# Propositielogica: deel I (syntaxis en semantiek)

Een propositie is een uitspraak die, gegeven een situatie, waar of onwaar kan zijn bv. 'Het regent'

Beweringen / proposities zijn atomair. We gaan ze niet verder analyseren en stellen ze voor door symbolen, nl. letters

We gebruiken ook symbolen voor logische connectieven:

* niet: ¬
* en: ∧
* of: ∨
* als-dan: ⟶
* dan-en-slechts-dan-als: ⟷

Bij een (formele) taal zijn 3 aspecten belangrijk:

* Het alfabet: welke symbolen men mag gebruiken
* De syntaxis (grammatica): geheel van regels die aangeeft op welke manier uitdrukkingen in de taal gevormd mogen worden
* De semantiek: de betekenis van syntactisch correcte uitdrukkingen in een taal

**Alfabet**

|  |  |
| --- | --- |
| Definitie: | Het alfabet van de propositielogica bestaat uit:   * + een verzameling propositieletters   + de logische symbolen: ¬, ∧, ∨, → en ↔   + de hulpsymbolen: ) en ( |

**Syntaxis - formule**

De symbolen uit het alfabet zijn te combineren via regels tot uitdrukkingen, formules genoemd.

|  |  |
| --- | --- |
| Definitie: | De formules in de propositielogica zijn als volgt gedefinieerd: |
|  | * 1. Elke propositieletter is een formule   2. Als ϕ en ψ formules zijn dan zijn ¬ϕ, (ϕ ∧ ψ), (ϕ ∨ ψ), (ϕ → ψ) en (ϕ ↔ ψ) ook formules   3. Niets anders is een formule |

* Kleine Griekse letters worden gebruikt om abstracte (willekeurige) formules weer te geven = formulevariabelen

Terminologie:

|  |  |  |
| --- | --- | --- |
| ¬ ϕ  (ϕ ∧ ψ)  (ϕ ∨ ψ)  (ϕ → ψ)  (ϕ ↔ ψ) | niet phi  phi en psi  phi of psi  als phi dan psi  phi dan en slechts dan als psi  (desda) | negatie conjunctie disjunctie implicatie equivalentie |

**Inductieve definitie**

* De manier waarop de formules zijn gedefinieerd heet inductief
* Een inductieve definitie bestaat uit
  + Eén of meerdere basisstappen waarin bepaalde dingen meteen tot objecten van de gewenste soort worden verklaard bv. elke propositieletter is een formule
  + Eén of meerdere opbouwstappen die de constructieprincipes geven om objecten te maken bv. als ϕ en ψ formules zijn dan zijn ¬ϕ, (ϕ ∧ ψ), (ϕ ∨ ψ), (ϕ → ψ) en (ϕ ↔ ψ) ook formules
  + Een afsluitende stap die bepaalt dat alles wat niet in eindig veel stappen met behulp van 1 en 2 gevormd kan worden, geen toegestaan object is bv. niets anders is een formule

Formuleschema:

Een vorm zoals (ϕ ↔ ¬ψ) is een abstracte vorm van een formule bv. ((p ∧ q) ↔ ¬(r ↔ ¬(s → q))

Een concrete formule ontstaat als voor ϕ en ψ worden ingevuld bv. Als phi wordt vervangen door p en psi door q. Dit heet een instantie van het formuleschema.

**Substitutie**

Voorbeeld: ((p ∧ q) → p) |p wordt vervangen door s| ((s ∧ q) → s)

Laat ϕ een formule zijn waar (mogelijk) een propositieletter p in voorkomt. Door elk voorkomen van p in ϕ te vervangen door een formule ψ, ontstaat een nieuwe formule.

|  |  |
| --- | --- |
| Definitie | Substitutie  1.  [ψ/p]ϕ = ψ als ϕ = p  [ψ/p]ϕ = ϕ als ϕ een propositieletter is verschillend van p    2.  [ψ/p]¬ϕ = ¬[ψ/p]ϕ  [ψ/p](ϕ ∧ χ) = ([ψ/p]ϕ ∧ [ψ/p]χ)  [ψ/p](ϕ ∨ χ) = ([ψ/p]ϕ ∨ [ψ/p]χ)  [ψ/p](ϕ → χ) = ([ψ/p]ϕ → [ψ/p]χ)  [ψ/p](ϕ ↔ χ) = ([ψ/p]ϕ ↔ [ψ/p]χ) |

Constructieboom

* De opbouw van een formule kan worden weergegeven door een constructieboom

S 
t 
( p) 

* Haakjes zijn belangrijk; ze geven het bereik van een connectief aan
* Ze leggen de constructie eenduidig vast
* Hoeveel constructiebomen bestaan er voor een formule?

In het algemeen is er precies één constructieboom per formule

Dit is niet steeds zo in de natuurlijke taal bv. "Als de baby niet huilt en trappelt, dan is hij gelukkig"

¬(huilen ∧ trappelen) niet huilt en niet trappelt (¬huilen ∧ trappelen) niet huilen en wel trappelen

**Semantiek**

De betekenis van een formule is zijn waarheidswaarde, nl. waar of onwaar

De waarheidswaarde van een formule wordt gegeven door de waarheidswaarden 'waar' (1) en 'onwaar' (0) van de onderdelen van de formule

Er is een waarheidstabel voor elke eonnectief: 
1 
o 
q) 
I 
1 
o 
1 
I 
o 
q) 
1 
1 
o 
1 
o 
I 
o 
I 
o 

Waarheidstabel voor een samengestelde formule: tabel met waarheidswaarden voor alle waarderingen (combinaties) van de voorkomende propositieletters en de deelformules

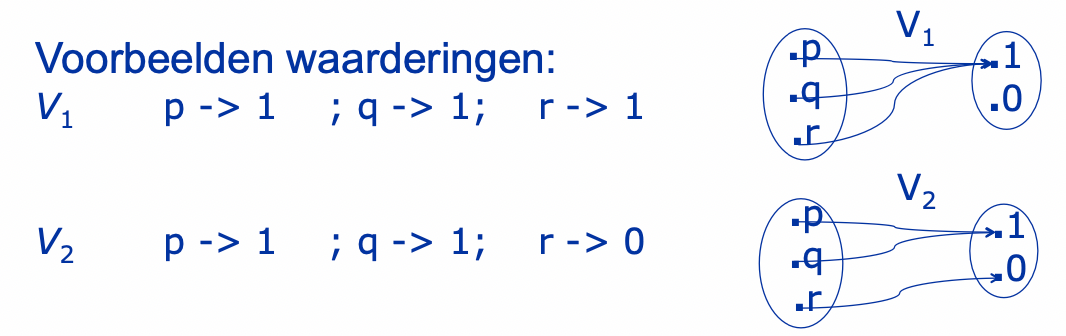
h 
1 
1 
1 
1 
o 
o 
o 
o 
s 
1 
1 
o 
o 
1 
1 
o 
o 
u 
1 
o 
1 
o 
h As 

Complexiteit waarheidstabellen

* Wanneer n verschillende propositieletters in een formule voorkomen dan heeft de waarheidstabel van de formule 2n rijen

**Waardering**

|  |  |
| --- | --- |
| Definitie | Een waardering is een functie van alle propositieletters naar de waarheidswaarden 'waar' en 'onwaar' |



Via een gegeven waardering kan men de waarheidswaarde van een willekeurige formule ϕ bepalen

**Model**

|  |  |
| --- | --- |
| Definitie | Een waardering V heet een model van een formule ϕ als geldt dat V(ϕ) = 1 |

* de verzameling van alle modellen van ϕ noteren we MOD(ϕ) = {V | V(ϕ) = 1}

**Model van een formuleverzameling**

|  |  |
| --- | --- |
| Definitie | Een waardering V heet een model van een formuleverzameling ∑ als V een model is van elke formule ϕ ∈ ∑ |

* de verzameling van modellen van ∑ wordt genoteerd als MOD(∑)

Merk op dat MOD(∑ ∪ {ϕ}) ⊆ MOD(∑)

* Hoe meer formules, hoe minder modellen, hoe meer informatieve inhoud
* Eén model betekent volledige informatie

**Modeleliminatie**

Techniek: modeleliminatie

* Als Σ1 ⊆ Σ2 dan geldt MOD(Σ2) ⊆ MOD(Σ1)
* bepaal eerst alle modellen van ϕ, vervolgens kijken we welke ook modellen zijn van ψ, en tenslotte welke ook modellen zijn voor χ

Voorbeeld:

Over het weer in Londen weten we het volgende:

Het waait of het regent.

Als het waait en regent, dan is het koud.

Als het regent, dan is het niet koud.

Als het niet waait, dan is het koud

1 
1 
1 
1 
O 
O 
o 
O 
Conclusie: 
r 
1 
1 
O 
O 
1 
1 
O 
O 
k (w vr) ((w A r) k) (r 
1 
O 
o 
1 
O 
Twee modellen blijven over: 1/3 en 1/4. 
Voor beide geldt: het waait en het regent niet. 

**Tautologie en contradictie**

|  |  |
| --- | --- |
| Definitie | Een formule ϕ is een tautologie als elke waardering een model is van ϕ, m.a.w. ∀V: V(ϕ) = 1 |

M.a.w. (intuïtief) een tautologie is een formule die altijd waar is bv. (p ⟶ (q ⟶ p)), alle instanties van (ϕ v ¬ϕ) en

¬(ϕ v ¬ϕ)

De tegenhanger van een tautologie heet een contradictie. Dus een contradictie is een formule die altijd onwaar is.

**Logisch equivalent**

|  |  |
| --- | --- |
| Definitie | Twee formules ϕ en ψ heten logisch equivalent als de formule (ϕ ⟷ ψ) een tautologie is |

Voorbeelden:

* ϕ en ¬¬ϕ
* (ϕ ∧ (ψ ∧ χ)) en ((ϕ ∧ ψ) ∧ χ) (associativiteit)
* de wetten van De Morgan:

(¬(ϕ ∧ ψ) ↔ (¬ϕ ∨ ¬ψ)) is een tautologie

(¬(ϕ ∨ ψ) ↔ (¬ϕ ∧ ¬ψ)) is een tautologie

* de principes van distributiviteit:

((ϕ ∧ (ψ ∨ χ)) ↔ ((ϕ ∧ ψ) ∨ (ϕ ∧ χ))) is een tautologie

((ϕ ∨ (ψ ∧ χ)) ↔ ((ϕ ∨ ψ) ∧ (ϕ ∨ χ))) is een tautologie

Logische equivalenties rechtvaardigen een lossere notatie:

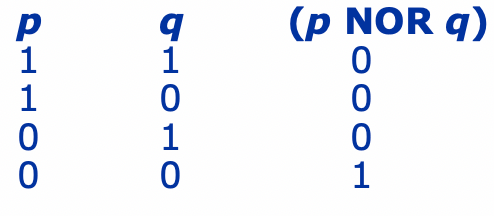
* Op grond van de logische equivalentie van ((ϕ ∧ (ψ ∧ χ)) en ((ϕ ∧ ψ) ∧ χ) laten we toe om haakjes weg te laten, dus (ϕ ∧ ψ ∧ χ) i.p.v. ((ϕ ∧ (ψ ∧ χ)) of ((ϕ ∧ ψ) ∧ χ)
* Dit noemt men associativiteit van ∧
* Ook '∨' is associatief
* Ook buitenste haakjes mogen weggelaten worden indien geen verwarring mogelijk ϕ ∧ ψ ∧ χ

**Functioneel volledig**

|  |  |
| --- | --- |
| Definitie | Een verzameling van connectieven C heet functioneel volledig als elk formule ϕ logisch equivalent is met een formule ψ die enkel connectieven uit C bevat |

* De verzameling connectieven {¬, ∧, ∨} is functioneel volledig (bewijs later)
* {¬, ∨} is dan ook functioneel volledig op grond van de tautologie ((ϕ ∧ ψ) ↔ ¬(¬ϕ ∨ ¬ψ)). Hierdoor kunnen alle voorkomens van ∧ vervangen worden door ¬ en ∨.

{NOR} is functioneel volledig



**Disjunctieve normaalvorm (syntactische opsomming)**

|  |  |
| --- | --- |
| Definitie | Een formule is in disjunctieve normaalvorm wanneer deze de syntactische vorm heeft van een disjunctie van conjuncties bestaande uit atomen of negaties van atomen: (ϕ1 ∧ … ∧ ϕn1) ∨ … ∨ (χ1 ∧ … ∧ χnk) waarbij ϕ1, …, χnk  atomen of negaties van atomen zijn |

Algemeen geldt dat voor iedere formule ϕ een logische equivalente formule ϕ\* bestaat die disjunctieve normaalvorm is

Hoe gaan we nu redeneren: essentieel 2 manieren:

Via de modellen: semantisch redeneren (geldig gevolg): kijken naar waarheidswaarden

Via afleidingsregels: syntactisch redeneren (natuurlijke deductie): kijken naar afleidingsregels en niet waarheidswaarden

geldig afleidbaar is op de semantische manier ⟷ geldig afleidbaar op syntactische manier

# Propositielogica: deel II (geldig gevolg)

**Geldig gevolg**

Uit (p ∧ q) kunnen we p concluderen (waar)

A number and a line with a red line

Description automatically generated with medium confidence

Het principe voor geldig gevolg: als het uitgangspunt waar is, dan is ook de conclusie waar.

|  |  |
| --- | --- |
| Definitie | Een formule ψ is een geldig gevolg van een verzameling formules ∑ als elk model van ∑ ook model is van ψ |

Notatie: ∑ ⊨ ψ of ook nog ∑ / ψ

* psi is een geldig gevolg van sigma
* Als ∑ = {φ1, …, φn} dan schrijven we φ1, …, φn ⊨ ψ

Voorbeeld

* p, p ⟶ q ⊨ q   
  Als ik een model heb die beide p en p ⟶ q waarmaakt, d.w.z. dat p waar moet zijn en de implicatie waar moet zijn, de enige manier om de implicatie dan waar te maken is dat q ook waar is

∑ ⊭ ψ

* d.w.z. ψ is geen geldig gevolg van ∑
* er bestaat dus een model van ∑ dat geen model is van ψ.  
  Dit model noemt men een tegenvoorbeeld van de gevolgtrekking ∑ ⊨ ψ

Voorbeeld: q, p ⟶ q ⊭ p

q waar en p onwaar dan (p ⟶ q) waar, maar p is onwaar

**Geldig gevolg versus implicatie**

Opgelet "geldig gevolg" en "waarheid" zijn verschillende begrippen!

Implicatie (⟶) en logisch gevolg (⊨) zijn twee verschillende concepten!

⟶ maakt deel uit van de syntaxis van de propositielogica; ⊨ niet

Als φ ⊨ ψ, dan is (φ ⟶ ψ) waar, maar niet omgekeerd: (φ ⟶ ψ) betekent niet noodzakelijk φ ⊨ ψ

* want (φ ⟶ ψ) kan ook onwaar zijn (als φ waar en ψ onwaar) en dan is er geen geldig gevolg meer
* Als (ϕ1 ∧ … ∧ ϕn) ⟶ ψ een tautologie is, dan hebben we ook ϕ1 ∧ … ∧ ϕn ⊨ ψ en omgekeerd

Bekende geldige gevolgen

Ex falso: φ, ¬φ ⊨ ψ phi moet waar zijn, maar niet phi ook dus 0 modellen

Uit een contradictie volgt alles

Contrapositie: φ ⟶ ψ ⊨ ¬ψ ⟶ ¬φ

Hypothetisch syllogisme: φ ⟶ ψ, ψ ⟶ χ ⊨ φ ⟶ χ

Disjunctief syllogisme: φ ⟶ ψ, ¬φ ⟶ ψ ⊨ ψ

Speciaal geval indien ∑ = ∅

⊨ ψ betekent dat elke waardering een model is voor ψ; m.a.w. ψ is een tautologie

**Hoe weten we dat een gevolgtrekking geldig is?**

Eerste mogelijkheid: stel een waarheidstabel op

Voorbeeld: —q, p * q / 
Model voor p -+ q } en voor —p 

* Maar de grootte van een waarheidstabel is exponentieel in het aantal proposities
* Het is efficiënter om naar een tegenvoorbeeld te zoeken d.m.v. techniek: semantische tableaus

Een waardering die alle formules aan de linkerkant waarmaakt en alle formules aan de rechterkant onwaar maakt, dan hebben we een tegenvoorbeeld gevonden en is er geen geldig gevolg. Als je het tegenvoorbeeld niet vindt, mogen we veilig besluiten dat er een geldig gevolg is.

**Semantisch tableau**

*Zoeken naar de einduitkomst van een formule “0”: links zetten, “1” zoeken: rechts zetten*

* Een sequent is een rijtje van de vorm:

φ1, ..., φn o ψ1, ..., ψm met φ1, ..., ψm formules, n ≥ 0, m ≥ 0

* Een waardering V is een tegenvoorbeeld van een sequent   
  φ1, ..., φn o ψ1, ..., ψm indien

V(φ1) = ... = (φn) = 1 en

V(ψ1) = ... = (ψn) = 0 en

* Belangrijk: een sequent φ1, ..., φn o ψ1, ..., ψm met φ1, ..., ψm heeft geen tegenvoorbeeld als er een i en een j bestaan zodat φi = ψj
* Om te bepalen of φ1, ..., φn ⊨ ψ gaan we onderzoeken of er al dan niet een tegenvoorbeeld is voor het sequent φ1, ..., φn o ψ

Dit doen we op een systematische manier: semantisch tableau

* Een semantisch tableau is een schema waarin op systematische wijze het mogelijk bestaan van tegenvoorbeelden van een gegeven sequent teruggebracht wordt (gereduceerd) tot dat van één of meer eenvoudigere sequenten.

'Nu als semantisch tableau: 
Heeft p v q, —q / p een tegenvoorbeeld? 
pvq,-qop 
pv q, —q op 
heeft een tegenvoorbeeld 
desda 
een tegenvoorbeeld heeft 
desda: dan en slechts dan als 
VRIJE 
UNIVERSITEIT 
BRUSSEL 
4— reductieregel 
p v q o p, q heeft een tegenvoorbeeld 
desda 
p o p, q een tegenvoorbeeld heeft 
of 
q o p, q een tegenvoorbeeld heeft 

Door het telkens toepassen van een reductieregel op de sequenten krijgen we een boom.

Als er geen reductieregel meer kan worden toegepast spreken we van een semantisch tableau.

* Elke tak correspondeert met een mogelijke route om een tegenvoorbeeld te vinden.

Indien links en rechts in het sequent dezelfde formule optreedt, is er geen tegenvoorbeeld. Men noemt de tak gesloten

Een tak is open indien de tak niet gesloten is en er kunnen geen reductiestappen meer worden toegepast op deze tak.

Een tableau is gesloten als al zijn takken gesloten zijn:

* Er is geen enkel tegenvoorbeeld te vinden, dus geldig gevolg!

Een tableau is open als er minstens één tak niet gesloten kan worden

* Dus tenminste één tegenvoorbeeld en dus geen geldig gevolg

**Semantisch consistent**

|  |  |
| --- | --- |
| Definitie | Een formuleverzameling ∑ is semantisch consistent als ∑ een model heeft (minstens 1 model) |

* Dus ∑ = {ϕ1 ∧ … ∧ ϕn} is consistent als ϕ1 ∧ … ∧ ϕn o een tegenvoorbeeld heeft

Het tegenvoorbeeld is de waardering die alle formules in ∑ (links in het sequent) waarmaakt (en rechts onwaar maakt - maar dat is leeg)

* Of nog ∑ = {ϕ1 ∧ … ∧ ϕn} is consistent als ϕ1 ∧ … ∧ ϕn o een open tableau heeft

Een formuleverzameling zonder model heet inconsistent. Het corresponderende tableau is gesloten.

Tautologie testen met behulp van een semantisch tableau:

* Als er geen waardering te vinden is zodat V(ϕ) = 0
* m.a.w. als er geen tegenvoorbeeld voor o ϕ is
* m.a.w. als het semantisch tableau voor o ϕ gesloten is

**Geldig gevolg - adequaatheidstelling**

* Semantische tableaus werden geïntroduceerd als een methode om de geldigheid van een gevolgtrekking te bepalen

|  |  |
| --- | --- |
| Adequaatstelling | ϕ1 ,…, ϕn ⊨ ψ  desda er bestaat een gesloten semantisch tableau voor ϕ1 ,…, ϕn o ψ  desda alle semantische tableaus voor ϕ1 ,…, ϕn o ψ zijn gesloten |

De adequaatstelling zegt dat alle mogelijke tableaus sluiten als ook maar 1 van hen sluit.

# Propositielogica: deel III (afleidingen)

**Afleidingen (natuurlijke deductie)**: hoe conclusies trekken uit aannames d.m.v. regels. Dit wordt afleiden genoemd

Afleidingsregels lineair voorgesteld:

bv. als φ ∧ ψ dan φ

Compactere voorstelling: boomvorm

j: Jan vertelt verhaal

p: Piet leest de krant

m: Marie lacht

w: Wilma kijkt televisie

A close up of a text

Description automatically generated

Als we met de gegeven afleidingsregels een formule ψ kunnen afleiden uit een formule φ (of formuleverzameling ∑),

dan schrijven we:

φ ⊢ ψ of ∑ ⊢ ψ

De formule φ (of formuleverzameling ∑) noemt men de aanname(s) in het bewijs

Vaak hebben we tijdens het afleiden hulpaannames nodig

* Om te zien op grond van welke aannames een formule geldt schrijven we:

ψ uit ∑

of

ψ uit φ1, ..., φn

* Notatie voor een hulpaanname: φ uit φ

|  |  |
| --- | --- |
| Definitie | Een formule φ is **afleidbaar** uit een verzameling aannames ∑ als er een afleiding van φ bestaat op basis van de gegeven afleidingsregels waarin aan het eind alleen nog aannames uit ∑ van kracht zijn |

Notatie: ∑ ⊢ φ

* Als φ niet afleidbaar is, noteren we dit ∑ ⊬ φ

Notatie:

* Alle afleidingsregels hebben een equivalente lineaire vorm bv. om de leesbaarheid te verhogen worden de namen van de toegepaste regels vaak weggelaten (tenzij expliciet gevraagd - bv. in de WPOs en op het examen)

(ΛΙ): ΑΙς φ uit Φ φ Λψ uit ΦΙ-) Ψ. 

**Stelling**

|  |  |
| --- | --- |
| Definitie | Als φ afleidbaar is zonder aannames, dan heet φ een stelling (Σ is leeg) |

Notatie: ⊢ φ

* Als φ geen stelling is, noteren we ⊬ φ

Natuurlijke deducties leiden tot stellingen indien men alle aannames kan intrekken met de regel ⟶ I

𝜑, 𝜓 ⊢ 𝜉 wordt 𝜑 ⊢ 𝜓 → 𝜉 en dit wordt ⊢ 𝜑 → (𝜓 → 𝜉)

Equivalentie 𝑝 ↔ 𝑞 kunnen we ook schrijven als (𝑝 → 𝑞) ∧ (𝑞 → 𝑝)

A screenshot of a computer screen

Description automatically generated

**Syntactisch consistent**

|  |  |
| --- | --- |
| Definitie | Een verzameling formules Γ heet syntactisch consistent wanneer er geen formule φ is, waarvoor zowel  Γ ⊢ φ als Γ ⊢ ¬ φ |

**Bewijstechnieken**

Rechtstreeks:

p ⟶ p1 p1 ⟶ p2 …pn ⟶ q

Daaruit volgt p ⟶ q

Onrechtstreeks:

* + Bewijs door contrapositie p ⟶ q is logisch equivalent met ¬ q ⟶ ¬ p dus je kan ¬ q ⟶ ¬ p bewijzen
  + Bewijs uit het ongerijmde (door contradictie) voor eender welke type stelling
  + Bewijs per inductie voor een aftelbaar eindig aantal

**Syntactisch consistent - beweringen**

|  |  |
| --- | --- |
| Bewering | Een formuleverzameling Γ is syntactisch consistent desda er bestaat een formule φ zodat Γ ⊬ φ  ⟹ TB: Als Γ consistent is, dan bestaat er een formule φ zodat Γ ⊬ φ  Als Γ consistent is, dan is er geen formule φ waarvoor Γ ⊢ φ en Γ ⊢ ¬φ  Voor een willekeurige formule ξ geldt dan: ofwel Γ ⊬ ξ ofwel Γ ⊬ ¬ξ  Als Γ ⊬ ξ dan is ξ de gezochte formule, zo niet is ¬ξ de gezochte formule  We hebben dus bewezen dat er een formule φ bestaat zodat Γ ⊬ φ  ⟸ TB: Als er een formule φ bestaat zodat Γ ⊬ φ dan is Γ consistent  Bewijs via contrapositie, m.a.w. we bewijzen:  Als Γ inconsistent is, dan bestaat er geen formule φ zodat Γ ⊬ φ  Bewijs: stel Γ inconsistent, dan bestaat er een formule ξ zodat Γ ⊢ ξ en Γ ⊢ ¬ξ  Maar dan is met de ¬E regel elke formule afleidbaar uit Γ, en dus kan er geen formule φ bestaan zodat Γ ⊬ φ |
| Bewering | Γ ⊬ φ ⟺ Γ ∪ {¬φ} is syntactisch consistent  Bewijs (beide richtingen ⟹⟸ contrapositie op toegepast)  ⟹ TB: Γ ⊬ φ dan Γ ∪ {¬φ} consistent, of nog: Γ ∪ {¬φ} inconsistent dan Γ ⊢ φ  Stel dus Γ ∪ {¬φ} is inconsistent  Dan is er een formule ξ waarvoor geldt Γ ∪ {¬φ} ⊢ ξ en Γ ∪ {¬φ} ⊢ ¬ξ  De ¬E\*regel geeft dan: Γ ⊢ φ (elke formule is afleidbaar)  ⟸ TB: Γ ∪ {¬φ} is consistent dan Γ ⊬ φ, of nog: Γ ⊢ φ dan Γ ∪ {¬φ} is inconsistent  Stel Γ ⊢ φ, dan ook Γ ∪ {¬φ} ⊢ φ. Omdat ook Γ ∪ {¬φ} ⊢ ¬φ, is Γ ∪ {¬φ} inconsistent |

**Axiomatisch afleiden?**

A close up of text

Description automatically generated

Dit axiomatisch systeem is equivalent met natuurlijke deductie op voorwaarde dat enkel de connectieven ⟶ en ¬ gebruikt worden (functioneel volledig)

Intuïtionistische logica: logica zonder ¬¬p (dubbele negatie = p) en ¬p ∨ p (uitgesloten derde)

**Volledigheidsstelling**

Deze stelling geeft het verband tussen syntactische afleidbaarheid en semantische geldigheid

|  |  |
| --- | --- |
| Volledigheidsstelling | Voor de propositielogica geldt  Als Σ een formuleverzameling is, en φ een formule, dan geldt: Σ ⊢ φ desda Σ ⊨ φ |

* Correctheid: afleidbare gevolgtrekkingen zijn semantisch geldig
* Volledigheid: semantisch geldige gevolgtrekkingen zijn afleidbaar

# Propositielogica: deel IV (metatheorie)

Metabeweringen: beweringen over het systeem (in tegenstelling tot de beweringen uitgedrukt in het systeem)

Voorbeelden: functionele volledigheid van connectieven en correctheid van bewijssystemen

**Formule-inductie**

Aantonen dat een bewering B geldt voor alle formules uit de propositielogica

* Gebaseerd op het principe dat formules zijn opgebouwd vanuit een basisverzameling van bouwstenen via eindige combinatie met logische operatoren
* Basisstap: laat zien dat B geldt voor de bouwstenen (propositieletters)
* Inductiestap: toon aan dat, als B geldt voor de formules φ, ψ, dan ook voor hun combinaties  
  ¬φ, (φ∧ψ), (φ ∨ ψ), (φ ⟶ ψ) en (φ ⟷ ψ)

De aanname in de inductiestap (dat B geldt voor de formules φ en ψ) heet de **inductiehypothese** (IH)

A close-up of a math formula

Description automatically generated

|  |  |
| --- | --- |
| Bewijs | Basisstap: voor φ = p, een propositieletter, geldt subf(p) = 1 en length(p) = 1  Dus bewering klopt voor de basisstap subf(p) = length(p), met p een willekeurig propositieletter  Inductiestap: Stel dat de bewering geldt voor de formules α en β (IH). Dan bewijzen we dat de bewering ook geldt voor:  φ = ¬α, φ = (α ∧ β), φ = (α ∨ β), φ = (α ⟶ β) en φ = (α ⟷ β)  voor φ = ¬α is  subf(¬α) = subf(α) + 1 =IH length(α) + 1 = length(¬α)  voor φ = (α ∧ β) is  subf(α) + subf(β) + 1 =IH length(α) + length (β) + 1 = length(α ∧ β)  voor φ = (α ∨ β) is  subf(α) + subf(β) + 1 =IH length(α) + length (β) + 1 = length(α ∧ β)  voor φ = (α ⟶ β) is  subf(α) + subf(β) + 1 =IH length(α) + length (β) + 1 = length(α ∧ β)  voor φ = (α ⟷ β)  subf(α) + subf(β) + 1 =IH length(α) + length (β) + 1 = length(α ∧ β) |

**Functioneel volledig**

De verzameling connectieven {¬, ∧} is functioneel volledig, m.a.w. voor elke formule φ is er een logisch equivalente formule φ’ die alleen de connectieven ¬ en ∧ bevat

|  |  |
| --- | --- |
| Bewijs | Basisstap: voor φ = p, een propositieletter, geldt dit automatisch want deze bevatten geen connectieven  Inductiestap:  IH: stel dat φ en ψ herschreven kunnen worden als logisch equivalente formule φ’ en ψ’, waarbij φ’ en ψ’ alleen met ¬ en ∧ zijn geconstrueerd. (φ ≡ φ’, ψ ≡ ψ’, ≡ herschreven)  TB: ook ¬φ, φ ∧ ψ, φ ∨ ψ, φ ⟶ ψ en φ ⟷ ψ zo te herschrijven  Bewijs:  (¬φ) ≡ ¬(φ) ≡IH ¬(φ’) ≡ (¬φ’)  (φ ∧ ψ) ≡IH (φ’ ∧ ψ’)  (φ ∨ ψ) ≡ IH (φ’ ∨ ψ’) ≡ ¬(¬φ’ ∧ ¬ψ’)  (φ ⟶ ψ) ¬(φ ∧ ¬ψ) ≡IH ¬(φ’ ∧ ¬ψ’)  (φ ⟷ ψ) ≡ (φ ⟶ ψ) ∧ (ψ ⟶ φ) ≡ IH ¬(φ’ ∧ ¬ψ’) ∧ ¬(ψ’ ∧ ¬φ’) |

**Adequaatheidsstelling**

ϕ1 ,…, ϕn ⊨ ψ

desda er bestaat een gesloten semantisch tableau voor ϕ1 ,…, ϕn o ψ

desda alle semantische tableaus voor ϕ1 ,…, ϕn o ψ zijn gesloten

Eerst veralgemening:

Φ o Ψ met Φ = ϕ1 ,…, ϕn en Ψ = ψ1 ,…, ψm

Het tableau met Φ o Ψ als topsequent gesloten als

𝜑1 ∧ … ∧ 𝜑𝑛 ⊨ 𝜓1 ∨ … ∨ 𝜓𝑚 of anders geschreven als 𝜑1 ,…, 𝜑𝑛 ⊨ 𝜓1 ,…, 𝜓𝑚 of ook wel Φ ⊨ Ψ

**Adequaatheidsstelling - bewijs**

|  |  |
| --- | --- |
| Bewering | Stel Φ o Ψ is de topsequent van een gesloten tableau  Dan geldt Φ ⊨ Ψ  Bewijs per inductie op het aantal knopen van het tableau  Basisstap: tableau heeft één knoop: omdat tableau gesloten is, betekent dit dat een zekere formule zowel links in Φ als rechts in Ψ voorkomt.  Dan kan er geen tegenvoorbeeld zijn en dus Φ ⊨ Ψ  Inductiestap: tableau heeft meerdere knopen: ontstaan door toepassen van een regel  Geval I (reductie zonder splitsing): door toepassing van ¬L regel   * Topsequent: Φ, ¬α o Ψ * Sequent eronder: Φ o α, Ψ. Hier geldt per inductie Φ ⊨ α, Ψ (want 1 knoop minder en gesloten) (IH)   TB: als V(Φ) = 1 en V(¬α) = 1 dan V(Ψ) = 1 (nl. Φ, ¬α ⊨ Ψ)  Stel dus een waardering V zodat V(Φ) = 1 en V(¬α) = 1. Aangezien V(Φ) = 1 volgt op basis van de IH, nu dat deze bewering dan α of Ψ waar moet maken. Omdat V(¬α) = 1 en dus V(α) = 0 moet V dus Ψ waar maken en hebben we dan bewezen: Φ, ¬α ⊨ Ψ  Geval II (reductie met splitsing): door toepassing ∧R regel   * Topsequent Φ o α ∧ β, Ψ * Sequent eronder: Φ o α, Ψ en Φ o β, Ψ. Hier geldt per inductie Φ ⊨ α, Ψ en Φ ⊨ β, Ψ (door IH, want beide tableaus een knoop minder en gesloten   TB: V(Φ) = 1 dan V(α ∧ β) = 1 of V(Ψ) = 1  Als V(Φ) = 1 dan zal die waardering ofwel Ψ waarmaken ofwel α en β waarmaken (door IH).  En dus Φ ⊨ α ∧ β, Ψ |
| Bewering | Stel een open tableau heeft een tak t die niet sluit. Zij V een waardering die alle propositieletters links op de tak t waarmaakt en al deze rechts onwaar maakt. Dan geldt voor elke formule φ: als φ links op de tak t voorkomt dan V(φ) = 1 en als φ rechts op de tak t voorkomt dan V(φ) = 0  Bewijs per formule-inductie op φ, en in een open tableau is elke toepasbare regel toegepast  Basisstap: tableau bestaat uit propositieletters, t.b. per definitie van V  Inductiestap: (bewijs enkel voor 2 gevallen, rest analoog)  Bewijs voor ¬φ  Als ¬φ links voorkomt, dan moet er een keer de ¬L regel zijn toegepast waar φ rechts voorkomt.  Dan is (door IH) V(φ) = 0, zodat V(¬φ) = 1  Bewijs voor φ ∧ ψ  Als φ ∧ ψ rechts voorkomt dan moet er een keer ∧R regel zijn toegepast.  De tak t volgt in de splitsing de linker- of rechterkant.  Neem bijvoorbeeld de linkerkant met φ rechts.  Door IH is V(φ) = 0 en dan is ook V(φ ∧ ψ) = 0  Indien rechterkant: door IH is V(ψ) = 0 en dat is ook V(φ ∧ ψ) = 0 |
| Bewijs | Adequaatheidsstelling  ϕ1 ,…, ϕn ⊨ ψ  desda er bestaat een gesloten semantisch tableau voor ϕ1 ,…, ϕn o ψ  ⟸ is een speciaal geval van bewering 1  ⟹ stel er is geen gesloten tableau. Dan moet er een open tableau zijn.  Door bewering 2 levert elke open tak dan een tegenvoorbeeld zodat de geldigheid weerlegt wordt (en dit is met een contradictie met het gegeven)  Bijkomend: als één tableau voor een sequent sluit dan sluiten alle tableaus voor dit sequent  Bewijs: als één tableau sluit dan is de gevolgtrekking geldig. Als er een open tableau zou zijn dan zou gevolgtrekking niet geldig zijn. Dit is een contradictie dus alle tableaus moeten sluiten |

**Volledigheidsstelling**

|  |  |
| --- | --- |
| Volledigheidsstelling | Voor de propositielogica geldt dat voor iedere formuleverzameling Σ en iedere formule φ Σ ⊢ φ desda Σ ⊨ φ   * correctheid: Σ ⊢ φ impliceert Σ ⊨ φ * volledigheid: Σ ⊨ φ impliceert Σ ⊢ φ   Met een lege verzameling Σ zegt dat deze stelling dat de tautologieën van de propositielogica precies de stellingen van het bewijssysteem natuurlijke deductie zijn |

# Predikaatlogica: deel 1

**Taal**

Predikaatlogica is nodig om eigenschappen (predicaten) van individuen te beschrijven en erover te redeneren

* 1. Predikaatzinnen bv. Brussel is een stad. Parijs is de hoofdstad van Fr. Stef is kind van Kees Ada

Predicaten worden voorgesteld via predikaatletters (hoofdletters) met vast aantal argumenten (volgorde is belangrijk). Individuen worden voorgesteld via constanten (kleine letters)  
V(p) (V = voetbalt, p = Piet)

* 1. Kwantoren: uitdrukkingen van hoeveelheid ∀ en ∃

Alle koninkrijken hebben een koning. Er is minstens één land dat een koninkrijk is.

* 1. Variabelen: gebruik van kwantoren introduceert de nood aan variabelen (kleine letters x, y, z…)

∀x((¬Mx ∨ ¬Fx ∨ ¬ Bx) “Alle mensen zijn niet mannelijk of niet filosoof of geen baard”

* 1. Functies: voorgesteld door kleine letters (f, g..)

¬∃x(J(I(x), (I(x))) “Niemand is jonger dan zichzelf”

**Alfabet**

|  |  |
| --- | --- |
| Definitie | Het alfabet van een predicaat logische taal bestaat uit:   * + Een verzameling C van individuele constanten: a, b, c…   + Een verzameling P van predikaatletters: P, Q, R…   + Een verzameling F van functieletters: f, g, h…   + De logische symbolen: ¬, ∧, ∨, ⟶, ⟷, ∀, ∃   + Een aantal individuele variabelen: u, v, w , x, y, z…   + De hulpsymbolen: ) en ( |

Plaatsigheid (aantal argumenten) voor functieletters en predikaatletters via een index : f3(a, b, c): 3-plaatsig

Propositieletters kunnen beschouwd worden als 0-plaatsige predikaatletters

**Syntaxis - term**

Termen duiden individuele objecten aan

|  |  |
| --- | --- |
| Definitie | De **termen** in de predikaatlogica zijn als volgt gedefinieerd (inductieve definitie):   * + Individuele variabelen en constanten zijn termen;   + Als f een k-plaatsige functieletter is en t1 ,…, tk zijn termen, dan is f(t1 ,…, tk) ook een term;   + Niets anders is een term |

Voorbeeld: f3(g2(x, h1(y)),a, g 2(a, y))

Meestal wordt in logica prefix notatie gebruikt. +(x, y)

**Syntaxis - formule**

|  |  |
| --- | --- |
| Definitie | De **formules** in de predikaatlogica zijn als volgt gedefinieerd (inductieve definitie):   * + Als P een k-plaatsige predikaatletter is en t1 ,…, tk zijn termen dan is P(t1 ,…, tk) een formule;   + Als φ en ψ formules zijn, dan zijn ook ¬φ, (φ ∧ ψ), (φ ∨ ψ), (φ ⟶ ψ) en (φ ⟷ ψ) formules;   + Als φ een formule is en x een individuele variabele, dan zijn ∀x φ en ∃x φ ook formules;   + Niets anders is een formule |

Een formule van de vorm P(t1 ,…, tk) heet een atomaire formule of atoom

Voorbeeld: ∀x (Mx ⟶ Gxm) “Max ziet alle mensen graag”

Predikaatlogica is geschikt om wiskundige beweringen te formaliseren bv. rekenkunde, meetkunde, verzamelingenleer…

**Bereik van kwantoren**

Het bereik van een kwantor is die subformule waarvoor die kwantor is ingevoegd tijdens de constructie van die formule

A diagram of a mathematical equation

Description automatically generated with medium confidence

**Vrije en gebonden variabelen**

|  |  |
| --- | --- |
| Definitie | Een voorkomen van een variabele x is vrij als het niet in het bereik van een kwantor ∀x of ∃x ligt  Een kwantor ∀x of ∃x bindt de voorkomens van variabelen x in zijn bereik, voor zover die nog vrij zijn.  Deze voorkomens zijn dan gebonden |

A diagram of a mathematical equation

Description automatically generated

|  |  |
| --- | --- |
| Definitie | Een **zin of een gesloten formule** is een formule zonder vrije variabelen  Een formule met vrije variabelen heet een open formule |

**Substitutie**

|  |  |
| --- | --- |
| Definitie | * Als t, t’ termen zijn, x een variabele dan is [t / x]t’ de term die ontstaat door elk voorkomen van x in t’ te vervangen door t * Als φ een formule is, t een term en x een variabele, dan is [t / x]φ de formule die ontstaat door elk voorkomen van x als vrije variabele in φ te vervangen door t   [t / x]φ wordt ook wel een instantie van φ genoemd  (omdat je een vrije variabele gaat vervangen door iets concreet) |

Voorbeeld: [y/x](∀x(Rx ∨ Sx) ∨ Pxx) = ∀x(Rx ∨ Sx) ∨ Pyy

**“Veilige” substitutie**

|  |  |
| --- | --- |
| Definitie | Een term t heet vrij voor x in φ als in [t / x]φ geen variabele van t gebonden wordt |

**Alfabetische variant**

Een alfabetische variant van een formule ontstaat door gebonden variabelen door nieuwe variabelen te vervangen

* Intuïtief verandert er hierdoor niets
* ∃y (y < z) heeft als alfabetische variant: ∃z (z < x)

Het wordt gebruikt om substitutie veilig uit te voeren

# Predikaatlogica: deel II (semantiek)

**Principe**

Drie aspecten zijn belangrijk

A close-up of a word

Description automatically generated  
De interpretatiefunctie geeft betekenis aan bouwstenen van de taal (predikaatletters, functieletters, constanten, variabelen)

Alle drie aspecten zijn essentieel (vergelijking met natuurlijke taal)

* Stel tekst gekend en woordenboek (interpretatiefunctie), maar niet de situatie  
  We kunnen niet weten wat gezegd wordt waar is
* Stel tekst gekend en de situatie, maar geen woordenboek (interpretatiefunctie)  
  We kunnen niet weten of wat gezegd wordt waar is
* Stel een woordenboek (interpretatiefunctie) en de situatie, maar niet de tekst  
  We weten niet wat er gezegd wordt

**Structuur**

Structuur is te vergelijken met een stukje werkelijkheid

Een structuur bestaat uit een domein waarop relaties en operaties of functies gedefinieerd zijn

Voorbeeld: domein (natuurlijke getallen, relaties (<, =), operaties (+, \*)

|  |  |
| --- | --- |
| Definitie | Een structuur D is drietal <D, R, O> bestaande uit een niet-lege verzameling D (het domein), een verzameling R van relaties op D en een verzameling O van operaties op D |

**Speciale structuren**

**A white background with blue and orange text

Description automatically generated**

**Interpretatiefunctie**

|  |  |
| --- | --- |
| Definitie | Laat D = <D, R, O> een structuur zijn  Een interpretatiefunctie I kent aan   * + Elke individuele constante c uit de predikaatlogica een welbepaald object uit D toe via een nul-plaatsige operatie, I(c) ∈ O   + Elke predikaatletter P een relatie uit R van dezelfde plaatsigheid toe, I(P) ∈ R   + Elke functieletter f een operatie uit O van dezelfde plaatsigheid toe, I(f) ∈ O |

**Model**

|  |  |
| --- | --- |
| Definitie | Een paar (D, I) met D een structuur en I een interpretatiefunctie is een model |

Een (on)eindig model is een model met een (on)eindig domein

**Bedeling**

|  |  |
| --- | --- |
| Definitie | Een bedeling b is een functie die aan elke variabele x een object uit het domein toekent, dus b(x) ∈ D |

Notatie: <N, {<, =}, {0, 1, +, .}). ook als < N, <, =, 0, 1, +, .} wanneer duidelijk is wat de relaties en operaties zijn

Zo ook: P i.p.v. I(P) en f i.p.v. I(f)

𝑏[𝑥 ⟼ 𝑑] is de bedeling b waarbij d aan de variabele x wordt toebedeeld dus (x) = d

**Waardering van termen**

|  |  |
| --- | --- |
| Definitie | Laat M = (D, I) een model zijn en b een bedeling  Dan is de semantisch waardering VM, b van termen als volgt gedefinieerd:   * + VM, b (x) = b(x) voor variabelen x   + VM, b (a) = I(a) voor constanten a   + VM, b (f(t1 ,…, tk )) = I(f)(VM, b (t1 ),…, VM, b (tk ))   De waarheidswaarden van formules zijn als volgt gedefinieerd:   * + VM, b(P(t1 ,..., tm)) = 1 desda I(P)(VM, b(t1),..., VM, b(tm)) geldt   + VM, b(¬φ) = 1 desda VM, b(φ) = 0   idem als in propositielogica voor φ ∧ ψ, φ ∨ ψ, φ ⟶ ψ en φ ⟷ ψ.   * + VM, b(∃x φ) = 1 desda er is een d ∈ D zodat VM, b[x ⟼ d](φ) = 1   + VM, b(∀x φ) = 1 desda voor alle d ∈ D geldt VM, b[x ⟼ d](φ) = 1 |

Een atomaire formule is waar in een structuur, als het feit dat wordt uitgedrukt waar is in de structuur

Eén formule φ kan in verschillende structuren heel verschillende beweringen uitdrukken

Bij gegeven φ en één structuur D kunnen verschillende interpretatiefuncties aan φ een andere waarheidswaarde geven

**Model van een formule**

|  |  |
| --- | --- |
| Definitie | Een paar M = (D, I) met structuur D en interpretatiefunctie I heet een model van een formule φ als voor iedere bedeling b geldt: VM, b (φ) = 1 |

Notatie:

* + VM, b (φ) = 1 φ is waar in M onder b
  + V(φ) = 1 als geen verwarring mogelijk
  + VM, b (φ) = 1 M, b ⊨ φ of M ⊨ φ[b]
  + VM, b (φ) = 0 M, b ⊭ φ of M ⊭ φ[b]

**Gelijkheid van termen**

De gelijkheidsrelatie (“=”) is niet standaard gedefinieerd in de taal:

Indien nodig: expliciet te definiëren; alsook de semantiek ervan

**Eigenschappen**

Eigenschappen van de waarheidsfunctie:

Waarheidswaarde van een formule hangt af van de structuur D, de interpretatiefunctie I en van het effect van de bedeling b op de vrije variabelen in die formule (en dus niet van de bedeling van de gebonden variabelen)

Dit is gebaseerd op de volgende bewering

|  |  |
| --- | --- |
| Bewering | Als een formule φ vrije variabelen x1 ,…, xk bevat en er zijn 2 bedelingen b1 en b2 met b1(xi) = b2(xi),  voor i = 1 ,…, k dan geldt VM, b2(φ) |

Gevolg: voor zinnen (gesloten formules) zijn er geen vrije variabelen, dus doet de bedeling er niet toe  
voor zinnen spreken we dus over de waarheid en onwaarheid in een model

**Substitutie – eigenschappen**

|  |  |
| --- | --- |
| Bewering | Voor alle termen t, t’ geldt VM, b([t / x]t’) = VM, b[x ⟼ VM, b(t)](t’) |

Links: eerst substitueren en dan de waarde van t’ bepalen

Rechts: waarde van t’ berekenen en we bedelen de waardering van t aan x

Dus wat betreft de waardering is substitutie in een term hetzelfde als substitutie in de bedeling

**Geldig gevolg**

|  |  |
| --- | --- |
| Definitie | Laat Σ een verzameling formules zijn en ψ een formule, dan ψ volgt uit Σ, Σ ⊨ ψ, desda:  voor elk model M en elke bedeling b geldt:  als voor elke φ ∈ Σ geldt dat VM, b(φ) = 1 dan ook VM, b(ψ) = 1 |

! oneindig veel mogelijkheden voor M en b !

|  |  |
| --- | --- |
| Definitie | Een formule ψ heet **universeel geldig** als ⊨ ψ |

Intuïtief: een universeel geldige formule is waar in alle modellen M onder iedere bedeling b

|  |  |
| --- | --- |
| Definitie | Twee formules φ en ψ heten **logisch equivalent** als ⊨ φ ⟷ ψ |

**Theorie**

|  |  |
| --- | --- |
| Definitie | {φ | φ is een zin en VM(φ) = 1} is een **theorie voor een model M** |

Notatie Th(M)

* + Intuïtief: de verzameling van ware zinnen is een theorie voor M
  + We kunnen ook een theorie weergeven door axioma’s

|  |  |
| --- | --- |
| Definitie | **Axiomaverzameling voor een model M**  Een formuleverzameling Σ axiomatiseert een theorie Th(M) als voor alle zinnen φ geldt:  φ ∈ Th(M) desda Σ ⊨ φ |

Een goede axiomatisering geeft de essentiële kenmerken van het model weer.

**Modelverzameling**

|  |  |
| --- | --- |
| Definitie | **Modelverzameling voor de zin** **φ**, MOD (φ) is {M | VM(φ) = 1}  **Modelverzameling voor de verzameling zinnen Σ**, MOD (Σ), is {M | VM(φ) = 1 voor alle φ ∈ Σ} |

MOD is verzameling van modellen die zin, resp. een verzameling zinnen, waarmaakt.

# Predikaatlogica: deel III (semantische tableaus)

Zoals in de propositielogica kunnen we semantische tableaus gebruiken om de geldigheid van een gevolgtrekking te testen

Hoofdidee: zoeken van een tegenvoorbeeld voor φ1 ,…, φn o ψ

Let op: tegenvoorbeeld in predikaatlogica is

Er bestaat een model en een bedeling φ1 ,…, φn waar maakt en 𝜓 onwaar maakt. Dus structuur (domein), interpretatiefunctie en bedeling nodig

A white background with blue and orange text

Description automatically generated

Mogelijkheden:

* 1. Het tableau sluit, de gevolgtrekking is geldig
  2. Er is een niet-sluitende tak. Deze kan:
     + Eindig afbreken, of
     + Oneindig doorlopen  
       In beide gevallen: tegenvoorbeeld; gevolgtrekking is niet geldig

Oneindige tak: tegenvoorbeeld met een oneindig domein

**Predikaatlogica & onbeslisbaarheid**

De predikaatlogica is niet beslisbaar

Er bestaat geen algoritme dat voor iedere willekeurige formule kan bepalen of deze afleidbaar is of logisch gevolg is uit de formuleverzameling.

Sommige fragmenten van de predikaatlogica zijn wel beslisbaar

Semantische tableaus zijn wel adequaat voor de predikaatlogica:

Een sequent heeft een gesloten tableau desda de corresponderende gevolgtrekkking is geldig.

Zo’n gesloten tableau wordt gevonden wanneer men zorgt voor **fair scheduling** van de reductieregels

(d.w.z. ze moeten allemaal aan de beurt komen)

# Predikaatlogica: deel IV (afleidingen)

**Afleidingen**

We beperken ons tot formules zonder vrije variabelen

A white background with black and red text

Description automatically generated

Conditie “Mits t vrij is voor x in φ” in ∃I is nodig om de volgende ongewenste afleiding te vermijden:

Uit ∀y Ryy mogen we niet afleiden:

∃x ∀y Ryx omdat y niet vrij voor x is in ∀y Ryx

Nl. φ = ∀y Ryx ; t = y dan [y / x] ∀y Ryx = ∀y Ryy

Dus y is nu gebonden! En dus was y niet vrij voor x in ∀y Ryx

Zoals bij propositielogica bestaat er ook een alternatief bewijssysteem, nl. axioma’s en afleidingsregels

# Predikaatlogica: deel V (een eenvoudige theorie)

**Substitutie**

|  |  |
| --- | --- |
| Bewering | Voor alle termen t, t’ geldt VM, b([t / x]t’) = VM, b[x ⟼ VM, b(t)](t’) |

Bewijs: met inductie naar de opbouw van t’

|  |  |
| --- | --- |
| Bewijs | VM, b([t / x]t’) = VM, b[x ⟼ VM, b(t)](t’)  Eerst gaan we het bewijzen voor het geval t’ een variabele is of een constante:   * + Als t’ = x dan [t / x]t’ = t, en dan V([t / x]t’) = V(t)   Anderzijds Vb[x ⟼ V(t)](t’) = Vb[x ⟼ V(t)](x) = V(t)   * + Als t’ = y en y is een andere variabele of constante, dan [t / x]t’ = y en V([t / x]t’) = V(y)   Anderzijds Vb[x ⟼ V(t)](t’) = Vb[x ⟼ V(t)](y) = V(y)  Nu moeten we het bewijzen voor de functie   * + Inductiehypothese: stel bewering geldt voor termen ti, i = 1 ,…, n   Dan geldt voor t’ = f(t1 ,…, tn) dat V([t / x]t’) = I(f)(V([t / x]t1) ,…, (V([t / x]tn))  = I(f)(Vb[x ⟼ V(t)](t1) ,…, Vb[x ⟼ V(t)](tn))  Anderzijds Vb[x ⟼ V(t)](t’) = Vb[x ⟼ V(t)](f(t1 ,…, tn)) = I(f) (Vb[x ⟼ V(t)](t1) ,…, (Vb[x ⟼ V(t)](tn) |

Niet kennen?

A screenshot of a white box with blue text

Description automatically generatedA white background with blue text

Description automatically generated

**Prenexvorm**

Er bestaat een verband tussen ∀ en ∃

Equivalenties met ∀, ∃ en ¬:

|  |  |
| --- | --- |
| Bewijs | * + ∀x ¬φ is logisch equivalent met ¬∃x φ   V(∀x ¬φ) = 1  Desda voor alle d ∈ D geldt dat V b[x ⟼ d] (¬φ) = 1 desda voor alle d ∈ D geldt dat V b[x ⟼ d] (φ) = 0  desda er is geen d ∈ D zodat V b[x ⟼ d] (φ) = 1 desda V(∃x φ) = 0 desda V(¬∃x φ) = 1   * + ∃x ¬φ is logisch equivalent met ¬∀x φ   V(∃x ¬φ) = 1  Desda er is een d ∈ D geldt dat V b[x ⟼ d] (¬φ) = 1  desda niet voor alle d ∈ D geldt dat V b[x ⟼ d] (¬φ) = 1 desda V(¬∀x φ) = 1 |

Equivalenties met ∀, ∃, ∧ en ∨:

A white background with blue text

Description automatically generated

Als x toch vrij voorkomt in ψ dan kunnen we in ∃x φ of ∀x φ overgaan op een alfabetische variant

Equivalenties met ∀, ∃ en ⟶:

A white background with blue text

Description automatically generated

Nl. (voorbeeld): (∃x φ) ⟶ ψ ≡ ¬(∃x φ) ∨ ψ ≡ (∀x ¬φ) ∨ ψ ≡ ∀x (¬φ ∨ ψ) ≡ ∀x (φ ⟶ ψ)

Equivalenties met ∀, ∃ en ⟷:

A screenshot of a computer

Description automatically generated

**Prenexvorm**

|  |  |
| --- | --- |
| Definitie | Een formule van de vorm Q1 x1 … Qn xn ψ, waarbij   * + Qi kwantoren (∃ of ∀) zijn   + i = 1 ,…, n   + ψ een formule is waarin geen kwantoren meer voorkomen heet een prenexvorm met Q1 x1 … Qn xn als prefix en ψ als matrix |

|  |  |
| --- | --- |
| Prenexstelling | Voor elke formule φ bestaat er een prenexformule ψ zodat φ en ψ logisch equivalent zijn  (V(φ) ≡ V(ψ)) |

Bewijs met inductie naar φ

De volgorde van de kwantoren vooraan hangt af van de volgorde waarin je ze naar voren haalt

**Fragmenten van de predikaatlogica**

Men beperkt zich vaak tot delen (fragmenten van de predikaatlogica

* Monadische taal: enkel één-plaatsige predikaatletters
  + Voldoende voor behandeling van syllogismen

Syllogismen: gevolgtrekkingen met 2 aannames en 1 conclusie van de vorm ‘alle / geen / sommige A is / zijn B

Bv. Alle kaaimannen zijn reptielen, geen reptiel kan fluiten, dus geen kaaiman kan fluiten

**Universele formules**: alleen universele kwantoren in hun prefix

**Horn-zinnen** zijn universele zinnen van de vorm ∀x1 … ∀xn (A1 ∧ … ∧ Ak) ⟶ B waarbij A1 ,…, Ak , B atomaire bewegingen zijn

# Predikaatlogica: deel VI (metatheorie)

Beweringen over de predikaatlogica

**Adequaatheidsstelling**

Semantische tableaus zijn een juiste methode om de geldigheid van een gevolgtrekking in de predikaatlogica te bewijzen of te weerleggen

Herinner: beperkt tot taal zonder functiesymbolen, zonder individuele constanten en zonder vrije variabelen

|  |  |
| --- | --- |
| Adequaatstelling | Als Σ een formuleverzameling is en φ een formule, dan geldt: Σ ⊨ φ  desda er bestaat een gesloten semantisch tableau voor Σ o φ |

**Volledigheidsstelling**

|  |  |
| --- | --- |
| Volledigheidsstelling | Voor de predikaatlogica geldt dat voor iedere formuleverzameling Σ en iedere formule φ Σ ⊢ φ desda Σ ⊨ φ   * Correctheid (“soundness”): Σ ⊢ φ impliceert Σ ⊨ φ * Volledigheid(“completeness”): Σ ⊨ φ impliceert Σ ⊢ φ |

# Lambda-calculus: deel I

Geïntroduceerd door Alonzo Church en Stephen Kleene in de jaren 1930s om het begrip functie abstract te kunnen bestuderen. Ook gebruikt om het begrip berekenbaarheid te formaliseren. Om na te gaan welke functies al dan niet berekenbaar zijn, of nog, welke (wiskundige) problemen oplosbaar zijn, en welke niet

(Een functie f is berekenbaar desda er bestaat een programma P dat de functie f berekent, m.a.w. als P stopt voor input a dan geeft die de correcte output en anders stopt P nooit)

Hoewel ontwikkeld vóór het bestaan van concrete computertalen vormt het formalisme de basis van functionele programmeertalen zoals Lisp en Scheme

Basis van deze programmeertalen zijn nl. functies

Invloed op programmeertalen:

* + “Call by name” mechanisme  
    parameters worden pas geëvalueerd indien nodig
  + Hogere orde functies

Functie met functies als parameters of

Output is een functie

**Functies**

Een numerieke functie is op twee manieren te definiëren

* Extensioneel (verzameling theoretisch)

{…, (-1, 1), (0, 2), (1, 3), …} extensie

* Intentioneel (functievoorschrift)

Bv. f(x) = 2 + x functievoorschrift

Prefixnotatie en geen naam nodig voor functie

**Basis**

2 fundamentele bewerkingen:

* Abstractie: het maken van een functievoorschrift
* Toepassen (of aanroepen) van het functievoorschrift op een parameter

**Syntax of λ-calculus**

|  |  |
| --- | --- |
| Definitie | Laat V een verzameling variabelen zijn V = {x, y, z…}  De **verzameling van λ-expressies** wordt als volgt gedefinieerd:   * 1. Alle variabelen (elementen van V) zijn λ-expressies   2. Als M en N λ-expressies zijn dan is (M)N een λ expressie (toepassing)   3. Als x ∈ V en M λ-expressie dan is λx.M een λ-expressie (abstractie)   4. Niets anders is een λ-expressie |

m.a.w.: definitie van functie in λ-calculus: λformele-parameter.functievoorschrift

aanroep van een functie in λ-calculus: (functie) actuele-parameter

De verzameling van λ-expressies wordt genoteerd als Λ

Toepassing van rechts naar links

A white background with blue text

Description automatically generated

**Currying**

Functie met meerdere parameter kan herleid worden tot functies met 1 parameter

A math equations and formulas

Description automatically generated with medium confidence

Deze techniek wordt currying genoemd, genoemd naar Amerikaanse wiskunde Curry

Voordeel: theorie te beperken tot functies met 1 argument

Nadeel: leesbaarheid

# Lambda-calculus: deel II

**Binden van variabelen**

Voor een λ-expressie van de vorm λx.M zegt men:

λx bindt x in de λ-expressie M

Het bereik van de binding is M, d.w.z. dat alle nog niet gebonden voorkomens van x in λx.M gebonden worden bv.



**Vrije en gebonden variabelen**

In een λ-expressie heten alle voorkomen van variabelen die niet gebonden zijn vrij

|  |  |
| --- | --- |
| Definitie | Laat M een λ-expressie zijn. Dan wordt de **verzameling VV(M) van vrije variabelen van M** als volgt gedefinieerd:   * + Voor alle variabelen x ∈ V: VV(x) = {x}   + VV(λx.M) = VV(M) \ {x}   + VV((M)N) = VV(M) ∪ VV(N) |
| Definitie | Een λ-expressie zonder vrije variabelen heet gesloten of een combinator. De verzameling van combinatoren wordt genoteerd als Λ0 |

**Substitutie**

A close-up of a math problem

Description automatically generated

Intuïtief: door elk voorkomen van de parameter x in M te vervangen door P

|  |  |
| --- | --- |
| Definitie | Zij P en M twee λ-expressies, en x een variabele x ∈ V  De substitutie {p / x}M van P voor x in M, is als volgt gedefinieerd:  (S1) {P / x}x = P  (S2) {P / x}y = y als y ∈ VV\{x}  (S3) {P / x}(F)Q = ({P / x}F){P / x}Q  (S4) {P / x}λx.M = λx.M  (S5) {P / x}λy.M = λy.{P / x}M als y ≠ x en y ∉ VV(P)  (S6) {P / x}λy.M = λz.{P / x}{z / y}M als y ≠ x en z ∉ VV(M) |

A screenshot of a computer

Description automatically generated

**β-gelijkheid**

|  |  |
| --- | --- |
| Definitie | De β-gelijkheid relatie =β ⊆ Λx Λ is gedefinieerd als volgt:  (β) (λx.M)P =β {P / x}M  (α) λx.M =β λz.{z / x}M als z ∉ VV(M)  (reflexief) M =β M  (symmetrisch) M =β N dan ook N =β M  (transitief) M =β N en N =β L dan ook M =β L  (congruent) M =β M’ en P =β P’ dan (M)P =β (M’)P’  (congruent) M =β M’ dan λx.M =β λx.M’ |

**Rekenen in λ-calculus**

λ-calculus kent geen constanten. Toch kan men de natuurlijke getallen voorstellen d.m.v. de λ-calculus

We kunnen zelf een volledige programmeertaal maken met: constanten, rekenkundige functies, booleans, if-then constructies, recursie…

|  |  |
| --- | --- |
| Definitie | Beschouw **λ-expressies** F en M. De expressie (F)n M wordt inductief gedefinieerd als volgt:  (F)0 M ≡ M  (F)1+n M ≡ (F)(F)n M |

**Rekenen in λ-calculus**

|  |  |
| --- | --- |
| Lemma (“+”) | Beschouw λ-expressies F en M  Voor alle n, m ∈ ℕ geldt dat (F)n+mM ≡ (F)n(F)mM  Bewijs: per inductie op n  Basisgeval n = 0  (F)0+mM ≡ (F)mM  Definieer M’ ≡ (F)mM  ≡ M’  ≡ (F)0M’ (def)  (F)0(F)mM  Stel lemma geldt voor het geval n = k (IH);  Nu bewijzen voor het geval n = k + 1 of 1 + k:  (F)1 + k + mM = F1 + (k + m)M  ≡ (F)(F)(k + m)M (def)  ≡ (F)(F)k(F)mM (IH)  ≡ (F)(F)kM’ (M’ ≡ (F)mM)  ≡ (F)1+kM’ (def)  ≡ (F)1+k(F)mM (M’ ≡ (F)mM) |

**Church getallen**

Voorstelling natuurlijke getallen:

A diagram of a function

Description automatically generated with medium confidence

|  |  |
| --- | --- |
| Definitie | De zogenaamde Church getallen cn (n ∈ ℕ) worden gedefinieerd als volgt:  cn ≡ λf.λx.(f)nx |

Functievoorschrift met 2 argumenten; het 1ste argument (een functie) wordt een aantal keer (n) toegepast op het 2de argument (de parameter voor de functie)

**λ-definieerbaar**

|  |  |
| --- | --- |
| Definitie | Een numerieke functie f: ℕp ⟶ ℕ met p parameters ∈ ℕ is λ-definieerbaar als er een combinator F bestaat zodat (((F)Cn1) Cn2) …) Cnp =β cf(n1, n2 ,…, np) |

m.a.w. er bestaat een combinator waarvan het effect op de Church getallen hetzelfde is als de operatie toegepast op de ‘echte’ getallen

**Successor**

Succ(n) = n + 1 is λ-definieerbaar

|  |  |
| --- | --- |
| Bewijs | Zoek succ zodat: (succ)cn =β cn+1  We hebben dus nodig:  (succ)c0 ≡ (succ) λf.λx =β λf.λx(f)x ≡ c1  (succ)c1 ≡ (succ) λf.λx.(f)x =β λf.λx(f)(f) x ≡ c2  In het algemeen moet (succ)) dus een extra “aanroep” van f toevoegen:  (succ)cn =β λf.λx.(f) voorschriftn  Voorschriftn vinden:  (succ)cn =β λf.λx.(f) voorschriftn  Herinner dat voorschriftn een afkorting is voor (f)nx en cn ≡ λf.λx.(f)nx  Dus als we de λf.λx. kunnen wegwerken, hebben we het voorschrift:  A screenshot of a math test  Description automatically generated |

**Optelling**

De optelling is λ-definieerbaar en de combinator is plus ≡ λn.λm.λf.λx.((n)f)((m)f)x

Intuïtief (gebaseerd op definitie van church getallen): n keer toepassen van f op m keer toepassen van f op x  
=> in totaal n + m keer toepassen van f op x

|  |  |
| --- | --- |
| Bewijs | plus ≡ λn.λm.λf.λx.((n)f)((m)f)x  ((plus) cn) cm |

**Vermenigvuldiging**

|  |  |
| --- | --- |
| Lemma (“x”) | Voor alle n, m ∈ ℕ geldt dat ((cn)f)m y =β (f)nxm y |
| Bewijs | Bewijs per inductie over m  TB: ((cn)f)m y =β (f)nxm y   * 1. Basisgeval m = 0   ((cn)f)0 y ≡ y (def)  ≡ (f)0 y (def)  ≡ (f)nx0 y   * 1. Stel bewezen voor geval m = k (IH), dan nu bewijs voor het geval m = k + 1 |
| Bewijs | times ≡ λn.λm.λf.(n)(m)f |

**Booleaanse expressies**

true ≡ λt.λf.t (geeft eerste argument terug)

false ≡ λt.λf.f (geeft tweede argument terug)

((true) A) B) =β A

((false) A) B) =β B

|  |  |
| --- | --- |
| Bewijs | ((λt.λf.t) A) B) =β (λf. A) B =β A  ((λt.λf.t) A) B) =β (λf.f) B =β B |

**“if” statement**

If ≡ λc.λd.λe.((c)d)e

Staat voor: if c then d else e

c (conditie) moet true of false geven

true => ((c)d)e geeft eerste argument terug, dus d

false => ((c)d)e geeft tweede argument terug, dus e

(((if) true) A) B) =β A

|  |  |
| --- | --- |
| Bewijs | λc.λd.λe.((c) d) e) true) A) B) =β ((λd.λe.((true) d) e) A)B  =β (λe. ((true) A) e) B  =β ((true) A) B  =β A |

# Lambda-calculus: deel III

**“iszero”**

Iszero ≡ λn.((n) λx.false) true

λx.false: geeft steeds false terug

((n) λx.false) true: als n een church getal, dan (betekenis church getal) n keer toepassen van de functie λx.false op het argument true: geeft steeds false

Maar bij n = c0 wordt de functie niet toegepast, maar wordt het argument (true) teruggegeven

Dus:

(iszero)c0 =β true

(iszero) cn =β false for n > 0

|  |  |
| --- | --- |
| Bewijs |  |

**“car en cdr”**

car ≡ λa.λd.a (geeft eerste argument terug)

cdr ≡ λa.λd.d (geeft tweede argument terug)

cons ≡ λa.λd.λz. ((z) a) d

((cons) A) B ≡ ((λa.λd.λz. ((z) a) d) A) B =β λz.((z) A) B)

(((cons) A) B) car =β A

(((cons) A) B) cdr =β B

|  |  |
| --- | --- |
| Bewijs |  |

**Andere numerieke functies als λ-expressies**

**Pred**(n) = n – 1 als n > 0

Idee: paren maken van de vorm (cn, cn-1)of ((cons) cn) cn-1)

Dan pred van cn definiëren als de cdr van het paar (cn, cn-1), nl. ((cons) cn) cn-1) cdr

Hoe creëren we van (cn, cn-1) -> (cn+1, cn)

(nextp)((cons) cn) cn-1 = ((cons) cn+1) cn

**Nextp** ≡ λp.((cons)(succ)(p) car) (p) car

≡ λp.λz. ((z) λn. λf. λx (f) ((n) f) x)(p) car)(p)car

A screenshot of a computer code

Description automatically generated

**Fixpunten**

Voor een numerieke functie f: D ⟶ D heet x ∈ D een fixpunt van f desda f(x) = x

Bv. 0 is een fixpunt voor \*2, nl. 0\*2 = 0

Voor een λ-expressie wordt dit:

|  |  |
| --- | --- |
| Definitie | X ∈ Λ is een fixpunt van F ∈ Λ desda (F)X =β X |
| Stelling | * 1. Iedere λ-expressie heeft een fixpunt: ∀F ∈ Λ ∃X: (F)X =β X   2. Er is een fixpunt combinator   Y ≡ λf. (λx. (f)(x) x) λx. (f)(x) x  Waarvoor geldt dat  ∀F ∈ Λ: (F)(Y) F =β (Y)F  m.a.w. deze Y laat toe om het fixpunt te berekenen voor een willekeurige F, nl. het fixpunt is (Y)F |
| Bewijs |  |

**Recursiviteit**

A screenshot of a computer

Description automatically generatedA close-up of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer code

Description automatically generated