupels $(1, 1), (2, 8), (-13, 13)$, maar niet de tupels $(0, 1), (3, 8), (2345, 13), (-3, 4)$
- De binaire relatie $\subseteq$ ("is-deelverzameling-van")

- De binaire relatie $<$ ("is-kleiner-dan")
- De set tupels $(x, y, \sqrt{x+y})$ met $x, y \in \mathbb{R}$ vormen een ternaire relatie over $\mathbb{R} \times \mathbb{R} \times \mathbb{C}$
- De verzameling van personen die in België wonen is een unaire relatie
- De lege verzameling $\{\}$ is een relatie over elk willekeurig cartesisch product

Oefening 1.6.1. *1. Stel $S = \{0, 1, 2\}$. Geef de grootst mogelijke ternaire relatie R over S^3 zodanig dat het derde element uit de tupels in R het verschil is van de eerste twee elementen. Met andere woorden, R bevat enkel tupels van de vorm $(x, y, x - y)$.*

2. Geef een plausibel cartesisch product waarover de binaire relatie $<$ ("is-kleiner-dan") gaat.

3. Als S een verzameling is met n elementen, hoeveel verschillende relaties bestaan er dan over S^k?

Definitie 1.6.5. Gegeven een cartesisch product $S_1 \times \ldots \times S_n$ en een verzameling S, dan is een *functie* F met *signatuur* $S_1 \times \ldots \times S_n \to S$ een relatie over $S_1 \times \ldots \times S_n \times S$ zodat voor elk tupel $(s_1, \ldots, s_n) \in S_1 \times \ldots \times S_n$ **exact één** $s \in S$ bestaat waarvoor $(s_1, \ldots, s_n, s) \in F$. We noemen zo'n s het *beeld* van $(s_1, \ldots, s_n)$, en schrijven $s = F(s_1, \ldots, s_n)$. We noemen $S_1 \times \ldots \times S_n$ het *domein* van F, en S het *co-domein* of *bereik* van F.

Een partiële functie is een functie behalve dat het beeld van sommige elementen uit het domein niet gedefinieerd is.

We gebruiken de volgende notatie om een functie te noteren. Bv.

$$F : \mathbb{N} \to \mathbb{N} : n \to n^2$$

is de kwadraatfunctie in de natuurlijke getallen, en is de verzameling

$$\{(n, n^2) \mid n \in \mathbb{N}\}$$

Voorbeelden:

- de delingfunctie / heeft signatuur $\mathbb{R} \times \mathbb{R}_0 \to \mathbb{R}$ en bevat onder andere de tupels $(6, 3, 2)$ en $(0, \sqrt{2}, 0)$, maar niet de tupels $(\pi, 2, \pi)$ en $(0, 0, 0)$.
- de wortelfunctie $\sqrt{}: \mathbb{R} \to \mathbb{C}$
- de functie $vader: P \to P$ met P de verzameling van alle personen, en $x = vader(y)$ indien $x, y \in P$ en x is vader van y.
- Het r-nummersysteem is een functie van de verzameling studenten van de KU Leuven naar de verzameling van natuurlijke getallen voorafgegaan door een r.

Merk op dat bovenstaande definitie stelt dat elke functie een relatie is, en dat een functie $F: S_1 \times \ldots \times S_n \to S$ een relatie met ariteit $n + 1$ is. We spreken echter af om bij het bepalen van de ariteit van een functie het co-domein te negeren:

Definitie 1.6.6. Een functie $F: S_1 \times \ldots \times S_n \to S$ heeft ariteit n.

$F: S_1 \times \ldots \times S_n \to S$ is dus zowel een **functie** met ariteit $\mathbf{n}$ als een **relatie** met ariteit $\mathbf{n + 1}$.

Oefening 1.6.2. *1. Neem $S = \{0, 1\}$, schrijf de functie $S \times S \to S : (x, y) \mapsto (x * y)$ uit.*

2. *Neem $S = \{0, 1, 2\}$. Stel dat R een relatie is over S^3, en $R = \{(x, y, z) \mid z = x - y\}$. Met andere woorden, R is de verzameling van alle tupels (x, y, z) waarbij $z = x - y$ en $x, y, z \in S$. Stelt R dan een binaire functie $S \times S \to S$ voor? Hint: schrijf R volledig uit, en check Definitie 1.6.5.*

3. *Als we de functie F beschouwen die uit tupels (x, y, z) bestaat zodat $z = x/y$, wat is dan een geschikt domein en co-domein voor F zodanig dat F een functie is volgens Definitie 1.6.5?*

4. *Als S een verzameling is die n elementen bevat, hoeveel verschillende functies $F : S^k \to S$ bestaan er?*

De verzameling van alle functies van D naar B wordt genoteerd als B^D.

Een functie f van verzameling D_1 naar verzameling D_2 is een *bijectie* als voor elk object $b \in D_2$ een uniek object $d \in D_1$ bestaat zodat $f(d) = b$.

Een functie f van verzameling D_1 naar verzameling D_2 is een *injectie* als voor elk paar $d_1, d_2 \in D_1$, $f(d_1)$ verschillend is van $f(d_2)$.

Een functie f van verzameling D_1 naar verzameling D_2 is een *surjectie* als voor elke $b \in D_2$ een $d \in D_1$ bestaat zodat $b = f(d)$.

Een bijectie is dus een functie die zowel injectief als surjectief is.

Een deelverzameling B van A komt overeen met een functie f_B van A naar $\{0, 1\}$, namelijk $f_B(x) = 1$ asa $x \in B$. Deze functie wordt de karakteristieke functie van B genoemd. Er is een bijectie van de delenverzameling 2^A van A naar de verzameling $\{0, 1\}^A$ van functies van A naar $\{0, 1\}$.

Vrijwel alle soorten van wiskundige objecten komen overeen met of kunnen geherformuleerd worden als een verzameling, of een verzameling van verzamelingen, of nog dieper.

- Bv. een n-tal $(d_1, \ldots, d_n)$ in domein D komt overeen met een functie van $[1, n] \to D$, wat dus overeenkomt met een verzameling koppels $\{(1, d_1), \ldots, (n, d_n)\}$.
- Bv. een $n \times m$-matrix komt overeen met een functie $[1, n] \times [1, m] \to D$.
- Een tabel met n-kolommen komt overeen met een verzameling van n-tallen en dus met een n-voudige relatie.
- Arrays uit programmeertalen

Dit is belangrijk in logica, want alle objecten worden voorgesteld als primitieve objecten, functies of relaties.

1.7 Inductieve definities, inductief bewijs

De meeste definities van concepten definiëren een concept in termen van andere reeds gedefinieerd concepten. Bv.

Een getal $n \in \mathbb{N}$ is *deelbaar door* $m \in \mathbb{N}$ als er een getal $p \in \mathbb{N}$ bestaat zodat $m \times p = n$.

Een natuurlijk getal n is *priem* als n enkel deelbaar is door 1 en door n.

Inductieve definities vormen hier een uitzondering op. Zij definiëren een concept in termen van zichzelf.

Een bekend voorbeeld is het concept van fibonacci-rij, hier gedefinieerd als een functie op $\mathbb{N}$:

$$fib(0) := 0;$$
$$fib(1) := 1;$$
$$fib(n+2) := fib(n) + fib(n+1)$$

Hieronder worden inductieve definities van verzamelingen en recursive definities van functies geïntroduceerd per voorbeeld. Nadien gaat het over inductieve bewijzen, een techniek om eigenschappen van inductief of recursief gedefinieerde concepten te bewijzen.

Definitie 1.7.1. De verzameling $\mathbb{N}$ van natuurlijke getallen wordt inductief gedefinieerd door de volgende regels:

- $0 \in \mathbb{N}$ (b)
- als $n \in \mathbb{N}$ dan $1 + n \in \mathbb{N}$. (i)

(b) en (i) zijn namen die we geven aan beide regels: (b) voor basisgeval, (i) voor inductief geval.

"Inductief gedefinieerd": wat wordt hiermee bedoeld? Er zijn twee manieren om uit te leggen wat dit betekent.

De eerste methode is de constructieve uitleg. Hierbij wordt $\mathbb{N}$ bekomen door geïtereerde uitvoering van de regels, vertrekkende van de lege verzameling:

- $\mathbb{N}^0 = \emptyset$.
  In deze verzameling zijn alle instantiaties van de inductieve regel (i) voldaan aangezien voor elk object n geldt dat $n \notin \mathbb{N}^0$. Maar de basisregel (b) is niet voldaan: $0 \notin \mathbb{N}^0$. Dus die regel kunnen we toepassen om 0 af te leiden:
- $\mathbb{N}^1 = \mathbb{N}^0 \cup \{0\} = \{0\}$.
  Nu is (b) en alle instantiaties van (i) voldaan behalve voor $n = 0$: immers $0 \in \mathbb{N}^1$ maar $1 = 1 + 0 \notin \mathbb{N}^1$. Dus we kunnen (i) toepassen om het element 1 toe te voegen.
- $\mathbb{N}^2 = \mathbb{N}^1 \cup \{1\} = \{0, 1\}$.
  Nu is opnieuw 1 instantiatie van (i) niet voldaan, namelijk voor $n = 1$.
- $\mathbb{N}^2 = \{0, 1, 2\}$.
- ...

We bekomen een oneindige groeiende rij van verzamelingen $\mathbb{N}^0, \mathbb{N}^1, \mathbb{N}^2, \ldots, \mathbb{N}^i, \mathbb{N}^{i+1}, \ldots$. De **limiet** van deze rij is de unie ervan: $\mathbb{N}$. Hiermee eindigt de constructie, want in $\mathbb{N}$ zijn alle regels voldaan: $\mathbb{N}$ bevat 0, en voor elke element n bevat $\mathbb{N}$ ook $n+1$. Dit proces om de inductief gedefinieerde verzameling te construeren door geïtereerd toepassen van regels noemt men het *inductie-proces*.

De tweede methode om deze inductieve definitie te interpreteren is door te stellen dat de gedefinieerde verzameling $\mathbb{N}$ de *kleinste* verzameling van objecten is die voldoet aan de twee regels. Anders gezegd, er zijn veel verzamelingen S die 0 bevatten en die als ze n bevatten ook $n+1$ bevatten, bv. $\mathbb{Z}$, de verzameling van de gehele getallen, voldoet ook aan (b) en (i). Maar, de gedefinieerde verzameling $\mathbb{N}$ is de kleinste verzameling die aan de regels voldoet. We kunnen ook stellen dat $\mathbb{N}$ de doorsnede is van alle verzamelingen die aan de regels voldoen.

Samengevat: een inductieve definitie van een verzameling bestaat uit een aantal regels (basisregels en inductieve regels). Er zijn twee manieren om uit te leggen wat de gedefinieerde verzameling is: (1) de gedefinieerde verzameling is degene die bekomen wordt als limiet van het inductie-proces; en (2) de gedefinieerde verzameling is de kleinste verzameling van alle verzamelingen die aan de regels van de definitie voldoen, en bijna equivalent, (3) de gedefinieerde verzameling is de doorsnede van alle verzamelingen die aan de regels voldoen.

Het maakt niet uit of we een inductive definitie interpreteren op manier (1), (2) of (3), aangezien men kan bewijzen dat voor het normale type van inductieve definitie deze verschillende zienswijzes equivalent zijn (zie Propositie 1.7.1). [2]

Oefening 1.7.1. *Geef andere voorbeelden van verzamelingen die aan de regels van de inductieve definitie voldoen en verifieer dat $\mathbb{N}$ een deelverzameling is.*

We zullen nu bewijzen voor de definitie van natuurlijke getallen dat die verschillende zienswijzen inderdaad overeenkomen.

Propositie 1.7.1. *$\mathbb{N}$ is een deelverzameling van elke verzameling S die aan beide regels (b) en (i) van Definitie 1.7.1 voldoet.*

Bewijs. Uit het ongerijmde. Veronderstel dat S voldoet aan (b) en (i) zodat $0 \in S$ en als $m \in S$ dan ook $1 + m \in S$. Veronderstel naar contradictie toe dat $\mathbb{N} \not\subseteq S$.

Neem de verzameling $\mathbb{N} \setminus S$. Deze is niet leeg aangezien $\mathbb{N} \not\subseteq S$. Elke niet-lege deelverzameling van $\mathbb{N}$ bevat een kleinste element. Zij m het kleinste getal van $\mathbb{N} \setminus S$. m is niet 0 want wegens (b) geldt $0 \in S$. Dus bestaat een getal $m' \in \mathbb{N}$ zodat $m = 1 + m'$ en $m' \in S$. Maar wegens (i) behoort dan ook $1 + m' = m$ tot S. Contradictie. ■

Oefening 1.7.2. *Een grafe is een paar (V, E) met V een verzameling van nodes, en $E \subseteq V \times V$ een verzameling van paren – gerichte bogen in een grafe. Een pad in een grafe is een sequentie $v_0, v_1, \ldots, v_n$ van nodes zodat $(v_i, v_{i+1}) \in E$. Dit is een pad van v_0 naar v_n. Paden kunnen ook oneindig zijn zoals $v_0, v_1, \ldots, v_i, v_{i+1}, \ldots$.*

Geef een inductieve definitie van de volgende concepten:

[2] De waarheid gebiedt mij echter te waarschuwen dat er ook inductieve definities zijn waar (1), (2) en (3) niet equivalent zijn. We zien later een voorbeeld hiervan: de definitie van waarheid in een structuur.

- *De bereikbaarheidsrelatie B van (V, E) is de verzameling van alle paren van nodes (v, w) zodat er een pad van v naar w bestaat in E.*
- *De terminale relatie T van (V, E) bestaat uit alle nodes v van V die geen oneindig pad $v, v_1, \ldots, v_i, v_{i+1}, \ldots$ hebben in E.*

Oefening 1.7.3. *Neem de volgende grafe (V, E):*

- $V = \{a, b, c\}$
- $E = \{(a, a), (b, c)\}$

Toon het inductieproces van de bereikbaarheidsrelatie B en van de terminal relatie T.

Een ander type van definitie is de recursieve definitie van een functie.

Definitie 1.7.2. We definiëren de som $m+n$ van twee natuurlijk getallen m, n op een recursieve manier:

- $\forall n \in \mathbb{N} : 0 + n = n$ (b)
- $\forall n, m \in \mathbb{N} : (1 + m) + n = 1 + (m + n)$. (i)

De (b) staat opnieuw voor "basisgeval", en (i) staat voor "inductie-geval".

Hoe deze definitie te interpreteren. Dit is gelijkaardig aan het inductieproces voor verzamelingen. We beginnen met een volledig ongedefinieerde functie. We bouwen een rij van functies $+^0, +^1, \ldots, +^i, +^{i+1}, \ldots$.

- $+^0$ is gedefinieerd voor geen enkel paar (n, m).
  We kunnen de basisregel (b) toepassen om voor een willekeurig getal n te definiëren dat $0 + n = n$.
- $+^1 = \{(0, n, n) \mid n \in \mathbb{N}\}$.
  In het rechterlid van (i) is $m' + n$ gekend indien $m' = 0$. Dus kunnen voor elke n afleiden wat $1 + n$ is.
- $+^2 = +^1 \cup \{(1, n, 1 + n) | n \in \mathbb{N}\}$
- $+^3 = +^2 \cup \{(2, n, 2 + n) | n \in \mathbb{N}\}$
- $\ldots$

De limiet van deze constructie is de functie + waarmee we vertrouwd zijn.

Wat is het verschil tussen een inductieve definitie en een recursieve definitie? Bij het eerste definieert men een **verzameling**; men begint met de lege verzameling, en voegt elementen toe totdat alle regels voldaan zijn. Bij het tweede definieert men een **functie**, meestal over een inductief gedefinieerd domein (bv. de natuurlijke getallen); men begint met een volledig

ongedefinieerde functie (gedefinieerd op geen enkel element van het domein) en breidt uit door in steeds meer elementen van het domein de functiewaarde te definiëren. Dat is het verschil.

De concepten van inductieve en recursieve definitie zijn gerelateerd tot het principe van *bewijs door inductie*. Bewijs door inductie is een methode om eigenschappen van de vorm $\forall x \in S : \Phi[x]$ te bewijzen voor verzamelingen S die inductief gedefinieerd werden en $\Phi[x]$ een eigenschap van x.

De verzameling $\mathbb{N}$ is inductief gedefinieerd. Om eigenschappen te bewijzen van de vorm $\forall n \in \mathbb{N} : \Phi[n]$ is het probleem dat er oneindig veel natuurlijke getallen zijn en we kunnen onmogelijk $\Phi[n]$ voor elk apart kunnen bewijzen. Dat wordt verholpen door een inductiebewijs. We weten dat $\mathbb{N}$ de kleinste verzameling is die voldoet aan (b) en (i). Men bewijst dus gewoon that $\{n \mid \Phi[n]\}$ voldoet aan de twee regels (b) en (i). Dan volgt automatisch dat $\mathbb{N} \subseteq \{n \mid \Phi[n]\}$ of, equivalent hiermee, dat voor alle $n \in \mathbb{N}, \Phi[n]$ geldt. Concreet, om $\forall n \in \mathbb{N} : \Phi[n]$ te bewijzen:

- **(basisstap)** Bewijs $\Phi[0]$.
- **(inductiestap)** Bewijs dat als $\Phi[n]$ dan $\Phi[n+1]$. In dit geval noemt men $\Phi[n]$ de inductiehypothese.

Dit is het bewijs per inductie (voor natuurlijke getallen). Verderop volgen voorbeelden.

De operatie + heeft een aantal nuttige, natuurlijke eigenschappen. Vele ervan kunnen enkel bewezen worden door middel van een inductiebewijs.

Propositie 1.7.2. *+ is associatief in* $\mathbb{N}$*:*

$$\forall x, y, z \in \mathbb{N} : (x + y) + z = x + (y + z)$$

Bewijs. Neem y, z willekeurige getallen. Het bewijs is per inductie op x; $\Phi[x]$ is $(x + y) + z = x + (y + z)$.

[Basisgeval: neem aan dat $x = 0$ (Ab)]. ((Ab) staat voor "Assumptie, basisgeval".)

$$\begin{aligned}(x + y) + z &\overset{(Ab)}{=} (0 + y) + z \\ &\overset{(b)}{=} y + z \\ &\overset{(b)}{=} 0 + (y + z) \\ &\overset{(Ab)}{=} x + (y + z)\end{aligned}$$

[Inductie-geval: neem aan dat $x = 1 + x'$ (Ai)]. (Ai) staat voor "Assumptie, inductiegeval". De inductiehypothese $H[x']$ is $(x' + y) + z = x' + (y + z)$.

$$\begin{aligned}(x + y) + z &\overset{(Ai)}{=} ((1 + x') + y) + z \\ &\overset{(i)}{=} (1 + (x' + y)) + z \\ &\overset{(i)}{=} 1 + ((x' + y) + z) \\ &\overset{(H[x'])}{=} 1 + (x' + (y + z)) \\ &\overset{(i)}{=} (1 + x') + (y + z) \\ &\overset{(Ai)}{=} x + (y + z)\end{aligned}$$

Nu kunnen we besluiten dat $\forall x \in \mathbb{N} : (x + y) + z = x + (y + z)$. Dit geldt voor willekeurige $y, z \in \mathbb{N}$. Bijgevolg geldt dat $\forall y, z : \forall x \in \mathbb{N} : (x+y)+z = x+(y+z)$. De volgorde van kwantoren speelt hier geen rol, zodat even goed volgt dat $\forall x, y, z \in \mathbb{N} : (x + y) + z = x + (y + z)$. ■

Woordje uitleg: in het bewijs herschrijven we de term $(x+y)+z$ door deeltermen te herschrijven met wetten ((b) of (i)) waarvan we de geldigheid kennen uit de definitie van $\mathbb{N}$, of door assumptie (Ai of Ab) of door de inductie-hypothese ($H[x']$). Verifieer dat elke overgang correct is en gebruik maakt van de aangegeven wet die staat boven het gelijkheidsteken.

Oefening 1.7.4. *Ook commutativiteit van + is een belangrijke eigenschap. Bewijs dat + commutatief is in $\mathbb{N}$:*

$$\forall n, m \in \mathbb{N} : n + m = m + n$$

Het is mogelijk dat je merkt dat je er niet raakt zonder eerst hulpeigenschappen te bewijzen. Hint: bewijs eventueel een paar hulpeigenschappen:

- **Eigenschap 1:** $\forall n \in \mathbb{N} : n + 0 = n$
- **Eigenschap 2:** $\forall n \in \mathbb{N} : 1 + n = n + 1$

In mijn bewijs van de commutativiteit had ik volgende afhankelijkheden van eigenschappen:

$$\textit{Definitie (b)+(i)} \rightarrow \textit{Associativiteit} \rightarrow \textit{Eig1} \rightarrow \textit{Eig2} \rightarrow \textit{commutativiteit}$$

Mocht 1 van deze eigenschappen niet gelden, dan stort het hele stelsel ineen.

Oefening 1.7.5. *Een ander type van inductiebewijs voor een eigenschap $\Phi[n]$ over natuurlijke getallen, is om te bewijzen dat dat voor elk natuurlijk getal n het volgende geldt:*

$$\forall m < n : \Phi[m]) \Rightarrow \Phi[n]$$

Als je dit kunt bewijzen voor elke n, dan volgt hieruit dat $\Phi[n]$ geldt voor elk natuurlijk getal, of dus dat $\forall n \in \mathbb{N} : \Phi[n]$. Argumenteer waarom dit zo is, en bewijs het.

Deze cursus bevat veel inductie-bewijzen. Zorg ervoor dat je dit principe verstaat en kan toepassen. Bewijzen per inductie zijn niet alleen fundamenteel in logica maar ook in computerwetenschappen. Bijna alle bewijzen van correctheid van programma's zijn inductieve bewijzen.

Oefening 1.7.6. *De volgende stelling bevat een verrassend resultaat, dat is wel het minste wat men kan zeggen.*

Stelling 1.7.1. *Als een wei minstens 1 wit paard bevat, dan zijn alle paarden in de wei wit.*

Bewijs. Per inductie op het aantal paarden in de wei.

Stel dat wei leeg is en dus 0 paarden bevat. Dan is de eigenchap triviaal voldaan.

Stel dat de wei 1 paard bevat. Als het minstens 1 wit paard bevat, dan zijn natuurlijk alle paarden wit.

Stel dat de wei $n > 1$ paarden bevat. Inductiehypothese: veronderstel dat de stelling voldaan is voor alle $m < n$.

Als de wei geen witte paarden bevat is er niets te bewijzen. Stel dat de wei een wit paard bevat. Kies een ander paard (p) waarvan nog niet bepaald is of het wit is. Kies een deelwei die het witte paard bevat en alle paarden behalve dat ene paard. Deze deelwei bevat $n - 1$ paarden waaronder 1 wit. Omwille van de inductiehypothese volgt dat alle paarden in deze deelwei wit zijn.

Kies nu een tweede deelwei bestaande uit n-1 paarden die het witte paard en het resterende paard bevat. Dan volgt opnieuw uit de inductiehypothese dat alle paarden in deze deelwei wit zijn.

Dus zijn alle paarden wit. ■

Waar zit de fout?

De stelling in dit voorbeeld is volkomen nonsens. Toch is het bewijs "bijna" correct. Het is correct voor alle waarden van n behalve voor één waarde. Welke? In een inductiebewijs is het belangrijk dat *elke* stap correct is.

Definities, stellingen en bewijzen in deze cursus Definities, stellingen en bewijzen worden nauwelijks gestudeerd in het secondair onderwijs. Sommige wiskunde-didactici beschouwen dit als "papegaaiewerk": het is enkel kennis, geen vaardigheid.

Daarentegen, Michael Sipser, een toonaangevend theoretisch computerwetenschapper, schrijft in zijn boek "Introduction to the Theory of Computation":

> Finally, theory is good for you because studying it expands your mind. Computer technology changes quickly. Specific technical knowledge, though useful today, becomes outdated in just a few years. Consider instead the abilities to think, to express yourself clearly and precisely, to solve problems, and to know when you haven't solved a problem. These abilities have lasting value. Studying theory trains you in these capabilities.

Wie heeft gelijk?

Sipser natuurlijk. De studie van definities+stellingen+bewijzen is misschien wel de belangrijkste oefening om te leren hoe op een precieze manier een concept te definiëren, hoe eigenschappen te formuleren, hoe argumenten op een precieze en correcte manier op te bouwen, en deze goed en duidelijk te verwoorden. Dit zijn vaardigheden van het grootste belang.

Bewijzen zijn bovendien een bron van inzicht in waarom eigenschappen gelden. Velen, hopelijk jullie ook, hebben ervaren dat een bewijs van een bepaalde stelling diep inzicht kan geven in waarom de stelling voldaan is. Dus is het trainen in bewijzen nuttig om ons inzicht te geven en ons analytisch vermogen te ontwikkelen.

Leer geen definities, eigenschappen, bewijzen van buiten als je de gedachte erachter niet begrijpt. Dat is tijdverlies; dan is het verloren moeite. Maar als je definities, eigenschappen en bewijzen leert, let op hoe zorgvuldig ze opgesteld zijn opdat ze helder, precies en correct de onderliggende gedachte zouden weergeven. Probeer ze te herformuleren maar wees kritisch voor jezelf: kijk na of je geen categorie-clashes maakt, vage taal gebruikt, ambiguïteiten veroorzaakt. Dan zul

je vaak merken dat het niet altijd gemakkelijk is om een definitie of een eigenschap in eigen woorden te herformuleren.

Het nut van het bestuderen van definities, stellingen en bewijzen, is niet alleen om kennis te verwerven over de gedefinieerde concepten, stellingen en bewijzen, wat overigens wel belangrijk is. Maar van universeler belang, het is de training die nodig is om te leren de gedachte achter tekst te onderscheiden, en omgekeerd, te leren hoe een gedachte, of een gedachtegang, of een redenering in duidelijke tekst te gieten. Dat is een uiterst nuttige vaardigheid. Om die vaardigheid te leren moet men beginnen met het bestuderen van goede voorbeelden: theorie dus. Ik ken geen andere manier.

Hoofdstuk 2

De predikatenlogica

Zoals gezien heeft de term "Logica" verschillende betekenissen: (1) de wetenschappelijke studie van het correct redeneren, (2) een formele taal om informatie uit te drukken (3) de wetenschappelijke studie van de informatie. Dat maakt logica tot een soort meta-discipline van de wiskunde en wetenschappen. Logica heeft een rijke geschiedenis met grote namen als Aristoteles, Leibniz, Boole, Frege, Gödel, Turing. Logica heeft een historische rol gespeeld in de grondslagen van de wiskunde en (analytische) filosofie. Logica stond aan de basis van computerwetenschappen, met boolse algebra voor computer hardware. Zoals we in deze cursus zullen ontdekken speelt logica de rol van een exacte taal voor formele specificaties van software en als taal om kennis in uit te drukken in declaratieve toepassingen: bv. in databanken, constraint programming, kennisbanken, ...

In dit hoofdstuk wordt de Predikatenlogica gedefinieerd.

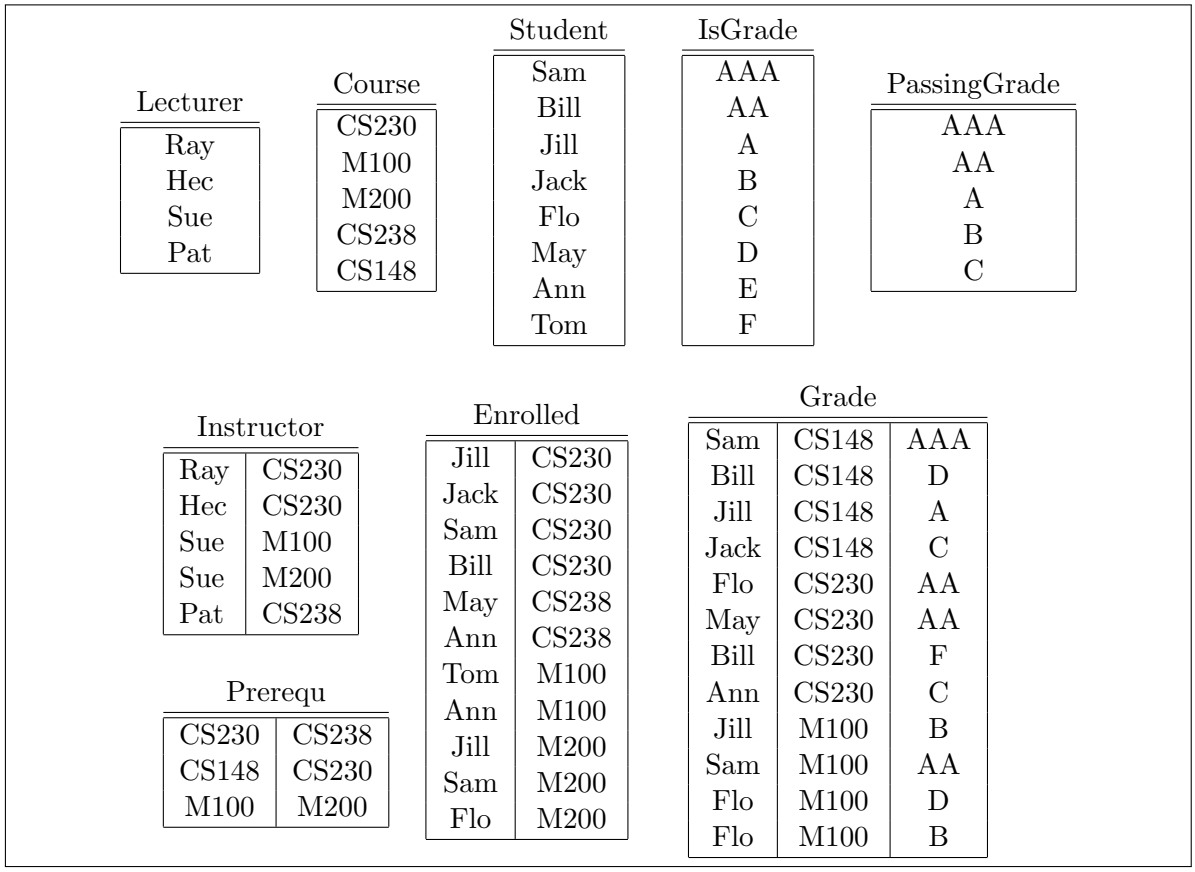
2.1 Inleiding tot de predikatenlogica

De logische symbolen van predicatenlogica zijn bekend omdat ze ook als afkortingen gebruikt worden in wiskundige teksten:

$\ldots \wedge \ldots$	... en ...
$\ldots \vee \ldots$	... of ... (of allebei)
$\neg \ldots$	het is niet waar dat ...
$\ldots \Rightarrow \ldots$	als ... dan ...
$\ldots \Leftrightarrow \ldots$	... als en slechts als ...
$\forall \ldots\ldots$	voor alle ... geldt dat ...
$\exists \ldots\ldots$	er bestaat ... zodat ...
$\ldots = \ldots$	... is gelijk aan ...

We introduceren de manier waarop die symbolen gebruikt worden in de predikatenlogica in de context van een informatietoepassing, namelijk een databank (of ook gegevensbank). Meer specifiek, de studentendatabank in Figuur 2.1 over vakken, graden, docenten en studenten aan een fictieve universiteit.

Deze databank bestaat uit een aantal tabellen. Deze tabellen beschouwen we als n-voudige relaties. Rijen in een tabel komen overeen met n-voudige tupels in de relatie. Bv., in wiskundige



Lecturer

Ray
Hec
Sue
Pat

Course

CS230
M100
M200
CS238
CS148

Student

Sam
Bill
Jill
Jack
Flo
May
Ann
Tom

IsGrade

AAA
AA
A
B
C
D
E
F

PassingGrade

AAA
AA
A
B
C

Instructor

Ray	CS230
Hec	CS230
Sue	M100
Sue	M200
Pat	CS238

Prerequ

CS230	CS238
CS148	CS230
M100	M200

Enrolled

Jill	CS230
Jack	CS230
Sam	CS230
Bill	CS230
May	CS238
Ann	CS238
Tom	M100
Ann	M100
Jill	M200
Sam	M200
Flo	M200

Grade

Sam	CS148	AAA
Bill	CS148	D
Jill	CS148	A
Jack	CS148	C
Flo	CS230	AA
May	CS230	AA
Bill	CS230	F
Ann	CS230	C
Jill	M100	B
Sam	M100	AA
Flo	M100	D
Flo	M100	B

Figuur 2.1: De studentendatabank

notatie uitgedrukt komt de tabel van *Prerequ* overeen met de verzameling

$$\{(CS230, CS238), (CS148, CS230), (M100, M200)\}.$$

Een databank bevat twee soorten symbolen: (1) tabelnamen komen overeen met relatienamen, zoals *Lecturer* en *Grade*; deze noemen we later predicaatsymbolen; (2) objectsymbolen zoals *Ray* en *AAA*.

De databank stelt een stand van zaken voor aan een universiteit. Om die stand van zaken te kennen moeten we de betekenis van de symbolen kennen. De informele betekenis van de tabelnamen is als volgt:

Lecturer(l)	: l is lesgever
Course(c)	: c is een cursus
Student(s)	: s is een student
IsGrade(g)	: g is een graad
PassingGrade(g)	: g is een slaaggraad
Instructor(l,c)	: l geeft het vak c
Prerequ(c1,c2)	: vak c1 is voorkennis voor vak c2
Enrolled(s,c)	: student s is ingeschreven voor vak c
Grade(s,c,g)	: student c behaalde graad g voor vak c

Ook elke constante komt overeen met een specifiek object uit de toepassing. Bv. Ray staat voor een welbepaalde docent, B voor een specifieke score op een examen, CS230 is de cursus logica, etc.

In welk opzicht beschrijft deze databank een stand van zaken? Het beschrijft de verzameling van bestaande entiteiten in de besloten context van de toepassing: de docenten, cursussen, studenten scores die "bestaan" binnen die opleiding. Deze verzameling wordt het *universum* of *domein* genoemd van de databank. Elke entiteit wordt door een symbool voorgesteld, bv. Ray, AAA,.... Verder worden verschillende *relaties* over dit universum beschreven door middel van tabellen.

Een databank bevat veel informatie, maar van een zeer eenvoudig soort, met name *data*. Bv. dat Ray docent is van CS230. Daarentegen zijn gebruikers van een databank meestal geïnteresseerd om complexe eigenschappen te weten. Bv. is iedereen geslaagd voor het vak CS230? We vormen deze complexe uitspraken uit eenvoudige atomaire proposities door middel van de logische operatoren. Dit was het idee dat oorspronkelijk door Leibniz werd bedacht en werd uitgewerkt door Boole en Frege. Hieronder introduceren we alle operatoren van predikatenlogica aan de hand van vragen die een gebruiker zou willen stellen aan de databank. Een vraag komt overeen met een uitspraak waarvan men wil weten of deze al dan niet waar is.

Hieronder geven we een overzicht van hoe de logische symbolen $\wedge, \vee, \neg, \ldots$ gebruikt kunnen worden om eigenschappen en queries over een databank te formuleren.

- *Behaalde Sam een AAA voor CS148?*
  De uitspraak die hiermee overeenkomt is: *Sam behaalde een AAA in CS148*. Vertaling:

  $$Grade(Sam, CS148, AAA)$$

  Antwoord: waar want $(Sam, CS148, AAA)$ behoort tot de tabel van $Grade$.

  Deze uitspraak drukt uit dat $(Sam, CS148, AAA)$ tot de relatie $Grade$ behoort. In wiskundige tekst zou dit geschreven worden als:

  $$(Sam, CS148, AAA) \in Grade$$

  In predikatenlogica drukt men uit dat een n-tal $(t_1, \ldots, t_n)$ behoort tot relatie R door:

  $$\boxed{R(t_1, \ldots, t_n)}$$

  We noemen dit een *atomaire formule* of kortweg een *atoom*.

  Andere voorbeelden:
  - $Grade(Sam, CS148, B)$: onwaar
  - $Student(CS230)$: onwaar
  - $Instructor(Ray, CS230)$: waar

- *Sam behaalde geen B voor CS148?*
  Het is niet waar dat Sam een B behaalde voor CS148?
  Vertaling:

  $$\neg Grade(Sam, CS148, B)$$

  Antwoord: waar want de tabel van $Grade$ bevat geen rij $(Sam, CS148, B)$.

$\boxed{\neg \text{ betekent "het is niet waar dat"}}$

Een formule $\neg A$ is waar als en slechts als (asa) A is onwaar. Bijgevolg is een formule $\neg A$ onwaar asa A is waar.

Dit verband tussen de waarheid van A en $\neg A$ wordt uitgedrukt in de volgende *waarheidstabel*:

A	$(\neg A)$
t	**f**
f	**t**

- *Sue is docent van zowel M100 als M200.*

$$Instructor(Sue, M100) \wedge Instructor(Sue, M200)$$

Antwoord: waar. Immers, beide feiten zijn waar.

$\wedge$ betekent "en"

Een formule van de vorm $A \wedge B$ is waar asa A, B beide waar zijn. $A \wedge B$ is dus onwaar als A of B of beide onwaar zijn.

A	B	$(A \wedge B)$
t	**t**	**t**
t	**f**	**f**
f	**t**	**f**
f	**f**	**f**

Een formule van de vorm $A \wedge B$ wordt een *conjunctie* genoemd, en de deelformules A, B de *conjuncten* ervan.

- *CS148 of M100 is voorkennis voor CS230*

$$Prerequ(CS148, CS230) \vee Prerequ(M100, CS230)$$

Antwoord: waar. Immers, $Prerequ(CS148, CS230)$ is waar.

$\vee$ betekent "of"

Een formule van de vorm $A \vee B$ is waar asa minstens één van A, B waar zijn. $A \vee B$ is dus onwaar als A, B beide onwaar zijn.

A	B	$(A \vee B)$
t	**t**	**t**
t	**f**	**t**
f	**t**	**t**
f	**f**	**f**

Een formule van de vorm $A \vee B$ wordt een *disjunctie* genoemd, en de deelformules A, B de *disjuncten* ervan.

Andere voorbeelden:
- $Prerequ(CS148, CS238) \vee Prerequ(M100, CS238)$: onwaar
- $Instructor(Ray, CS230) \vee Instructor(Hec, CS230)$: waar. Beide feiten zijn waar.

- *Ofwel CS148 ofwel M100 is voorkennis voor CS230.*

  In het nederlands maken we onderscheid tussen *inclusieve* en *exclusieve disjunctie*. Een exclusieve disjunctie is waar indien exact één van beide disjuncten waar is. Bovenstaande uitspraak is een voorbeeld.

A	B	$(A\ XOR\ B)$
t	**t**	**f**
t	**f**	**t**
f	**t**	**t**
f	**f**	**f**

  Predikatenlogica heeft geen operator voor exclusieve disjunctie. Maar deze uitspraak kan uitgedrukt worden, bv. als volgt:

$$(Prerequ(CS148, CS230) \lor Prerequ(M100, CS230)) \\ \land \\ \neg(Prerequ(CS148, CS230) \land Prerequ(M100, CS230))$$

  Antwoord: waar.

- *Als Jill geregistreerd is voor CS230, dan ook Ann.*

$$Enrolled(Jill, CS230) \Rightarrow Enrolled(Ann, CS230)$$

$\boxed{\ldots \Rightarrow \ldots \text{ betekent “Indien } \ldots \text{ dan } \ldots\text{”}}$

  Antwoord: onwaar want Jill is ingeschreven voor CS230 maar Ann niet.

  Een formule van de vorm $A \Rightarrow B$ wordt een *materiële implicatie* genoemd. A wordt de *conditie* of de *premisse* genoemd, B de *conclusie*.

  De enige situatie waarin $A \Rightarrow B$ onwaar is, is als de conditie A waar is maar de conclusie B onwaar. Dus $A \Rightarrow B$ is onwaar asa A waar is en B onwaar. Omgekeerd, $A \Rightarrow B$ is waar asa A onwaar is of B waar: de premisse is onwaar of de conclusie is waar.

A	B	$(A \Rightarrow B)$
t	**t**	**t**
t	**f**	**f**
f	**t**	**t**
f	**f**	**t**

  Hieruit volgt:

$\boxed{A \Rightarrow B \text{ is logisch equivalent met } \neg A \lor B.}$

  Ander voorbeelden:
  - $Enrolled(Ann, CS230) \Rightarrow Enrolled(Jill, CS230)$: waar want premisse onwaar.
  - $Enrolled(Jill, CS230) \Rightarrow Enrolled(Jack, CS230)$: waar want conclusie waar.
  - $Enrolled(Ann, CS230) \Rightarrow Passinggrade(F)$: waar want premisse onwaar.
  - $Passinggrade(AA) \Rightarrow Passinggrade(F)$: onwaar want premisse waar, conclusie onwaar.

- *CS238 vereist voorkennis.*

  Of anders gezegd: *er bestaat een object dat voorkennis is voor CS238.*

  $$\exists x\colon Prerequ(x, CS238)$$

  Antwoord: waar omdat voor $x = CS230$ de formule $Prerequ(x, CS238)$ waar is.

  $\exists x\colon \ldots$ betekent "er bestaat een object x in het domein zodat ..."

  Het symbool $\exists$ wordt de *existentiële kwantor* genoemd. Het symbool x wordt een *logische variabele* genoemd. Dit is een symbool dat varieert over het domein. Een formule $\exists x\colon A$ wordt een *existentieel gekwantificeerde formule* genoemd.

  Om de waarheid van zo'n formule te kunnen evalueren, moet het duidelijk zijn over welke verzameling er wordt gekwantificeerd: dit is het domein (of universum) van de databank: zoals eerder gezegd de verzameling van alle objecten die voorkomen in de tabellen:

  $$\{Ray, Hec, \ldots, CS230, \ldots, Jill, \ldots, AAA, \ldots\}.$$

  Een formule $\exists x\colon A$ is waar asa er een waarde voor x in het domein van de databank bestaat waarvoor A waar is.

  De formule $\exists x\colon A$ is onwaar indien voor elke waarde voor x uit het domein de formule A onwaar is.

  Een soort waarheidstabel is als volgt:

$A[c_1]$	$\ldots$	$A[c_i]$	$\ldots$	$\exists x : A[x]$
$\ldots$	$\ldots$	**t**	$\ldots$	**t**
f	**f**	**f**	**f**	**f**

- *Ray doceert al de vakken die voorkennis zijn voor CS238.*

  Of anders gezegd, voor alle objecten x, als x een vak is en voorkennis van CS238, dan doceert Ray x:

  $$\forall x\colon Course(x) \wedge Prerequ(x, CS238) \Rightarrow Instructor(Ray, x)$$

  Antwoord: waar. Immers, voor alle objecten verschillend van $CS230$ toegekend aan x (dus voor alle studenten, docenten, graden en alle cursussen behalve $CS230$) is $Course(x)$ en $Prerequ(x, CS238)$ onwaar; voor $x = CS230$ zijn deze premisses waar maar is de conclusie $Instructor(Ray, x)$ ook waar.

  $\forall x\colon \ldots$ betekent "Voor elk object x uit het domein geldt ..."

  Het symbool $\forall$ wordt de *universele kwantor* genoemd. Een dergelijke formule wordt een *universeel gekwantificeerde formule* genoemd. Opnieuw is x een logische variabele.

  Een formule $\forall x\colon A$ is waar asa de formule A waar is voor elke waarde in het domein toegekend aan x.

$A[c_1]$	$\ldots$	$A[c_i]$	$\ldots$	$\forall x : A[x]$
t	**t**	**t**	**t**	**t**
$\ldots$	$\ldots$	**f**	$\ldots$	**f**

  De formule $\forall x\colon A$ is bijgevolg onwaar indien er een waarde uit het domein kan toegekend worden aan x zodat de formule A onwaar is.

- Is de volgende zin equivalent met de vorige?
  Voor elk object uit het domein geldt dat het geen cursus is of geen prerequisite van CS238 of het wordt gedoceerd door Ray.

  $$\forall x\colon \neg Course(x) \vee \neg Prerequ(x, CS238) \vee Instructor(Ray, x)$$

  Hier hebben we een $A \Rightarrow B$ omgezet in $\neg A \vee B$. Deze formules zijn wel degelijk equivalent. Ga dit na.

- *Er is hoogstens één student.*
  Logica beschikt niet over "hoogstens één" waardoor deze zin niet rechtstreeks in logica omgezet kan worden. We kunnen herformuleren als "voor alle objecten x en y uit het domein, als x student is en y student is dan is x gelijk aan y.

  $$\forall x\colon \forall y\colon Student(x) \wedge Student(y) \Rightarrow x = y$$

  Antwoord: onwaar; voor $x = Jack$ en $y = Jill$ geldt de premisse maar niet de conclusie $Jack = Jill$.

$\ldots = \ldots$ betekent "$\ldots$ is gelijk aan $\ldots$"

  Het symbool $=$ wordt het *gelijkheidspredikaat* genoemd.

- *Er is een student die precies één vak volgt?* Logica beschikt niet over "precies één". We kunnen simuleren als volgt:

  $$\exists x\colon \exists w\colon Student(x) \wedge Course(w) \wedge Enrolled(x, w) \wedge \forall u\colon Course(u) \Rightarrow (Enrolled(x, u) \Rightarrow u = w)$$

  Antwoord: waar: neem $x = Jack$ en $w = CS230$.

$\ldots = \ldots$ betekent "$\ldots$ is gelijk aan $\ldots$"

  Het symbool $=$ wordt het *gelijkheidspredikaat* genoemd.

- *Iemand is lesgever als en slechts als die persoon een vak doceert*:

  $$\forall x\colon Lecturer(x) \Leftrightarrow (\exists c : Course(l) \wedge Instructor(x, c))$$

  Deze zin is waar.

$\Leftrightarrow$ is het equivalentie-connectief

  $A \Leftrightarrow B$ noemt men een equivalentie.

  $A \Leftrightarrow B$ is waar indien A, B beide waar zijn of beide onwaar.

A	B	$(A \Leftrightarrow B)$
t	**t**	**t**
t	**f**	**f**
f	**t**	**f**
f	**f**	**t**

  $A \Leftrightarrow B$ is dus onwaar indien één ervan waar en de andere onwaar is.

- Wat te denken van de volgende formules in de databank?

$$VriendVan(Jill, Sam)$$

$$\exists s\colon Grade(John, CS230, s)$$

  Deze zinnen bevatten symbolen ($VriendVan, John$) die niet voorkomen in de studentendatabank. Deze zinnen kunnen niet geëvalueerd worden in de databank. Volgens logica is de waarheid van deze zinnen in de databank *onbepaald*.

Alle symbolen van predikatenlogica werden geïntroduceerd. We zijn klaar om de predikatenlogica wiskundig te definiëren.

2.2 Syntax van predikatenlogica

Predikatenlogica is een wiskundige taal. Dat betekent dat zowel de uitdrukkingen als de betekenis ervan wiskundig gedefinieerd worden. Deze sectie gaat over de syntax.

Een logische formule is een string die is samengesteld uit *symbolen* die zelf vaak gevormd zijn uit een reeks van tekens. Er zijn de volgende types van symbolen:

- **Hulptekens**: ( ) , :
- **Logische symbolen**: de betekenis van deze symbolen ligt vast in de logica.
  - Logische connectieven:
    - de conjunctie $\wedge$
    - de disjunctie $\vee$
    - de negatie $\neg$
    - de implicatie $\Rightarrow$
    - de equivalentie $\Leftrightarrow$
  - De kwantoren
    - de existiële kwantor $\exists$
    - de universele kwantor $\forall$.
  - Het gelijkheidssymbool $=$
- **Niet-logische symbolen**: de betekenis van deze symbolen ligt niet vast. Er zijn drie soorten:
  - **Objectsymbolen**, ook **constantesymbolen** genoemd.
    - Voorbeelden: $Ray, Hec, CS230, \ldots$, variabelen x, y, z.
    - We geven aan dat een symbool c een objectsymbool is door te schrijven $c/0$:.
  - **Predikaatsymbolen**, ook **relatiesymbolen** genoemd. Elk predikaatsymbool heeft een vast aantal argumenten. Dit aantal wordt de **ariteit** van het predikaatsymbool genoemd.
    - Vaak gebruikte predikaatsymbolen : P, Q, R.
    - We geven aan dat P een predikaatsymbool met ariteit n is door te noteren P/n.
    - Voorbeelden: $Grade/3, Enrolled/2, PassingGrade/1, MooiWeerVandaag/0$.
    - Een predikaatsymbool met ariteit 0 zoals $MooiWeerVandaag/0$ noemen we een **propositioneel symbool**. Het heeft geen argumenten.
  - **Functiesymbolen**, ook **functoren** genoemd, elk met een ariteit, net zoals voor predikaatsymbolen.

- Vaak gebruikte functiesymbolen : F, G.
- We geven aan dat G een functiesymbool met ariteit n is door te noteren G/n :. De ":" geeft aan dat het om een functiesymbool gaat.
- Voorbeeld: $Plus/2$:, $LeeftijdVan/1$:, respectievelijk de som functie van getallen en de leeftijdsfunctie van mensen.
- Een objectsymbool zoals Ray is een functiesymbool van ariteit 0.

Vocabularium. Een **vocabularium** is een verzameling van niet-logische symbolen. Een vocabularium wordt vaak aangeduid door het symbool Σ.

De eerste stap in een toepassing van logica is het kiezen van een aantal niet-logische symbolen om concepten uit het domein uit te drukken. Deze symbolen vormen een vocabularium waarin we vervolgens eigenschappen over het domain uitdrukken.

Bv. in de studentendatabank bestond het vocabularium uit de predikaatsymbolen (de tabelnamen), bv. $Lecturer/1$, $Grade/3$, en objectsymbolen zoals $Ray, M200, Jill, AAA, \ldots$.

Een objectsymbool uit het vocabularium wordt een **constante** genoemd. Een gekwantificeerd objectsymbool in een formule is een variabele en behoort niet tot het vocabularium.

Termen en formules Logische en niet-logische symbolen kunnen samengesteld worden tot complexe strings.

Definitie 2.2.1. Een **term** is een string die inductief gedefinieerd wordt door de volgende regels:

- Een objectsymbool is een term.
- Als $t_1, \ldots, t_n$ termen zijn en G een n-voudig functiesymbool, dan is $G(t_1, \ldots, t_n)$ een term.

Deze definitie zegt dat elke term kan bekomen worden door een eindig aantal deze regels toe te passen. Bv., gegeven

Dit is een inductieve definitie. Wat het definieert is de verzameling van termen. De definitie kan ook en misschien beter geformuleerd worden als: "De verzameling van termen is een verzameling van strings inductief gedefinieerd door de volgende regels: ...". Maar bovenstaande manier van verwoorden "Een term wordt inductief gedefinieerd ..." is gebruikelijk.

Definitie 2.2.2. Een **formule** is een string die inductief gedefinieerd wordt door de volgende regels:

- Als $t_1, \ldots, t_n$ n termen zijn en P/n een relatiesymbool van ariteit n is, dan is $P(t_1, \ldots, t_n)$ een formule. We noemen dit een **atoom**.
- Als t_1, t_2 termen zijn, dan is $t_1 = t_2$ een formule. Dit noemen we een gelijkheidsatoom.

- Als A, B formules zijn die we al geconstrueerd hebben, dan zijn $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$, $(A \Leftrightarrow B)$ ook formules.
- Als x een variabele is en A een formule, dan zijn $(\exists x\colon A)$ en $(\forall x\colon A)$ ook formules.

In beide gevallen wordt de gedefinieerde verzameling geconstrueerd wordt door iteratief toepassen van de regels. Een paar voorbeelden met $F/2:, a/0:$.

Voorbeeld 2.2.1. Is $F(a, F(a, a))$ een term? Ja, de volgende rij construeert deze:

$$\emptyset \to a \to F(a, a) \to F(a, F(a, a))$$

We kunnen ook terugwaarts redeneren:

- $F(a, F(a, a))$ is een term asa a en $F(a, a)$ termen zijn.
- $F(a, a)$ is een term asa a en a termen zijn.
- a is een term door de basisregel.

Voorbeeld 2.2.2. Is $F(a, F(a, a)$ een term? Nee, het kan nooit geconstrueerd worden. Dit kunnen we aantonen door terugwaarts te redeneren:

- $F(a, F(a, a)$ is een term asa a en $F(a, a$ termen zijn.
- $F(a, a$ kan door geen enkele regel geproduceerd worden.

Voorbeeld 2.2.3. Is de oneindige string $F(a, F(a, F(a, \dots)))$ een term? Toepassing van de regels op een verzameling van eindige termen kan enkel eindige termen afleiden. Een oneindige term kan dus nooit afgeleid worden. We kunnen hetzelfde ook aantonen door terugwaarts te redeneren:

- $F(a, F(a, F(a, \dots)))$ is een term asa a en $\dots F(a, F(a, F(a, \dots)))$ termen zijn.

Dus om te kunnen afleiden dat dit een term is, moeten we al gevonden hebben dat het een term is. Dit is onmogelijk. Dus, oneindige strings zoals deze zijn geen termen. Alle termen zijn eindig.

Om dezelfde redenen zijn er ook geen oneindige formules in de predicatenlogica. Bv.

$$P(0) \vee P(1) \vee P(2) \vee \dots$$

Intuïef is het duidelijk wat we willen uitdrukken: er bestaat een natuurlijk getal waarvoor P geldt. Maar dit is geen formule van de predikatenlogica.

Notatie 2.2.1. Formules worden genoteerd als A, B, C, termen door $t, t_1, \dots$.

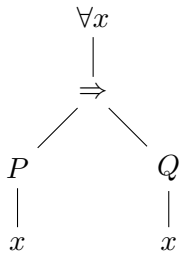
Voorbeeld 2.2.4. Neem predikaatsymbolen $S/0, P/2, Q/1, R/3$, functiesymbolen $F/1:, G/2:$ en objectsymbolen c, x, y.

- c en x zijn termen.
- $F(x), G(c, x), F(G(G(c, x), F(x)))$ zijn termen.

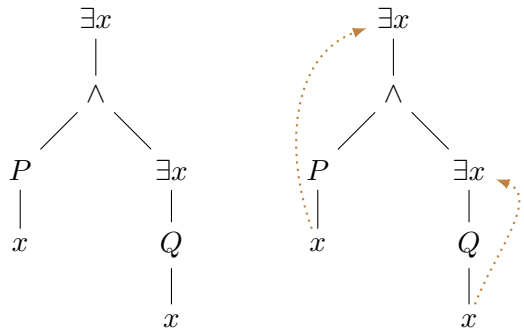
- $F(x, G(y))$ is geen term (de ariteit van F klopt niet met het aantal argumenten)
- $F(G(x, F(x)$ is geen term wegens ontbrekende haakjes.
- S is een formule (want S heeft 0 argumenten).
- $(P(x, y) \wedge \forall v\colon R(c, x, G(x, F(c))))$ is een formule.
- $S \wedge F(x)$ is geen formule (een conjunctie met een term is niet toegestaan)
- $P(c, c, c)$ is geen formule wegens een verkeerd aantal argumenten.
- $P(x, y) = Q(c)$ is geen formule (gelijkheid tussen formules is niet toegelaten)

Abstracte syntax-boom Een abstracte syntax-boom van een term of formule toont de structuur van een formule, en laat syntactische details weg, zoals waar de haakjes of komma's staan. Hier volgen een paar voorbeelden.

- $(\forall x : (P(x) \Rightarrow Q(x)))$

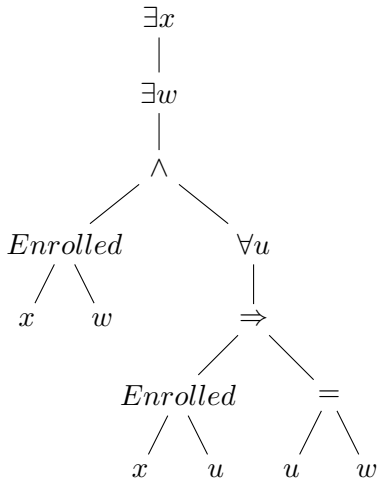


- $(\exists x : (P(x) \wedge \exists x : Q(x)))$ In deze zin wordt er twee keer gekwantificeerd over x. Deze voorkomens moeten zorgvuldig onderscheiden worden, zoals de volgende syntaxboom aangeeft. **Referentiebogen** geven aan waar een voorkomen van x gekwantififceerd is.



De voorkomens x in $P(x)$ en $Q(x)$ zijn eigenlijk verschillende variabelen, gekwantificeerd door verschillende kwantoren. De formule drukt uit dat P zowel als Q niet leeg zijn, maar niet dat ze een zelfde element bevatten.

- $(\exists x : (\exists w : (Enrolled(x, w) \wedge (\forall u : (Enrolled(x, u) \Rightarrow u = w)))))$



Een abstracte syntax-boom (of kortweg syntax-boom) toont de logische structuur beter dan een string.

Termen versus Atomen Termen denoteren objecten, atomen en formules denoteren feiten. Dit is erg verschillend. Feiten kunnen waar of onwaar zijn, termen niet.

Vrije versus gebonden symbolen Een symbool kan meerdere voorkomens hebben in een formule. Bv.

$$(Q(x, c) \wedge (\exists x\colon P(x, x)))$$

Hierin heeft x heeft 4 voorkomens, Q, P en c elk 1 voorkomen.

Definitie 2.2.3.

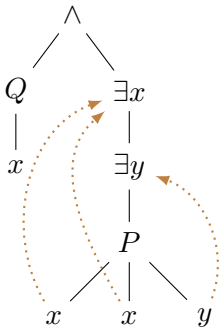
- Een voorkomen van symbool x in formule A is *gebonden* indien het zich bevindt in een deelformule $\exists x\colon B$ of $\forall x\colon B$ van A.
- Een voorkomen van x in A is *vrij* indien niet gebonden.
- Een symbool τ is een **vrij symbool** van een formule of een term indien het er een vrij voorkomen in heeft.

We noteren met $Vrij(A)$ de verzameling van niet-logische symbolen met vrije voorkomens in A. Dit bestaat uit alle niet-logische symbolen tenzij diegene die enkel gebonden voorkomen. Enkel variabelen hebben gebonden voorkomens.

Voorbeeld 2.2.5.

$$(Q(x) \wedge (\exists x\colon (\exists y\colon P(x, x, y))))$$

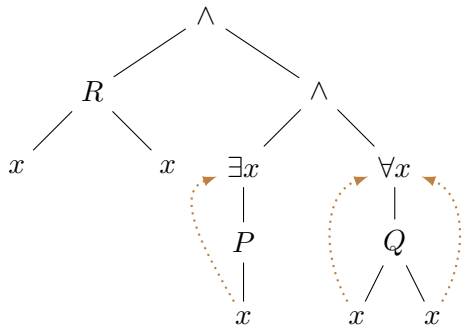
De syntax-boom toont dat x een vrij voorkomen heeft (zonder referentie-boog) en twee gebonden voorkomens die refereren naar dezelfde kwantificatie. De predikaatsymbolen Q en P hebben elk 1 voorkomen en dit is vrij. y heeft enkel gebonden voorkomens. De verzameling van vrije symbolen is $\{Q, P, x\}$.



Voorbeeld 2.2.6.

$$(R(x,x) \wedge ((\exists x\colon P(x)) \wedge (\forall x\colon Q(x,x))))$$

Zoals de syntax-boom laat zien kunnen de voorkomens van x opgesplitst worden in 3 groepen: alle vrije voorkomens, en twee kwantificaties van x. In feite zijn deze 3 groepen te beschouwen als verschillende variabelen of symbolen.



Het is verwarrend om dezelfde variabele met verschillende betekenissen te hergebruiken in een formule. Men vermijdt dat best zoveel mogelijk.

De predikatenlogicataal van Σ

Definitie 2.2.4. t is een *term over* Σ als elk vrij symbool van t behoort tot Σ.

Een formule A is een **zin** over Σ als elk vrij symbool van A behoort tot Σ.

Een **theorie** over Σ is een verzameling van zinnen over Σ.

Om te bepalen of een formule A een zin is over Σ wordt dus geen rekening gehouden met de variabelen met gebonden voorkomens.

Definitie 2.2.5. De verzameling van alle zinnen van Σ wordt de **predikatenlogicataal** van Σ genoemd.

Stel dat we in een toepassing een vocabularium Σ opgesteld hebben. De symbolen in Σ zijn de enige symbolen waaraan we een betekenis, een concept in het toepassingsdomein hebben toegekend. Dan zijn enkel termen en zinnen over Σ betekenisvol in dit toepassingsdomein. Dan is de predikatenlogicataal van Σ de verzameling van alle betekenisvolle zinnen over de toepassing die we kunnen uitdrukken in predikatenlogica. Zinnen die andere vrije symbolen bevatten hebben geen betekenis voor ons.

Voorbeeld 2.2.7. In de context van de studentendatabank zagen we $VriendVan(Jill, Sam)$ en $\exists x : Grade(John, CS230, x)$. Zij bevatten vrije symbolen $VriendVan$ en $John$ die niet voorkomen in de studentendatabank en dus niet behoren tot de predikatenlogicataal van deze toepassing.

Voorbeeld 2.2.8.

$$(Q(x) \wedge (\exists x\colon (\exists y\colon P(x, c, y))))$$

Deze formule is een zin over het vocabularium $\Sigma = \{P/3, Q/1, x/0 :, c/0 :\}$. Het is geen zin over $\Sigma = \{P/3, Q/1, c/0 :\}$ omdat x vrij voorkomt. De volgende zin is wel een zin over $\Sigma = \{P/3, Q/1, c/0 :\}$:

$$(\exists x\colon (Q(x) \wedge (\exists x\colon (\exists y\colon P(x, c, y)))))$$

Soorten formules. Een formule $R(t_1, \ldots, t_n)$ noemen we een *atoom*.

Een formule $t_1 = t_2$ noemen we een *gelijkheidsatoom* en $=$ is de gelijkheidsrelatie.

Een formule $A \wedge B$ noemen we een *conjunctie*, met A en B de *conjuncten*.

Een formule $A \vee B$ noemen we een *disjunctie*, met A en B de *disjuncten*.

Een formule $A \Rightarrow B$ noemen we een *implicatie*, A de *premisse* of *conditie* en B de *conclusie*.

Een formule $\neg A$ noemen we een *negatie*.

De negatie van een gelijkheidsatoom $\neg(t_1 = t_2)$ wordt vaak genoteerd als $t_1 \neq t_2$. Dit is slechts notatie; in predikatenlogica bestaat geen predikaat $\neq$.

Prefix- of infix-notatie Een atoom $t_1 = t_2$ is een speciale schrijfwijze van het atoom $=(t_1, t_2)$. We noemen $t_1 = t_2$ een *infix*-notatie, omdat $=$ in het midden van zijn argumenten gezet worden. We noemen $R(t_1, t_2)$ een *prefix*-notatie omdat R voor zijn argumenten staat.

In predikaatlogica wordt de prefix-notatie gebruikt voor niet-logische predikaat- en functiesymbolen en de infix-notatie voor $=$ en trouwens ook voor de connectieven $A \wedge B, A \Rightarrow B, \ldots$. De infix-notatie zal later ook vaak gebruikt worden voor numerieke symbolen zoals $<, \leq$ en functiesymbolen $+, \times$.

Over de haakjes. Omwille van de duidelijkheid zullen we soms afwijken van de strikte vormingsregels van de formules. Er zijn verschillende technieken.

- Gebruik verschillende soorten haakjes (), [] en {} om duidelijker subformules af te bakenen.
- Splits complexe formules over verschillende lijnen en gebruik indentatie. Bv. vergelijk:

$$\begin{aligned}&\forall x\colon ((\exists w\colon S(w, w, x)) \wedge (\exists u\colon \exists v\colon \exists w\colon [L(u, v) \wedge L(v, w) \wedge L(w, x)])\\ &\Rightarrow (\exists y\colon \exists z\colon [P(y) \wedge P(z) \wedge S(y, z, x)])\end{aligned}$$

met

$$\begin{array}{ll} \forall x\colon & \{ \\ & \quad [(\exists w\colon S(w,w,x)) \wedge (\exists u\colon \exists v\colon \exists w\colon [L(u,v) \wedge L(v,w) \wedge L(w,x)]] \\ & \quad \Rightarrow \\ & \quad [\exists y\colon \exists z\colon [P(y) \wedge P(z) \wedge S(y,z,x)]] \\ & \} \end{array}$$

Indentatie verhoogt leesbaarheid en doet een formule meer lijken op een Abstracte Syntax Boom.

- Haakjes mogen weggelaten worden indien het duidelijk is waar ze terug ingevoegd moeten worden. Hiervoor bestaan strikte regels.
  - Buitenste haakjes mogen weg.
  - Conventies om haakjes weg te laten zijn zoals in rekenkundige formules — bv. $1 \times 3 + 4$ staat voor $(1 \times 3) + 4$. Waar haakjes terug ingevoerd worden, wordt bepaald door *voorrangsregels* en sterkte van binding.

    $$\neg > \wedge > \vee > \Rightarrow > \Leftrightarrow > \forall, \exists$$

    Dus $\neg$ bindt sterker dan alle andere connectieven.
  - Voor identieke connectieven is de binding rechts sterker dan links.

Voorbeelden:

- $\neg P \wedge Q(c)$ staat voor:
  - $((\neg P) \wedge Q(c))$,
  - en niet $(\neg(P \wedge Q(c)))$.

  want $\neg > \wedge$.
- $Q(x) \vee P(c,c) \wedge Q(c)$ staat voor:
  - $(Q(x) \vee (P(c,c) \wedge Q(c)))$ of
  - en niet $((Q(x) \vee P(c,c)) \wedge Q(c))$.

  want $\wedge > \vee$.
- $Q \vee P \Rightarrow Q \wedge R$ staat voor $((Q \vee P) \Rightarrow (Q \wedge R))$
- $\exists x\colon P(x) \wedge Q(x)$ staat voor
  - $(\exists x\colon (P(x) \wedge Q(x))$,
  - niet voor $((\exists x\colon P(x)) \wedge Q(x))$?

  want $\wedge > \exists$.
- $Q(x) \vee P \vee Q(c)$ staat voor
  - $(Q(x) \vee (P \vee Q(c)))$
  - en niet voor $((Q(x) \vee (P \vee Q(c))$.

want de binding van $\wedge$ rechts is sterker dan links.

- $Q(x) \Rightarrow P(c,c) \Rightarrow Q(c)$ staat voor
  - $(Q(x) \Rightarrow (P(c,c) \Rightarrow Q(c)))$
  - en niet voor $((Q(x) \Rightarrow (P(c,c)) \Rightarrow Q(c))$.

De strikte regels dienen ervoor dat de betekenis van de formules duidelijk blijft. Daarom moet het duidelijk zijn waar de haakjes terug ingevoegd moeten worden.

Let op:

- Teveel haakjes weg laten kan ook de leesbaarheid van de formule verminderen. Bv. $P \vee (Q \wedge R)$ lijkt duidelijker dan $P \vee Q \wedge R$.
- Een afgekorte formule is geen formule!

  In het nederlands is "bv." geen woord maar enkel een afgekorte schrijfwijze van het woord "bijvoorbeeld". Analoog is een verkorte formule $P(x,y) \vee Q(c) \wedge Q(y)$ geen formule, wel een *afkorting* van een formule.
- Niet elk substring van een formule is een deelformule is. Bv. "$\wedge Q$" is een deelstring van "$P \wedge Q$" maar geen deelformule want het is geen formule.

  Maar neem de afkorting $P(x,y) \vee Q(c) \wedge Q(y)$: dan is $P(x,y) \vee Q(c)$ wel een deelstring met de vorm van een (afgekorte) formule. Toch is het geen deelformule. Immers, de bedoelde formule $(P(x,y) \vee (Q(c) \wedge Q(y)))$ en daarin komt $(P(x,y) \vee Q(c))$ niet voor.

Oefening 2.2.1. *Schrijf de ASB op voor de volgende formules waar haakjes zijn weggelaten:*

1. $\exists x\colon P(x) \wedge Q(x)$
2. $\neg P(c) \wedge Q(c)$
3. $Q(x) \vee P(c,c) \wedge Q(c)$
4. $Q \vee P \vee R$
5. $Q \Rightarrow P(c,c) \Rightarrow Q(c)$

Oefening 2.2.2. *Ga na of $P(x,y) \vee Q(c)$ een deelformule is van $P(x,y) \vee Q(c) \wedge Q(y)$ met behulp van de ASB van die laatste.*

Oefening 2.2.3. *Stel de AST op van de volgende twee zinnen en probeer op die manier het verschil in hun betekenis te achterhalen.*

$\exists x : (Triangle(x) \wedge \forall y : (Square(y) \wedge \forall z : (Pentagon(z) \Rightarrow RightOf(y,z)) \Rightarrow FrontOf(x,y)))$

$\exists x : (Triangle(x) \wedge \forall y : (Square(y) \wedge (\forall z : (Pentagon(z) \Rightarrow RightOf(y,z))) \Rightarrow FrontOf(x,y)))$

Oefening 2.2.4. *Stel de abstracte syntax-boom op van de volgende zin :*

$$\begin{aligned} \forall x\colon \forall w\colon \ & \{Enrolled(x,w) \Rightarrow \\ & \quad \forall t\colon [Prerequ(t,w) \Rightarrow \exists s\colon (\ \ Grade(x,t,s) \wedge \\ & \qquad\qquad\qquad\qquad\qquad\qquad PassingGrade(s))]\} \end{aligned}$$

Wat betekenen dit?

Opmerking 2.2.1. Het systeem LogicPalet wordt gebruikt in de oefenzittingen van de cursus en ook voor de huiswerken. LogicPalet ondersteunt zowel een Unicode-syntax als een ASCII-syntax waarin symbolen

$$\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow, \neq, \forall, \exists$$

vervangen zijn door de ASCII-karakters

$$\&\,, |\,, \sim\,, =>\,, <=>\,, \sim=\,, !\,, ?$$

LogicPalet laat toe van Unicode-syntax te vertalen naar ASCII-syntax en omgekeerd, en ook naar de syntax van Spass en van het IDP systeem.

LogicPalet laat toe om formules uit electronische documenten (pdf) met copy en paste over te brengen naar LogicPalet.

2.3 Formele semantiek van predikatenlogica

De vorige sectie bevatte een wiskundige theorie van de syntax van logica, die de syntactische vorm van alle logische formules beschrijft. Deze sectie geeft een wiskundige definitie van de semantiek van logica. Dit noemt men de formele semantiek van de logica. De formele semantiek is een wiskundige theorie die de betekenis van alle logische formules beschrijft.

Hoe beschrijven we de betekenis van logische zinnen? Het basisidee is aanwezig in werk van de filosoof Wittgenstein, die beweerde dat het begrijpen van wat een zin betekent, neer komt om te weten in welke situaties de zin waar is en in welke situaties de zin onwaar is. Dat impliceert bv. dat om in te zien dat twee zinnen een verschillende betekenis hebben, men in staat moet zijn om een situatie te bedenken waarin de ene waar en de andere onwaar is. Als je geen zo'n situatie vind, dan zijn de zinnen equivalent. In de context van filosofische logica wordt een situatie vaak een **stand van zaken** (Engels: state of affairs) genoemd.

Dit idee kan rechtstreeks vertaald worden in wiskunde. Men introduceert wiskundige objecten $\mathfrak{A}$, **structuren** genaamd. Het zijn wiskundige abstracties, abstracte wiskundige voorstellingen van "situaties". Vervolgens definieert men de **waarheidsrelatie** tussen structuren en formules: deze wordt genoteerd als $\mathfrak{A} \models A$. Dit betekent: de formule A is waar in de structuur $\mathfrak{A}$.

Hoe kunnen we "situaties" op een wiskundige manier voorstellen? We zagen al een voorbeeld: de studentendatabank is een duidelijk voorbeeld van een abstracte voorstelling van een "situatie". Het is het eerste voorbeeld van een structuur in deze cursus.

Structuren, en symbolen versus waardes We beschrijven de betekenis van formules door te beschrijven in welke formele situaties ze **waar** zijn. Het is duidelijk dat het concept van waarheid niet absoluut is, maar relatief tot een situatie (of "stand van zaken"). Bv.

$$\neg\exists x : Course(x) \wedge Prerequ(x, M100)$$

is waar in de situatie voorgesteld door de studentendatabank in Figuur 2.1, maar is onwaar in de alternatieve situatie beschreven door in de databank de rij $(CS148, M100)$ toe te voegen aan de tabel van $Prerequ$.

Er bestaan wel uitspraken die absoluut waar zijn, zoals bv. 1+1=2, maar in dit geval komt dit omdat de waardes van deze symbolen dezelfde zijn in alle situaties die we bekijken. In de predicatenlogica zijn de enige symbolen met een vaste waarde de logische symbolen.

De essentie van een "situatie" is dat het een unieke, welgedefinieerde **waarde** bepaalt voor elk niet-logisch symbool. Bv. de waarde van $Prerequ$ in de studentendatabank is de relatie $\{(CS230, CS238), (CS148), CS230), (M100, M200)\}$.

Eens de waarde van de logische symbolen vastligt, en de waarde van niet-logische symbolen gegeven is door een structuur, dan ligt de waarheid van logische zinnen vast, relatief tot de structuur.

Definitie 2.3.1. Een **structuur** $\mathfrak{A}$ bestaat uit een niet-lege verzameling $D_{\mathfrak{A}}$, het **domein** of **universum** van $\mathfrak{A}$, en een toekenning van waarden $\tau^{\mathfrak{A}}$ aan niet-logische symbolen τ:

- De waarde $c^{\mathfrak{A}}$ voor een objectsymbool c is een element uit het domein $D_{\mathfrak{A}}$.
- De waarde $F^{\mathfrak{A}}$ voor een functie-symbool F/n : is een totale functie $F^{\mathfrak{A}} : {D_{\mathfrak{A}}}^n \to D_{\mathfrak{A}}$. Dit is een functie die n-tallen $(a_1, \ldots a_n)$ uit het domein op elementen van het domein afbeeldt.
- De waarde $P^{\mathfrak{A}}$ voor een predikaatsymbool P/n is een n-voudige relatie $P^{\mathfrak{A}}$ in $D_{\mathfrak{A}}$, dus $P^{\mathfrak{A}} \subseteq {D_{\mathfrak{A}}}^n$.

We noemen $\tau^{\mathfrak{A}}$ de **waarde** of de **interpretatie** van τ in $\mathfrak{A}$.

De verzameling van symbolen symbolen die een interpretatie hebben in $\mathfrak{A}$ wordt genoteerd als $\Sigma(\mathfrak{A})$. We zeggen dat $\mathfrak{A}$ een structuur is over Σ indien $\Sigma = \Sigma_{\mathfrak{A}}$. We zeggen dat $\mathfrak{A}$ het vocabularium Σ interpreteert indien $\Sigma \subseteq \Sigma(\mathfrak{A})$.

Een structuur kent een *waarde* toe aan elk symbool van Σ. Het bepaalt ook het *domein* of *universum* $D_{\mathfrak{A}}$ en daarmee bepaalt het ook over welke objecten we spreken als we zeggen "voor alle x geldt dat ..." of "er bestaat x zodat ...". Het domein van een structuur kan bestaan uit eender wat: uit strings, getallen, lijsten, bomen, of andere wiskundige objecten, maar ook uit bv. alle Belgen, alle 1ste bachelors van de KUL, al mijn familieleden, enz. of de unie van dergelijke verzamelingen.

De kracht van deze notie van structuur ligt in het feit dat deze notie van structuur een uiterst krachtige manier is om zeer uiteenlopende situaties te beschrijven: studentendatabank, computerconfiguraties, familierelaties, contracten om woningen te kopen, etc.

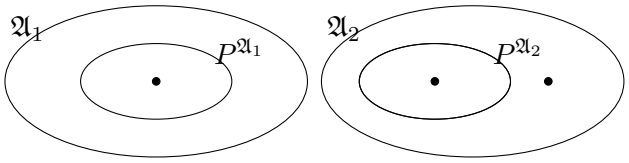
Notatie van structuren Structuren zijn belangrijk in deze cursus. In veel problemen is een structuur gegeven of moet een structuur opgesteld worden. Vandaar dat een goede notatie nodig is.

Voorbeeld 2.3.1. Zij $\Sigma = \{P/1\}$. We introduceren twee structuren over Σ met dezelfde waarde voor het symbool P maar met andere domeinen:

- $\mathfrak{A}_1$: $D_{\mathfrak{A}_1} = \{a\}$; $P^{\mathfrak{A}_1} = \{a\}$.
- $\mathfrak{A}_2$: $D_{\mathfrak{A}_2} = \{a, b\}$; $P^{\mathfrak{A}_2} = \{a\}$.

Hier is $P/1$ het enige element van Σ. Hoe zit het dan met a en b? Wel, om iets symbolisch op te schrijven heeft men symbolen nodig. De symbolen a en b dienenom het domein van $\mathfrak{A}_1, \mathfrak{A}_2$ te kunnen opschrijven. Ze behoren niet tot Σ.

Voorbeeld 2.3.2. De structuren $\mathfrak{A}_1$ en $\mathfrak{A}_2$ kunnen ook grafisch voorgesteld worden. Hierbij wordt het universum $D_{\mathfrak{A}}$ en de interpretaties van unaire predicaten voorgesteld door Venndiagrammen die punten bevatten.



De domeinelementen zijn punten op het vlak. Namen voor domeinelementen zijn in principe overbodig.

De grafische notatie is beperkt tot eenvoudige structuren, voor vocabularia bestaande uit unaire en binaire predicaatsymbolen en objectsymbolen en unaire functiesymbolen. Een objectsymbool wordt geplaatst naast het punt dat zijn waarde is in $\mathfrak{A}$. De waarde van een binair predicaatsymbool $Q/2$ wordt weergeven als een verzameling van bogen $\xrightarrow{Q^{\mathfrak{A}}}$. Idem voor de waarde van een unair functiesymbool.

Oefening 2.3.1. *We hebben dezelfde structuren $\mathfrak{A}_1, \mathfrak{A}_2$ op verschillende manieren genoteerd. Zorg dat je het verband ziet. Waar liggen a en b in de grafische notatie? Ga na in de grafische notatie in welke structuur de formules $\forall x : P(x)$ en $\exists x : \neg P(x)$ waar zijn.*

Voorbeeld 2.3.3. De structuur van de natuurlijke getallen met de somfuncie en de "kleiner dan'- relatie vormt een structuur. Neem $\Sigma = \{zero/0 :, plus/2 :, \leq /2\}$. De structuur $\mathfrak{A}$ wordt als volgt genoteerd:

- $D_{\mathfrak{A}} = \mathbb{N}$;
- $zero^{\mathfrak{A}} = 0$;
- $plus^{\mathfrak{A}} = \{((n, m), n + m) | n, m \in \mathbb{N}\}$;
- $\leq^{\mathfrak{A}} = \{(n, m) \in \mathbb{N}^2 | n \leq m\}$.

Soms schrijft men de tupels in een functie als in $(n, m) \to n+m$ in plaats van als $((n, m), n+m)$.

Een speciaal geval: structuren van propositionele vocabularia Een propositioneel vocabularium Σ bestaat uit propositionele symbolen, relatiesymbolen van ariteit 0. Volgens de definitie van structuur is de waarde van zo'n symbool "een relatie van ariteit 0". Dit zijn de twee booleaanse waardes $\mathbf{f}$ en $\mathbf{t}$. Een propositionele structuur heeft geen nood aan een universum.

Definitie 2.3.2. Een propositionele structuur $\mathfrak{A}$ voor Σ uit een waarde $P^{\mathfrak{A}} \in \{\mathbf{t}, \mathbf{f}\}$ voor elk symbool $P \in \Sigma$.

Anders gezegd, $\mathfrak{A}$ is een functie van Σ naar $\{\mathbf{t}, \mathbf{f}\}$.

Een propositionele structuur kan geschreven zoals in $\mathfrak{A} = \{(DeZonSchijnt, \mathbf{f}), (HetRegent, \mathbf{t})\}$. Deze beschrijft de stand van zaken waarin het regent en de zon niet schijnt.

Maar als Σ duidelijk is, bv. $\Sigma = \{DeZonSchijnt, HetRegent\}$, dan noteert men vaak de structuur als de verzameling van *ware* symbolen; in dit geval als $\mathfrak{A} = \{HetRegent\}$.

In wiskunde en wetenschappen gebruikt men vaak verschillende wiskundige formaten om hetzelfde object te beschrijven. Hierboven zijn $\{(DeZonSchijnt, \mathbf{f}), (HetRegent, \mathbf{t})\}$ en $\{HetRegent\}$ twee verschillende verzamelingen die gebruikt worden om **dezelfde** structuur te denoteren. Laat u hierdoor niet verwarren!

Herbrand- en databankstructuren Er is doorgaans een verschil tussen een symbool en een waarde. Bv. '3' en '12' zijn symbolen, maar de getallen waarvoor ze staan zijn geen symbolen. Het symbool "Koning Filip I" is een symbool maar onze koning is een mens van vlees en bloed. Een vocabularium Σ is een verzameling van symbolen τ maar hun waardes $\tau^{\mathfrak{A}}$ in een structuur zijn doorgaans waardes en geen symbolen.

Er is echter een uitzondering: Σ-Herbrandstructuren $\mathfrak{A}$. Daarin bestaat het domein $D_{\mathfrak{A}}$ uit de verzameling van termen die gevormd kunnen worden uit Σ.

Zij Σ een niet-propositioneel vocabularium.

Definitie 2.3.3. Een structuur $\mathfrak{A}$ van Σ is een Herbrand-structuur van Σ indien het de volgende voorwaarden voldoet:
- $D_{\mathfrak{A}}$ is de verzameling van termen over Σ.
- Voor elk objectsymbool $c \in \Sigma$ geldt $c^{\mathfrak{A}} = c$.
- Voor elk functiesymbool $F \in \Sigma$ is $F^{\mathfrak{A}}$ de functie die termen $t_1, \ldots, t_n$ over Σ afbeeldt op de term $F(t_1, \ldots, t_n)$.

Ga na dat voor functiesymbool F/n, $F^{\mathfrak{A}}$ wel degelijk een functie is en dus geen element van $D_{\mathfrak{A}}$.

Als Σ geen functiesymbolen bevat dan bestaat $D_{\mathfrak{A}}$ uit de objectsymbolen in Σ en er geldt voor elk objectsymbool $C \in \Sigma$ dat $C^{\mathfrak{A}} = C$. Dit is het geval voor een databank DB, wiens het vocabularium Σ bestaat uit de tabelpredicaten en objectsymbolen in tabellen. Een databank DB zoals de studentendatabank zullen we beschouwen als een Herbrand-structuur $\mathfrak{A}_{DB}$ zodat voor elk predicaatsymbool $P/n \in \Sigma$ geldt dat $P^{\mathfrak{A}_{DB}}$ bestaat uit alle tupels in de tabel van P.

Voorbeeld 2.3.4. De studentendatabank in Figuur 2.1 definieert een vocabularium Σ_{DB} en een Herbrand-structuur $\mathfrak{A}_{DB}$. In de inleiding van dit hoofdstuk hebben we alvast tientallen zinnen in deze structuur geëvalueerd, gebaseerd op ons intuïtief begrip van de logische symbolen.

De studentendatabank is een formele structuur van Σ. Maar dat is niet het eind van het verhaal. In een applicatie heeft de gebruiker een informele interpretatie van de symbolen in Σ als entiteiten en relaties in het applicatiedomein. Bv. *Ray* stelt een docent voor, en *Enrolled* de relatie tussen studenten en de vakken waarvoor ze hebben geregistreerd. De actuele stand van zaken van het applicatiedomein komt overeenkomt met een informele structuur $\mathfrak{A}'$ van Σ. De databankstructuur $\mathfrak{A}_{DB}$ dient om die actuele stand van zaken $\mathfrak{A}'$ voor te stellen. Een databank is **correct** indien de formele databankstructuur $\mathfrak{A}_{DB}$ *isomorf* is met deze structuur $\mathfrak{A}$. Bv. zodat een atoom zoals $Instructor(Ray, M100)$ waar is in de databankstructuur $\mathfrak{A}_{DB}$ asa Ray het vak M100 doceert in de actuele stand van zaken.

De actuele stand van zaken komt overeen met slechts één Herbrand structuur die correct is. Alle andere zijn niet correct.

Voorbeeld 2.3.5. We definiëren een alternatieve Σ-structuur $\mathfrak{B}$. Het domein $D_{\mathfrak{B}}$ is het singleton $\{1\}$. Met dit domein ligt de waarde vast van elk object en zijn er maar 2 mogelijke n-aire relaties: de lege verzameling en $\{(\underbrace{1,\ldots,1}_{n})\}$. Voor $\mathfrak{B}$ kiezen we de tweede mogelijkheid:

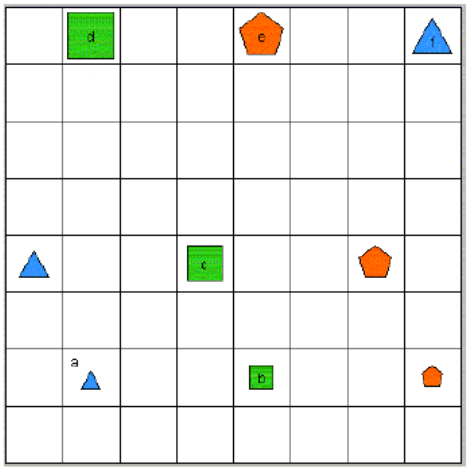
- $D_{\mathfrak{B}} = \{1\}$
- voor elk objectsymbool C geldt $C^{\mathfrak{A}} = 1$. Bv., $Ray^{\mathfrak{A}} = AAA^{\mathfrak{A}} = Jill^{\mathfrak{A}} = \cdots = 1$.
- de waarde van elk predikaatsymbool P/n is $\{(1,\ldots,1)\}$. Bv., $Lecturer^{\mathfrak{B}} = \{1\}$, $Instructor^{\mathfrak{B}} = \{(1,1)\}$, $Grade^{\mathfrak{A}} = \{(1,1,1)\}$.

Deze $\mathfrak{B}$ is duidelijk geen Herbrand-structuur en is absoluut niet *correct* in de zin zoëven gedefinieerd.

Nochtans heeft $\mathfrak{B}$ een interessante eigenschap, namelijk elke atomische formule en dus elk expliciet feit opgeslagen in de databank is waar in $\mathfrak{B}$. Bv. $Enrolled(Jill, CS2390)$ is waar in $\mathfrak{B}$. Immers $Jill^{\mathfrak{B}} = CS230^{\mathfrak{B}} = 1$ en $(1,1) \in Enrolled^{\mathfrak{B}}$. Blijkbaar volstaat het niet voor een structuur om alle feiten uit een databank waar te maken om correct te zijn!

Oefening 2.3.2. *Definieer een vocabularium Σ om 1 bankenrij van boven tot onder uit het auditorium te beschrijven en welke studenten waar zitten. Je wilt kunnen bepalen welke posities vrij zijn, wie naast wie zit, wie aan de zijkant zit, wie op de 4-de rij zit.*

Oefening 2.3.3. *Definieer een vocabularium Σ en een Σ-structuur $\mathfrak{A}$ die precies de geoworld in Figuur 2.3.3 beschrijft. Een aantal figuren komen voor op posities in een rooster. Deze figuren zijn 3-, 4- of 5-hoeken, en zijn klein of groot.*



Figuur 2.2: Een Geo-wereld

De waarde van termen en formules in een structuur De verzameling van termen en formules die een waarde hebben in een structuur $\mathfrak{A}$ hangt af van de verzameling van symbolen τ die geïnterpreteerd zijn in $\mathfrak{A}$, dus waarvoor $\tau^{\mathfrak{A}}$ bestaat.

Definitie 2.3.4. Voor een structuur $\mathfrak{A}$ definiëren we $Symbols(\mathfrak{A})$ als de verzameling van symbolen die een interpretatie hebben in $\mathfrak{A}$.

Een structuur $\mathfrak{A}$ **interpreteert** een term t of logische formule A indien elk vrij symbool van t of A een interpretatie heeft in $\mathfrak{A}$ (t.t.z. als $Vrij(A) \subseteq Symbols(\mathfrak{A})$).

Notatie 2.3.1. Zij $\mathfrak{A}$ een structuur, τ een symbool en v een geldige waarde voor het symbool τ in het domein van $\mathfrak{A}$. Dan is $\mathfrak{A}[\tau : v]$ de structuur die identiek is aan $\mathfrak{A}$ behalve dat $\tau^{\mathfrak{A}[\tau:v]} = v$. In het algemeen, $\mathfrak{A}[\tau_1 : v_1, \ldots, \tau_n : v_n]$ is hetzelfde als $\mathfrak{A}$ behalve dat het de waardes v_i toekent aan symbool τ_i.

Dus, als τ geen waarde heeft in $\mathfrak{A}$ dan breidt $\mathfrak{A}[\tau : v]$ de structuur $\mathfrak{A}$ uit met een waarde voor τ. Als τ wel een waarde heeft in $\mathfrak{A}$, dan wordt deze waarde overschreven door v.

Als $\mathfrak{A}$ een term t interpreteert (dus al zijn symbolen een waarde geeft), dan denoteert t een uniek element $t^{\mathfrak{A}}$ in het domein van $\mathfrak{A}$. Dit definiëren we als volgt.

Definitie 2.3.5. Zij gegeven een structuur $\mathfrak{A}$ die een term t interpreteert. We definiëren de interpretatie $t^{\mathfrak{A}}$ van t in $\mathfrak{A}$ door inductie op het aantal symbolen van t:

- Als t een object-symbool is dan is $t^{\mathfrak{A}}$ exact de interpretatie van t in $\mathfrak{A}$.
- Als t een term $F(t_1, \ldots, t_n)$ is, dan is $t^{\mathfrak{A}} = F^{\mathfrak{A}}(t_1^{\mathfrak{A}}, \ldots, t_n^{\mathfrak{A}})$, de functie $F^{\mathfrak{A}}$ toegepast op het koppel bestaande uit de interpretaties van $t_1, \ldots, t_n$.

Dit is een recursieve definitie van een functie: de functie die een term op zijn waarde afbeeldt. Het is een definitie **per inductie op het aantal symbolen in** t. Daarmee wordt bedoeld dat voor samengestelde termen $F(t_1, \ldots, t_n)$, de interpretatie ervan gedefinieerd wordt als een functie van de interpretaties van termen met stikt minder symbolen. Inderdaad, de termen $t_1, \ldots, t_n$ hebben strikt minder symbolen dan t.

We kunnen ook zeggen dat dit een **definitie per inductie op de structuur van** t is: daarmee bedoelen we dat de interpretatie van $F(t_1, \ldots, t_n)$ gedefiniëerd wordt in termen van de interpretaties van deeltermen $t_1, \ldots, t_n$ van $F(t_1, \ldots, t_n)$. Bovenstaande definitie is zowel door inductie op het aantal symbolen als door inductie op de structuur, aangezien $t_1, \ldots, t_n$ zowel minder symbolen hebben dan $F(t_1, \ldots, t_n)$ als deeltermen ervan zijn.

Oefening 2.3.4. *Toon aan met een inductiebewijs dat als $\mathfrak{A}$ een Herbrand-structuur is, dan geldt voor elke term t dat $t^{\mathfrak{A}} = t$. De waarde van een term is zichzelf.*

Definitie 2.3.6. Zij A een formule, $\mathfrak{A}$ een structuur die A interpreteert.

We definiëren dat A *waar* is in $\mathfrak{A}$ (notatie $\mathfrak{A} \models A$) door middel van inductie op het aantal symbolen in A (men zegt ook: door inductie op de structuur van A):

- $\mathfrak{A} \models P(t_1, \ldots, t_n)$ asa $(t_1^{\mathfrak{A}}, \ldots, t_n^{\mathfrak{A}}) \in P^{\mathfrak{A}}$. (Atoom-regel)
- $\mathfrak{A} \models t_1 = t_2$ asa $t_1^{\mathfrak{A}} = t_2^{\mathfrak{A}}$. (=-regel)
- $\mathfrak{A} \models \neg A$ asa het niet het geval is dat $\mathfrak{A} \models A$; in wiskundige notatie, asa $\mathfrak{A} \not\models A$. ($\neg$-regel)
- $\mathfrak{A} \models A \wedge B$ asa $\mathfrak{A} \models A$ en $\mathfrak{A} \models B$. ($\wedge$-regel)
- $\mathfrak{A} \models A \vee B$ asa $\mathfrak{A} \models A$ of $\mathfrak{A} \models B$ (of allebei). ($\vee$-regel)

- $\mathfrak{A} \models A \Rightarrow B$ asa $\mathfrak{A} \not\models A$ of $\mathfrak{A} \models B$ (of allebei). ($\Rightarrow$-regel)
- $\mathfrak{A} \models A \Leftrightarrow B$ asa $\mathfrak{A} \models A$ en $\mathfrak{A} \models B$ of als $\mathfrak{A} \not\models A$ en $\mathfrak{A} \not\models B$. M.a.w. als A en B dezelfde waarheidswaarde hebben. ($\Leftrightarrow$-regel)
- $\mathfrak{A} \models \exists x\colon A$ asa er een domeinelement $a \in D_{\mathfrak{A}}$ bestaat zodat $\mathfrak{A}[x : a] \models A$. ($\exists$-regel)
- $\mathfrak{A} \models \forall x\colon A$ asa er voor elk domeinelement $a \in D_{\mathfrak{A}}$ geldt dat $\mathfrak{A}[x : a] \models A$. ($\forall$-regel)

We definiëren dat A *onwaar* is in $\mathfrak{A}$ (notatie $\mathfrak{A} \not\models A$) als A niet waar is in $\mathfrak{A}$.

De waarheidswaarde van A in $\mathfrak{A}$ is *waar* indien $\mathfrak{A} \models A$ en *onwaar* indien $\mathfrak{A} \not\models A$. We noteren deze waarde als $A^{\mathfrak{A}}$.

De waardes *waar* en *onwaar* worden de booleaanse waardes genoemd. We korten ze af tot **t** (true) en **f** (false), of soms tot 1 en 0. De verzameling $\{\mathbf{t}, \mathbf{f}\}$ van booleaanse waarden wordt genoteerd als $\mathbb{B}$.

Terminologie 2.3.1. Als $\mathfrak{A} \models A$, zeggen we ook dat

- A is waar in $\mathfrak{A}$
- A is voldaan in $\mathfrak{A}$
- $\mathfrak{A}$ voldoet aan A
- $\mathfrak{A}$ is een **model** van A

$\mathfrak{A}$ is een model van een theorie T (een verzameling zinnen) als elke zin in T waar is in $\mathfrak{A}$.

Definitie 2.3.6 van waarheid van een zin is fundamenteel. Vele concepten in logica zijn erop gebaseerd. We bespreken deze definitie in detail.

Een inductieve definitie We definiëren $\mathfrak{A} \models A$ *door middel van inductie op het aantal symbolen in* A. Daarmee wordt bedoeld dat de waarheid van formules A gedefinieerd wordt in termen van de waarheid van formules *met strikt minder symbolen dan* A. Dat klopt: elke inductieve regel definieert de waarheid van een samengestelde formule A in termen van de waarheid van de componentformules van A. Deze componentformules bevatten strikt minder symbolen dan A zelf. Men zegt ook dat dit een definitie is *op de structuur van* A. Immers, in elke regel wordt de waarheid van een samengestelde zin gedefinieerd in termen van de waarheid van componentzinnen (of de deelzinnen).

Het inductieproces van deze definitie verloopt als altijd: vertrekkend van de lege verzameling, toepassen van regels totdat alle regels voldaan zijn. Er is hier wel een caveat: de regels moeten toegepast worden volgens de inductie-orde: dus eerst de regels toepassen voor de kleinste formules.

Het is absoluut noodzakelijk om de inductie-orde te volgen. Stel dat we dat niet doen. In stap 0 van het inductieproces is de waarheidsrelatie nog leeg: $\models_0 = \emptyset$. In deze situatie is voor willekeurige $\mathfrak{A}$ en A voldaan aan de conditie van de negatie-regel om $\mathfrak{A} \models \neg A$ af te leiden, namelijk $(\mathfrak{A}, A) \notin \models_0$. Bv. neem $\mathfrak{A} = \{(P, \mathbf{t})\}$ en $A = P$, dan is in stap 0 de conditie van de negatieregel voldaan in de eerste stap: $\mathfrak{A} \not\models_0 P$. Als we regels in willekeurige volgorde toepassen, kunnen we in stap 1 afleiden dat $\mathfrak{A} \models \neg P$, en in stap 2 de atoom-regel toepassen om af te leiden dat $\mathfrak{A} \models P$. We bekomen inconsistentie. Dat is helemaal fout. Men moet wachten om een regel toe te passen totdat de waarheid van alle componentformules in de condities van die regel veilig berekend is. Dus: de waarheid van P in $\mathfrak{A}$ berekenen vooralleer de waarheid van $\neg P$ te berekenen.

Waarheid is niet altijd gedefineerd $\mathfrak{A} \models A$ is enkel gedefinieerd als $\mathfrak{A}$ de formule A interpreteert. De waarde van A is *onbepaald* in $\mathfrak{A}$ als A vrije symbolen heeft die geen waarde hebben in $\mathfrak{A}$. Denk aan de formules $VriendVan(Jill, Sam)$ en $\exists x\colon Grade(John, CS230, x)$ in Sectie 2.1 die symbolen $VriendVan$ en $John$ bevatten zonder waarde in de studentendatabank.

Verband logische en nederlandse connectieven De regels in Definitie 2.3.6 zijn eenvoudig: ze vertalen een connectief of kwantor naar het nederlands.

Bv., de $\wedge$-regel drukt uit in wiskundige notatie wat de volgende zin in woorden uitdrukt: $A \wedge B$ *is waar in* $\mathfrak{A}$ *asa* A *en* B *waar zijn in* $\mathfrak{A}$. Bv., in woorden zegt de $\neg$-regel het volgende: $\neg A$ *is waar in* $\mathfrak{A}$ *asa* A *niet waar is in* $\mathfrak{A}$.

Waarheidstabellen voor connectieven Elk connectief komt overeen met een booleaanse functie. Een booleaanse functie is gedefinieerd op de verzameling $\mathbb{B} = \{\mathbf{t}, \mathbf{f}\}$. De functies van de connectieven zijn zoals gezien in de inleiding van het hoofdstuk:

A	$(\neg A)$
t	**f**
f	**t**

A	B	$(A \wedge B)$	$(A \vee B)$	$(A \Rightarrow B)$	$(A \Leftrightarrow B)$
t	**t**	**t**	**t**	**t**	**t**
t	**f**	**f**	**t**	**f**	**f**
f	**t**	**f**	**t**	**t**	**f**
f	**f**	**f**	**f**	**t**	**t**

Bv., de waarheidstabel voor $(\neg A)$ drukt uit dat als A waar is in $\mathfrak{A}$ dan is $\neg A$ onwaar in $\mathfrak{A}$ en als A onwaar is in $\mathfrak{A}$ dan is $\neg A$ waar in $\mathfrak{A}$. Dus, $\mathfrak{A} \models \neg A$ asa $\mathfrak{A} \not\models A$.

Kwantoren Om de waarheid van $\forall x\colon A$ en $\exists x\colon A$ in $\mathfrak{A}$ te bepalen, wordt $\mathfrak{A}$ met alle domeinalementen $a \in D_{\mathfrak{A}}$ uitgebreid tot $\mathfrak{A}[x : a]$ waarna A geëvalueerd wordt in $\mathfrak{A}[x : a]$.

Oefening 2.3.5. *De waarheid van een formule hangt niet alleen af van de waarde van de symbolen maar ook van het domein van* $\mathfrak{A}$. *Neem* $\Sigma = \{P/1\}$ *en de zin* $\forall x\colon P(x)$. *Neem de volgende twee structuren met dezelfde waarde voor symbolen maar met andere domeinen:*

- $\mathfrak{A}_1$*:* $D_{\mathfrak{A}_1} = \{a\}$ *en* $P^{\mathfrak{A}_1} = \{a\}$.
- $\mathfrak{A}_2$*:* $D_{\mathfrak{A}_2} = \{a, b\}$ *en* $P^{\mathfrak{A}_2} = \{a\}$.

Verifieer dat $\forall x\colon \neg P(x)$ *en* $\exists x\colon \neg P(x)$ *een andere waarheidswaarde hebben in* $\mathfrak{A}_1$ *dan in* $\mathfrak{A}_2$.

Voorbeeld 2.3.6. Dit voorbeeld illustreert de werking van Definitie 2.3.6 voor formules die kwantificeren over dezelfde variabele. Neem de formule A van de vorm

$$\exists x\colon (P(x) \wedge \exists x\colon Q(x)).$$

Hoewel de deelformule $\exists x : Q(x)$ voorkomt in het bereik van de buitenste kwantor $\exists x$, worden alle voorkomens van x in $\exists x : Q(x)$ als een nieuwe variabele beschouwd. De zin betekent daarom niet dat P en Q een niet lege doorsnede hebben maar dat P en Q beide niet leeg zijn.

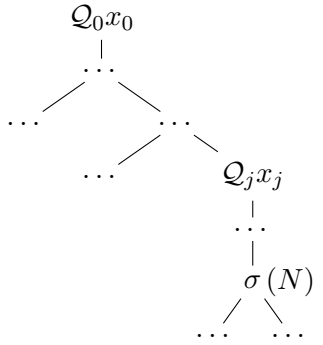
Neem een structuur waarin P en Q niet leeg zijn maar lege doorsnede hebben. Bv. de structuur $\mathfrak{A}$ met $D_{\mathfrak{A}} = \{a, b\}$, $P^{\mathfrak{A}} = \{a\}$ en $Q^{\mathfrak{A}} = \{b\}$.

Aangezien $a \in P^{\mathfrak{A}}$ volgt uit de atoom-regel dat $\mathfrak{A}[x : a] \models P(x)$.

Aangezien $b \in Q^{\mathfrak{A}}$ volgt dat $\mathfrak{A}[x : b] \models Q(x)$. Merk op dat $\mathfrak{A}[x : a][x : b] = \mathfrak{A}[x : b]$. Dus geldt $\mathfrak{A}[x : a][x : b] \models Q(x)$. Uit $\exists$-regel volgt dat $\mathfrak{A}[x : a] \models \exists x\colon Q(x)$.

Uit $\wedge$-regel volgt $\mathfrak{A}[x : a] \models P(x) \wedge \exists x\colon Q(x)$. Uit $\exists$-regel volgt $\mathfrak{A} \models \exists x\colon (P(x) \wedge \exists x\colon Q(x))$.

Waardes berekenen in de abstracte syntax-boom De definitie van de waarheidsrelatie kan ook geïnterpreteerd worden als een inductieve definitie van waardes van nodes in abstracte syntax bomen.



Figuur 2.3: Syntax-bomen van formule en de hernoeming

Stel dat we een zin A willen evalueren in een structuur $\mathfrak{A}$ en de abstracte syntax-boom ziet eruit als in Figuur 2.3. Elke node N van die boom komt overeen met een voorkomen van een deel-expressie van A: een deelformule of een deelterm. Elke node N bevat ofwel een logisch of niet-logisch symbool σ, ofwel een kwantificatie $\mathcal{Q}x$. In Figuur 2.3 wordt een willekeurige "tak" van de formule getoond, met een sequentie van kwantoren $\mathcal{Q}_0 x_0, \ldots, \mathcal{Q}_j x_j$, met $\mathcal{Q}_i$ gelijk aan $\forall$ of $\exists$. Merk op dat dezelfde variabele meerdere keer kan voorkomen in de rij $x_1, \ldots, x_j$.

Een node zoals N zal geëvalueerd worden in een structuur bekomen als $\mathfrak{A}[x_0 : v_0, \ldots, x_j : v_j]$. In deze structuur is de waarde van een variabele x gelijk aan v_i, met i de hoogste index zodat x gelijk is aan x_i.

Als σ een object symbool is (een gebonden variabele of een vrij symbool) niet-logisch symbool is, dan is de waarde van N gelijk aan de waarde van σ in $\mathfrak{A}[x_0 : v_0, \ldots, x_j : v_j]$. Als σ een functie- of predicaatsymbool is, dan wordt de waarde van N bekomen door de waarde van σ in diezelfde structuur toe te passen op de waardes van de kindnodes.

Als N een logisch symbool σ bevat $(\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow, =)$ dan wordt de waarde van N bekomen door de boolse operator geassocieerd aan σ toe te passen op de waardes van de kindnodes.

Als N een kwantificatie $\mathcal{Q}_i x_i$ bevat dan wordt de waarde bepaald uit de verzameling van waardes $V = \{N'^{\mathfrak{A}[x_0:v_0,\ldots,x_i:v_i]} | v_i \in D_{\mathfrak{A}}\}$ van de kindnode N' van N. Dan is V gelijk aan $\{\mathbf{t}\}, \{\mathbf{f}\}$ of $\{\mathbf{t}, \mathbf{f}\}$. (Waarom kan V niet leeg zijn?) Als $\mathcal{Q}_i = \forall$, dan geldt $N^{\mathfrak{A}} = \mathbf{t}$ asa $\mathbf{f} \notin V$; als $\mathcal{Q}_i = \exists$ dan geldt $N^{\mathfrak{A}} = \mathbf{t}$ asa $\mathbf{t} \in V$.

Fundamentele eigenschap We willen bewijzen dat de waarheid van een formule in $\mathfrak{A}$ enkel afhangt van de waardes van de vrije symbolen van de formule in $\mathfrak{A}$. Als deze eigenschap niet zou voldaan zijn, dan zou het duiden op een groot probleem.

Propositie 2.3.1. *Zij A formule en $\mathfrak{A}$ en $\mathfrak{B}$ twee structuren met identiek domein ($D_{\mathfrak{A}} = D_{\mathfrak{B}}$) en identieke waardes voor alle vrije symbolen van A ($\tau^{\mathfrak{A}} = \tau^{\mathfrak{B}}$ voor alle $\tau \in Vrij(A)$).*

Dan geldt dat $\mathfrak{A} \models A$ asa $\mathfrak{B} \models A$.

Meer algemeen, elke expressie (term of formule) heeft dezelfde (waarheids)waarde in alle structuren met identiek domein en identieke waarde voor de vrije symbolen van de expressie. Het is deze uitbreiding die bewezen wordt. Voor een voorbeeld dat laat zien dat de waarheid van een formule afhangt van het domein, zie Oefening 2.3.5.

Dus, de waarheidswaarde van A is onafhankelijk van de waarde van symbolen die niet voorkomen in A en evenmin van de waarde van symbolen die enkel voorkomen als gebonden variabelen van A.

Bew. Het bewijs is per inductie op de structuur van expressies. Dit is een variant van bewijs per inductie op de natuurlijke getallen.

Voor e een expressie (term of formule), zij $\mathcal{H}(e)$ de uitspraak dat als twee structuren $\mathfrak{A}$ en $\mathfrak{B}$ identiek zijn op hun domein en alle vrije symbolen van e, dan geldt $e^{\mathfrak{A}} = e^{\mathfrak{B}}$.

We moeten aantonen dat $\mathcal{H}(e)$ voldaan is voor alle expressies. Om dat te bewijzen volstaat het volgende aan te tonen: als $\mathcal{H}(e')$ voldaan is voor alle subexpressies e' van e, dan is $\mathcal{H}(e)$ voldaan. Dit is het principe van *bewijs door inductie op de structuur van e*.

Hier in dit geval moeten we het volgende aantonen: Gegeven dat $\mathcal{H}(e')$ voldaan is voor elke subexpressies e' van e. Gegeven dat $\mathfrak{A}$ en $\mathfrak{B}$ hetzelfde domein hebben en waardes voor vrije symbolen van e. Bewijs dan dat $e^{\mathfrak{A}} = e^{\mathfrak{B}}$.

Voor een volledig bewijs moet elke soort expressie e beschouwd worden. We zullen hier 4 gevallen beschouwen.

- Basisgeval: zij e een objectsymbool c. Er zijn geen subexpressies, maar c is een vrij symbool van e en dus $c^{\mathfrak{A}} = c^{\mathfrak{B}}$. Dus geldt $\mathcal{H}(e)$.

- Zij e een atoom $P(t_1, \ldots, t_n)$. Alle vrije symbolen van t_i zijn vrij in e, en dus hebben ze dezelfde waardes in $\mathfrak{A}$ en $\mathfrak{B}$. Uit $\mathcal{H}(t_i)$ volgt ${t_i}^{\mathfrak{A}} = {t_i}^{\mathfrak{B}}$. Ook P is een vrij symbool van e en heeft dezelfde waarde in $\mathfrak{A}$ en $\mathfrak{B}$. Uit de atoom-regel volgt: $\mathfrak{A} \models P(t_1, \ldots, t_n)$ asa $({t_1}^{\mathfrak{A}}, \ldots, {t_n}^{\mathfrak{A}}) \in P^{\mathfrak{A}}$ asa $({t_1}^{\mathfrak{B}}, \ldots, {t_n}^{\mathfrak{B}}) \in P^{\mathfrak{B}}$ asa $\mathfrak{B} \models P(t_1, \ldots, t_n)$. Dus geldt $\mathcal{H}(e)$.
- Zij e gelijk aan $A \wedge B$. Alle vrije symbolen van A zijn vrije symbolen van $A \wedge B$, en dus zijn hun waardes gelijk in $\mathfrak{A}$ en $\mathfrak{B}$. Aangezien $\mathcal{H}(A)$ geldt heeft A dezelfde waarheidswaarde in $\mathfrak{A}$ en $\mathfrak{B}$. Idem voor B. Uit de $\wedge$-regel volgt $\mathfrak{A} \models A \wedge B$ asa $\mathfrak{A} \models A$ and $\mathfrak{A} \models B$ asa $\mathfrak{B} \models A$ and $\mathfrak{B} \models B$ asa $\mathfrak{B} \models A \wedge B$. Dus, $\mathcal{H}(e)$ geldt.
- Zij e gelijk aan $\exists x\colon A$. De vrije symbolen van A zijn die van e plus x. Voor elke d geldt dat $\mathfrak{A}[x : d]$ en $\mathfrak{B}[x : d]$ dezelfde waardes hebben op alle vrije symbolen van e en op x. Uit $\mathcal{H}(A)$ volgt dat $\mathfrak{A}[x : d] \models A$ asa $\mathfrak{B}[x : d] \models A$. Aangezien $\mathfrak{A}$ en $\mathfrak{B}$ hetzelfde domein hebben, bestaat $d \in D_{\mathfrak{A}}$ zodat $\mathfrak{A}[x : d] \models A$ asa er bestaat $d \in D_{\mathfrak{B}}$ zodat $\mathfrak{B}[x : d] \models A$. Uit de $\exists$-regel volgt dat $\mathfrak{A} \models e$ asa $\mathfrak{B} \models e$. Zodoende geldt $\mathcal{H}(e)$.

De andere gevallen zijn analoog.

Q.E.D.

Interpretatie van n-voudige relatievoorschriften In een databank willen gebruikers meestal meer complexe vragen stellen dan of een zin waar is. Bv. ze willen alle studenten weten die geslaagd zijn voor logica? Dergelijke vragen kunnen gesteld worden in een uitbreiding van predikatenlogica met *verzamelingexpressies*. Dit zijn expressies van de vorm:

$$\{(x_1, \ldots, x_n) : A\}$$

waarbij A een formule is over het vocabularium van de studentendatabank en de variabelen $x_1, \ldots, x_n$. Als $n > 1$ dan stelt zo'n expressie een relatie voor en dan noemen we de expressie eventueel een *relatievoorschrift*.

Voorbeeld 2.3.7. Gevraagd: alle koppels (x,y) zodat student x geslaagd is voor vak y.

$$\{(x, y) : (\exists z\colon Grade(x, y, z) \wedge PassingGrade(z))\}$$

Het antwoord op deze query is de *waarde* van deze expressie in de databank, zoals gedefinieerd in de volgende definitie.

Definitie 2.3.7. De interpretatie van $\{(x_1, \ldots, x_n) : A\}$ in $\mathfrak{A}$ is de volgende relatie:

$$\{(a_1, ..., a_n) \in {D_{\mathfrak{A}}}^n \mid \mathfrak{A}[x_1 : a_1, \ldots, x_n : a_n] \models A\}$$

Notatie: $\{(x_1, \ldots, x_n) : A\}^{\mathfrak{A}}$.

Deze definitie legt vast welk antwoord een databanksysteem moet geven als antwoord op een n-voudige query.

Voorbeeld 2.3.8. De waarde van de 2-voudige query

$$\{(x, y) : \exists z\colon Grade(x, y, z) \wedge PassingGrade(z)\}$$

in de studentendatabank is de relatie uit deze tabel:

Sam	CS148
Jill	CS148
Jack	CS148
Flo	CS230
May	CS230
Ann	CS230
Jill	M100
Sam	M100
Flo	M100

Semantische afgeleide concepten In de wiskunde zijn semantische concepten zoals consistentie, logisch gevolg, en equivalentie van groot belang. In logica kunnen deze gedefinieerd worden in termen van de waarheidsrelatie.

Definitie 2.3.8.
- A is logisch consistent indien A waar is in minstens één structuur. A is logisch inconsistent of contradictorisch indien A onwaar is in elke structuur die A interpreteert.
- A is logisch waar of is een tautologie indien A waar is in alle structuren die A interpreteren.
- A is een logisch gevolg van B indien voor elke structuur $\mathfrak{A}$ die beide interpreteert geldt: als $\mathfrak{A}$ voldoet aan B dan voldoet $\mathfrak{A}$ aan A.
- A is logisch equivalent met B indien A en B dezelfde waarheidswaarde hebben in elke structuur die beide interpreteeert.

We zullen uitgebreider terugkomen op deze concepten in het volgend hoofdstuk.

Bv. alle uitspraken die waar zijn in de studentendatabank zijn logisch consistent. En ook de negatie van elke onware uitspraak.

Bv. Voor elk paar A, B van uitspraken is $A \wedge B$ logisch equivalent met $B \wedge A$.

Bv. De uitspraak $\forall x\colon Instructor(x, CS238) \Rightarrow Lecturer(x)$ is een logisch gevolg van

$$\forall x\colon \forall y\colon Instructor(x, y) \Rightarrow Lecturer(x) \wedge Course(y)$$

Opmerking 2.3.1. In LogicPalet is het tool ASKSpass beschikbaar om de logische equivalentie van twee formules na te gaan. Zoals de naam suggereert wordt hiervoor de theoremprover Spass aangeroepen.

2.4 De informele semantiek van Predicatenlogica

We willen logica gebruiken om in een toepassingsdomein kennis, eigenschappen, queries, integriteitsbeperkingen, etc. uit te drukken. Dat is enkel mogelijk als we begrijpen wat de formules betekenen als uitspraken over het toepassingdomein. Leibniz zou zeggen: voor elke logische zin moeten we de "gedachte" die het uitdrukt kunnen reconstrueren. De informele semantiek van predikatenlogica is de theorie over hoe dit te bereiken.

Er is hierbij een probleem. De gedachte die we hebben bij zin zoals:

$$\neg\exists x : Course(x) \wedge Prerequ(x, M100)$$

is een gedachte over het toepassingsdomein. Logica kent geen concepten uit toepassingsdomeinen. Hoe kan dit werken?

Wanneer we logica willen toepassen in een toepassingsdomein dan kiezen wij of iemand anders een vocabularium Σ bestaande uit predicaat- en functiesymbolen. We kiezen deze symbolen als logische voorstelling van concepten uit het toepassingsdomein. E.g., studentendatabank:

- $Enrolled(x, y)$ :: "(student) x is ingeschreven voor (cursus) y"
- $Grade(x, y, z)$ :: "(student) x legde examen voor (cursus) y af en behaalde score z"
- $AgeOf(x)$ :: "de leeftijd van (persoon) x"
- Sam :: "(student) Sam"
- AA :: "(score) AA"

Dit noemen we een **informele interpretatie** $\mathcal{I}$ van Σ.

Uit de wiskunde of de cursus logica kennen we ook de betekenis van de logische symbolen.

$\ldots \wedge \ldots$	$\ldots$ en $\ldots$
$\ldots \vee \ldots$	$\ldots$ of $\ldots$ (of allebei)
$\neg \ldots$	het is niet waar dat $\ldots$
$\ldots \Rightarrow \ldots$	als $\ldots$ dan $\ldots$
$\ldots \Leftrightarrow \ldots$	$\ldots$ als en slechts als $\ldots$
$\forall \ldots\ldots$	voor alle $\ldots$ geldt dat $\ldots$
$\exists \ldots\ldots$	er bestaat $\ldots$ zodat $\ldots$
$\ldots = \ldots$	$\ldots$ is gelijk aan $\ldots$

Hiermee kunnen we de informele betekenis van logische zinnen onder $\mathcal{I}$ opstellen. Bv.

$$\forall c : Enrolled(Sam, c) \Rightarrow Grade(s, c, AA)$$

Deze zin betekent onder de informele betekenis $\mathcal{I}$ van de studentendatabank: "*Voor alle objecten c, als Sam ingeschreven is voor c dan legde Sam examen af voor c en behaalde score AA*"

Anders gezegd, Sam behaalde AA voor alle vakken waarvoor zij ingeschreven was.

Om de omzetting van logische formules naar Nederlands te vereenvoudigen, specifiëren we $\mathcal{I}$ met de juiste taalpatronen:

- Met een predicaatsymbool $P(x_1, \ldots, x_n)$ specifiëren we $\mathcal{I}(P)$ als een werkwoordzin met parameters $x_1, \ldots x_n$.

  Bv. $Enrolled(x, y)$ :: "(student) x is ingeschreven voor (cursus) y".

- Voor een functiesymbool $f(x_1, \ldots, x_n)$ specifiëren we $\mathcal{I}(f)$ als een functionele zinsnede met parameters $x_1, \ldots x_n$.

  Bv., $Lft(x)$ :: "de leeftijd van x"

Dan kan een tekstuele vorm voor $\mathcal{I}(A)$ automatisch berekend worden.

Bv.

$$\forall c : Enrolled(Sam, c) \Rightarrow Grade(Sam, c, AA)$$

kan vertaald worden naar:

"*Voor alle c, als student Sam ingeschreven is voor vak c dan student c behaalde AA voor vak c*"

Dat is dezelfde uitspraak als daarstraks.

Vanuit logisch standpunt hebben symbolen van een vocabularium geen inherente betekenis in een toepassingsgebied. Dus is de informele betekenis van een zin relatief tot $\mathcal{I}$.

Neem bv. de alternatieve informele interpretatie $\mathcal{I}_2$:

- $Enrolled(x, y)$ :: "x werkt op project y"
- $Grade(x, y, z)$ :: "x krijgt budget z voor project y"
- Sam :: Marc Denecker
- AA :: 100.000 euro

Onder deze informele interpretatie betekent

$$\forall c : Enrolled(Sam, c) \Rightarrow Grade(s, c, AA)$$

iets anders:

"Op all projecten waar Marc Denecker op werkt heeft hij een budget van 100.000 euro."

Oefening 2.4.1. *Zoek nog meer informele interpretaties $\mathcal{I}$ van de symbolen van de studentendatabank en kijk na wat verschillende formules die we zagen betekenen onder $\mathcal{I}$.*

Oefening 2.4.2. *Hier volgt een voorbeeld van de subtiliteit van natuurlijke taal. In de context van de voetbalkalender, neem de volgende uitspraak:*

> Elke speeldag heeft hoogstens één andere eerst volgende speeldag.

Wat wordt hier gezegd?

- *Dat voor elke speeldag x, de verzameling van eerst volgende speeldagen verschillend van x hoogstens 1 element bevat? Dus, de verzameling van eerst volgende speeldagen kan hoogstens 2 elementen bevatten: mogelijks x zelf, en mogelijks nog 1 andere element dan x. Of,*
- *dat voor elke speeldag x, de verzameling van eerst volgende speeldagen hoogstens 1 element bevat, en dit element is niet x.*

Dat is duidelijk verschillend. De combinatie van "één" en "ander" in deze zin is ambigu.

Hier maakt het niet uit omdat we weten dat een speeldag nooit temporeel op zichzelf kan volgen. In andere contexten maakt het wel degelijk een verschil.

Druk de twee betekenissen van de uitspraak uit gebruikmakend met de predicaten $Speeldag/1$ en $VolgtOp/2$.

Abstracte betekenis van een zin in een structuur We kunnen logische zinnen ook interpreteren zonder de bedoelde betekenis van de symbolen te kennen. Dan beschouwen we ze als voorstellingen van abstracte relaties en functies.

Zij $\mathfrak{A}$ een structuur, neem als $\mathcal{I}$ de "informele" interpretatie die elk symbool interpreteert zoals in $\mathfrak{A}$.

Bv. neem een structuur wiens domein a en b bevat zodat $Sam^{\mathfrak{A}} = a, AA^{\mathfrak{A}} = b$. Neem opnieuw de formule:

$$\forall c : Enrolled(Sam, c) \Rightarrow Grade(Sam, c, AA)$$

Onder deze abstracte informele betekenis betekent deze zin het volgende:

"*Voor alle elementen c zodat* $(a, c) \in Enrolled^{\mathfrak{A}}$ *geldt dat* $(a, c, b) \in Grade^{\mathfrak{A}}$*"* (*)

Er is een grote gelijkenis tussen het opstellen van de informele semantiek van logische zinnen in een structuur, en de inductieve regels van de definitie van de waarheidsrelatie. In die regels gebeurt een soortgelijke omzetting. Bv.

$$\mathfrak{A} \models A \underline{\wedge} B \text{ indien } \mathfrak{A} \models A \text{ } \underline{\text{en}} \text{ } \mathfrak{A} \models B$$

We kunnen inderdaad een informele interpretatie van een zin A in een structuur $\mathfrak{A}$ op een wiskundige manier bekomen door toepassing van Definitie 2.3.6 van de waarheidsrelatie $\models$. We illustreren dit proces. De onderlijnde zinsneden worden telkens geëvalueerd door toepassing van een inductieve regel van de definitie.

$$\frac{\mathfrak{A} \models \forall c : Enrolled(Sam, c) \Rightarrow Grade(Sam, c, AA)}{\text{asa } (\forall\text{-regel})}$$

$$\text{voor elke } d \in D_{\mathfrak{A}}\text{: } \frac{\mathfrak{A}[c : d] \models Enrolled(Sam, c) \Rightarrow Grade(Sam, c, AA)}{\text{asa } (\Rightarrow\text{-regel})}$$

$$\text{voor elke } d \in D_{\mathfrak{A}}\text{: } \frac{\mathfrak{A}[c : d] \not\models Enrolled(d) \text{ of } \mathfrak{A}[c : d] \models Grade(Sam, c, AA)}{\text{asa } (\neg\text{-regel})}$$

$$\frac{\text{voor elke } d \in D_{\mathfrak{A}}\text{: het is onwaar dat } \mathfrak{A}[c : d] \models Enrolled(Sam, c) \text{ of } \mathfrak{A}[c : d] \models Grade(Sam, c, AA)}{\text{asa (atoom-regel)}}$$

$$\text{voor elke } d \in D_{\mathfrak{A}}\text{: } (a, d) \notin Enrolled^{\mathfrak{A}} \text{ of } \underline{\mathfrak{A}[c : d] \models Grade(Sam, c, AA)}$$

$$\text{asa (vereenvoudigen+ atoom-regel)}$$

$$\text{voor elke } d \in D_{\mathfrak{A}}\text{: als } (a, d) \in Enrolled^{\mathfrak{A}} \text{ dan } (a, d, b) \in Grade^{\mathfrak{A}}.$$

We bekomen een zin die exact hetzelfde zegt als (*). Dit geeft ons vertrouwen dat onze informele interpretatie van zinnen conform de formele definitie is.

In de inleiding van dit hoofdstuk werd een informele interpretatie $\mathcal{I}$ van de symbolen van de studentendatabank voorgesteld, maar ook een databankstructuur $\mathfrak{A}_{DB}$. We hebben nederlandse zinnen geformuleerd en vertaald in logica, waarbij we onmiskenbaar gebruik gemaakt hebben van $\mathcal{I}$. Maar om te bepalen of de logische zinnen waar waren in de databank hebben we gebruik gemaakt van de abstracte interpretatie van de zinnen in $\mathfrak{A}_{DB}$.

De informele interpretatie $\mathcal{I}(A)$ geeft ons een eenduidige vertaling van logische zinnen naar natuurlijke taal zoals gebruikt wordt in wiskundige teksten. Als de concepten in $\mathcal{I}$ precies zijn, dan zal $\mathcal{I}(A)$ een precieze zin zijn met een duidelijke betekenis.

2.5 Berekenen en bewijzen van waarheid in een structuur

Propositionele structuren De waarheidswaarde van een propositionele formule in zo'n structuur wordt berekend als een uitbreiding van de waarheidstabel.

In de volgende voorbeeld wordt voor de structuur $\{(P, \mathbf{f}), (Q, \mathbf{t}), (R, \mathbf{t})\}$ de waarde van een formule berekend. In elke lijn wordt een nieuwe stap gezet in het inductie-proces en wordt de volgende laag van formules berekend. We gebruiken 0 en 1 als voorstelling voor $\mathbf{f}$ en $\mathbf{t}$.

P	Q	R	(	P	$\vee$	Q	)	$\wedge$	R	$\Leftrightarrow$	(	P	$\wedge$	R	)	$\vee$	(	Q	$\wedge$	R	)
0	1	1		0		1			1			0		1				1		1	)
0	1	1		0	1	1			1			0	0	1				1	1	1	
0	1	1		0	1	1		1	1			0	0	1		1		1	1	1	
0	1	1		0	1	1		1	1	**1**		0	0	1		1		1	1	1	

De formule is dus waar in deze structuur.

Waarheidstabellen worden vaak gebruikt in propositielogica. Dan worden alle waardes van deelformules ineens ingevuld. Dezelfde waarheidstabel kan gebruikt worden om de formule te evalueren in meerdere structuren.

P	Q	R	(	P	$\vee$	Q	)	$\wedge$	R	$\Leftrightarrow$	(	P	$\wedge$	R	)	$\vee$	(	Q	$\wedge$	R	)
0	1	1		0	1	1		1	1	**1**		0	0	1		1		1	1	1	
1	1	1		1	1	1		1	1	**1**		1	1	1		1		1	1	1	
1	0	1		1	1	0		1	1	**1**		1	1	1		1		0	0	1	
1	1	0		1	1	1		0	0	**1**		1	0	0		0		1	0	0	
...																					

Structuren met eindig domein We bewijzen de waarheid of onwaarheid van een aantal formule A in dergelijke structuur $\mathfrak{A}$ door middel van Definitie 2.3.6. Dit geeft inzicht in de regels van de waarheidsdefinitie. Het is ook een oefening in het rederenen over een complexe inductieve definitie.

Een dergelijk bewijs bestaat uit 3 soorten redeneerstappen:

- Een definitie ontvouwen: in een uitspraak die een gedefinieerd concept bevat, dit gedefinieerd concept vervangen door zijn definitie.

  Bv. in een uitspraak die $\mathfrak{A} \models A$ bevat (in symbolen of in woorden), de deeluitspraak $\mathfrak{A} \models A$ vervangen door "$\mathfrak{A} \models A$ en $\mathfrak{A} \models B$".

  Of in een uitspraak die $\mathfrak{A} \not\models A$ bevat, deze deeluitspraak vervangen door "het is niet het geval dat $\mathfrak{A} \models A$ en $\mathfrak{A} \models B$". Of een stapje verder, door "$\mathfrak{A} \not\models A$ of $\mathfrak{A} \not\models B$".

- Omgekeerd, een definitie opvouwen: in een uitspraak die een deeluitspraak bevat die overeenkomt met het lichaam van een definitie, deze deeluitspraak vervangen door het gedefinieerde concept.

  Bv. een deeluitspraak "$Ray^{\mathfrak{A}} = Ray$ en $CS238^{\mathfrak{A}} = CS238$ en $(Ray, CS238)$ behoort tot $Instructor^{\mathfrak{A}}$" vervangen door "$\mathfrak{A} \models Instructor(CS238, Ray)$".

- Het vereenvoudigen van een zin. Bv. "voor alle objecten x van het universum geldt dat als x behoort tot $Lecturer^{\mathfrak{A}}$, dan bestaat een y in het universum zodat (x, y) behoort tot $Instructor^{\mathfrak{A}}$". Dit kunnen we vervangen door "voor alle $x \in Lecturer^{\mathfrak{A}}$, bestaat een koppel $(x, y) \in Instructor^{\mathfrak{A}}$".

Deze 3 soorten stappen zijn universeel toepasbaar in bewijzen over gedefinieerd concepten, ook als de definities niet inductief zijn.

Een tweede aspect dat wel specifiek is voor inductieve definities over een inductie-orde, is dat het bewijs bottom-up gevoerd kan worden, van beneden naar boven met de inductie-orde mee, of ook top-down, tegen de inductie-orde in.

We beginnen met de bottom-up bewijsmethode: we evalueren de formule vanaf de bladeren van de abstracte syntax-boom naar boven tot aan de wortel, t.t.z., de formule zelf. Kortom, het bewijs volgt de inductie-orde.

Voorbeeld 2.5.1. T.B. $\mathfrak{A} \models \exists x\colon Instructor(Ray, x)$.

De informele betekenis van deze zin is dat Ray een vak doceert. In de studentendatabank is deze zin waar. Immers, Ray doceert het vak $CS230$. Een wiskundig bewijs is als volgt.

Bew.

Er geldt $Ray^{\mathfrak{A}} = Ray$ en $(Ray, CS230) \in Instructor^{\mathfrak{A}}$.

Dus volgt uit de atoom-regel dat $\mathfrak{A}[x : CS230] \models Instructor(Ray, x)$, t.t.z. $Instructor(Ray, x)$ is waar in $\mathfrak{A}[x : CS230]$.

Uit de $\exists$-regel volgt dan dat $\exists x\colon Instructor(Ray, x)$ waar is in $\mathfrak{A}$. Q.E.D.

Voorbeeld 2.5.2. T.B. $\mathfrak{A} \not\models \exists x\colon Course(x) \wedge Prerequ(x, x)$.

De zin betekent dat er een cursus bestaat die voorkennis is van zichzelf, wat inderdaad onwaar is in de studentendatabank.

Bew.

Zij d een willekeurig domeinelement van $\mathfrak{A}$.

Het is waar dat $(d, d) \notin Prerequ^{\mathfrak{A}}$.

Uit de atoom-regel volgt dat $\mathfrak{A}[x : d] \not\models Prerequ(x, x)$.

Uit de $\wedge$-regel volgt dat $\mathfrak{A}[x : d] \not\models Course(x) \wedge Prerequ(x, x)$.

Aangezien dit geldig is voor elke $d \in \mathfrak{A}$ kunnen we besluiten dat er geen $d \in D_{\mathfrak{A}}$ bestaat waarvoor $\mathfrak{A}[x : d] \models Course(x) \wedge Prerequ(x, x)$.

Uit de $\exists$-regel volgt dan dat $\mathfrak{A} \not\models \exists x\colon Course(x) \wedge Prerequ(x, x)$. Q.E.D.

Een regel uit Definitie 2.3.6 toegepast worden om zowel $\mathfrak{A} \models A$ als $\mathfrak{A} \not\models A$ aan te tonen. Bv. neem de regel

$$\mathfrak{A} \models A \wedge B \text{ asa } \mathfrak{A} \models A \text{ en } \mathfrak{A} \models B.$$

Deze regel betekent hetzelfde als

$$\mathfrak{A} \not\models A \wedge B \text{ asa } \mathfrak{A} \not\models A \text{ of } \mathfrak{A} \not\models B.$$

Bijgevolg, als we vaststellen dat $\mathfrak{A} \not\models B$, dan volgt uit de $\wedge$-regel dat $\mathfrak{A} \not\models A \wedge B$.

Voorbeeld 2.5.3. T.B. $\mathfrak{A} \models \forall x\colon \neg Prerequ(x, CS238) \vee Instructor(Ray, x)$.

De zin betekent alle voorkennis van CS238 gedoceerd wordt door Ray. De enige voorkennis van CS238 is CS230, en deze wordt door Ray gegeven.

Bew.

Zij $d \in D_{\mathfrak{A}}$ een willekeurig domeinelement.

We maken een gevalsonderscheid.

- A) Zij d verschillend van $CS230$. Dan geldt dat $(d, CS238) \notin Prerequ^{\mathfrak{A}}$.
  Uit de atoom-regel volgt $\mathfrak{A}[x : d] \not\models Prerequ(x, CS238)$.
  Uit de $\neg$-regel volgt $\mathfrak{A}[x : d] \models \neg Prerequ(x, CCS238)$.
  Uit de $\vee$-regel kunnen we afleiden dat $\mathfrak{A}[x : d] \models \neg Prerequ(x, CS238) \vee Instructor(Ray, x)$.
- B) Zij $d = CS230$.
  Er geldt $(Ray, CS230) \in Instructor^{\mathfrak{A}}$.
  Uit de atoom-regel volgt $\mathfrak{A}[x : CS230] \models Instructor(Ray, x)$.
  Uit de $\vee$-regel leiden we af dat $\mathfrak{A}[x : CS230] \models \neg Prerequ(x, CS238) \vee Instructor(Ray, x)$.

Aangezien we in A) en B) alle mogelijke waardes voor $d \in D_{\mathfrak{A}}$ hebben overlopen, is onze gevalstudie volledig. We hebben aangetoond dat voor elke $d \in D_{\mathfrak{A}}$ geldt dat $\mathfrak{A}[x : d] \models \neg Prerequ(x, CS238) \vee Instructor(Ray, x)$.

Uit de $\forall$-regel volgt dat $\mathfrak{A} \models \forall x\colon \neg Prerequ(x, CS238) \vee Instructor(Ray, x)$. Q.E.D.

Om $\mathfrak{A} \models \forall x\colon A$ aan te tonen, moet we $\mathfrak{A}[x : d] \models A$ aantonen voor elk domeinelement $d \in D_{\mathfrak{A}}$. Om dit exhaustief te doen voor grotere domeinen is onbegonnen werk. In plaats daarvan maken we een gevalsonderscheid waarbij een geval bestaat uit meerdere domeinelementen met een identiek bewijs. Bv. $d \neq CS238$ versus $d = CS238$.

Dezelfde werkwijze is ook bruikbaar voor $\mathfrak{A} \not\models \exists x\colon A$. Ook daar moet een eigenschap aangetoond worden voor alle domeinelementen d, namelijk dat $\mathfrak{A}[x : d] \not\models A$.

Top-down bewijzen Bottom up bewijzen volgen de inductie-orde en passen regels van de definitie toe van rechts naar links, of naar omhoog in de abstracte syntaxboom. Een probleem met dit soort bewijs is echter dat het niet altijd evident is om initieel te weten te komen welke feiten uit de databank relevant zijn om het bewijs te kunnen opbouwen.

Een alternatief soort bewijs dat helpt om de relevante feiten te zoeken is een top-down bewijs dat naar omlaag werkt in de abstracte syntaxboom. We vertrekken van de te bewijzen uitspraak $\mathfrak{A} \models A$ of $\mathfrak{A} \not\models A$, en we reduceren deze door de regels van de waarheidsdefinitie van links naar rechts toe te passen. Zo is het eenvoudiger om de relevante basisfeiten voor het bewijs te ontdekken.

Voorbeeld 2.5.4. Voorbeeld 2.5.1 herwerkt met top-down ontvouwing gevolgd door bottom-up opvouwing:

TB (1) $\mathfrak{A} \models \exists x\colon Instuctor(Ray, x)$

(toepassen $\exists$-regel voor $x : CS230$)
TB (2) $\mathfrak{A}[x : CS230] \models Instructor(Ray, x)$
($Ray, CS230) \in Instructor^{\mathfrak{A}}$ en $Ray^{\mathfrak{A}} = Ray$
(2) volgt nu uit Atoom-regel
(1) volgt uit $\exists$-regel toegepast op (2)

Q.E.D.

In dit voorbeeld wordt de te bewijzen initiële uitspraak door top-down ontvouwing iteratief herleidt tot eenvoudiger te bewijzen uitspraken, totdat beneden in de syntax-boom. Vervolgens passen we de regels van de definitie in omgekeerde volgorde op bottom-up wijze toe tot bij de initiële uitspraak.

Die bottom-up fase is feitelijk overbodig. Doordat we systematisch het te bewijzen hebben herleidt door middel van ontvouw-stappen, staat het vast dat als de uitspraken beneden in de syntax-boom geldig zijn, dat de omgekeerde weg zal slagen. Dus, we kunnen die bottom up fase weglaten.

Voorbeeld 2.5.5. Voorbeeld 2.5.1 herwerkt met enkel top-down ontvouwing:

TB $\mathfrak{A} \models \exists x\colon Instuctor(Ray, x)$
(toepassen $\exists$-regel voor $x : CS230$)
TB $\mathfrak{A}[x : CS230] \models Instructor(Ray, x)$
(toepassen atoom-regel)
($Ray, CS230) \in Instructor^{\mathfrak{A}}$ en $Ray^{\mathfrak{A}} = Ray$

Q.E.D.

Voorbeeld 2.5.6. Voorbeeld 2.5.2 herwerkt:

TB $\mathfrak{A} \not\models \exists x\colon Course(x) \wedge Prerequ(x, x)$
($\exists$-regel)
TB voor alle $d \in D_{\mathfrak{A}}$ geldt $\mathfrak{A}[x : d] \not\models Course(x) \wedge Prerequ(x, x)$
Zij d een willekeurig element van $D_{\mathfrak{A}}$.
TB $\mathfrak{A}[x : d] \not\models Course(x) \wedge Prerequ(x, x)$
($\wedge$-regel)
TB $\mathfrak{A}[x : d] \not\models Prerequ(x, x)$
(atoom-regel)
Er geldt $(d, d) \notin Prerequ^{\mathfrak{A}}$.

Q.E.D.

Voorbeeld 2.5.7. Voorbeeld 2.5.3 herwerkt:

TB $\mathfrak{A} \models \forall x\colon \neg Prerequ(x, CS238) \vee Instructor(Ray, x)$
($\forall$-regel; gevallenstudie op $d = CS230$)
A) Zij d verschillend van $CS230$.
TB $\mathfrak{A}[x : d] \models \neg Prerequ(x, CS238) \vee Instructor(Ray, x)$

(∨-regel)
TB $\mathfrak{A}[x:d] \models \neg Prerequ(x, CS238)$
(¬-regel)
TB $\mathfrak{A}[x:d] \not\models Prerequ(x, CS238)$
Er geldt $(d, CS238) \notin Prerequ^{\mathfrak{A}}$.

B) Zij $d = CS230$.
TB $\mathfrak{A}[x:CS230] \models \neg Prerequ(x, CS238) \vee Instructor(Ray, x)$
(∨-regel)
TB $\mathfrak{A}[x:CS230] \models Instructor(Ray, x)$
(atoom-regel)
Er geldt $(Ray, CS230) \in Prerequ^{\mathfrak{A}}$.

Q.E.D.

Oefening 2.5.1. *Geef een bewijs van de waarheid of onwaarheid van een andere formule uit Sectie 2.1.*

Oefening 2.5.2. *Hoe zou je aantonen dat de interpretatie van de volgende verzamelingexpressie*

$$\{x : \exists y\colon Prerequ(x, y) \wedge \neg\exists y\colon Prerequ(y, x)\}$$

de volgende verzameling is : $\{CS149, M100\}$*?*

Opmerking 2.5.1. LogicPalet bevat een uitlegtool om uit te leggen waarom een zin waar of onwaar is in een gegeven structuur. Deze verklaringen zijn in feite niets anders dan bewijzen van de waarheid of onwaarheid van een zin in een structuur. Probeer deze functionaliteit uit om te zien welke soort verklaringen je krijgt.

Structuren met oneindig domein In de wiskunde zijn vrijwel alle "interessante" structuren oneindig. Bv. de verschillende soorten getallen $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \ldots$.

Neem het vocabularium $\Sigma = \{P/1, L/2, S/2 :\}$ en structuur $\mathfrak{A}$ met domein $\mathbb{N}$ zodat:
- $P^{\mathfrak{A}}$ is de verzameling van priemgetallen.
- $L^{\mathfrak{A}} = \{(n, m) | n < m\}$, de "strikt kleiner dan"-relatie
- $S^{\mathfrak{A}} = \mathbb{N}^2 \to \mathbb{N} : (n, m) \to n + m$, de som-functie.

Voorbeeld 2.5.8.

T.B. $\mathfrak{A} \models \forall x\colon \neg L(x, x)$

In $\mathfrak{A}$ betekent deze zin dat geen enkel getal strikt kleiner is dan zichzelf.

Bew. Zij $a \in \mathbb{N}$ een willekeurig getal $a \in \mathbb{N}$.

Het geldt dat $a \not< a$, bijgevolg $(a, a) \notin L^{\mathfrak{A}}$.

Toepassing van de atoom-regel gevolgd door ¬-regel leidt af dat $\mathfrak{A}[x:a] \models \neg L(x, x)$.

Aangezien dit geldt voor willekeurige $a \in \mathbb{N}$ volgt het te bewijzen nu uit de ∀-regel. Q.E.D.

Voorbeeld 2.5.9.

T.B. $\mathfrak{A} \models \forall x\colon \exists y\colon L(x, y)$.

Bew. We zullen deze keer gebruik maken van de nederlandse vertaling van deze zin. De vertaling is: voor elk getal n bestaat er een getal m zodat $n < m$. Dat is zo, bv. $m = n + 1$. Q.E.D.

Voorbeeld 2.5.10.

T.B. $\mathfrak{A} \not\models \exists y\colon \forall x\colon L(x,y)$.

Bew. Deze zin betekent dat er een getal n bestaat zodat elk getal m strikt kleiner is dan n. Dat is niet het geval aangezien bv. $n \not< n$. Q.E.D.

Opmerking 2.5.2. De zinnen in Voorbeelden 2.5.9 en 2.5.10 zijn identiek behalve dat de kwantoren verwisseld werden: $\forall x\colon \exists y\colon \ldots$ versus $\exists y\colon \forall x\colon \ldots$. Dit laat zien dat kwantoren verwisselen de betekenis van een zin verandert.

Voorbeeld 2.5.11. De volgende zin is ook waar, maar niet zo makkelijk te bewijzen.

$$\forall x\colon P(x) \Rightarrow \exists y\colon P(y) \wedge L(x,y)$$

In $\mathfrak{A}$ betekent deze zin: voor elk priemgetal bestaat een strikt groter priemgetal. Dat is het geval want er bestaan oneindig veel priemgetallen.

Bew. (niet kennen voor het examen)
We bewijzen deze eigenschap uit het ongerijmde[1]. Veronderstel dat de zin onwaar is in $\mathfrak{A}$. Dus bestaat een priemgetal n zodat er geen groter priemgetal bestaat dan n. Dan is n het grootste priemgetal. Nu is het getal $m = (2 \times 3 \times \cdots \times n - 1 \times n) + 1$ zeker groter dan n en dus geen priemgetal, en aangezien $m-1$ deelbaar door elk getal in $[2,n]$ is m zelf ondeelbaar voor elk van die getallen. Maar elk getal dat geen priemgetal is, is te schrijven als een product van een aantal priemgetallen. Deze priemgetallen zijn delers van m en dus strikt groter dan n. Dat is een contradictie. Q.E.D.

Voorbeeld 2.5.12. De laatste zin die we bekijken is:

$$\begin{array}{ll} \forall x\colon & \{ \\ & \quad \exists w\colon S(w,w) = x \wedge \exists u\colon \exists v\colon \exists w\colon [L(u,v) \wedge L(v,w) \wedge L(w,x)] \\ & \quad \Rightarrow \\ & \quad \exists y\colon \exists z\colon [P(y) \wedge P(z) \wedge S(y,z) = x] \\ & \} \end{array}$$

Wat betekent deze zin? Daarvoor moeten we hem eerst ontleden.
- $\exists w\colon S(w,w) = x$ betekent dat x een even getal is.
- $\exists u\colon \exists v\colon \exists w\colon L(u,v) \wedge L(v,w) \wedge L(w,x)$ betekent dat x strikt groter is dan 2.
- de conclusie zegt dat x de som is van twee priemgetallen.

Deze zin betekent dus dat elk even getal $n > 2$ de som is van twee priemgetallen.

Is deze zin waar in $\mathfrak{A}$? Niemand weet het! Dit is *Goldbach's conjectuur*. De hypothese dateert al van 1742 en is nog altijd niet bewezen of ontkracht.

Oefening 2.5.3. *Ga na wat de betekenis is van de Goldbach's zin in de structuur $\mathfrak{A}_1$ waarin L geïnterpreteerd wordt als "... is strikt groter dan ..." in plaats van "... is kleiner dan ...". Bewijs dat de zin onwaar is. (door een waarde voor x te geven zodat de formule onwaar is.*

Oefening 2.5.4. *Neem Σ bestaande uit 1 relatiesymbool, namelijk $L/2$. Beschouw twee structuren:*

- *$\mathfrak{A}_1$ heeft als domein $\mathbb{Z}$, en $L^{\mathfrak{A}_1} = \{(n,m) \in \mathbb{Z} \times \mathbb{Z} | n < m\}$.*

[1] Een bewijs uit het ongerijmde vertrekt vanuit de hypothese dat het gegeven waar en het te bewijzen onwaar is. Vervolgens wordt een contradictie afgeleid.

- $\mathfrak{A}_2$ *heeft als domein* $\mathbb{Q}$*, en* $L^{\mathfrak{A}_2} = \{(n, m) \in \mathbb{Q} \times \mathbb{Q} | n < m\}$.

Toon aan dat de volgende zin waar is in één en onwaar in de andere structuur.

$$\forall x \colon \forall y \colon L(x, y) \Rightarrow \exists z \colon L(x, z) \wedge L(z, y)$$

Intermezzo 1. Een belangrijke vraag is of er algoritmes bestaan om de waarheid van zinnen in zo'n oneindige structuren te bepalen.

Voor sommige structuren is dat het geval, voor andere niet. Bv. voor de structuur $\mathfrak{A}$ bestaande uit de natuurlijke getallen met de standaard waardes voor de symbolen $zero/0 :, one/0 :, Som/2 :$ bestaat een algoritme om de waarheid van elke zin te bepalen. Deze eigenschap wordt de *beslisbaarheid van Presburger-rekenkunde* genoemd.

Maar als we $\mathfrak{A}$ uitbreiden voor een nieuw functiesymbool $Times/2$: waarvoor $Times^{\mathfrak{A}_2} : \mathbb{N}^2 \to \mathbb{N} : (n, m) \to n \times m$ (de productfunctie $\times$), dan bekomen we een structuur waarvoor geen algoritme bestaat dat de waarheid van elke zin kan bepalen. Deze eigenschap wordt de *onbeslisbaarheid van de natuurlijke getallen* of ook de *Stelling van Church-Turing* genoemd naar de wiskundige logici die dit oorspronkelijk bewezen hebben.

Beide eigenschappen zijn beroemde resultaten uit de logica. Het algoritme voor Presburger rekenkunde heeft veel toepassingen in de informatica.

2.6 Construeren van modellen van een zin

Het is vaak nuttig om (eindige) modellen te construeren voor een gegeven formule. Enerzijds geeft dit inzicht in de betekenis van formules. Anderzijds zijn er veel informaticatoepassingen waarin het op te lossen probleem essentieel neer komt op het zoeken van een model, of een deel van een model, van de gegeven informatie.

Voorbeeld 2.6.1. Veel studenten verwachten dat de volgorde van de kwantoren $\forall\exists$ en $\exists\forall$ geen rol speelt. We zoeken dit uit door een structuur op te stellen waarin de volgorde wel een rol speelt.

Neem Σ bestaande uit 1 predikaatsymbool $L/2$. We zoeken een structuur over Σ die voldoet aan volgende zin:

$$(\forall x \colon \exists y \colon L(x, y)) \wedge \neg(\exists y \colon \forall x \colon L(x, y))$$

De twee deelzinnen verschillen door een omwisseling van de kwantoren.

Vaak bestaan er heel kleine structuren waarin een zin waar is. Dit is zo'n geval. Neem de structuur $\mathfrak{A}$ met 2 domeinelementen:

- $D = \{a, b\}$
- $L^{\mathfrak{A}} = \{(a, a), (b, b)\}$

Het is makkelijk te verifiëren dat de zin voldaan is in deze structuur. Er geldt dat $\mathfrak{A} \models \forall x \colon \exists y \colon L(x, y)$ en dat $\mathfrak{A} \not\models \exists y \colon \forall x \colon L(x, y)$. Dus is de conjunctie van de eerste en de negatie van de tweede waar.

De volgende eigenschap laat het nut zien van het construeren van een structuur.

Eigenschap 2.6.1. De zinnen $\forall x \colon \exists y \colon L(x, y)$ en $\exists y \colon \forall x \colon L(x, y)$ zijn niet logisch equivalent.

Bew. Neem de structuur $\mathfrak{A}$ uit het vorige voorbeeld. Hierin is de eerste zin waar en de tweede zin onwaar. Dus zijn ze niet niet equivalent. Q.E.D.

Om te bewijzen dat het niet waar is dat een eigenschap geldig is in alle structuren volstaat 1 tegenvoorbeeld: 1 structuur waarin de eigenschap niet geldig is.

De eigenschap geeft aan dat het omwisselen van kwantoren de equivalentie niet bewaart. We komen hierop terug in Sectie 2.9.

Oefening 2.6.1. *Zoek een structuur die de gelijkaardige zin waar maakt:*

$$\neg\forall x\colon \exists y\colon L(x,y) \wedge \exists y\colon \forall x\colon L(x,y)$$

Vind je er zo één? Zo ja, bekijk dan eigenschap 3.4.6(3) in de discussie over het verwisselen van kwantoren in Hoofdstuk 3.

Oefening 2.6.2. *Construeer een structuur voor de zin:*

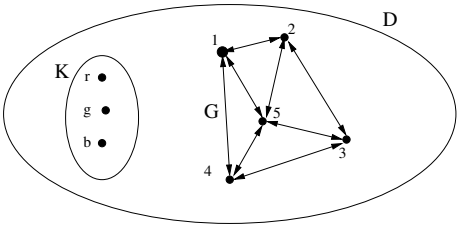
$$(\forall x\colon P(x) \Rightarrow Q(x)) \wedge \neg(\exists x\colon P(x) \wedge Q(x))$$

Dit is een structuur waarin alle P's Q's zijn en toch bestaat geen P die Q is.

Voorbeeld 2.6.2. Zij Σ bestaande uit predikaatsymbolen $G/2, K/1$ en functiesymbool $C/1$:. We zoeken een structuur die aan deze zin voldoet:

$$\forall x\colon K(C(x)) \wedge \forall x\colon \forall y\colon G(x,y) \Rightarrow \neg(C(x) = C(y))$$

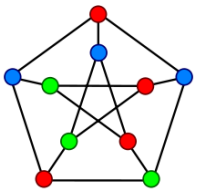
Bovendien moeten de interpretaties $K^{\mathfrak{A}}$ en $G^{\mathfrak{A}}$ zijn zoals de volgende figuur:



Dus $K^{\mathfrak{A}}$ bestaat uit 3 objecten, terwijl $G^{\mathfrak{A}}$ een symmetrische grafe is bestaande uit 16 koppels.[2] Tussen haakjes, een grafe is een 2-voudige relatie. Een symmetrische grafe bevat voor elke boog (a,b) ook zijn omgekeerde boog (b,a). Het probleem hier is dus om een functie $C^{\mathfrak{A}}$ te vinden zodat de volledige structuur voldoet aan de bovenstaande zin. Een oplossing is:

x	1	2	3	4	5	r	g	b
$C^{\mathfrak{A}}(x)$	b	g	b	g	r	r	g	b

Dit probleem is een *grafenkleuringsprobleem*, een veralgemening van het *kaartkleuringsprobleem* (map colouring), waarin het de bedoeling is om elk land zo te kleuren dat aanpalende landen verschillende kleuren hebben. $K^{\mathfrak{A}}$ kan beschouwd worden als een verzameling kleuren, $G^{\mathfrak{A}}$ als een grafe, en $C^{\mathfrak{A}}$ als een kleuring van elk node van de grafe (en ook van de kleuren, maar deze zijn irrelevant). Een grafekleuring is een toekenning van een kleur aan elke node zodat twee verbonden nodes verschillende kleuren hebben.



[2]Een symmetrische grafe bevat voor elke boog ook zijn omgekeerde. In de wiskunde worden symmetrisch grafen meestal gebruikt om ongerichte grafen voor te stellen.

Oefening 2.6.3. *Zoek een structuur voor Σ bestaande uit predikaatsymbool $L/2$ die aan volgende zin voldoet.*

$$(\forall x\colon \forall y\colon \forall z\colon L(x,y) \wedge L(y,z) \Rightarrow L(x,z)) \wedge$$
$$\neg(\exists x\colon L(x,x)) \wedge$$
$$(\forall x\colon \exists y\colon L(x,y))$$

Uw structuur mag oneindig zijn. Argumenteer waarom deze zin onwaar is in elke eindige structuur.

Oefening 2.6.4. *Beschouw de volgende uitspraak over de studentendatabank afkomstig uit Hoofdstuk 1:*

$$\exists x\colon \forall y\colon Lecturer(y) \Rightarrow \exists u\colon Instructor(y,u) \wedge Enrolled(x,u)$$

Pas de studentendatabank aan zodat deze zin waar wordt.

Dit is een variatie van het probleem van het construeren van een structuur omdat er al een structuur gegeven is.

Merk op dat een radicale oplossing is om de tabel van Lecturer leeg te maken. Immers dan is de premisse van de implicatie onwaar voor elke y, en dus is de implicatie voldaan. Zoek liever naar een kleine *aanpassing van de databank zodat de zin voldaan is. Dan doe je eigenlijk hetzelfde als wat een databankmanager doet als hij uitvist dat een integriteitsbeperking niet voldaan is en vervolgens de databank aanpast om de fout te herstellen. Let wel, misschien zijn er meerdere mogelijkheden.*

Je zult merken dat het opstellen van structuren die aan een zin voldoen veel inzicht geeft in deze zin. Maar het kan lijken alsof zoiets weinig praktisch nut heeft. Niets is minder waar. Een brede klasse van informaticaproblemen komt neer op het opstellen van structuren die aan bepaalde zinnen voldoen. Alleen al het grafekleuringsprobleem heeft veel toepassingen in de informatica. We zullen daarop terugkomen in Hoofdstuk 4.

2.7 Geo-werelden en Decawerelden

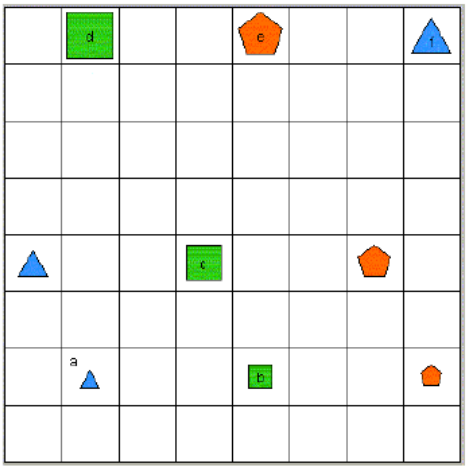
Een onderdeel van deze cursus zijn de oefeningen via het softwaretool LogicPalet. Die leren je (onder andere):

- hoe eigenschappen te vertalen in logica,
- hoe de waarheid van logische zinnen in een structuur te bepalen,
- hoe structuren te construeren waarin een logische zin waar of onwaar is.

Een eerste toepassing is Geo-werelden. Een Geo-wereld is een structuur van een vocabularium Σ_{Geo} dat een aantal constanten $a, b, c, \ldots$ bevat en de volgende relaties.

$Triangle/1$	... is een driehoek
$Square/1$	... is een vierkant
$Pentagon/1$	... is een vijfhoek
$Small/1$	... is klein
$Medium/1$	... is middelgroot
$Large/1$	... is groot
$Smaller/2$	... is strikt kleiner dan ...
$Larger/2$	... is strikt groter dan ...
$LeftOf/2$	... is strikt meer naar links gelegen dan ...
$RightOf/2$	... is strikt meer naar rechts gelegen dan ...
$FrontOf/2$	... is strikt meer naar voren gelegen dan ...
$BackOf/2$	... is strikt meer naar achteren gelegen dan ...
$Between/3$	... bevindt zich tussen ... en ...

Deze structuren worden door LogicPalet grafisch voorgesteld zoals in Figuur 2.1. In deze wereld



Figuur 2.4: Een Geo-wereld

geldt bv. $LeftOf(d, e)$ en $LeftOf(d, c)$ (merk op dat d en c niet op dezelfde hoogte hoeven te liggen).

Met LogicPalet kun je vervolgens:

- een nederlandse zin vertalen in een logische zin van Σ_{Geo}
- LogicPalet de waarheid van deze zin in een Geo-wereld laten berekenen
- met een grafische editor een nieuwe Geo-wereld samenstellen, bv. waarin een gegeven logische zin waar is.
- LogicPalet Geo-werelden laten genereren die aan een bepaalde logische zin voldoen
- De equivalentie van twee zinnen over Geo-werelden laten controleren door LogicPalet, of verifiëren of één zin de andere impliceert.

In deze toepassing ligt het vocabularium Σ_{Geo} vast (op de constanten na). In een andere deel van LogicPalet, DecaWorld, kun je zelf het vocabularium bepalen, maar het domein is beperkt tot hoogstens 10 elementen. Op het grafische na is de functionaliteit van het systeem dezelfde als voor Geo-werelden.

Er zijn een aantal huiswerken en tussentijdse toetsen die gemaakt moeten worden via LogicPalet. Informatie hierover wordt beschikbaar gemaakt via Toledo. Voor meer uitleg verwijzen we naar Toledo, de oefenzittingen en naar de documentatie bij LogicPalet.

2.8 Definities

Een definitie van een concept drukt uit welke objecten behoren tot het concept, en van alle resterende objecten, dat ze niet behoren tot het concept.

Bv., de definitie van het concept "priemgetal": het bestaat uit natuurlijke getallen strikt groter dan 1 die enkel deelbaar zijn door 1 en zichzelf. Elk object dat niet aan deze voorwaarde voldoet is geen priemgetal. Anders gezegd, een object is een priemgetal als en slechts als het een natuurlijk getal is dat strikt groter is dan 1 en enkel deelbaar is door 1 en zichzelf.

In FO kunnen we dit als volgt kunnen uitdrukken:

$$\forall n : Priem(n) \Leftrightarrow (Natnr(n) \wedge 1 < n \wedge \forall m : Deelt(m, n) \Leftrightarrow m = 1 \vee m = n)$$

Hierin interpreteren we het predicaatsymbool $Natnr/1$ als de verzameling van de natuurlijke getallen, $< /2$ als de gewone orde op de natuurlijke getallen, en $Deelt/2$ als de delingsrelatie.

Waarom is deze definitie correct? Wel, in elk model $\mathfrak{A}$ van dit axioma waarin $Nat, <$ en $Deelt$ de correcte waarden hebben [3] geldt dat $Priem^{\mathfrak{A}}$ gelijk is aan de verzameling van alle priemgetallen.

Maar is dit wel voldoende om een correcte definitie te zijn? Niet helemaal. Immers, neem de volgende logische zin:

$$Priem(0) \wedge \neg Priem(0)$$

Deze zin is overduidelijk geen correcte definitie van priemgetallen, want ze is contradictorisch en heeft geen modellen. Maar er geldt wel degelijk dat in *elk* model $\mathfrak{A}$ van deze zin, $Priem^{\mathfrak{A}}$ bestaat uit alle priemgetallen (want zijn geen modellen). Een goede definitie mag dus nooit inconsistent zijn.

Definities vormen een belangrijke en speciale vorm van kennis in wiskunde en wetenschappen. Vaak willen we een nieuw concept definiëren in termen van een aantal bestaande concepten die we de parameterconcepten van de definitie noemen. Een definitie moet aan de volgende eigenschappen voldoen:

- **uniciteit**: de definitie bepaalt een unieke waarde van het gedefinieerde concept uit de waarden voor de parameterconcepten;
- **consistentie**: voor elke waarde van de parameterconcepten moet de definitie een waarde bepalen voor het gedefinieerde concept. Merk op dat de inconsistente zin hierboven niet hieraan voldoet.

Nu volgt een verzameling van formules in predikatenlogica waarmee definities kunnen uitgedrukt worden.

[3] t.t.z. $Natnr^{\mathfrak{A}} = \mathbb{N}, <^{\mathfrak{A}} = \{(x, y) \in \mathbb{N}^2 | x < y\}$, en $Deelt^{\mathfrak{A}} = \{(n, m) \in \mathbb{N}^2 | \exists k : n \times k = m\}$.

Definitie 2.8.1. Een expliciete definitie Δ van een predicaat P/n is een logische zin van de vorm:

$$\forall x_1 : \cdots : \forall x_n : P(x_1, \ldots, x_n) \Leftrightarrow A_P[x_1, \ldots, x_n]$$

zodat P niet voorkomt in A_P.

Dus: n is de ariteit van P, er zijn n verschillende variabelen $x_1, \ldots, x_n$; ze zijn universeel gekwantificeerd en het zijn de enige gekwantificeerde variabelen; het gedefinieerde atoom $P(x_1, \ldots, x_n)$ bevat geen andere termen dan die variabelen. Vervolgens volgt het symbool $\Leftrightarrow$ en dan de definiërende formule A_P waarin P niet voorkomt.

De volgende stelling toont aan dat dit soort formules aan de condities (uniciteit) en (consistentie) voldoet. We noemen een structuur $\mathfrak{A}'$ een uitbreiding van $\mathfrak{A}$ als $\mathfrak{A}'$ bekomen wordt uit $\mathfrak{A}$ door waardes toe te kennen aan symbolen die niet geïnterpreteerd zijn door $\mathfrak{A}$.

Stelling 2.8.1. *Zij Δ een expliciete definitie over Σ voor predicaat P. Voor elke structuur $\mathfrak{A}$ van $\Sigma \setminus \{P\}$ bestaat exact één Σ-structuur $\mathfrak{A}'$ die $\mathfrak{A}$ uitbreidt en een model is van Δ, namelijk:*

$$\mathfrak{A}' = \mathfrak{A}[P : \{(x_1, \ldots, x_n) | A_P\}^{\mathfrak{A}}]$$

Dus, de unieke uitbreiding van $\mathfrak{A}$ die voldoet aan Δ is $\mathfrak{A}$ uitgebreid met de waarde $\{(x_1, \ldots, x_n) | A_P\}^{\mathfrak{A}}$ voor P. Dit is de verzameling $\{(d_1, \ldots, d_n) \in {D_{\mathfrak{A}}}^n | \mathfrak{A}[x_1 : d_1, \ldots, x_n : d_n] \models A_P\}$.

Bew. Het bewijs is in 2 stappen. De eerste stap is om aan te tonen dat $\mathfrak{A}'$ zoals gedefinieerd in de stelling, een model is van Δ. De tweede stap is om aan te tonen dat als een uitbreiding $\mathfrak{B}$ van $\mathfrak{A}$ voor P een model is van Δ, dan geldt dat $P^{\mathfrak{B}} = (\{(x_1, \ldots, x_n) | A_P\})^{\mathfrak{A}}$, i.e., $\mathfrak{B} = \mathfrak{A}'$. Beide stappen zijn eenvoudig. Zij D het domein van $\mathfrak{A}$.

Stap 1. Door toepassingen van de $\forall$-, de $\Leftrightarrow$- en de atoomregel bekomen we:

$\mathfrak{A}' \models \Delta$
asa $\forall d_1 \in D, \ldots, \forall d_n \in D : (d_1, \ldots, d_n) \in P^{\mathfrak{A}'}$ asa $\mathfrak{A}'[x_1 : d_1, \ldots, x_n : d_n] \models A_P$
asa $\forall d_1 \in D, \ldots, \forall d_n \in D : (d_1, \ldots, d_n) \in (\{(x_1, \ldots, x_n) | A_P\})^{\mathfrak{A}}$ asa $\mathfrak{A}[x_1 : d_1, \ldots, x_n : d_n] \models A_P$

In tweede herschrijving werd in het rechterlid $\mathfrak{A}'$ vereenvoudigd tot $\mathfrak{A}$; dit mag omdat P niet voorkomt in A_P (zie Propositie 2.3.1). De derde uitspraak is voldaan, per definitie van $(\{(x_1, \ldots, x_n) | A_P\})^{\mathfrak{A}}$. Dus geldt ook $\mathfrak{A}' \models \Delta$.

Stap 2. Zij $\mathfrak{B}$ een uitbreiding van $\mathfrak{A}$ zodat $\mathfrak{B} \models \Delta$. Hieruit volgt $\forall d_1 \in D, \ldots, \forall d_n \in D : (d_1, \ldots, d_n) \in P^{\mathfrak{B}}$ asa $\mathfrak{A}[x_1 : d_1, \ldots, x_n : d_n] \models A_P$. Dit betekent dat $P^{\mathfrak{B}} = (\{(x_1, \ldots, x_n) | A_P\})^{\mathfrak{A}}$.

Q.E.D.

Oefening 2.8.1. *Argumenteer waarom de expliciete definitie van $Priem/1$ als de verzameling van priemgetallen correct is, m.a.w., waarom in elk model $\mathfrak{A}$, $Priem^{\mathfrak{A}}$ bestaat uit de verzameling van priemgetalen. Gebruik Stelling 2.8.1.*

De stelling toont aan dat als we een nieuw symbool P introduceren en definiëren door middel van een expliciete definitie, dat geen beperkingen opgelegd worden aan de bestaande symbolen, en

dat de waarde van het nieuwe symbool P volkomen bepaald is door de waarde van de bestaande symbolen en de expliciete definitie.

Zelfs een geringe afwijking van het formaat van een expliciete definitie kan leiden tot een fout.

Voorbeeld 2.8.1. Veronderstel dat we het concept van koortsige mensen willen definiëren. Dit zijn de mensen met een temperatuur groter dan 37C. Wat te denken van de volgende zin?

$$\forall x\colon \forall t\colon Koorts(x) \Leftrightarrow Mens(x) \wedge Temperatuur(x,t) \wedge t > 37.$$

Merk op: deze zin wijkt enkel af van een expliciete definitie door de kwantificatie $\forall t$ vooraan.

We analyzeren deze zin gebruik makend van twee logische equivalenties: $A \Leftrightarrow B$ is logisch equivalent met $(A \Rightarrow B) \wedge (A \Leftarrow B)$, en $\forall x : (A \wedge B)$ is logisch equivalent met $(\forall x : A) \wedge (\forall x : B)$. Gebruikmakend hiervan kan de formule voor $Koorts$ eerst opgesplitst worden in een conjunctie van twee implicaties; vervolgens kunnen de kwantoren $\forall x, \forall t$ gedistribueerd worden over de conjunctie. We bekomen 2 implicaties:

- $\forall x\colon \forall t\colon Koorts(x) \Leftarrow Mens(x) \wedge Temperatuur(x,t) \wedge t > 37$. Deze zin stelt dat elk object dat een mens is met temperatuur boven 37, koorts heeft. Dit is correct.
- $\forall x\colon \forall t\colon Koorts(x) \Rightarrow Mens(x) \wedge Temperatuur(x,t) \wedge t > 37$. Deze zin stelt dat voor een willeurig paar van objecten x en t, als x koorts heeft, dan is x een mens met temperatuur t en t is strikt meer dan 37. Dit is duidelijk totaal verkeerd. Neem bv.een koortsige hond als waarde voor x en 0 als waarde voor t, dan zou moeten gelden dat die hond een mens is en 0¿37.

De correcte manier om koorts te definiëren is door de volgende expliciete definitie:

$$\forall x\colon Koorts(x) \Leftrightarrow \exists t\colon Mens(x) \wedge Temperatuur(x,t) \wedge t > 37 \tag{2.1}$$

Oefening 2.8.2. *Bewijs de gebruikte equivalenties uit het voorbeeld ofwel rechtstreeks, ofwel door gebruik te maken van de wetten in het volgende hoofdstuk.*

Oefening 2.8.3. *Neem de volgende "definitie" van een even getal.*

$$\forall n\colon Even(2 \times n) \Leftrightarrow Nat(n)$$

Deze zin is geen expliciete definitie. Toon aan dat de zin niet correct uitdrukt wat even getallen zijn. Schrijf zelf een expliciete definitie op die $Even/1$ correct definieert.

Niet alle formules met $\Leftrightarrow$ zijn definities. Bv. we kunnen de exclusieve disjunctie "ofwel A, ofwel B" uitdrukken door

$$A \Leftrightarrow \neg B$$

Deze formule is correct, en drukt geen definitie uit.

Inductieve definities? De conditie op expliciete definities is dat het gedefinieerde predicaat P niet mag voorkomen in de definiërende formule A_P. Hiermee kunnen dus geen inductieve definities voorgesteld worden.

In de cursus Modelleren van Complexe Systemen in de 1ste Master Computerwetenschappen zullen we bewijzen dat inductieve definities in het algemeen niet uitgedrukt kunnen worden in predikatenlogica. Soms wel, soms niet. Bv. we kunnen de recursieve definitie van + in de natuurlijke getallen uitdrukken; maar niet de inductieve definitie in het volgende voorbeeld.

Voorbeeld 2.8.2. Zij G een binaire relatie. G kan beschouwd worden als een gerichte grafe met een pijl van x naar y voor elk koppel $(x, y) \in G$.

We zeggen dat er een pad is van objecten x naar y in G als er een eindige rij $x_0, \ldots, x_n$ is zodat $x = x_0, y = y_n$, en voor elke $i \in [0, n-1]$, geldt $(x_i, x_{i+1}) \in G$. In een grafische notatie: als we y kunnen bereiken vanuit x door pijlen van G te volgen. Zij P de relatie bestaande uit alle koppels (x, y) zodat een pad van x naar y in G bestaat.

Men kan aantonen dat P inductief gedefinierd kan worden als volgt:
- $(x, y) \in P$ indien $(x, y) \in G$
- $(x, y) \in P$ indien er bestaat een object z zodat $(x, z), (z, y) \in P$

Deze inductieve definitie definieert P als de kleinste relatie die G omvat en die transitief is.

De definitie van P in functie van G kan niet uitgedrukt worden in Predikatenlogica. Probeer maar eens. Als je een oplossing hebt waarvan je niet weet wat er fout is, zet het op het discussieforum en geef me een seintje.

Oefening 2.8.4. *Toon aan door inductie dat de inductief gedefinieerde relatie P bestaat uit exact alle koppels (x, y) zodat er een pad bestaat van x naar y in G. Kies vervolgens een kleine grafe G (zorg dat er lussen inzitten) en bereken P door het inductie-proces uit te voeren.*

2.9 Pragmatiek van predikatenlogica

De informele betekenis $\mathcal{I}(A)$ van een logische zin A is een gedachte die geformuleerd kan worden als een soort wiskundige zin. Dit levert ons een precieze vertaling van logische zinnen naar natuurlijke taal.

De omgekeerde weg is veel moeilijker: de omzetting van uitspraken in natuurlijke taal naar logica. Natuurlijke taal (NT) is complex, onsystematisch, soms ambigu. Vele woorden zijn overladen: ze hebben meerdere betekenissen, en er zijn veel zinspatronen die niet betekenen wat ze lijken te zeggen.

Bv. “Kunt u mij het zout aanreiken?”: strikt genomen is dit de vraag of iemand in staat is om het zout aan te reiken, maar bedoeld is het verzoek om dat zout aan te geven.

Ambiguïteit in natuurlijke taal komt vrij frequent voor. Bv. gehoord op de radio:

De nieuwe gevangenis van Tielt gaat eindelijk open.

Eén betekenis is dat in Tielt een nieuwe gevangenis in werking treedt. Een andere betekenis is dat de nieuwe gevangenis in Tielt zijn deuren open zet en gevangen laat ontsnappen.

Pragmatiek is de kunst om NT uitdrukkingen te interpreteren en om te zetten in logica.

Voorbeeld 2.9.1. Vergelijk twee gesprekken tussen personen A en B.

- Eerste gesprek:

  A: “Vertel me over uw familie.”
  B: “**Ik heb een zus die Evangelina heet.** ”
  Hierna zwijgt B.

- Tweede gesprek:

  A: "Mijn vrouw heet Evangelina"
  B (verbaasd): "**Ik heb een zus die Evangelina heet.**"

In beide gevallen zegt B hetzelfde. Maar is de betekenis van B's antwoord hetzelfde in beide gesprekken? Zeker niet. In het eerste gesprek bedoelt B dat hij slechts 1 zus heeft, Evangelina. Waarschijnlijk bedoelt hij ook dat hij geen broers of levende ouders heeft. In het tweede gesprek ontbreekt deze bedoeling. We komen enkel te weten dat B een zus heeft die Evangelina heet. B kan nog meer zussen hebben, en ook broers en levende ouders. We weten het gewoon niet.

Voorbeeld 2.9.2. Dezelfde zin uitgesproken met verschillende nadruk. De benadrukte woorden zijn onderlijnd:

- Ik heb niet gereden met uw auto.
- Ik heb niet gereden met uw auto.

Hier wordt hetzelfde gezegd. Maar wordt hetzelfde bedoeld? Duidelijk niet.

Voorbeeld 2.9.3. Zelfs in de wiskunde komen er taalpatronen voor die iets anders bedoelen dan wat ze letterlijk lijken te zeggen.

Definitie: *Een natuurlijk getal $n > 1$ is priem als 1 en n de enige delers zijn van n.*

Deze definitie heeft de vorm van de volgende logische formule:

$$\forall n : (Natnr(n) \wedge n > 1 \Rightarrow (Priem(n) \Leftarrow (\forall m : Deelt(m, n) \Leftrightarrow m = 1 \vee m = n)))?$$

Maar drukt deze logische zin uit wat de definitie zegt over het concept "priem"? In geen geval.

De oorspronkelijke definitie bedoelt niet alleen om aan te geven wat priem is maar ook wat niet priem is.

De logische zin daarentegen geeft wel aan welke objecten priem zijn maar niet welke niet priem zijn. Volgens de logische zin zou elk ander object in het universum ook priem *kunnen* zijn.

Oefening 2.9.1. *Denk na over een methode om deze laatste uitspraak formeel na te gaan : hoe nagaan dat volgens die zin elk object priem zou kunnen zijn? Vervolgens, pas uw methode toe op de logische zin. Je mag ervan uitgaan dat Natnr en Deelt de bedoelde betekenis hebben: Natnr bevat de natuurlijke getallen en Deelt bestaat uit de koppels (n, m) van natuurlijke getallen zodat n een deler is van m.*

De correcte manier om deze definitie uit te drukken is door middel van de expliciete definitiea:

$$\forall n : (Priem(n) \Leftrightarrow (Natnr(n) \wedge n > 1 \wedge (\forall m : Deelt(m, n) \Leftrightarrow m = 1 \vee m = n)))!!!$$

Oefening 2.9.2. *Denk na hoe je kan nagaan dat deze zin uitdrukt wat priem is en van alle andere dingen, dat ze niet priem zijn. Gebruik je methode vervolgens om na te gaan dat deze zin correct is.*

Linguïsten spreken van de logische betekenis van een zin en de pragmatische betekenis van een NT zin. We zouden dit kunnen noemen: de letterlijke betekenis van een zin en de bedoelde betekenis.

De voorbeelden hierboven laten zien dat de pragmatische betekenis van een NT uitspraak afhankelijk is van context (bv. in welk gesprek vindt de uitspraak plaats, is de zin al dan niet een definitie?) of zelfs van de manier waarop de zin wordt uitgesproken (bv., de nadruk). Meestal is de pragmatische betekenis sterker dan de letterlijke betekenis. De extra gedachten die niet letterlijk gezegd worden maar wel bedoeld zijn, worden **implicaturen** genoemd. Bv. een implicatuur van de NT definitie van "priem" is dat alle objecten die niet voldoen aan de conditie: "natuurlijk getal groter dan 2 en enkel gedeeld door 1 en zichzelf" niet "priem" zijn. [4]

Wie een logische theorie bouwt in een toepassing doet dit meestal op basis van teksten. In dat geval is het de pragmatische betekenis van de tekst die in logica uitgedrukt moet worden, niet de letterlijke.

Pragmatiek van predikatenlogica is de kunst om de pragmatische betekenis van gesproken of geschreven uitspraken om te zetten in een logische theorie.

Er bestaat voorlopig geen algoritme hiervoor. Het is een kunst die we moeten leren. De eerste stap is om goed te analyseren wat de pragmatische betekenis van de NT uitspaak is. We zijn allemaal uitgerust met een groot onbewust vermogen om de pragmatische betekenis van zinnen te detecteren. Soms detecteren we dat zinnen inherent onduidelijk en ambigu zijn. Soms maken we fouten en interpreteren we de zinnen toch verkeerd. Maar vaak zijn we in staat te detecteren wat bedoeld wordt. Denk aan al de keren dat een communicatie succesvol is, dat je iemand of een tekst goed begrepen hebt. In elk van deze situaties heb je de implicaturen in de woorden van de spreker of de tekst correct opgevangen en begrepen.

Oefening 2.9.3. *Wat zijn de implicaturen in Voorbeeld 2.9.1 en Voorbeeld 2.9.2? Introduceer in beide voorbeelden een logisch vocabularium Σ voor de relevante concepten die voorkomen in de uitspraak en schrijf zowel de letterlijke betekenis als de bedoelde betekenis van deze zinnen uit in logica.*

Hieronder gaan we in op een aantal punten waar het gebruik van logica gemakkelijk tot subtiele fouten kan leiden.

2.9.1 Functiesymbolen stellen totale functies voor

Natuurlijke taal kent veel functionele concepten. Bv. de leeftijd van ..., de moeder van ..., de som van twee getallen, de partner van ..., de kleur van een node (confer Voorbeeld 2.6.2 over het kleuren van grafes). We zijn geneigd om deze in predikatenlogica voor te stellen door middel van functiesymbolen.

Er is echter een semantische mismatch tussen NT en logica op het punt van functies: in natuurlijke taal zijn de functionele concepten altijd partiële functies terwijl functiesymbolen in predicatenlogica verondersteld zijn totale functies voor te stellen.

In natuurlijke taal zijn functionele expressies nooit gedefinieerd in alle objecten. Bv. een getal heeft geen partner of leeftijd, de som van een cursus en een student is niet gedefinieerd, en de kleur-functie om een grafe te kleuren is niet gedefinieerd in kleuren.

Predikatenlogica veronderstelt dat alle n-voudige functiesymbolen f/n : totale functies zijn. Zie daarvoor Definitie 2.3.1: daarin staat uitdrukkelijk dat de waarde van f in structuur $\mathfrak{A}$ een totale functie is van $D_{\mathfrak{A}}{}^n$ naar $D_{\mathfrak{A}}$.

[4] Er zijn ook NT uitspraken waarvan de pragmatische betekenis de negatie is van wat gezegd wordt. Bv. sarcastische zinnen.

Oefening 2.9.4. *Bewijs dat voor elk functiesymbool f/n : de volgende zin logisch waar is :*

$$\forall x_1 \colon \ldots \forall x_n \colon \exists y \colon f(x_1, \ldots, x_n) = y$$

In vele logische theoriën worden functiesymbolen gebruikt om partiële functies voor te stellen. Dit geeft dan aanleiding tot functionele termen die informeel niet gedefinieerd zijn. Bv., de overtollige bogen in Voorbeeld 2.6.2 die de kleur van een kleur definieert (bv. $C(r) = r$ waar r staat voor de rode kleur).

Een oplossing is: met de nodige voorzichtigheid kunnen we deze overtollige functie-elementen aanvaarden en negeren. Bv. door in elke uitspraak over $C(x)$ te specifiëren dat x een node is, zoals in de zin:

$$\forall x : Node(x) \Rightarrow Kleur(C(x))$$

Een alternatieve oplossing is om de mismatch te vermijden door een n-voudige partiële functie voor te stellen door een $n+1$-voudig predikaatsymbool.

Oefening 2.9.5. *Herformuleer de theorie van Voorbeeld 2.6.2 door het functiesymbool $C/1$: te vervangen door het predikaat $C/2$.*

2.9.2 Kwantificeren over een klasse

Kwantificatie in logica is verschillend dan in natuurlijke taal. In logica kwantificeren we over alle objecten van het volledige domein. Dit noemt men unaire kwantificatie. In natuurlijke taal kwantificeren we altijd over een bepaalde klasse van objecten. Dit noemt men binaire kwantificatie. Hetzelfde geldt voor wiskunde, waar notaties bestaan om de klasse waarover gekwantificeerd wordt uit te drukken. Bv. $\forall n \in \mathbb{N} : n - n = 0$.

Bv. alle *rijke managers* hebben een villa; sommige *rijke managers* hebben een rolls royce; alle *hoorcolleges van dit vak* vinden plaats in gebouw N. Alle *daklozen in Brussel* hebben recht op gratis soep van het OCMW. Sommige *indiase guru's* zijn steenrijk. Geen enkel *vak* is voorkennis van zichzelf. De meeste *olmen in Vlaanderen* zijn ziek. Enz.

De reden voor dit verschil is de volgende. in het nederlands kwantificeren we vrijwel nooit over alle objecten van het universum. Immers, het universum is zo divers dat er geen enkele interessante eigenschap gedeeld wordt door *alle* objecten uit het universum. Daarom kwantificeren we altijd over een specifieke klasse van objecten of wezens. In logica daarentegen zijn er heel wat toepassingen waar het universum bestaat uit een uniforme verzameling. Bv. een logische theorie over de natuurlijke getallen. In zo'n situatie is unaire kwantificatie prima. Binaire kwantificatie kan bovendien gesimuleerd worden met unaire.

De betekenis van een binaire gekwantificeerde zin hangt vanzelfsprekend af van de klasse waarover gekwantificeeerd wordt. In logica kan men binaire kwantificatie simuleren. Bv. om uit te drukken dat alle P's voldoen aan Q :

$$\forall x : P(x) \Rightarrow Q(x)$$

of

$$\forall x : \neg P(x) \vee Q(x)$$

Bv. om uit te drukken dat er een P bestaat die voldoet aan Q :

$$\exists x : P(x) \wedge Q(x)$$

Let op de assymmetrie tussen $\forall$ en $\exists$: de manier waarop de gekwantificeerde klasse uitgedrukt worden is verschillend ($P(x) \Rightarrow \ldots$ versus $P(x) \wedge \ldots$). Sommige studenten halen die twee manieren door elkaar, wat leidt tot catastrofale fouten.

Laat ons vergelijken hoe de volgende uitspraken in logica vertaald worden:

- "*Alle rijke managers bezitten een villa*"
- "*Er is een rijke manager die een villa bezit*"

Als we deze vertalen naar logica, gaat deze structurele gelijkheid verloren:

- $\forall x\colon Rich(x) \wedge Mgr(x) \Rightarrow Villa(x)$
- $\exists x\colon Rich(x) \wedge Mgr(x) \wedge Villa(x)$

Wat gebeurt er als de vertaling voor de ene kwantor gebruikt wordt voor de andere? Voor de alle-kwantor levert dit de volgende logische zin op:

$$\forall x\colon Rich(x) \wedge Mgr(x) \wedge Villa(x)$$

Deze zin zegt dat het universum alleen bestaat uit rijke managers met een villa. Moest dat waar zijn! Dit is volkomen verkeerd.

Voor de existentiële kwantor krijgen we de volgende zin:

$$\exists x\colon Rich(x) \wedge Mgr(x) \Rightarrow Villa(x) \quad \text{of equivalent} \quad \exists x\colon \neg(Rich(x) \wedge Mgr(x)) \vee Villa(x)$$

Deze zin is waar indien er een object in het domein bestaat die geen rijke manager is, hetgeen zeker voldaan is. De zin is ook waar indien er een object is dat een villa bezit. Ook deze zin is volkomen verkeerd.

We zien dat die foute vertalingen een totaal andere betekenis hebben dan de bedoelde. Dit zijn catastrofale fouten.

Opmerking 2.9.1. Veel huiswerken en opgaves in GeoWorld en DecaWorld bevatten kwantificatie over subklassen. Bovenstaande fout zal gedetecteerd worden door het systeem. LogicPalet kan ook gebruikt worden om uit te vinden waarom een zin waar of onwaar is.

2.9.3 Veralgemeende kwantoren in natuurlijke taal

In natuurlijke taal verschillen kwantoren van de logische kwantoren op minstens 2 manieren. Ten eerste wordt zoals gezegd in natuurlijke taal nooit gekwantificeerd over alle objecten van het universum maar steeds over een klasse. Zie vorig punt. Ten tweede zijn er veel meer kwantoren dan de twee logische.

- Alle P's zijn Q's.
- Minstens 1 P is een Q.
- Geen P's zijn Q's.
- Niet alle P's zijn Q's.
- Enkel P's zijn Q's.
- Behalve c zijn alle P's Q's.
- Niet één P is een Q.

- Minstens “n” P’s zijn Q’s.
- Hoogstens “n” P’s zijn Q’s.
- Precies “n” P’s zijn Q’s.
- De meeste P’s zijn Q’s.
- Weinig P’s zijn Q’s.
- ...

Elke gekwantificeerde zin in het nederlands bestaat uit 3 delen:
- een *kwalificatie* : de klasse van objecten waarover men kwantificeert
- een *kwantor* : die selecteert over welke objecten van die klasse men wilt spreken : alle, minstens 1, enz.
- een *assertie*: de uitspraak over de geselecteerde objecten.

Bv. *“Alle olmen in Vlaanderen zijn ziek”*: kwalificatie “*olmen in Vlaanderen*”, kwantor: “*alle*”, assertie: *“ze zijn ziek”*.

Hierbij kan de kwalificatie P zelf een samengestelde zin zijn die zelf ook kwantoren bevat, zoals in “Alle driehoeken links van een vierkant zijn blauw”. De deelzin “driehoeken links van een vierkant” is een existentieel gekwantificeerde deelzin: “er bestaat een vierkant waarvan de driehoek links ligt”. Dus, in deze deelzin speelt het lidwoord “een” de rol van een existentiële kwantor.

De meeste kwantoren in de lijst hierboven kunnen uitgedrukt worden met de twee logische kwantoren. Al kunnen de formules wel lang worden.

Oefening 2.9.6. *Veronderstel dat de kwalificatie uitgedrukt kan worden als een formule $A[x]$ met variabele x die een plaatshouder is voor de objecten waarover we spreken; en de assertie door $B[x]$, een uitspraak over diezelfde variabele x. Probeer de volgende veralgemeende kwantificaties naar logica te vertalen:*
- *Elke x die voldoet aan $A[x]$ voldoet aan $B[x]$.*
- *Minstens 1*
- *Geen*
- *Niet alle*
- *Niet één*
- *Minstens “n”*
- *Hoogstens “n”*
- *Niet meer (minder) dan “n”*
- *Precies “n”*
- *De meeste*
- *Weinig*

Een paar kwantoren kunnen niet uitgedrukt worden met $\forall, \exists$. Dit is het geval met “de meeste P’s” en met “weinig P’s”. Dit zijn “vage” kwantoren.

Je zult vaststellen dat predikatenlogica niet erg gebruiksvriendelijk is om precies n, hoogstens n, minstens n uit te drukken, zeker niet voor grotere n. Om een goede specificatietaal te bekomen moet predikatenlogica uitgebreid worden met extra kwantoren zoals $\exists_{\geq n}, \exists_{\leq n}, \exists_{=n}$.

Oefening 2.9.7. *Wat is de negatie van elke kwantor? Welke van deze kwantoren zijn equivalent? Welke worden geïmpliceerd door andere?*

Hieronder volgen nog een paar subtiele aspecten van kwantificatie.

Kwantificeren over lege verzamelingen Normaal impliceert $\forall x : P(x) \Rightarrow Q(x)$ dat $\exists x : P(x) \wedge Q(x)$, maar niet altijd.

ALs P leeg is dan is $\forall x : P(x) \Rightarrow Q(x)$ waar, en $\exists x : P(x) \wedge Q(x)$ onwaar.

In natuurlijke taal vermijden we kwantificaties met een lege kwalificatie. Waar ze voorkomen zijn ze verdacht. Vaak, maar niet altijd, zijn het fouten die sprekers maken.

Geneste kwantoren. Kwantoren in nederlandse zinnen komen vaak genest voor.

Neem de volgende Geo-world eigenschap: "Minstens één driehoek bevindt zich vóór elk vierkant dat rechts van alle vijfhoeken ligt." In deze zin komen drie kwantoren in het nederlands voor die in mekaar genest zijn: minstens één driehoek ... elk vierkant ... alle vijfhoeken.

Door deze zin zorgvuldig te ontleden is het niet zo moeilijk om te vertalen naar logica.

Oefening 2.9.8. *Zet deze zin om in logica.*

Oefening 2.9.9. *Als een ander voorbeeld, ontleed en druk uit "Minstens twee vierkanten liggen achter al de driehoeken die zich rechts van alle vijfhoeken bevinden".*

Volgorde van kwantoren De volgorde van kwantoren is van belang. Dit hebben we al eerder vastgesteld, namelijk in Opmerking 2.5.2 en Voorbeeld 2.6.1. Vergelijk de volgende zinnen:

- er is een sleutel die op elke deur past
- elke deur heeft een sleutel die erop past.

Ze verschillen in de volgorde van kwantoren en in betekenis. Formeel:

- $\exists x\colon Sleutel(x) \wedge \forall y\colon Deur(y) \Rightarrow Past(x, y)$
- $\forall y\colon Deur(y) \Rightarrow \exists x\colon Sleutel(x) \wedge Past(x, y)$

Daarentegen zullen we in het volgende hoofdstuk bewijzen dat gelijke kwantoren wel omgewisseld mogen worden. Bv.

- $\exists x\colon \exists y\colon Grade(x, y, AAA)$
- $\exists y\colon \exists x\colon Grade(x, y, AAA)$

Deze formules verschillen in de volgorde van de kwantoren maar zijn equivalent.

Ambigue kwantificatie In nederlandse tekst is de volgorde en zelfs de aard van de kwantificatie niet altijd eenduidig vastgelegd in de tekst. Soms komt de buitenste logische kwantor achteraan in de NT zin.

- Er is een bord waaruit alle aanwezigen eten: $\exists\forall$.
- Er is een bord om uit te eten voor alle aanwezigen: $\forall\exists$.

Beide zinnen bevatten een existentiële en een universele kwantor, in dezelfde tekstuele volgorde. Bij nader toezien is de *logische* volgorde van de kwantoren verschillend in beide zinnen. De tweede zin zegt dat elke aanwezige een (eigen) bord heeft. Uitgedrukt in logica:

- $\exists x\colon Bord(x) \wedge \forall y\colon Aanwezig(y) \Rightarrow EetUit(y, x)$
- $\forall y\colon Aanwezig(y) \Rightarrow \exists x\colon Bord(x) \wedge Bezit(y, x)$.

Een ander probleem is dat hetzelfde woord in het nederlands soms gebruikt kan worden voor verschillend kwantoren. Neem volgende zinnen:

- *Een mercedes is duur.*

– *Een mercedes staat voor mijn deur.*

Wat is de kwantor in beide zinnen? Beide zinnen beginnen met "Een" en zijn in het enkelvoud gesteld. Toch is de onderliggende kwantor verschillend. Ons taalgevoel vertelt ons dat in de eerste zin de kwantificatie universeel is, over alle mercedessen, in de tweede is het existentieel.

Hoewel ons taalgevoel ons duidelijk vertelt wat de kwantificatie is in beide gevallen, is het niet makkelijk om dit verschil in betekenis te verklaren. Dit laat zien dat taalinterpretatie een complex onderbewust proces is. Ook dat het expliciteren van impliciete kwantoren in tekst oefening vereist. Maar, oefening baart kunst.

2.9.4 Implicaturen van definities

In wiskundige teksten en ook elders worden definities vaak geformuleerd op een manier dat de letterlijke betekenis beduidend afwijkt van de pragmatische betekenis.

Bv., zie Voorbeeld 2.9.3 voor de definitie van priem getal.

Bv., men definieert: een student krijgt een studiebeurs als hij of zij inwoont bij de ouders, en het inkomen van de ouders onder een bepaalde grens ligt. Een implicatuur is natuurlijk dat iemand die niet aan deze voorwaarden voldoet, geen studiebeurs krijgt.

Natuurlijke taal bevat veel uidrukkingen die neerkomen op definities. Bv. het woordje "enkel" (niet het lichaamsdeel, maar het bijwoord). [5]

Bv. veel opsommingen hebben als implicatuur om een bepaald predicaat te definiëren.

Bv., neem de tabellen van een databank. Elke tabel somt de exacte inhoud van het tabelpredicaat op. Het bezit de definitie-implicatuur.

Oefening 2.9.10. *Druk de betekenis van de tabel van $Prerequ/2$ van de studentendatabank uit in een formule. Gebruik je een expliciete definitie? Leg uit waarom (niet).*

Voorbeeld 2.9.4. Stel A geeft een feest.
B vraagt: "Wie komt er allemaal?"
A antwoordt: "Bill en Jill'
Of A antwoordt:"Enkel Bill en Jill komen naar het feest".
In beide gevallen is A's implicatuur wellicht dat Bill en Jill de enige feestgangers zijn. In logica:

$$\forall x : Feest(x) \Leftrightarrow x = Bill \vee x = Jill$$

Voorbeeld 2.9.5. Niet alle opsommingen bevatten de definitie-implicatuur. Stel dat A graag wil dat B komt naar het feest. Wetende dat B's beste vrienden Jill en Bill zijn, zegt A: "Ik nodig je uit op mijn feest. Jill en Bill komen ook."
Ook deze zin bevat een opsomming, maar hier is wellicht geen definitie-implicatuur. In logica:

$$Feest(Jill) \wedge Feest(Bill)$$

2.9.5 Implicaturen van namen

Neem de volgende formule A:

$$\forall x\colon Aanwezig(x) \Leftrightarrow x = Jan \vee x = An$$

[5] Zie ook hoe in de definitie van priemgetallen n de conditie "enkel gedeeld door 1 en zichzelf" werd geformaliseerd.

We zouden verwachten dat deze formule impliceert dat er exact twee objecten behoren tot *Aanwezig*. Dus dat in elk model $\mathfrak{A}$ van A, $Aanwezig^{\mathfrak{A}}$ twee objecten bevat. Dit is niet zo.

We zouden verwachten dat A impliceert dat Jill niet aanwezig is. Dus dat in model $\mathfrak{A}$ van A het volgende geldt:

$$\mathfrak{A} \models \neg Aanwezig(Jill)$$

Ook dat is niet zo. Een tegenvoorbeeld is de structuur $\mathfrak{A}$ met domein $D_{\mathfrak{A}} = \{a\}$, $Jan^{\mathfrak{A}} = An^{\mathfrak{A}} = Jill^{\mathfrak{A}} = a$ en $Aanwezig^{\mathfrak{A}} = \{a\}$. Er is maar 1 iemand aanwezig. Er geldt:

$$\mathfrak{A} \models Aanwezig(Jill)$$

In natuurlijke taal schuilt er een implicatuur in het gebruik van **namen**: dat ze verschillende objecten aanduiden. In predikatenlogica moet dit expliciet uitgedrukt worden. In dit geval:

$$\neg(Jan = An) \wedge \neg(Jan = Jill) \wedge \neg(An = Jill)$$

Na toevoeging van deze zin volgen de gewenste eigenschappen.

2.9.6 Het domein uitdrukken

Soms willen we uitspraak doen over het domein/universum . Bv. veronderstel dat we willen uitdrukken dat de enige objecten in het domein Ray en CS248 zijn:

$$\forall x\colon x = Ray \vee x = CS248$$

2.9.7 De materiële implicatie.

Het implicatiesymbool $\Rightarrow$ in predikatenlogica wordt de **materiële implicatie** genoemd.

Over het algemeen is er een zeer goede match in de betekenis van de logische connectieven en hun natuurlijke taal versie ($\wedge$ versus "en", $\vee$ versus "(inclusieve) of", enz.). Dat is niet verbazend aangezien de regels van Definitie 2.3.6 de logische connectieven gewoon vertalen in de NT versie. [6]

De uitzondering is $\Rightarrow$. We lezen die als "als... dan...". In de waarheidsdefinitie werd $\mathfrak{A} \models A \Rightarrow B$ niet vertaald tot " als $\mathfrak{A} \models A$ dan $\mathfrak{A} \models B$" maar in "$\mathfrak{A} \not\models A$ of $\mathfrak{A} \models B$". De reden is dat *"als ... dan ..."* een taaluitdrukking is die wij mensen onbewust op teveel verschillende manieren interpreteren *al naargelang de context.* [7] Daarom werd het gebruik van "als... dan..." in Definitie 2.3.6 vermeden.

Logica mag niet ambigu zijn, dus moet elk logisch symbool een unieke betekenis hebben. Zo ook $\Rightarrow$. In predikatenlogica kreeg het de betekenis $\neg \cdots \vee \ldots$, een betekenis die vaak van toepassing is, zeker in de wiskunde en in informaticatoepassingen.

[6] Maar let op: ook "en" en "of" hebben andere betekenissen dan gebruikt in Definitie 2.3.6, bv. de temporele "en" en de exclusieve "of".

[7] Linguïsten noemen "als... dan..." een conditionele uitdrukking. Het is één van de meest overladen uitdrukkingen van natuurlijke taal. Linguïsten hebben meer dan 20 verschillende betekenissen ontdekt. Naast de gewone materiële, is er bv. ook de definitionele "als" zoals gebruikt werd in de NT definitie van Priem. Er is ook een causale betekenis zoals in "Als je een vuurtje stookt dan wordt het warm". Deze betekenen alle 3 iets verschillend. Zie het concept "conditional" in Wikipedia voor een beetje uitleg.

Onvermijdelijk is $\Rightarrow$ niet geschikt zijn voor het uitdrukken van een nederlandse zin die gebruikt maakt van een andere betekenis van "als ... dan ...". Bv., in de NT definitie van priemgetal komt een conditionele uitdrukking voor. Zoals we zagen in Sectie 2.8 was $\Rightarrow$ veel te zwak om die definitie uit te drukken; een expliciete definitie was vereist. Vele "als... dan..."-zinnen hebben de definitionele implicatuur, en voor hen is $\Rightarrow$ ongeschikt.

Er zijn nog meer frappante en onverwachte "als ... dan ... " zinnen waar materiële implicatie niet de juiste interpretatie is.

Voorbeeld 2.9.6. De volgende zinnen gaan over de condities om de Bachelor Informatica te behalen. Het universum waarover ze spreken is de verzameling van vakken. Bekijk de volgende conditionele zinnen en een voorgestelde vertaling in predicatenlogica:

- Als je slaagt voor alle vakken, dan behaal je de bachelor Informatica:

$$(\forall c\colon Succ(c)) \Rightarrow Pass.$$

- Er is een vak zodat, als je ervoor slaagt, dan behaal je de bachelor Informatica:

$$\exists c\colon (Succ(c) \Rightarrow Pass).$$

Zijn deze vertalingen correct?

De nederlandse zinnen zijn zeker niet equivalent. Immers, in het huidig examenregelment is de eerste NT zin waar voor de bachelor Informatica, de tweede is onwaar.

Vreemd genoeg zijn de logische formules wel logisch equivalent. Ze kunnen omgezet worden in mekaar door middel van de fundamentele equivalenties die we in het volgende hoofdstuk zullen zien.

$$\begin{array}{lll} \mathfrak{A} \models \exists c : (Succ(c) \Rightarrow Pass) & \text{asa} & \mathfrak{A} \models \exists c : (\neg Succ(c) \vee Pass) \\ & \text{asa} & \mathfrak{A} \models (\exists c : \neg Succ(c)) \vee Pass \\ & \text{asa} & \mathfrak{A} \models (\neg(\forall c : Succ(c))) \vee Pass) \\ & \text{asa} & \mathfrak{A} \models (\forall c : Succ(c)) \Rightarrow Pass. \end{array}$$

Deze afleiding is correct en laat zien dat de twee logische zinnen equivalent zijn. Dit terwijl de NT zinnen niet equivalent zijn. Dus minstens één van de twee nederlandse zinnen werd verkeerd vertaald.

De fout zit in de tweede zin. De nederlandse zin "Er is een vak c zodat, als je slaagt ervoor, dan behaal je de bachelor" zegt: er bestaat een vak c zodat in alle mogelijke situaties s: als men in situatie s slaagt voor het vak c, dan behaalt men de bachelor.

Het probleem met deze zin is dat hij een impliciete kwantificatie over alle mogelijke situaties bevat. Dergelijke impliciete kwantoren kunnen niet uitgedrukt worden in predicatenlogica. In predikatenlogica komt situaties overeen met structuren $\mathfrak{A}$. Zoals de waarheidsdefinitie duidelijk laat zien, worden alle uitspraken van predicatenlogica geëvalueerd in dezelfde $\mathfrak{A}$. Daarom kan deze NT zin niet uitgedrukt worden in predikatenlogica.

2.10 Isomorfisme

Een structuur dient als wiskundige voorstelling van een stand van zaken. Dit is heel duidelijk bij een databank, bv. de studentendatabank $\mathfrak{A}$.

Het blijkt echter dat dezelfde stand van zaken bijna altijd door veel verschillende structuren kan voorgesteld worden, namelijk als ze dezelfde interne vorm hebben. Dat is als twee structuren identiek zijn, op een hernoeming van de domeinelementen na. Zo'n structuren worden isomorf genoemd. Isomorfe structuren zijn voorstellingen van dezelfde stand van zaken.

Voorbeeld 2.10.1. De volgende structuren $\mathfrak{A}$ en $\mathfrak{B}$ voor het vocabularium $\Sigma = \{N/0 :, P/1\}$:

- $D_{\mathfrak{A}} = \{1, 2, 3\}$
  $N^{\mathfrak{A}} = 1$
  $P^{\mathfrak{A}} = \{2, 3\}$.
- $D_{\mathfrak{B}} = \{a, b, c\}$
  $N^{\mathfrak{B}} = a$
  $P^{\mathfrak{B}} = \{b, c\}$.

Als we in de eerste structuur $\mathfrak{A}$ de 3 objecten hernoemen van $1, 2, 3$ tot a, b, c, dan bekomen we $\mathfrak{B}$. Men kan zeggen: $\mathfrak{A}$ en $\mathfrak{B}$ hebben dezelfde inwendige structuur, ze stellen dezelfde stand van zaken voor; ze verschillen enkel door naamgeving van de domeinelementen.

Definitie 2.10.1. Zijn $\mathfrak{A}, \mathfrak{B}$ twee structuren die hetzelfde vocabularium interpreteren. We noemen b een **isomorfisme** van $\mathfrak{A}$ naar $\mathfrak{B}$ indien b een bijectie is van $D_{\mathfrak{A}}$ naar $D_{\mathfrak{B}}$ zodat:

- als $C^{\mathfrak{A}} = d$ dan is $C^{\mathfrak{B}} = b(d)$, voor elke constante symbool $C \in \Sigma$;
- als $F^{\mathfrak{A}}(d_1, \ldots, d_n) = d$ dan is $F^{\mathfrak{B}}(b(d_1), \ldots, b(d_n)) = b(d)$, voor elk functie symbool $F \in \Sigma$;
- $(d_1, \ldots, d_n) \in P^{\mathfrak{A}}$ asa $(b(d_1), \ldots, b(d_n)) \in P^{\mathfrak{B}}$, voor elk predikaat symbool $P \in \Sigma$.

Definitie 2.10.2. Een structuur $\mathfrak{A}$ is isomorf met een structuur $\mathfrak{B}$ indien er een isomorfisme van $\mathfrak{A}$ naar $\mathfrak{B}$ bestaat. Notatie: $\mathfrak{A} \cong \mathfrak{B}$.

Er is krachtige manier om de definitie van isomorfisme te vereenvoudigen. Elke functie $b : D \to E$ kunnen we "liften" tot een functie $\overline{b}$ die
- koppels $(d_1, \ldots, d_n) \in D^n$ afbeeldt op het overeenkomstige koppel $(b(d_1), \ldots, b(d_n))$ in E^n;
- die relaties $R \subseteq D^n$ afbeeldt op de overeenkomstige relaties $\overline{b}(R) = \{\overline{b}(\bar{x}) | \bar{x} \in R\}$ in E;
- die functies $f : D^n \to D$ afbeeldt op functies $\overline{b}(f) : E^n \to E$, zodat $\overline{b}(f) = \{\overline{b}(\bar{x}) | \bar{x} \in f\}$.

Als b een bijectie is van D naar E, dan is $\overline{b}$ een bijectie van de koppels, de relaties en de functies op D naar die op E.

Oefening 2.10.1. *In het bovenstaand voorbeeld geldt voor de functie* $b = \{(1, a), (2, b), (3, c)\}$ *dat* $\overline{b}(N^{\mathfrak{A}}) = \overline{b}(1) = / + a = N^{\mathfrak{B}}$, *en* $\overline{b}(P^{\mathfrak{A}}) = \overline{b}(\{1, 2\}) = \{b, c\} = P^{\mathfrak{B}}$. *Maar wat is* $\overline{b}(\{3, (\{1, \{2\}\}, (2, 3))\})$*?*

Een alternatieve maar equivalente manier om een isomorfisme te definiëren is als volgt.

Definitie 2.10.3. Een functie b is een **isomorfisme** van $\mathfrak{A}$ naar $\mathfrak{B}$ indien $\mathfrak{A}, \mathfrak{B}$ dezelfde symbolen interpreteren, b een bijectie is van $D_{\mathfrak{A}}$ naar $D_{\mathfrak{B}}$, en voor elk geïnterpreteerd symbool σ geldt dat $\overline{b}(\tau^{\mathfrak{A}}) = \tau^{\mathfrak{B}}$.

Voorbeeld 2.10.2. Neem de Herbrand-structuur $\mathfrak{A}$ van de studentendatabank uit Figuur 2.1. Zijn domein $D_{\mathfrak{A}} = \{Ray, \ldots, CS230, \ldots, Jill, \ \ldots, \ AAA, \ldots\}$ bestaat uit 25 elementen. We definiëren nu een nieuwe structuur $\mathfrak{B}$ die isomorf is met $\mathfrak{A}$.

- Definieer $D_{\mathfrak{B}} = \{a, \ldots, y\}$. Dit zijn 25 letters, precies evenveel. Kies een correspondentie, bv. $Ray - 1, Hec - 2, \ldots$. Dit definieert een bijectie $b : D_{\mathfrak{A}} \to D_{\mathfrak{B}}$.
- Voor elk objectsymbool C, definieer $C^{\mathfrak{B}} = b(C)$. Dus, $Ray^{\mathfrak{B}} = a, \ldots$.
- Voor elk predicaatsymbool P van de studentendatabank, neem de tabel van P en vervang elk objectsymbool C door de letter $b(C)$. Definieer $P^{\mathfrak{B}}$ als de relatie die met de resulterende tabel overeenkomt.

Dan is b een isomorfisme van $\mathfrak{A}$ naar $\mathfrak{B}$.

Stelling 2.10.1. *In isomorfe structuren zijn dezelfde formules waar. Formeel, indien $\mathfrak{A} \cong \mathfrak{B}$, dan geldt voor elke geïnterpreteerd formule A dat $\mathfrak{A} \models A$ asa $\mathfrak{B} \models A$.*

Bew. We geven een schets van het bewijs.

Het bewijs is per inductie. Maar zoals vaak gebeurt in bewijzen van inductie, moeten we een algemener stelling bewijzen, een stelling waaruit het te bewijzen zal volgen.

We hadden al een isomorfisme b uitgebreid tot de functie $\overline{b}$ die alle waardes bewaart en voegen er nog aan toe : $\overline{b}(\mathbf{t}) = \mathbf{t}$, en $\overline{b}(\mathbf{f}) = \mathbf{f}$. Wat we aantonen: dat voor elke expressie e (term of formule), voor elke isomorfisme $b : \mathfrak{A} \to \mathfrak{B}$ waarbij $\mathfrak{A}, \mathfrak{B}$ de expressie e interpreteren, geldt: $\overline{b}(e^{\mathfrak{A}}) = e^{\mathfrak{B}}$. Als e een formule is, dan zijn $e^{\mathfrak{A}}$ en $e^{\mathfrak{B}}$ de waarheidswaardes van e in $\mathfrak{A}$ en $\mathfrak{B}$ en er geldt: $e^{\mathfrak{A}} = \overline{b}(e^{\mathfrak{A}}) = e^{\mathfrak{B}}$. Hieruit volgt het te bewijzen: als e een formule is dan geldt $\mathfrak{A} \models e$ asa $\mathfrak{B} \models e$.

Het bewijs is per inductie op het aantal symbolen van e (of op de structuur van e).

Basisgeval: stel dat e een object symbool of propositioneel symbool is, en b een isomorfisme van $\mathfrak{A}$ naar $\mathfrak{B}$. Omdat b een isomorfisme is, geldt $\overline{b}(e^{\mathfrak{A}}) = e^{\mathfrak{B}}$.

Inductiegeval: stel dat e een samengestelde term of formule is, en b een isomorfisme van $\mathfrak{A}$ naar $\mathfrak{B}$. Stel dat voor alle componenten e' van e geldt dat voor alle isomorfiën $b' : \mathfrak{A}' \to \mathfrak{B}'$ geldt dat $\overline{b'}(e'^{\mathfrak{A}'}) = e'^{\mathfrak{B}'}$. Dan moeten we bewijzen dat $\overline{b}(e^{\mathfrak{A}}) = e^{\mathfrak{B}}$. De inductie-stap is door een gevalsanalyse op het topsymbool van e. We bekijken maar twee gevallen. De rest moet je zelf doen.

Zij e een term $f(t_1, \ldots, t_n)$ en $b : \mathfrak{A} \to \mathfrak{B}$ een isomorfisme zodat $\mathfrak{A}, \mathfrak{B}$ e interpreteren. Er geldt dus $\overline{b}(f^{\mathfrak{A}}) = f^{\mathfrak{B}}$. De structuren $\mathfrak{A}, \mathfrak{B}$ interpreteren ook alle subtermen $t_1, \ldots, t_n$, en dus volgt uit de inductiehypothese dat $\overline{b}(t_i{}^{\mathfrak{A}}) = t_i{}^{\mathfrak{B}}$ voor elke $i \in [1, n]$. Hieruit volgt:

$$\begin{array}{lll}
\overline{b}(f(t_1,\ldots,t_n)^{\mathfrak{A}}) & = \overline{b}(f^{\mathfrak{A}}({t_1}^{\mathfrak{A}},\ldots,{t_n}^{\mathfrak{A}})) & \\
& = f^{\mathfrak{B}}(\overline{b}({t_1}^{\mathfrak{A}}),\ldots,\overline{b}({t_n}^{\mathfrak{A}})) & (b \text{ is isomorfisme}) \\
& = f^{\mathfrak{B}}({t_1}^{\mathfrak{B}},\ldots,{t_n}^{\mathfrak{B}}) & (\text{inductiehypothese}) \\
& = e^{\mathfrak{B}}. &
\end{array}$$

Zij e een zin $\forall x : A$, en $b : \mathfrak{A} \to \mathfrak{B}$ een isomorfisme. Het is gemakkelijk in te zien dat voor elke $d \in D_{\mathfrak{A}}$, b een isomorfisme is van $\mathfrak{A}[x : d]$ naar $\mathfrak{B}[x : b(d)]$. Door de inductiehypothese geldt $A^{\mathfrak{A}[x:d]} = \overline{b}(A^{\mathfrak{A}[x:d]}) = A^{\mathfrak{B}[x:b(d)]}$. Dan volgt:

$\mathfrak{A} \models \forall x : A$
asa voor alle $d \in D_{\mathfrak{A}}$, $\mathfrak{A}[x : d] \models A$ (ontvouwen met $\forall$-regel van de waarheidsdefinitie)
asa voor alle $d \in D_{\mathfrak{A}}$, $\mathfrak{B}[x : b(d)] \models A$ (inductiehypothese))
asa voor alle $u \in D_{\mathfrak{B}}$, $\mathfrak{B}[x : u] \models A$ (b een bijectie)
asa $\mathfrak{B} \models \forall x : A$ (vouwen met $\forall$-regel van de waarheidsdefinitie).

De cruciale stap in het bewijs is de derde, waar we gebruik maken van het feit dat b een bijectie is: de verzameling $\overline{b}(D_{\mathfrak{A}}) = \{b(d) | d \in D_{\mathfrak{A}}\}$ is $D_{\mathfrak{B}}$.

De andere gevallen zijn analoog.

Q.E.D.

De isomorfie-relatie $\cong$ is een voorbeeld van een equivalentierelatie, namelijk op de klasse van structuren.

Definitie 2.10.4. Een binaire relatie $\sim$ (gebruikt in infix-notatie) in een verzameling D is een equivalentierelatie als het voldoet aan 3 voorwaarden:
- reflexiviteit: $\forall x \in D : x \sim x$
- symmetrie: $\forall x \in D : \forall y \in D : x \sim y \Rightarrow y \sim x$
- transitiviteit: $\forall x, y, z \in D : x \sim y \wedge y \sim z \Rightarrow x \sim z$

De equivalentie-klasse van een element $d \in D$ is de verzameling $\{z \in D | z \sim d\}$. Notatie: $d_{\sim}$.

Stelling 2.10.2. *$\cong$ is een equivalentierelatie op structuren.*

Oefening 2.10.2. *(eenvoudig) Bewijs dit gebruik makend van de volgende hints:*
- *$\cong$ is reflexief: stel voor elke structuur $\mathfrak{A}$ een isomorfisme van $\mathfrak{A}$ naar $\mathfrak{A}$ voor.*
- *$\cong$ is symmetrisch: gegeven een isomorfisme $b : \mathfrak{A} \to \mathfrak{B}$, stel een isomorfisme $b' : \mathfrak{B} \to \mathfrak{A}$ voor.*
- *$\cong$ is symmetrisch: gegeven isomorfismes $b : \mathfrak{A} \to \mathfrak{B}$, $b' : \mathfrak{B} \to \mathfrak{B}'$, stel een isomorfisme van $\mathfrak{A}$ naar $\mathfrak{B}'$ voor.*

De equivalentieklasse van een structuur $\mathfrak{A}$ bevat alle structuren die isomorf zijn met $\mathfrak{A}$.

Hoofdstuk 3

Logisch gevolg en redeneren

3.1 Logisch waar, logisch gevolg

Concepten zoals een gevolg, equivalentie, consistentie, inconsistentie, contradictie: het zijn eigenschappen van of relaties tussen "gedachten", "informaties", "proposities". In logica waar logische zinnen zulke "gedachten" formeel uitdrukken, kunnen we deze concepten formeel definiëren gebruikmakend van het basisconcept van logica: de definitie van waarheid van een zin in een structuur. Op basis van dit concept kunnen een aantal andere fundamenteel belangrijke concepten in verband met informatie *formeel* gedefinieerd worden:

- een propositie kan een *tautologie* zijn, namelijk altijd voldaan in elke toestand van de wereld. Bv. er bestaat een les van dit vak die niet doorgaat in de K06, ofwel gaat de volgende les door in de K06.
- één propositie kan een andere propositie *impliceren* of als *gevolg* hebben. Bv. als alle lessen van dit vak doorgaan in de K06, dan impliceert dit dat de volgende les doorgaat in K06.
- Een propositie kan *inconsistent* zijn, ook *tegenstrijdig* of *contradictorisch* genoemd. Bv. de propositie bestaande uit de voorgaande en dat de volgende les doorgaat in de N04, is inconsistent.
- Twee proposities kunnen logisch equivalent zijn.

Deze vertrouwde maar informele eigenschappen of relaties tussen "informaties" kunnen nu wiskundig geformaliseerd worden op basis van de definitie van waarheid in een structuur.

In Definitie 2.3.8 werden deze concepten gedefinieerd. Hieronder bespreken we deze concepten. Ter herhaling:

- Een logische formule A is **logisch waar**, of is een **tautologie** als ze waar is in alle structuren $\mathfrak{A}$ die A interpreteren. We noteren dit in een wiskundige notatie als $\models A$.
- Een logische formule A is **logisch consistent** als ze waar is in minstens één structuur.
- Een logische formule A is **logisch inconsistent** of **logisch tegenstrijdig** of **logisch contradictorisch** als er geen structuur bestaat waarin A waar is.

In het vervolg zullen we vaak het woord "logisch" weglaten van voor deze termen en spreken van gevolg, consistent, inconsistent, etc.

Let op het verschil tussen waarheid en logische waarheid. We kunnen niet van een logische zin zeggen dat hij waar is, tenzij we er een structuur bij geven die deze zin interpreteert. Darentegen is een zin logisch waar indien hij waar is in elke structuur.

Voorbeeld 3.1.1. De zin $\forall x\colon LesCur(x, LvI) \Rightarrow InAud(x, G06)$ is waar in de structuur van onze wereld waarbij $LesCur(x, y)$ betekent dat x een les is van cursus y. Maar deze zin is niet logisch waar.

De zin $[(\forall x\colon LesCur(x, LvI) \Rightarrow InAud(x, G06)) \wedge LesCur(Les4, LvI)] \Rightarrow InAud(Les4, G06)$ is logisch waar, want waar in elke wereld, zelfs in die waar alle lessen van LvI doorgaan in N04, of zelfs waarin $LesCur$ iets totaal anders betekent dan lesgeven e.d., (Bv. waar $LesCur(x, y)$ betekent dat x minder curry bevat dan y).

Oefening 3.1.1. *Geef een andere logische zin A en twee structuren $\mathfrak{A}_1, \mathfrak{A}_2$ die A interpreteren zodatA waar is in $\mathfrak{A}_1$ en onwaar in $\mathfrak{A}_2$. Is A logisch waar? Wat denk je van de uitspraak: A is logisch waar in $\mathfrak{A}_1$. (Hint: soms is "nonsens" een correct antwoord).*

De drie eigenschappen in de vorige definitie zijn natuurlijk sterk gecorreleerd.

Propositie 3.1.1. *Een formule A is logisch waar asa $\neg A$ logisch inconsistent is asa $\neg A$ niet logisch consistent is.*

We bewijzen deze eigenschap door een lid te herschrijven op equivalentie-bewarende manier, o.a. door het ontvouwen/vouwen van de definities van deze termen.

Bew. Een zin A is logisch inconsistent asa (ontvouw)
er is geen structuur waarin A waar is asa
het is niet waar dat er een structuur is waarin A waar is asa (vouw)
A is niet logisch consistent.

Een zin A is logisch waar asa
A is waar in elke structuur die A interpreteert asa
$\neg A$ is niet waar in elke structuur die A interpreteert asa
er is geen structuur waarin $\neg A$ waar is asa
$\neg A$ is logisch inconsistent asa
$\neg A$ is niet logisch consistent. Q.E.D.

Bewijzen van logische waarheid vereist dat aangetoond wordt dat een zin waar is in *alle* structuren. Op het eerste gezicht zou men misschien denken dat dit onbegonnen werk is, want er zijn er zo oneindig veel, en bovendien zijn veel structuren zelf oneindig groot! Maar toch is dat vaak heel eenvoudig.

Voorbeeld 3.1.2. Nemen we een Nederlandse zin die evident waar is: "*Als alle lessen van dit vak plaats vinden in lokaal R1 of R2, en de volgende les van dit vak gaat niet door in lokaal R1, dan gaat ze door in lokaal R2* ". Deze zin is een instantiatie van een logische zin die logisch waar is:

T.B. $[(\forall x\colon P(x, R1) \vee P(x, R2)) \wedge \neg P(C, R1)] \Rightarrow P(C, R2)$ is logisch waar.

We geven twee bewijzen, om twee vaak voorkomende bewijstechnieken te illustreren: bewijs uit het ongerijmde en bewijs door gevallenstudie. We noteren bovenstaande zin als A.

Bew. (door gevallenstudie) We bewijzen dat de uitspraak waar is in elke structuur van het vocabularium. Voor elke structuur $\mathfrak{A}$ geldt één van de volgende gevallen:

- Ofwel geldt $\mathfrak{A} \models P(C, R2)$ (lees: $P(C, R2)$ is waar in $\mathfrak{A}$). Dus voldoet $\mathfrak{A}$ aan de conclusie van A, en uit de $\Rightarrow$-regel volgt dat A waar is in $\mathfrak{A}$.
- Ofwel geldt $\mathfrak{A} \not\models P(C, R2)$. Nu kunnen we opnieuw een gevallenonderscheid maken:
  - Ofwel geldt $\mathfrak{A} \models P(C, R1)$. In dit geval is de tweede conjunct van de premisse onwaar, en dus is de premisse zelf ook onwaar. Uit de $\Rightarrow$-regel volgt dan dat de hele zin waar is in $\mathfrak{A}$.
  - Ofwel geldt $\mathfrak{A} \not\models P(C, R1)$. In dit geval geldt voor $d = C^{\mathfrak{A}}$ dat $\mathfrak{A}[x : d] \not\models P(x, R1) \vee P(x, R2)$, en dus $\mathfrak{A} \not\models \forall x \colon (P(x, R1) \vee P(x, R2))$ ($\forall$-regel). Dus is de premisse zelf ook niet voldaan ($\wedge$-regel). Ook nu volgt $\mathfrak{A} \models A$.

Er zijn geen andere gevallen meer, dus is het bewijs geleverd. Q.E.D.

Merk op dat we hierboven de oneindige klasse van alle structuren van het vocabularium van A hebben onderverdeeld in drie oneindige deelklassen: degene waarin $P(C, R2)$ waar is, degene waarin $P(C, R2)$ onwaar en $P(C, R1)$ waar is, en ten slotte, degene waarin $P(C, R2)$ en $P(C, R1)$ beide onwaar zijn. Elke structuur behoort tot exact één van deze subklassen. Hoewel alle structuren in een dergelijke deelklasse maar heel weinig met elkaar gemeen hebben, weten we er toch genoeg over om te kunnen beslissen dat in elk de formule voldaan is. Zo kunnen we dus afleiden dat de formule voldaan is in elk van die oneindige klassen van structuren.

Bew. (uit het ongerijmde) Veronderstel dat deze zin A niet logisch waar is. Dan bestaat er een structuur $\mathfrak{A}$ waarin de zin onwaar is. Het volgt uit de $\Rightarrow$-regel dat daarin de premisse waar en de conclusie onwaar is. Als we de premisse met de $\wedge$-regel en de $\neg$-regel vereenvoudigen, krijgen we:

$$(1)\ \mathfrak{A} \models \forall x \colon P(x, R1) \vee P(x, R2) \quad \text{en} \quad (2)\ \mathfrak{A} \not\models P(C, R1) \quad \text{en} \quad (3)\ \mathfrak{A} \not\models P(C, R2)$$

Uit de $\forall$-regel volgt dat voor elk element d uit het domein van $\mathfrak{A}$ het volgende geldt: $\mathfrak{A}[x : d] \models P(x, R1) \vee P(x, R2)$. Bijgevolg geldt ook $\mathfrak{A}[x : C^{\mathfrak{A}}] \models P(x, R1) \vee P(x, R2)$. Volgens de $\vee$-regel volgt dan dat er twee mogelijke gevallen zijn:

- Ofwel geldt $\mathfrak{A}[x : C^{\mathfrak{A}}] \models P(x, R1)$, wat in tegenspraak is met (2).
- Ofwel geldt $\mathfrak{A}[x : C^{\mathfrak{A}}] \models P(x, R2)$, wat in tegenspraak is met (3).

Er zijn geen andere mogelijkheden, dus bekomen we een contradictie. Q.E.D.

Opmerking 3.1.1. Het bewijs door gevallenonderscheid is een rechtstreeks bewijs: het werkt door de oneindige klasse van structuren op te splitsen in een beperkt aantal nog steeds oneindige deelklassen, zodat je voor elke deelklasse toch voldoende informatie hebt om te kunnen aantonen dat A waar is in elk element ervan. Cruciaal hier is hoe die subklassen te bepalen.

Het bewijs uit het ongerijmde lijkt anderzijds wel eleganter en doelgerichter dan een bewijs door gevallenonderscheid, ook al omdat je mag redeneren over één structuur. In het vervolg

kiezen we dus deze manier van bewijzen. M.a.w., om te bewijzen dat een zin A logisch waar is, veronderstel dat er een structuur $\mathfrak{A}$ bestaat zodat $\mathfrak{A} \not\models A$ en leidt daaruit een contradictie af.

Overigens, elk bewijs uit het ongerijmde kan geherformuleerd worden als een bewijs door gevallenonderscheid en omgekeerd. Er is dus geen verschil in *bewijskracht* tussen deze methodes.

Analoog kun je bewijzen dat een formule A contradictorisch is door een gevallenonderscheid te maken en zo aan te tonen dat A onwaar is in elke structuur, maar een bewijs uit het ongerijmde is meestal gemakkelijker.

Tenslotte, de enige manier om te bewijzen dat een formule consistent is, is een model te construeren, t.t.z. een structuur waarin de formule waar is. Hoe dit te doen bespraken we al in het vorig hoofdstuk.

Oefening 3.1.2. *De formule* $\forall x_1 \colon \ldots \forall x_n \colon \exists y \colon y = F(x_1, \ldots, x_n)$ *is logisch waar. Om dit te bewijzen moet je gebruik maken van het feit in elke structuur* $F^{\mathfrak{A}}$ *een totale functie is en dus dat voor elk koppel* $(d_1, \ldots, d_n)$, $F^{\mathfrak{A}}(d_1, \ldots, d_n)$ *een bestaand element is van het domein.*

We kunnen in principe dus geen partiële functie modelleren met een functiesymbool.

Voorbeeld 3.1.3. De zin $\exists x \colon \forall y \colon y = x$ is niet logisch waar. Inderdaad, deze zin is waar in een structuur $\mathfrak{A}$ asa het universum van $\mathfrak{A}$ slechts één element bevat. Maar dus is deze zin wel consistent.

Dus ook de zin $\neg\exists x \colon \forall y \colon y = x$ is niet logisch waar maar wel consistent want het drukt uit dat het domein minstens 2 elementen bevat.

Dit laatste voorbeeld laat zien dat $\not\models A$ niet impliceert dat $\models \neg A$. Merk op dat daarentegen $\mathfrak{A} \not\models A$ wel impliceert dat $\mathfrak{A} \models \neg A$. Dit volgt namelijk uit de $\neg$-regel. Dus, logische waarheid en gewone waarheid hebben andere eigenschappen!

De volgende definitie herhaalt een belangrijk semantisch concept van Definitie 2.3.8:

Definitie 3.1.1. Een logische formule A is een **logisch gevolg** van B als in elke structuur $\mathfrak{A}$ die beide formules interpreteert en waarin B waar is, ook A waar is. De wiskundige notatie om dit aan te geven is $B \models A$.

Of in wiskundige notatie, A is een logisch gevolg van B indien voor alle structuren $\mathfrak{A}$ die A en B te interpreteren en waarin $\mathfrak{A} \models B$ geldt dat $\mathfrak{A} \models A$. Wanneer $B \models A$ dan zeggen we ook wel: B **impliceert** A.

Opmerking 3.1.2. Het wiskundige teken $\models$ wordt ondertussen al in drie verschillende betekenissen gebruikt wordt:

- $\mathfrak{A} \models A$ betekent "formule A is waar in structuur $\mathfrak{A}$".
- $\models A$ betekent "formule A is logisch waar".
- $B \models A$ betekent "formule A is een logisch gevolg van formule B".

Logici zijn zuinig met hun tekens! De betekenis van het symbool moet je halen uit de context. Wees zorgvuldig.

Propositie 3.1.2.

(a) $B \models A$ *asa* $\models B \Rightarrow A$. *In woorden,* A *is een logisch gevolg van* B *asa de zin* $B \Rightarrow A$ *logisch waar is.*

(b) A *en* B *zijn logisch equivalent asa* $A \Leftrightarrow B$ *logisch waar is.*

Bew. We bewijzen enkel (a).

(⇒) Veronderstel dat $B \models A$. Neem een willekeurige structuur $\mathfrak{A}$ die B en A interpreteert. Ofwel geldt $\mathfrak{A} \not\models B$, en dan volgt uit de ⇒-regel dat $\mathfrak{A} \models B \Rightarrow A$. Ofwel geldt $\mathfrak{A} \models B$. Omdat A een logisch gevolg is van B geldt ook $\mathfrak{A} \models A$, en toepassing van de ⇒-regel geeft opnieuw $\mathfrak{A} \models B \Rightarrow A$. Dus geldt $\models B \Rightarrow A$.

(⇐) Veronderstel dat $\models B \Rightarrow A$, dus voor elke $\mathfrak{A}$ geldt $\mathfrak{A} \models B \Rightarrow A$. Uit de ⇒-regel volgt dat als $\mathfrak{A} \models B$ dan $\mathfrak{A} \models A$. Bijgevolg geldt $B \models A$.

Q.E.D.

Definitie 3.1.2.

- Een zin A is **logisch inconsistent** of **logisch tegenstrijdig** of **logisch contradictorisch met** B indien A onwaar is in elk model van B dat ook A interpreteert.
- Een zin A is **logisch consistent met** zin B indien A voldaan is in minstens één model van B.
- Een zin A is **logisch equivalent met** zin B indien voor elke $\mathfrak{A}$ die beide zinnen interpreteert, $A^{\mathfrak{A}} = B^{\mathfrak{A}}$ (ze hebben dezelfde waarheidswaarde, beide zijn waar of beide zijn onwaar).

Dezelfde termen worden hier opnieuw gebruikt in een andere betekenis dan voorheen, namelijk om relaties tussen twee zinnen aan te duiden. Er is dan ook een sterk verband tussen bv. het logisch inconsistent zijn van een formule, en het logisch inconsistent zijn van een formule met een andere formule, zoals de volgende propositie aantoont.

Propositie 3.1.3. *A is logisch inconsistent met B asa $A \wedge B$ is logisch inconsistent. A is logisch consistent met B asa $A \wedge B$ is logisch consistent.*

Oefening 3.1.3. *Bewijs deze propositie. (Eenvoudig!)*

Opmerking 3.1.3. Een theorie $T = \{A_1, \ldots A_n\}$ betekent hetzelfde als de fomule $A_1 \wedge \cdots \wedge A_n$. Bijgevolg kunnen we elk van de definities hierboven (gevolg, consistent, inconsistent) op evidente manier uitbreiden tot theorieën:

- een logische theorie T is logisch waar als elke zin in T logisch waar is.
- Een logische theorie T is inconsistent als er in elke structuur minstens één zin van T onwaar is.
- Twee logische theorie T, T' zijn consistent met elkaar als er een model bestaat van beide, en inconsistent met elkaar als er geen structuur bestaat die een model is van beide theorieën.

3.2 Propositielogica

Propositielogica is het fragment van predikatenlogica beperkt tot propositionele symbolen $P/0$.

Historisch gezien werd propositielogica ontwikkeld vóór de predikatenlogica, namelijk door Boole (1854). Propositielogica wordt daarom, in de electronica, ook vaak Boolse algebra genoemd. De titel van Boole's werk was *"An Investigation of the Laws of Thought"*, wat illustratief is voor de bedoeling van Boole en andere vroege logici om "gedachten" te bestuderen met wiskundige methodes. Dit is een bedoeling die wellicht weinig electronici nog zoeken in Boolse algebra, maar die wij in deze cursus zeker nastreven omdat we willen leren hoe informatie voor te stellen in logica.

Definitie 3.2.1. Een propositioneel symbool is een 0-voudig predikaatsymbool. Een propositioneel vocabularium Σ is een vocabularium dat enkel bestaat uit propositionele symbolen. Een propositionele formule is een formule opgebouwd uit propositionele symbolen en de logische connectieven $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$.

Een propositionele formule bevat dus geen kwantoren of variabelen of $=$ en is bijgevolg een logische zin. Een voorbeeld is $P \wedge (P \Rightarrow Q)$. Bv. veronderstel dat de volgende betekenis toekennen aan P, Q, R: P wordt geïnterpreteerd als "De trein is laat", Q als "John mist zijn bus", dan zegt de zin $P \wedge (P \Rightarrow Q)$ dat de trein te laat is en dat als de trein te laat is dat John dan zijn bus mist.

Een propositioneel symbool is een 0-voudig relatiesymbool. Zijn waarde in een structuur is dus een 0-voudige relatie. Maar wat is een 0-voudige relatie? Een verzameling van 0-voudige koppels? Er is maar één 0-voudig koppel, namelijk $()$, en dus zijn er maar twee 0-voudige relaties: de lege verzameling $\emptyset$ en $\{()\}$. De lege verzameling $\emptyset$ kunnen we identificeren met $\mathbf{f}$ en $\{()\}$ met $\mathbf{t}$. De waarde van een propositioneel symbool in een structuur is dus een element van $\mathbb{B} = \{\mathbf{t}, \mathbf{f}\}$.

In de context van een propositioneel vocabularium en theorie zijn gekwantificeerde zinnen zinloos. Ook is het domein van een structuur volkomen irrelevant. We kunnen bijgevolg de notie van structuur vereenvoudigen, zoals in de volgende definitie gebeurt.

Definitie 3.2.2. Een structuur van een propositioneel vocabularium Σ is een booleaanse functie van Σ naar $\mathbb{B} = \{\mathbf{t}, \mathbf{f}\}$. Voor $P \in \Sigma$ zullen we $\mathfrak{A}(P)$ blijven noteren als $P^{\mathfrak{A}}$.

Als het duidelijk is wat Σ is wordt een structuur van Σ ook wel genoteerd als de verzameling

van ware symbolen. Dus voor $\Sigma = \{P, Q\}$ denoteert $\{P\}$ de structuur waarin P waar is en Q onwaar.

Opmerking 3.2.1. Het is echt van belang dat Σ vastligt als je die notatie gebruikt, anders is ze ambigu. Bv. $\{P\}$ noteert telkens een verschillende structuur indien $\Sigma = \{P\}$, $\Sigma = \{P, Q\}$, $\Sigma = \{P, Q, R\}$, etc.

Definitie 2.3.6 van waarheid blijft (bijna) onveranderd gelden, maar de regels voor $=$, $\exists$ en $\forall$ zijn overbodig. Van de resterende regels hoeft enkel de atoom-regel een minimale aanpassing. Deze wordt: $\mathfrak{A} \models P$ asa $P^{\mathfrak{A}} = \mathbf{t}$. De concepten van logische waarheid, logisch gevolg, logisch consistent, logisch inconsistent blijven onveranderd gelden.

Voorbeeld 3.2.1 (Een logische puzzel)**.** *Er zijn 3 dozen 1, 2 en 3. Eén ervan bevat goud, de anderen zijn leeg. Er zijn 3 boodschappen. Eén is waar, de anderen zijn niet waar.*

- *Boodschap 1: Het goud zit niet in doos 1*
- *Boodschap 2: Het goud zit niet in doos 2*
- *Boodschap 3: Het goud zit in doos 2*

Waar is het goud?

We maken een theorie om deze informatie uit te drukken.

- We maken een propositioneel vocabularium Σ:
  - H_i: "boodschap i is waar"
  - G_i: "het goud zit in doos i"
- Dat er maar één boodschap waar is, drukken we als volgt uit:

$$(H_1 \wedge \neg H_2 \wedge \neg H_3) \vee (\neg H_1 \wedge H_2 \wedge \neg H_3) \vee (\neg H_1 \wedge \neg H_2 \wedge H_3)$$

- Dat er in juist één doos goud zit, drukken we als volgt uit:

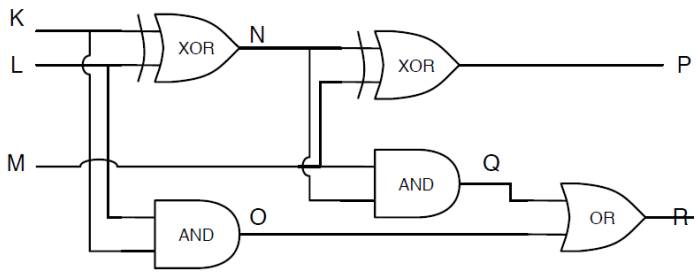
$$(G_1 \wedge \neg G_2 \wedge \neg G_3) \vee (\neg G_1 \wedge G_2 \wedge \neg G_3) \vee (\neg G_1 \wedge \neg G_2 \wedge G_3)$$

- De betekenis van de boodschappen: $H_1 \Leftrightarrow \neg G_1$, $H_2 \Leftrightarrow \neg G_2$, $H_3 \Leftrightarrow G_2$.

Nu we een theorie hebben kunnen we ons de vraag stellen hoe we deze kunnen gebruiken om de puzzel op te lossen. De oplossing (of eventueel meerdere oplossingen) zit verscholen in de modellen van deze theorie. Elk model $\mathfrak{A}$ is een situatie die consistent is met de puzzel, en zal één doos aanwijzen waar het goud in zit, namelijk als $G_i^{\mathfrak{A}}$ waar is. Er blijkt echter maar één model te zijn.

Bijgevolg, als we deze puzzel willen laten oplossen door een computer, dan hebben we een tool nodig dat het volgende berekent:

> Inferentie-probleem:
> Input: een (propositionele) theorie T.
> Output: een model van T



Figuur 3.1: Een digitaal schakelnetwerk

Dit is een probleem met veel toepassingen in de informatica (zie verderop).

Oefening 3.2.1. *Wat zijn de modellen en de corresponderende oplossingen? Je kunt de puzzel oplossen door een* waarheidstabel *op te stellen. (Zie verderop).*

Schakelnetwerken. Een electronisch schakelnetwerk bestaat uit poorten en schakelaars. Zo'n netwerk heeft een aantal invoerpoorten waar een spanning 0 of 1 (of **f** of **t**) kan aangelegd worden en een aantal uitvoerpoorten. Tussenin liggen de schakelaars die logische operaties toepassen op hun input.

Een voorbeeld van een booleaans circuit is te vinden in Figuur 3.1. In dit netwerk zijn K, L, M invoer- en P en R uitvoerpoorten. Drie soorten logische operaties worden gebruikt: $\wedge, \vee$ en `XOR` , een nieuwe logische operator met de volgende waarheidstabel:

A	B	A `XOR` B
t	**t**	**f**
t	**f**	**t**
f	**t**	**t**
f	**f**	**f**

Het is makkelijk in te zien dat A `XOR` B logisch equivalent is met $(A \vee B) \wedge \neg(A \wedge B)$. Dit wordt ook wel de exclusieve disjunctie genoemd, integenstelling tot $\vee$ die de inclusieve disjunctie genoemd wordt.

De uitvoerpoort P komt overeen met de formule die je bekomt door vanuit P alle paden te volgen tot aan de invoerpoorten: dit geeft $(K$ `XOR` $L)$ `XOR` M. Het signaal aan de uitvoerpoort P komt overeen met de waarheidswaarde van deze formule in de structuur $\mathfrak{A}$ van $\{K, L, M\}$ waarbij $K^{\mathfrak{A}}, L^{\mathfrak{A}}, M^{\mathfrak{A}}$ overeenkomt met de invoer die opgelegd wordt aan de invoerpoorten. Op die manier komt elke propositionele formule dus overeen met een schakelnetwerk en omgekeerd.

Bij het ontwerp van electronische schakelnetwerken zijn de volgende vormen van redeneren van cruciaal belang. Veronderstel dat we een bepaalde logische zin willen implementeren met een schakelnetwerk. Elke logisch equivalente zin realiseert dezelfde uitgangspoort, maar met een ander schakelnetwerk. Men heeft hier dus veel keuze. In elk geval kun je zien dat de volgende vorm van redeneren van belang is om na te gaan of een gekozen netwerk de gewenste logische zin realiseert:

Inferentie-probleem:
Input: propositionele formules A en B
Output: $\mathbf{t}$ als $\models A \Leftrightarrow B$, anders A.

Dus, gegeven een gewenste logische formule A, en een voorstel voor een netwerk dat B implementeert, dan bestaat de *verificatie* van het netwerk erin om na te gaan of B logisch equivalent is met A.

Een veel ambitieuzer redeneertaak is de volgende:

Inferentie-probleem:
Input: A
Output: een propositionele formule B die logisch equivalent is aan A en zo weinig mogelijk logische poorten bevat.

Dat betekent dat het digitale netwerk die B realiseert kleiner is dan A maar toch dezelfde uitvoer genereert. Het netwerk van B is kleiner, goedkoper, minder energieverbruikend dan dat van A. Het berekenen van minimale formule is een uiterst complex zoekprobleem - praktisch onoplosbaar voor grote formules.

Waarheid van een formule in een structuur berekenen Het berekenen van de waarheidswaarde van een formule A in structuur $\mathfrak{A}$ is eenvoudig in propositionele logica. Een manier om dit te noteren is geïllustreerd als volgt:

P	Q	(	P	XOR	Q	)	$\Leftrightarrow$	((	P	$\vee$	Q	)	$\wedge$	$\neg$	(	P	$\wedge$	Q	))
1	1		1	0	1		1		1	1	1		0	0		1	1	1	

Onder elk symbool in de formule schrijven we zijn waarde en we vervolledigen door van beneden tot bovenaan de abstracte syntax-boom de waarheidswaarde te berekenen.

Logisch ware formules berekenen met waarheidstabellen. Elke propositionele zin wordt geïnterpreteerd door oneindig veel structuren. Maar elke zin A bevat slechts een eindig aantal niet-logische symbolen Σ_A. Er zijn slechts eindig veel structuren van Σ_A. Of A al dan niet waar is in een structuur $\mathfrak{A}$ hangt enkel af van de waardes van de symbolen van Σ_A. Dit volgt uit de volgende eigenschap.

Eigenschap 3.2.1. Als $\mathfrak{A}$ een zin A interpreteert, en $\mathfrak{B}$ is de structuur bekomen door $\mathfrak{A}$ te beperken tot Σ_A, de niet-logische symbolen van A, dan geldt dat $\mathfrak{A} \models A$ asa $\mathfrak{B} \models A$.

Deze eigenschap volgt onmiddelijk uit Eigenschap 2.3.1.

Dat betekent dat we kunnen berekenen of een propositionele zin A logisch waar of consistent is door gewoon alle structuren van Σ_A op te sommen en daarin de waarheidswaarde van A te berekenen. Een systematische manier om dat te doen is met behulp van een *waarheidstabel*. We illustreren dit met een voorbeeld.

Zij P, Q, R propositionele symbolen. Dan is

$$[(P \vee Q) \wedge R] \Leftrightarrow [(P \wedge R) \vee (Q \wedge R)]$$

een tautologie. We verifiëren dit door de waarheidstabel op te stellen. In deze tabel gebruiken we 0 voor **f** en 1 voor **t**.

P	Q	R	$(P \vee Q)$	$(P \vee Q) \wedge R$	$P \wedge R$	$Q \wedge R$	$(P \wedge R) \vee (Q \wedge R)$	$(P \vee Q) \wedge R \Leftrightarrow (P \wedge R) \vee (Q \wedge R)$
1	1	1	1	1	1	1	1	1
1	1	0	1	0	0	0	0	1
1	0	1	1	1	1	0	1	1
1	0	0	1	0	0	0	0	1
0	1	1	1	1	0	1	1	1
0	1	0	1	0	0	0	0	1
0	0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	0	1

In deze tabel gaan we alle structuren af van de propositionele symbolen van de formule. Strikt genomen hebben we natuurlijk niet aangetoond dat alle structuren die deze zin interpreteren de zin ook waar maken. Immers, een structuur kan ook veel meer symbolen interpreteren dan alleen maar P, Q, R. Maar het volstaat om deze 8 structuren te controleren.

Er is ook een compactere manier om een waarheidstabel op te schrijven die we illustreren met een ander voorbeeld namelijk $(P \texttt{ XOR } Q) \Leftrightarrow ((P \vee Q) \wedge \neg(P \wedge Q))$:

P	Q	$($	P	XOR	Q	$)$	$\Leftrightarrow$	$(($	P	$\vee$	Q	$)$	$\wedge$	$\neg$	$($	P	$\wedge$	Q	$))$
1	1		1	0	1		1		1	1	1		0	0		1	1	1	
1	0		1	1	0		1		1	1	0		1	1		1	0	0	
0	1		0	1	1		1		0	1	1		1	1		0	0	1	
0	0		0	0	0		1		0	0	0		0	1		0	0	0	

Merk op dat onder $\Leftrightarrow$ alleen maar 1 voorkomt, dus is deze formule logisch waar.

We kunnen met waarheidstabellen nog andere essentiële vragen oplossen. We zagen al dat een formule A een logisch gevolg is van B asa $B \Rightarrow A$ logisch waar is; A, B zijn logisch equivalent asa $A \Leftrightarrow B$ logisch waar is; A is inconsistent met B asa $\neg(A \wedge B)$ logisch waar is. Dus het bepalen van logisch gevolg, equivalentie of inconsistentie kunnen we met waarheidstabellen oplossen.

Oefening 3.2.2. *Bewijs dat* $P \texttt{ XOR } Q$ *logisch equivalent is met* $(P \Leftrightarrow \neg Q)$.

Oefening 3.2.3. *In het vorig hoofdstuk werd beweerd dat de formule* $P \wedge (\neg P \Rightarrow Q)$ *equivalent is aan* P. *Bewijs dit met een waarheidstabel.*

Deze methode werkt goed, maar enkel voor kleine formules. Immers als A n propositionele symbolen bevat, dan moeten 2^n structuren nagegaan worden, namelijk voor elke combinatie van 0 of 1 van elk symbool. Dit betekent dat de grootte van de waarheidstabel exponentieel is in het aantal symbolen. Voor een formule met 3 symbolen moet je 8 structuren testen, met 10 symbolen moet je al 1024 structuren testen; voor 100 symbolen is het aantal te testen structuren een getal met 30 cijfers.

Je kunt je indenken dat in de huidige processoren het aantal invoerpoorten veel groter is dan 100. Tegenwoordig voeren computers operaties uit op 64-bit getallen. Alleen al om de som van twee registers te berekenen zijn al 124 invoerbits nodig. Elke invoerbit komt overeen met een propositioneel symbool! Dat betekent dat het onbegonnen werk is om de correctheid van een digitale processor te berekenen met behulp van een waarheidstabel.

Gelukkig bestaan er veel efficiëntere methodes om de logische waarheid of de consistentie van een propositionele zin te berekenen. Dergelijke methodes worden ontwikkeld in het domein van *satisfiability checking*, kortweg SAT. SAT-systemen worden toegepast bij verificatie van digitale

schakelnetwerken, maar ook steeds meer en steeds verder daarbuiten. Het blijkt immers dat veel informaticaproblemen kunnen opgelost worden door ze te reduceren tot een SAT probleem. Bv. niet alleen verificatieproblemen bij verificatie van schakelnetwerken, maar ook puzzels, uurroosters, planning, diagnose, verificatie van software, simulatie, enz. SAT solvers worden ook gebruikt in LogicPalet. Wanneer je vraagt een GEO-world of een Deca-world te construeren waarin een logische zin voldaan is, roept LogicPalet het IDP-systeem op, dat op zijn beurt een SAT solver oproept.

Toch is het zo dat ook SAT solvers voor sommige problemen uiterst inefficiënt zijn. Een openstaande wetenschappelijke vraag is of het wel mogelijk is om efficiënte algoritmes te ontwikkelen om de consistentie of logische waarheid van een booleaanse formule te bewijzen. Dit probleem wordt op dit ogenblik beschouwd als 1 van de 10 belangrijkste wiskundeproblemen! Het probleem heet het P=NP probleem.

Ter illustratie, een amerikaans instituut, het Clay mathematical institute, looft een prijs van **één miljoen dollar** uit voor wie dit probleem kan oplossen! Wie voelt zich geroepen?

Tenslotte zagen we al verschillende belangrijke redeneerproblemen voor propositionele logica: logische waarheid, consistentie, berekenen van minimale equivalente formules. Er zijn nog heel wat andere vormen van redeneren die eveneens veel toepassingen hebben.

Input: propositionele zin A
Output: het aantal modellen van A over Σ_A

Input: een verzameling T van propositionele zinnen
Output: een structuur $\mathfrak{A}$ die aan zoveel mogelijk zinnen van T voldoet.

Er zijn er nog heel wat andere waar we hier niet op ingaan, maar die bv. optreden wanneer we willen bepalen welke fout er in een digitaal netwerk zit, of welke invoerpoorten geen effect kunnen hebben op de uitvoer, enz.

3.3 Wetten van het denken van propositionele logica

Als het niet waar is dat eigenschappen X en Y gelden, betekent dat hetzelfde als dat eigenschap X niet geldt of dat eigenschap Y niet geldt? Als we weten dat niet alle objecten eigenschap X hebben, betekent dat hetzelfde als dat er een object is dat de eigenschap X *niet* heeft? Ons taalgevoel vertelt ons dat dit zo is. Dit waren het soort **wetten van het denken** dat mensen zoals Aristoteles, Leibniz, Boole, De Morgan, Frege wilden onderzoeken.

We zullen nu een aantal fundamentele equivalenties zien die zulke "wetten van het denken" uitdrukken, en die eenvoudig volgen uit de definities van waarheid en logische waarheid. We kunnen deze intuïtieve wetten dus bewijzen!

In deze sectie beginnen we met wetten uit de propositionele logica, dus nog niet van de kwantoren.

Wetten van de connectieven.

Eigenschap 3.3.1 (basiswetten van $\wedge$ en $\vee$)**.** Zij P, Q, R propositionele symbolen, dan zijn de volgende propositionele zinnen tautologieën.

1. $(P \vee Q) \Leftrightarrow (Q \vee P)$ (commutativiteit van $\wedge$ en $\vee$)
   $(P \wedge Q) \Leftrightarrow (Q \wedge P)$
2. $[(P \vee Q) \vee R] \Leftrightarrow [P \vee (Q \vee R)]$ (associativiteit van $\wedge$ en $\vee$)
   $[(P \wedge Q) \wedge R] \Leftrightarrow [P \wedge (Q \wedge R)]$
3. $(P \vee P) \Leftrightarrow P$ (idempotentie)
   $(P \wedge P) \Leftrightarrow P$

Bew. Alle eigenschappen in deze sectie kunnen bewezen worden door een waarheidstabel op te stellen. Q.E.D.

Oefening 3.3.1. *Doe dat.*

Uit de combinatie van commutativiteit, associativiteit en idempotentie van $\wedge$, volgt dat in een grote conjunctie de positie van de haakjes, de volgorde van de symbolen en het aantal voorkomens van een symbool er niet toe doen. We zullen dat vaak gebruiken om een conjunctie neer te schrijven als

$$A_1 \wedge \cdots \wedge A_n$$

waarin geen enkele formule meer dan 1 keer voorkomt. Strikt genomen komt het overeen met:

$$A_1 \wedge (A_2 \wedge \cdots \wedge (A_{n-1}) \wedge A_n) \dots)$$

Maar we kunnen de haakjes ook elders plaatsen. Het maakt niets uit voor de betekenis van de formule.

Hetzelfde geldt voor samengestelde disjuncties, die we vaak schrijven als:

$$A_1 \vee \cdots \vee A_n$$

Opmerking 3.3.1. Over de commutativiteit van $\wedge$. Vergelijk volgende Nederlandse zinnen:

"hij laadde zijn geweer en schoot" en "Hij schoot en laadde zijn geweer"

Ons taalgevoel zegt ons dat in beide de volgorde van de acties verschilt, dus is "en" hier niet commutatief. Dit laat zien dat wij mensen ook het woord "en" verschillend zullen interpreteren al naar gelang de context. Een "en" van gebeurtenissen interpreteren we meestal als dat de eerst vermelde gebeurtenis ook het eerst gebeurt in de tijd, zeker als de beide gebeurtenissen niet gelijktijdig kunnen plaatsvinden. Daarentegen, de "en" in de zin "$\mathfrak{A} \models A \wedge B$ asa $\mathfrak{A} \models A$ en $\mathfrak{A} \models B$" in Definitie 2.3.6 waar we vastleggen wat $\wedge$ betekent, heeft niet die temporele betekenis maar de gewone "logische". Bijgevolg is dat de betekenis van $\wedge$ in alle logische formules.

Ter herinnering, als we schrijven

$$\models A$$

dan wordt bedoeld: A is logisch waar, of A is een tautologie.

Als we schrijven

$$\not\models A$$

dan wordt bedoeld: A is niet logisch waar.

Eigenschap 3.3.2 (distributiviteit van $\wedge$ en $\vee$)**.**

1. $\models [P \wedge (Q \vee R)] \Leftrightarrow [(P \wedge Q) \vee (P \wedge R)]$ (distributiviteit van $\wedge$ t.o.v. $\vee$)
2. $\models [P \vee (Q \wedge R)] \Leftrightarrow [(P \vee Q) \wedge (P \vee R)]$ (distributiviteit van $\vee$ t.o.v. $\wedge$)

De eerste wet laat toe om $\wedge$ door $\vee$ naar binnen in de formule te schuiven, of anders gezegd, om $\vee$ naar buiten de formule te schuiven. De tweede doet hetzeflde voor $\vee$ en $\wedge$.

Eigenschap 3.3.3 (basiswetten van $\neg$)**.**

1. $\models \neg\neg P \Leftrightarrow P$ (dubbele ontkenning)
2. $\models P \vee \neg P$ (uitgesloten derde)
3. $\models\models \neg(P \vee Q) \Leftrightarrow (\neg P \wedge \neg Q)$ (wetten van De Morgan[1])
   $\models \neg(P \wedge Q) \Leftrightarrow (\neg P \vee \neg Q)$
4. $\models \neg(P \Rightarrow Q) \Leftrightarrow (P \wedge \neg Q)$ (negatie van de implicatie)
5. $\models \neg(P \Leftrightarrow Q) \Leftrightarrow (P \Leftrightarrow \neg Q)$ (negatie van de equivalentie)

Deze eigenschappen laten toe om $\neg$ door $\wedge, \vee, \Rightarrow, \Leftrightarrow$ te schuiven, of juist omgekeerd, $\neg$ naar buiten te schuiven.

Eigenschap 3.3.4 (De betekenis van $\Rightarrow, \Leftrightarrow$)**.** Zij P, Q propositionele symbolen, dan zijn de volgende propositionele zinnen tautologieën.

1. $(P \Rightarrow Q) \Leftrightarrow (\neg P \vee Q)$ ($\Rightarrow$ in termen van $\neg, \vee$)
2. $(P \Leftrightarrow Q) \Leftrightarrow [(P \Rightarrow Q) \wedge (Q \Rightarrow P)]$ ($\Leftrightarrow$ in termen van implicaties)
3. $(P \Leftrightarrow Q) \Leftrightarrow [(P \wedge Q) \vee (\neg P \wedge \neg Q)]$ ($\Leftrightarrow$ in termen van $\wedge$ en $\vee$)
4. $(P \Leftrightarrow Q) \Leftrightarrow [(P \vee \neg Q) \wedge (\neg P \vee Q)]$ ($\Leftrightarrow$ in termen van $\wedge$ en $\vee$)

Eigenschap 3.3.5 (Contrapositie)**.**

1. $\models (P \Rightarrow Q) \Leftrightarrow (\neg Q \Rightarrow \neg P)$ (contrapositie bij $\Rightarrow$)
2. $\models (P \Leftrightarrow Q) \Leftrightarrow (\neg P \Leftrightarrow \neg Q)$ (contrapositie bij $\Leftrightarrow$)

Als "als A dan B" een materiële implicatie is, dan is het dus hetzelfde te zeggen dat "als A dan B" of dat "als niet B dan niet A". Bv. stel twee studenten Jan en Kevin. Ik stel vast dat altijd als Jan in de les zit, ook Kevin in de les zit. Dat is precies hetzelfde als vast te stellen dat als Kevin niet in de les zit, dan zit ook Jan niet in de les. Ga maar na.

Opmerking 3.3.2. Zoals gezegd heeft "als ... dan..." veel betekenissen.

In andere betekenissen dan de materiele implicatie, geldt contrapositie niet. Bv. laat ons een causale "als .. dan.." nemen. Nu interpreteren we "als Jan in de les zit, zit ook Kevin in de les" als dat Jans komst naar de les veroorzaakt dat ook Kevin komt (bv. Kevin houdt rekening met Jan's komst). De causale contrapositie zegt dat Kevin's wegblijven een oorzaak is dat Jan niet komt. Hier is het dus Kevin die rekening houdt met Jan's komst. Dat is iets anders.

Of vergelijk volgende Nederlandse zinnen:

[1] De Morgan was een Engelse logicus die zijn resultaten ongeveer tergelijkertijd als Boole publiceerde (1847) maar niet in een wiskundige vorm zoals Boole.

"Als het morgen mooi weer is dan ga ik naar zee" en "Als ik morgen niet naar zee ga, dan is het geen goed weer."

Zijn ze equivalent? Volgens mijn taalgevoel niet. In de eerste zin gebruik ik een betekenis van "als ... dan ... " die we nog niet eerder gezien hebben: namelijk, het drukt uit dat ik de *intentie* heb om, als het morgen mooi weer is, naar zee te gaan. De tweede zin, zijn contrapositie, drukt geen intentie uit: ik heb niet de intentie om, als ik morgen niet aan zee zit, het geen goed weer te laten zijn.

Het is een kenmerk van "echte" materiele implicaties dat de contrapositie ervan wel geldig is. Vergelijk "als alle lessen door gaan in K06, dan gaat de volgende les door in K06" met "als de volgende les niet doorgaat in K06, dan gaan niet alle lessen door in K06". Deze zinnen zijn onmiskenbaar equivalent. Daar kan niemand onderuit.

$\Rightarrow$ en $\Leftrightarrow$ zijn transitief, en $\Leftrightarrow$ is commutatief maar $\Rightarrow$ natuurlijk niet.

Eigenschap 3.3.6 (Eigenschappen van $\Rightarrow, \Leftrightarrow$)**.**

1. $\models [(P \Rightarrow Q) \wedge (Q \Rightarrow R)] \Rightarrow (P \Rightarrow R)$ (transitiviteit van $\Rightarrow$)
2. $\models (P \Leftrightarrow Q) \Leftrightarrow (Q \Leftrightarrow P)$ (commutativiteit $\Leftrightarrow$)
3. $\models [(P \Leftrightarrow Q) \wedge (Q \Leftrightarrow R)] \Rightarrow (P \Leftrightarrow R)$ (transitiviteit van $\Leftrightarrow$)
4. $\models [P \Rightarrow (Q \Rightarrow R)] \Leftrightarrow [(P \wedge Q) \Rightarrow R]$
   $\models [P \Rightarrow (Q \Rightarrow (R \Rightarrow S))] \Leftrightarrow [(P \wedge Q \wedge R) \Rightarrow S]$
   enzovoort.

Oefening 3.3.2. *Zijn $\Rightarrow$ en $\Leftrightarrow$ associatief? Ga na of geef een tegenvoorbeeld.*

Eigenschap 3.3.7 (Bewijsprincipes)**.**

1. $\models (P \wedge \neg P) \Rightarrow Q$ (uit een contradictie volgt alles)
2. $\models [\neg P \Rightarrow (Q \wedge \neg Q)] \Rightarrow P$ (bewijs uit het ongerijmde)
3. $\models P \Leftrightarrow [(Q \Rightarrow P) \wedge (\neg Q \Rightarrow P)]$ (bewijs door gevallenonderscheid)

Waarom noemen we dit bewijsprincipes? Omdat ze een schema vormen voor hoe bepaalde eigenschappen te bewijzen. In voorbeeld 3.1.2 bewezen we voor de daar gespecifieerde formule A dat $\mathfrak{A} \models P(C, R2)$ impliceert dat $\mathfrak{A} \models A$ maar dat ook $\mathfrak{A} \not\models P(C, R2)$ impliceert dat $\mathfrak{A} \models A$. Dat bewijs kun je zien als een toepassing van de hierboven vermelde regel van bewijs door gevallenonderscheid. Ook het tweede gedeelte van het bewijs, namelijk dat $\mathfrak{A} \not\models P(C, R2)$ impliceert dat $\mathfrak{A} \models A$ werd bewezen door een tweede toepassing van hetzelfde principe.

Bew. van Eigenschappen 3.3.1-3.3.7. Elk van de equivalenties in deze eigenschappen kan gemakkelijk bewezen worden met waarheidstabellen.

Oefening 3.3.3. *Elk van deze equivalenties kan gemakkelijk bewezen worden met waarheidstabellen. Pik er een paar uit en verifieer.*

Oefening 3.3.4. *Sta bij elk van deze equivalenties even stil. Elk ervan drukt een eigenschap uit van Nederlandse voegwoorden "en", "of", en zinnen "als ... dan ... " en "... als en slechts als ... ". Vraag je telkens af of je het ermee eens bent.*

3.3.1 Normaalvormen van propositionele logica

Vele logische tools vereisen dat hun invoerformules in een speciale vorm staan. Dit noemt men normaalvormen.

Definitie 3.3.1.

Een **literal** is een formule van de vorm $P(t_1, \ldots, t_n)$ of $\neg P(t_1, \ldots, t_n)$.

Als $L_1, \ldots, L_n$ literals zijn, dan noemen we een formule $L_1 \wedge \cdots \wedge L_n$ een simpele conjunctie en $L_1 \vee \cdots \vee L_n$ een simpele disjunctie. Een simpele disjunctie wordt ook een **clause** genoemd.

Een formule is in **disjunctieve normaal vorm** (DNF) indien het een disjunctie is van simpele conjuncties.

Een formule is in **conjunctieve normaal vorm** (CNF) indien het een conjunctie is van simpele disjuncties. Het is dus een conjunctie van clauses.

Een formule is in **negatie-normaalvorm** als $\wedge, \vee$ en $\neg$ de enige connectieven zijn en als het negatiesymbool $\neg$ enkel in literals voorkomt.

Merk op dat formules in disjunctieve of conjunctieve normaalvorm kwantor-vrij zijn.

Zoals we zagen in de vorige sectie laten de wetten van connectieven en kwantoren toe negatie naar binnen te schuiven zodat een negatie-normaalvorm bekomen wordt..

De volgende eigenschap toont het nut aan van deze normaalvormen.

Propositie 3.3.1. *Een propositionele zin in disjunctieve normaalvorm is consistent asa het minstens één conjunctie bevat zonder tegengestelde literals P en $\neg P$.*

Een propositionele zin in conjunctieve normaalvorm is logisch waar asa elke clause een literal P en zijn negatie $\neg P$ bevat.

Bew. Als elke conjunctie B in de DNF formule A literals P en $\neg P$ bevat, dan zal elke conjunctie en dus A zelf onwaar zijn in elke structuur. Omgekeerd, als A een conjunctie $B = L_1 \wedge \cdots \wedge L_n$ bevat zonder tegengestelde literals, kies dan een willekeurige structuur $\mathfrak{A}$ die $L_1, \ldots, L_n$ waar maakt. $\mathfrak{A}$ voldoet aan A.

Het bewijs van het tweede deel is analoog. Q.E.D. .

We zien hier een potentieel belangrijk voordeel van formules in DNF en in CNF: we kunnen gemakkelijk herkennen of ze consistent, respectievelijk logisch waar zijn.

Propositie 3.3.2. *Een kwantor-vrije formule A is logisch equivalent met een formule B in disjunctieve normaalvorm en met een formule C in conjunctieve normaalvorm.*

De transformatie in een normaalvorm maakt gebruik door herschrijving van de oorspronkelijke formule door middel van wetten van het denken, t.t.z., door middel van equivalentiebewarende herschrijving.

Bew. Eerst brengen we A' in negatie-normaalvorm door $\neg$ door $\wedge$ en $\vee$ te schuiven gebruik makend van de wetten van de Morgan (Eig. 3.3.3). De laatste stap hangt af of we disjunctieve dan wel conjunctieve normaalvorm willen bereiken. Indien we disjunctieve normaalvorm willen bekomen, dan schuiven we $\wedge$ door $\vee$ door middel van de distributiviteit van $\wedge$ t.o.v. $\vee$. Willen we de conjunctieve normaalvorm, dan passen we herhaald de distributiviteit van $\vee$ t.o.v. $\wedge$ toe en schuiven zo $\vee$ door $\wedge$.

Ook van toepassing zijn idempotentie-regels $P \wedge P \equiv P$ en $P \vee P \Leftrightarrow P$, om herhalingen van dezelfde deelformule in een conjunctie of disjunctie te verwijderen, en het verwijderen van inconsistenties $P \wedge \neg P$. Q.E.D.

Ter herinnering, als we schrijven:

$$A \equiv B$$

dan bedoelen we:

$$A \text{ is logisch equivalent met } B$$

of

$$\text{voor elke } \mathfrak{A} \text{ geldt } \mathfrak{A} \models A \text{ asa } \mathfrak{A} \models B$$

Voorbeeld 3.3.1. $\neg(P \wedge Q) \wedge (R \vee \neg T)$ is niet in disjunctieve normaalvorm. We hebben echter de volgende logische equivalenties:

$$\begin{aligned}
&\neg(P \wedge Q) \wedge (R \vee \neg T) \\
&\qquad \equiv (\neg P \vee \neg Q) \wedge (R \vee \neg T) \quad \text{De Morgan} \\
&\qquad \equiv [(\neg P \vee \neg Q) \wedge R] \vee [(\neg P \vee \neg Q) \wedge \neg T] \quad \text{distributiviteit} \\
&\qquad \equiv [(\neg P \wedge R) \vee (\neg Q \wedge R)] \vee [(\neg P \wedge \neg T) \vee (\neg Q \wedge \neg T)] \\
&\qquad \equiv (\neg P \wedge R) \vee (\neg Q \wedge R) \vee (\neg P \wedge \neg T) \vee (\neg Q \wedge \neg T)
\end{aligned}$$

De laatste zin is in disjunctieve normaalvorm.

Voorbeeld 3.3.2. $\neg(P \Rightarrow Q) \Leftrightarrow (R \vee Q)$ is niet in disjunctieve normaalvorm. We hebben echter de volgende equivalenties:

$$\begin{aligned}
&\neg(P \Rightarrow Q) \Leftrightarrow (R \vee Q) \\
&\qquad \equiv [\neg(P \Rightarrow Q) \Rightarrow (R \vee Q)] \wedge [(R \vee Q) \Rightarrow \neg(P \Rightarrow Q)] \\
&\qquad \equiv [(P \Rightarrow Q) \vee (R \vee Q)] \wedge [\neg(R \vee Q) \vee \neg(P \Rightarrow Q)] \\
&\qquad \equiv \big[(\neg P \vee \underline{Q}) \vee (R \vee \underline{Q})\big] \wedge [\neg(R \vee Q) \vee \neg(\neg P \vee Q)] \\
&\qquad \equiv [\neg P \vee Q \vee R] \wedge [(\neg R \wedge \neg Q) \vee (P \wedge \neg Q)] \\
&\qquad \equiv [(\neg P \vee Q \vee R) \wedge (\neg R \wedge \neg Q)] \vee [(\neg P \vee Q \vee R) \wedge (P \wedge \neg Q)] \\
&\qquad \equiv (\neg P \wedge \neg R \wedge \neg Q) \vee (\underline{Q} \wedge \neg R \wedge \neg \underline{Q}) \vee (\underline{R} \wedge \neg \underline{R} \wedge \neg Q) \vee (\neg \underline{P} \wedge \underline{P} \wedge \neg Q) \\
&\qquad\qquad\qquad\qquad \vee (\underline{Q} \wedge P \wedge \neg \underline{Q}) \vee (R \wedge P \wedge \neg Q) \\
&\qquad \equiv (\neg P \wedge \neg Q \wedge \neg R) \vee (P \wedge \neg Q \wedge R)
\end{aligned}$$

De laatste zin is in disjunctieve normaalvorm. Merk op dat we conjuncties met een symbool en zijn negatie verwijderd hebben.

Voorbeeld 3.3.3. Breng in disjunctieve normaalvorm

$$[(P \wedge R) \Rightarrow \neg Q] \wedge (Q \vee R).$$

$$\begin{aligned}
[(P \wedge R) \Rightarrow & \neg Q] \wedge (Q \vee R) \\
& \equiv [\neg(P \wedge R) \vee \neg Q] \wedge (Q \vee R)) \\
& \equiv [\neg P \vee \neg R \vee \neg Q] \wedge (Q \vee R)) \\
& \equiv [\neg P \wedge (Q \vee R)] \vee [\neg R \wedge (Q \vee R)] \vee [\neg Q \wedge (Q \vee R)]) \\
& \equiv (\neg P \wedge Q) \vee (\neg P \wedge R) \vee (\neg R \wedge Q) \vee \underline{(\neg R \wedge R)} \vee \underline{(\neg Q \wedge Q)} \vee (\neg Q \wedge R)) \\
& \equiv (\neg P \wedge Q) \vee (\neg P \wedge R) \vee (\neg R \wedge Q) \vee (\neg Q \wedge R)
\end{aligned}$$

Oefening 3.3.5. *Breng de volgende formule in conjunctieve normaalvorm*

$$[(P \Rightarrow R) \Leftrightarrow \neg(R \wedge (Q \Rightarrow P))]$$

Je zou verwachten dat propositionele solvers die berekenen of een formule consistent of logisch waar is, deze formule omzetten naar DNF, respectievelijk CNF. Maar helaas werkt dat helemaal niet goed: immers vaak is de DNF en CNF vorm van een formule exponentieel veel groter dan de oorspronkelijke formule. Men heeft dus andere methodes ontwikkeld. Dit gebeurt in het domein van SAT solvers.

3.4 Wetten van het denken van predikatenlogica

Strikt genomen hebben we tot nog toe ongeveer een 25-tal logische waarheden gezien. Op de oneindig veel formules die er bestaan is dat niet vet. We hebben mechanismes nodig om meer logisch ware zinnen te ontdekken.

Generaliseren van logische waarheden. Neem de logisch ware zin $[(P \vee Q) \wedge \neg P] \Rightarrow Q$. We voelen intuïtief aan dat deze zin staat voor oneindig veel andere logische waarheden, namelijk door P and Q te vervangen door andere formules.

- Door P te vervangen door S, en door Q te vervangen door $\exists x\colon R(x)$ bekomen we de zin $[(S \vee \exists x\colon R(x)) \wedge \neg S] \Rightarrow \exists x\colon R(x)$. Deze lijkt ook logisch waar.
- Door P te vervangen door $R(x, y)$ en Q door $Q(y)$, en vervolgens de vrije symbolen x, y universeel te kwantificeren bekomen we $\forall x\colon \forall y\colon [(R(x, y) \vee Q(y)) \wedge \neg R(x, y)] \Rightarrow Q(y)$. Ook deze zin lijkt logisch waar.

Deze voorbeelden laten twee operaties zien die een logisch ware zin omzet in een andere logisch ware zin: (1) het vervangen van propositionele symbolen P door formules B_P, (2) het quantificeren van vrije object-symbolen in een logisch ware zin. De correctheid van (1) wordt bewezen in de Generalisatieproposition 3.4.1. De correctheid van (2) in Propositie 3.4.2.

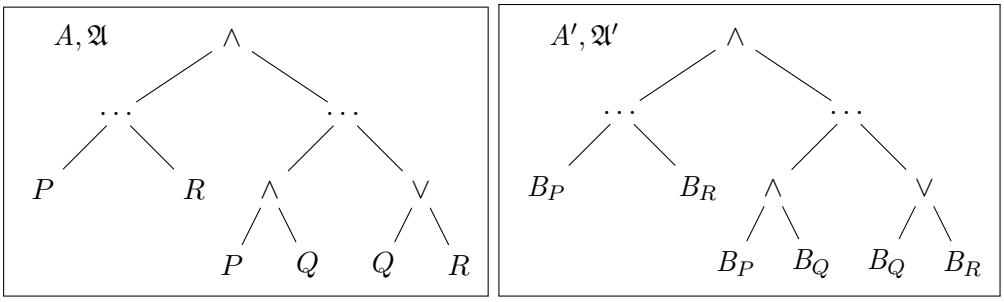
Propositie 3.4.1 (Generalisatiepropositie)**.** *Zij A een logisch ware propositionele zin. Kies voor elk symbool P in A een logische formule B_P en beschouw de formule A' bekomen uit A door elk symbool P te vervangen door B_P. Dan geldt dat A' logisch waar is.*

Zijn A een logisch ware zin. Te bewijzen is dat A' logisch waar is.

Het lijkt logisch dat we hiervoor een inductiebewijs opstellen. Maar hoe zou de inductiehypothese eruit zien? Bv. voor elke logisch ware A van grootte $\leq n$ is A' logisch waar? Maar dit werkt niet, omdat de deelformules van A niet voldoen aan deze inductiehypothese: de deelformules van een logisch ware zin zijn niet noodzakelijk logisch waar. Bv. $\neg A$ is logisch waar asa A logisch inconsistent is. Bv. $(\forall x : P(x)) \vee \neg(\forall x : P(x))$: deze zin is logisch waar, maar de deelformules niet. Dit idee werkt niet.

Volgens de definitie van logische waarheid, moet bewezen worden dat A' waar is in elke structuur $\mathfrak{A}'$ die A' interpreteert. Dit zullen we doen door een structuur $\mathfrak{A}$ te construeren die A interpreteert zodat $A^{\mathfrak{A}} = {A'}^{\mathfrak{A}'}$. Anders gezegd, $\mathfrak{A} \models A$ asa $\mathfrak{A}' \models A'$. Bijgevolg, aangezien A logisch waar is geldt $\mathfrak{A} \models A$ en dus ook $\mathfrak{A}' \models A'$. Hieruit volgt dat A' logisch waar is. Dit is het idee van het bewijs.

Om te zien hoe $\mathfrak{A}$ uit $\mathfrak{A}'$ te construeren vergelijken we A en A'. De syntax-bomen van A en A' hebben dezelfde structuur tot aan de bladeren van A, maar die van A' is dieper omdat de formules B_P zelf complexe formules kunnen zijn:



Het idee is nu om voor de gegeven $\mathfrak{A}'$, de structuur $\mathfrak{A}$ zo te definiëren dat de bladeren P van de syntaxboom van A dezelfde waarde hebben in $\mathfrak{A}$ als de formules B_P in $\mathfrak{A}'$. Dan zullen de gelijke waarheidswaardes in de bladeren omhoog propageren naar de wortels van de bomen zodat $A^{\mathfrak{A}} = {A'}^{\mathfrak{A}'}$. Om dit idee wiskundig sluitend te maken is een inductie-bewijs vereist.

Laat ons recapituleren. Zij A de logisch ware propositionele formule uit de Generalisatiepropositie, en voor elk symbool P in A, zij B_P een logische formule, mogelijks over een heel ander vocabularium. Zij Σ het vocabularium bestaande uit de symbolen die voorkomen in A, en Σ' het vocabularium bestaande uit de symbolen in de formules B_P (object-, functie- en predicaatsymbolen). Met elke formule B over Σ komt dan een getransformeerde formule B' over Σ' overeen, namelijk door elk symbool P in B te vervangen door B_p.

Zij $\mathfrak{A}'$ een willekeurige structuur die Σ' interpreteert (dus ook A'). Definieer de structuur $\mathfrak{A}$ voor Σ, door voor elke $P \in \Sigma$ te definiëren: $P^{\mathfrak{A}} := {B_P}^{\mathfrak{A}'}$.

Lemma 1. *Voor elke B over Σ geldt dat $B^{\mathfrak{A}} = {B'}^{\mathfrak{A}'}$.*

Dit lemma gaat dus over alle formules B over Σ, en niet alleen over A. Het vereist ook niet dat B logisch waar moet zijn.

Vooralleer het bewijs aan te vatten, nog een woord over verschillende soorten inductiebewijzen. Een inductiebewijs is meestal over de natuurlijke getallen. Het lemma kan men bewijzen door inductie op de grootte van B, t.t.z. op het aantal connectieven in B. Een alternatief is *inductie op de structuur van* B. Dit gaat als volgt: (basisstap) men toont aan dat de stelling geldt voor atomische formules; (inductiestap) men toont aan dat als de stelling geldt voor de componenten van een samengestelde formule, de stelling ook geldt voor de samengestelde formule zelf. Uit deze twee soorten stappen volgt dat de stelling geldt voor *alle* formules.

De twee soorten bewijzen zijn te herleiden tot elkaar, maar bewijs op de structuur van formules is vaak iets eenvoudiger omdat er niet geteld moet worden. Hieronder illustreren we deze laatste werkwijze.

Bew. We bewijzen $B^{\mathfrak{A}} = B'^{\mathfrak{A}'}$ per inductie op de structuur van B.

- (basisstap) Zij B een atoom. Dan bestaat een atoom P zodat $B = P$. Dan is $B' = B_P$. Per definitie van $\mathfrak{A}$ geldt dat $B^{\mathfrak{A}} = B'^{\mathfrak{A}'}$.
- (inductiestap) Zij B een samengestelde formule. We maken een gevallenonderscheid.
  - Veronderstel dat $B = C \wedge D$. Veronderstel dat $C^{\mathfrak{A}} = C'^{\mathfrak{A}'}$ en $D^{\mathfrak{A}} = D'^{\mathfrak{A}'}$ dan volgt hieruit dat $B^{\mathfrak{A}} = (C \wedge D)^{\mathfrak{A}} = (C' \wedge D')^{\mathfrak{A}'} = B'^{\mathfrak{A}'}$. De cruciale tweede stap $(C \wedge D)^{\mathfrak{A}} = (C' \wedge D')^{\mathfrak{A}'}$ volgt uit het feit dat als de componenten van twee conjuncties dezelfde waarheidwaarde hebben, dan hebben de conjuncties dezelfde waarheidswaarde.
  - De andere gevallen voor $\neg, \vee, \Rightarrow, \Leftrightarrow$ gaan analoog.

Uit het principe van bewijs door structurele inductie volgt dat voor alle B over Σ geldt dat $B^{\mathfrak{A}} = B'^{\mathfrak{A}'}$.

Q.E.D.

Let op: we zijn er nog niet. We moeten nog altijd de Generalisatiepropositie zelf bewijzen.

Bewijs. (van de Generalisatiepropositie 3.4.1)) Zij A, A' zoals gegeven. Zij $\mathfrak{A}'$ een willekeurige structuur die A' interpreteert. Lemma 1 garandeert dan dat $A^{\mathfrak{A}} = A'^{\mathfrak{A}'}$. Aangezien A logisch waar is geldt $A^{\mathfrak{A}} = \mathbf{t}$, zodat ook $A'^{\mathfrak{A}'} = \mathbf{t}$. Aangezien $\mathfrak{A}'$ willekeurig is volgt uit de definitie van logische waarheid dat A' logisch waar is.

■

Opmerking 3.4.1. Bij de Generalisatie-propositie staat veel uitleg om de studenten inzicht te geven hoe dat bewijs tot stand is gekomen en hoe het ineen zit. Maar begin niet met die uitleg van buiten te blokken aub. Als op het examen de vraag komt om de Generalisatiepropositie te geven en te bewijzen, dan volstaat het de propositie te geven, het lemma, en de formele bewijzen van beide. Dat volstaat voor een topscore.

Oefening 3.4.1. *Vaak komen we in een situatie dat we niet weten of een bepaalde interessante eigenschap al dan niet geldig is. Dan hebben we nood aan twee strategiën: één om de eigenschap te bewijzen en een andere om de negatie te bewijzen, en we proberen beide tergelijkertijd. Daarom, stel dat Stelling 3.4.1 niet waar was. Formuleer een nieuwe stelling die de negatie van de stelling*

uitdrukt. Bedenk vervolgens een strategie waarmee je deze stelling zou bewijzen. (Het bewijs zou natuurlijk falen, maar dat doet er niet toe.)

Propositie 3.4.2. *Zij A een formule met vrije objectsymbolen $v_0, \ldots, v_n$. Dan is A logisch waar asa $(\forall v_0 \colon \ldots \forall v_n \colon A)$ logisch waar is.*

Bew. ($\Rightarrow$) Uit het ongerijmde. Veronderstel dat $\models A$ en $\not\models \forall v_0 \colon \ldots \forall v_n \colon A$. Uit de definitie van logische waarheid volgt dat er een structuur $\mathfrak{A}$ bestaat zodat $\mathfrak{A} \not\models \forall v_0 \colon \ldots \forall v_n \colon A$. Door n-toepassingen van de $\forall$-regel vinden we dat er domeinelementen $d_0, \ldots, d_n$ bestaan zodat $\mathfrak{A}[x_0 : d_0, \ldots, x_n : d_n] \not\models A$. Dan is $\mathfrak{A}[x_0 : d_0, \ldots, x_n : d_n]$ dus een structuur waarin A onwaar is. Dus volgt uit de definitie van logische waarheid dat A niet logisch waar is. Contradictie.

($\Leftarrow$) Veronderstel dat $\not\models A$ en $\models \forall v_0 \colon \ldots \forall v_n \colon A$. Uit de definitie van logische waarheid volgt dat er een structuur $\mathfrak{A}$ bestaat die A interpreteert en zodat $\mathfrak{A} \not\models A$. Aangezien $v_0, \ldots, v_n$ vrije voorkomen in A worden zij geïnterpreteerd door $\mathfrak{A}$. Dus kan $\mathfrak{A}$ geschreven worden als een structuur $\mathfrak{A}'[v_0 : d_0, \ldots, v_n : d_n]$. Er geldt dus $\mathfrak{A}'[v_0 : d_0, \ldots, v_n : d_n] \not\models A$ waaruit volgt dat $\mathfrak{A}' \not\models \forall v_0 \colon \ldots \forall v_n \colon A$. Dus is deze laatste formule niet logische waar. Contradictie. Q.E.D.

Hier is nog een nuttige vervangingseigenschap.

Propositie 3.4.3 (Vervangpropositie)**.** *Zij A, B logisch equivalente formules. Zij C een formule waarin A één of meerdere keren als deelformule voorkomt. Veronderstel dat de formule C' kan bekomen worden door één of meerdere voorkomens van A te vervangen door B. Dan geldt dat C en C' logisch equivalent zijn.*

Oefening 3.4.2. *Zet de abstracte syntax-bomen van C en C' naast elkaar en bewijs de Vervangpropositie. Geef het informeel bewijs, en het formeel bewijs. Dit is gelijkaardig als de Generalisatiepropositie.*

Voorbeeld 3.4.1. Als toepassing van deze eigenschappen geven we nu een alternatief bewijs van één van de fundamentele equivalenties die we zonet zagen: de contrapositie eigenschap 3.3.5(2): $(P \Leftrightarrow Q) \Leftrightarrow (\neg P \Leftrightarrow \neg Q)$.

$$\begin{array}{ll} \mathfrak{A} \models P \Leftrightarrow Q & \text{asa } \mathfrak{A} \models (P \wedge Q) \vee (\neg P \wedge \neg Q) \\ & \text{asa } \mathfrak{A} \models (\neg\neg P \wedge \neg\neg Q) \vee (\neg P \wedge \neg Q) \\ & \text{asa } \mathfrak{A} \models (\neg P \Leftrightarrow \neg Q) \end{array}$$

Bijgevolg zijn de eerste en de laatste formule waar in exact dezelfde structuren. Uit Propositie 3.1.2 volgt dat $(P \Leftrightarrow Q) \Leftrightarrow (\neg P \Leftrightarrow \neg Q)$ logisch waar is.

Laat ons deze stappen in detail bekijken. Voor de eerste stap vertelt Eig. 3.3.4(3) dat $(P \Leftrightarrow Q) \Leftrightarrow [(P \wedge Q) \vee (\neg P \wedge \neg Q)]$ logisch waar is. Prop. 3.1.2 geeft ons dat $P \Leftrightarrow Q$ logisch equivalent is met $(P \wedge Q) \vee (\neg P \wedge \neg Q)$ wat per definitie betekent dat voor elke structuur $\mathfrak{A}$, $\mathfrak{A} \models P \Leftrightarrow Q$ asa $\mathfrak{A} \models (P \wedge Q) \vee (\neg P \wedge \neg Q)$.

In de tweede stap gebruiken we de logisch ware formule $\neg\neg P \Leftrightarrow P$ (Eig. 3.3.3(1)). Uit de Generalisatiepropositie 3.4.1 volgt dat ook $\neg\neg Q \Leftrightarrow Q$ logisch waar is. Prop. 3.1.2 impliceert dat $\neg\neg P$ en P logisch equivalent zijn, en net zo $\neg\neg Q$ en Q. Uit de Vervangpropositie 3.4.3

volgt dat we het eerste voorkomen van P door $\neg\neg P$ en van Q door $\neg\neg Q$ mogen vervangen, met behoud van de waarheidswaarde.

Voor de derde stap gaat de precieze redenering als volgt: aangezien $(P \Leftrightarrow Q) \Leftrightarrow [(P \wedge Q) \vee (\neg P \wedge \neg Q)]$ logisch waar is (Eig. 3.3.4(3)), volgt uit de generalisatiepropositie (vervang P door $\neg P$ en Q door $\neg Q$) dat $(\neg P \Leftrightarrow \neg Q) \Leftrightarrow [(\neg P \wedge \neg Q) \vee (\neg\neg P \wedge \neg\neg Q)]$ logisch waar is.

Disjunctie is commutatief: $\models (P \vee Q) \Leftrightarrow (Q \vee P)$ 3.3.1(1). Door toepassing van de generalisatiepropositie bekomen we $\models [(\neg P \wedge \neg Q) \vee (\neg\neg P \wedge \neg\neg Q)] \Leftrightarrow [(\neg\neg P \wedge \neg\neg Q) \vee (\neg P \wedge \neg Q)]$. Dus zijn $(\neg P \wedge \neg Q) \vee (\neg\neg P \wedge \neg\neg Q)$ en $(\neg\neg P \wedge \neg\neg Q) \vee (\neg P \wedge \neg Q)$ logisch equivalent. Door de vervangpropositie geldt dat $\models (\neg P \Leftrightarrow \neg Q) \Leftrightarrow [(\neg\neg P \wedge \neg\neg Q) \vee (\neg P \wedge \neg Q)]$.

Dus zijn $(\neg P \Leftrightarrow \neg Q)$ en $(\neg\neg P \wedge \neg\neg Q) \vee (\neg P \wedge \neg Q)$ logisch equivalent. Ze hebben dus dezelfde waarheidswaarde in $\mathfrak{A}$.

Q.E.D.

We zullen niet doorgaan met bewijzen te leveren op zo'n gedetailleerde manier, maar het is goed om zien dat we elke stap in ons bewijs kunnen argumenteren op basis van eerdere eigenschappen en proposities waarvan we de correctheid hebben bewezen. Het is hierop dat het bouwwerk van de wiskunde gebaseerd is. Dit is wat de Oude Grieken ontdekt hebben, en het is uit de studie van dergelijke gedetailleerde bewijzen dat logica gegroeid is!

Hernoeming van gebonden variabelen Intuïtief verwachten we dat de naamkeuze van een gebonden variabele x in een (deel)-formule $\exists x : A$ of $\forall x : A$ niet belangrijk is. Bv. dat

$$\forall x : P(x) \Rightarrow Q(x)$$

logisch equivalent is met de formule bekomen door x te substitueren door y: [2]

$$\forall y : P(y) \Rightarrow Q(y)$$

Het is logisch om bij hernoeming van de gebonden variabele x in $\forall x : A$, enkel de voorkomens van x te substitueren die gebonden zijn door die kwantor. Dat zijn precies de vrije voorkomens van x in A. Bv., in de volgende formule zijn er twee kwantificaties van x, waarvan 1 genest in de andere:

$$\exists x : (P(x) \wedge \exists x : Q(x))$$

Beide quantificaties kunnen apart hernoemd worden:

$$\underline{\exists x} : (P(x) \wedge \exists x : Q(x)) \longrightarrow \exists y : (P(y) \wedge \exists x : Q(x))$$

$$\exists x : (P(x) \wedge \underline{\exists x} : Q(x)) \longrightarrow \exists x : (P(x) \wedge \exists y : Q(y))$$

Dit leidt tot de volgende definitie. Daarin staat $\mathcal{Q}$ voor de quantifier $\forall$ of $\exists$.

[2] Dit is 1 van die stellingen die heel intuïtief aanvoelt, maar die niet gemakkelijk om te bewijzen is. Het bewijs volgt, maar probeer eerst zelf eens. Bij het bewijzen komen problemen aan de oppervlakte die gemakkelijk over het hoofd gezien worden. Het is een voorbeeld van een bewijs dat inzicht geeft dat moeilijk op een andere manier bereikt kan worden.

Definitie 3.4.1. De formule bekomen uit $\mathcal{Q}x : A$ **door hernoeming van** x **door** y is de formule $\mathcal{Q}y : A'$ waar A' bekomen wordt door alle vrije voorkomens van x in A te substitueren door y.

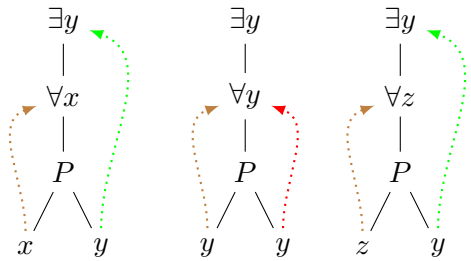
De vraag nu is: is de hernoeming van een gebonden variabele altijd equivalentie-bewarend? Niet altijd helaas. Bv., neem twee voorbeelden van een hernoeming van x door y in de volgende zinnen:

$$\underline{\forall x} : P(x, y) \longrightarrow \forall y : P(y, y)$$

$$\exists y : \underline{\forall x} : P(x, y) \longrightarrow \exists y : \forall y : P(y, y)$$

In beide gevallen is het niet moeilijk om in te zien dat de zinnen links en rechts niet equvialent zijn. Dus deze hernoemingen zijn niet equivalentie-bewarend. (Kun je dit aantonen?)

Het probleem is dat bij beide hernoemingen, bepaalde hernoemde voorkomens van variabelen door een andere quantor gekwantificeerd worden. Anders gezegd dat deze hernoemingen de posities van de referentie-bogen in de syntax-boom veranderen. Figuur 3.2 toont de foute (zie de rode boog) en een correcte hernoeming (van x door z):



Figuur 3.2: Hernoemingen van x door y en z in $\exists y : \underline{\forall x} : P(x, y)$

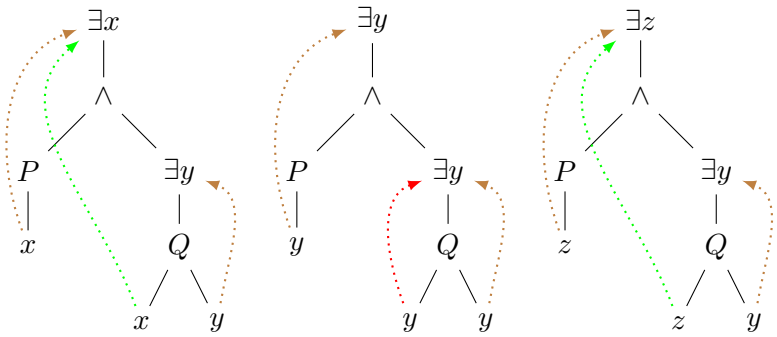
Als een ander voorbeeld, neem de volgende hernoeming van x door y:

$$\underline{\exists x} : (P(x) \wedge \exists y : Q(x, y)) \longrightarrow \exists y : (P(y) \wedge \exists y : Q(y, y))$$

Ook deze hernoeming bewaart duidelijk de equivalentie niet. (Kun je dit aantonen?) Opnieuw blijkt deze hernoeming de referentie-bogen in de syntax-boom te veranderen. Figuur 3.3 toont de foute en een correcte hernoeming:

De hypothese is bijgevolg dat als een hernoeming van x door y in $\mathcal{Q}x : A$ de structuur van de syntax-boom bewaart, inclusief de referentie-bogen, dan is ze equivalentie-bewarend. De voorbeelden illustreren de 2 mogelijke situaties waarin de referentie-bogen verlegd worden: wanneer y een vrij voorkomen heeft in A, en wanneer x in een vrij voorkomen heeft in een deelformule $\mathcal{Q}'y : B$ van A.

Definitie 3.4.2. Een hernoeming van x door y in $\mathcal{Q}x : A$ is **veilig** indien y geen vrije voorkomens heeft in A en indien vrije voorkomens van x in A niet voorkomen in een deelformule $\mathcal{Q}'y : B$ van A.

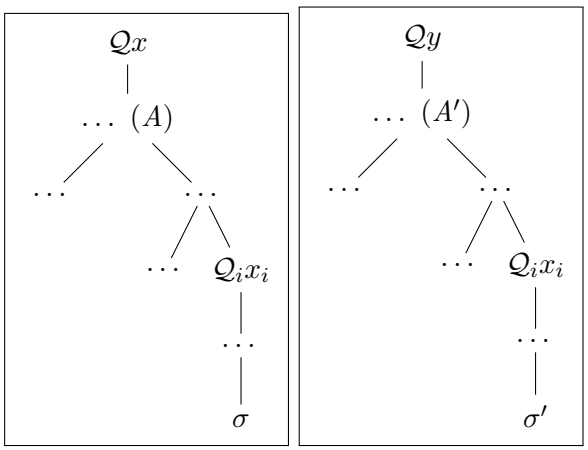


Figuur 3.3: Hernoemingen van x door y en door z in $\underline{\exists x} : (P(x) \wedge \exists y : Q(x, y))$

Propositie 3.4.4 (Veilige hernoemingen van gebonden variabelen)**.** *Een veilige hernoeming van een gebonden variabele x door y in een formule $\mathcal{Q}x : A$ is equivalentie-bewarend.*

Bew. (schets, niet verplicht voor het examen) Stel dat de oorspronkelijke formule $\mathcal{Q}x : A$ is en de hernoeming $\mathcal{Q}y : A'$. De twee formules hebben structureel dezelfde syntaxbomen, enkel verschillend doordat sommige nodes die x bevatten en een referentie-boog naar de topnode $\mathcal{Q}x$ bezitten, veranderd zijn in een node met y met referentie-boog naar de topnode $\mathcal{Q}y$. Dus, voor twee overeenkomstige nodes σ en σ' in beide bomen geldt: ze bevatten ofwel hetzelfde symbool, ofwel x, respectievelijk y en σ en σ' hebben allebei een referentie-boog naar de top. (Indien de hernoeming niet veilig was, zullen er nodes σ, σ' zijn, die x, respectievelijk y bevatten en die referentie-bogen naar niet-overeenkomstig gelegen nodes in de beide syntaxbomen verwijzen.)

Er dient aangetoond te worden dat voor elke structuur $\mathfrak{A}$, $(\mathcal{Q}x : A)^{\mathfrak{A}} = (\mathcal{Q}y : A')^{\mathfrak{A}}$. Dit wordt aangetoond per inductie op de structuur van de syntaxboom. Er is evenwel een technisch probleem: deelformules van deze formules worden niet in $\mathfrak{A}$ zelf geëvalueerd, maar in uitbreidingen ervan $\mathfrak{A}[x : v, x_1 : v_1, \ldots, x_n : v_n]$ met $x, x_1, \ldots, x_n$ de reeks van gekwantificeerde variabelen die liggen op het pad van de top van de syntaxboom, naar de deelformule. Dit compliceert de inductiehypothese. Zie figuur 3.4.



Figuur 3.4: Syntax-bomen van formule en de hernoeming

Veronderstel dat σ en σ' twee overeenkomstige nodes in de syntaxbomen zijn. Veronderstel ook dat de sequentie van quantoren naar σ als volgt is: $\mathcal{Q}x, \mathcal{Q}_1x_1, \ldots \mathcal{Q}_nx_n$. Dan zal in de hernoemde formule, de sequentie van quantoren naar σ' gelijk zijn aan $\mathcal{Q}y, \mathcal{Q}_1x_1, \ldots \mathcal{Q}_nx_n$.

Wat nu bewezen moet worden is het volgende: dat voor elk paar van uitbreidingen $\mathfrak{A}[x : v, x_1 : v_1, \ldots, x_n : v_n]$ en corresponderende uitbreiding $\mathfrak{A}[y : v, x_1 : v_1, \ldots, x_n : v_n]$ geldt dat:

$$\sigma^{\mathfrak{A}[x:v,x_1:v_1,\ldots,x_n:v_n]} = (\sigma')^{\mathfrak{A}[y:v,x_1:v_1,\ldots,x_n:v_n]}$$

Dit moet bewezen worden per inductie op de syntaxboom, dus beginnend bij de bladeren van de syntaxboom, en dan voor steeds hogere nodes.

Stel dat σ en σ' voorkomens van x en y bevattten, met referentie-boog naar de top. Dit is een basisgeval. Het is een geval van een voorkomen van x dat hernoemd werd tot y. Maar aangezien ze beide referentie-bogen hebben naar de top, worden deze variabelen geëvalueerd in de top toekenning aan x, respectievelijk y. Dus geldt

$$\sigma^{\mathfrak{A}[x:v,x_1:v_1,\ldots,x_n:v_n]} = v = (\sigma')^{\mathfrak{A}[y:v,x_1:v_1,\ldots,x_n:v_n]}$$

In alle andere gevallen geldt dat σ en σ' dezelfde symbool bevatten. Basigevallen zijn indien dit symbool een variabele of een object symbool is. Dan is de gelijkheid evident (zelfs als het symbool in kwestie een niet hernoemde x of y is). In het andere geval bevatten de nodes een connectief of een quantificatie $\mathcal{Q}z$. Dan volgt de gelijkheid uit de inductiehypothese die geldt voor de onderliggende nodes. Q.E.D.

Oefening 3.4.3. *Werk dit bewijs zorgvuldig uit. Let vooral op de gevallen dat σ een quantificatie $\mathcal{Q}z$ of een niet hernoemd voorkomen van x of y bevat.*

Oefening 3.4.4. *In de volgende formule, verifieer of x hernoemd mag worden door y en y door x. Verifieer in beide gevallen of er wat verkeerd gaat. Verifeer welke conditie van de hernoemstelling gebroken zijn:*

$$\exists x : \forall y : x = y$$

Propositie 3.4.5 (Eenvoudig geval)**.** *Als y geen vrije of gebonden voorkomens heeft in $\mathcal{Q}x : A$, dan is de hernoeming van x door y in $\mathcal{Q}x : A$ veilig.*

Oefening 3.4.5. *Bewijs dit zelf.*

Notatie 3.4.1. Vaak wordt een formule A die een vrij variabele symbool x bevat genoteerd als $A[x]$. In dat geval denoteert de uitdrukking $A[y]$ de formule bekomen door het hernoemen van elk vrij voorkomen van x door y. Dit is echter onveilig indien y vrij of gebonden voorkomt in A. Dus best is om deze operatie enkel uit te voeren indien y niet voorkomt in A.

Wetten van de kwantificatie.

Eigenschap 3.4.1. Zij A een formule.

1. $\models (\forall x\colon A) \Rightarrow (\exists x\colon A)$ is logisch waar.

Bew. Neem een willekeurige structuur $\mathfrak{A}$ die $(\forall x\colon A) \Rightarrow (\exists x\colon A)$ interpreteert. Er geldt:

$\mathfrak{A} \models (\forall x\colon A) \Rightarrow (\exists x\colon A)$ asa
$\mathfrak{A} \not\models \forall x\colon A$ of $\mathfrak{A} \models \exists x\colon A$ ($\Rightarrow$-regel).

We tonen aan dat deze disjunctie altijd voldaan is, of equivalent daarmee, dat als de eerste conditie niet voldaan is de tweede conditie wel voldaan is. Veronderstel dus dat de eerste conditie onwaar is. Er geldt:

$\mathfrak{A} \models \forall x\colon A$ asa
voor alle $d \in D(\mathfrak{A})$ geldt $\mathfrak{A}[x : d] \models A$ ($\forall$-regel).

Aangezien het domein $D(\mathfrak{A})$ van een structuur niet leeg is bestaat minstens één $d \in D(\mathfrak{A})$. Voor deze geldt $\mathfrak{A}[x : d] \models A$. Uit de $\exists$-regel volgt dan dat $\mathfrak{A} \models \exists x\colon A$. Dus is de tweede conditie voldaan.

We stellen vast dat de disjunctie voldaan is voor willekeurige $\mathfrak{A}$. Dus geldt $\mathfrak{A} \models (\forall x\colon A) \Rightarrow (\exists x\colon A)$ voor willekeurige $\mathfrak{A}$. Bijgevolg is de formule logisch waar. Q.E.D.

Eigenschap 3.4.2. Zij A een formule waarin x niet vrij voorkomt.

1. $\models (\forall x\colon A) \Leftrightarrow A$
2. $\models (\exists x\colon A) \Leftrightarrow A$

Deze eigenschap kan gebruikt worden om $(\forall x\colon A)$ te vereenvoudigen tot A.

De volgende eigenschap toont hoe $\neg$ op equivalentie bewarende wijze door een kwantor geschoven kan worden.

Eigenschap 3.4.3 (negatie van een kwantor)**.** Zij A een formule en x een variabele. De volgende equivalenties zijn logisch waar.

1. $\neg(\exists x\colon A) \Leftrightarrow (\forall x\colon \neg A)$
2. $\neg(\forall x\colon A) \Leftrightarrow (\exists x\colon \neg A)$

Bew. Wat (1) betreft, geldt het volgende:

$\mathfrak{A} \models \neg(\exists x\colon A)$ asa $\mathfrak{A} \not\models \exists x\colon A$
asa voor geen enkel domein element d van $\mathfrak{A}$ geldt $\mathfrak{A}[x : d] \models A$
asa voor elk domein element d van $\mathfrak{A}$ geldt $\mathfrak{A}[x : d] \not\models A$
asa voor elk domein element d van $\mathfrak{A}$ geldt $\mathfrak{A}[x : d] \models \neg A$
asa $\mathfrak{A} \models \forall x\colon \neg A$.

Het bewijs van (2) is analoog. Q.E.D.

In sommige gevallen kunnen we een kwantor op equivalentiebewarende wijze door een connectief schuiven: $\exists$ door $\vee$ en $\forall$ door $\wedge$. Of de omgekeerde bewerking is ook mogelijk: een kwantor buiten een connectief schuiven. Deze laatste bewerking zal gebruikt worden bij het omzetten naar prenex-normaalvorm.

Maar voor andere combinaties kwantoren en connectieven gaat de equivalentie verloren. Tenzij in speciale omstandigheden.

Eigenschap 3.4.4 (Doorschuiven van kwantoren door $\wedge, \vee$)**.**

1. $\models (\exists x\colon (A \vee B)) \Leftrightarrow (\exists x\colon A) \vee (\exists x\colon B)$ $\exists$ schuift door $\vee$

2. $\models (\forall x\colon (A \wedge B)) \Leftrightarrow (\forall x\colon A) \wedge (\forall x\colon B)$ $\forall$ schuift door $\wedge$

3. $\models (\exists x\colon (A \wedge B)) \Rightarrow (\exists x\colon A) \wedge (\exists x\colon B)$ maar
   $\not\models (\exists x\colon (A \wedge B)) \Leftarrow (\exists x\colon A) \wedge (\exists x\colon B)$ tenzij
   $\models (\exists x\colon (A \wedge B)) \Leftrightarrow (\exists x\colon A) \wedge B$ als x geen vrije constante is van B.

4. $\models (\forall x\colon (A \vee B)) \Leftarrow (\forall x\colon A) \vee (\forall x\colon B)$ maar
   $\not\models (\forall x\colon (A \vee B)) \Rightarrow (\forall x\colon A) \vee (\forall x\colon B)$ tenzij
   $\models (\forall x\colon (A \vee B)) \Leftrightarrow (\forall x\colon A) \vee B$ als x geen vrije constante is van B

Slechts in een beperkt aantal gevallen kunnen we dus de kwantoren doorschuiven. We bewijzen de gevallen (1), (3). De andere zijn analoog of kunnen door contrapositie bewezen worden uit de vorige.

Bew. van (1). Zij $\mathfrak{A}$ een willekeurige structuur die A, B interpreteert.

($\Rightarrow$) Veronderstel dat $\mathfrak{A} \models \exists x\colon A \vee B$. Uit de $\exists$-regel volgt dat er een $d \in D_{\mathfrak{A}}$ bestaat zodat $\mathfrak{A}[x : d] \models A \vee B$. Dus er zijn twee gevallen: $\mathfrak{A}[x : d] \models A$ of $\mathfrak{A}[x : d] \models B$. In het eerste geval volgt uit de $\exists$-regel dat $\mathfrak{A} \models \exists x\colon A$, in het tweede volgt dat $\mathfrak{A} \models \exists x\colon B$. In beide situaties kan uit de $\vee$-regel afgeleid worden dat $\mathfrak{A} \models (\exists x\colon A) \vee (\exists x\colon B)$.

($\Leftarrow$) Veronderstel dat $\mathfrak{A} \models (\exists x\colon A) \vee (\exists x\colon B)$. Uit de $\vee$-regel volgt dat (a) $\mathfrak{A} \models \exists x\colon A$ of dat (b) $\mathfrak{A} \models \exists x\colon B$.

In het geval (a) bestaat omwille van de $\exists$-regel een $d \in D_{\mathfrak{A}}$ zodat $\mathfrak{A}[x : d] \models A$. Dan volgt uit de $\vee$-regel dat $\mathfrak{A}[x : d] \models A \vee B$. Uit de $\exists$-regel volgt dan $\mathfrak{A} \models \exists x\colon (A \vee B)$.

In geval (b) komen we tot precies dezelfde conclusie.

Bew. van (3). Neem een willekeurige $\mathfrak{A}$ zodat $\mathfrak{A} \models \exists x\colon A \wedge B$. Hieruit volgt met de $\exists$-regel en de $\wedge$-regel dat een $d \in D_{\mathfrak{A}}$ bestaat zodat $\mathfrak{A}[x : d] \models A$ en $\mathfrak{A}[x : d] \models B$. Door op beide de $\exists$-regel toe te passen en vervolgens de $\wedge$-regel bekomen we $\mathfrak{A} \models (\exists x\colon A) \wedge (\exists x\colon B)$.

We hebben nu bewezen dat $(\exists x\colon A) \wedge (\exists x\colon B)$ een logisch gevolg van $\exists x\colon (A \wedge B)$. Bijgevolg is $(\exists x\colon (A \wedge B)) \Rightarrow (\exists x\colon A) \wedge (\exists x\colon B)$ logisch waar.

Om aan te tonen dat $\not\models (\exists x\colon (A \wedge B)) \Leftarrow (\exists x\colon A) \wedge (\exists x\colon B)$ moeten we A en B kiezen en een structuur construeren die deze implicatie onwaar maakt. Kies $A = P(x)$, $B = Q(x)$ en de structuur $\mathfrak{A}$ met domein $\{a, b\}$, $P^{\mathfrak{A}} = \{a\}$, $Q^{\mathfrak{A}} = \{b\}$. Verifieer dat in deze structuur de premisse waar is en de conclusie onwaar. Of ook, kies $A = (x = C), B = (\neg x = C)$.

Maar wat als x niet vrij is in B (en dus $\exists x\colon B$ en B logisch equivalent zijn) dan geldt de omgekeerde richting wel! Als $\mathfrak{A} \models (\exists x\colon A) \wedge B$, dan bestaat $d \in D_{\mathfrak{A}}$ zodat $\mathfrak{A}[x : d] \models A$ en $\mathfrak{A} \models B$. Aangezien x niet vrij is in B is de waarde van x in $\mathfrak{A}[x : d]$ van geen belang, zodat $\mathfrak{A}[x : d] \models B$. Bijgevolg geldt $\mathfrak{A} \models \exists x\colon (A \wedge B)$. Q.E.D.

Eigenschap 3.4.5 (Doorschuiven van kwantoren door $\Rightarrow$)**.**

5. $\models (\exists x\colon (A \Rightarrow B)) \Leftrightarrow (\forall x\colon A) \Rightarrow (\exists x\colon B)$

6. $\models (\forall x\colon A \Rightarrow B) \Rightarrow [(\exists x\colon A) \Rightarrow (\exists x\colon B)]$ maar
   $\not\models (\forall x\colon (A \Rightarrow B)) \Leftarrow [(\exists x\colon A) \Rightarrow (\exists x\colon B)]$

7. $\models (\forall x\colon (A \Rightarrow B)) \Rightarrow [(\forall x\colon A) \Rightarrow (\forall x\colon B)]$ maar
   $\not\models (\forall x\colon (A \Rightarrow B)) \Leftarrow [(\forall x\colon A) \Rightarrow (\forall x\colon B)]$

We bewijzen enkel (5) en (6).

Bew. van (5). Dit volgt door $A \Rightarrow B$ te vertalen naar $\neg A \vee B$, en de bestaande toe te passen.

Bew. van (6). Veronderstel $\mathfrak{A} \models \forall x\colon (A \Rightarrow B)$ en $\mathfrak{A} \models \exists x\colon A$. Dan geldt voor een $d \in D_{\mathfrak{A}}$ dat $\mathfrak{A}[x : d] \models A$ (met de $\exists$-regel). Uit de $\forall$-regel volgt $\mathfrak{A}[x : d] \models A \Rightarrow B$. Bijgevolg $\mathfrak{A}[x : d] \models B$ ($\Rightarrow$-regel). En dus geldt $\mathfrak{A} \models \exists x\colon B$ ($\exists$-regel).

Voor het niet gelden van de omgekeerde richting: kies een structuur waarin $P^{\mathfrak{A}}$ en $Q^{\mathfrak{A}}$ beide niet leeg zijn maar een lege doorsnede hebben. Dan geldt wel de rechtse deelformule, maar niet de linkse deelformule. Q.E.D.

Hoe zit het met het doorschuiven van kwantoren door $\Rightarrow$? Er zitten een paar merkwaardige gevallen bij. Bv.

$$\models (\exists x\colon (A \Rightarrow B)) \Leftrightarrow [(\forall x\colon A) \Rightarrow (\exists x\colon B)]$$

Vreemd! De meeste regels stroken met ons taalgevoel aan, maar deze? We botsen hier opnieuw tegen een materiële implicatie $\exists x\colon (A \Rightarrow B)$ die wij mensen vaak anders interpreteren dan in logica. Wie zich daarvan wil vergewissen moet eens kijken naar de formule $\exists x\colon (Succeeds(x) \Rightarrow Pass)$ in het laatste voorbeeld van Hoofdstuk 2. Doorschuiven van $\exists$ geeft $(\forall x\colon Succeeds(x)) \Rightarrow (\exists x\colon Pass)$, wat vereenvoudigt tot $(\forall x\colon Succeeds(x)) \Rightarrow Pass$. Beide zinnen zijn dus logisch equivalent, en dat is onverwacht.

Let op als je logische zinnen neerschrijft van de vorm $\exists x\colon (A \Rightarrow B)$: ze drukken zelden uit wat je bedoelde.

Oefening 3.4.6. *Zie Oefening 2.9.6, Oefening 2.9.7. In het vorig hoofdstuk heb je de veralgemeende kwantoren vertaald naar logica, en heb je ook bepaald welke kwantoren elkaars negatie waren. Bewijs dit nu formeel met behulp van de wetten van negatie. Bijvoorbeeld:*

- *Alle P's zijn Q's:* $\forall x\colon (P(x) \Rightarrow Q(x))$.
- *Minstens één P is geen Q:* $\exists x\colon (P(x) \wedge \neg Q(x))$.

De corresponderende zinnen zijn inderdaad elkaars negatie:

$$\begin{aligned}
\neg(\forall x\colon (P(x) \Rightarrow Q(x))) & \\
&\equiv (\exists x\colon \neg(P(x) \Rightarrow Q(x))) \\
&\equiv \exists x\colon \neg(\neg P(x) \vee Q(x)) \\
&\equiv \exists x\colon (\neg\neg P(x) \wedge \neg Q(x)) \\
&\equiv \exists x\colon (P(x) \wedge \neg Q(x))
\end{aligned}$$

Eigenschap 3.4.6 (verwisselen van kwantoren)**.** Zij A een formule en zij x, y variabelen, dan

1. $\models (\forall x\colon \forall y\colon A) \Leftrightarrow (\forall y\colon \forall x\colon A)$
2. $\models (\exists x\colon \exists y\colon A) \Leftrightarrow (\exists y\colon \exists x\colon A)$
3. $\models (\exists x\colon \forall y\colon A) \Rightarrow (\forall y\colon \exists x\colon A)$ maar
   $\not\models (\exists x\colon \forall y\colon A) \Leftarrow (\forall y\colon \exists x\colon A)$

Bew. van (1) en (2): evident.

Bew. van (3). De logische waarheid in de $\Rightarrow$ richting is evident: voor de tweede formule kies je voor elke y die ene x die volgens de eerste deelformule bestaat en A waar maakt.

Een tegenvoorbeeld voor de logische waarheid van de $\Leftarrow$ richting hebben we al eerder gegeven in Voorbeeld 2.6.1.

Doorschuiven van negatie Veronderstel dat we weten dat een uitspraak niet waar is. Wat is dan wel waar?

De wetten van connectieven en kwantoren laten toe om elke negatieve formule $\neg A$ om te zetten tot een positieve formule A' met dezelfde structuur maar waarin $\neg$ enkel in literals voorkomt. De resulterende formule A' kan bekomen worden uit A door eenvoudige rekenregels:
- schrap $\neg\neg$
- als A een atoom is, vervang A door $\neg A$
- $\wedge \to \vee$
- $\vee \to \wedge$
- $\forall \to \exists$
- $\exists \to \forall$.

Voorbeeld 3.4.2. $\neg(\exists x\colon [(P(x) \vee Q(x)) \wedge R(x)])$ wordt $\forall x\colon [(\neg P(x) \wedge \neg Q(x)) \vee \neg R(x)])$.

Andere herschrijfregels zijn
- $\neg(A \Rightarrow B) \to (A \wedge \neg B)$
- $\neg(A \Leftrightarrow B) \to (A \Leftrightarrow \neg B)$

Nadien kan $\neg B$ verder vereenvoudigd worden door $\neg$ door te schuiven in B.

Dezelfde wetten zijn ook van toepassingen in de context van nederlandse uitspraken over wiskundig. Meestal kunnen we de negatie van een nederlandse zin bijna automatisch omzetten in een positieve zin, zelfs zonder te weten wat de gegeven zin betekent. Als voorbeeld neem de volgende (onbegrijpelijke) uitspraak:

> Voor elke verzameling S behorend tot $\mathcal{F}$ bestaat er een element $y \in S$ zodat elke justification J met als wortel y transitief of symmetrisch is.

Wat geldt als deze uitspraak niet voldaan is?

> Er bestaat een verzameling S behorend tot $\mathcal{F}$ zodat voor elk element $y \in S$ een justification J met als wortel y bestaat die niet transitief is en niet symmetrisch.

Oefening 3.4.7. *Neem de volgende definitie:*

> *Een functie $f : \mathbb{R} \to \mathbb{R}$ is discontinu in het getal $a \in \mathbb{R}$ asa er bestaat een $\epsilon > 0$ zodat voor elke $\delta > 0$, er bestaat een $x \in]a-\delta, a+\delta[$ zodat $x \notin]f(a)-\epsilon, f(a)+\epsilon[$.*

Definieer dat een functie f continu is, t.t.z. niet discontinu is in het getal $a \in \mathbb{R}$. Is het mogelijk dat een functie tergelijkertijd continu en discontinu is in a?

3.4.1 Normaalvormen van predicatenlogica

Ook formules van predicatenlogica hebben hun normaalvormen.

Definitie 3.4.3. Een formule A is in **prenex-normaalvorm** als geen enkele kwantor voorkomt in een conjunctie, disjunctie, implicatie of equivalentie. Dus als elke kwantor boven/buiten elk connectief staat.

Voorbeeld 3.4.3.

$\exists z\colon \forall x\colon \exists v\colon (P(z) \Rightarrow Q(w,x)) \wedge \neg P(u)$	in prenex-normaalvorm
$\exists z\colon (\exists x\colon P(x)) \wedge (\exists u\colon Q(v,u))$	niet in prenex-normaalvorm
$(\exists z\colon P(z)) \wedge Q(z)$	niet in prenex-normaalvorm

Merk op dat in deze laatste zin $\exists$ syntactisch gezien vooraan staat, maar toch komt deze $\exists$ voor binnen de conjunctie. M.a.w., de formule is $(\exists z\colon P(z)) \wedge Q(z)$ en is niet in prenex-normaalvorm.

Propositie 3.4.6 (Prenex-normaalvorm)**.** *Elke formule is logisch equivalent met een formule in prenex-normaalvorm.*

Bew. Elke formule kan in prenex-normaalvorm gebracht worden met equivalentie bewarende transformatiestappen. Eerst vertalen we $\Rightarrow$ en $\Leftrightarrow$ in $\wedge, \vee$ en $\neg$. Vervolgens worden de kwantoren naar buiten geschoven. Daarvoor gebruiken we de volgende transformaties :

- $\neg\forall x\colon A$ wordt $\exists x\colon \neg A$ (Eig. 3.4.3)
- $\neg\exists x\colon A$ wordt $\forall x\colon \neg A$ (Eig. 3.4.3).
- $A \wedge \exists x\colon B[x]$ wordt $\exists y\colon A \wedge B[y]$
  Hierbij is y een nieuwe variabele die niet voorkomt in de formule. We hernoemen x door y (Prop. 3.4.4) en schuiven vervolgens $\exists y\colon$ naar buiten (Eig. 3.4.4(3iii) is van toepassing aangezien y niet voorkomt in A). Deze methode is van toepassing voor alle formules hieronder.
  Merk op dat als x niet voorkomt in A, hernoeming overbodig is.
- $\exists x\colon A[x] \wedge B$ wordt $\exists y\colon A[y] \wedge B$
- $A \vee \exists x\colon B[x]$ wordt $\exists y\colon A \vee B[y]$
- $(\exists x\colon A[x]) \vee B$ wordt $\exists y\colon A[y] \vee B$
- $A \wedge \forall x\colon B[x]$ wordt $\forall y\colon A \wedge B[y]$
- $(\forall x\colon A[x]) \wedge B$ wordt $\forall y\colon A[y] \wedge B$
- $A \vee \forall x\colon B[x]$ wordt $\forall y\colon A \vee B[y]$
- $(\forall x\colon A[x]) \vee B$ wordt $\forall y\colon A[y] \vee B$.

Door herhaalde toepassing van deze equivalentie bewarende omzettingen toe te passen, kunnen we al de kwantoren naar voren schuiven. Q.E.D.

Voorbeeld 3.4.4. Breng $(\exists x\colon P(x)) \Rightarrow (\forall y\colon Q(x,y))$ in prenex-normaalvorm.

Oplossing :

$$\begin{aligned}(\exists x\colon P(x)) \Rightarrow (\forall y\colon Q(x,y)) &\equiv \neg(\exists x\colon P(x)) \vee (\forall y\colon Q(x,y)) \\ &\equiv (\forall x\colon \neg P(x)) \vee (\forall y\colon Q(x,y)) \\ &\equiv \forall z\colon\ [\neg P(z) \vee (\forall y\colon Q(x,y))] \\ &\equiv \forall z\colon \forall y\colon\ [(\neg P(z)) \vee Q(x,y)]\ .\end{aligned}$$

Voorbeeld 3.4.5. Breng $(\exists x\colon P(x)) \wedge (\exists x\colon R(x))$ in prenex-normaalvorm.

Oplossing :

$$\begin{aligned}(\exists x\colon P(x)) \wedge (\exists x\colon R(x)) &\equiv \exists x\colon\ [P(x) \wedge \exists x\colon R(x)] \\ &\equiv \exists x\colon\ [P(x) \wedge \exists y\colon R(y)] \\ &\equiv \exists x\colon \exists y\colon\ [P(x) \wedge R(y)]\ .\end{aligned}$$

Opmerking 3.4.2. In de procedure voor het bekomen van de prenex-normaalvorm mogen vrije symbolen nooit van naam veranderd worden.

Definitie 3.4.4.

Een logische zin is in **prenex-conjunctieve normaalvorm** (pre-CNF) als het in prenex-normaalvorm is, en de kwantorvrije deelformule is in conjunctieve normaalvorm.

Een logische zin is in **prenex-disjunctieve normaalvorm** (pre-DNF) als het in prenex-normaalvorm is, en de kwantorvrije deelformule is in disjunctieve normaalvorm.

Propositie 3.4.7. *Een formule A is logisch equivalent met een formule B in prenex-disjunctieve normaalvorm en met een formule C in prenex-conjunctieve normaalvorm.*

Merk op dat de prenex-disjunctieve normaalvorm van een propositionele zin geen kwantoren bevat en dus in DNF is, en analoog voor CNF.

Bew. Eerst brengen we A in prenex-normaalvorm A'. Veronderstel dat de kwantorvrije deelformule ervan gelijk is aan B. Dan kunnen we B in DNF zoals als in CNF brengen zoals eerder gezien. Het resultaat is een prenex-disjunctieve of prenex-conjunctieve normaal vorm. Q.E.D.

3.5 Een formele bewijsmethode voor predikatenlogica

In de wiskunde wordt een wiskundig bewijs, zoals bv. van de Generalisatiepropositie 3.4.1 en van alle andere proposities en stellingen tot nog toe, ook wel eens een *formeel bewijs* genoemd.

Maar deze bewijzen zijn helemaal niet "formeel". Het zijn zorgvuldig geformuleerde, sluitende redeneringen over wiskundige objecten, maar ze zijn geformuleerd in een nederlands doorspekt met wiskundige symbolen. Om ze te begrijpen moeten we beroep doen op ons intuïtief begrip van de nederlands. Er zijn geen vast omschreven regels waarmee zulke bewijzen geconstrueerd mogen worden. Dus, wat men in de wiskunde vaak een formeel (of wiskundig) bewijs noemt, is zelf geen formeel wiskundig object. De logica is precies ontstaan met als doel dergelijke wiskundige bewijzen te onderzoeken en een formele notie van bewijs te introduceren. Een bewijs volgens de methode die we hier zullen zien is wel een formeel wiskundig object: het is een boom van strings die opgesteld wordt volgens strikt wiskundige regels.

In propositielogica hebben we een methode om na te gaan of een zin logisch waar is, of logisch consistent, of om na te gaan of een zin een logisch gevolg is van een andere zin. Namelijk door een waarheidstabel op te stellen. Daarin wordt elke mogelijke structuur opgesomd. Dit werkt niet voor predicatenlogica aangezien er oneindig veel structuren zijn.

In de loop van de 20ste eeuw werden talloze formele bewijssystemen voorgesteld voor predikatenlogica. We zullen één methode zien. Deze methode noemt men de KE-bewijsmethode (van *Klassiek Eliminatie-bewijs*). Omdat deze methode wel formeel is bestaan er computerprogramma's die de correctheid van zo'n bewijs kunnen verifiëren. Zo'n programma is beschikbaar in LogicPalet. In principe is het zelfs mogelijk om computerprogramma's te bouwen die zo'n bewijs automatisch kunnen genereren.

De methode dient om de inconsistentie van een theorie te bewijzen. Dat volstaat echter om ook andere logische redeneertaken op te lossen, namelijk die herleid kunnen worden tot bewijs van inconsistentie:

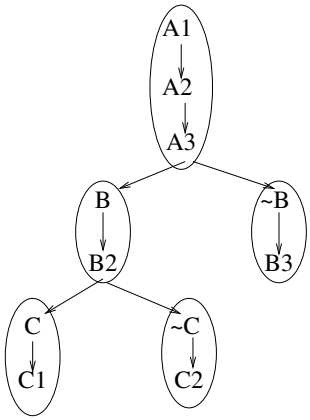
- Bewijs dat een zin A logisch waar is: immers A is logisch waar asa $\neg A$ is inconsistent.
- Bewijs dat een zin B een logisch gevolg is van een logische theorie $T = \{A_1, \ldots, A_n\}$. Immers, dit is het geval asa $\{A_1, \ldots, A_n, \neg B\}$ inconsistent is.
- Bewijs dat een zin B inconsistent is met de logische theorie T. Dat is het geval asa $\{A_1, \ldots, A_n, B\}$ inconsistent is.

De KE-bewijsmethode dient om inconsistentie te bewijzen, maar soms kan ze ook gebruikt worden om de consistentie van een theorie te bewijzen. Niet altijd evenwel zoals we later zullen zien.

Een KE-redenering uit een theorie $T = \{A_1, A_2, \ldots A_n\}$ bestaat uit boom van logische zinnen. Figuur 3.5 toont de structuur van een dergelijke boom. Hierin zijn A_i, B_i, C_i zinnen. De deelrijen komen overeen met de 5 omcirkelde rijen zinnen, bv. $A1, A2, A3$. Waar de boom splitst begint één tak met een formule en de andere met de negatie van deze formule. Anders gezegd, aan het eind van een rij kan de rij opgesplitst worden in twee deelrijen die met tegengestelde formules $A, \neg A$ beginnen.

KE-redeneringen worden geconstrueerd vertrekkende vanuit de lege boom door toepassing van uitbreidingsregels. Er zijn twee manieren om een tak van een KE-redenering onderaan uit te breiden:

- door een nieuwe zin toe te voegen die ofwel afkomstig is uit T, ofwel bekomen wordt door toepassing van een *propagatie-regel* op één of meerdere formules hogerop in de tak; deze propagatie-regels worden verderop uitgelegd;



Figuur 3.5: Structuur van een KE-bewijs

- ofwel door een gevalsonderscheid te maken voor een zin A en $\neg A$ en dus de tak op te splitsen.

Voorbeeld 3.5.1. We willen bewijzen dat de logische zin uit Voorbeeld 3.1.2 logisch waar is:

$$[(\forall x\colon P(x, R1) \vee P(x, R2)) \wedge \neg P(C, R1)] \Rightarrow P(C, R2)$$

Onze theorie bestaat dus uit de negatie van deze zin. Het KE-bewijs is de formele versie van het bewijs[3] uit het ongerijmde in Voorbeeld 3.1.2.

$\neg\{[(\forall x\colon P(x, R1) \vee P(x, R2)) \wedge \neg P(C, R1)] \Rightarrow P(C, R2)\}$	1. Theorie
$\neg P(C, R2)$	2. $\neg\Rightarrow$-prop(1)
$[(\forall x\colon P(x, R1) \vee P(x, R2)) \wedge \neg P(C, R1)]$	3. $\neg\Rightarrow$-prop(1)
$(\forall x\colon P(x, R1) \vee P(x, R2))$	4. $\wedge$-prop(3)
$\neg P(C, R1)$	5. $\wedge$-prop(3)
$(P(C, R1) \vee P(C, R2))$	6. $\forall$-prop(4)
Case A	
$P(C, R1)$	7. Case $P(C, R1)$
Contradictie 7,5	
Case B	
$\neg P(C, R1)$	8. Case $\neg P(C, R1)$
$P(C, R2)$	9. $\vee$-prop(6,8)
Contradictie 2,9	

Bv. de tweede stap $\neg(C, R2)$ werd afgeleid door de $\neg\Rightarrow$-prop propagatieregel toegepast op stap 1. De uitleg van deze regel volgt. Aan het eind gebeurt een gevalsonderscheid op $P(C, R1)$, de eerste disjunct van (6). Uit de negatie ervan in Case B volgt de tweede disjunct, die in contradictie is met de zin in stap 2. Alle takken leiden dus tot een contradictie. Dus is de oorspronkelijke zin inderdaad inconsistent en zijn negatie logisch waar.

Definitie 3.5.1. Een KE-redenering voor theorie T wordt gedefinieerd door inductie:

[3]Er bestaat overigens een verkorte versie van dit bewijs waarin geen gevalsonderscheid voor komt: namelijk door de $\vee$-prop regel toe te passen op 2,6 bekomt men meteen contradictie.

- De lege KE-redenering is een KE redenering (met 1 lege tak).
- Gegeven een KE-redenering voor T en een uitbreidingsregel die van toepassing is op een tak ervan, dan is het resultaat van toepassing van die uitbreidingsregel op die tak een KE-redenering.

Vervolgens introduceren we de uitbreidingsregels van KE-redeneringen.

Hypothesen-regel Deze regel is van toepassing op elke tak van elke KE-redenering. Toepassing van de hypothesenregel op een tak voegt een zin van T toe onderaan de tak.

Als strategieregel spreken we af om de hypothesen-regel 1 keer toe te passen voor elke zin van T vooraleer we andere uitbreidingsregels toepassen. De hypothesen-regel wordt dus enkel in het begin toegepast, om de theorie te kopiëren naar de initiële tak.

Propositionele propagatie-regels. Een propositionale propagatie-regel voegt een specifiek logisch gevolg van bepaalde formules op een tak toe onderaan de tak. Een dergelijke regel is van toepassing indien zinnen van de juiste vorm op de tak staan, zoals uitgelegd hieronder.

Bv. als $A \wedge B$ voorkomt op de geselecteerde tak, zal toepassing van de propagatie-regel $\wedge$-prop A en B onderaan de tak toevoegen.

Bv. als $A \vee B$ en $\neg A$ voorkomen op de tak, zal toepassing van $\vee$-prop B toevoegen aan de tak.

De propagatie-regels vertrekken allemaal van hetzelfde idee: als we een samengestelde formule A hebben dan kan uit de waarheidswaarde van A en sommige van zijn deelformules de waarheidswaarde van andere deelformules afgeleid worden. Bv. als $A \wedge B$ waar is, dan is A waar en ook B. Als $A \wedge B$ onwaar is en A waar, dan moet B onwaar zijn.

De propagatie-regels zijn de volgende:

$\neg$	$\neg\neg A$	$\longrightarrow$	A	$\neg\neg$-prop
$\wedge$	$A \wedge B$	$\longrightarrow$	A, B	$\wedge$-prop
	$\neg(A \wedge B),\quad B$	$\longrightarrow$	$\neg A$	$\neg\wedge$-prop
	$\neg(A \wedge B),\quad A$	$\longrightarrow$	$\neg B$	$\neg\wedge$-prop
$\vee$	$A \vee B\ ,\quad \neg A$	$\longrightarrow$	B	$\vee$-prop
	$A \vee B,\quad \neg B$	$\longrightarrow$	A	$\vee$-prop
	$\neg(A \vee B)$	$\longrightarrow$	$\neg A$, $\neg B$	$\neg\vee$-prop
$\Rightarrow$	$A \Rightarrow B,\quad A$	$\longrightarrow$	B	$\Rightarrow$-prop
	$A \Rightarrow B,\quad \neg B$	$\longrightarrow$	$\neg A$	$\Rightarrow$-prop
	$\neg(A \Rightarrow B)$	$\longrightarrow$	$A, \neg B$	$\neg\Rightarrow$-prop
$\Leftrightarrow$	$A \Leftrightarrow B,\quad A$	$\longrightarrow$	B	$\Leftrightarrow$-prop
	$A \Leftrightarrow B,\quad B$	$\longrightarrow$	A	$\Leftrightarrow$-prop
	$A \Leftrightarrow B,\quad \neg A$	$\longrightarrow$	$\neg B$	$\Leftrightarrow$-prop
	$A \Leftrightarrow B,\quad \neg B$	$\longrightarrow$	$\neg A$	$\Leftrightarrow$-prop
	$\neg(A \Leftrightarrow B),\quad A$	$\longrightarrow$	$\neg B$	$\neg\Leftrightarrow$-prop
	$\neg(A \Leftrightarrow B),\quad B$	$\longrightarrow$	$\neg A$	$\neg\Leftrightarrow$-prop
	$\neg(A \Leftrightarrow B),\quad \neg A$	$\longrightarrow$	B	$\neg\Leftrightarrow$-prop
	$\neg(A \Leftrightarrow B),\quad \neg B$	$\longrightarrow$	A	$\neg\Leftrightarrow$-prop

Merk op dat elke regel uit een samengestelde zin een deelzin of zijn negatie afleidt.

We voeren de volgende afspraak in: als een propagatie een formule van de vorm $\neg\neg A$ produceert, dan wordt de dubbele negatie meteen verwijderd. Dat bespaart ons een toepassing van de $\neg\neg$-prop regel.

Oefening 3.5.1. *Elk van deze propagatie-regels komt voort uit een tautologie, namelijk door $\longrightarrow$ te vervangen door $\Rightarrow$ en "," door $\wedge$. Bv. $\neg\neg A \Rightarrow A$. Wat is een geschikte methode om deze te bewijzen? Bewijs deze.*

ERule : $\exists$-regel. Deze regel is van toepassing op een tak waarin $\exists x\colon A[x]$ of $\neg\forall x\colon A[x]$ voorkomt. In het eerste geval zal toepassing ervan $A[c]$ toevoegen onderaan die tak, in het tweede geval $\neg A[c]$ met c een nieuwe constante die nog niet voorkomt in deze tak noch in T.

Opmerking 3.5.1. $A[x]$ duidt een formule aan waarin x een vrij voorkomen heeft. Veronderstel dat t een term is die geen enkel symbool y bevat dat gekwantificeerd voorkomt in $A[x]$. Dan is $A[t]$ de formule bekomen door elk vrij voorkomen van x te vervangen door t.

Let op: het is altijd en overal verboden om vrije voorkomens van x in $A[x]$ te vervangen door een term die gekwantificeerde variabelen uit $A[x]$ bevat. Bv. neem voor $A[x]$ de formule $Q(x) \vee \forall y\colon P(x,y)$, dan mag x nooit vervangen worden door y. Dit zou $Q(y) \vee \forall y\colon P(y,y)$ produceren: het probleem is dat de y binnen en buiten de kwantor eigenlijk een verschillend object aanduidt. We mogen x evenmin door $f(y), g(a,y), \ldots$ vervangen.

ARule: $\forall$-regel. Deze regel is van toepassing op een tak met $\forall x\colon A[x]$ of $\neg\exists x\colon A[x]$. In het eerste geval zal toepassing ervan $A[t]$ toevoegen aan die tak, in het tweede geval $\neg A[t]$. Hierbij is t een willekeurige term die geen gebonden variabelen van A bevat.

Een strategieregel hierbij is dat t bestaat uit functiesymbolen en constantes die eerder in de tak of in de theorie voorkomen. Enkel als er geen constanten in de tak noch in T voorkomen, mag je een nieuwe constante introduceren.

Gelijkheid-regels. De eerste gelijkheidsregel is van toepassing op elke tak en bestaat erin om onderaan de tak de gelijkheid $t = t$ toe te voegen, voor t een willekeurige term.

Dezelfde strategieregel als voor de Arule geldt: t bestaat uit functie- en constantesymbolen die voorkomen in de theorie of eerder in de tak.

Zij t_1, t_2 termen zodat de zinnen $t_1 = t_2$ en A_1 voorkomen in een tak en t_1 heeft een voorkomen in A_1 waarin elk symbool van t_1 vrij is (dus niet in de scoop van een kwantor zit). Dan is de tweede regel van toepassing. Toepassing ervan zal de formule A_2 toevoegen die je bekomt door dit voorkomen van t_1 in A_1 te vervangen door t_2.

Ter herinnering: $t_1 \neq t_2$ is een afkorting van $\neg t_1 = t_2$.

Voorbeeld 3.5.2. We willen bewijzen dat

$$\neg\left(\exists y\colon [Triangle(y) \wedge \forall x\colon \neg LeftOf(x,y)]\right)$$

een logisch gevolg is van

$$\exists x\colon \forall y\colon [Triangle(y) \Rightarrow LeftOf(x,y)].$$

Dit redeneerprobleem kan herleid worden tot het bewijzen van inconsistentie van de volgende theorie

$$\left\{ \begin{array}{l} \exists x\colon \forall y\colon [Triangle(y) \Rightarrow LeftOf(x,y)], \\ \neg\,[\neg\,(\exists y\colon [Triangle(y) \wedge \forall x\colon \neg LeftOf(x,y)])] \end{array} \right\}.$$

Hier is een KE-bewijs zonder gevalsonderscheid.

$\exists x\colon \forall y\colon [Triangle(y) \Rightarrow LeftOf(x,y)]$	1. Hypothesis
$\neg\,[\neg\,(\exists y\colon [Triangle(y) \wedge \forall x\colon \neg LeftOf(x,y)])]$	2. NegatedConclusion
$\exists y\colon [Triangle(y) \wedge \forall x\colon \neg LeftOf(x,y)]$	3. $\neg\neg$-regel(2)
$\forall y\colon [Triangle(y) \Rightarrow LeftOf(a,y)]$	4. ERule(1)
$Triangle(b) \wedge \forall x\colon \neg LeftOf(x,b)$	5. ERule(3)
$Triangle(b)$	6. $\wedge$-prop(5)
$\forall x\colon \neg LeftOf(x,b)$	7. $\wedge$-prop(5)
$Triangle(b) \Rightarrow LeftOf(a,b)$	8. ARule(4)
$LeftOf(a,b)$	9. $\Rightarrow$-prop(6,8)
$\neg LeftOf(a,b)$	10. ARule(7)
Contradiction. 9,10	

Je kunt verifiëren dat elke stap correct is zonder zelfs te weten wat deze zinnen betekenen. Om zo'n bewijs zelf op te stellen is het voor ons meestal noodzakelijk om bij de les te zijn en goed te begrijpen wat de zinnen betekenen, en zelf eerst een argument op te stellen waarom deze theorie inconsistent is. Het formele bewijs zal dan een herwerking van dat argument zijn.

In dit geval drukt de hypothese uit dat er een Geo-figuurtje bestaat (noem het a) dat links staat van elke driehoek. De te bewijzen zin specifieert dat er geen driehoek is waar geen enkel object links van voorkomt, t.t.z. elk object ligt minstens op gelijke hoogte of rechts ervan.

Gegeven: er bestaat een Geo-figuurtje dat links staat van elke driehoek.

T.B. : er is geen driehoek waar geen enkel object links van ligt.

Het bewijs dat volgt is de informele variant van het bovenstaande KE-bewijs zoals gebruikelijk in wiskundige teksten. Het is een bewijs uit het ongerijmde:

Bew. (uit het ongerijmde)

Gegeven is dat een figuurtje links staat van elke driehoek (stap 1). Noem dat figuurtje a (stap 4).

Veronderstel dat het te bewijzen onwaar is (stap 2). Dus, er is een driehoek waar geen enkel object links van ligt (stap 3). Noem deze driehoek b (stap 5).

b is dus een driehoek (stap 6). Dus staat a links van b (stap 8+9).

Maar b heeft de eigenschap dat geen enkel object er links van staat (stap 7). Dus staat a niet links van (b) (stap 10).

Contradictie. Q.E.D.

Gevalsonderscheiding-regel Deze regel is van toepassing op elke tak. Toepassing van ervan splitst de tak onderaan op door twee nieuwe deeltakken te openen, één beginnend met een formule A, de andere met $\neg A$.

Hier mag A eender welke zin zijn. Als strategieregel zullen we A kiezen als een deelzin van een zin die reeds voorkomt in de tak.

Voorbeeld 3.5.3. Toon aan met een KE-bewijs dat

$$\exists x\colon \forall y\colon [Triangle(y) \Rightarrow Square(x)]$$

een logisch gevolg is van

$$(\exists x\colon Triangle(x)) \Rightarrow (\exists x\colon Square(x))$$

KE-bewijs:

$(\exists x\colon Triangle(x)) \Rightarrow (\exists x\colon Square(x))$	1. Hypothesis
$\neg\{\exists x\colon \forall y\colon [Triangle(y) \Rightarrow Square(x)]\}$	2. NegatedConclusion
Case A	
$\exists x\colon Triangle(x)$	3. Case A
$\exists x\colon Square(x)$	5. $\Rightarrow$prop(1,3)
$Square(a)$	6. ERule(5)
$\neg\forall y\colon [Triangle(y) \Rightarrow Square(a)]$	7. ARule(2)
$\neg(Triangle(b) \Rightarrow Square(a))$	8. ERule(7)
$Triangle(b)$	9. PropRule(8)
$\neg Square(a)$	10. PropRule(8)
$Contradiction.10, 6$	
Case B	
$\neg\exists x\colon Triangle(x)$	4. Case B
$\neg\{\forall y\colon [Triangle(y) \Rightarrow Square(a)]\}$	12. ARule(2)
$\neg(Triangle(b) \Rightarrow Square(a))$	13. ERule(12)
$Triangle(b)$	14. PropRule(13)
$\neg Square(a)$	15. PropRule(13)
$\neg Triangle(b)$	16. ARule(4)
Contradiction: 14,16	

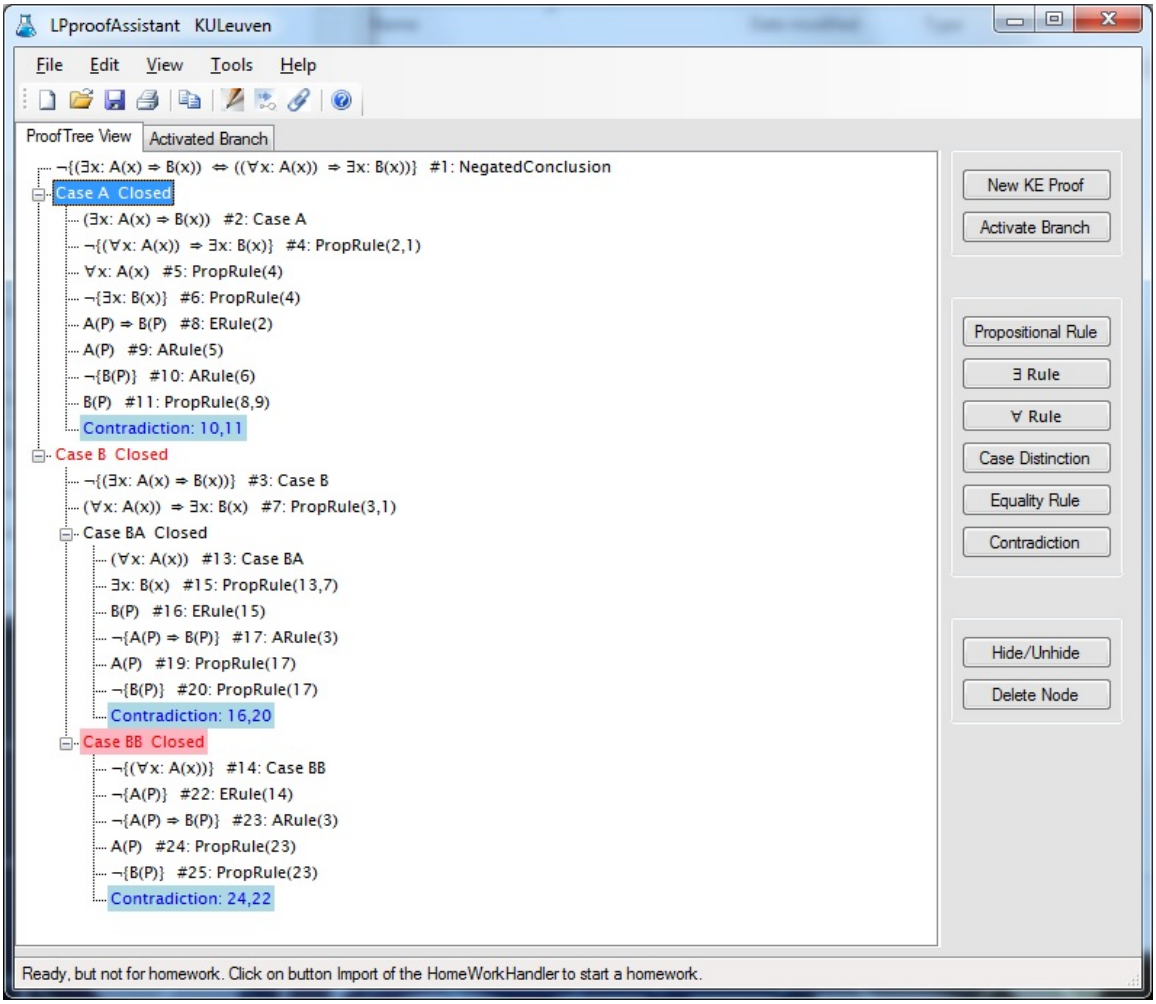
Merk op dat we voor een bewijs van een logisch gevolg onderscheid maken tussen theorie-stappen die een hypothese introduceren en een theorie-stap die de negatie van het gevolg introduceert.

Voorbeeld 3.5.4. We construeren een bewijs dat de formule $(\forall x\colon P(x, y)) \Rightarrow (\exists x\colon P(x, y))$ uit Eigenschap 3.4.1 logisch waar is.

$\neg\forall y\colon [(\forall x\colon P(x, y)) \Rightarrow (\exists x\colon P(x, y)]$	1. NegatedConclusion
$\neg[(\forall x\colon P(x, c_1)) \Rightarrow (\exists x\colon P(x, c_1)]$	2. Erule(1)
$(\forall x\colon P(x, c_1))$	3. $\neg\Rightarrow$-prop(2)
$\neg\exists x\colon P(x, c_1)$	4. $\neg\Rightarrow$-prop(4)
$P(c_1, c_1)$	5. Arule(3)
$\neg P(c_1, c_1)$	6. Arule(4)
Contradictie(5,6)	

Definitie 3.5.2. Een tak van een KE-redenering is *gesloten* indien er een contradictie in voorkomt: een zin A en ook $\neg A$.

Een KE-redenering uit T is een *KE-bewijs van inconsistentie van* T als elke tak gesloten is.



Figuur 3.6: Een KE-bewijs van $(\exists x\colon (A(x) \Rightarrow B(x))) \Leftrightarrow (\forall x\colon A(x)) \Rightarrow (\exists x\colon B(x))$

Gezien het formeel karakter van KE-redeneringen en bewijzen is het mogelijk om een computerproramma te schrijven dat een KE-bewijs verifieert.

Het is ook mogelijk om een computerprogramma te schrijven dat dit doet op een interactieve manier: de gebruiker kiest de uitbreidingsstappen en de computer bepaalt of ze van toepassing zijn en voert ze uit. De tool LPproofAssistant van de software LogicPalet is zo'n programma.

Het tool LPproofAssistant is een handig hulpmiddel om KE-bewijzen te construeren. Er zijn knoppen voorzien voor elk van de KE-regels. Daarop klikken past de regel toe en schrijft automatisch het resultaat. Er is een foutmelding wanneer de regel niet toepasbaar is waardoor het onmogelijk wordt om een foutief bewijs te construeren.

In Figuur 3.6 staat een KE-bewijs dat met de proof-assistant van LogicPalet is uitgewerkt.

Correctheid en volledigheid van KE-bewijzen De KE-bewijsmethode is een mechanische methode om formele bewijzen op te stellen. Maar is ze wel correct? Er stellen zich twee fundamentele vragen:

- Als er een KE-bewijs van inconsistentie van theorie T bestaat, zijn we dan zeker dat T

inconsistent is, t.t.z., geen enkel model heeft? Dit noemen we de **correctheid** van deze methode (in het engels, **soundness**). Dit zullen we bewijzen.

- Omgekeerd, als T inconsistent is, bestaat er dan ook een KE-bewijs van inconsistentie van T? Dit bespreken we verderop, maar zullen we niet bewijzen.

We beginnen dit onderdeel met een propositie die de werking van KE-redeneringen toont. Als T een model $\mathfrak{A}$ heeft, dan zal er in elke KE-redenering een tak zijn zodat alle formules op de tak (waar zijn in $\mathfrak{A}$. Elke uitbreidingstap aan die tak bewaart deze eigenschap en voegt een formule toe die waar is in $\mathfrak{A}$ (of in een uitbreiding van $\mathfrak{A}$).

Het nut van deze eigenschap is als volgt. Niet elke tak in een KE-redenering heeft een structuur waarin alle formules waar zijn. Bv. een gesloten tak bevat formules $A, \neg A$, en er is geen enkele structuur waarin beide voldaan zijn. Bijgevolg, als de KE-redenering een KE-bewijs is, t.t.z. alle takken ervan zijn gesloten, dan heeft T geen enkel model. Dan is T logisch inconsistent. Dit toont de correctheid van de KE-bewijsmethode aan. We werken deze redenering wiskundig uit.

Propositie 3.5.1. *Als een logische theorie T een model $\mathfrak{A}$ heeft, dan bevat elke KE-redenering voor T minstens 1 tak* `t` *zodat een uitbreiding $\mathfrak{A}_1$ van $\mathfrak{A}$ bestaat die alle nieuwe constanten c in die tak interpreteert, zodat alle zinnen van de tak* `t` *waar zijn in $\mathfrak{A}_1$.*

Takken van een KE-redenering voor T bevatten mogelijks nieuwe symbolen c die niet voorkomen in T maar die geïntroduceerd werden door toepassing van uitbreidingsregels, vooral Erule. Een model $\mathfrak{A}$ van T interpreteert deze symbolen niet, bijgevolg worden vele formules uit een KE-redenering niet geïnterpreteerd door $\mathfrak{A}$. Maar de stelling zegt dat de KE-redenering een tak `t` heeft en $\mathfrak{A}$ een uitbreiding $\mathfrak{A}_1$ met waardes $c^{\mathfrak{A}_1}$ voor die symbolen zodat alle formules op de tak `t` wel geïnterpreteerd worden door $\mathfrak{A}_1$; bovendien zijn alle formules op de tak voldaan in $\mathfrak{A}_1$.

Bew. Veronderstel dat $\mathfrak{A}$ een model is van T.

De propositie wordt bewezen op de lengte n van de KE-redeneringen voor T (t.t.z. het aantal uitbreidingsstappen die nodig zijn om de KE-redenering te construeren).

De stelling is voldaan voor de KE-redenering van lengte 0. Deze is leeg en bevat geen enkele formule. Neem dan $\mathfrak{A}_1 = \mathfrak{A}$; "alle" formules in de lege KE-redenering zijn voldaan in $\mathfrak{A}_1$.

Veronderstel dat de stelling voldaan is voor alle KE-redeneringen van lengte $n - 1 \geq 0$ (dit is de inductie-hypothese). Dan bewijzen we dat de stelling ook geldig is voor een willekeurige KE-redenering van lengte $n > 0$.

Een KE-redenering van lengte $n > 0$ wordt bekomen door een KE-redenering van lengte $n - 1$ uit te breiden met één stap. We mogen veronderstellen dat de inductiehypothese geldt voor die KE-redenering van lengte $n - 1$: dus, deze KE-redenering van lengte n-1 bevat een tak `t` en $\mathfrak{A}$ heeft een uitbreiding $\mathfrak{A}_1$ zodat alle formules in `t` geïnterpreteerd en voldaan zijn in $\mathfrak{A}_1$. Het volstaat dan te bewijzen dat de inductiehypothese ook geldt voor de KE-redenering bekomen na stap n.

In stap n worden één of twee nieuwe nodes toegevoegd onderaan een tak door toepassing van een uitbreidingsregel. Het is ofwel de tak `t` die uitgebreid wordt, ofwel een andere. Als een andere tak uitgebreid wordt dan `t`, dan is `t` nog altijd een volledige tak van de uitgebreide

KE-redenering van lengte n. Alle formules in `t` zijn voldaan in $\mathfrak{A}_1$ en dus voldoet de uitgebreide KE-redenering aan de stelling.

Stel dat het tak `t` zelf is die uitgebreid wordt op stap n. Dan voegt een uitbreidingsregel dus één of twee formules toe als kinderen onderaan `t`. Het volstaat dan te bewijzen dat minstens één ervan waar is in $\mathfrak{A}_1$, of eventueel in een uitbreiding van $\mathfrak{A}_1$.

Het bewijs is door een gevalstudie van de n-de uitbreidingstap. We bekijken een paar gevallen (maar niet allemaal).

- Veronderstel dat stap n een theorie-stap is die een zin $A \in T$ toevoegt onderaan `t`. Aangezien $\mathfrak{A}_1$ een uitbreiding is van $\mathfrak{A}$, en $\mathfrak{A}$ een model is van T, geldt dat $\mathfrak{A}_1 \models A$. Dus geldt de inductiehypothese voor n.

- Veronderstel dat stap n een gevalsonderscheid is voor een formule A. Dan wordt tak `t` uitgebreid tot 2 takken, één met onderdaan A, een andere met onderaan $\neg A$. Er geldt in elk geval dat $\mathfrak{A}_1 \models A$ ofwel $\mathfrak{A}_1 \not\models A$. In het tweede geval volgt uit de definitie van $\models$ dat $\mathfrak{A}_1 \models \neg A$. Dus is de inductiehypothese voldaan voor één van de twee takken.

- Veronderstel dat stap n een toepassing is van een propositionele propagatie-regel. Ofwel wordt een propagatie-regel $A \longrightarrow C$ toegepast, ofwel een propagatie-regel $A, B \longrightarrow C$. In het eerste geval komt A voor op de tak `t` en wordt C toegevoegd onderaan de tak. Uit de inductiehypothese volgt $\mathfrak{A}_1 \models C$. Aangezien achter elke propagatie-regel $A \longrightarrow C$ een tautologie schuilt $A \Rightarrow C$ volgt $\mathfrak{A}_1 \models C$. Een analoge redenering geldt voor propagatie-regels $A, B \longrightarrow C$. Dus is de inductiehypothese voor n voldaan.

- Veronderstel dat stap n een toepassing is van de Erule op $\exists x : A[x]$ in `t`. Onderaan `t` wordt dan de formule $A[c]$ toegevoegd met c een nieuw symbool. Het symbool c en dus ook $A[c]$ zijn niet geïnterpreteerd door $\mathfrak{A}_1$, maar $\mathfrak{A}_1$ kan als volgt uitgebreid worden. Omwille van de inductiehypothese geldt $\mathfrak{A}_1 \models \exists x : A[x]$. Dus bestaat $d \in D_{\mathfrak{A}_1}$ zodat $\mathfrak{A}_1[x : d] \models A[x]$. We breiden $\mathfrak{A}_1$ uit als $\mathfrak{A}_1[c : d]$ (dus c krijgt interpretatie d). Aangezien gold dat $\mathfrak{A}_1[x : d] \models A[x]$, geldt na de uitbreiding dat $\mathfrak{A}_1 \models A[c]$. Dus is de inductiehypothese voldaan voor de bekomen KE-redenering van lengte n.

- Arule: uit een formule $\forall x\colon A[x]$ wordt $A[t]$ afgeleid, met t een willekeurige term. Als t objectsymbolen bevat die niet door $\mathfrak{A}_1$ geïnterpreteerd zijn, dan breiden we $\mathfrak{A}_1$ uit door deze symbolen een waarde toe te kennen. De waarde mag willekeurig gekozen worden. Veronderstel dat $t^{\mathfrak{A}_1} = d$. Uit de inductiehypothese weten we dat $\mathfrak{A} \models \forall x\colon A[x]$. Hieruit volgt dat $\mathfrak{A}_1[x : d] \models A[x]$ en bijgevolg geldt $\mathfrak{A}_1 \models A[t]$. Opnieuw is de inductiehypothese voldaan voor n.

- Enz.

Q.E.D.

Oefening 3.5.2. *De enige soort stap waarvan we de correctheid niet bewezen hebben zijn de gelijkheidsstappen. Bewijs de correctheid ervan.*

Oefening 3.5.3. *Veronderstel dat de Erule bestaande constanten zou mogen "hergebruiken" in plaats van nieuwe te introduceren. Geef een voorbeeld die aantoont dat de bewijsmethode niet correct zou zijn. Toon aan in het bewijs waar het fout loopt.*

Stelling 3.5.1 (correctheid)**.** *Indien $T = \{A_1, \ldots, A_n\}$ een KE-bewijs van inconsistentie heeft, dan is T logisch inconsistent.*

Bew. Veronderstel dat T een KE-bewijs van inconsistentie heeft. Mocht T consistent zijn, en dus een model hebben, dan volgt uit Propositie 3.5.1 dat er een tak is en een structuur $\mathfrak{A}_1$ die alle formules op die tak waar maakt. Dat kan niet want elke tak bevat een contradictie, dus is T niet consistent. Q.E.D.

Definitie 3.5.3. Een KE-bewijs van een formule B uit een theorie T is hetzelfde als een KE-bewijs van inconsistentie van de theorie $T \cup \{\neg B\}$.

Stelling 3.5.2 (correctheid 2)**.** *Indien B een KE-bewijs uit theorie T heeft, dan is B een logisch gevolg van T.*

Dit volgt uit het feit dat $T \cup \{\neg B\}$ inconsistent is als en slechts als B een logische gevolg is van T.

De volgende vraag is of het omgekeerde ook geldt: als een theorie inconsistent is, bestaat er een KE-bewijs van inconsistentie voor?

Stelling 3.5.3 (Gödels volledigheidsstelling)**.** *Indien een eindige theorie T inconsistent is, dan bestaat er een KE-bewijs van de inconsistentie. Indien B een logisch gevolg is van T dan bestaat een KE-bewijs voor B uit T.*

Dit is een hele straffe stelling, waarvan men lang niet heeft geweten hoe deze te bewijzen. Kurt Gödel gaf een wiskundig bewijs van deze stelling in 1930. Gödel bewees dit voor een ander soort van formele bewijzen, maar zijn methode kan aangepast worden voor KE-bewijzen.

Strategie voor de KE-bewijsmethode Dankzij de stelling van Gödel weten we dat als T inconsistent is, er een KE-bewijs van inconsistentie bestaat. Dat betekent echter niet dat elke mogelijke constructie van een KE-bewijs voor T tot een dergelijke bewijs zal leiden. Op elk punt in de constructie van een KE-redenering moet gekozen worden tussen potentieel oneindig veel uitbreidingsregels. Bv. gevalsonderscheidingsregels, of Arule waarbij t willekeurig gekozen mag worden uit een oneindige verzameling van termen. Vele zijn nutteloos. Bv. we kunnen elke regel ∞ keer herhalen, wat nutteloos is.

Het laat zien dat bij een slechte keuze van de toegepaste regels, de methode onvolledig is (dus niet tot een bewijs leidt terwijl er wel een bewijs is) en ook tot inefficiëntie.

Omwille van de volledigheid en de efficiëntie van de KE-bewijsmethode is het belangrijk beperkingen in te stellen op de keuze van de toegepaste regels, liefst regels die potentieel van nut kunnen zijn om de inconsistentie te bewijzen. Dergelijke regels noemen we strategieregels. Hoe strenger de strategieregels, hoe minder keuzes, en hoe efficiënter de constructie van de KE-redenering. Maar de strategieregels mogen ook niet te streng zijn, anders is het mogelijk dat een bewijs niet gevonden kan worden.

Definitie 3.5.4. Een *strategie* bestaat uit een aantal strategieregels. Een strategieregel is een beperking opgelegd aan de toepassing van uitbreidingsregels.

Elke bijkomende strategieregel vermindert het aantal KE-redeneringen, en maakt de procedure om inconsistentie te bewijzen potentieel efficiënter.

Maar als je te veel strategieregels oplegt wordt de strategie onvolledig want dan kan de inconsistentie van sommige formules niet meer bewezen worden.

Definitie 3.5.5. We noemen een strategie S *volledig* indien geldt: elke KE-redenering voor een inconsistente theorie T kan uitgebreid worden tot een KE-bewijs van inconsistentie van T met strategie S.

Een nuttige strategie waarvan we sommige strategieregels al zagen is de volgende:

- hypothesenregel eerst toepassen;
- leid nooit een formule af die al op de tak zit;
- Erule : pas de rule maar 1 keer toe op een formule $\exists x\colon A$;
- propagatieregels en Erule toepassen van zodra mogelijk;
- Arule: introduceer geen nieuwe functiesymbolen en introduceer enkel een nieuwe constante als de theorie T en de tak hierboven geen enkele constante bevat;
- sluit de huidige tak vooralleer aan een andere tak te werken;
- gevalsonderscheiding voor formule A alleen maar toepassen als twee voorwaarden voldaan zijn:
  - andere uitbreidingsregels kunnen niet toegepast worden in de huidige tak;
  - A leidt tot propagatie: de tak bevat een formule B waarvan A een deelformule is en waarvoor er een propagatie-regel $A, B \longrightarrow C$ bestaat. Door een gevalsonderscheid toe te passen wordt deze propagatieregel van toepassing in 1 van de nieuwe takken.

Het kan bewezen worden dat deze strategie volledig is indien er geen gelijkheden (of ongelijkheden) of functiesymbolen in T voorkomen. We gaan daar niet verder op in. In het bijzonder is deze strategie volledig voor propositielogica.

Bewijzen van consistentie met KE-redeneringen Voor voldoende sterke strategieën is het mogelijk dat voor bepaalde takken geen enkele uitbreidingsregel mag toegepast worden hoewel ze niet gesloten zijn. Zo'n tak noemen we *volledig*.

Stelling 3.5.4. *Als een KE-redenering uit theorie T werd gebouwd met een volledige strategie S en deze redenering heeft een volledige maar niet gesloten tak volgens strategie S, dan is T consistent.*

Bew. Als T inconsistent zou zijn, dan zou volgens de definitie van volledige strategie, deze KE-redenering een uitbreiding moeten hebben tot een KE-bewijs van inconsistentie. Maar zo'n uitbreiding bestaat niet omwille van de volledige maar niet gesloten tak. Q.E.D.

M.a.w., de KE-methode met volledige strategie kan gebruikt worden om consistentie te bewijzen.

In de vorige paragraaf hebben we een strategie voorgesteld waarvan bewezen werd dat ze volledig is voor propositionele logica. Dus kan de KE-bewijsmethode onder deze strategie gebruikt worden om de consistentie te bewijzen van een propositionele zin of theorie. Sterker zelfs, uit de verzameling van literals in een volledige maar niet gesloten tak kan een model berekend worden van de theorie.

Voorbeeld 3.5.5. Veronderstel dat we willen bewijzen dat $\neg(P \Rightarrow Q) \Leftrightarrow (R \vee Q)$ logisch consistent is.

$\neg[\neg(P \Rightarrow Q) \Leftrightarrow (R \vee Q)]$	1. Theorie
Case A	
$R \vee Q$	2. Case A
$(P \Rightarrow Q)$	3. $\neg\Leftrightarrow$-prop(1,2)
Case A.A	
$\neg Q$	4. Case A.A
R	5. $\vee$-prop(2,4)
$\neg P$	6. $\Rightarrow$-prop(3,4)
Tak volledig; niet gesloten	

Op dit punt kan met onze strategie geen enkele uitbreidingsregel meer toegepast worden. Door het vinden van deze volledige maar niet gesloten tak, hebben we aangetoond dat de zin (1) niet inconsistent is en bijgevolg heeft het geen zin om verder te gaan. Bij inspectie van deze tak blijken we bovendien ook een structuur gevonden te hebben die voldoet aan (1): dat is de structuur $\mathfrak{A}$ waarin de literals in de tak waar zijn, namelijk $\neg P, \neg Q, R$.

In het algemeen kan de KE-methode gebruikt worden als een efficiënte methode om modellen van propositionele formules te construeren, efficiënter dan het construeren van een waarheidstabel.

Oefening 3.5.4. *Construeer een KE-redenering met een volledige doch niet gesloten tak voor de zin* $[(P \wedge R) \Rightarrow \neg Q] \wedge (Q \vee R)$.

Deductieve inferentie We hebben in dit hoofdstuk een aantal nieuwe redeneertaken bestudeerd:

Bereken of input T inconsistent is:
Input: theorie T
Output: **t** als T inconsistent is, anders **f**.

Bereken of $T \models A$:
Input: T, A
Output: **t** als $T \models A$, anders **f**.

Bereken of A inconsistent is met theorie T:
Input: T, A
Output: **t** als A inconsistent is met theorie T, anders **f**.

Definitie 3.5.6. Deze redeneervormen noemt men **deductief redeneren** of **deductieve inferentie** of kortweg **deductie**.

Merk op: elk van deze taken kan herleid worden tot elkaar. Dus als we één taak kunnen oplossen, dan kunnen we ze allemaal oplossen.

3.6 Samenvatting

In dit hoofdstuk werden fundamentele concepten bestudeerd: waarheid, consistentie, gevolg, equivalentie, bewijs, inferentie. Dit zijn essentiële concepten uit wiskunde en wetenschappen. In de context van logica kunnen deze concepten geformaliseerd en bestudeerd worden. Dat laat toe om een aantal fundamentele wetten van het denken te op te stellen en te bewijzen. We hebben ook een formele bewijsmethode gezien, de KE-bewijsmethode die correct en volledig is.

Predicatenlogica is vooral geïntroduceerd als tool om deductieve inferentie te bestuderen, maar we zagen al veel vormen van inferentie. In de volgende hoofdstukken zullen we hiervan toepassingen zien in de context van de informatica.

Hoofdstuk 4

Modelleren en redeneren in informatica-toepassingen

We bestuderen een aantal informatica-domeinen. Welke informatie is daarin beschikbaar? Hoe kunnen we deze informatie voorstellen in logica? Welke informaticaproblemen kunnen opgelost worden met die informatie? Welke vorm van redeneren hebben we daarbij nodig? Dit zijn de vragen die we in dit hoofdstuk bestuderen.

De droom hierachter is: stel de informatie in uw toepassingsgebied voor in logica, en laat uw probleem oplossen door een logisch redeneerssysteem!

We zullen ook een eigenschap zien die logica (en formele modelleertalen in het algemeen) onderscheidt van alle programmeertalen: de mogelijkheid om verschillende informaticaproblemen op te lossen op basis van *dezelfde* logische theorie.

Namelijk de mogelijkheid om informatie in het toepassingsgebied uit te drukken in logica, om vervolgens de softwareware problemen op te laten lossen door een logisch redeneerssysteem. Niet programmeren dus maar modelleren! Dit was het oorspronklijk doel van de Artificiële Intelligentie (1960), het is ook het doel van het domein van "Declarative Problem Solving" (1975). Het is de "dream, the quest, the holy grail" in het citaat van Bill Gates in Hoofdstuk 1.

Voor het examen Er komt gegarandeerd een vraag in het examen van de volgende vorm:

> Gegeven een theorie ABC of formule of structuur die de kennis over het applicatiedomein beschrijft (vb: wat een geldig lessenrooster is, of een structuur die een voorstel voor een lessenrooster is). Neem het volgende probleem : UVW (bv. nagaan of lessentrooster geldig is). Welke vorm van inferentie is nodig om UVW op te lossen gebruikmakende van ABC?

Het antwoord op een dergelijke vraag bevat de volgende elementen: (1) een beschrijving hoe de verschillende inputs voor het probleem omzetten in een logische input (bv. data over te geven lessen en studentengroepen wordt voorgesteld als een structuur), (2) benoemen van de inferentie (bv. deductie) en definitie van de vorm van inferentie, (3) uitleg hoe uit het antwoord van de inferentie-tool het antwoord op het oorspronkelijke probleem kan geextraheerd worden. (Bv. als het inferentie het model $\mathfrak{A}$ berekent, dan is het berekende uurrooster te vinden in de combinatie van de relaties $RoomOf^{\mathfrak{A}}$ en $TimeOf^{\mathfrak{A}}$. Dit is slechts een voorbeeld.)

4.1 Modelleren in logica: inleiding

De eerste stap bij het gebruik van logica in een toepassing is altijd dezelfde: het modelleren van informatie over het toepassingsgebied in een logische theorie.

Om dergelijke informatie te modelleren in formele talen zoals predikatenlogica zijn er twee grote stappen die ondernomen moeten worden:

1. het kiezen van een vocabularium Σ om "objecten" of "concepten" uit het toepassingsgebied te noteren;
2. het uitdrukken van een aantal informele eigenschappen over het toepassingsgebied in formele zinnen over Σ.

Laat ons dat even illustreren in de context van een eenvoudig domein: namelijk de verschillende familierelaties. Waarover we willen spreken is over kinderen en ouders, broers, zussen, neven en nichten, nonkels en tantes, grootouders en voorouders, echtgenoten en echtgenotes. We moeten ook een onderscheid maken tussen mannen en vrouwen. Misschien (maar niet noodzakelijk) willen we ook spreken over bepaalde specifieke mensen, zoals mezelf en mijn directe familieleden.

Nu kiezen we hiervoor een vocabularium waarin elk symbool overeenkomt met één van die relaties of personen. Moeilijk is dat hier niet, bv. $Man/1, Vrouw/1, OuderVan/2, GetrouwdMet/2$, $ZoonVan/2, DochterVan/2, BroerVan/2, OomVan/2, TanteVan/2, NeefVan/2, \ldots, Marc$, $Roel$, $Edgar, An, \ldots$.

We hebben niet al die symbolen nodig want het is duidelijk dat sommige relaties uitgedrukt of *gedefinieerd* kunnen worden in termen van andere. Bv. een zuster is een vrouw waarmee je twee gemeenschappelijke ouders hebt. We zouden kunnen kiezen voor een minimale verzameling van symbolen en alle andere relaties uitdrukken door middel van formules. Een minimale set van concepten zijn: $Man/1, Vrouw/1, Ouder/2, GetrouwdMet/2$.

Ofwel kunnen we symbolen introduceren voor elke relatie maar dan moeten we de logische verbanden tussen die symbolen uitdrukken, t.t.z. dan moeten we de *definities* van die symbolen uitdrukken met logische formules.

In elk geval, het resultaat van deze eerste fase is een vocabularium met formele symbolen waaraan we een specifieke betekenis hechten (bv. $Man/1$ betekent mannelijk). We associëren de formele symbolen met informele concepten uit het toepassingsdomein.

Wat betekent dit nu precies? Dat we de waarde kennen van elk symbool in de echte wereld? Zeker niet. Maar vanaf nu kunnen we elke logische zin interpreteren als een Nederlandse zin over het toepassingsdomein waarin we geïnteresseerd zijn. Omgekeerd, door het verband te leggen tussen de formele symbolen en de informele concepten kunnen we wiskundige structuren van ons vocabularium zien als wiskundige voorstellingen (of wiskundige abstracties) van *mogelijke* of *onmogelijke* toestanden van dat toepassingsdomein.

Bv. neem de structuur $\mathfrak{A}$ met domein $\{d\}$ en $OuderVan^{\mathfrak{A}} = GetrouwdMet^{\mathfrak{A}} = \{(d,d)\}$, $Man^{\mathfrak{A}} = Vrouw^{\mathfrak{A}} = \{d\}$. Wat stelt deze structuur voor? Een situatie met maar 1 mens, zowel man als vrouw, die zijn eigen ouder en echtgenoot is. Volkomen onmogelijk dus. Het doel van het modelleren zal zijn om een logische theorie op te stellen waarvan de modellen corresponderen met mogelijke toestanden van de wereld.

Bv. als we P kiezen als "de trein is te laat" en Q als "John mist zijn bus", dan komen de 4 wiskundige structuren van $\Sigma = \{P, Q\}$ overeen met de 4 situaties waarin de trein al dan niet te

laat is en John al dan niet zijn bus mist. Eén van die situaties is onmogelijk, namelijk die waarin de trein te laat is en John toch zijn bus niet mist. Deze onmogelijke situatie komt overeen met de structuur $\{P\}$. We zullen dit modelleren door $P \Rightarrow Q$. De modellen van deze formule komen exact overeen met de 3 mogelijke toestanden terwijl het de ene onmogelijke toestand uitsluit.

Een vocabularium met gekende betekenis van de symbolen in het probleemdomein wordt een *ontologie* genoemd.

Modelleren De volgende fase van het modelleren bestaat om bepaalde eigenschappen te modelleren als zinnen in het gekozen vocabularium.

In sommige toepassingen, bv. om te weten of een eigenschap voldaan is in een databank, moet een eigenschap geformaliseerd worden als een logische zin wiens informele betekenis overeenkomt met de eigenschap.

In andere toepassingen willen we een toepassingsdomein *axiomatiseren*. We willen een axioma of een theorie bouwen waarvan de modellen corresponderen met de toestanden van de wereld die we als mogelijk houden. Of in sommige gevallen, wanneer we een specificatie bouwen van een toekomstig systeem en de wereld die we modelleren dus nog niet bestaat, moeten de modellen van de theorie overeenstemmen met de gewenste toestanden van de toekomstige wereld.

Hoe dan ook, er moet er een serie van logische zinnen worden neergeschreven waarvan we weten dat ze waar zijn in het toepassingsdomein.

Maar waarheid in het toepassingsdomein is geen voldoende criterium. We willen bv. geen tautologieën uitschrijven want ze voegen niets toe aan de specificatie. Het criterium om een nieuwe zin toe te voegen aan de theorie is: (1) dat deze waar is in alle mogelijke toestanden van de wereld, en (2) dat de nieuwe zin onwaar is in minstens 1 model van de huidige theorie die we als onmogelijk beschouwen. M.a.w., de nieuwe zin sluit minstens 1 onmogelijke structuur uit.

Door dit proces te itereren bekomen we tenslotte een theorie waarvan de modellen overeen komen met wat we beschouwen als de mogelijke toestanden van het toepassingsdomein.

Dat is een kunst, het is niet altijd zo eenvoudig, maar je kunt het leren, en dat is één van de bedoelingen van deze cursus.

In het geval van de familierelaties moeten we twee zaken formaliseren: de eigenschappen van de ouderrelatie, en de definities van alle afgeleide concepten.

De volgende eigenschappen gelden:

- Iedereen is man of vrouw en niet beide:

$$\forall x\colon Man(x) \Leftrightarrow \neg Vrouw(x)$$

- Iedereen heeft twee ouders, een man en een vrouw:

$$\begin{aligned}\forall x\colon \exists y\colon \exists z\colon\ & OuderVan(y,x) \wedge Man(y) \wedge OuderVan(z,x) \wedge Vrouw(z) \wedge \\ & \forall u\colon OuderVan(u,x) \Rightarrow u = y \vee u = z\end{aligned}$$

Zijn we klaar? Zijn er nog onmogelijke werelden die toch voldoen aan alle axioma's? Neem bv. $\mathfrak{A}$:

- $D_{\mathfrak{A}} = \{a, b\}$

- $Man^{\mathfrak{A}} = \{a\}; Vrouw^{\mathfrak{A}} = \{b\}$
- $OuderVan^{\mathfrak{A}} = \{(a,a),(a,b),(b,a),(b,b)\}$
- $GetrouwdMet^{\mathfrak{A}} = \{(a,a),(b,a)\}$

Verifieer dat elk axioma voldaan is. Toch is dit een onmogelijke wereld en wel om verschillende redenen: a en b zijn hun eigen ouders, a is zijn eigen echtenoot, a is echtgenoot van twee personen, a is echtgenoot van b maar niet omgekeerd, etc. Elk verschillend antwoord op de vraag waarom deze wereld onmogelijk is geeft ons tergelijkertijd een nieuw axioma dat we kunnen toevoegen aan onze theorie.

- Niemand is zijn eigen ouder, grootouder, overgrootouder, etc.

$$\begin{aligned}
&\neg\exists x\colon OuderVan(x,x)\\
&\neg\exists x\colon \exists y\colon (OuderVan(x,y) \wedge OuderVan(y,x))\\
&\neg\exists x\colon \exists y\colon \exists z\colon (\ OuderVan(x,y) \wedge OuderVan(y,z) \wedge\\
&\qquad\qquad\qquad OuderVan(z,x))\\
&\dots
\end{aligned}$$

  Oneindig veel axioma's zijn nodig. lastig! Dit is een soort eigenschap die niet gemakkelijk uitgedrukt kan worden in FO.

- De partnerrelatie is symmetrisch en iedereen is getrouwd met hoogstens 1 andere persoon.

$$\begin{aligned}
&\forall x\colon \forall y\colon GetrouwdMet(x,y) \Rightarrow GetrouwdMet(y,x)\\
&\forall x\colon \forall y\colon \forall z\colon GetrouwdMet(x,y) \wedge GetrouwdMet(x,z) \Rightarrow y = z
\end{aligned}$$

Hoe zien de modellen van de bekomen theorie eruit? Zoals verwacht: mannen versus vrouwen, monogame symmetrische partnerrelaties, iedereen één vader en één moeder, en geen lussen in de ouderrelatie. Dat ziet er goed uit, behalve misschien het volgende. Omdat iedereen ouders heeft en er geen lussen in de ouderrelatie zitten, zijn er oneindig veel mensen en gaat de ouderrelatie oneindig ver terug in de tijd! Dit is niet erg realistisch. Dit is het gekende kip en ei probleem: iedereen heeft ouders, bijgevolg heeft iedereen een oneindig aantal voorouders.

Oefening 4.1.1. *Los het probleem op volgens het oud-testament, door te veronderstellen dat twee personen $Adam, Eva$ bestaan die geen ouders hebben en waarvan alle andere mensen afstammen. Pas de theorie aan.*

In termen van dit basisvocabularium kunnen vervolgens de andere familierelaties beschrijven als formules. Bv. de formule $A_{BroerVan}[x,y]$ die uitdrukt dat x een broer is van y.

$$Man(x) \wedge \neg(x = y) \wedge \exists z\colon (OuderVan(z,x) \wedge OuderVan(z,y) \wedge x \neq y))$$

Ofwel kunnen we een nieuw predikaat toevoegen, en dat expliciet definiëren.

$$\begin{aligned}
\forall x\colon \forall y\colon (\ &BroerVan(x,y) \Leftrightarrow\\
&Man(x) \wedge\\
&\exists z\colon (OuderVan(z,x) \wedge OuderVan(z,y) \wedge x \neq y))
\end{aligned}$$

In het eerste geval zullen we overal waar nodig uitdrukken dat een term t een broer is van term t' door $A_{BroerVan}[t,t']$. Kan knap vervelend worden. In het tweede geval schrijven we

$BroerVan(t, t')$. Bv. Jack heeft hoogstens 1 broer:

$$\forall x\colon \forall y\colon (BroerVan(x, Jack) \wedge BroerVan(y, Jack) \Rightarrow x = y)$$

versus

$$\begin{array}{l}\forall x\colon \forall y\colon (\ Man(x) \wedge \exists z\colon (OuderVan(z, x) \wedge OuderVan(z, Jack) \wedge x \neq Jack)) \wedge \\ \qquad Man(y) \wedge \exists z\colon (OuderVan(z, y) \wedge OuderVan(z, Jack) \wedge y \neq Jack)) \\ \qquad \Rightarrow x = y)\end{array}$$

Een ander voorbeeld is:

$$\begin{array}{l}\forall x\colon \forall y\colon (StiefMoederVan(x, y) \Leftrightarrow\ \ Vrouw(x) \wedge \neg OuderVan(x, y) \wedge \\ \qquad\qquad\qquad\qquad\qquad\qquad \exists z\colon (GetrouwdMet(x, z) \wedge OuderVan(z, y)))\end{array}$$

Oefening 4.1.2. *Laat zien dat de volgende symbolen volstaan om alle andere familierelaties te beschrijven:* $Man/1, Vrouw/1, OuderVan/2, GetrouwdMet/2$. *Schrijf een formule* $A[x, y]$ *met vrije variabelen* x, y *zodat* $\{(x, y) : A[x, y]\}$ *een relatievoorschrift is voor de volgende relaties:*

- x *is moeder, vader, stiefmoeder, schoonmoeder van* y*;*
- x *is zoon, dochter, stiefzoon van* y*;*
- x *is zuster, broer, halfbroer, stiefbroer van* y*;*
- x *is oom, tante van* y*;*
- x *is grootouder van* y*;*
- x *is voorouder van* y*.*

Oefening 4.1.3. *Herformuleer een paar van de logische formules van de vorige oefeningen gebruik makend van functiesymbolen* $Moeder/1 :, Vader : /1$ *in plaats van* $OuderVan/2$.

Oefening 4.1.4. *Een groep* $\langle G, e, +/2\rangle$ *is een structuur bestaande uit een verzameling, een neutraal element* e *en een binaire operator* $+$ *die aan de volgende eigenschappen voldoet:* $+$ *is een totale functie, associatief,* e *is links en rechts neutraal element en elk element heeft een invers element. Kies een vocabularium en schrijf deze eigenschappen uit in logica. Merk op dat je met DecaWorld van LogicPalet nu zelf groepen kunt genereren van hoogstens 10 elementen.*

Oefening 4.1.5. *Neem volgende zin:*

$$\forall x\colon \forall y_1\colon \forall y_2\colon \forall y_3\colon \left[\begin{array}{l} OuderVan(y_1, x) \wedge OuderVan(y_2, x) \wedge OuderVan(y_3, x) \\ \Rightarrow y_1 = y_2 \vee y_2 = y_3 \vee y_3 = y_1 \end{array}\right]$$

Wat betekent deze zin? Is deze zin logisch waar? Is deze zin waar in het toepassingsdomein?

4.2 Oplossen van informaticaproblemen met behulp van inferentie

Een basisidee van deze cursus is dat informatie kan uitgedrukt worden in logica, en dat informaticaproblemen en taken opgelost kunnen worden met behulp van die informatie door middel van bepaalde vormen van inferentie. In de cursus hebben we verschillende vormen van inferentie gezien. Hieronder geven we een algemene definitie van wat we bedoelen met inferentie.

Definitie 4.2.1. Een inferentie-probleem is een probleem met een input en een gewenste output. De input bestaat uit één of meerdere logische objecten. Hiermee bedoelen we

- een symbool of vocabularium
- een waarde uit een domein
- een structuur: een toekenning van waardes aan symbolen
- een logische uitdrukking: een term, een verzamelingenuitdrukking, een formule of zin, een theorie.

De output bestaat uit één of meerdere logische objecten die aan een bepaalde logische voorwaarde in termen van de invoer voldoen.

Deze algemene definitie illustreren we met een aantal voorbeelden die eerder in de cursus voorkwamen of die verderop uitgelegd worden.

- Een query-probleem:
  - Input: een structuur $\mathfrak{A}$, een zin A over het vocabularium van $\mathfrak{A}$.
  - Output: **t** indien $\mathfrak{A} \models A$, **f** indien $\mathfrak{A} \not\models A$.
- Het veralgemeende query-probleem:
  - Input: een structuur $\mathfrak{A}$, een verzamelingexpressie $\{(x_1, \ldots, x_n) : A\}$ over het vocabularium van $\mathfrak{A}$.
  - Output: $\{(x_1, \ldots, x_n) : A\}^{\mathfrak{A}}$, de relatie voorgesteld door de verzamelingexpressie in $\mathfrak{A}$.
- Een consistentie-probleem
  - Input: een zin of theorie T
  - Output: **t** indien T een model heeft, **f** indien T geen model heeft.
- Een modelgeneratie-probleem:
  - Input: een zin of theorie T
  - Output: één of meerdere of alle structuren $\mathfrak{A}$ zodat $\mathfrak{A} \models T$.
- Een modelexpansie-probleem:
  - Input: een zin of theorie T, een structuur $\mathfrak{A}_i$ die een deel van de symbolen van T interpreteert.
  - Output: een structuur $\mathfrak{A}$ zodat $\mathfrak{A} \models T$ en $\mathfrak{A}$ is een expansie van $\mathfrak{A}_i$, t.t.z., $\mathfrak{A}_i$ en $\mathfrak{A}$ hebben hetzelfde domein, en voor elk geïnterpreteerd symbool σ van $\mathfrak{A}_i$ geldt $\sigma^{\mathfrak{A}_i} = \sigma^{\mathfrak{A}}$.
- Een deductie-probleem:

- Input: een theorie T, een logische zin A
- Output: $\mathbf{t}$ indien $T \models A$; $\mathbf{f}$ indien $T \not\models A$.

Voor de meeste van deze problemen en vormen van inferentie hebben we reeds eerder voorbeelden in de cursus gezien. Bv. in Hoofdstuk 2 zagen we het query-probleem. Theoremprovers zijn gespecialiseerd in het oplossen van deductieproblemen. Het grafenkleuringsprobleem (Voorbeeld 2.6.2) is een voorbeeld van een modelexpansie-probleem: de input is een structuur $\mathfrak{A}_i$ die een grafe en een domein van kleuren specifieert. De output is een structuur $\mathfrak{A}$, of meer precies de relatie $C^{\mathfrak{A}}$ waarbij $\mathfrak{A}$ een model is van een theorie en een uitbreiding van $\mathfrak{A}_i$.

Een basisidee van deze cursus is dat we in een logische theorie of formule A de relevante informatie van een applicatiedomein kunnen vastleggen. Vaak of misschien wel altijd kan een informaticaprobleem in dit domein herleid worden tot een inferentieprobleem over A. Door middel van een inferentie-programma zoals SPASS of IDP die gebruikt worden in LogicPalet kan dit probleem dan opgelost worden. In dit hoofdstuk zien we hier vele voorbeelden van.

4.3 Databanken revisited

Databanken vormen een van de belangrijkste technologieën van de software-industrie. Databanken dienen om *informatie* op te slaan en erover te redeneren. Aangezien logica ook over informatie en redeneren gaat, is het nuttig om databanken te bekijken vanuit logisch standpunt.

We zagen eerder een sterk verband tussen databanken DB en structuren $\mathfrak{A}$ in logica:

- Informeel: een databank dient om een stand van zaken voor te stellen. Bv. de stand van zaken van docenten, vakken, studenten en examens zoals in de studentendatabank in Figuur 2.1. In logica stelt een structuur $\mathfrak{A}$ een stand van zaken voor.
- Formeel: een databank bevat namen voor tabellen en voor de entiteiten in de tabellen. Dit komt overeen met een logisch vocabularium Σ bestaande uit predicaatsymbolen P/n en objectsymbolen C.
- Formeel: een databank bestaat uit tabellen. Een logische structuur $\mathfrak{A}$ bevat relaties $P^{\mathfrak{A}}$, t.t.z., wiskundige tabellen. De waarde van objectsymbolen c kunnen we kiezen als $c^{\mathfrak{A}} = c$ zodat een Herbrand structuur bekomen wordt.

We beschouwen een databank dus als een Herbrand structuur $\mathfrak{A}$ en bestuderen verschillende modelleer-aspecten:

- Databanken evolueren in de tijd. Op elk moment moet de databank voldoen aan voorwaarden om *zinvol* te zijn. Deze voorwaarden noemen we *integriteitsbeperkingen.* We onderzoeken hun rol via logica.
- Gebruikers willen vragen stellen aan de databank. Ze modelleren een bepaalde eigenschap of een verzamelingenexpressie en vragen het databanksysteem om deze te evalueren. Men noemt dit een query. Het opstellen van een query is een modelleertaak.
- Een databank bevat informatie, net zoals een logisch theorie. We stellen een databank voor als een structuur, maar waarom niet als een logische theorie? We onderzoeken dit om te begrijpen welk soort informatie aanwezig is in een databank in vergelijking met logische theorieën.

4.3.1 Zinvolle en zinloze databanken en Integriteitsbeperkingen.

In elk domein moet een *zinvolle* databank voldoen aan bepaalde voorwaarden. Voor een studentendatabank:

- Voor elk koppel (x, y, z) in de Grade tabel, moet x een student zijn, y een vak en z een score.
- Een vak mag geen voorkennis zijn van zichzelf.
- enzovoort.

De eigenschappen die voldaan moeten zijn om zinvol te zijn noemen we *integriteitsbeperkingen*. We noemen een databank *zinvol* indien deze voldoet eraan, en anders *zinloos*.

Zij Σ het databank vocabularium, zij IB een logische theorie over Σ bestaande uit alle integriteitsbeperkingen.

Definitie 4.3.1. We zeggen dat een databankstructuur $\mathfrak{A}$ **zinvol** is indien $\mathfrak{A}$ voldoet aan IB.

Een zinloze databank kan onmogelijk de stand van zaken van het domein voorstellen. Dat betekent niet dat alle data in een zinloze databank waardeloos is. Best mogelijk dat vele vragen nog correct beantwoord kunnen worden. Maar in een zinloze databank kan men foute antwoorden krijgen op queries. Men zal daarom altijd proberen om een databank zoveel mogelijk aan die integriteitsbeperkingen te laten voldoen. Er kunnen bv. regelmatige tests gebeuren of dit het geval is, of men zal bij elke update van de databank ervoor waken dat deze geen fouten introduceert.

Om een databanksysteem toe te laten te controleren of de integriteitsbeperkingen voldaan zijn moeten ze gemodelleerd worden in een formele taal. In het geval van de studentengegevensbank hebben sommige integriteitsbeperkingen te maken met het welgetypeerd zijn van de relaties. Deze type-beperkingen kunnen geformuleerd worden als:

- $\forall x\colon \forall y\colon Instructor(x, y) \Rightarrow Lecturer(x) \wedge Course(y)$
- $\forall x\colon \forall y\colon Enrolled(x, y) \Rightarrow Student(x) \wedge Course(y)$
- ...

Andere integriteitsbeperkingen drukken uit dat bepaalde types disjunct zijn: Disjunctheidsbeperkingen:

- $\neg\exists x\colon Lecturer(x) \wedge Student(x)$
- $\neg\exists x\colon Lecturer(x) \wedge Score(x)$
- ...

Andere integriteitsbeperkingen van de studentendatabank zijn als volgt:

- *Geen enkel vak is (directe) nodige voorkennis voor zichzelf.*

$$\neg\exists w\colon Prerequ(w, w)$$

- *Niemand kan meer dan één score hebben voor hetzelfde vak.*

$$\neg\exists x\colon \exists w\colon \exists s\colon \exists t\colon (Grade(x, w, s) \wedge Grade(x, w, t) \wedge s \neq t)$$

– *Iedereen die ingeschreven is voor een vak moet geslaagd zijn voor alle vakken die (directe) nodige voorkennis zijn voor dat vak.*

$$\forall x\colon \forall w\colon (Enrolled(x,w) \Rightarrow \forall t\colon (Prerequ(t,w) \Rightarrow (\exists s\colon (Grade(x,t,s) \wedge PassingGrade(s)))))$$

Een gegevensbank wordt aangepast en evolueert. Daarbij kunnen er fouten optreden. Om deze tegen te gaan zijn bepaalde redeneertaken vereist.

Om na te gaan of een databank(structuur) $\mathfrak{A}$ zinvol is, moet geëvalueerd worden of $\mathfrak{A}$ voldoet aan IB.

> Evaluatie-inferentie: Bereken of IB voldaan is in $\mathfrak{A}$:
> Input: $\mathfrak{A}, IB$
> Output: $\mathbf{t}$ indien $\mathfrak{A} \models IB$, anders $\mathbf{f}$.

Mogelijks is dit een zeer kostelijke operatie, bv. bij een enorm grote databank (bv. terabytes). Slimme inferentiemethodes proberen het optreden van fouten te vermijden tijdens het updaten van de databank. Het idee is gebaseerd op het feit dat bij een update van de databank $\mathfrak{A}$ er vaak weinig verandert. Als we ervan uitgaan dat IB voldaan was in $\mathfrak{A}$, moet er vaak niet zoveel gecontroleerd worden om te kunnen garanderen de aangepaste databank $\mathfrak{A}'$ ook voldoet aan IB. Door hiermee rekening te houden kunnen veel berekeningen vermeden worden. Bv. als we een koppel (s, c, sc) toevoegen aan $Grade^{\mathfrak{A}}$, dan hoeven we enkel de types van s, c, cs controleren, en verder of de databank geen score bevat voor student s en cursus c.

Een andere methode is om alle updates te implementeren in transactie-programma's en vooraf te bewijzen dat het uitvoeren van een transacie de integriteitsbeperkingen zal bewaren. Dat vereist verificatie van de transacties. We zullen dit later bespreken.

Als een databankstructuur $\mathfrak{A}$ niet voldoet aan IB moeten $\mathfrak{A}$ gecorrigeerd worden. Daarbij is een andere vorm van inferentie vereist:

> Revisie-inferentie: pas een structuur $\mathfrak{A}$ aan zodat ze aan een IB voldoet.
> Input: $\mathfrak{A}, IB$
> Output: $\mathfrak{A}'$ zodat $\mathfrak{A}' \models IB$ en $\mathfrak{A}'$ is een "kleine" aanpassing van $\mathfrak{A}$.

Dat is een probleem dat niet eenvoudig geautomatiseerd kan worden aangezien er bijna altijd meerdere oplossingen bestaan, wat menselijke tussenkomst nodig maakt. Bv. er zijn twee scores voor Flo voor het vak CS230, en de docent moet beslissen welke de juiste is.

4.3.2 Queries modelleren en oplossen

Een databank $\mathfrak{A}$ is een abstracte voorstelling van de huidige stand van zaken van het toepassingdomein. Gebruikers gebruiken de databank om informatie te weten te komen over die stand van zaken. Dit gebeurt door een "query" op te stellen en deze te laten oplossen door het databanksysteem.

De eerste stap is het formuleren van de query. Het opstellen van een query is een *modelleerstap*. Een query is een uitspraak A, of een relatievoorschrift $\{(x_1, \ldots, x_n) : A\}$. Het stelt een vraag

voor: is A waar in $\mathfrak{A}$, of wat is de relatie voorgesteld door $\{(x_1, \ldots, x_n) : A\}$ in $\mathfrak{A}$? De oplossing van de query is dan de waarde $A^{\mathfrak{A}}$ of $\{(x_1, \ldots, x_n) : A\}^{\mathfrak{A}}$ van die query.

Vervolgens wordt het databanksysteem opgeroepen. Deze implementeert de volgende soorten inferentie:

Evaluatie-inferentie van formules: Bereken of A voldaan is in $\mathfrak{A}$:
Input: $\mathfrak{A}, A$
Output: **t** indien $\mathfrak{A} \models A$, anders **f**. Anders, en korter gezegd, het antwoord is $A^{\mathfrak{A}}$.

Evaluatie-inferentie van verzamelingenuitdrukking: Bereken de relatie voorgesteld door een relatievoorschrift in structuur $\mathfrak{A}$ met eindig domein:
Input: $\mathfrak{A}, \{(x_1, \ldots, x_n) : A\}$
Output: $\{(x_1, \ldots, x_n) : A\}^{\mathfrak{A}}$.

Correcte vertalingen. Een gebruiker met een vraag over de stand van zaken beschreven door de databank DB, dient een query op te stellen. Als men wil weten of een bepaalde eigenschap geldt in DB, is de query een logische zin A. Als de gebruiker een relatie tussen entiteiten wil weten is de query een verzamelingenuitdrukking (of relatievoorschrift) $\{x_1, \ldots, x_n : A\}$. Na het opstellen van de query, roept de gebruiker het databanksysteem op om de waarde van de query in DB te berekenen.

Bepalen of een query een *correcte vertaling* van een Nederlandse query is, komt neer op de vraag of de *informele semantiek* van de query dezelfde gedachte weergeeft als de Nederlandse query. In de inleiding van het hoofdstuk (en de sectie over pragmatiek) bleek al dat een correcte vertaling van een Nederlandse query vaak niet zomaar een eenvoudige vertaalslag is. Denk bv. aan hoe uit te drukken : "er is precies 1 student". [1] Het vereist een goed begrip van de betekenis van de Nederlandse query en van de betekenis van de logische symbolen om de vertaling te doen. En ook de integriteitsbeperkingen spelen een rol.

De gebruiker kent de inhoud niet van DB, anders zou men geen queries hoeven te stellen. Daarom moet de query zodanig opgesteld zijn dat het een correct antwoord heeft, onafhankelijk van de concrete inhoud van DB. Hoogstens zal men erop rekenen dat DB voldoet aan de theorie IB van de integriteitsbeperkingen ($DB \models IB$). Dit leidt tot de volgende definitie van *correcte vertaling*.

Definitie 4.3.2. Een logische query (een zin of een verzamelingenuitdrukking) is een *correcte vertaling* van een Nederlandse query wanneer het antwoord op de logische query in **elke zinvolle databank** correct is, dus in elk model van IB.

[1] Sommige queries kunnen zelfs niet uitgedrukt worden in predikatenlogica, zoals "hoeveel studenten zijn geslaagd voor het vak M100?". Ze kunnen wel uitgedrukt worden in de huidige databanktalen die expressiever zijn.

Bv. neem de query of Sam ingeschreven is voor minstens één vak. Een correcte manier om dit uit te drukken is:

$$\exists v : Course(v) \wedge Enrolled(Sam, v)$$

Maar ook de volgende query lijkt ok, hoewel het type van v hierin niet uitgedrukt wordt:

$$\exists v : Enrolled(Sam, v)$$

Nochtans zijn deze twee zinnen niet logisch equivalent.

Propositie 4.3.1. *Deze zinnen zijn niet logisch equivalent.*

Bewijs. Om dit aan te tonen volstaat het om een structuur $\mathfrak{A}$ te vinden waarin de zinnen een verschillende waarheidswaarde hebben. Een heel kleine structuur is deze $\mathfrak{A}$:

- $D_{\mathfrak{A}} = \{a\}$;
- $Sam^{\mathfrak{A}} = a$;
- $Course^{\mathfrak{A}} = \emptyset$;
- $Enrolled^{\mathfrak{A}} = \{(a, a)\}$.

Het is een eenvoudige oefening om te berekenen dat de eerste zin onwaar is in $\mathfrak{A}$ en de tweede zin waar is. Doe dit zelf. ■

Oefening 4.3.1. *Zijn er realistische situaties waarin beide queries een verschillende waarheidswaarde zouden hebben? Zeker! Neem een variant $\mathfrak{A}$ van de studentenbank waarin Sam niet ingeschreven is voor een cursus. Veronderstel dat per ongeluk het koppel (Sam, Sam) wordt toegevoegd aan $Enrolled^{\mathfrak{A}}$ (een bug!). Evalueer de beide formules in de resulterende $\mathfrak{A}$.*

Men verwacht dat de twee formules hetzelfde antwoord geven in de databank omdat we verwachten dat de databank goed getypeerd is en voldoet aan de type-integriteitsbeperking:

$$\forall st : \forall v : Enrolled(st, v) \Rightarrow Student(st) \wedge Course(v)$$

Want dan is een object waarvoor Sam "enrolled" is een vak.

We zien dat een gebruiker zijn query op meerdere niet-equivalente manieren kan opstellen en toch steeds het correcte antwoord zal krijgen. Tenminste, als de databank voldoet aan de theorie IB van integriteitsbeperkingen.

Definitie 4.3.3. We zeggen dat twee formules A, B equivalent zijn volgens IB indien $IB \models A \Leftrightarrow B$.

Dus, A is equivalent met B volgens IB indien in elk model van IB, de waarheidswaarde van A en B gelijk is. Als een query A een correcte vertaling van een Nederlanse query is, dan is elke equivalente query B *volgens* IB ook een correcte vertaling. Dit is het geval voor de twee queries hierboven.

In bovenstaand geval volgt de equivalentie uit de type-integriteitsbeperking. Maar het kan een gevolg zijn van elke integriteitsbeperking, of een combinatie ervan.

Bv. veronderstel dat Sam ingeschreven is voor $M200$. Neem de vraag of Sam een AAA behaalde voor elke vak dat voorkennis is van $M200$. Hier zijn twee zinnen:

$$\forall v : Prerequ(v, M200) \Rightarrow (\forall sc : Grade(Sam, v, sc) \Rightarrow sc = AAA)$$
$$\forall v : Prerequ(v, M200) \Rightarrow (\exists sc : Grade(Sam, v, sc) \wedge sc = AAA)$$

Deze zinnen zijn niet logisch equivalent. Maar ze zijn wel equivalent volgens een combinatie van twee integriteitsbeperkingen: dat er hoogstens 1 graad is per vak en student, en dat elke student die ingeschreven is voor een vak v, geslaagd is voor elk vak dat voorkennis is van v. Probeer dat in te zien.

Oefening 4.3.2. *Toon aan dat deze formules niet equivalent zijn? Kies een zo klein mogelijke structuur waarin beide zinnen ongelijke waarde hebben. Argumenteer dat ze wel equivalent zijn volgens IB.*

Expliciet en impliciet getypeerde queries In natuurlijke taal kwantificeren we nooit over het hele universum maar altijd over een deelklasse of een deeltype. De wereld is immers veel te heterogeen om uitspraken te maken die over alle objecten gelden. Predikatenlogica is evenwel ongetypeerd en de quantoren gaan over het volledige universum. We introduceren daarom een modelleertechniek om de types expliciet te modelleren.

We veronderstellen dat het universum gepartitioneerd kan worden in homogene types, en dat elk type voorgesteld wordt door een unair type-predicaat $T/1$. Bv. in de studentendatabank waren dit $Lecturer/1, Course/1, Student/1, \ldots$.

Definitie 4.3.4. Een *expliciet getypeerde logische formule* is van de vorm waarin elke gekwantificeerde deelformule eruit ziet als $\exists x\colon (T(x) \wedge A)$ of $\forall x\colon (T(x) \Rightarrow A)$, met T een type-predicaat. We noemen T het type van de variabele x. Een *expliciet getypeerde n-voudige query* is een relatievoorschrift van de vorm $\{(x_1, \ldots, x_n) : T_1(x_1) \wedge \cdots \wedge T_n(x_n) \wedge A\}$, met A een expliciet getypeerde formule.

De expliciet getypeerde zin A' van een logische zin A wordt bekomen door bij elke quantor $\exists x : \ldots$ en $\forall x : \ldots$ het gepaste type predicaat te inserteren: $\exists x : T(x) \wedge \ldots$ and $\forall x : T(x) \Rightarrow \ldots$.

Elke zinvolle formule (of query) waarvan de types van alle variabelen duidelijk zijn, kan geformuleerd worden als een expliciet getypeerde formule. Dit geeft ons een methodiek om op correcte manier de relevante types in aanmerking te nemen in formules.

Voorbeeld 4.3.1. Voorbeelden van expliciet getypeerde formules en queries over de studentendatabank $\mathfrak{A}$ zijn:

(1) $\exists c\colon (Course(c) \wedge Grade(Sam, c, AAA))$
(2) $\forall y\colon [Lecturer(y) \Rightarrow \exists w\colon (Course(w) \wedge Instructor(y, w) \wedge Enrolled(Jack, w))]$
(3) $\exists x : Course(x) \wedge Prerequ(x, x)$
(4) $\exists x : Course(x) \wedge \neg Prerequ(x, x)$
(5) $\{x : Course(x) \wedge \neg Prerequ(x, x)\}$

Vaak echter gebruikt men ook niet expliciet getypeerde zinnen en queries. Soms zijn ze correct, soms niet.

Voorbeeld 4.3.2. De volgende vereenvoudigde zinnen zijn equivalent met de gelijkgenummerde expliciet getypeerde zinnen, volgens IB:

(1) $\exists c\colon Grade(Sam, c, AAA)$
(2) $\forall y\colon [Lecturer(y) \Rightarrow \exists w\colon Instructor(y, w) \wedge Enrolled(Jack, w)]$
(3) $\exists x : Prerequ(x, x)$

Andere niet. De volgende zinnen en verzamelingenexpressies zijn niet equivalent met de gelijkgenummerde zinnen, volgens IB:

(2) $\forall y\colon \exists w\colon Course(w) \wedge Instructor(y, w) \wedge Enrolled(Jack, w)$
(4) $\exists x : \neg Prerequ(x, x)$
(5) $\{x : \neg Prerequ(x, x)\}$

Oefening 4.3.3. *Kun je verklaren waarom wel $Course(w)$ maar niet $Lecturer(y)$ geschrapt kan worden uit zin (2) van Voorbeeld 4.3.1?*

Daarover gaat de rest van deze sectie.

Impliciet getypeerde formules en type-insertie (te kennen voor het examen)

Zij gegeven een vocabularium Σ bestaande uit een aantal type predicaten $T/1$ en gewone predicaten. Veronderstel dat IB voor elk gewoon predicaat P/n een type-integriteitsbeperking bevat:

$$\forall x_1 \ldots \forall x_n : P(x_1, \ldots, x_n) \Rightarrow T_1(x_1) \wedge \cdots \wedge T_n(x_n)$$

Deze theorie bepaalt wat de expliciete typering is van alle formules zoals beschreven in Definitie 4.3.4.

Definitie 4.3.5. We noemen een formule A **impliciet getypeerd** volgens IB als A en zijn expliciet getypeerde formule A_t equivalent zijn volgens IB.

Dus, om na te gaan of A impliciet getypeerd is, bewijst men dat

$$IB \models A \Leftrightarrow A_t$$

waarbij A_t de expliciete typering is van A zoals beschreven in Definitie 4.3.4.

Oefening 4.3.4. *Gebruik de KE methode of LogicPalet om te bewijzen dat uit*

$$\{\forall x \forall y : Prerequ(x, y) \Rightarrow Course(x) \wedge Course(y)\}$$

logisch volgt dat

$$(\exists x : Prerequ(x, x)) \Leftrightarrow (\exists x : Course(x) \wedge Prerequ(x, x))$$

Verifieer ook dat dezelfde methode niet werkt om te bewijzen dat

$$(\exists x : \neg Prerequ(x, x)) \Leftrightarrow (\exists x : Course(x) \wedge \neg Prerequ(x, x))$$

Er is een eenvoudige methode om na te gaan of een niet-expliciet getypeerde query equivalent is met zijn expliciet getypeerde formule volgens IB. Deze methode noemen we **type-insertie**.

1. Vervang in de formule A atomen
$$P(t_1, \ldots, t_n)$$
door
$$T_1(t_1) \wedge \ldots T_n(t_n) \wedge P(t_1, \ldots, t_n)$$
of eventueel een deel van deze type atomen. Hier worden de types van bepaalde argumenten van P/n geïnserteerd. Deze transformatie noemen we *type-insertie*. Deze stap is equivalentiebewarend volgens IB. (Probeer in te zien waarom).

2. Pas equivalentie-bewarende transformaties toe om de resulterende formule te herschrijven tot een expliciet getypeerde formule. Als dit lukt is A impliciet getypeerd. Anders is de query A niet goed getypeerd.

Voorbeeld 4.3.3. Neem de zin

$$\exists c\colon Grade(Sam, c, AAA)$$

Door type-insertie bekomen we:

$$\exists c\colon Score(c) \wedge Grade(Sam, c, AAA)$$

Deze zin is expliciet getypeerd en equivalent met de vorige volgens IB. Dus, de oorspronkelijke zin was impliciet getypeerd.

Voorbeeld 4.3.4. Neem de zin
$$\exists x : \neg Prerequ(x, x)$$

Door type-insertie bekomen we:

$$\exists x : \neg(Course(x) \wedge Prerequ(x, x))$$

hetgeen equivalent is met:

$$\exists x : \neg Course(x) \vee \neg Prerequ(x, x)$$

Deze zin is niet expliciet getypeerd. Sterker, het is het soort zin waarvan we weten dat het nooit correct is.

Voorbeeld 4.3.5. Neem verzamelingenuitdrukking

$$\{x : \neg Prerequ(x, x)\}$$

Door type-insertie bekomen we:

$$\{x : \neg(Course(x) \wedge Prerequ(x, x))\}$$

hetgeen equivalent is met

$$\{x : \neg Course(x) \vee \neg Prerequ(x, x)\}$$

Ook dit is geen expliciet getypeerde zin.

Voorbeeld 4.3.6. Neem de complexere zin is:

$$\forall y\colon \exists w\colon Instructor(y,w) \wedge Enrolled(Jack,w)$$

Door type-insertie voor y en w in $Instructor(y,w)$ bekomen we:

$$\forall y\colon \exists w\colon Lecturer(y) \wedge Course(w) \wedge Instructor(y,w) \wedge Enrolled(Jack,w)$$

De kwantificatie over w kan men naar binnen doorschuiven om te bekomen:

$$\forall y\colon Lecturer(y) \wedge \exists w\colon Course(w) \wedge Instructor(y,w) \wedge Enrolled(Jack,w)$$

De uitdrukking $\forall y : Lecturer(y) \wedge \ldots$ is niet expliciet getypeerd op de correcte manier. Het is één van die totaal verkeerde zinnen, hier omdat geïmpliceerd wordt dat alle elementen van het domein lesgevers zijn. De oorspronkelijke zin was dus niet impliciet getypeerd.

Voorbeeld 4.3.7. Neem de verwante zin:

$$\forall y\colon Lecturer(y) \Rightarrow \exists w\colon (Instructor(y,w) \wedge Enrolled(Jack,w))$$

Door type-insertie voor w in $Instructor(y,w)$ bekomen we:

$$\forall y\colon Lecturer(y) \Rightarrow \exists w\colon (Course(w) \wedge Instructor(y,w) \wedge Enrolled(Jack,w))$$

Deze zin is expliciet getypeerd. Dus was de oorspronkelijke zin correct impliciet getypeerd.

Oefening 4.3.5. *Bepaal of volgend formules/queries impliciet getypeerd zijn, en zo niet voeg typering toe:*

- *Er is een student die geen enkel examen deed:*

$$\exists s\colon \forall v\colon \forall c\colon \neg Grade(x,v,c)$$

- *Jack volgt les bij elke docent.*

$$\forall y\colon \exists w\colon (Instructor(y,w) \wedge Enrolled(Jack,w))$$

Oefening 4.3.6. *Ga na met type insertie of de volgende query goed getypeerd is:*

$$\forall y\colon [\exists w\colon Instructor(y,w) \Rightarrow \exists w\colon (Instructor(y,w) \wedge Enrolled(Jack,w))]$$

Oefeningen

Oefening 4.3.7. *Formuleer volgende queries en integriteitsbeperkingen voor de StudentenData-bank. Let op of uw query impliciet getypeerd is, en zoniet, voeg expliciete typering toe.*

1. *Alle studenten die ingeschreven zijn voor CS230, zijn geslaagd voor dat vak?*
2. *Er is een student die ingeschreven is voor alle vakken die Sue doceert? Uw query moet ook juist zijn als Sue geen enkel vak doceert.*
3. *Er is een student die ingeschreven is voor meer dan twee vakken?*
4. *Ray en Hec zijn docenten voor CS230 en tevens de enigen hiervoor?*

5. *Gevraagd: elke student die geslaagd is voor alle door hem afgelegde vakken? (Opgepast: een student kan meerdere scores voor een vak hebben! De hoogste telt.)*

6. *Gevraagd: alle docenten die meer dan één vak doceren.*

7. *Integriteitsbeperking: Niemand mag ingeschreven zijn voor een vak waarvoor hij/zij al geslaagd is.*

8. *Integriteitsbeperking: Iedereen die ingeschreven is voor een vak moet geslaagd zijn voor alle vakken die (directe) nodige voorkennis zijn voor dat vak.*

Oefening 4.3.8. *1. Toon aan door te rekenen met logische equivalenties dat de volgende twee queries logisch equivalent zijn:*

$$\{x : \exists y\colon [Enrolled(x,y) \vee \exists z\colon Grade(x,y,z)] \wedge \forall y\colon \forall z\colon [Grade(x,y,z) \Rightarrow PassingGrade(z)]\} \tag{4.1}$$

$$\{x : \forall z\colon \exists t\colon [\ [\exists y\colon Enrolled(x,y) \vee \exists s\colon Grade(x,s,t)] \wedge [\exists y\colon Grade(x,y,z) \Rightarrow PassingGrade(z)]]\} \tag{4.2}$$

2. *Verifieer dat ook met AskSpass van LogicPalet. (Dit stelt geen probleem alhoewel x een vrije variabele is.)*

3. *Formuleer de query 4.1 zo kort mogelijk in het Nederlands.*

4. *Formuleer volgende query zo kort mogelijk in het Nederlands:*

$$\{x : [(\exists y\colon Enrolled(x,y)) \vee \exists y\colon \exists z\colon Grade(x,y,z)] \wedge \forall y\colon [\exists z\colon Grade(x,y,z) \Rightarrow \exists z\colon (Grade(x,y,z) \wedge PassingGrade(z))\} \tag{4.3}$$

5. *Toon aan door een redenering dat (4.1) en (4.3) logisch equivalent zijn volgens de volgende integriteitsbeperking*

$$\neg\exists x\colon \exists w\colon \exists s\colon \exists t\colon [Grade(x,w,s) \wedge Grade(x,w,t) \wedge s \neq t]. \tag{4.4}$$

(Spass verifieert dit in minder dan 0,1 seconde!)

Oefening 4.3.9. *Voor queries met geneste kwantoren zijn er soms meerdere redeneerstappen nodig om ze na de type-insertie om te zetten in een equivalente expliciet getypeerde formule. Neem de query of er studenten bestaan die van elke docent een cursus volgen:*

$$\exists x\colon \forall y\colon [\exists w\colon Instructor(y,w) \Rightarrow \exists w\colon (Instructor(y,w) \wedge Enrolled(x,w)))]$$

Doe de type-insertie en toon aan dat je die kunt omzetten naar een expliciet-getypeerde query.

Opmerking 4.3.1. Van deze sectie moet je onthouden wat expliciet en impliciet getypeerde queries zijn, en je moet kunnen herkennen door middel van de type-insertie of een query impliciet getypeerd is.

4.3.3 Een databank als een logische theorie

Een databank DB, bv. de studentendatabank Figuur 2.1, is een voorstelling van de stand van zaken van het toepassingsdomein. We hebben DB voorgesteld als een Herbrand-structuur $\mathfrak{A}_{DB}$, omdat structuren nu eenmaal standen van zaken voorstellen. Anderzijds is een databank natuurlijk ook een vat vol informatie. Aangezien logische formules en theorieën dienen om

informatie voor te stellen, rijst de vraag hoe de informatie in DB door een logische theorie $Theo(DB)$ voor te stellen.

Wat is het correctheidscriterium voor $Theo(DB)$? Wanneer is $Theo(DB)$ een *correcte logische vertaling* van DB? Wanneer kunnen we zeker zijn dat $Theo(DB)$ *alle* informatie uit DB voorstelt? Dat is wanneer $Theo(DB)$ ook diezelfde unieke stand van zaken bepaalt als de databank. Dat is wanneer $\mathfrak{A}_{DB}$ het enige model van $Theo(DB)$ is. Dus wanneer $\mathfrak{A} \models Theo(DB)$ asa $\mathfrak{A} = \mathfrak{A}_{DB}$.

Evenwel, dit is onmogelijk want we weten dat als een structuur $\mathfrak{A}$ een model is van een logische zin of theorie, dan zal elke isomorfe structure $\mathfrak{B}$ ook een model zijn. Er zijn geen theorieën die maar 1 model hebben. Dus moet het correctheidscriterium hergeformuleerd worden als volgt:

Definitie 4.3.6. $Theo(DB)$ is een correcte logische vertaling van DB asa elk model $\mathfrak{A}$ van $Theo(DB)$ isomorf is met $\mathfrak{A}_{DB}$.

De volgende stelling geeft aan dat als $Theo(DB)$ een correcte logische vertaling is van DB, dan kunnen queries ook beantwoord worden met $Theo(\mathfrak{A})$ in plaats van met $\mathfrak{A}_{DB}$.

Stelling 4.3.1. *Voor elke logische zin A over Σ_{DB} geldt dat $Theo(DB) \models A$ asa*[2] $\mathfrak{A}_{DB} \models A$.

In woorden: A is een logisch gevolg van de theorie $Theo(DB)$ asa A waar is in de structuur $\mathfrak{A}_{DB}$. Dat betekent dat -in principe- een theoremprover uit $Theo(DB)$ dezelfde queries kan beantwoorden als een databanksysteem. In zekere zin betekent het dat een databanksysteem beschouwd kan worden als een superefficiënte theoremprover voor de theorie $Theo(DB)$.

Bew. Een formule A is waar in $\mathfrak{A}_{DB}$ asa A waar is in elke structuur isomorf met $\mathfrak{A}_{DB}$ asa A waar is in elk model van $Theo(DB)$ asa $Theo(DB) \models A$. Q.E.D.

Zodoende beschrijft een theorie T die een correcte vertaling van DB is, de unieke stand van zaken van het applicatiedomein; T impliceert het correcte antwoord op alle queries, net als $\mathfrak{A}_{DB}$ zelf. Dit garandeert dat T alle informatie bevat die in de databank DB vervat zit.

Poging 1 Nemen we als voorbeeld de studentendatabank Figuur 2.1 met zijn vocabularium Σ. Een natuurlijk eerste idee om alle information uit DB voor te stellen is als de theorie T_{DB} bestaande uit alle atomische formules overeenkomend met de rijen van de tabellen:

$$\left\{ \begin{array}{c} Instructor(Ray, CS230), \ldots, Instructor(Pat, CS238), \\ Prerequ(CS230, CS238), \ldots, Prerequ(M100, M200), \\ Enrolled(Jill, CS230), \ldots, Enrolled(Flo, M200), \\ Grade(Sam, CS148, AAA), \ldots, Grade(Flo, M100, B), \\ PassingGrade(AAA), \ldots, PassingGrade(F) \end{array} \right\}$$

Voldoet deze theorie aan ons criterium? Aboluut niet. Deze theorie heeft een grote klasse van modellen waarvan vele niet in het minst lijken op de bedoelde stand van zaken. Als voorbeeld van zo'n model hernemen we de structuur $\mathfrak{B}$ uit Voorbeeld 2.3.5:

[2] Let op: het symbool $\models$ links staat voor logisch gevolg, rechts voor waarheid in structuur.

– $D_{\mathfrak{B}} = \{1\}$;
– $Ray^{\mathfrak{B}} = \cdots = CS230^{\mathfrak{B}} = \cdots = Jill^{\mathfrak{B}} = \cdots = AAA^{\mathfrak{B}} = \cdots = F^{\mathfrak{B}} = 1$.
– $Lecturer^{\mathfrak{A}} = Course^{\mathfrak{A}} = Student^{\mathfrak{A}} = Score^{\mathfrak{A}} = \{1\}$.
– $Instructor^{\mathfrak{A}} = Enrolled^{\mathfrak{A}} = Prerequ^{\mathfrak{A}} = \{(1,1)\}$;
– $Grade^{\mathfrak{A}} = \{(1,1,1)\}$.'

Het is gemakkelijk in te zien dat alle atomen van T_{DB} waar zijn in $\mathfrak{B}$. Dus $\mathfrak{B}$ is een model.

Niet alleen komt $\mathfrak{B}$ niet overeen met de stand van zaken, bovendien laat $\mathfrak{B}$ zien dat veel queries die waar zijn volgens DB niet geïmpliceerd worden door T_{DB}. Bv.:

- $\neg Ray = Hec$
- $\neg Instructor(Sue, CS230)$
- $\neg\exists x\colon Prerequ(x,x)$
- $\forall x : Prerequ(x, CS238) \Rightarrow Instructor(Ray, x)$

Elk van deze zinnen is waar in $\mathfrak{A}_{DB}$ en zou door een databanksysteem met **t** beantwoord worden.

Propositie 4.3.2. *Geen enkele van deze 4 zinnen volgt uit T_{DB}.*

Bew. In $\mathfrak{B}$ zijn alle atomen waar, inclusief alle gelijkheidsatomen. Hieruit volgt dat de eerste drie formules onwaar zijn in $\mathfrak{B}$. Bv., $\mathfrak{B} \not\models \neg(Ray = Hec)$ want $Ray^{\mathfrak{B}} = Hec^{\mathfrak{B}} = 1$.

De vierde formule $\forall x : Prerequ(x, CS238) \Rightarrow Instructor(Ray, x)$ is waar in $\mathfrak{B}$ want er is maar één element in $D_{\mathfrak{B}}$, namelijk 1, en voor x gelijk aan 1 geldt de conclusie van deze zin: $\mathfrak{B}[x:1] \models Instructor(Ray, x)$. Om te laten zien dat deze zin niet geïmpliceerd wordt, moeten we dus een andere structuur vinden.

Men bekomt zo'n structuur door $\mathfrak{B}$ aan te passen tot $\mathfrak{B}'$ door een nieuw element 2 toe te voegen aan het domein en één koppel toe te voegen aan $Prerequ^{\mathfrak{B}}$:

$$Prerequ^{\mathfrak{B}'} = \{(1,1),(2,1)\}$$

$\mathfrak{B}'$ voldoet nog steeds aan elk atoom uit T_{DB} en er geldt $\mathfrak{B}'[x:2] \not\models Prerequ(x, CS238) \Rightarrow Instructor(Ray, x)$ (ga dit na), dus is de vierde formule niet voldaan. Q.E.D.

Merk op dat het element 2 van $\mathfrak{B}'$ door geen enkel objectsymbool wordt gedenoteerd aangezien de waarde van elk objectsymbool in $\mathfrak{B}'$ gelijk is aan 1.

We hebben nu aangetoond dat elke theoremprover voor predikatenlogica die erin slaagt één van deze 4 zinnen te bewijzen uit T_{DB} een bug bevat! Het probleem ligt in de theorie T_{DB} zelf die niet alle informatie in de databank uitdrukt. Dat is wat de bizarre en ongewenste modellen uit bovenstaand bewijs aantonen.

Het laat zien dat een databank veel meer informatie bevat dan die atomische formules. We botsen hier tegen een **uiterst belangrijk aspect** van het modelleren van informatie in logica. Dat is dat mensen beschikken over allerlei impliciete kennis die zo evident is voor hen dat ze er niet eens aan denken om deze kennis te expliciteren. Elders hebben we dit soort kennis een **implicatuur** genoemd.

Wat zijn dus de *implicaturen* van een databank? Er zijn drie soorten kennis die impliciet aanwezig is in een databank en die expliciet gemaakt moeten worden in $Theo(DB)$:

- Elk paar van verschillende object symbolen C, D stellen verschillende objecten voor.
- Er zijn geen elementen in het universum die niet voorgesteld worden door een objectsymbool.
- Een atomische formule $P(D_1, \ldots, D_k)$ is waar asa $(D_1, \ldots, D_n)$ voorkomt in de tabel van P. Dus, als $(D_1, \ldots, D_k)$ niet voorkomt in de tabel van P, dan is $P(D_1, \ldots, D_k)$ onwaar.

Elk van deze drie implicaturen kwam eerder voor in de cursus, namelijk in deelparagraaf 2.9.5, respectievelijk deelparagraaf 2.9.6 en deelparagraaf 2.9.4.

Zij S de verzameling van objectsymbolen uit DB. Bv. in het geval van de studentendatabank uit Figuur 2.1 bestaat S uit de symbolen $Ray, Hec, \ldots, Jill, \ldots, CS230, \ldots, AAA, \ldots$.

Definitie 4.3.7. De **Unique Name Axioma's** voor S, notatie $UNA(S)$, is de verzameling van alle formules $\neg(C = D)$ voor elk paar van verschillende objectsymbolen $C, D \in S$.

Voor de studentendatabank van Figuur 2.1 bevat deze theorie axiomas zoals

$$\neg(Ray = Hec), \neg(Ray = CS230), \neg(Ann = Jill), \neg(AAA = B), \ldots$$

Elk model van $UNA(S)$ voldoet aan $\neg(Ray = Hec)$, in tegenstelling tot het model $\mathfrak{B}$ van de theorie T_{DB} uit poging 1.

Definitie 4.3.8. Zij $S = \{C_1, \ldots, C_n\}$. Het Domain Closure Axioma voor S, notatie $DCA(S)$, is de zin:

$$\forall x\colon (x = C_1 \vee \cdots \vee x = C_n)$$

In elk model $\mathfrak{A}$ van $DCA(S)$ bestaat voor elk object $d \in D_{\mathfrak{A}}$ een objectsymbool C_i zodat $d = {C_i}^{\mathfrak{A}}$ (ga dit na door middel van Definitie 2.3.6). Het ongewenste model $\mathfrak{B}'$ van T_{DB} in het bewijs van Propositie 4.3.2 voldoet niet aan $DCA(S)$. Immers, $\mathfrak{B}'[x : 2] \not\models x = Ray \vee \cdots \vee x = F$ want de waarde van elk objectsymbool is 1.

De databank DB bevat voor elk databankpredicaat P/k een tabel (of equivalent, een relatie). We zullen deze tabel als volgt noteren ($C_{i,j}$ is het symbool op de i-de rij en de j-de kolom):

$C_{1,1}$	$\ldots$	$C_{1,k}$
$\ldots$	$\ldots$	$\ldots$
$C_{m,1}$	$\ldots$	$C_{m,k}$

Definitie 4.3.9. De *definitie* van het predicaat P, notatie $Def(P)$, is de logische zin:

$$\begin{aligned}\forall x_1\colon \ldots \forall x_k\colon [P(x_1, \ldots, x_k) \Leftrightarrow\ & (x_1 = C_{1,1} \wedge \cdots \wedge x_k = C_{1,k}) \\ & \vee \cdots \vee \\ & (x_1 = C_{m,1} \wedge \cdots \wedge x_k = C_{m,k})]\end{aligned}$$

Voorbeeld 4.3.8. $Def(Prerequ)$ voor de tabel van $Prerequ$ in Figuur 2.1 is:

$$\begin{aligned}\forall x\colon \forall y\colon [Prerequ(x,y) \Leftrightarrow\ & (x = CS148 \wedge y = CS230)\vee\\ & (x = CS230 \wedge y = CS238)\vee\\ & (x = M100 \wedge y = M200)]\end{aligned}$$

Oefening 4.3.10. *Ga na dat $Def(P)$ een expliciete definitie is volgens Definitie 2.8.1.*

Als een structuur $\mathfrak{A}$ voldoet aan $Def(P)$ dan behoort een tupel $(d_1, \ldots, d_k)$ tot $P^{\mathfrak{A}}$ asa er bestaat een rij $i \in [1, m]$ zodat $d_1 = {C_{i,1}}^{\mathfrak{A}}, \ldots, d_k = {C_{i,k}}^{\mathfrak{A}}$. Ga dit na met de Definitie 2.3.6 van waarheid.

Zij DB een databank over Σ met verzameling S van objectsymbolen.

Definitie 4.3.10. We definiëren $Theo(DB)$ als de theorie bestaande uit:

- $UNA(S)$;
- $DCA(S)$;
- $Def(P)$, voor elk predikaatsymbool $P \in \Sigma$.

Nu bewijzen we dat $Theo(DB)$ een correcte vertaling is van DB. M.a.w. een Σ-structuur $\mathfrak{B}$ is een model van deze theorie asa er een isomorfisme b is tussen $\mathfrak{A}_{DB}$ en $\mathfrak{B}$. Ter herinnering, de databank DB bepaalt een unieke Herbrand-structuur $\mathfrak{A}_{DB}$:

- $D_{\mathfrak{A}_{DB}} = S$;
- voor elk objectsymbool $C \in S$ geldt $C^{\mathfrak{A}_{DB}} = C$;
- voor elk predicaatsymbool P/k geldt dat $P^{\mathfrak{A}}$ de relatie is bestaande uit de rijen van de tabel van P.

Voor de definitie van isomorfisme, zie Definitie 2.10.1.

Stelling 4.3.2. *$\mathfrak{B}$ is een model van $Theo(DB)$ asa er bestaat een isomorfisme b tussen $\mathfrak{A}_{DB}$ en $\mathfrak{B}$.*

Neem voor de structuur $\mathfrak{B}$ de afbeelding b die een objectsymbool C afbeeldt op $C^{\mathfrak{B}}$; in wiskundige notatie $b : S \to D_{\mathfrak{B}} : C \mapsto C^{\mathfrak{B}}$.

Lemma 2. *Als er een isomorfisme van $\mathfrak{A}_{DB}$ naar $\mathfrak{B}$ bestaat dan is het b. Voor elke $\mathfrak{B}$ voldoet b aan de eerste voorwaarde om een isomorfisme te zijn, Definitie 2.10.1.*

Bew. Stel dat h een isormorfisme is van $\mathfrak{A}_{DB}$ naar $\mathfrak{B}$. Dan is h een bijectie van S naar $D_{\mathfrak{B}}$ die voldoet aan de eerste voorwaarde van isomorfisme: als $C^{\mathfrak{A}_{DB}} = d$ dan $C^{\mathfrak{B}} = b(d)$. Maar als $C^{\mathfrak{A}_{DB}} = d$ dan is $d = C$ en dus geldt $C^{\mathfrak{B}} = h(C)$, voor alle $C \in S$. Enkel b voldoet hieraan.

Voor elke $\mathfrak{B}$ geldt dat als $C^{\mathfrak{A}_{DB}} = d$ (t.t.z., $d = C$) dan $C^{\mathfrak{B}} = b(d) = b(C)$, zodat b voldoet aan de eerste voorwaarde van Definitie 2.10.1. Q.E.D.

Lemma 3. *$\mathfrak{B} \models UNA(S) \wedge DCA(S)$ asa b is een bijectie.*

Bew. Veronderstel eerst dat $\mathfrak{B} \models UNA(S) \wedge DCA(S)$. Uit $\mathfrak{B} \models UNA(S)$ volgt voor elk paar van verschillende objectsymbolen $C, D \in S$ dat $\mathfrak{B} \models \neg(C = D)$, dus $b(C) = C^{\mathfrak{A}} \neq D^{\mathfrak{B}} = b(D)$. (Dat betekent dat b een injectie is.) Uit $\mathfrak{B} \models DCA(S)$ volgt voor elk domeinelement $d \in D_{\mathfrak{B}}$ dat $\mathfrak{A}[x : d] \models (x = C_1 \vee \cdots \vee x = C_n)$. Gebruik makend van de definitie van waarheid (Definitie 2.3.6) kunnen we afleiden dat een C_i bestaat zodat $d = {C_i}^{\mathfrak{B}} = b(C_i)$. (Dat betekent dat b een surjectie is.) Dus is b een bijectie.

Omgekeerd, als b een bijectie is dan volgt voor een paar van verschillende objectsymbolen $C, D \in S$ dat $C^{\mathfrak{B}} = b(C) \neq b(D) = D^{\mathfrak{B}}$, waaruit volgt dat $\mathfrak{B} \models \neg(C = D)$. Dit geldt voor elk zo'n paar; bijgevolg $\mathfrak{B} \models UNA(S)$. Verder geldt dat voor elke $d \in D_{\mathfrak{B}}$ dat een C_i bestaat zodat ${C_i}^{\mathfrak{B}} = b(C_i) = d$. Opnieuw met de definitie van waarheid kunnen we gemakkelijk afleiden dat $\mathfrak{A} \models DCA(S)$. Q.E.D.

Lemma 4. *Een structuur $\mathfrak{B}$ is een model van $Def(P)$ asa: een koppel $(d_1, \ldots, d_k)$ behoort tot $P^{\mathfrak{B}}$ asa er bestaat een rij $i \in [1, m]$ zodat ${C_{i,1}}^{\mathfrak{B}} = d_1, \ldots, {C_{i,n}}^{\mathfrak{B}} = d_n$.*

Bew. Dit is een gemakkelijke oefening. Probeer zelf. Q.E.D.

Bew. van Stelling 4.3.2.

Als $\mathfrak{B} \models Theo(DB)$ dan is b een bijectie (Lemma 3) die aan de eerste voorwaarde van isomorfisme voldoet (Lemma 2) en ook aan de derde voorwaarde (Lemma 4). En ook aan de tweede voorwaarde aangezien er geen functiesymbolen zijn in Σ. Dus is b een isomorfisme.

Omgekeerd, als er een isomorfisme bestaat dan is het b (Lemma 2); aangezien b een bijectie is voldoet $\mathfrak{B}$ aan $UNA(S) \wedge DCA(S)$ (Lemma 3); aangezien b toegepast op de tabel van P gelijk is aan $P^{\mathfrak{B}}$ volgt dat $\mathfrak{B}$ voldoet aan $Def(P)$ (Lemma 4). Q.E.D.

De stelling toont aan dat $Theo(DB)$ alle informatie van DB uitdrukt.

4.3.4 SQL

Niet voor het examen.

Bijna alle commerciële databanksystemen gebruiken de SQL-taal voor het formuleren van queries. SQL is gebaseerd op de principes van de predikatenlogica maar heeft een totaal andere syntax. Toch is het niet moeilijk het verband met logica te zien. In 2Ba is er een vak over databanken, met meer details over SQL. We beperken ons hier tot enkele voorbeelden en een bespreking van het verband met logica.

In een SQL database wordt het vocabularium het *database-schema* genoemd. Het bestaat uit tabel-declaraties met kolom-namen:

$$Tab(TabVeld_1, TabVeld_2, \ldots, TabVeld_n)$$

waarbij Tab de naam van de tabel is, en $TabVeld_i$ de naam van de i-de kolom van de tabel.

Voorbeeld 4.3.9. In een SQL-databank:
- $Grade/3 : Grade(student, vak, score)$
- $PassingGrade/1$: $PassingGrade(score)$
- $Prerequ/2 : Prerequ(vakpre, vakpost)$ en
- $Enrolled/2 : Enrolled(student, score)$.

In SQL-queries wordt een query gezien als een instructie om een tabel te construeren. Bv. de tabel bestaande uit all studenten die een examen hebben gedaan vooor een vak:

Voorbeeld 4.3.10. Alle koppels (x,y) zodat student x geslaagd is voor vak y.

$$\{(x, y) : \exists z \colon (Grade(x, y, z))\}$$

De SQL-query:

$$SELECT\ student, vak$$
$$FROM\ Grade$$

Deze query selecteert de waardes uit de kolommen *student* en *vak* in rijen van *Grade*. Merk op hoe kolomnamen gebruikt worden als variabelen.

Het volgend voorbeeld is een verfijning waarin enkel geslaagde examens in rekening worden gebracht.

Voorbeeld 4.3.11. Alle koppels (x,y) zodat student x geslaagd is voor vak y.

$$\{(x, y) : \exists z \colon (Grade(x, y, z) \wedge PassingGrade(z))\}$$

De SQL-query:

$$SELECT\ student, vak$$
$$FROM\ Grade$$
$$WHERE\ EXISTS\ (SELECT\ score$$
$$FROM\ PassingGrade$$
$$WHERE\ Grade.score = score)$$

Deze query selecteert de waardes uit de kolommen *student* en *vak* in rijen van *Grade* die aan een voorwaarde voldoen, namelijk dat de volgende tabel niet leeg is:

$$SELECT\ score$$
$$FROM\ PassingGrade$$
$$WHERE\ Grade.score = score$$

Deze tabel selecteert de kolom *score* uit de *PassingGrade* tabel (deze tabel heeft slechts 1 kolom), maar enkel indien de score gelijk is aan *Grade.score*, de waarde van het score veld uit de *Grade*-tabel. Deze tabel is ofwel leeg ($\emptyset$) indien *Grade.score* geen slaagscore is; ofwel is het de verzameling $\{Grade.score\}$ met 1 element.

In wiskundige notatie kunnen we dit herschrijven als:

$$\{(x, y) | \exists z : Grade(x, y, z) \in Grade \wedge \\ \emptyset \neq \{z' | z' \in PassingGrade \wedge z = z'\}\}$$

De geneste verzameling is ofwel het singleton $\{z\}$, indien z een slaagscore is, ofwel is het de lege verzameling, genoteerd als $\emptyset$, indien z geen slaagscore is. Maw, de tabel die met deze verzameling overeenkomt is leeg asa *sc* een slaagscore is. Enkel als deze tabel niet leeg is, behoort het koppel (x, y) uit *Grade* tot de verzameling. Dit is natuurlijk het geval als en slechts als $PassingGrade(z)$ waar is.

In SQL wordt de existentiële kwantor aangeduid met EXISTS. Op het eerste gezicht een belangrijk verschil met logica is dat deze operator loopt over de rijen van een tabel, niet over de elementen van het databankdomein. Er bestaat geen universele kwantor. Deze moet je uitdrukken met NOT EXISTS NOT.

Voorbeeld 4.3.12. Gevraagd: Alle excellente studenten. Daarmee bedoelen we studenten die minstens één examen afgelegd hebben en op alle afgelegde examens een AA of AAA behaalden.

$$\{x : \exists y\colon \exists z\colon Grade(x, y, z) \wedge \forall y\colon \forall z\colon [Grade(x, y, z) \Rightarrow z = AA \vee z = AAA]\}$$

Voor deze query worden in SQL twee rij-variabelen gebruikt, $Gr1$ en $Gr2$:

SELECT student
*FROM Grade AS Gr*1
WHERE NOT EXISTS (SELECT score
*FROM Grade AS Gr*2
*WHERE Gr*1.*student* = *Gr*2.*student AND NOT* (*score* = 'AA' *OR score* = 'AAA'))

Deze rij-variabelen staan voor rijen in een tabel. De query selecteert de waarde in het veld *student* in rijen $Gr1$ van de tabel *Grade* waarvoor de volgende tabel leeg is: de tabel bestaande uit de scores in rijen $Gr2$ in de tabel *Grade* waarvoor geldt dat $Gr1$ en $Gr2$ dezelfde student bevatten, en de score (in $Gr2$) niet minstens AA is.

SQL in het echt. Historisch gezien is SQL gegroeid uit logica en deze taal omvat een deeltaal die beschouwd kan worden als een syntactische variant van logica. Wellicht behoort de grote meerderheid van SQL queries die in de praktijk gesteld worden tot die deeltaal. Hierdoor is SQL een declaratieve programmeertaal: een programmeertaal gebaseerd op logica. Het is waarschijnlijk de meest gebruikte declaratieve programmeertaal ter wereld.

Typisch voor zo'n soort programmeertaal is dat ze elke instructie aan een soort operatie koppelen. Hier bv. zijn queries gezien als instructies om een tabel te construeren. De echt declaratieve interpretatie is dat het symbolische notaties zijn voor verzamelingen, geen opdracht om tabellen te construeren.

Oefening 4.3.11. *De OnderdelenVerdelersDatabank bestaat uit volgende tabellen:*

- $Ver(id, naam, stad)$ *met respectievelijk de identifier (een primary key), de naam en de stad van een verdeler.*
- $Ond(id, naam, kleur, stad)$*, met kolommen die de identifier (een primary key), naam, kleur en stad (waar het onderdeel opgestapeld is) van het onderdeel.*
- $Pro(id, naam, stad)$ *met id (primary key), naam en stad van een project dat onderdelen vergt.*
- $VOP(verdeler\_id, onderdeel\_id, project\_id)$*: wie levert wat aan welk project.*

Kies een geschikt logisch vocabularium, en formuleer volgende queries in predikatenlogica:

1. *Gevraagd: alle id's van onderdelen geleverd door een verdeler in London aan een project in London.*

2. *Gevraagd: alle id's van projecten waaraan minstens één verdeler uit een andere stad levert.*

3. *Gevraagd: alle id's van projecten waaraan geen enkele verdeler in London een rood onderdeel levert.*

4. *Gevraagd: alle id's van onderdelen die geleverd worden aan alle projecten in London.*

5. *Gevraagd: alle id's van verdelers die eenzelfde onderdeel leveren aan alle projecten.*

4.4 De oudste logische theorie: de Peano axioma's

In het grondslagenonderzoek dat begon in het 2de deel van de 19de eeuw was de studie van Peano over de natuurlijke getallen een van de eerste bijdrages.

Peano was een 19de eeuwse logicus en wiskundige die in 1898 een logische studie publiceerde over de natuurlijke getallen. De bedoeling van zijn theorie was om de structuur van de natuurlijke getallen te beschrijven in logica. Hierdoor kon deze structuur grondig wiskundig geanalyseerd worden.

De structuur van de natuurlijke getallen bestaat uit de verzameling $\mathbb{N}$ van natuurlijke getallen, het getal 0, de successor functie S die een getal op het volgend getal afbeeldt, en verder de functies $+$ en $\times$. Om een structuur uit de logica te bekomen dienen we niet-logische symbolen te kiezen, en hun waardes vast te leggen. Men kiest gewoon de wiskundig gangbare symbolen: $\Sigma_P = \{0/0 :, S/1 :, +/2 :, \times/2 :\}$.

De notaties van termen en formules over dit vocabularium wijken enigzins af van de standaard notatie in predicatenlogica. De symbolen $+$ en $\times$ worden op infix manier gebruikt, dus men schrijft de term $S(0) + 0$ in plaats van $+(S(0), 0)$. In dat opzicht lijkt de logisch notatie meer op de notatie in wiskundige teksten.

De waardes in de structuur $\mathfrak{A}_\mathbb{N}$ zijn:
- $D_{\mathfrak{A}_\mathbb{N}} = \mathbb{N}$;
- $0^{\mathfrak{A}_\mathbb{N}} = 0$;
- $S^{\mathfrak{A}_\mathbb{N}} = S = \{(n, n+1) | n \in \mathbb{N}\}$;
- $+^{\mathfrak{A}_\mathbb{N}} = + = \{((n, m), n + m) | n, m \in \mathbb{N}\}$;
- $\times^{\mathfrak{A}_\mathbb{N}} = \times = \{((n, m), n \times m) | n, m \in \mathbb{N}\}$.[3]

Aangezien Σ_P uit enkel functie-symbolen bestaat, zijn de enige atomaire formules over Σ_P gelijkheidsatomen. Dus alle formules zijn opgebouwd uit gelijkheidsatomen.

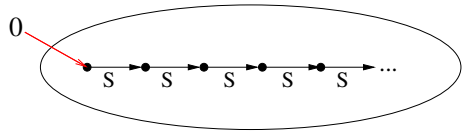
Een getal $n \in \mathbb{N}$ wordt voorgesteld door de term $\underbrace{S(S(\ldots S}_{n \text{ keer } S}(0)\ldots))$. We verkorten dit soms tot de wiskundige notatie $S^n(0)$. Dus $S^n(0)$ is geen term, het is een afgekorte notatie voor de term $S(\ldots S(0)\ldots)\ldots)$ met n voorkomens van S. Merk op dat $(S^n(0))^{\mathfrak{A}_\mathbb{N}} = n$, voor elke $n \in \mathbb{N}$.

Overlading van symbolen en disambiguering door context In de symbolische beschrijving van de structuur $\mathfrak{A}_\mathbb{N}$ hierboven staan opvallende uitdrukkingen zoals $0^{\mathfrak{A}_\mathbb{N}} = 0$. Let op: het linkse voorkomen van "0" verwijst naar het symbool $0 \in \Sigma_p$, het rechtse refereert naar het getal 0. Symbolen en getallen: het zijn verschillende dingen. Verderop gebeurt hetzelfde met "+" en "$\times$". [4]

Het gebeurt wel vaker in de wiskunde en wetenschappen dat hetzelfde concept in verschillende teksten door verschillende wiskundige symbolen wordt aangeduid. En omgekeerd, dat hetzelfde symbool in verschillende teksten gebruikt wordt om verschillende concepten aan te duiden.

[3] Let op de manier waarop de functies $S, +$ en $\times$ hier symbolisch voorgesteld worden. Volgens de grondslagen van de wiskunde is elke functie f de verzameling van koppels $\{(d, f(d)) | d \in dom(f)\}$. Bv. de productfunctie $\times$ is de verzameling $\{((n, m), n \times m) | n, m \in \mathbb{N}\}$. Een element $((n, m), n \times m)$ van deze verzameling is een paar bestaande uit een paar (n, m) en het getal dat hun product is, namelijk $n \times m$. Bv., een element is $((2, 3), 6)$.

[4] Het onderscheid tussen symbool en waarde dat we hier zien komt ook voor in de volgende twee zinnen: waarde: "Dokter, de voorgeschreven medicijnen werkten niet", en symbool: "Dokter, de voorgeschreven medicijnen waren onleesbaar".



Figuur 4.1: De structuur van de natuurlijke getallen

Binnen in dezelfde tekst is het altijd te vermijden om hetzelfde symbool in verschillende betekenissen te gebruiken (met uitzondering van gekwantificeerde symbolen met een beperkte scoop die wel vaak hergebruikt worden). Want dit leidt potentieel tot ambiguïteit. Maar, in "$0^{\mathfrak{A}_{\mathbb{N}}} = 0$" wordt hetzelfde symbool zelfs binnen dezelfde uitdrukking in verschillende betekenissen gebruikt: de linker "0" verwijst naar een symbool, de rechter "0" naar de waarde 0. Excuseer daarvoor, maar ... er zijn goede redenen voor.

Ten eerste, deze tekst volgt de conventie: het is standaard om in logische teksten en boeken en systemen de wiskundige symbolen $0, +, \times$ zowel als wiskundige waarde en als niet-logisch symbool te gebruiken. [5]

Ten tweede is het een kans om te wijzen op hoe mensen omgaan met overladen symbolen of woorden. Een overladen symbool of woord heeft meerdere betekenissen. Dit leidt niet noodzakelijk tot ambiguïteit. In zo'n geval gebruikt men, bewust of onbewust, context om de bedoelde betekenis van het symbool te bepalen.

Neem de uitdrukking "$+^{\mathfrak{A}_{\mathbb{N}}} = +$". In het linkerlid wordt de interpretatie van "+" in de structuur $\mathfrak{A}_{\mathbb{N}}$ genomen, dus moet "+" bedoeld zijn als symbool. In het rechterlid wordt "+" gelijk gesteld aan een waarde, dus moet de bedoelde betekenis de som-functie zijn.

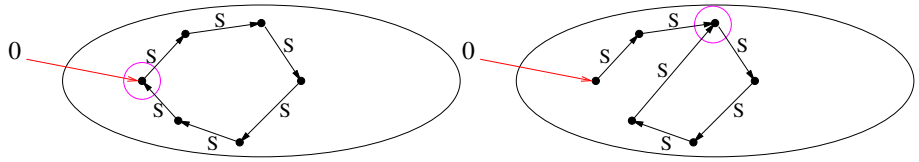
Peano's theorie Peano's doel was om de structuur $\mathfrak{A}_{\mathbb{N}}$ van de natuurlijke getallen te axiomatiseren door middel van een theorie. We weten al dat geen enkele theorie enkel $\mathfrak{A}_{\mathbb{N}}$ als model zal hebben, immers, als een theorie voldaan is in $\mathfrak{A}_{\mathbb{N}}$, dan is ze voldaan in elke structuur isomorf aan $\mathfrak{A}_{\mathbb{N}}$. Peano's doel was dus een theorie $Theo(\mathfrak{A}_{\mathbb{N}})$ op te stellen wiens modellen isomorf zijn aan $\mathfrak{A}_{\mathbb{N}}$. Dus Peano's doel was gelijkaardig aan ons doel bij de opstelling van $Theo(DB)$ voor een databank DB.

Hoe zien structuren $\mathfrak{A}$ eruit die isomorf zijn aan $\mathfrak{A}_{\mathbb{N}}$? Het zijn oneindige lineaire sequenties als in Figuur 4.1 die beginnen in $0^{\mathfrak{A}}$ en zodat alle elementen van het universum bereikbaar zijn vanuit $0^{\mathfrak{A}}$ door herhaald toepassen van de functie $S^{\mathfrak{A}}$. Dan volgt hieruit dat $S^n(0)^{\mathfrak{A}}$ het n-de element in de rij is. We zijn dus op zoek naar axiomas die dit soort structuur characteriseren. Dit is het moeilijkste deel van Peano's project.

De theorie $Theo(DB)$ bestaat uit 3 delen: UNA, DCA, en expliciete definitie-formules voor de tabellen. Peano's theorie heeft dezelfde structuur, al zien de axiomas er wel volledig verschillend uit.

UNA: **Unique Names Axiomas:** dit deel van de theorie dient om uit te drukken dat verschillende termen $S^n(0), S^m(0)$ $(n \neq m)$ een verschillende waarde hebben. Dit kan door de oneindige theorie $\{\neg(S^n(0) = S^m(0)) | n, m \in \mathbb{N}, n \neq m\}$. Peano vond een compacte eindige

[5] Google bv. "Peano rekenkunde" en let erop welke symbolen in de logische axiomas gebruikt worden.



Figuur 4.2: Structuren waarin Peano_1 en Peano_2 onwaar zijn

voorstelling:

$$\forall x\colon \neg(0 = S(x)) \qquad (\text{Peano}_1)$$
$$\forall x\colon \forall y\colon (S(x) = S(y) \Rightarrow x = y) \qquad (\text{Peano}_2)$$

Het eerste axioma drukt uit dat 0 niet in het beeld van S zit; het tweede dat S een injectieve functie is: het beeldt verschillende objecten af op verschillende objecten.

Figuur 4.4 illustreert 2 soorten structuren die geëlimineerd worden door deze axiomas.

Oefening 4.4.1. *Toon aan dat als $n \neq m$, deze theorie de formule $\neg(S^n(0) = S^m(0))$ impliceert. In symbolen, argumenteer dat $Peano_{1-2} \models \neg(S^n(0) = S^m(0))$. Hint: stel dat k het minimum is van n en m. Pas k keer $Peano_2$ toe en vervolgens $Peano_1$.*

Oefening 4.4.2. *Gebruik de KE-bewijsmethode om aan te tonen dat $Peano_{1-2} \models \neg(S(S(0)) = S(0))$.*

***DCA*: Domain Closure Axioma:** dit deel van Peano's theorie drukt uit dat elk object van het universum de waarde is van minstens één van de termen $0, S(0), S^2(0), \ldots$. Het idee wordt gevat door de volgende oneindige"zin" analoog aan de DCA in $Theo(DB)$:

$$\forall x\colon (x = 0 \vee x = S(0) \vee x = S(S(0)) \vee \ldots)$$

Maar helaas, predikatenlogica bevat geen oneindige zinnen. Elke formule in predikatenlogica is eindig.

Oefening 4.4.3. *Waarom is dit geen logische zin? Toon aan dat dit geen formule is.*

Later, in de cursus Modelleren van Complexe Systemen in de 1ste Master, zal bewezen worden dat DCA niet in Predicatenlogica kan uitgedrukt worden.

Peano vond een manier om DCA uit te drukken in *tweede-orde-logica*, een uitbreiding van predikatenlogica die ook door Frege werd geïntroduceerd. Het enige verschil is dat men ook over verzamelingen kan kwantificeren. Peano's axioma is het volgende:

$$\forall P\colon P(0) \wedge (\forall x\colon P(x) \Rightarrow P(S(x)) \Rightarrow \forall x\colon P(x) \qquad (\text{Peano}_3)$$

In woorden: elke verzameling P die 0 bevat en als het n bevat dan ook $n+1$ bevat alle elementen van het domein. Dit noemt men het *inductie-axioma*.

We zeggen dat een verzameling X van natuurlijke getallen *gesloten is* onder 0 en S indien $0 \in X$ en indien $n \in X$ impliceert dat $S(n) \in X$.

Dus, het inductie axioma drukt uit dat de enige deelverzameling van het domein die gesloten is onder 0 en S gelijk is aan het volledige domein.

Of nog: er zijn geen strikte deelverzamelingen van het domein die gesloten zijn onder 0 en S.

De "definities" van + en ×. Een correcte theorie impliceert alle ware gelijkheidsatomen $S^n(0) + S^m(0) = S^{n+m}(0)$ en $S^n(0) \times S^m(0) = S^{n\times m}(0)$. Peano vond een eindige manier om deze uit te drukken:

$$\begin{array}{ll} \forall x\colon (0 + x = x) & (\text{Peano}_4) \\ \forall x\colon \forall y\colon (S(x) + y = S(x + y)) & (\text{Peano}_5) \\ \forall x\colon (0 \times x = 0) & (\text{Peano}_6) \\ \forall x\colon \forall y\colon (S(x) \times y = y + (x \times y)) & (\text{Peano}_7) \end{array}$$

Merk echter op dat het niet volstaat dat een correcte definiërende theorie van + en × de ware gelijkheidsatomen impliceert. Ze moet ook de negatie van de onware atomen impliceren. Dus, elke negatieve literal $\neg(S^n(0) + S^m(0) = S^k(0))$ met $k \neq n + m$, moet ook logisch volgen uit de theorie.

Gelukkig is dit het geval. Immers, voor elke $k \neq n + m$ impliceert de UNA theorie dat $\neg(S^{n+m}(0) = S^k(0))$. Dus als de theorie impliceert dat $S^n(0) + S^m(0) = S^{n+m}(0)$ zal het samen met de UNA axioma's ook impliceren dat $\neg(S^{n+m}(0) = S^k(0))$, voor $k \neq n + m$.

Oefening 4.4.4. *Bewijs dat Peano_{1-7} elk atoom $S^n(0) + S^m(0) = S^{n+m}(0)$ logisch impliceert. Hint: gebruik Peano_5 n keer waarna Peano_6.*

Oefening 4.4.5. *Bewijs dat Peano_{4-7} elk atoom $S^n(0) \times S^m(0) = S^{n\times m}(0)$ logisch impliceert.*

Merk de gelijkenis op tussen de axiomas Peano_{4-7} en de inductieve definities van + en × uit de inleidende Sectie 1.7. Bovenstaande oefeningen zijn gelijkaardig aan oefeningen in die bundel.

Stelling 4.4.1 (Peano)**.** *Een structuur $\mathfrak{A}$ over Σ_P is een model van Peano_{1-7} asa $\mathfrak{A}$ is isomorf met $\mathfrak{A}_{\mathbb{N}}$.*

Dat betekent dat Peano's theorie inderdaad correct en volledig is. Dit is een gelijkaardig resultaat als dat we bewezen voor $Theo(DB)$.

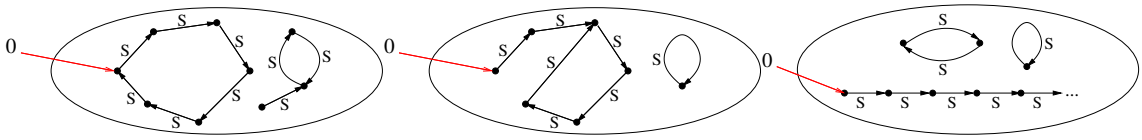
Peano's theorie bevat 6 zinnen in predikatenlogica en 1 zin in tweede-orde-logica. Dit is dus geen theorie van predikatenlogica, omwille van het inductie-axioma. Er bestaat echter een "benadering" van dit axioma in predikatenlogica waarmee men bijna alle belangrijke eigenschappen van de natuurlijke getallen kan bewijzen. Deze "benadering" noemt men het *inductie-schema*. Het is een oneindige verzameling van logische zinnen.

Definitie 4.4.1. Het **inductieschema** voor een vocabularium Σ dat minstens $0, S/1$ bevat, bestaat uit alle zinnen van de predikatenlogicataal van Σ van de volgende vorm:

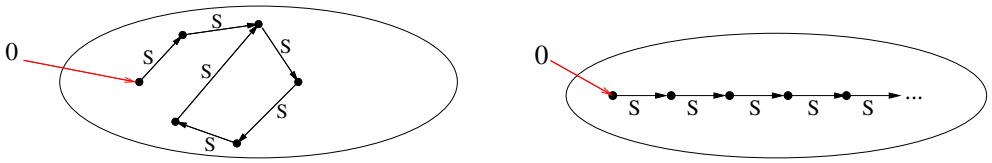
$$\forall y_1\colon \ldots \forall y_n\colon (\varphi[0] \wedge \forall x\colon (\varphi[x] \Rightarrow \varphi[S(x)])) \Rightarrow \forall x\colon \varphi[x])$$

waarbij $\varphi[x]$ een willekeurige formule is van predikatenlogica met vrije constante-symbolen x en $y_1, \ldots, y_n$ die niet voorkomen in Σ.

Men merkt een sterk verband tussen een zin uit het inductieschema en het bewijsprincipe door inductie. Hierbij speelt $\varphi[n]$ de rol van de inductiehypothese $\mathcal{H}[n]$.



Figuur 4.3: Mogelijke en onmogelijke werelden van Peano_1 en Peano_2



Figuur 4.4: Meer mogelijke en onmogelijke werelden van UNA en DCA

De oneindige theorie, verkregen door in de Peano axioma's het inductie-axioma Peano_3 te vervangen door het inductieschema, wordt **Peano rekenkunde** genoemd.

Oefening 4.4.6. *De Figuur 4.4 bevat drie grafen die structuren van het vocabularium* $\{0/0 : , S/1 :\}$ *voorstellen (noem ze van links naar rechts* $\mathfrak{A}_1, \mathfrak{A}_2, \mathfrak{A}_3$*). Merk de 3 puntjes op in* $\mathfrak{A}_3$*: dit is een weinig wiskundige maar courante manier om een oneindig patroon aan te geven. Hier geeft het aan dat* $\mathfrak{A}_3$ *een oneindige ketting van domeinelementen bevat, met telkens een boog van een element naar het volgende element. De ketting begint in* $0^{\mathfrak{A}_3}$*. Werk een symbolische representatie uit voor* $\mathfrak{A}_1$ *en* $\mathfrak{A}_3$*.*

Oefening 4.4.7. *De Figuur 4.4 bevat drie grafen die structuren van het vocabularium* $\{0/0 : , S/1 :\}$ *voorstellen (noem ze* $\mathfrak{A}_1, \mathfrak{A}_2, \mathfrak{A}_3$*). De derde structuur bevat een oneindige ketting die begint in* $0^{\mathfrak{A}_3}$*. Ga na in elk ervan of* Peano_1 *en* Peano_2 *waar zijn. Duidt aan waar in geval van onwaarheid.*

Oefening 4.4.8. *In de 3 structuren van Figuur 4.4, welke zijn de objecten die de waarde zijn van een term* $S^n(0)$*? In welke structuren is de DCA (t.t.z.,* Peano_3*) voldaan? Voor elke structuur waar mogelijk, teken een deelverzameling* P *gesloten onder* 0 *en* S *die het inductie-axioma* Peano_3 *tegenspreekt.*

Oefening 4.4.9 (Isomorfisme)**.** *Voor de derde structuur*$\mathfrak{A}_3$ *van Figuur 4.4, neem de functie* b *die de 3 top elementen op respectievelijk* $0, 1, 2$ *afbeeldt, en vervolgens de elementen van de oneindige ketting te beginnen met* $0^{\mathfrak{A}_3}$ *op 3, 4, 5, ... Ga na dat dit een bijectie is tussen* $D_{\mathfrak{A}_3}$ *en* $\mathbb{N}$*. Ga na of* b *een isomorfisme is.*

Oefening 4.4.10. *In de Figuur 4.4, ga na over UNA voldaan is (*Peano_{1-2}*) en DCA (*Peano_3*). Zijn er structuren die isomorf zijn met de natuurlijke getallen (i.e.,* $\mathfrak{A}_{\mathbb{N}}$ *beperkt tot* 0 *en* S*)?*

Geschiedenis De structuur $\mathfrak{A}_{\mathbb{N}}$ is een model van deze theorie en voldoet aan elk axioma uit het inductieschema. Maar in tegenstelling tot Peano's theorie met het inductie-axioma, heeft deze oneindige theorie ook modellen die niet isomorf zijn met $\mathfrak{A}_{\mathbb{N}}$. M.a.w., Peano-rekenkunde heeft zogenaamde "niet-standaard modellen".

De Peano rekenkunde is dus logisch zwakker dan Peano's theorie Peano_{1-7}. Toch zijn bijna alle "klassieke" stellingen over $\mathbb{N}$ over Σ_P een logisch gevolg van Peano rekenkunde.

Daarom dacht men vóór 1931 dat elke zin van Σ_P die waar is in $\mathfrak{A}_{\mathbb{N}}$, bewijsbaar zou zijn uit deze theorie. Kurt Gödel (1931) bewees echter dat dit niet waar is (zie volgend Hoofdstuk).

[6] Sindsdien werden ook interessante eigenschappen ontdekt die wel voldaan zijn in de structuur van de natuurlijke getallen maar niet geïmpliceerd worden door Peano-rekenkunde. Deze eigenschappen zijn niet voldaan in niet-standaard modellen.

Wat Gödel bewees is sterk gerelateerd tot de *onbeslisbaarheid van natuurlijke getallen* (of de stelling van Church-Turing) die we in volgend hoofdstuk zullen zien. Doet de naam Turing een belletje rinkelen? Hij is één van de aartsvaders van de informatica. Hij was de uitvinder van het eerste theoretische model voor computers (Turing machines genoemd). Dit was rechtstreeks geïnspireerd door Gödels resultaten.

De rol van Peano's rekenkunde in de informatica is niet te onderschatten. In veel complexe informaticaproblemen moet geredeneerd worden over getallen. Bv. compileroptimalisatiesystemen gebruiken Presburger rekenkunde om de volgorde van operaties te optimaliseren, parallelisme mogelijk te maken en geheugengebruik te reduceren. Bv. bewijstechnieken om de correctheid van programma's bewijzen zijn gebaseerd op Peano's inductieschema. De (interactieve) theoremprovers Coq en Isabelle zijn gespecialiseerd voor inductiebewijzen.

4.5 Zoekproblemen en modelgeneratie

Veel informaticaproblemen bestaan uit het zoeken van "waardes" die aan bepaalde "beperkingen" voldoen.

- Een kleuring zoeken van nodes in een grafe zodat twee aanpalende nodes verschillend gekleurd zijn.
- Oplossingen voor logische puzzels, sudoku's, n koninginnen op een $n \times n$ schaakbord plaatsen zodat ze elkaar niet aanvallen.
- Uurroosterproblemen: lessenroosters, examenroosters, ziekenhuisroosters, enz.
- Planningsproblemen: een serie van acties berekenen om een gewenste doelstelling te bereiken.
- Logistieke problemen: DHL zoekt een verzameling van verzend- en verscheepacties om al hun postpakketten ter plaatse te brengen.

De natuurlijke manier om dergelijke problemen aan te pakken in logica is om een vocabularium Σ met de relevante concepten, een theorie T op te stellen met de beperkingen, en modellen te berekenen van deze theorie. Modellen komen overeen met standen van zaken waarin de "beperkingen" voldaan zijn, bijgevolg zijn de modellen oplossingen voor het probleem. Verschillende vormen van inferentie kunnen nuttig zijn.

[6] De intuïtieve reden is de volgende: het inductie-axioma zegt dat voor elke verzameling P die 0 bevat en gesloten is onder S, dat P alle domeinelementen bevat. Het inductieschema zegt hetzelfde, maar niet voor alle dergelijke verzamelingen P, maar enkel voor de verzamelingen van de vorm $\{x|\varphi[x]\}$ die uitgedrukt worden door een formule. Er zijn echter veel meer verzamelingen dan er formules zijn. Bv. in de context van de natuurlijke getallen: hoeveel deelverzamelingen van de natuurlijke getallen bestaan er? Hoeveel formules bestaan er? Daarom is het inductieschema zwakker dan het inductie-axioma.

Modelgeneratie:
Input: theorie T
Output: een model $\mathfrak{A}$ van T, of UNSAT indien T inconsistent is.

Het probleem hiermee is dat modellen oneindig kunen zijn. In het algmeen kunnen oneindige modellen niet berekend worden. Maar in veel situaties is de gezochte oplossing een eindige model. Deze kunnen wel berekend worden. (Zie Sectie 5.3.3).

In heel veel situaties is bovendien het domein eindig en gekend, en zijn veel data gegeven. We kunnen deze informatie vastleggen door middel van een eindige structuur $\mathfrak{A}_i$ die het domein en de waardes van gegeven symbolen vastlegt. We zoeken dan $\mathfrak{A}_i$ uit te breiden met waardes van symbolen die voorkomen in T maar niet door $\mathfrak{A}_i$ geïnterpreteerd worden, zodat we een model van T bekomen.

Definitie 4.5.1. We noemen $\mathfrak{A}$ een expansie van $\mathfrak{A}_i$ indien $D_{\mathfrak{A}} = D_{\mathfrak{A}_i}$ en voor elk symbool $\sigma \in \Sigma_{\mathfrak{A}_i}$ geldt dat $\sigma^{\mathfrak{A}} = \sigma^{\mathfrak{A}_i}$.

Laten we $\mathfrak{A}|_\Sigma$ definiëren als de structuur bekomen uit $\mathfrak{A}$ door deze te beperken tot de symbolen van Σ. Dan is $\mathfrak{A}$ een expansie van $\mathfrak{A}_i$ indien $\mathfrak{A}|_{\Sigma_{\mathfrak{A}_i}} = \mathfrak{A}_i$.

In de context van dergelijke problemen is de volgende vorm van modelgeneratie van toepassing:

Modelexpansie:
Input: theorie T, inputstructuur $\mathfrak{A}_i$
Output: een expansie $\mathfrak{A}$ van $\mathfrak{A}_i$ die voldoet aan T, of UNSAT indien T logisch inconsistent is.

De waardes die de oplossing vormen voor zulke problemen zijn meestal de in $\mathfrak{A}_i$ ongeïnterpreteerde symbolen van T.

Dergelijke problemen worden ook wel *constraint problemen* genoemd en het wetenschapsdomein dat zich bezig houdt met dergelijke problemen wordt *Constraint Programming* genoemd. Men zoekt men waarde, functies, arrays, matrices, die aan een aantal beperkingen voldoen. Arrays en matrices in Logica? Dat zijn functies! Bv. array a[1..n] of int is een functie $\{1, \ldots, n\} \to \mathbb{N}$. Een booleaanse array b[1..n] of boolean is een predicaat met domein $\{1, \ldots, n\}$. "Beperkingen" zijn eigenschappen en ze moeten in een formele taal uitgedrukt worden. In logische termen, zoekt men dus naar *modellen* van een aantal logische *zinnen*. Vertaald naar het logisch perspectief is het precies dit wat er gebeurt in Constraint Programming maar ook in informaticadisciplines zoals scheduling en planning.

De logische werkwijze voor dergelijke problemen is: stel een vocabularium Σ en een logische theorie T van Σ met de gewenste eigenschappen van de structuur die we zoeken. Zeer vaak is reeds een deel van het model gegeven: bv. het domein en de interpretaties van een aantal predikaat- en functiesymbolen. In dat geval is modelexpansie van toepassing.

Heel vaak zoekt men oplossingen die aan bepaalde optimaliteitscondities voldoen of zo goed mogelijk voldoen.

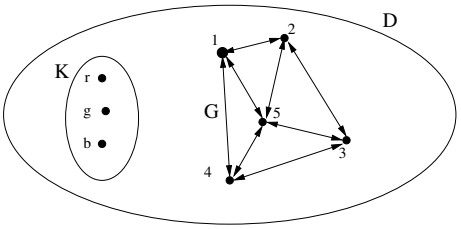
- We zoeken een plan dat zo snel mogelijk het doel bereikt en met zo weinig mogelijk middelen, bv. grondstoffen, energieverbruik, ...
- DHL zoekt een plan dat zoveel mogelijk postpakketten zo snel mogelijk ter plaatse brengt, met zo laag mogelijke transportkosten.

Er bestaan krachtige constraintprogrammatie-systemen met veel industriële toepassingen. De talen in deze systemen worden steeds expressiever en lijken ook steeds meer op logica. In LogicPalet zit modelexpansie in Decaworld. Hiervoor wordt het IDP-systeem gebruikt. Modelexpansie is 1 van de vormen van inferentie die het IDP-systeem aanbiedt. Het IDP-systeem in LogicPalet is één van de weinige logische modelgeneratoren voor predikatenlogica en ondersteunt een zeer rijke uitbreiding ervan. Op de webpagina van het IDP-systeem staan vele voorbeelden van modelexpansie-toepassingen en andere vormen van redeneren. `https://dtai.cs.kuleuven.be/software/idp/` (onder menu-items Online IDE, Demos, Examples). Het systeem kan ook gedownload worden vanop die pagina.

Voorbeeld 4.5.1. Het grafekleurprobleem, zie Voorbeeld 2.6.2. We kozen $\Sigma = \{G/2, K/1, C/1 : \}$. We zochten een structuur die aan de volgende formule voldoet:

$$\forall x\colon K(C(x)) \wedge \forall x\colon \forall y\colon (G(x,y) \Rightarrow \neg C(x) = C(y))$$

De grafe en de kleuren zijn gegeven in de vorm van een structuur $\mathfrak{A}_i$ die K en G interpreteert.



Het probleem hier is dus om deze structuur uit te breiden voor het symbool $C/1$ zodat we een model $\mathfrak{A}$ van deze theorie krijgen. De functie $C^{\mathfrak{A}}$ is dan de gewenste oplossing.

Voorbeeld 4.5.2. Het n-koninginneprobleem. Plaats n koninginnen op een nxn bord zodat ze elkaar niet kunnen aanvallen. Het vocabularium bestaat uit een predikaatsymbool $Dim/1$ zodat $Dim(i)$ betekent dat i een rij- of kolom-index is (het bord is vierkant), een functiesymbool $Queen/1$ zodat $Queen(i)$ de kolompositie van de koningin van de i-de rij is, en verder $+$.

Een oplossing van dit probleem voldoet aan volgende axioma's:

- Als r een rij-index van het schaakbord is, dan ook $Queen(r)$.

$$\forall r\colon (Dim(r) \Rightarrow Dim(Queen(r))$$

- Er staat hoogstens 1 koningin op een kolom:

$$\forall r1\colon \forall r2\colon (Dim(r1) \wedge Dim(r2) \wedge Queen(r1) = Queen(r2) \Rightarrow r1 = r2)$$

- Er staat hoogstens 1 koningin op een stijgende diagonaal:

$$\forall r1\colon \forall r2\colon (Dim(r1) \wedge Dim(r2) \wedge Queen(r1) + r2 = Queen(r2) + r1 \Rightarrow r1 = r2)$$

- Er staat hoogstens 1 koningin op een dalende diagonaal:

$$\forall r1\colon \forall r2\colon (Dim(r1) \wedge Dim(r2) \wedge Queen(r1) + r1 = Queen(r2) + r2) \Rightarrow r1 = r2)$$

We zoeken een model $\mathfrak{A}$ van deze theorie met domein $\mathbb{N}$ en $Dim^{\mathfrak{A}} = \{1, \ldots, n\}$. Het antwoord wordt gegeven door $Queen^{\mathfrak{A}}$ beperkt tot $\{1, \ldots, n\}$.

Voorbeeld 4.5.3. We bekijken een vereenvoudigd uurroosterprobleem in de context van de studentendatabank. We voegen een nieuwe relatie toe aan de databank: $Timeslot/1$ met alle time slots waarin lessen gegeven mogen worden in de campus. Het doel is om voor elk vak een tijdstip te bepalen zodat er geen conflicten optreden. We zullen deze -ongekende- relatie voorstellen door $Schedule/2$. We veronderstellen dat alle instructors van een vak steeds aanwezig willen zijn in de les. We maken ook gebruik van de type-predikaten $Vak/1, Docent/1, Student/1, Score/1$. We kunnen deze ofwel vervangen door hun definiërende formule, ofwel hun definitie toevoegen aan de theorie.

Aan welke voorwaarden voldoet een goed uurrooster?

- Typering van $Schedule$:

$$\forall x\colon \forall t\colon [Schedule(x,t) \Rightarrow Vak(x) \wedge Timeslot(t)]$$

- Elke vak vindt plaats op één tijdstip.

$$\forall x\colon [Vak(x) \Rightarrow \exists t\colon [Timeslot(t) \wedge Schedule(x,t) \wedge \forall s\colon (Schedule(x,s) \Rightarrow s = t)]]$$

- Een instructor geeft geen twee vakken op hetzelfde tijdstip.

$$\forall i\colon \forall v\colon \forall w\colon \forall t\colon (Instructor(i,v) \wedge Instructor(i,w) \wedge Schedule(v,t) \wedge Schedule(w,t) \Rightarrow v = w)$$

- Een student moet geen twee vakken op hetzelfde tijdstip bijwonen.

$$\forall i\colon \forall v\colon \forall w\colon \forall t\colon (Enrolled(s,v) \wedge Enrolled(s,w) \wedge Schedule(v,t) \wedge Schedule(w,t) \Rightarrow v = w)$$

Hoe kunnen we nu een oplossing voor ons uurroosterprobleem berekenen met deze theorie? We hebben een databank $\mathfrak{A}$, een theorie T en we zoeken een structuur $\mathfrak{B}$ die een model is van T en die een uitbreiding is van $\mathfrak{A}$. In zo'n model geeft $Schedule^{\mathfrak{B}}$ de gewenste schedule. Anders gezegd zoeken we $\mathfrak{A}$ uit te breiden met een interpretatie $Schedule^{\mathfrak{A}}$ zodat we een model van T bekomen.

Oefening 4.5.1. *De Einsteinpuzzel.*

Let us assume that there are five houses of different colors next to each other on the same road. In each house lives a man of a different nationality. Every man has his favorite drink, his favorite brand of cigarettes, and keeps pets of a particular kind.

- *The Englishman lives in the red house.*
- *The Swede keeps dogs.*
- *The Dane drinks tea.*
- *The green house is just to the left of the white one.*
- *The owner of the green house drinks coffee.*
- *The Pall Mall smoker keeps birds.*
- *The owner of the yellow house smokes Dunhills.*

- *The man in the center house drinks milk.*
- *The Norwegian lives in the first house.*
- *The Blend smoker has a neighbor who keeps cats.*
- *The man who smokes Blue Masters drinks beer.*
- *The man who keeps horses lives next to the Dunhill smoker.*
- *The German smokes Prince.*
- *The Norwegian lives next to the blue house.*
- *The Blend smoker has a neighbor who drinks water.*

The question to be answered is: Who keeps fish?

Kies een vocabularium om de basistypes (huizen, nationaliteiten, kleuren, cigaretten, huisdieren en drank) en hun interrelaties voor te stellen. Druk de eigenschappen uit. Een oplossing voor het probleem bekom je door een model te berekenen van deze theorie en te kijken wie de fish *bezit.*

Zo'n raadsel wordt in een fractie van een seconde opgelost door het IDP-systeem.

Opmerking 4.5.1. Van belang in deze sectie is om de definities van de relevante vormen van inferentie te kennen en het principe hoe dit type van problemen kunnen herleid worden tot deze inferentieproblemen. Er kunnen ook oefeningen gevraagd worden (zie de oefenzittingen hiervoor).

4.6 Automated theoremproving (ATP)

De typisch vorm van inferentie die velen associëren met predicatenlogica is deductieve inferentie (zie Hoofdstuk 3). Hiervan zijn verschillende instantiaties: bewijzen van inconsistentie, bewijs van logisch gevolg, bewijs van logische waarheid. De tweede is van deze vorm:

Deductieve inferentie:
Input: theorie T, formule A
Output: **t** indien $T \models A$, anders **f**.

Zoals we zullen zien in Hoofdstuk 5 kan deze vorm van inferentie niet volledig opgelost worden door middel van computers.

LogicPalet biedt een automatische theoremprover (ATP) aan via AskSpass. Spass is een bekend automatische ATP systeem. In ATP worden automatische theoremprovers ontwikkeld, methodes die gebruikt kunnen worden om inconsistentie, logische waarheid of logisch gevolg aan te tonen.

Er bestaat een grote diversiteit aan formele bewijsmethoden. KE-bewijzen zien er heel anders uit dan de bewijsmethodes die ontwikkeld werden ten tijde van Gödel. In het college 'Artificiële intelligentie' van 2BA wordt de *resolutiemethode* besproken die dateert van 1965 en wiens computerimplementatie soms sneller werkt maar waarvan de geleverde formele bewijzen veel minder begrijpbaar zijn voor ons, mensen.

Hedendaagse theoremprovers zijn zeer complexe systemen waarin vaak veel verschillende methodes en technieken gecombineerd worden. Bv. het systeem Spass dat gebruikt wordt in LogicPalet is gebaseerd op een combinatie van de resolutiemethode met een methode die zeer verwant is aan

de KE-methode. De geleverde formele bewijzen zijn echter moeilijk begrijpbaar voor mensen. Naast SPASS bestaan er nog veel andere systemen zoals bijvoorbeeld OTTER of Vampire.

Eén van de eerste successen van theoremprovers was het oplossen van het Robbins probleem in Boolse algebra, een open wiskundig probleem sinds 1933, dat in 1996 in 8 dagen werd opgelost door de theorem prover EQP en daarbij de New York Times haalde. Maar hoewel er sinds 1996 nog zeer grote vooruitgang geboekt werd is, in tegenstelling tot andere intellectuele activiteiten zoals bv. schaken, het bewijzen van wiskundige stellingen vooralsnog een domein waar mensen, wiskundigen, veruit superieur zijn aan computers.

Toch worden theoremprovers steeds vaker ingezet voor minder moeilijke maar vaak voorkomende deelproblemen in de context van informatica-toepassingen.

Theoremprovers hebben toepassingen in velden zoals:

- Deductieve databanken (zie Hoofdstuk 4).
- Software-verificatie (zie Hoofdstuk 4). Bv. theoremprovers worden door Siemens gebruikt in toepassingen zoals de ontwikkeling van de software (confer Metro 14 in Hoofdstuk 1).
- Hardware-verificicatie. Het ACL2 systeem werd gebruikt om de correctheid te bewijzen van floating point deling in AMD's PENTIUM-like AMD5K86 microprocessor. (Dit was overigens kort nadat Intel een milioenenverlies leed doordat een fout in hun floating point processor was ontdekt, waarna ze al die processoren gratis moesten vervangen.)
- Correctheids-bewijzen van programma's.
- Programmeertalen die volledig gebaseerd zijn op predikatenlogica en automated reasoning, bijvoorbeeld Prolog.
- Theoremproving wordt ook gebruikt in het semantisch web.

Een goed vertrekspunt om meer informatie te vinden is via de webpagina: `https://en.wikipedia.org/wiki/Automated_theorem_proving`.

In de volgende sectie zien we een toepassing van deductieve inferentie.

4.7 Formele methoden in Software Engineering

Meestal werken meerdere groepen mensen aan een groot software systeem. Elke groep werkt aan een onderdeel (component, klasse). Duidelijke en precieze communicatie tussen die groepen is essentieel. Het Nederlands is soms niet precies genoeg. Code is te gedetailleerd. Ondubbelzinnige specificatie van wat elke component moet doen is dan noodzakelijk.

Dit is op dit ogenblik vooral van belang voor systemen waar bugs zeer gevaarlijk zijn, op fysiek of economisch vlak.

- Controlesystemen voor de luchtvaart.
- Lanceren van ruimtetuigen[7]
- Medische toepassingen[8]
- Besturingssystemen voor kerncentrales.

[7] Meer dan 1 raket is neergestort door bugs in software.

[8] Door bugs in software van bestralingssystemen zijn mensen omgekomen.

- Besturingssystemen voor metros.
- Systemen die van primordiaal commercieel belang zijn voor een bedrijf[9]

Kritische systeemeigenschappen moeten met absolute zekerheid gegarandeerd kunnen worden.

Voorbeeld 4.7.1. Twee vliegtuigen mogen nooit opstijgen vanop dezelfde startbaan binnen een interval van 1 minuut.

Men zoekt dus naar methodes om de correctheid van systemen te bewijzen tijdens de ontwerpfase. Een oplossing is: gebruik een formele taal voor specificaties.

We illustreren een manier om dit te doen in de context van een vereenvoudigde databankapplicatie.

BibSys Als voorbeeld behandelen we een vereenvoudigd informatiesysteem voor het beheer van een bibliotheek, BibSys genaamd. BibSys bestaat uit:

- Een databank bestaande uit de volgende relaties:
  - $InHoldings/1$: de boeken in bezit van de bib.
  - $OnLoan/2$: boek ... is uitgeleend aan persoon ...
  - $Available/1$: boeken aanwezig in bib.
- Bibsys is een dynamisch systeem in de zin dat de databank steeds van toestand verandert. De programma's om bewerkingen op de databank uit te voeren noemt men *transacties*. Zij dienen om de bibliotheekdatabank aan te passen met nieuwe informatie:
  - `Borrow(u,x)`: registreer: persoon u leent boek x uit.
  - `Return(u,x)`: registreer: persoon u geeft boek x terug aan bib.
  - `Remove(x)`: registreer: boek x wordt uit het bezit van de bib verwijderd.
  - `Add(x)`: registreer: boek x wordt aangekocht door bib.

  Belangrijk: deze transacties zijn operaties op de databank. Het zijn geen predikaatsymbolen!

Gegeven is het databankvocabularium Σ bestaande uit $\{InHoldings/1, Available/1, OnLoan/2\}$ uitgebreid met symbolen voor boeken en gebruikers. De verschillende toestanden van de databank komen overeen met databankstructuren van Σ.

De eerste stap om deze toepassing te implementeren op een correcte manier is het expliciet maken van de integriteitsbeperkingen voor deze databank. Het zijn condities die voldaan zijn in elke correcte toestand van de databank. Ze worden uitgedrukt als logische zinnen over Σ.

- $E1$: Een boek is in het bezit van de bib asa het aanwezig is of uitgeleend is. Formeel:

$$\forall x\colon [InHoldings(x) \Leftrightarrow Available(x) \vee \exists u\colon OnLoan(x,u)]$$

  Deze eigenschap zal natuurlijk vaak niet voldaan zijn in een echte bib waar regelmatig boeken verloren gaan of gestolen worden. We maken daar abstractie van.

- $E2$: Geen enkel bibliotheekboek is tegelijkertijd uitgeleend en aanwezig. Formeel:

$$\forall x\colon [InHoldings(x) \Rightarrow \neg\exists u\colon OnLoan(x,u) \vee \neg Available(x)]$$

[9] Een bug in een Pentium chip kostte Intel ooit 600 miljoen dollar.

- $E3$: Geen enkel bibliotheekboek is uitgeleend aan meer dan één persoon. Formeel:

$$\forall x\colon [InHoldings(x) \Rightarrow \neg\exists u\colon \exists w\colon (u \neq w \wedge OnLoan(x,u) \wedge OnLoan(x,w))]$$

De theorie $\{E1, E2, E3\}$ (of de equivalente formule $E1 \wedge E2 \wedge E3$) zullen we verderop aanduiden als IB, de verzameling van Integriteits-Beperkingen van de databank.

Oefening 4.7.1. *Leg uit waarom de volgende twee verzamelingenuitdrukkingen niet logisch equivalent maar wel equivalent volgens IB zijn:*

- $\{x|InHoldings(x) \wedge Available(x)\}$
- $\{x|InHoldings(x) \wedge \exists u : OnLoan(x,y)\}$

De volgende stap is om de transacties te beschrijven. De databank kan enkel aangepast worden door middel van een aantal transactie procedures $\texttt{Trans}(\texttt{x}_1, \ldots, \texttt{x}_\texttt{n})$. Elke transactie beschrijven we door middel van twee formules: een pre-conditie en een postconditie :

- de **pre-conditie**-formule $Pre_{\texttt{Trans}}[x_1, \ldots, x_n]$ in Σ met vrije variabelen $x_1, \ldots, x_n$ die beschrijft onder welke voorwaarden de transactie mag plaats vinden.
- de **post-conditie**-formule $Post_{\texttt{Trans}}[x_1, \ldots, x_n]$ met vrije variabelen $x_1, \ldots, x_n$ die het effect op de databank beschrijft.

De preconditie kan uitgedrukt worden als een formule in Σ. Maar de postconditie kan niet gespecifieerd worden in Σ. Immers, een postconditie moet het verband beschrijven tussen de toestand vóór en de toestand ná de transactie. Je moet dus kunnen verwijzen naar twee toestanden van de databank in dezelfde formule! We hebben een nieuw stel symbolen Σ_N nodig: het zijn gewoon kopieën van de symbolen van Σ, voorafgegaan door N. Bv. $NInHoldings, NOnLoan, NAvailable$. De "$N$" slaat op "Next". Een postconditie zal dan een formule zijn van $\Sigma \cup \Sigma_N$.

We bespreken pre- en postcondities van de vier transacties.

- $\texttt{Borrow}(\texttt{u}, \texttt{x})$: $Pre_{\texttt{Borrow}}[u, x]$ is:

$$Available(x)$$

  De postcondities informeel:
  - $InHoldings$ blijft onveranderd. I.e., $InHoldings$ en $NInHoldings$ zijn identiek.
  - Een boek y is uitgeleend aan persoon w na transactie asa ($y = x$ en $w = u$) of y is uitgeleend aan w vóór transactie.
  - Een boek y is in $Available$ na de transactie asa $y \neq x$ en y is available vóór de transactie.

  Postconditie formeel, $Post_{\texttt{Borrow}}[u, x]$:

$$\begin{array}{l}\forall y\colon [NInHoldings(y) \Leftrightarrow InHoldings(y)] \wedge \\ \forall y\colon \forall w\colon [NOnLoan(y,w) \Leftrightarrow y = x \wedge w = u \vee OnLoan(y,w)] \wedge \\ \forall y\colon [NAvailable(y) \Leftrightarrow y \neq x \wedge Available(y)]\end{array}$$

- $\texttt{Return}(\texttt{u}, \texttt{x})$: Persoon u brengt boek x terug. De preconditie-formule $Pre_{\texttt{Return}}[u, x]$:

$$OnLoan(x,u)$$

  De postcondities informeel:

- *Inholdings* is onveranderd.
- Een boek y is uitgeleend aan persoon w na transactie asa $\neg(y = x$ en $w = u)$ en y is uitgeleend aan w vóór transactie.
- Een boek y is in *Available* na transactie asa $y = x$ of y is available vóór transactie.

Postcondities formeel: $Post_{\texttt{Return}}[u, x]$:

$$\forall y\colon [NInHoldings(y) \Leftrightarrow InHoldings(y)] \wedge$$
$$\forall y\colon \forall w\colon [NOnLoan(y, w) \Leftrightarrow \neg(y = x \wedge w = u) \wedge OnLoan(y, w)] \wedge$$
$$\forall y\colon [NAvailable(y) \Leftrightarrow y = x \vee Available(y)]$$

- `Remove(x)`: boek x wordt verwijderd. $Pre_{\texttt{Remove}}[x]$:

$$Available(x)$$

Postcondities informeel:

- Een boek y is *InHoldings* na de transactie asa $y \neq x$ en y is in *InHoldings* vóór de transactie.
- *OnLoan* is onveranderd.
- Een boek y is in *Available* na de transactie asa $y \neq x$ en y is in *Available* vóór de transactie.

Postcondities formeel: $Post_{\texttt{Remove}}[x]$:

$$\forall y\colon [NInHoldings(y) \Leftrightarrow y \neq x \wedge InHoldings(y)] \wedge$$
$$\forall y\colon \forall w\colon [NOnLoan(y, w) \Leftrightarrow OnLoan(y, w)] \wedge$$
$$\forall y\colon [NAvailable(y) \Leftrightarrow y \neq x \wedge Available(y)]$$

- `Add(x)`: boek x wordt aangeschaft. $Pre_{\texttt{Add}}[u, x]$:

$$\neg InHoldings(x)$$

Postcondities informeel:

- Een boek y is in *InHoldings* na de transactie asa $y = x$ of y is in *InHoldings* vóór de transactie.
- *OnLoan* is onveranderd.
- Een boek y is in *Available* na de transactie asa $y = x$ of y is in *Available* vóór de transactie.

Postcondities formeel: $Post_{\texttt{Add}}[u, x]$:

$$\forall y\colon [NInHoldings(y) \Leftrightarrow y = x \vee InHoldings(y)] \wedge$$
$$\forall y\colon \forall w\colon [NOnLoan(y, w) \Leftrightarrow OnLoan(y, w)] \wedge$$
$$\forall y\colon [NAvailable(y) \Leftrightarrow y = x \vee Available(y)]$$

Hiermee is de werking van Bibsys op logische manier gespecifieerd. Nu zien we drie toepassingen van deze specificaties. Telkens wordt een verschillend type van probleem opgelost door middel van de gepaste inferentie.

Verifiëren van correctheid van Bibsys databank De eerste toepassing de formele specificatie is om de correctheid van de Bibsys databank te controleren. De vorm van inferentie is waarheidevaluatie-inferentie.

- input: $\mathfrak{A}, IB$
- output: "waar" indien $\mathfrak{A} \models IB$, anders "onwaar"

Verifiëren van correctheid van pre- en postcondities Een belangrijke eigenschap voor een correcte databanksysteem is dat de transacties een zinvolle databank omzetten in een zinvolle databank, anders gezegd een databank die voldoet aan IB omzetten in een databank die opnieuw voldoet aan IB. Dit garandeert dat als de programmeur de transacties conform pre- en postcondities implementeert, dan zullen de transactie-programma's de integriteitsbeperkingen IB bewaren. Men zegt dan: de integriteitsbeperkingen zijn *invarianten* van de transacties.

Definieer IB_N de kopie van deze theorie door elk Σ-symbool te vervangen door zijn kopie in Σ_N.

De inferentietaak om dit probleem op te lossen is validiteitschecking-inferentie:

- Input: $IB, Pre_{\texttt{Trans}}[x_1, \ldots, x_n], Post_{\texttt{Trans}}[x_1, \ldots, x_n]$
- Output: "waar" indien de volgende logische zin logisch waar is:

$$\forall x_1 \colon \ldots \forall x_n \colon (IB \wedge Pre_{\texttt{Trans}}[x_1, \ldots, x_n] \wedge Post_{\texttt{Trans}}[x_1, \ldots, x_n] \Rightarrow IB_N)$$

  Anders "onwaar'.

Immers deze zin drukt uit dat als IB en de preconditie waar is in een toestand, en de postconditie geldt in de nieuwe toestand dan is ook IB voldaan in de nieuwe toestand.

Het bewijzen van logische waarheid van een formule is een deductieve redeneertaak. Hiervoor kunnen we Spass gebruiken, of andere theoremprovers. Vier formules moeten bewezen worden van logisch waar te zijn. De 4 formules verschillen enkel in de middelste gedeeltes, t.t.z. de pre- en postconditie.

Invariantie onder `Borrow[u,x]`**.** Te bewijzen: volgende formule is logisch waar:

$$
\begin{array}{l}
\forall u \colon \forall x \colon [\\
\forall x \colon [InHoldings(x) \Leftrightarrow Available(x) \vee \exists u \colon OnLoan(x,u)] \wedge \\
\forall x \colon [InHoldings(x) \Rightarrow \neg \exists u \colon OnLoan(x,u) \vee \neg Available(x)] \wedge \\
\forall x \colon [InHoldings(x) \Rightarrow \neg \exists u \colon \exists w \colon (u \neq w \wedge OnLoan(x,u) \wedge OnLoan(x,w))] \\
\wedge \\
Available(x) \\
\wedge \\
\forall y \colon [NInHoldings(y) \Leftrightarrow InHoldings(y)] \wedge \\
\forall y \colon \forall w \colon [NOnLoan(y,w) \Leftrightarrow y = x \wedge w = u \vee OnLoan(y,w)] \wedge \\
\forall y \colon [NAvailable(y) \Leftrightarrow y \neq x \wedge Available(y)] \\
\Rightarrow \\
\forall x \colon [NInHoldings(x) \Leftrightarrow NAvailable(x) \vee \exists u \colon NOnLoan(x,u)] \wedge \\
\forall x \colon [NInHoldings(x) \Rightarrow \neg \exists u \colon NOnLoan(x,u) \vee \neg NAvailable(x)] \wedge \\
\forall x \colon [NInHoldings(x) \Rightarrow \neg \exists u \colon \exists w \colon (u \neq w \wedge NOnLoan(x,u) \wedge NOnLoan(x,w))]]
\end{array}
$$

Oefening 4.7.2. *In deze formule wordt dezelfde variabele x 7 keer universeel gekwantificeerd, waaronder 6 kwantoren die voorkomen in de scoop van de eerste kwantor. Dit is verwarrend. Welke voorkomens van x worden gekwantificeerd door de eerste kwantor? Hernoem de variabelen zodat elke variabele maar 1 keer gekwantificeerd wordt. Verifieer of de condities van Propositie 3.4.4 voldaan zijn.*

Invariantie onder `Remove(x)`. Te bewijzen: volgende formule is logisch waar:

$$
\begin{array}{l}
\forall x\colon [\\
\forall x\colon [InHoldings(x) \Leftrightarrow Available(x) \vee \exists u\colon OnLoan(x,u)] \wedge\\
\forall x\colon [InHoldings(x) \Rightarrow \neg\exists u\colon OnLoan(x,u) \vee \neg Available(x)] \wedge\\
\forall x\colon [InHoldings(x) \Rightarrow \neg\exists u\colon \exists w\colon (u \neq w \wedge OnLoan(x,u) \wedge OnLoan(x,w))]\\
\wedge\\
Available(x)\\
\wedge\\
\forall y\colon [NInHoldings(y) \Leftrightarrow y \neq x \wedge InHoldings(y)] \wedge\\
\forall y\colon \forall w\colon [NOnLoan(y,w) \Leftrightarrow OnLoan(y,w)] \wedge\\
\forall y\colon [NAvailable(y) \Leftrightarrow y \neq x \wedge Available(y)]\\
\Rightarrow\\
\forall x\colon [NInHoldings(x) \Leftrightarrow NAvailable(x) \vee \exists u\colon NOnLoan(x,u)] \wedge\\
\forall x\colon [NInHoldings(x) \Rightarrow \neg\exists u\colon NOnLoan(x,u) \vee \neg NAvailable(x)] \wedge\\
\forall x\colon [NInHoldings(x) \Rightarrow \neg\exists u\colon \exists w\colon (u \neq w \wedge NOnLoan(x,u) \wedge NOnLoan(x,w))]]
\end{array}
$$

Oefening 4.7.3. *Schrijf de formules op die de invariantie van de transacties* `Return(u,x)` *en* `Add[x]` *uitdrukt.*

SPASS bewijst de logische waarheid van elk van de 4 formules in minder dan 0,1 seconde.

Oefening 4.7.4. *Bewijs de logische waarheid van de formule voor* `Remove(x)` *gebruikmakend van de KE-bewijsmethode.*

Een implementatie van Bibsys met logische inferentie De derde toepassing is: het gebruik van de specificatie om transacties te verrichten. Dus, in dit geval hoeven (bijna) geen programma's geschreven worden, maar wordt de transactie verricht door middel van logisch inferentie.

We introduceren een paar nuttige notaties en operatoren. Een databank is een Σ-structuur. Een structuur $\mathfrak{A}$ van het vocabularium $\Sigma \cup \Sigma_N$ stelt twee verschillende toestanden $\mathfrak{A}_o, \mathfrak{A}_n$ van de databank voor in 1 structuur:

- De "oude" databank $\mathfrak{A}_o$: de interpretatie van symbolen van Σ in $\mathfrak{A}$.
- De "nieuwe" databank $\mathfrak{A}_n$: de interpretatie van symbolen van Σ_N in $\mathfrak{A}$

Formeel, gegeven een $\Sigma \cup \Sigma_N$-structuur $\mathfrak{A}$, definiëren we twee Σ-databankstructuuren $O(\mathfrak{A})$ en $N(\mathfrak{A})$. Beide hebben hetzelfde domein als $\mathfrak{A}$ en dezelfde interpretatie voor alle constanten. Voor elk predicaatsymbool $P \in \Sigma$ geldt:

- $P^{O(\mathfrak{A})} = P^{\mathfrak{A}}$, de oude waarde in $\mathfrak{A}$.
- $P^{N(\mathfrak{A})} = NP^{\mathfrak{A}}$, de nieuwe waarde in $\mathfrak{A}$.

De vereiste inferentietaak wordt update-inferentie genoemd:

- Input: de oude databank $\mathfrak{A}_o$, de transactie $\mathtt{Trans}(\mathtt{c_1}, \ldots, \mathtt{c_n})$ met preconditie $Pre_{\mathtt{Trans}}[x_1, \ldots, x_n]$ en postconditie $Post_{\mathtt{Trans}}[x_1, \ldots, x_n]$.

- Output: "ERROR" indien $\mathfrak{A} \models \neg Pre_{\texttt{Trans}}[c_1, \ldots, c_n]$
  anders bereken de $\Sigma \cup \Sigma_N$-structuur $\mathfrak{A}$ die een expansie is van $\mathfrak{A}_o$ en voldoet aan de theorie $Post_{\texttt{Trans}}[c_1, \ldots, c_n]$. Return $N(\mathfrak{A}')$, de nieuwe databank in $\mathfrak{A}$.

Voorbeeld 4.7.2. Veronderstel dat we de transactie `Remove(B143)` willen uitvoeren in de huidige databank $\mathfrak{A}_o$.

- Stap 1: bereken de waarheid van

  $$\mathfrak{A}_o \models Available(B143)$$

  Indien niet voldaan, geef output "ERROR" en stop: de preconditie van de transactie is niet voldaan.

- Stap 2: voer modelexpansie uit op $\mathfrak{A}_o$ en de postconditie-theorie:

  $$\begin{array}{l} \forall y\colon [NInHoldings(y) \Leftrightarrow y \neq B143 \wedge InHoldings(y)] \wedge \\ \forall y\colon \forall w\colon [NOnLoan(y,w) \Leftrightarrow OnLoan(y,w)] \wedge \\ \forall y\colon [NAvailable(y) \Leftrightarrow y \neq B143 \wedge Available(y)] \end{array}$$

  Modelexpansie vindt exact 1 $\Sigma \cup \Sigma_N$-model $\mathfrak{A}'$. Want deze 3 formules definiëren de waardes van de Σ_N-symbolen in termen van die van de Σ-symbolen. Bv. $NOnLoan^{\mathfrak{A}'} = OnLoan^{\mathfrak{A}'} = OnLoan^{\mathfrak{A}}$. Tenslotte, geef als output $N(\mathfrak{A}')$, dit is de nieuwe toestand van de databank.

We hebben hier beschreven hoe een update-inferentie voor een transactie kan geïmplementeerd worden door middel van elementaire inferentiestappen. De vorm(en) van inferentie die gebruikt werden zijn: *waarheidevaluatie en modelexpansie.*

De methode die hier beschreven wordt is niet efficient en is niet practisch bruikbaar. Er bestaan geen modelgeneratoren die dit probleem kunnen oplossen voor databanken die gigabytes groot kunnen zijn. Er bestaan echter veel slimmere methodes die toch gebruik maken van een logische specificatie van transacties. We gaan hierop niet in.

Programming by contract. Programmeurs schrijven code voor elke transactie. Zij moeten de pre- en postcondities van een transactie beschouwen als een contract dat geïmplementeerd moet worden.

Correctheid van de code voor een transactie betekent: wanneer de code uitgevoerd wordt in toestand die aan de precondities voldoet dan stopt het programma na een eindig aantal stappen in een toestand die voldoet aan de postcondities. M.a.w, het contract is correct geïmplementeerd.

Zelfs als het contract van transacties correct werd gevolgd, de invarianten van de integriteitsbeperkingen werden bewezen en de initiële databank voldeed aan de integriteitsbeperkingen, dan biedt dit toch geen 100% garantie dat er geen fouten kunnen binnensluipen. Bv. een computer kan crashen midden in een transactie, met de transactie slechts half uitgevoerd. Databanksystemen voorzien oplossingen hiervoor.

Oefening 4.7.5. *Beschouw volgende transacties op StudentenDatabank:*
- *$AddGrade(student, vak, score)$: toevoegen aan tabel Grade.*
- *$EnrollStudent(student, vak)$: toevoegen aan tabel Enrolled.*

Formuleer de voor de hand liggende pre- en postcondities voor deze transacties, zodanig dat het volgende gegarandeerd is:

- *Een student die een score voor een vak verkrijgt moet daarvoor ingeschreven zijn.*
- *Een student die een score voor een vak verkrijgt moet automatisch voor dat vak uit de tabel Enrolled verdwijnen. Hij mag zich eventueel later terug inschrijven.*
- *Iedereen die ingeschreven is voor een vak moet geslaagd zijn voor alle vakken die nodige voorkennis zijn voor dat vak.*
- *Iemand die ingeschreven is of reeds geslaagd is voor een vak mag zich niet inschrijven voor dat vak.*

4.8 Conclusie

In elk toepassingsdomein van de Informatica is er informatie. Overal kan men de volgende vragen stellen:

- Welke soort of soorten informatie is beschikbaar?
- Welke formele talen of taalconstructies hebben we nodig om deze uit te drukken?
- Hoe kan de informatie voorgesteld worden in logica?
- Welke soorten van redeneren hebben we nodig om concrete informaticaproblemen op te lossen met de gegeven informatie.

In dit hoofdstuk hebben we dergelijke vragen onderzocht in verschillende deelgebieden van de informatica. We hebben gezien hoe de informatie in logica voor te stellen en hoe we een concreet informaticaprobleem kunnen herleiden tot een inferentie-probleem over deze theorie. Als we dan beschikken over een efficiënte generische solver voor dit type van inferentieprobleem, dan hoeven we geen programma meer te schrijven! Dat is de “dream, the quest, the holy grail” waarover Bill Gates het had in het citaat pagina 10 van deze cursus.

Het inferentie-probleem waar het gebruik van logica en generische solvers het sterkst is doorgedrongen in de praktijk is voor databanken. SQL (of toch een belangrijk fragment ervan) kan beschouwd worden als een expressieve logische taal. De kost om een programma te schrijven om een query-probleem op te lossen is een veelvoud van het formuleren ervan in SQL; en bovendien zal een databasesysteem de query meestal efficiënter kunnen oplossen dan een zelfgeschreven programma. Het query-probleem is een eenvoudig type van inferentie-probleem. Ook voor meer complexe inferentie-problemen zoals vereist in constraint programming zijn belangrijke successen geboekt. In andere informatica-domeinen is nog een lange weg af te leggen vooralleer Gates’ droom werkelijkheid wordt. Maar zelfs dan is logica vaak een uitzonderlijk krachtig instrument om inzicht te krijgen in de aard van informatie en de probleemstelling.

Hiermee sluiten we de toepassingen van logica en redeneren voor de informatica af.

Hoofdstuk 5

Algoritmes, Berekenbaarheid, het Halting problem, Onbeslisbaarheid en de Onvolledigheidsstelling van Gödel

In dit hoofdstuk gaan we in op het concept van berekening en we onderzoeken op een ruwe manier hoe de vormen van inferentie berekend kunnen worden.

Wat is een berekening? Een eindige opeenvolging van elementaire berekeningsstappen, bv. in de natuurlijke getallen $+, \times, -, \div, \%, \ldots$. Als we if-then-else testen invoeren, kunnen we *bomen* van berekeningen introduceren door zulke testen te nesten. Daarbij is elke tak een opeenvolging van testen en berekeningsstappen. Zo'n boom kan oneindig groot zijn en oneindige takken bevatten of oneindig veel takken bevatten. Wat is een *algoritme*? Een eindige beschrijving van een potentieel onbegrensde berekeningsboom. Bv. een algoritme dat bij invoer een getal n krijgt en berekent of n een priemgetal is, daarvan zal de berekeningsboom onbegrensd groot zijn aangezien voor steeds grotere n steeds meer stappen gezet moeten worden om te bepalen of n een priemgetal is. Hoe kunnen we onbegrensd grote berekeningsbomen op eindige manier voorstellen? Door middel van de while-lus, of for-lus of de GOTO. Hierdoor kunnen stukken berekening herhaald worden (op andere waardes).

Wat kan allemaal berekend worden? Bv. we kunnen som van twee eindige natuurlijke getallen of floating point getallen, of matrices berekenen. We kunnen in principe berekenen of een getal n al dan niet priem is. Kunnen we ook oneindige objecten berekenen? Bv. kunnen we de verzameling van alle priemgetallen berekenen? Niet in eindige tijd natuurlijk, maar wel als we het programma oneindig lang mogen laten lopen.

Computergeheugens bestaan uit bits 0 of 1. Alle soorten objecten in een computergeheugen (strings, floating points, bools, matrices, arrays) bestaan uit bitvectoren. Bv. floating point getallen $0,26542 * 10^{23}$ worden intern voorgesteld door een combinatie van een eindige mantisse 2654 en een exponent 23. Bitvectoren komen overeen met getallen of verzamelingen van natuurlijke getallen van begrensde grootte. Maw, alles wat te berekenen valt, valt te herleiden tot berekeningen van natuurlijke getallen. Dus is het logisch om de vraag "wat kan berekend worden?" te bestuderen in de context van de natuurlijke getallen.

Hierboven werden een aantal vragen gesteld over de fundamenten van de informatica (wat is een

algoritme? wat kunnen we berekenen?). We zullen hier een paar van deze vragen beantwoorden.

5.1 Registermachines, Berekenbaarheid en de hypothese van Church

De eerste stap is een formele definitie van wat een "algoritme" nu precies is. We definiëren daarvoor een soort geïdealiseerde machinetaal, de taal van *registermachines*.

Definitie 5.1.1. Een registermachine bestaat uit

- Een eindig aantal *registers* $R_0, R_1, R_2, R_3, \cdots$ die elk een willekeurig groot natuurlijk getal kunnen bevatten. R_0 speelt een speciale rol en wordt de *accumulator* genoemd.
- Een *processor* die operaties kan uitvoeren op de getallen in de registers, en dit volgens een welbepaald *programma* dat ingebouwd is in de machine.
- Het programma van de machine is een genummerde eindige rij van *instructies*. Een instructie is een van de volgende bevelen:

Symbool	Betekenis
B R A n	Breng getal in Register R_n naar Accumulator. $R_0 := R_n$; ga naar de volgende instructie.
B A R n	Breng getal in Accumulator naar Register R_n. $R_n := R_0$; ga naar de volgende instructie.
C O N n	Breng het getal n (een concreet vast Constant getal) in de accumulator. $R_0 := n$; ga naar de volgende instructie.
O P T n	$R_0 := R_0 + R_n$; ga naar de volgende instructie.
A F T n	$R_0 := R_0 \dot{-} R_n$; ga naar de volgende instructie. – $R_0 \dot{-} R_n = 0$ indien $R_0 - R_n < 0$ – $R_0 \dot{-} R_n = R_0 - R_n$ indien $R_0 - R_n \geq 0$
V E R n	$R_0 := R_0 \times R_n$; ga naar de volgende instructie.
D E L n	$R_0 := R_0 \div R_n$; ga naar de volgende instructie. – $R_0 \div R_n = 0$ indien $R_n = 0$. – $R_0 \div R_n$ is gehele deling indien $R_n \neq 0$.

S P R n	(Sprong) Ga naar de instructie met nummer n.
V S P n	(Voorwaardelijke Sprong) If R0=0 then GOTO n else ga naar volgende instructie.
S T O P	Stop.

De n na een bevel is een concreet natuurlijk getal. Voor sommige bevelen moet n verwijzen naar een bestaand register; voor andere naar een bestaande instructie. Bijvoorbeeld **S P R** n met n groter dan het aantal instructies is niet toegelaten.

We spreken af dat de registers van een registermachine exact de registers zijn die gebruikt worden in zijn programma. Ook spreken we af dat **S T O P** de de laatste instructie is van een programma en nergens anders voorkomt. (Vanop andere plaatsen waar je wilt stoppen spring je gewoon naar de laatste instructie).

Bespreking De vraag waarmee we deze sectie zijn begonnen was "wat is een algoritme?". Het antwoord dat we hier geven is:

een algoritme is een registermachine.

Dit roept veel vragen op. Bv. kunnen we alles wat computers kunnen berekenen implementeren met een registermachine? Dat is inderdaad zo. Daar komen we straks op terug (bij de hypothese van Church).

Een registermachine is een idealisering van een computer. Het is een "denkbeeldige" computer. In een reële computer is het geheugen begrensd en kunnen er in de geheugencellen slechts getallen met een beperkt aantal cijfers staan. Een registermachine daarentegen heeft registers waarin getallen van onbegrensde grootte kunnen opgeslagen worden. Anderzijds kan de hardware van een reële computer altijd aangepast worden met registers die grotere getallen kunnen bevatten naargelang de noden. Het maken van idealiseringen is overigens gangbare en onvermijdelijke praktijk in de wetenschap.

Er lijkt ook een kloof te gapen tussen registermachines en hedendaagse computers wat betreft het soort datastructuren. In moderne programmeertalen zijn er integers, floats, arrays, strings, records, pointerstructuren, op maat gemaakte klassen, enz. De afstand tot de eindige serie van getallenregisters van een registermachine lijkt groot. Maar zoals gezegd, deep down in een computer wordt alles voorgesteld door middel van een serie bits. Al die complexe datastructuren kunnen gecodeerd worden als getallen. Die complexe datastructuren vormen dus geen fundamenteel verschil met registermachines.

Verderop in het hoofdstuk zullen we trouwens zien hoe registermachines zelf gecodeerd kunnen worden als een getal.

Een registermachine is deterministisch in de zin dat het voor elke invoer een uniek en welbepaald gedrag zal vertonen. De invoer bepaalt de sequentie van toestanden van de registermachine vol-

ledig. In die zin komt een registermachine eerder overeen met een programma in machinecode dan met een computer. Overigens, ook de vroegste computers waren deterministisch aangezien het programma vastgecodeerd was. Maar laat ons ook niet vergeten dat in computers programmas worden opgeslagen als sequenties van bits, wat overeenkomt met een getal. Een computer kan dus beschouwd worden als een deterministisch programma dat als invoer onder andere een getal krijgt dat een programma codeert. Dus is de afstand tussen registermachines en computers niet zo groot.

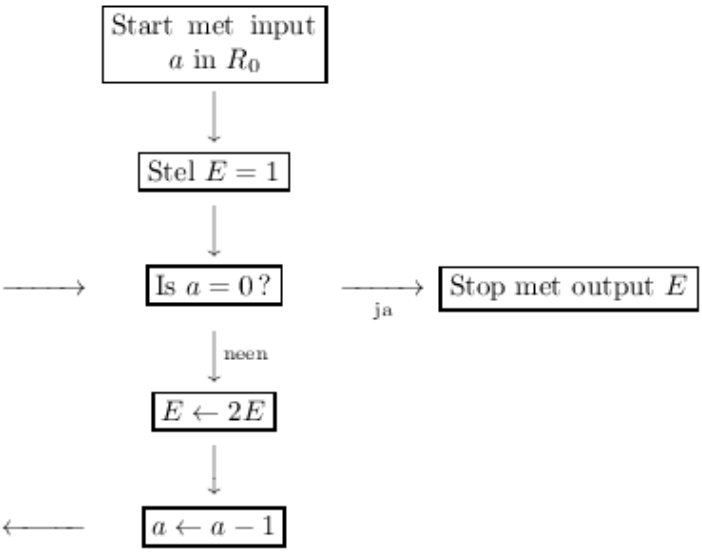
Het is overigens ook mogelijk een universele registermachine te bouwen die als invoer o.a. een getal krijgt dat een registermachine codeert, en bij uitvoering de laatste zal simuleren. Deze ene registermachine kan dus elke andere registermachine uitvoeren! We zullen hier verder niet op ingaan.

Definitie 5.1.2. We zeggen dat een registermachine M bij *input* $a_0, a_1, a_2, \cdots, a_k$ stopt na een eindig aantal stappen met *output* b (in de accumulator) als, wanneer men de machine start met in registers $R_0, R_1, \cdots, R_k$ de natuurlijke getallen $a_0, a_1, \cdots, a_k$ *en in alle andere registers nul*, de machine stopt na een eindig aantal stappen met uiteindelijk b in register R_0 (en niet nader bepaalde getallen in de andere registers).

Berekenbaarheid

Voorbeeld 5.1.1. Geef een programma voor een registermachine die bij input $a \in \mathbb{N}$ stopt na een eindig aantal stappen met output 2^a.

Oplossing. Vooraleer we het programma schrijven, stellen we eerst een flowchart op:



Om dit te encoderen kiezen we registers die overeenkomen met de "variabelen" van deze flowchart. We plaatsen a in register R_1, E in register R_2, en 1 in register R_3.

Het programma is als volgt:

1 **B A R** 1
2 **C O N** 1
3 **B A R** 2
4 **B A R** 3
5 **B R A** 1
6 **V S P** 14
7 **C O N** 2
8 **V E R** 2
9 **B A R** 2
10 **B R A** 1
11 **A F T** 3
12 **B A R** 1
13 **S P R** 6
14 **B R A** 2
15 **S T O P**

Hieruit volgt dat de functie $\mathbb{N} \to \mathbb{N} : a \to 2^a$ berekenbaar is met een registermachine.

Definitie 5.1.3. Een afbeelding

$$f : \mathbb{N}^k = \underbrace{\mathbb{N} \times \mathbb{N} \times \cdots \times \mathbb{N}}_{k \text{ keren}} \to \mathbb{N}$$

is berekenbaar door middel van een registermachine als er een registermachine M bestaat die bij iedere input $a_1, a_2, \cdots, a_k \in \mathbb{N}$ stopt na een eindig aantal stappen met output $f(a_1, \cdots, a_k)$.

Definitie 5.1.4. Een relatie R op $\mathbb{N}$ met k argumenten is *berekenbaar (of beslisbaar) door middel van een registermachine* als de afbeelding

$$\begin{aligned} f_R : \mathbb{N}^k \to \mathbb{N} : (a_1, \cdots, a_k) &\mapsto 1 && \text{als } (a_1, \cdots, a_k) \in R \\ (a_1, \cdots, a_k) &\mapsto 0 && \text{als } (a_1, \cdots, a_k) \notin R \end{aligned}$$

berekenbaar is door middel van een registermachine.

We zeggen dat R *semi-berekenbaar (of semi-beslisbaar) is door middel van een registermachine* als er een registermachine bestaat die voor elke input $(a_1, \ldots, a_n) \in R$ eindigt en 1 antwoordt, en voor elke input $(a_1, \ldots, a_n) \notin R$ eindigt en 0 antwoordt ofwel niet eindigt.

Bij een gegeven input kan een registermachine die een booleaanse functie implementeert op drie manieren reageren: antwoorden met 1, met 0, of in een oneindige lus gaan. Een berekenbare relatie R heeft een registermachine die voor elke invoer $a_1, \cdots, a_k$ zal eindigen en correct aangeeft of $(a_1, \cdots, a_k) \in R$. Een semi-berekenbare relatie R heeft een registermachine waarvoor slechts gedeeltelijk garantie is dat het zal stoppen, namelijk als $(a_1, \cdots, a_k) \in R$. Het verschil tussen berekenbaarheid en semi-berekenbaarheid zit hem dus in het termineren van de registermachine.

Het spreekt vanzelf dat een berekenbare relatie ook semi-berekenbaar is. Het omgekeerde geldt niet.

Daar een deelverzameling van $\mathbb{N}$ een 1-voudige relatie is, kunnen we dus ook spreken over deelverzamelingen van $\mathbb{N}$ die berekenbaar (beslisbaar) of semi-berekenbaar (semi-beslisbaar) zijn door middel van een registermachine.

Wat hebben we tot nog toe gedaan? We hebben een soort van machinetaal voorgesteld waarin programma's kunnen beschreven worden om bepaalde relaties te berekenen. Maar er zijn zo

veel machinetalen. Kunnen daarmee andere relaties berekend worden? Bv. kun je nieuwe nuttige en praktisch realiseerbare operaties bedenken die als je ze zou toevoegen aan de taal van registermachines zouden toelaten om méér relaties te berekenen? Men gaat ervan uit dat dit onmogelijk is.

Hypothese 5.1.1 (van Church)**.** *Elke afbeelding $f : \mathbb{N}^k \to \mathbb{N}$ die berekenbaar is door middel van een "algoritme" ($\stackrel{def}{=}$ een rekenwijze die op een computer kan geprogrammeerd worden) is berekenbaar door middel van een registermachine.*

De bovenstaande hypothese kan niet wiskundig bewezen worden. Omdat de definitie van "algoritme" vaag en niet formeel wiskundig gedefinieerd is. Als men met "algoritme" een programma van een registermachine bedoelt, dan valt er niets te bewijzen. Er is echter een hele reeks wiskundige definities bedacht voor "algoritme". Voor elke redelijke definitie van "algoritme" die men ooit heeft voorgesteld, heeft men de hypothese van Church bewezen. Deze hypothese wordt daarom algemeen aanvaard.

Voor een uitvoerige bespreking verwijzen we naar hoofdstuk V in het boek van Minsky : Computation, finite and infinite machines, Prentice-Hall (1972).

5.2 Onbeslisbaarheid van het stop-probleem

Wanneer een informaticus een probleem krijgt om op te lossen zal hij zich de vraag stellen of hij een programma kan schrijven dat de oplossing van het probleem berekent.

Voor eenvoudige taken is dit niet zo moeilijk: bereken het maximum van twee getallen, sorteer een eindige rij van getallen van klein naar groot, enz. Maar wat met een probleem van de vorm: ga na of een formule waar is in de natuurlijk getallen, ga na of een formule logisch consistent is, of logisch waar? Dit zijn veel complexere problemen, en de vraag moet gesteld worden: zijn er wel programmas die deze problemen kunnen oplossen?

In de context van registermachines wordt de vraag herleid tot de vraag: is een gegeven functie of relatie berekenbaar door middel van een registermachine? Het is eenvoudig in te zien dat er verzamelingen, relaties en functies bestaan die niet berekenbaar zijn. Er is maar een aftelbaar aantal registermachines en dus zijn er maar een aftelbaar aantal berekenbare verzamelingen en functies. Daarentegen zijn er overaftelbaar veel deelverzamelingen en functies van de natuurlijke getallen. Dus bestaan er verzamelingen en functies die niet berekenbaar zijn.

Het is dus eenvoudig in te zien dat er onberekenbare verzamelingen moeten bestaan. Echter, het is daarom niet eenvoudig voor een gegeven verzameling of functie te bepalen of deze berekenbaar is.

In deze sectie introduceren we een beroemd informatica-probleem: het *halting problem*, of het stop-probleem. We tonen aan dat het niet berekenbaar is.

Definitie 5.2.1. Het stop-probleem is het probleem om te bepalen of een gegeven registermachine bij een gegeven input al dan niet na een eindig aantal stappen zal stoppen.

De vraag die we ons hier stellen is: is het mogelijk om een algoritme te schrijven dat het stop-probleem oplost? Zo'n algoritme zou als invoer een voorstelling van een registermachine

M en een invoer m krijgen, en zou dan na een eindig aantal rekenstappen stoppen met het antwoord 1 indien M stopt met invoer m, en anders met 0. Door de hypothese van Church zou er dan ook een registermachine M_{stop} moeten bestaan die dit berekent! Er moet daarvoor wel eerst een praktisch probleem opgelost worden: een registermachine neemt enkel getallen als invoer, geen andere registermachines. Dus ontwikkelen we eerst een methode om willekeurige registermachines M te "coderen" als een natuurlijk getal.

We gaan aan elke registermachine M een natuurlijk getal associëren, het *codenummer* M genoemd. Eerst associëren we aan elke instructie een codenummer, volgens de onderstaande tabel:

Instructie	Codenummer
B R A n	$1 + 10n$
B A R n	$2 + 10n$
C O N n	$3 + 10n$
O P T n	$4 + 10n$
A F T n	$5 + 10n$
V E R n	$6 + 10n$
D E L n	$7 + 10n$
S P R n	$8 + 10n$
V S P n	$9 + 10n$
S T O P	10

Uit elk getal kunnen we berekenen of het met een instructie overeenkomt, en zo ja, welke instructie. Bv. 20 komt niet overeen met een instructie. Het getal 34 komt overeen met **O P T** 3.

Zij nu M een registermachine met programma P. Het *codenummer* van M is per definitie

$$\prod_i p_i^{(\text{codenummer van de } i^{\text{de}} \text{ instructie van } P)}$$

waarbij p_i het i^{de} priemgetal is, (bijvoorbeeld $p_1 = 2, p_2 = 3, p_3 = 5, p_4 = 7, p_5 = 11, \dots$).

Hier stelt $\prod_i$ het product voor over alle $i = 1, 2, \cdots$, (aantal instructies in P). Omdat elk getal van $\mathbb{N}_0$ slechts op één wijze in priemfactoren kan ontbonden worden, kan men uit het codenummer van M het programma P van M volledig reconstrueren. Anderzijds is niet elk getal in $\mathbb{N}_0$ een codenummer van een registermachine.

Notatie 5.2.1. De registermachine met codenummer n stellen we voor door M_n.

Indien $n \in \mathbb{N}$ geen codenummer van een registermachine is, dan spreken we af dat M_n de machine is met programma enkel de instructie **S T O P**. Merk op dat de code van het programma met enkel de instructie **S T O P** gelijk is aan 2^{10}. Het programma van de machine $M_{2^{10}}$ is dus

1 **S T O P**.

Dus, als n geen codenummer van een registermachine is, geldt $M_n = M_{2^{10}}$.

Nu keren we terug naar onze vraag: kan er een registermachine M_{stop} bestaan om het stopprobleem op te lossen? Deze machine zou twee inputs hebben: een code n (van de registermachine M_n) en een getal m. M_{stop} zou na eindig aantal stappen stoppen met 1 in de accumulator R_0 als M_n stopt met invoer m, en met 0 in R_0 als M_n niet stopt bij invoer m.

We zullen bewijzen dat zo'n registermachine niet kan bestaan, en dus, door de Hypothese van Church, dat er geen algoritme kan bestaan om het stop-probleem op te lossen.

Neem de volgende verzameling:

$$\{n \in \mathbb{N}_0 | M_n \text{ bij input } n \text{ stopt na een eindig aantal stappen } \}$$

Dit is een speciaal geval van het stop-probleem, namelijk voor $n = m$. Als het stop-probleem berekenbaar is, dan zou deze verzameling zeker ook berekenbaar zijn. De volgende stelling zegt echter dat deze verzameling *niet* berekenbaar is!

Stelling 5.2.1. *(Onbeslisbaarheid van het stop-probleem) De verzameling*

$$\{n \in \mathbb{N}_0 | M_n \textit{ bij input } n \textit{ stopt na een eindig aantal stappen } \}$$

is niet berekenbaar door middel van een registermachine.

Anders gezegd, er bestaat geen registermachine die voor elke input n stopt en 1 antwoordt indien M_n stopt bij invoer n, en 0 indien M_n niet stopt bij invoer n. Uit de hypothese van Church volgt dan dat er geen algoritme bestaat voor het stop-probleem.

Bew. Neem de afbeelding $h : \mathbb{N} \to \mathbb{N}$ die overeenkomt met de verzameling van de stelling (volgens Definitie 5.1.4). Deze is gedefiniëerd door

$$\begin{array}{ll} h(n) = 1 \ , & \text{als } M_n \text{ bij input } n \text{ stopt na een eindig aantal stappen} \\ h(n) = 0 \ , & \text{in het ander geval.} \end{array}$$

We dienen dus aan te tonen dat h niet berekenbaar is door middel van een registermachine.

Het bewijs is uit het ongerijmde. Veronderstel dat h *wel* berekenbaar is door middel van een registermachine. Dan bestaat er een registermachine M die bij elke input $n \in \mathbb{N}$ stopt na een eindig aantal stappen met output $h(n)$ in de accumulator R_0. We moeten nu tot een contradictie trachten te komen.

Zij P het programma van M met N het rangnummer van de laatste instructie **S T O P**, en zij P' het programma bekomen door in het programma P deze instructie **STOP** te vervangen door de volgende 3 bevelen toe te voegen:

$$\begin{array}{ll} (N) & \mathbf{V\ S\ P}\ N+2 \\ (N+1) & \mathbf{S\ P\ R}\ N \\ (N+2) & \mathbf{S\ T\ O\ P} \end{array}$$

Zij M' de registermachine met programma P'. Als er op het ogenblik dat gesprongen wordt naar $N + 1$ een nul staat in de accumulator, dan stopt M' bij de volgende stap. In het andere geval gaat M' in een oneindige lus.

We hebben dus:

$$M' \text{ bij input } n \text{ stopt na een eindig aantal stappen}$$
$$\Updownarrow$$

M bij input n stopt na een eindig aantal stappen en heeft dan als output 0 in de accumulator

$$\Updownarrow$$

$$h(n) = 0$$

$$\Updownarrow$$

M_n bij input n stopt *niet* na een eindig aantal stappen

Zij e het codenummer van de registermachine M'. Dus M' is M_e. Dan hebben we voor elke $n \in \mathbb{N}$:

M_e bij input n stopt na een eindig aantal stappen

$$\Updownarrow$$

M_n bij input n stopt *niet* na een eindig aantal stappen

Stellen we nu n gelijk aan e, dan bekomen we de volgende contradictie:

M_e bij input e stopt na een eindig aantal stappen

$$\Updownarrow$$

M_e bij input e stopt *niet* na een eindig aantal stappen

Dit is natuurlijk onmogelijk. Dus was onze assumptie verkeerd. M.a.w. er bestaat geen registermachine M die kan beslissen of M_n stopt bij input n. Deze contradictie beëindigt het bewijs van de stelling. Q.E.D.

Een ietwat meer verbose formulering van het bewijs vind je in Figuur 5.1.

5.3 Berekenbaarheid van inferentieproblemen

In vorig hoofdstuk hebben we verschillende vormen van inferentie gezien. We bestuderen hier of ze te implementeren zijn en zo ja, hoe. Inferentieproblemen werden gespecifeerd in de vorm:

> Inferentieprobleem
> Input: een logische formule of theorie, en eventueel andere argumenten
> Output: ...

Een inferentieprobleem definieert een functie van input naar output. Als de input en output ervan codeerbaar zijn als (n-tallen van) getallen, komt een dergelijke functies overeen met een functie of relatie op de natuurlijke getallen. Dan stelt zich de vraag of deze functie of relatie berekenbaar is.

Niet alle gespecifieerde vormen van inferentie kunnen berekenbaar zijn. Bv. het berekenen van waarheid van een formule A in een willekeurige structuur $\mathfrak{A}$ kan onmogelijk berekenbaar zijn. Immers er zijn overaftelbaar veel structuren en dus is de klasse van structuren niet codeerbaar in de natuurlijke getallen. Daarentegen zijn er wel aftelbaar veel structuren met een eindig domein (modulo isomorfisme) en deze kunnen wel gecodeerd worden als getallen. Ook formules kunnen gecodeerd kunnen worden als getallen. Deze techniek werd uitgevonden door Gödel en is geheel gelijkaardig aan de manier waarop registermachines gecodeerd kunnen worden als getallen. Dus kan het probleem van bepalen van waarheid van formules in een eindige structuur gecodeerd worden als een binaire relatie op de natuurlijke getallen. Op die relatie is de theorie van berekenbaarheid van toepassing.

De lusvinder genept – een logisch recept
een eenvoudig bewijs van de onbeslisbaarheid van het stopprobleem

Geen programma kan zeggen of een ander ooit klaar is.
Dat beweer ik, en meer: ik laat zien dat het waar is.
Hoe je ook sputtert, ik toon aan dat het klopt:
Je kunt niet voorspellen of een programma ooit stopt.

Stel dat P, een programma, dat toch kan voorspellen,
en dus broncodes checkt om aan jou te vertellen
of die code in een lus raakt en maar draait en maar spint,
en P schrijft "dit stopt" als hij géén lussen vindt.

Je geeft P jouw code, met invoer erbij.
En P leest en rekent, en checkt allebei,
en gaat na of dit alles wel stopt na een tijd
en niet door blijft draaien, analyse ten spijt.

Nu, het zit zo: zo'n P kan niet bestaan.
Want stel, jij schrijft hem en komt er mee aan.
Dan kan ik hem gebruiken voor een logische val
zo vilein dat je brein erop breekt met een knal.

De truc die ik uithaal vertel ik je nu.
Ik maak een recept, en ik noem het ding Q.
Q neemt een programma en gaat na of dat stopt
met P kan dat immers, dus je snapt dat dit klopt.

En dit is niet alles. Als P zegt "een lus"
dan zal Q het beamen, en Q stopt daarna dus.
Maar als P zegt "dit stopt" gaat Q eindeloos voort
met dat steeds maar herhalen in een lus die ontspoort.

En dit programma, dat ik Q heb gedoopt
gaan we gebruiken, zoals je vast had gehoopt.
Als Q opstart en vraagt om zijn invoer? Zeg: Q.
Q leest dan zichzelf, en ... Wat krijgen we nu?

Als Q in een lus raakt hebben we brokken.
P zegt dan "een lus", want P zal niet jokken.
Maar dan moet Q dit beamen en stoppen.
Nu, je begrijpt, dit programma zal floppen.

Maar als Q juist mooi stopt na een tijd
dan zal P dit zien, en vort met de geit
Q moet dan een lus in, want dat was het recept.
Je ziet, ook dit kan niet, dus ik heb je genept.

Wat P ook doet, 't is niet goed of het deugt niet.
Want P zegt hoe het staat en Q doet dat teniet.
Als P eerlijk wil spreken dan liegt hij je wat voor
En zo P wil liegen: het klopt toch door en door.

Het recept dat je hier zag is bekend aan veel koks
uit de logische keuken, en het heet "paradox."
Toen je aannam dat die P kon bestaan
kreeg ik je te pakken, en toen ging je eraan.

Hoe kun je ontsnappen aan dit logisch debacle?
Geloof niet in die P, want dat was het obstakel.
Van de tegenspraak die ik liet zien kun je leren:
dat zo'n P kan bestaan kan niemand beweren.

De moraal: niemand kan jou een methode vertellen
die het op "tilt" gaan van je PC kan voorspellen.
Word niet boos en blijf vrolijk, is wat ik je brom,
en zoek zelf naar de fouten, want computers zijn dom.

Figuur 5.1: van Jan van Eijck (zeer vrij naar Scooping the Loop Snooper van Geoffrey Pullum, in Mathematics Magazine, oktober 2000) en mij bezorgd door Lucjan Despiegeleire

We zullen in deze sectie volop beroep doen op de Hypothese van Church: de algoritmes die we bespreken zullen informeel geformuleerd worden, en niet als registermachines (dat zou ondoenbaar zijn). De algoritmes zullen ook werken op de oorspronkelijke input van de inferentie, en niet op getallen die deze coderen.

We introduceren bepaalde algoritmes enkel om de berekenbaarheid van bepaalde inferentievormen aan te tonen. We zullen geen enkele poging doen om efficiënte algoritmes op te stellen.

5.3.1 Waarheid van formules berekenen in een eindige structuur

Deze vorm van inferentie neemt als input een eindige structuur $\mathfrak{A}$ en een formule A die geïnterpreteerd wordt door $\mathfrak{A}$, en berekent of $\mathfrak{A} \models A$.

> Bereken of A voldaan is in een structuur $\mathfrak{A}$ met eindig domein:
> Input: $\mathfrak{A}, A$
> Output: **t** indien $\mathfrak{A} \models A$, anders **f**.

> Bereken de relatie voorgesteld door een relatievoorschrift in structuur $\mathfrak{A}$ met eindig domein:
> Input: $\mathfrak{A}, \{(x_1, \ldots, x_n) : A\}$
> Output: $\{(x_1, \ldots, x_n) : A\}^{\mathfrak{A}}$.

Dit is het soort inferentie dat gebruikt wordt in databanken om queries op te lossen.

Beide vormen van inferentie zijn berekenbaar. Figuur 5.2 stelt een procedure voor die dit inferentieprobleem oplost. Het is geschreven in *pseudocode*. Dit is een vereenvoudigde informele notatie die gebruikt wordt om algoritmes in te specifiëren.

De procedure krijgt als input een formule Q en structuur $\mathfrak{A}$ met eindig domein en geeft **t** terug indien $\mathfrak{A} \models Q$ en **f** anders. De procedure stopt altijd na eindige tijd.

De procedure is rechtstreeks afgeleid uit Definitie 2.3.6 van waarheid in een structuur. Deze definitie gelijkt immers sterk op een procedure om de waarheid van A in structuur $\mathfrak{A}$ te berekenen. Aangezien deze definitie inductief/recursief is, is de corresponderende procedure recursief: ze roept zichzelf op.

We maken een aantal vereenvoudigingen. We veronderstellen dat de structuur $\mathfrak{A}$ een databankstructuur is, t.t.z. een Herbrandstructuur voor een eindige verzameling van constanten en predicaten. Verder, dat de inputzin Q geïnterpreteerd wordt door $\mathfrak{A}$ en dus geen functiesymbolen bevat. We gaan er ook van uit dat enkel $\wedge, \vee, \neg, \exists$ gebruikt worden. Andere connectoren kunnen gemakkelijk vertaald worden in termen van deze.

Opmerking 5.3.1. De procedure is recursief en roept zichzelf op. Hierdoor onstaat tijdens uitvoering een stapel van oproepen. Wanneer **return** uitgevoerd wordt, stopt de uitvoering van de procedure en keert de uitvoering terug naar de hoger gelegen oproep.

We gebruiken ook de volgende notatie: als A een formule is met vrije variabele x, dan is $A[d/x]$ de formule bekomen door de vrije voorkomens van x te vervangen door d.

```
function eval(Zin Q, structuur 𝔄):Boolean {
  // Input: Een zin Q, een structuur 𝔄 die Q interpreteert.
  // Output: t of f.
  if ( Q is een atoom P(d_1,...,d_n)) then {
    if ( (d_1,...,d_n) ∈ P^𝔄 ) then return t
    else return f;
    }
  else if ( Q heeft vorm d_1 = d_2 ) then {
    if (d_1 en d_2 zijn identiek) then return t
    else return f;
    }
  else if (Q heeft vorm A ∧ B) then return (eval(A) && eval(B))
  else if ( Q heeft vorm A ∨ B ) then return (eval(A) || eval(B))
  else if ( Q heeft vorm ¬A ) then return (!eval(A))
  else if ( Q heeft vorm ∃x: A ) then {
    for ( Object d ∈ D_𝔄) do { //voor elk element d in het universum doe:
      Zij B := A[d/x];
      if eval(B) then return t;
      }
    return f; // voor geen enkele d voldoet B
    }
  }
```

Figuur 5.2: Query-algorithm

Tenslotte, in deze pseudo-code worden booleaanse operatoren gebruikt: && staat voor $\wedge$, || staat voor $\vee$ en ! staat voor $\neg$. Dit zijn standaard notaties van deze connectoren in programmeertalen. Je ziet dat concepten uit de propositionele logica ook in programmeertalen gebruikt worden!

Oefening 5.3.1. *Pas deze functie toe op een klein voorbeeld in de studentendatabank.*

Opmerking 5.3.2. In LogicPalet zit een tool om uit te leggen waarom een formule waar of onwaar is in een structuur. Experimenteer met deze tool. Je zult zien dat deze tool de functie **eval** stapsgewijs uitvoert.

Deze procedure legt veel beperkingen op aan $\mathfrak{A}$ en Q. Ze is gemakkelijk uit te breiden voor andere connectoren, kwantoren en functiesymbolen. De procedure kan ook gemakkelijk uitgebreid worden om de waarde van verzamelingenexpressies $\{(x_1, \ldots, x_n) : A\}$ te berekenen in een structuur $\mathfrak{A}$.

Oefening 5.3.2. *We hebben ervoor gekozen om $\Rightarrow, \Leftrightarrow, \forall$ weg te vertalen, maar dat is helemaal niet nodig. Voeg expliciete code toe voor $\Rightarrow, \Leftrightarrow, \forall$. Breid de functie ook uit zodat verzamelingenexpressies berekend kunnen worden. Verder breid uit zodat $\mathfrak{A}$ functies kan bevatten en Q functie symbolen.*

Implementatie Het algoritme in Figuur 5.2 is naïef en is niet in staat is om queries te beantwoorden in grote databanken. De query-methodes die geïmplementeerd zijn in databanken zijn sterk geoptimaliseerd.

De belangrijkste optimalisatie is dat de databankalgoritmes nooit alle domeinelementen van de databank overlopen: ze doorlopen enkel koppels in tabellen. Bv. om de query $\exists c\colon Grade(Sam, c, AAA)$ te berekenen, volstaat het de koppels van *Grade* af te lopen op zoek naar koppels van de vorm $Grade(Sam, c, AAA)$. Bv. om de relatie $\{(x, y) : \exists c\colon Instructor(x, c) \wedge Enrolled(c, y)\}$ te berekenen volstaat het alle paren van koppels $(x, c), (c, y)$ op te zoeken in beide tabellen, en vervolgens c weg te laten. Dergelijke methodes behoren tot het domein van de *relationele algebra*. Ze leveren enorme tijdswinst op.

Een "beperking" van deze relationele algebra methodes is dat queries die betrekking hebben op het hele domein niet opgelost kunnen worden. Immers, een databank bevat geen tabel met het domein $D_{\mathfrak{A}}$. Voorbeelden zijn: $\{x : x = x\}$ en $\exists x\colon \neg Prerequ(x, x)$. Dergelijke queries en formules noemt met *domeinafhankelijk*.

Dat blijkt echter geen praktisch bezwaar. Zoals we al eerder zagen hebben formules en queries in dagdagelijks gebruik zo goed als altijd betrekking op een specifiek deeldomein dat expliciet voorgesteld wordt in een relatie. Deze kunnen uitgedrukt worden met behulp van *veilige queries*. Ze zijn *domeinonafhankelijk* en kunnen opgelost worden met relationele algebra.

Query optimalisatie Zij A, B logisch equivalente formules met vrije variabelen $x_1, \ldots, x_n$. Dan hebben de "queries" $\{(x_1, \ldots, x_n : A\}$ en $\{(x_1, \ldots, x_n) : B\}$ hetzelfde antwoord in elke databank/structuur. Het is echter best mogelijk dat het een databanksysteem véél meer tijd kost om het antwoord op één query te berekenen dan op het andere, hoewel de antwoorden uiteindelijk identiek zijn.

Voorbeeld 5.3.1. Is er een docent die ooit score AAA gegeven heeft voor een voorkennisvak dat hij nu doceert? Deze kan op verschillende manieren uitgedrukt worden. Als Q1:

$$\exists x\colon \exists y\colon \exists s\colon \exists t\colon (Instructor(x, y) \wedge Grade(s, y, AAA) \wedge Prerequ(y, t) \wedge Instructor(x, t))$$

versus als Q2:

$$\exists x\colon \exists y\colon Instructor(x, y) \wedge (\exists s\colon Grade(s, y, AAA)) \wedge (\exists t\colon Prerequ(y, t) \wedge Instructor(x, t))$$

Het is gemakkelijk te zien dat beide equivalent zijn. In het query-algoritme in Figuur 5.2 vereist Q1 vier geneste for-lussen met toekenningen aan achtereenvolgens x, y, s, t, terwijl Q2 eerst twee geneste for-lussen heeft voor x, y, en daarbinnen een geneste lus voor s en apart voor t: dus maximaal drie geneste lussen. Dat maakt de tweede query Q2 veel efficiënter dan Q1 voor het recursief query algoritme.

Om een idee te geven, veronderstel dat er 1000 domeinelementen zijn in de database. Dan zal het algoritme voor de eerste query $1000^4 = 10^{12}$, dus 1000 miljard toekenningen doen aan de binnenste variabele t. Voor de tweede formule zijn er twee binnenste variabelen s en t en aan elk zijn er slechts 1 miljard toekenningen. 500 keer minder. Een gigantisch verschil!

De meeste hedendaagses databanksystemen hebben een krachtige optimizer die queries omzet naar logisch equivalente queries die minder rekenwerk vergen, zonder dat de gebruiker daar iets van merkt. Zo'n optimizer verricht de volgende logische redeneertaak:

> Query-optimalisatie-inferentie:
> Input: $\{(x_1, \ldots, x_n) : A\}$
> Output:$\{(x_1, \ldots, x_n) : B\}$ zodat A, B logisch equivalent zijn maar de tweede query kan efficiënter opgelost worden.

Merk op dat in het voorbeeld Q1 de prenex-normaalvorm is van Q2. Vanuit het standpunt van efficiëntie is omzetten naar prenex-normaalvorm zowat het slechtste wat men kan doen. Integendeel, bij het optimaliseren van een query zal men juist proberen om kwantoren zo diep mogelijk in formules te drijven.

5.3.2 Waarheid van formules over $\mathbb{N}$: onbeslisbaarheid van $\mathbb{N}$

In de vorige sectie zagen we dat waarheid van formules in eindige structuren berekenbaar is. Dat is wat databanksystemen doen.

Zijn er ook algoritmes om de waarheid van formules na te gaan in oneindige structuren? Specifiek bekijken we de eenvoudigste oneindige structuur die er bestaat: de verzameling van de natuurlijke getallen. De formules die we zullen bekijken zijn over het vocabularium Σ_P gedefinieerd in Sectie 4.4 over de Peano axiomas. Het bestaat uit $zero, Succ, Plus, Maal$, afgekort als $0, S, +, \times$. We kunnen dus eigenschappen uitdrukken over $\mathbb{N}$ in de predikatenlogicataal van Σ_P.

Stelling 5.3.1 (Onbeslisbaarheid van $\mathbb{N}$)**.** *De verzameling van formules A over Σ_P die waar zijn in $\mathbb{N}$ is niet beslisbaar.*

M.a.w. er bestaat geen eindigend algoritme om na te gaan of een zin A van Σ_P al dan niet waar is in $\mathbb{N}$. Er bestaat zelfs geen correct algoritme dat zal eindigen indien $\mathbb{N} \models A$.

Bew. Het bewijs van deze stelling is te moeilijk om hier te geven, maar we leggen wel uit op welk idee het bewijs berust. Men bewijst eerst dat het stop-probleem "definieerbaar" is in Σ_P, daarmee bedoelt men dat er een formule $A[v_1]$ met één vrije variabele v_1 bestaat zodat voor alle $n \in \mathbb{N}$ geldt:

$$M_n \text{ bij input } n \text{ stopt na eindig aantal stappen}$$
$$\Updownarrow$$
$$\mathbb{N} \models A[S^n(0)]$$

Het bewijs van de "definieerbaarheid"van het stop-probleem is moeilijk – we geven het niet hier. Het is eenvoudig om in te zien dat dit de Stelling impliceert. Inderdaad, als er een algoritme zou bestaan om na te gaan of een willekeurige zin van Σ_P waar is in $\mathbb{N}$, dan zouden we dat algoritme kunnen gebruiken om te testen of $\mathbb{N} \models A[S^n(0)]$. Dit zou een algoritme geven om na te gaan of M_n stopt bij input n, maar zo'n algoritme bestaat niet. Q.E.D.

Deze eigenschap wordt de *onbeslisbaarheid van de natuurlijke getallen* of ook de *Stelling van Church-Turing* genoemd naar de wiskundige logici Church en Turing die dit oorspronkelijk bewezen hebben.

Opmerking 5.3.3. Er bestaat geen algemeen algoritme om de waarheid van formules over Σ_P in $\mathbb{N}$ te bepalen. Dat betekent niet dat er geen grote en belangrijke subklassen van zinnen kunnen zijn waarvoor dergelijke algoritmes wel bestaan. Bv. als we het vocabularium beperken door $\times$ weg te laten, dan bestaat er wel een algoritme. Deze eigenschap wordt de *beslisbaarheid van Presburger-rekenkunde* genoemd, naar de wiskundige Presburger die dit algoritme ontdekte.

Opmerking 5.3.4. Het is gekend dat ook $\mathbb{Z}$ en $\mathbb{Q}$ onbeslisbaar zijn. Het verbazende is dat $\mathbb{R}$ wel beslisbaar is, t.t.z. er bestaat wel een algoritme om na te gaan of een zin van Σ_P al dan niet waar is in $\mathbb{R}$. Dit werd bewezen door Tarski. Dat is merkwaardig en zelfs tegenintuïtief;

immers de natuurlijke getallen zijn een deelverzameling van de reële getallen. Maar toch is het zo. Immers, zinnen in Σ_P over de natuurlijke getallen kunnen niet geformuleerd worden als zinnen in Σ_P over $\mathbb{R}$. Want om zinnen over de natuurlijke getallen te formuleren in de context van $\mathbb{R}$, moet voor elke variabele in de zin aangegeven worden dat de variabele een natuurlijk getal is. Maar er bestaat geen formule $A[n]$ over Σ_P die in $\mathbb{R}$ uitdrukt dat n een natuurlijk getal is.

Uit de volgende propositie volgt dat de verzameling van uitspraken over Σ_P die waar zijn in de natuurlijke getallen ook niet semi-beslisbaar is. Voor elk vocabularium Σ en Σ-structuur $\mathfrak{A}$ geldt:

Propositie 5.3.1. *Als $\{A$ over $\Sigma \mid \mathfrak{A} \models A\}$ semi-beslisbaar is, dan is deze verzameling beslisbaar.*

Bew. Het bewijs is informeel maar boeiend. Op een computer worden vele programma's tegelijkertijd uitgevoerd. Dit noemt men coroutining. De processor besteedt aan elk programma in uitvoering een beetje tijd zodat tenslotte alle programma's volledig worden uitgevoerd, zij het trager. Hetzelfde is mogelijk voor registermachines. Gegeven n registermachines, dan kan men een nieuwe registermachine bouwen die al deze registermachines tegelijkertijd uitvoert: men geeft achtereenvolgens elke machine een seconde en herbegint dan opnieuw. Zo krijgt elke machine onbegrensd veel tijd. Dat dit mogelijk is kan ook formeel bewezen worden voor registermachines, maar dat zullen we niet bewijzen.

Veronderstel dus dat er een registermachine M bestaat die $\{A$ over $\Sigma \mid \mathfrak{A} \models A\}$ "semi-beslist". Bouw dan een nieuwe registermachine M' met input A die twee runs van M in co-routine uitvoert: op A en op $\neg A$. Wanneer 1 van de subroutines eindigt keert de controle terug naar M' die uit het antwoord van de subroutine het correcte antwoord berekent ($\mathbf{t}$ indien A waar is, en anders $\mathbf{f}$) en stopt. M' stopt altijd aangezien minstens 1 van de subroutines zal eindigen. Immers, A of $\neg A$ behoort tot de verzameling, en M semi-beslist deze verzameling. Dus M' berekent $\{A$ over $\Sigma \mid \mathfrak{A} \models A\}$. Q.E.D.

We kunnen zelfs een aftelbaar aantal registermachines uitvoeren met 1 registermachine. Rangschik de registermachines $M_1, M_2, \ldots$. Geef M_1 1 seconde, geef vervolgens elk van M_1, M_2 1 seconde, herbegin en geef achtereenvolgens M_1, M_2, M_3 elk 1 seconde rekentijd; vervolgens $M_1, \ldots, M_4$, enzovoort. Op die manier krijgt elke machine M_i onbegrensd veel rekentijd.

5.3.3 Berekenbaarheid van modelgeneratie en modelexpansie

Bestaan er algoritmes om de volgende vormen van inferentie te berekenen?

Modelgeneratie:
Input: theorie T
Output: een model $\mathfrak{A}$ van T, of UNSAT indien T inconsistent is.

Modelexpansie:
Input: theorie T, inputstructuur $\mathfrak{A}_i$
Output: een expansie $\mathfrak{A}$ van $\mathfrak{A}_i$ die voldoet aan T, of UNSAT indien T logisch inconsistent is.

Modellen van theorieën kunnen eindig of oneindig zijn. Het is bij voorbaat uitgesloten om oneindige modellen te berekenen met een eindigend algoritme. [1] Dus concentreren we ons op eindige modellen.

Voor elke n is het aantal structuren met n domeinelementen voor een gegeven vocabularium Σ heel groot maar eindig (modulo isomorfisme). Dus kunnen we een algoritme schrijven met een loop over $n = 1, 2, 3, \ldots$ dat bij elke n 1 of alle structuren $\mathfrak{A}$ van die grootte berekent en vervolgens het algoritme van de vorige sectie oproept om na te gaan of T voldaan is in $\mathfrak{A}$. Het algoritme zal alle eindige modellen berekenen. Als we het algoritme aanpassen zodat het stopt na het vinden van het eerste model, zal het algoritme stoppen indien de theorie minstens 1 eindig model heeft. Hieruit volgt volgende propositie.

Propositie 5.3.2. *De volgende vorm van inferentie is semi-beslisbaar:*

Eindige consistentie:
Input: theorie T
Output: **t** *als T een eindig model heeft, anders* **f**.

Op dezelfde manier valt het aan te tonen dat er een eindigend algoritme bestaat om het inferentieprobleem van modelexpansie met eindige theorie T en eindige inputstructuur $\mathfrak{A}_i$ te berekenen. Het aantal $\Sigma(T)$-structuren die $\mathfrak{A}_i$ uitbreiden voor alle vrije symbolen van T is eindig. Het algoritme genereert systematische elke structuur van $\Sigma(T)$ die $\mathfrak{A}_i$ uitbreidt, checkt of T waar is en stopt als dat het geval is. Aangezien er maar eindig veel structuren zijn zal het algoritme eindigen. Indien het geen model gevonden heeft geeft het "UNSAT" terug.

Conclusie is dat modelexpansie voor eindige structuren berekenbaar is, terwijl modelgeneratie van eindige modellen semi-berekenbaar is. Er bestaat een algoritme dat een eindig model genereert als er zo'n model bestaat, en anders in een oneindige lus gaat. Er bestaat ook een algoritme dat alle eindige modellen zal genereren, en nooit stopt.

Oefening 5.3.3. *Argumenteer waarom het aantal structuren van grootte n voor een eindig vocabularium Σ eindig is (modulo isomorfisme). Argumenteer dat het aantal Σ-structuren met domein $\{1, \ldots, n\}$ eindig is en dat elke structuur met n domeinelementen isomorf is met een structuur met domein $\{1, \ldots, n\}$.*

5.3.4 Semi-onbeslisbaarheid van deductie in predikatenlogica.

Beschouw de volgende redeneertaken:

[1] Alhoewel het soms mogelijk is om eindige representaties van oneindige modellen te berekenen. Hier gaan we niet dieper op in.

Bereken of theorie T inconsistent is.

Bereken of zin A een logisch gevolg is van theorie T.

Bereken of zin A inconsistent is met theorie T.

Deze redeneervormen noemt men **deductief redeneren** of kortweg **deductie**. Merk op: elk van deze taken kan herleid worden tot elkaar. Dus als we één taak kunnen oplossen, dan kunnen we ze allemaal oplossen.

Elk van deze drie taken heeft ook een duale taak:

Bereken of theorie T consistent is.

Bereken of zin A niet logisch waar is.

Bereken of zin A geen logisch gevolg is van theorie T.

Bereken of zin A consistent is met theorie T.

Een "ja" antwoord op een vraag betekent een "nee" antwoord op zijn duale vraag en omgekeerd.

Als we 1 van deze vragen kunnen oplossen, kunnen we ze allemaal oplossen.

Een belangrijke vraag is nu of er algoritmes bestaan die deze vragen kunnen oplossen. Het zal direct blijken dat het antwoord "half en half" is.

Stelling 5.3.2. *Het deductieve inferentieprobleem met als input een eindige T en output* **t** *indien T inconsistent is en anders* **f** *is semi-beslisbaar.*

Anders gezegd, er bestaat een practische dat bij input van zinnen $T = \{A_1, \ldots, A_n\}$ stopt na een eindig aantal stappen met output **t** indien T inconsistent is en niet stopt of antwoordt met output **f** als T consistent is.

Deze stelling geeft aan dat we deductieve problemen alvast partieel kunnen oplossen.

Bew. We geven enkel een schets van het bewijs. Een formeel bewijs zou erg lastig zijn (en weinig bijbrengen).

Een KE-redenering voor een theorie is een eindige boom van formules. Noemen we de grootte van een KE-redenering het totaal aantal symbolen dat erin voorkomt. Door gebruik te maken van de gepaste volledige strategie is het niet moeilijk ervoor te zorgen dat voor elke n, het aantal KE-redeneringen van grootte n eindig is.

Dit suggereert het volgende algoritme: bouw een loop die voor $n = 1, 2, 3, \ldots$ telkens alle KE-redeneringen voor T van grootte n construeert. Vervolgens wordt voor elke KE-redenering getest

of alle takken gesloten zijn. Indien dit het geval is stopt de uitvoering en geeft het **t** als antwoord. In het andere geval blijft het doorrekenen.

Het is duidelijk genoeg dat dit algoritme kan geïmplementeerd worden, en door de hypothese van Church kan het geïmplementeerd worden met een registermachine. Indien T inconsistent is zal dit algoritme eindigen. Immers door de volledigheidsstelling van Gödel weten we dat er dan een KE-bewijs van inconsistentie bestaat, dit heeft eindige grootte n en vroeg of laat wordt dit bewijs ontdekt door het algoritme. Als T consistent is dan zal het algoritme nooit eindigen.

Dit algoritme voldoet aan de voorwaarden van de stelling. Q.E.D.

Deze eigenschap wordt ook kortweg de *semi-beslisbaarheid van predikatenlogica* genoemd. Merk op dat het betrekking heeft op deductieve inferentie.

Het algoritme in het bewijs is buitengewoon naïef en zou in de praktijk nooit werken. Maar hier zijn we nog niet geïnteresseerd in zulke praktische zaken zoals efficiëntie.

We hebben een partiële oplossing (want we kunnen niet garanderen dat het algoritme stopt. Kan er een algoritme gevonden worden dat ook stopt als T consistent is? Het antwoord is negatief.

Stelling 5.3.3 (Stelling van Church (1936))**.** *Het deductieve inferentieprobleem met als input een eindige T in predikatenlogica en output* **t** *indien T inconsistent is en anders* **f** *is niet beslisbaar.*

Zonder bewijs.

Anders gezegd, er bestaat geen eindigend practische om deductieve inferentietaken op te lossen.

Men vat de stelling van Church ook wel eens samen door te zeggen : "*Predikatenlogica is onbeslisbaar*". Daarmee bedoelt men dus dat deductieve inferentie in predikatenlogica onbeslisbaar is. De logicus Church heeft dit resultaat voor het eerst bewezen.

Let wel, zoals we net zagen hebben hij en Allan Turing ook bewezen dat er geen computerprogramma kan bestaan dat voor elke zin kan beslissen of deze waar is in de natuurlijke getallen. Deze stelling heet: de onbeslisbaarheid van de natuurlijke getallen. Verwar deze twee stellingen niet!

De twee andere deductieve inferentietaken zijn: de inferentie met input T en zin A die bepaalt of A is een logisch gevolg is van T; en de inferentie met input A die bepaalt of A logisch waar is. Deze drie taken zijn herleidbaar tot elkaar. Bijgevolg zijn ook ze alle drie niet beslisbaar maar wel semi-beslisbaar. Voor alle drie bestaat een correct algoritme dat zal eindigen in geval van een positief antwoord.

De stelling betekent ook dat voor de drie duale redeneertaken (is T consistent, is A geen logisch gevolg van T, is A niet logisch waar?) geen programma bestaat dat voor elke input in eindige tijd het antwoord vindt indien dat antwoord positief is. Daarentegen bestaat er wel een dergelijk programma dat altijd zal eindigen indien het antwoord negatief is.

Samengevat stellen we het volgende vast.

Stelling 5.3.4. *Deductieve inferentie in predikatenlogica is onbeslisbaar maar wel semi-beslisbaar.*

5.4 De Onvolledigheidsstelling van Gödel

Niet voor het examen

Deze sectie gaat over een stelling die van enorme betekenis is geweest in de filosofie van de wiskunde en de studie naar de *grondslagen van de wiskunde*.

De structuur $\mathbb{N}$ is één der meest fundamentele objecten in de wiskunde. De studie van de eigenschappen van de natuurlijke getallen (getaltheorie, getallenleer) is moeilijk en veel problemen zijn nog steeds niet opgelost. Zo vermoedt men bijvoorbeeld dat elk even natuurlijk getal (groter dan twee) kan geschreven worden als de som van twee priemgetallen. Dit is het vermoeden van Goldbach (1742) dat ondertussen al bijna 300 jaar oud is en nog steeds onbewezen. Een ander voorbeeld is de stelling van Fermat, waarvoor het meer dan 300 jaar heeft geduurd vooralleer ze bewezen kon worden.

Vele eigenschappen van de natuurlijke getallen zijn verre van evident. Bijvoorbeeld: een priemgetal p kan geschreven worden als de som van twee kwadraten van natuurlijke getallen asa p is een veelvoud van 4 plus 1 of $p = 2$. Zulke eigenschappen die niet evident zijn wil men kunnen bewijzen.

Om eigenschappen van de natuurlijke getallen te bewijzen gaat men als volgt te werk. Men stelt verzamelingen van axioma's op die "evident waar" zijn in $\mathbb{N}$. Men noemt zo'n verzameling een *axiomasysteem*. Men probeert daaruit andere niet evidente eigenschappen te bewijzen.

Zo'n axiomasysteem kan oneindig zijn. Maar dan eisen we dat de verzameling van axiomas *berekenbaar* is. T.t.z. er bestaat een algoritme om na te gaan of een uitspraak behoort tot het axiomasysteem T. Anders zou ons systeem T onbruikbaar zijn!

De Peano rekenkunde PA vormen duidelijk zo een axiomasysteem. Het is oneindig door het inductieschema:

$$A[0] \wedge (\forall x\colon A[x] \Rightarrow A[S(x)]) \Rightarrow \forall x\colon A[x]$$

Het is evenwel niet moeilijk in te zien dat er een eindigend algoritme bestaat om na te gaan of een zin al dan niet een instantiatie is van dit schema. Uit de hypothese van Church volgt dan dat dit kan gedaan worden door middel van een registermachine. (Het gebruik van de hypothese van Church kan zoals altijd vermeden worden en vervangen worden door een saaie en omslachtige oefening in het programmeren van registermachines.)

Vóór 1931 dacht men dat elke ware uitspraak over $\mathbb{N}$ (geformuleerd als een zin in Σ_P) bewijsbaar zou zijn uit PA. Het was dan ook een dramatische verrassing toen Kurt Gödel in 1931 bewees dat dit niet waar is. Op zich hoeft dat niet erg te zijn. Misschien bestaat er een ander vollediger axiomasysteem T zodat elke ware uitspraak over $\mathbb{N}$ bewijsbaar is uit dat systeem T? Helaas, dit is niet mogelijk. Gödel bewees dat zo een axiomasysteem T niet kan bestaan! Dit is de beroemde *Onvolledigheidsstelling van Gödel*.

Stelling 5.4.1 (De onvolledigheidsstelling van Gödel (1931))**.** *Zij T een verzameling van zinnen van Σ_P in de predikatenlogica. Veronderstel dat*

1. *$\mathbb{N}$ is een model voor T (d.w.z. elke zin van T is waar in $\mathbb{N}$), en*
2. *T is berekenbaar (d.w.z. er bestaat een algoritme om na te gaan of een zin al dan niet tot T behoort).*

Dan bestaat er een zin A in Σ_P zodat A waar is in $\mathbb{N}$ maar zodat A toch geen logisch gevolg is van T.

Opmerking 5.4.1. Ter herinnering, A is een logisch gevolg van T indien A waar is in alle modellen van T. A is *bewijsbaar* uit T indien er een KE-bewijs van A uit T bestaat (of een KR-bewijs van inconsistentie van $T \cup \{\neg A\}$. Omwille van de correctheid en volledigheid van KE-bewijzen (zie Hoofstuk 3) geldt dat beide concepten samenvallen. Dus is de zin A uit de stelling niet bewijsbaar uit T.

Opmerking 5.4.2. Indien we voor T de verzameling PA der Peano axioma's nemen, dan bekomen we dat er een zin in Σ_P bestaat die waar is in $\mathbb{N}$ maar niet bewijsbaar uit/geen logisch gevolg van PA. (Inderdaad PA is berekenbaar.)

Opmerking 5.4.3. Indien T een ander axiomasysteem voor $\mathbb{N}$ is, zoals besproken in het begin van deze sectie, dan zijn de voorwaarden (1) en (2) in Stelling 5.4.1 voldaan.

Opmerking 5.4.4. Zonder voorwaarde (2) zou Stelling 5.4.1 niet waar zijn, want dan zouden we voor T de verzameling van alle ware zinnen (in $\mathbb{N}$) kunnen nemen. Deze verzameling is echter niet berekenbaar, want $\mathbb{N}$ is onbeslisbaar!

Bew. Bewijs van de onvolledigheidsstelling.

Uit het ongerijmde. Veronderstel dat voor elke zin A geldt dat

$$\text{als } A \text{ waar is in } \mathbb{N}, \text{ dan is } A \text{ een logisch gevolg van } T \tag{5.1}$$

Zij A een willekeurige zin van Σ_P. Dan is A ofwel waar, ofwel onwaar in $\mathbb{N}$. Als A waar is, dan hebben we volgens (5.1) dat A een logisch gevolg is van T. Als A onwaar is, dan hebben we dat $\neg A$ waar is, en dus volgens (5.1) dat $\neg A$ een logisch gevolg is van T. We besluiten dus

$$\text{voor elke zin } A \text{ geldt dat ofwel } A, \text{ ofwel } \neg A, \text{ een logisch gevolg is van } T \tag{5.2}$$

Indien T eindig is, dan weten we op grond van de semi-beslisbaarheid van de predikatenlogica (zie Hoofdstuk 3) dat er een computerprogramma P_1 bestaat dat bij input A stopt na een eindig aantal stappen met output "ja", als en slechts als A een logisch gevolg is van T. Deze bewering blijft ook geldig wanneer T oneindig is, op voorwaarde dat T berekenbaar is (dit moet je zelf kunnen inzien!).
Omwille van dezelfde reden bestaat er ook een computerprogramma P_2 dat bij input A stopt na een eindig aantal stappen met output "nee", als en slechts als $\neg A$ een logisch gevolg is van T.
Bij input A laten we tijdens de dag het programma P_1 werken, en tijdens de nacht het programma P_2. Omwille van (5.2) komt er dan steeds na een eindige tijd een antwoord "ja" of een antwoord "nee", en op dat ogenblik breken we de werking van beide programma's af. Het gegeven antwoord is "ja" indien A waar is in $\mathbb{N}$ (want dan is, volgens (5.1), de zin A een logisch gevolg van T), en het gegeven antwoord is "nee" in het andere geval. Maar dit geeft ons dus (steunende op de onware veronderstelling (5.1) van ons bewijs uit het ongerijmde) een algoritme (dat steeds na een eindig aantal stappen stopt) om na te gaan of een zin A al dan niet waar is in $\mathbb{N}$. Dit is in strijd met de onbeslisbaarheid van $\mathbb{N}$. Deze contradictie beëindigt het bewijs van de onvolledigheidsstelling. Q.E.D.

De *Onvolledigheidsstelling van Gödel* heeft drastisch de "filosofische" opvattingen over de wiskunde veranderd. Voordien geloofde men een axiomasysteem te kunnen opstellen voor de ganse

wiskunde, zodat elke ware uitspraak in de wiskunde uit dit axiomasysteem zou kunnen bewezen worden. Dit was de ambitie van de *axiomatische methode*. Eén van de grote voorvechters in de jaren 20 van vorige eeuw daarvan was de Duitse wiskundige Hilbert. Maar dat bleek dus onmogelijk.

Nochtans blijft de axiomatische methode nuttig en onontbeerlijk. Uit de praktijk blijkt bijvoorbeeld dat *bijna* alle eigenschappen van de natuurlijke getallen, die men ooit heeft kunnen bewijzen, reeds bewijsbaar zijn uit PA.

Opmerking 5.4.5. We hebben hier de onvolledigheidsstelling van Gödel ($\pm$) bewezen uit de onbeslisbaarheid van $\mathbb{N}$. Historisch gezien is dit heel anders gelopen. Gödel heeft de onvolledigheidsstelling bewezen vooralleer het concept berekenbaarheid formeel gedefinieerd werd, en dus chronologisch voor de Church-Turing stelling van onbeslisbaarheid van de natuurlijke getallen. De notie van algoritme was al bekend bij de arabieren in de middeleeuwen, zij het als een informele notie. Gödel sprak over theorieën T die berekenbaar waren met een algoritme in termen van die informele notie van algoritme.

Het is in feite de onvolledigheidsstelling geweest die de aanzet was om berekenbaarheid formeel te definiëren. Mensen als Church en Turing en ongetwijfeld Gödel zelf zagen al spoedig in dat de onbeslisbaarheid van de natuurlijke getallen onmiddelijk volgde uit de de onvolledigheidsstelling. [2] Het onderzoek van Gödel vormde dus de onmiddelijke aanleiding om het concept van berekenbaarheid en de onbeslisbaarheid van de natuurlijke getallen uit te werken.

5.5 Conclusie

In dit hoofdstuk hebben we een formeel model voor algoritmes gezien: de registermachines. We hebben aangetoond dat er problemen zijn die niet berekenbaar zijn, met als beroemd voorbeeld het haltingsprobleem.

Vervolgens werd aandacht besteed aan een aantal verschillende redeneertaken. We zagen dat berekenen van waarheid in eindige structuren efficiënt geïmplementeerd kan worden, maar dat berekenen van waarheid in de natuurlijke getallen onberekenbaar is. Modelexpansie voor eindige inputstructuren is berekenbaar, maar berekenen van consistentie en berekenen van willekeurige modellen is niet berekenbaar.

Tenslotte hebben we gekeken naar deductieve inferentie.

- bepalen of een zin logisch waar is
- bepalen of een zin inconsistent is
- bepalen of een zin een logisch gevolg is van een theorie.

Elk van deze taken heeft een duale taak (niet logisch waar, consistent, geen logisch gevolg).

In propositielogica blijken al deze taken oplosbaar (propositielogica is beslisbaar!). In predikatenlogica echter blijken deze taken maar half oplosbaar. Er bestaan wel methodes die in

[2] Het bewijs: neem de verzameling T van alle ware uitspraken in de natuurlijke getallen. Deze theorie heeft een triviaal bewijs voor elke ware uitspraak A in de natuurlijk getallen, aangezien $A \in T$. Uit de correctheidseigenschap volgt dat elke ware uitspraak in $\mathbb{N}$ een logisch gevolg is van T. Uit de onvolledigheidsstelling volgt dat T niet berekenbaar kan zijn. Dus is waarheid in de natuurlijke getallen niet beslisbaar.

eindige tijd het antwoord "ja" op deze vragen kan produceren, maar geen enkele correcte methode kan in eindige tijd antwoorden met "ja" of "nee". Dit noemt men de *onbeslisbaarheid* en *semi-beslisbaarheid* van predikatenlogica.

Let wel, in feite is de uitspraak "*predikatenlogica is onbeslisbaar*" misleidend, omdat deze zin niet expliciet maakt over welke redeneertaak het gaat. Vergeet niet dat andere redeneertaken (bv. berekenen van waarheid of queries in een structuur, berekenen van modellen met een gegeven eindig domein, enz.) ongetwijfeld veel meer dagdagelijkse toepassingen hebben in de informatica dan het bewijzen of een formule logisch waar is of inconsistent of een logisch gevolg. En er bestaan wel eindigende en zelfs zeer efficiënte programma's om die taken te verrichten voor predikatenlogica, of voor talen die daarvan afgeleid zijn. Databanksystemen zijn voorbeelden daarvan.

Historisch gezien is er veel te doen geweest rond de onbeslisbaarheid van predikatenlogica. Er zijn nogal wat mensen waarvoor predikatenlogica bijna *synoniem* is voor de studie van deductief redeneren[3]. Daar is een historische reden voor: herinner u uit Hoofdstuk 1 dat predikatenlogica ontstond uit de studie van het wiskundig, deductief redeneren. Aangezien gebruikers van softwaresystemen nood hebben aan eindigende programma's die het antwoord op onze vragen vinden, lijkt predikatenlogica voor mensen met deze visie onbruikbaar voor praktisch gebruik in de informatica! Maar dit is een te enge visie op predikatenlogica, en de conclusie strookt gewoon niet met de feiten: de vele toepassingen van andere vormen van redeneren, en de efficiëntie waarmee deze geïmplementeerd kunnen worden.

[3]Predikatenlogica wordt door sommigen ook wel *deductive logic* genoemd

Bijlage A

Appendix

A.1 Meer over SQL en logica

Het verband met logica. Het verband tussen SQL-databanken en logica is niet eenduidig. T.t.z., er zijn meerdere mogelijkheden om een SQL-databank te zien als een logische databank., Dit is afhankelijk van hoe we het logische vocabularium Σ afleiden uit het database-schema:

- De "klassieke" manier: we kiezen een n-voudig predikaatsymbool per tabel met n kolommen.
- De "object georiënteerde" manier: we kiezen de tabelnaam Tab als een unair type-predikaat en de kolomnamen $TabVeld_i$ als 2-voudige relaties.
- Allerlei tussenvormen zijn mogelijk.

De keuze van het logisch vocabularium Σ heeft een grote impact op hoe SQL-queries vertaald worden in logica.

De beste manier om het vocabularium op te stellen is om de intuitie te volgen en de meest eenvoudige en natuurlijke symbolen te kiezen.

In databanken is het gebruikelijk om tabellen met zeer veel kolommen te introduceren. In zo'n situatie werkt de "klassieke" vertaling slecht. Stel u een tabel voor om de werknemers in een bedrijf voor te stellen: *Werknemer(id,voornaam, naam, straat, straatnr, gemeente, zip, geboortedatum, job, loon, anciënniteit).* Stel dat we de klassieke omzetting doen, wat resulteert in een predicaatsymbool met 11 argumenten. Neem de query naar de werknemers die minstens 100.000 euro per jaar verdienen. In SQL wordt dit

$$\begin{array}{l} SELECT\ id \\ FROM\ Werknemer \\ WHERE\ Werknemer.loon \geq 100.000; \end{array}$$

Gebruikmakend van hett 11-voudig predikaatsymbool $Werknemer$ wordt de query:

$$\begin{array}{l} \{id: \ \exists vn: \exists n: \exists st: \exists stnr: \exists g: \exists z: \exists geb: \exists j: \exists l: \exists a: \\ \qquad (Werknemer(id, vn, n, st, stnr, g, z, geb, j, l, a) \wedge l > 100.000)\} \end{array}$$

Men zou voor minder een indigestie krijgen.

In deze situatie is het duidelijk dat een rij een werknemer voorstelt, en de kolommen verschillende attributen ervan. Hier is "object georienteerde" omzetting superieur. In deze formulering krijgen we bv.:

$$\{id : \exists w\colon [WerknemerId(w, id) \wedge \exists l\colon (WerknemerLoon(w, l) \wedge l \geq 100.000)]\}$$

Deze kan nog verder vereenvoudigd worden. Veel SQL-tabellen bevatten een zogenaamde *primary key*. Dit is een kolom die unieke identifiers bevat voor elke rij van de tabel. Zo'n primary keys krijgen vaak de naam "id" of zoiets. In zo'n geval kunnen we de de primary keys gebruiken als identifier van de rij-objecten en hoeven we geen nieuwe type van objecten te introduceren voor de rijen.

Dit is hier van toepassing. Veronderstel dat de eerste kolom *id* van de Werknemer tabel een primary key is. De tabel kan dan gemodelleerd worden met 1 (redelijk overbodig) type-predikaatsymbool $Werknemer/1$ die bestaat uit de primary keys en 10 2-voudige predicaten. In deze voorstelling wordt de query:

$$\{id : \exists l\colon (WerknemerLoon(id, l) \wedge l \geq 100.000)]\}$$

Conclusie: het kiezen van een vocabularium. Zoals we gezien hebben heeft de keuze van het vocabularium een enorme impact op de vorm en elegantie van de logische zinnen en queries.

De lessen die we hier geleerd hebben over het kiezen van een vocabularium zijn niet enkel van toepassing in de context van databanken maar in modelleringsproblemen in het algemeen. Kiezen we één groot predicaatsymbool met meerdere argumenten of introduceren we nieuwe types en splitsen we op in kleinere predicaten? Welke types van objecten beschouwen we? Men moet hierin een gulden middenweg zoeken met als doel dat logische zinnen zo compact en elegant mogelijk zijn. Vaak komt dit erop neer om de meest eenvoudigste en elementaire concepten door symbolen voor te stellen.

We hebben gezien dat te grote predikaten beter opgesplitst worden, maar teveel opsplitsing (zoals bij $Grade/3$) is evenmin goed. Voor de studentdatabank geldt dat de oorspronkelijke modellering van de databank met $PassingGrade/1$, $Enrolled/2$, $Prerequ/2$, $Instructor/2$ wellicht de beste keuze was.