MOTIVATIE
en
SCOTT'S TRALIETHEORETHISCHE BENADERING
voor een
WISKUNDIGE SEMANTIEK
van
PROGRAMMEERTALEN

door

M.M. FOKKINGA
(T.H. DELFT)

28 MEI 1974

Deze monograph is een herziening en uitbreiding van mijn manuscript "Scott's Mathematical Semantics of Comp. Languages" ('72/73) welke resulteerde uit een zeer bescheiden literatuurstudie in december 1972. Een gedeelte van deze monograph is voorgedragen op het 10e Ned. Math. Congres, Twente april 1974, in het kader van een overzicht over Semantiek van Programmeertalen.

Inhoud:

1 MOTIVATIE VOOR SEMANTIEK, VOOR WISKUNDIGE SEMANTIEK

.1 Strachey (zie [12]) onderkent een probleem van schrijfwijzen, notaties. Vergelijkend met bijv. de ontwikkeling van matrix- en vectornotatie kun je stellen dat bij de huidige programmeertalen de hoeveelheid nieuwe notatie en de snelheid waarmee deze wordt ingevoerd bijna te groot is om er een algemeen aanvaarde betekenis bij te laten ontstaan. Voor eenzelfde begrip is vaak meer dan een notatie gangbaar. Maar erger nog, we weten niet precies welke schrijfwijzen voor eenzelfde begrip staan. Daarom is het hoogtijd dat voor de verschillende notaties de betekenis eens wordt vastgelegd, met als mogelijk maar zeker wenselijk gevolg dat er meer eenheid van notatie ontstaat (zoals er voor matrix- en vectornotatie maar een schrijfwijze gangbaar is).

.2 Nieuwe expressies zijn vrijwel altijd gelijktijdig van hun begripsaanduidingen vergezeld gegaan. Soms waren het de begrippen die zich eerder ontwikkelden (elementaire rekenkunde) zonder dat er direkt een schrijfwijze voorhanden was, meestal ging het gelijk op (matrices, . ~~...~~ .) en een enkele keer was de taal er voordat de wiskundige objecten gevonden waren (λ-calculus). Heel redelijk is dus de vraag naar de objecten behorende bij de nieuw ingevoerde expressies. Dat is de vraag naar een semantiek.

.3 Nog krachtiger wordt dit argument als we bedenken dat er soms écht verschillende maar wel intuitief gerechtvaardigde betekenissen gehecht kunnen worden aan taalconstructies. Zo geeft Scott in [13] een amusante opsomming van mogelijke betekenissen (met hun rechtvaardigingen) voor de propositielogica. (zie aanhangsel)

.4 In sommige gevallen is het mogelijk de semantiek te verdoezelen door als model (de normaalvormen van) de expressies zelf te nemen. (Bij de natuurlijke getallen

met als beschrijvende taal de binaire notatie is dit bijvoorbeeld mogelijk. De normaalvormen zijn de binaire strings zonder de insignificante nullen.*))
Er zijn een aantal redenen om dit beslist niet te doen:

a. ook al (of: juist als) alle modellen isomorf zijn (nat. get.) is het niet eerlijk om er één in het bijzonder uit te kiezen en te stellen boven andere.

b. niet altijd is zoiets mogelijk: de reele getallen bijv. laten zich niet in normaalvorm opschrijven.

c. we zoeken juist gemeenschappelijke kenmerken van verschillende maar gelijkende structuren. We willen eenzelfde semantische benadering voor die gehéle klasse van structuren.

.5 Voor eenzelfde object, begrip kunnen in een taal verschillende expressies bestaan. Deze equivalenties te verklaren is een veel te belangrijke taak om aan syntax overgelaten te worden. Want behalve dat het syntactisch niet altijd lukt (zie 4.4), zijn er ook equivalenties van expressies uit verschillende talen. Zonder een semantiek zouden hier equivalentievragen niet eens zin hebben.

Tot dusver zijn er nog geen nadrukken gelegd op het adjectief wiskundig. Dat komt nu.

.6 We willen begripsmatig tot een betekenis komen. Het is niet een formele vertaling in een andere taal die we wensen, maar daarentegen willen we direkt de begrippen hebben die de betekenis zijn.

.7 Met name willen we dat de verschillende kenmerken duidelijk onderscheiden blijven. De semantiek

---

*) Dat dit mogelijk is komt vanwege het slimme gebruik van de nul en de positionele notatie. Deze ontdekking van de taal tast de objecten of hun eigenschappen niet aan.
In programmeertalen zijn waarschijnlijk dergelijke situaties.

moet ons in staat stellen ieder kenmerk afzonderlijk te behandelen. Dit houdt in dat we zeker géén semantiek willen hebben in termen van een _abstracte_ machine. Daar vervagen de kenmerken van de taal tot één geheel met de machine-eigen instructies.

.8 Als zodanig moet het adjectief wiskundig dan ook gezien worden in tegenstelling tot operationeel. Bij een operationele benadering worden allerlei keuzen gedaan (bijv. representatie van de verscheidene datasoorten), die niet essentieel zijn voor een begripsmatig begrijpen van de taal.

.9 Het is juist heel redelijk te vragen de betekenis van nieuwe begrippen te verklaren in termen van oude, reeds bekende. Derhalve moeten operationele kenmerken van de taal niet in een operationele aanpak verklaard worden, aangezien "commando's" nog niet tot het arsenaal van de wiskundige behoren. Alleen funkties zijn een bekend gereedschap.

.10 Dus in het bijzonder betekent een wiskundige semantiek een funktionele benadering. De betekenis wordt bijv. (zie 3.2) aangegeven in termen van toestandstransformaties, i.e. funkties van de toestandsverzameling S naar S.

Het mag niet vreemd klinken om funkties te geven als betekenis: wij denken in het dagelijkse leven als wiskundige voortdurend in termen van funkties. Oppervlakte wordt geassocieerd met de funktie integraal, snelheid met de funktie afgeleide. In het bijzonder zou de hele rekenkunde onmogelijk zijn zonder veelvuldig gebruik van funkties, terwijl het idee van "commando's" (de operationele benadering) vrijwel helemaal ontbreekt.

.11 Een heel belangrijk voordeel van een funktionele of begripsmatige benadering is dat het vaak veel gemakkelijker is daarmee bepaalde eigenschappen van programma's e.d. in te zien én te bewijzen dan met een operationele benadering. Bovendien levert het een kriterium waarmee intuitie over de programmeertaal of programma's beves- tigd of weerlegd kan worden.

.12 Opmerking.

De implementatoren moeten hun -operationele- aanpak toetsen aan de conceptuele semantische definitie.

2 WAT EEN WISK. MODEL IS EN HOE DE WISK. SEM. ERUIT ZIET.

.1 Een model bestaat uit de specifikatie van
- enige basisdomeinen
- enige basisfunkties, -relaties, -constanten en -operatoren

(die alle echt moeten kunnen bestaan).
De uiteindelijke betekenis van een uitdrukking in de programmeertaal moet een element zijn van (constructies uit) de basisdomeinen en -funkties ed. De basisfunkties ed. hebben de rol van primitieve betekenis voor de atomaire uitdrukkingen van de taal. Bij verschillende interpretaties (i.e. evaluatie tot verschillende modellen) kunnen andere basisdomeinen en -funkties ed. gekozen worden.

.2 De toegestane interpretaties van een taal worden vastgelegd door de
- semantische evaluatie funkties

Deze geven, in afhankelijkheid van een te kiezen model, aan iedere uitdrukking in de taal een waarde in het model. Daarbij worden sommige symbolen of uitdrukkingen onafhankelijk van de keuze van het model geïnterpreteerd (zij leggen de kenmerken van de programmeertaal vast), terwijl andere symbolen juist door de keuze van het model hun betekenis krijgen.

Bijvoorbeeld, de symbolen <u>and</u>, <u>not</u>, = en de constructie ....;.... (in ALGOL) hebben een vaste interpretatie, nl. de conjugatie, de negatie, de gelijkheidsrelatie en de sequentiële compositie. [merk op hoe verschillend deze symbolen in hun betekenis zijn]. De betekenis van de letters OUT(..) is in de meeste programmeertalen een of ander nader te bepalen (uitvoer) funktie.

Met betrekking tot de semantische evaluatiefunkties kunnen we hetvolgende stellen.

.3 Toegespitst op het adjectief wiskundig eisen we een precieze wiskundige begripsvorming (notatie), waartoe bijvoorbeeld het aangeven van het "logische type" van funkties behoort. Een funktie heet van type

$S \times D \to [D' \to [S \to S]]$

te zijn, als hij aan een paar van elten van S en D een funktie toevoegt die aan een elt van D' een toestands-transformatie toevoegt. Analoog voor andere typen.
We eisen ook dat operaties op (constructies van) de basis-domeinen, zoals paringsoperatoren en hun inversen, door bijv. funktievergelijkingen krachtig worden vastgelegd.
En het zo voor de semantische evaluatiefunkties zelf.

.4 Uit de stellingname dat we de afzonderlijke kenmerken apart moeten kunnen bediscussieren (zie 1.7) volgt dat de sem. evaluatie syntax gericht moet zijn. De betekenis moet gevonden worden, is in te zien, via de clausules in de syntactische definitie van de beschouwde uitdrukking.

.5 Dat we de uiteindelijke betekenis willen hebben in termen van toestandtransformaties (zie 1.10) wordt als volgt gemotiveerd. Een uitdrukking heeft in het algemeen niet een unieke betekenis, maar zijn waarde zal afhangen van de toestand van het systeem (de computer) op het moment van (het begin van) de evaluatie.
Een nadere precisiering is ook het onderkennen van de afhankelijkheid (van de betekenis van een uitdrukking) van de omgeving. Omgevingen zijn associaties van toestandtransformaties aan namen voor de commando's, ofwel om het wiskundig te formuleren, een omgeving ρ is een funktie $\rho : NM \to [S \to S]$.
Deze precisieringen hangen sterk af van de syntactische categorie waarvan de uitdrukkingen worden beschouwd.
Een voorbeeld is uitgewerkt in het volgende hoofdstuk.

.6 Strachey begon dit project al in '64 ([12]). Hij gebruikte een recursief systeem van syntax-gerichte funktievergelijkingen. Maar omdat hij nogal wezenlijk gebruik maakte van het type-vrij karakter van de λ-calculus (in termen waarvan hij de semantiek wilde definieren) en omdat wij niet louter een vertaling in een andere taal willen (in casu de λ-calculus), valt te verwachten dat we een wiskundig systeem moeten vinden dat ook als model van de λ-calculus kan optreden.

.7 Nog duidelijker wordt dit wanneer men bedenkt dat in hogere programmeertalen procedures ook procedures en met name zichzelf als argument kunnen nemen. Een model dat een oplossing biedt voor de problemen die rijzen bij zelf-applicatie is vrijwel een λ-calculus model.

3 EEN VOORBEELD

Om u enig idee te geven van de voorgestelde aanpak, geven we voor een eenvoudige programmeertaal een zeer summiere uitwerking. De syntax is zo eenvoudig mogelijk gehouden om ons helemaal op de semantiek te kunnen concentreren.

.1 Twee syntactische klassen worden mbv. een context-vrije grammatica (BNF) als volgt gedefinieerd:

(A) CMD : $\gamma ::= (\gamma) \mid \psi \mid \underline{\text{dummy}} \mid \varepsilon \rightarrow \gamma_1 , \gamma_2 \mid \gamma_1 ; \gamma_2$

(B) EXP : $\varepsilon ::= (\varepsilon) \mid \pi \mid \underline{t} \mid \underline{f} \mid \varepsilon_0 \rightarrow \varepsilon_1 , \varepsilon_2$

Zij staan voor Commando's en test-Expressies.

.2 Als basisdomeinen voor ons model hebben we nodig

- S : de Toestandsverzameling
- T : de verzameling waarheidswaarden; $T \ni$ true, false

De basisfunkties, -relaties en -operaties hierop geven we aan wanneer we ze tegenkomen.

De semantische evaluatiefunkties voor ieder van de syntactische klasen hebben het volgende logische type

(I) $\mathcal{C}$: CMD $\rightarrow [S \rightarrow S]$

(II) $\mathcal{E}$: EXP $\rightarrow [S \rightarrow T \times S]$,

zodat $\mathcal{C}$, gegeven een commando en gegeven een toestand, een nieuwe toestand afgeeft, en $\mathcal{E}$ gegeven een expressie en gegeven een toestand zowel een waarheidswaarde alsook een nieuwe toestand (zij effect) afgeeft. Zij worden met inductie gedefinieerd (en hierbij worden syntactische uitdrukkingen voor het gemak steeds met de haakjes $[\![$ en $]\!]$ omsloten):

(II1) $\mathcal{E}[\![(\varepsilon)]\!] = \mathcal{E}[\![\varepsilon]\!]$

(II2) $\mathcal{E}[\![\pi]\!] =$ nog vrij te kiezen elt van $[S \rightarrow T \times S]$

(II3) $\mathcal{E}[\![\underline{t}]\!] = \mathbb{P}(\text{true})$,

waarbij $\mathbb{P}$ de paringsoperator $\in [T \rightarrow [S \rightarrow T \times S]]$ i

(II4) $\mathcal{E}[\![\varepsilon_0 \rightarrow \varepsilon_1 , \varepsilon_2]\!] = \text{Cond}(\mathcal{E}[\![\varepsilon_1]\!], \mathcal{E}[\![\varepsilon_2]\!]) * \mathcal{E}[\![\varepsilon_0]\!]$

waarbij Cond : $[S \rightarrow T \times S] \times [S \rightarrow T \times S] \rightarrow [T \rightarrow [S \rightarrow T \times$

en $V^{\neq}$, afhankelijk van de door de $*$-operator afgeleverde T component van $\mathcal{E}[\![\varepsilon_0]\!]$, zijn eerste

V^{F}:, na evaluatie van $\mathcal{E}[\![\varepsilon_0]\!]$ op de toestand dan wel zijn tweede argument laat werken op de vervolgens door de $*$-operator afgeleverde S-component van $\mathcal{E}[\![\varepsilon_0]\!]$.

en voor de commando's definieren we de betekenis als volgt:

(I1) $\mathcal{C}[\![(\gamma)]\!] = \mathcal{C}[\![\gamma]\!]$

(I2) $\mathcal{C}[\![\ \psi\ ]\!] =$ nog vrij te kiezen elt van $[S \to S]$

(I3) $\mathcal{C}[\![\underline{\text{dummy}}]\!] = I$, de identiteit

(I4) $\mathcal{C}[\![\gamma_1 ; \gamma_2]\!] = \mathcal{C}[\![\gamma_2]\!] \circ \mathcal{C}[\![\gamma_1]\!]$, waarbij $\circ$ de compositie op. is

(I5) $\mathcal{C}[\![\varepsilon \to \gamma_1, \gamma_2]\!] = Cond'(\mathcal{C}[\![\gamma_1]\!], \mathcal{C}[\![\gamma_2]\!]) *' \mathcal{E}[\![\varepsilon_0]\!]$

waarbij $Cond' : [S \to S] \times [S \to S] \to [T \to [S \to S]]$, en, na evaluatie van $\mathcal{E}[\![\varepsilon_0]\!]$ op de toestand, afhankelijk van de door de $*$-operator afgeleverde T-component van $\mathcal{E}[\![\varepsilon_0]\!]$, zijn eerste dan wel zijn tweede argument laat werken op de vervolgens door de $*'$-operator afgeleverde S-component van $\mathcal{E}[\![\varepsilon_0]\!]$.

.3 We introduceren nu commando namen ξ met een syntactische klasse NM die we niet verder zullen specificeren. Namen hebben een tijdelijke betekenis, gegeven door de omgeving ρ waarin ze geëvalueerd worden. We gebruiken OMG als afkorting van $[NM \to [S \to S]]$, en laten ρ varieren over de OMG. Passen we het bovenstaande aan dan krijgen we:

(A)' CMD : $\gamma ::= \cdots\cdots\cdots\cdots \mid \xi$

(B)' EXP : $\varepsilon ::= \cdots\cdots\cdots$

(I)' $\mathcal{C}$: CMD $\to [OMG \to [S \to S]]$

(II)'' $\mathcal{E}$: EXP $\to [S \to T \times S]$

en voor de definities van $\mathcal{E}$ en $\mathcal{C}$

(II1/4)' als (II1/4)

(I1/5)' te verkrijgen uit (I1/5) door overal $\ldots\mathcal{C}[\![\ldots]\!]\ldots$ te vervangen door $\ldots\mathcal{C}[\![\ldots]\!](\rho)\ldots$

(I6)' $\mathcal{C}[\![\xi]\!](\rho) = \rho(\xi)$

Door de aanwezigheid van namen kunnen we nu ook recursieve commando's invoeren: breidt CMD nogmaals uit:

(A)' CMD : $\gamma ::= \ldots\ldots\ldots \mid \langle \xi_1, \xi_2, \ldots \xi_n : \gamma_1, \gamma_2, \ldots \gamma_n \rangle$

waarbij met de laatste clausule hetvolgende bedoeld wordt. De commando-namen $\xi_1, \ldots, \xi_n$ moeten als namen van de commando's $\gamma_1, \ldots, \gamma_n$ gezien worden. In de $\gamma_1, \ldots, \gamma_n$ kunnen de namen $\xi_1, \ldots, \xi_n$ en andere weer voorkomen zodat het geheel een recursief systeem is.

We kunnen aangeven dat de ξ_i als namen van de γ_i gezien moeten worden door de omgeving ρ aan te passen tot $\rho[\gamma^n/\xi^n]$ i.e. "omgeving ρ met γ_i voor ξ_i", met de definitie dat

$$\rho[\gamma^n/\xi^n](\xi_i) = \mathcal{C}[\![\gamma_i]\!](\rho[\gamma^n/\xi^n])$$

Stellen we eens $n=1$, dan wensen we zeker dat

(II7)* $\mathcal{C}[\![\langle \xi : \gamma \rangle]\!](\rho) = \mathcal{C}[\![\gamma]\!](\rho[\gamma/\xi])$

maar neemt u γ van de vorm $\varepsilon_0 \to \psi ; \xi$, dummy dan blijkt na vier keer uitschrijven van $\mathcal{C}[\![\gamma]\!](\rho[\gamma/\xi])$ de term $\mathcal{C}[\![\gamma]\!](\rho[\gamma/\xi])$ weer terug te komen. De vraag rijst of er überhaupt wel iets gedefinieerd wordt: wanneer alle funkties die we tegenkomen totaal moeten zijn volgt er hier een inconsistentie!

Na lezing van § 4.2 zal blijken dat we de betekenis ook als volgt kunnen definieren:

(II7)' $\mathcal{C}[\![\langle \xi^n : \gamma^n \rangle]\!](\rho) = \Pi_1^n(Y(\lambda\Theta^n.\ \mathcal{C}[\![\gamma^n]\!](\rho[\Theta^n/\xi^n])))$,

als volgt te lezen:

de toestandstransformatie toegekend aan het commando $\langle \xi^n : \gamma^n \rangle$ is de eerste component (Π_1^n) van de kleinste dekpunt (Y) van het n-tal vergelijkingen $\Theta_i = \mathcal{C}[\![\gamma_i]\!](\rho[\Theta^n/\xi^n])$ de welke de gebonden toestandstransformaties Θ_i gelijkstellen aan de betekenis van de γ_i in de zó gewijzigde oorspronkelijke omgeving ρ dat de ξ_j de betekenis Θ_j hebben.

.4 Behalve in het laatste geval hebben we ons nog niet bezig gehouden of de wiskundige objecten die we definieerden wel bestaan, consistent zijn. Om precies te zijn zouden we moeten aantonen dat mét de ruimtes T en S óók de ruimtes $[S \to S]$ ("funktieruimte(?)")

en $T \times S$ (Cartesisch produkt), $[S \rightarrow T \times S]$, $[ENV \rightarrow [S \rightarrow S]]$ ... ,etcetera, alle bestaan. En óók dat de operatoren P, Cond, * en • etcetera alle bestaan en in het bijzonder de operator Y. Bovendien zou zelfs het bestaan van een ruimte $[CMD \rightarrow [ENV \rightarrow [S \rightarrow S]]]$ en het element $\mathcal{C}$ ervan, en de ruimte $[EXP \rightarrow [S \rightarrow T \times S]]$ en het element $\mathcal{E}$ ervan, aangetoond moeten worden.

.5 Opgemerkt kan worden dat met het bestaan van ruimtes R_1 en R_2 ook het bestaan van de somruimte $R_1 + R_2$, ~~de~~ de productruimte $R_1 \times R_2$ en de funktieruimte $R_2^{R_1}$ (soms aangeduid met $R_1 \rightarrow R_2$) gegarandeerd is volgens welbekende wiskunde. Ook het bestaan van de paringsoperator en dergelijke is niet moeilijk te verifieren. Moeilijker wordt het voor de kleinstedekpunt operator Y. Grote moeilijkheden krijgen we als we onze taal nog verder uitbreiden met procedures en dán consistente modellen willen bouwen.

Stel dat we de taal hebben uitgebreid met procedures. Stel dat we de betekenis van procedures geven als objecten van een of andere ruimte P. We willen dat zo'n object van P, gegeven een waarde als argument en gegeven een toestand zowel een nieuwe waarde als ook een nieuwe toestand geeft:

(a) $P = [W \rightarrow [S \rightarrow [W \times S]]]$

Het domein W der waarden moet naast, laten we zeggen, integerwaarden, N, en de waarheidswaarden, T, ook commando's, $[S \rightarrow S]$, en – zo willen we en zo is het toch bij programmeertalen van enig niveau – zelfs procedures, P, bevatten. Dus

(b) $W = N + T + [S \rightarrow S] + P$

En hier stuiten we op het probleem dat ook al in 2.7 is genoemd: laten we de ruimten, genoteerd met $[R \rightarrow R']$, bestaan uit de verzameling van <u>alle</u> funkties van R naar R', dan leveren (a) en (b) een tegenspraak op: volgens (a) is de cardinaliteit van P echt groter dan die van W, en volgens (b) is de cardinaliteit van W groter dan of gelijk aan (gelijkheid in geval van aftelbare oneindig-heid) die van P!

In het volgende hoofdstuk wordt een oplossing voor deze problemen geboden.

4 WAAROM WE TERECHT KOMEN BY COMPLETE CONTINUE TRALIES MET BEREKENBARE BASES

De redenen hiervoor zijn velerlei. Ik som ze achter-eenvolgens op om ze daarna afzonderlijk toe te lichten.

1. Intuitief zien we in dat datatypen dergelijke tralies zijn!

2. Ze verschaffen ons de wiskunde om problemen op te lossen die rijzen bij het bouwen van een model voor de semantiek.

3. In termen van tralies is een unifikatie mogelijk.

4. Een verregaande unifikatie is mogelijk via een bijzonder compleet continue tralie met berekenbare basis.

.1 Intuitief zien we in dat datatypen dergelijke tralies zijn.

Een toelichting hierop staat erg duidelijk in ([6]). Ik wil heat hier slechts kort herhalen en ik wil vooral de zeer natuurlijke verschijning van de complete continue tralies benadrukken en er op wijzen dat het allerminst gekunstelde technische constructies zijn, zoals het in de literatuur nogal eens op mij afkomt, die zomaar uit de lucht gegrepen zijn en waarmee alles ineens op te lossen schijnt te zijn.

De grondgedachte is dat we ons bezighouden met berekenbare zaken.

Een datatype D is de verzameling van tijdens berekeningsprocessen optredende objecten van een datasoort, dus een datatype is de verzameling van de wiskundige objecten die tijdens een berekeningsproces optreden en de betekenis zijn van datasoorten zoals integers, lists, procedures en graphs etcetera.

Duidelijk is dat tijdens een berekeningsproces een element

van de datatype al enigszins, maar nog niet volkomen gespecificeerd kan zijn. Bij de gehele getallen ligt dit minder voor de hand, die zijn er of die zijn er niet, maar bij recursieve procedures bijvoorbeeld zijn de objecten toegekend aan de tot zekere recursiediepte geevalueerde procedures slechts benaderingen van het eigenlijk bedoelde object. Dus voor verschillende $x, y \in D$ kan x een benadering zijn van y, aangeduid door $x \sqsubseteq y$, ook te lezen als (de informatie) y is consistent met (de informatie) x, y is beter of meer gedefinieerd dan x, etc. Dit benaderingsbegrip bestaat intuitief op iedere datatype (maar is soms triviaal, zoals bij de gehele getallen: geen getal is een benadering van, is consistent met, is beter gedefinieerd dan een ander getal). Daarom kunnen we dit als axioma postuleren. Bovendien zien we intuitief in dat die relatie een partiele ordening is:
<u>axioma 1</u>: datatypen zijn (door $\sqsubseteq$) partieel geordende verz^n^.

Uit het oogpunt van berekenbaarheid onderkennen we dat de afbeeldingen, i.e. berekeningsprocessen, die we beschouwen tussen datatypen, op betere informatie over het argument minstens even goede informatie omtrent de waarde moeten geven. Dit kan gepostuleerd worden als
<u>axioma 2</u>: berekenbare afbeeldingen tussen datatypen zijn monotoon (mbt $\sqsubseteq$).

Een verdere bewustwording van het wezen der datatypen doet ons inzien dat er voor iedere ketting $x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq \ldots \sqsubseteq x_n \sqsubseteq \ldots$ van steeds beter gedefinieerde objecten een limiet y bestaat, dat is een element y dat precies de gezamelijke informatie bevat van de $\{x_i\}$, genoteerd met $y = \bigsqcup\{x_i\}$. Voor wiskundige algemeenheid, maar heus niet in strijd met onze intuitie, nemen we aan dat voor iedere deelverzameling X van D

zo'n "kleinste bovengrens" (m.b.t. $\sqsubseteq$) $\bigsqcup X$ bestaat, die dus precies de gezamelijke informatie van de verzameling X bevat. Dit impliceert het bestaan van een element $\top = \bigsqcup D$ met tegenstrijdige informatie, ook wel het overgedefinieerde element genoemd, en het bestaan van de grootste ondergrens $\sqcap X$ (m.b.t. $\sqsubseteq$) voor iedere deelverzameling X van D, die precies de gemeenschappelijke informatie van de elten van X bevat, nl. $\sqcap X = \bigsqcup \{y : y \sqsubseteq x \text{ voor alle } x \in X\}$, en het bestaan van $\perp = \sqcap D$, het ongedefinieerde, i.e. ongespecificeerde element met loze, nietszeggende ofwel géén informatie.
Deze intuitie postuleren we als
axioma 3: datatypen zijn complete tralies.

Nader gevolg van de berekenbaarheid der afbeeldingen, i.e berekeningsprocessen, is dat voor eindige hoeveelheid informatie omtrent de waarde er ook slechts eindig veel informatie omtrent het argument nodig behoeft te zijn. Preciezer gezegd: zij x de limiet van $x_0 \sqsubseteq x_1 \sqsubseteq \dots \sqsubseteq x_n \sqsubseteq \dots$, dan geldt: als $\bigsqcup_{(i\in I)} f(x_i) \neq x$ dan I is eindig, en in contrapositie luidt dit: als I oneindig dan $\bigsqcup_{(i\in I)} f(x_i) = x$, ofwel $\bigsqcup_{i=0}^{\infty} f(x_i) = x$, hetgeen we noemen
axioma 4: berekenbare afbeeldingen tussen datatypen zijn continu.

Gemotiveerd door de fysische realiseerbaarheid (i.e. berekenbaarheid) van de objecten, luidt onze gedachtegang over datatypen dat ieder elt voor te stellen moet zijn als een benadering van "eindige" elten. Dat wil zeggen dat iedere x limiet is van "zijn eindige benaderingen". Dit noemen we continuiteit van tralies:
axioma 5: datatypen zijn continue tralies.
Voor het intuitieve begrip "eindige benadering van" kunnen we een formele definitie geven omdat continuiteit

van funkties een topologie induceert en in de topologie is een eindigheidsbegrip bekend. Het blijkt bovendien helemaal in overeenstemming met ons intuïtief begrip, wanneer we zeggen dat van een totale funktie $f: \mathbb{N} \to \mathbb{N}$ de restricties tot $[0, n]$ (en voor (n, ∞) dus "ongedefinieerd" zijnde) de "eindige benaderingen van f" zijn, en inderdaad f de limiet ervan is.

Uit het oogpunt van berekenbaarheid zegt onze intuïtie ook nog dat er een berekenbare basis is, via welke ieder element te benaderen is:
axioma 6: datatypen zijn continue tralies met berekenbare bases.
De berekenbaarheid van een basis E houdt in dat alle elementen ervan effectief zijn op te sommen en dat de relaties "is eindige benadering van", $e_1 \prec e_2$, en "is benadering van", $e_1 \sqsubseteq e_2$, voor E beslisbaar zijn.
Van een continue tralie met berekenbare basis E noemen we een elt x berekenbaar als er een effectieve opsomming van elten uit E is, waarvan x de limiet is.

E.H.B.O: Een Heel Belangrijke Opmerking
Zoals al in (3·4) is gesteld willen we bepaalde constructies van datatypen, zoals $T \times S$, $S \to S$, $[S \to T \times S]$ etcetera, ook weer als datatype beschouwen. Welnu, dat kan, want als domeinen D en D' aan ax. 1-6 voldoen, dan voldoen (met geschikte definities) het cartesisch product $D \times D'$, de disjuncte vereniging $D + D'$ en de ruimte der continue afbeeldingen $[D \to D']$ (die dus alle berekenbare funkties : $D \to D'$ bevat) ook aan ax. 1-6.
Hiermee zijn alle bouwsels die in 3·4 genoemd zijn gerechtvaardigd, maar is er nog niets gezegd over het grote probleem van 3·5.

.2 Ze verschaffen ons de wiskunde om problemen op te lossen die rijzen bij het bouwen van een model.

Recursie.

Voor een domein D en een afbeelding $f: D \to D$ die aan ax. 1-3 voldoen kan het bestaan van een dekpunt x_0 bewezen worden, dwz. x_0 voldoet aan $x_0 = f(x_0)$, en bovendien blijkt die x_0 de kleinste te zijn van de dekpunten, dus uniek. Daarom is het zinvol een dergelijke kleinste dekpunt te beschouwen als gedefinieerd door de recursieve vergelijking $x = f(x)$. (In geval D bestaat uit functies en f dus een transformatie is van funkties, kan $X = f(X)$ gelezen worden als de recursieve definitie van een funktie X uit D met "naam x en body $f(x)$"!)

Maar zonder ax. 4 kan (het bestaan van) de kleinste dekpunt niet constructief aangegeven worden, mét ax. 4 kan dat wel en dan blijkt de kleinste dekpunt zelfs gelijk te zijn aan de limiet van de ketting $x_0 \sqsubseteq x_1 \sqsubseteq \dots \sqsubseteq x_n \sqsubseteq \dots$ gevonden door successievelijke benaderingen $x_i = f(x_{i-1})$ uitgaande van het object $x_0 = \perp$ zonder informatie. (Dus precies de wijze waarop we bij een recursieve definitie als $x = f(x)$ de x berekenen!) Dus de kleinste dekpunt wordt uniform gevonden, zeg d.m.v. de dekpuntsoperator Y: $x_0 = Y(f)$.

Ruimteconstructies, zelfapplicatie.

Laat D een continu tralie zijn, en zet $D_0 = D$, $D_{n+1} = [D_n \to D_n]$. Deze zijn alle continu en in elkaar omvat te denken: dit en het volgende is voor $D = T_0 = t\overset{T}{\underset{\perp}{\diamond}}f$ in 4.4 verder uitgewerkt. Verder is met D_∞ = de completering van $\bigcup_{n=1}^{\infty} D_n$ tot een compleet, zelfs continu, tralie, een identificatie

$$D_\infty = [D_\infty \to D_\infty]$$

mogelijk. Dus de ruimte is zijn eigen "funktieruimte": een model voor de λ-calculus! Het probleem van 3.5, het cardinaliteitsconflict ten gevolge van zelfapplicatie, is opgelost! Door de beperking tot continue afbeeldingen

als elementen van $[D \to D']$ is dit mogelijk gemaakt, en aangezien berekenbare afbeeldingen continu zijn hebben we ook niet meer nodig!
Ook andere limietconstructies zoals $D_\infty = D_0 + D_\infty \times D_\infty$, een ruimte van ~~eindige rijtjes~~ lijsten van aftelbare lengte, zijn mogelijk.

De semantische evaluatiefunktie.
Zoals in ~~hoofdstuk~~ 3 is geschetst wordt een sem. ev. funktie $\mathcal{E}$: SYNT. KLASSE $\to [S \to S]$ induktief, en volgens (II 7)* in 3.5 zelfs recursief naar de verschillende syntactische clausules gedefinieerd. Het bestaan van zo'n recursief gedefinieerde $\mathcal{E}$ kan net als hierboven via de dekpuntmethode bewezen worden, mits het domein SYNT. KLASSE een compleet tralie is. Dat laatste is op grond van de argumentatie in 4.1 duidelijk, en wordt hieronder nogmaals genoemd.

.3 In termen van tralies is een unifikatie mogelijk.

De dekpuntsoperator Y.
Hierboven hebben we de dekpuntsoperator Y gedefinieerd als de uniforme manier waarop de kleinste dekpunt van een funktie $\in [D \to D]$ gevonden kan worden. De operator Y blijkt zelf continu te zijn, dus $\in [[D \to D] \to D]$! Daarom zijn we vrij hem toe te passen zonder dat we gevaar lopen objecten te definieren buiten de tralies. Dus met gebruik van Y passen ook vele andere operatoren en bewerkingen ~~op~~ gerechtvaardigd in de theorie.

Syntactische klassen.
Ook syntactische klassen – of wiskundige objecten die ermee te identificeren zijn – zijn als continue tralies te definieren. Voor het bekende voorbeeld van stroomdiagrammen kunnen we bijvoorbeeld uitgaan van basisdomeinen F, de elementaire funkties, waaronder de identiteit I,

en B, de keuzefunkties. Dan definieren we $E_0 = F$ en $E_{n+1} = E_0 + (E_n ; E_n) + (B \to E_n, E_n)$ en E als de limietcompletering. Nu is te bewijzen: ieder object van E is ofwel een elementaire funktie uit F, ofwel een sequentie van objecten uit E, ofwel een test op een keuzefunktie van B met alternatieven uit E. Zie ([7]).

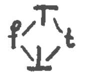
En verder.

En verder zijn allerlei discrete basisverzamelingen direkt als continue tralies op te vatten:  , geheel in overeenstemming met de intuitieve betekenis van de benaderingsrelatie $\sqsubseteq$.

En zoals hierboven voor de semantische evaluatie funktie is geillustreerd kunnen in de tralietheorie allerlei gelijksoortige problemen op uniforme wijze worden opgelost.

.4 Een verregaande unificatie is mogelijk via een bijzonder continu tralie met berekenbare basis.

Alvorens hier op in te gaan eerst de constructie van zo'n tralie. We kiezen de limietconstructie voor de "logische ruimte" T_∞.

Zij $T_0 =$  en $T_{n+1} = [T_n \to T_n]$. Inbeddingen van T_n in T_{n+1} zijn mogelijk door de continue i_n, j_n, $i_n \in [T_n \to T_{n+1}]$ en $j_n \in [T_{n+1} \to T_n]$ gedefinieerd door
$i_0 x = \lambda y \in T_0 .\, x$ en $j_0 = \lambda f \in T_1 .\, f(\bot)$ en
$i_{n+1} = \lambda x \in T_n .\, i_{n-1} \circ x \circ j_{n-1}$ en $j_{n+1} = \lambda f \in T_{n+1} .\, j_{n-1} \circ f \circ i_{n-1}$.
Dan is $i_n(T_n) \subset T_{n+1}$ en bovendien $i_n \circ j_n x \sqsubseteq x$, $j_n \circ i_n x = x$.

We kunnen nu alle T_n als deelruimtes $\overline{T}_n$ van een rijtjesruimte opvatten, door $x \in T_n$ te identificeren met
$\bar{x} = \langle \ldots, j_{n-2} j_{n-1} x, j_{n-1} x, x, i_n x, i_{n+1} i_n x, \ldots \rangle$ (x op de i-de pl.)
en funktie applicatie coordinaatsgewijze te nemen (steeds $i+1$-de op de i-de coordinaat toepassen). Nu geldt $\overline{T}_n \subset \overline{T}_{n+1}$.

De limietruimte $\overline{T}_\infty$ bestaat uit de rijtjes $\langle x_n \rangle_{n=0}^{\infty}$

met voor alle n : $x_n \in T_n$ en $x_n = j_n x_{n+1}$; het is de completering van $\bigcup_{n=0}^{\infty} \overline{T}_n$ tot een volledig tralie die continu blijkt te zijn en $\bigcup_{n=0}^{\infty} \overline{T}_n$ als berekenbare basis heeft.

Er geldt $\overline{T}_\infty = [\overline{T}_\infty \to \overline{T}_\infty]$:
ieder elt van $\overline{T}_\infty$ is met genoemde definitie van applicatie als elt van $[\overline{T}_\infty \to \overline{T}_\infty]$ op te vatten. Bovendien is ook iedere continue $f : \overline{T}_\infty \to \overline{T}_\infty$ wegens de continuiteit in alle coordinaten, als elt van $\overline{T}_\infty$ op te vatten.

Neem $\overline{T}_\infty$ als T_∞.

Ruimteconstructies.

Een retraction is een continue funktie A (zeg van T_∞ in T_∞) met $A \circ A = A$. Een retract, dat is het waardengebied van een retraction A, vormt een continu tralie, die we ook maar A zullen noemen.
Er zijn continue funkties $+$, $\times$, $\to$ van T_∞ $(=[T_\infty \to T_\infty]\,!)$ naar T_∞ $(=[T_\infty \to T_\infty]\,!)$ die sterke gelijkenissen vertonen met de operatoren $+$, $\times$, $\to$ die van tralies weer tralies maken: de funkties $+, \times, \to$ maken van retracts weer retracts, óók in dekpuntsdefinities, die inderdaad als disjuncte vereniging, cartesisch product en functieruimte (der continue afbeeldingen) te beschouwen zijn.

Op deze manier zijn allerlei domeinen en constructies daaruit als retracts binnen T_∞ te verkrijgen en kunnen we met recursieve definities domeinen definieren! Bijvoorbeeld, het tralie der natuurlijke getallen $\mathbb{N}$ = 
wordt als volgt verkregen: zij $\bar{o}$ een retract van T_∞, definieer dan $\hat{N}$ door $\hat{N} = \bar{o} + \hat{N}$ (dit is een afkorting voor de dekpuntsdefinitie $\hat{N} = Y(\lambda f.\ \bar{o} + f)$). Nu heeft $\hat{N}$ op enige details na de gewenste vorm.

De methode om nieuwe datatypen te creeren en de methode om in datatypen nieuwe elementen te definieren zijn dus uniform!!

De taal LAMBDA en het tralie.

Er is een λ/combinator-achtige taal die elementen van het tralie beschrijft. Alle berekenbare elementen van T_∞ zijn met een LAMBDA-uitdrukking te beschrijven en alle door een LAMBDA-uitdrukking beschreven objecten van T_∞ zijn berekenbaar.

Een formele semantiek, precies zoals we die propageren ~~voor~~ voor programmeertalen, is gegeven voor de taal LAMBDA; deze kent dus aan uitdrukkingen van de taal elementen toe in het tralie. Equivalentie van uitdrukkingen wordt dan gedefinieerd als hetzelfde betekenend in het tralie. Deze equivalentie is een niet recursief opsombare relatie en kan dus niet via een formeel systeem, i.e. recursief opsombaar, gekarakteriseerd worden. Vgl. 1.5. Wel kunnen véél equivalenties formeel afgeleid worden. (Scott stelt daartoe een Gentzen-achtige calculus van sequenten voor, met axioma's en afleidingsregels voor gelijkheid, substitutie, de part. ord. $\sqsubseteq$, de voorwaardelijke expressie ..⊃..,.. , de applicatie en abstractie, en een inductieregel voor de dekpuntoperator. Er zijn al vele varianten in onderzoek.)

Machtige bewijsmogelijkheid.

Het feit dat in de limietconstructies ieder element limiet is van zijn projecties op de (eindige) tralies verschaft de mogelijkheid eigenschappen te bewijzen inductief langs die rij benaderende projecties . Zo blijkt in [11] dat in T_∞, een λ-calculus model, verscheidene gelijkheden van elementen berekend, dus bewezen, kunnen worden, zónder dat dat in de λ-calculus mogelijk is voor de bijbehorende λ-calculus uitdrukkingen. Formele regels kunnen dus veel minder equivalenties verklaren dan berekening in een model !! Vgl. 1.5. Dit is trouwens niet verwonderlijk gezien Gödels onvolledigheidsstelling voor formele systemen.

Opmerking.

Wanneer men de limietconstructies nogal gekunsteld vindt, kan men tegenwerpen dat de limietconstructies sterke analogieen hebben met de constructie van de reele getallen. En niemand kan de constructie der reele getallen toch gekunsteld vinden?

Er zijn nog meer aanwijzingen dat continue tralies niet zo zinloze ruimten zijn. Jedere T_0-ruimte (topologische ruimten waarin de punten eenduidig bepaald worden door de klasse van hun omgevingen, dus (?) veel voorkomende wiskundige ruimten) kan ingebed worden in zo'n tralie (dat zelfs als retract van T_∞ te verkrijgen is). En ze hebben nog meer belangrijke topologische eigenschappen.

5 OPMERKINGEN

Zeer onvolledig blijvend wil ik toch nog de volgende opmerkingen maken.

.1 Behalve de geschetste limietconstructie T_∞, zijn er ook andere λ-calculus modellen, continue tralies, gevonden. Een ervan is het zeer elegante tralie der deelverzamelingen der natuurlijke getallen, met een geschikte definitie van functie-applicatie (wat is het beeld wanneer een deelverzameling van $\mathbb{N}$ op een andere deelverzameling wordt "toegepast" ?). Overeenkomstig zijn er al verscheidene vormen van een taal LAMBDA i $(i = 1, 2, \ldots)$ in onderzoek.

.2 Om technische redenen is het niet nodig te werken met complete continue tralies, maar voldoen de zg. semi-tralies ook of schijnbaar zelfs beter. In semi-tralies ontbreekt het overgedefinieerde element $\top$. De discussie over de voordelen en nadelen van tralies ~~t.o.~~ ten opzichte van semitralies is nog lang niet afgerond.

.3 Op de in hoofdstuk 3 geschetste wijze is voor een echte programmeertaal, PURE LISP, de semantiek gedefinieerd en gelijkwaardig bewezen met de operationele semantiek. Hierbij werd gewerkt met semi-tralies i.p.v. tralies. Ook voor PAL, ALGOL 60, ALGOL 68 zijn volledige beschrijvingen gegeven.

.4 Een andere mogelijkheid voor de semantiek dan het toekennen van funkties aan programmateksten is het toekennen van processen als betekenis. Een proces geeft op een argument, net als een funktie, een waarde af maar bovendien een nieuw proces dat als voortzetting gezien moet worden. Deze benadering biedt betere mogelijkheden om parallellisme te behandelen en om aan niet-terminerende maar wel zinvolle programma's (zij effecten) een betekenis toe te kennen.

.5 Interessante ontwikkelingen doen zich voor bij pogingen om de bestaande theorie voor "polymorfe" funkties uit te breiden. Polymorfe funkties accepteren een (data-) type als argument en geven dan een funktie van (constructies uit) dat datatype af. Bijvoorbeeld wanneer u in een programmeertaal die een volledige parameterspecificatie voorschrijft, ALGOL 68, een verwisselingsprocedure wilt hebben, dan moet u die voor ieder type, zoals integers e.a., apart declareren. Een (in ALGOL 68 niet mogelijke) procedure die een type als argument zou nemen en een verwisselingsprocedure voor dat type zou afgeven, is dan een polymorfe procedure.

.6 Er zijn ook onderzoekingen gaande om zoveel mogelijk van de abstracte wiskundige modelbouw syntactisch te verwerken. Wanneer u bijv. een syntactisch symbool Ω introduceert staande voor de "ongespecificeerde syntactische vorm" en de relatie "is beter gespecificeerd dan" tot uitdrukking brengt met een partiele ordening $\leq$ voor syntactische vormen (vgl. 4.1 en 4.2), dan is het mogelijk voor eenvoudige recursieve programmaschema's de betekenis te definieren die gerechtvaardigd is op grond van syntactisch gemanipuleer in plaats van abstracte wiskundige modelbouw.

6. VERANTWOORDING

Hoofdstuk 1 t/m 4 zijn volledig geput uit [6]–[13]. Met name is hfdst. 1 gehaald uit al die verwijzingen, hfdst. 2 voornamelijk uit [8] en [10], hfdst. 3 uit [8], hfdst. 4 uit [6][11][7].

Hoofdstuk 5 geeft recentelijke ontwikkelingen aan. De achtereenvolgende paragrafen zijn gebaseerd op [4][1][1][2][5][3].

VERWYZINGEN

[1] Gordon, M.J.C., Models of Pure LISP (a worked example in semantics). Exp. Progr. Rep. 31, Dept. Mach. Intell. Univ. Edinburgh 1974.

[2] Milner, R., An approach to the semantics of parallel programs. Edinburgh Techn. Memo, Univ. of Edinburgh (1973).

[3] Nivat, M., On the interpretation of rec. pr. schemes: an algebraic approach. Lectures given at an Adv. Course in Saarbrücken, 1974

[4] Park, D., Lattice Theoretic Models for Formal Semantics. Lectures given at Adv. Course on Semantics of Pr. Lang., Saarbrücken, 1974

[5] Reynolds, Towards a theory of Type-Structure. in Lecture Notes in Comp. Science, Springer Verlag, to appear (june '74 ?)

[6] Scott, D., Outline of a Math. Theory of Computation. Proc. 4-th ann. Princeton Conf on Inf. Sc. and Systems, Princ. Un. (1970)

[7] Scott, D., The lattice of flow-diagrams. Springer lecture notes in math.'s (ed. E. Engeler) no. 188 p. 311-366, 1971

[8] Scott, D., Strachey, C., Towards a math. sem. for comp. lang. Proc. Symp. Comp. and automata. P.I.B. Symp. Ser. Vol. 24 (p. 19-46) 1972

[9] Scott, D., Lattice theory, datatypes and semantics. NYU Symp. Formal Sem.'s. Prentice-Hall 1972 (p. 65-106)

[10] Scott, D., Math. concepts in progr. lang. sem.'s. AFIPS Conf. Proc. vol 40 (p. 225-234). Spring Joint Comp. Conf 1972.

[11] Scott, D., Datatypes as Lattices. Lecture notes, amsterdam 1972.

[12] Strachey, C., Towards a formal Sem.'s. Formal Language Description languages (ed. T.B. Steel), North-Holland 1966 (p. 198-220).

[13] Scott, D., The problem of giving precise semantics for formal languages. ---- ? -------- ? ---- (±1969)

AANHANGSEL

<u>Taal</u> gegeven door de regels $S \to p \mid q \mid r \mid [S \supset S]$.

<u>Semantiek</u>

I. "<u>materiele implicatie</u>"

Ken waarheidswaarden **t**, f toe aan p, q, r. De waarheidswaarde van een uitdrukking $S_1 \supset S_2$ wordt op de volgende manier gevonden: $\begin{cases} f & \text{als aan } S_1 \ t \text{ is toegekend en aan } S_2 \ f \\ t & \text{anders} \end{cases}$

Een expressie heet een tautologie ("waar principe") als voor alle mogelijke toekenningen aan p, q, r de waarde t toegekend moet worden aan de expressie.

II "<u>materiele implicatie</u>", <u>venn-diagrammen</u>.

Ken deelverz. van een universumverzameling U toe aan p, q, r. Aan een expresie $S_1 \supset S_2$, waarbij aan S_1 en S_2 al U_1 resp U_2 zijn toegekend, wordt $(U - U_1) \cup U_2$ toegekend.

Tautologie: als voor alle mogelijke toekenningen aan p, q, r de hele U wordt toegekend aan de expressie

III "<u>strikte implicatie</u>"

Ken deelverz. van een universumverzameling U toe aan p, q, r. Aan een expresie $S_1 \supset S_2$, waarbij aan S_1 en S_2 al U_1 resp U_2 zijn toegekend, wordt $\begin{cases} U & \text{als } (U - U_1) \cup U_2 = U \\ \varnothing & \text{anders} \end{cases}$ toegekend.

(informeel: $p \supset q$ geldt wanneer er <u>geen enkele</u> situatie is waarin p wél en q niet geldt.) Tautologie: als bij II.

IV "<u>implicatie in de tegenwoordig tockomende tijd</u>"

Ken deelverz. van de tijdsuniversumverzameling T = {reele getallen} toe aan p, q, r ("de tijden waarop ze gelden").

Aan een expresie $S_1 \supset S_2$, waarbij aan S_1 en S_2 al T_1 resp T_2 zijn toegekend, wordt de volgende deelverz. van T toegekend:

t is er elt van <u>slals</u> er is een niet leeg interval I om t zó dat voor alle $t' \in I$: als t' in T_1 dan t' in T_2.

(informeel: $p \supset q$ geldt op tijd t slechts als de materiele implicatie geldt op zekere tijd $t' < t$ en voortduurt tot voorbij t.) Tautologieen: als bij II.

V Als IV met bovendien "geen-begin-en-geen-eind dogma"
Ken open deelverz≃ van het tijdsuniversum toe aan p, q, r (motivatie: niets gebeurt plotseling, alles heeft een aanlooptijd). Verder als bij IV.

VI "Constructieve implicatie"
Ken aan p, q, r verzamelingen van mogelijke constructies toe om p, q, r te vestigen. Aan een expressie $S_1 \supset S_2$, waarbij aan S_1 en S_2 al de verz≃ C_1 en C_2 zijn toegekend wordt de verzameling van alle functies $\tau : C_1 \to C_2$ toegekend. Tautologie: een expressie waaraan voor alle mogelijke toekenningen aan p, q, r een niet-lege verz≃ van constructiefuncties wordt toegekend.

Merk op dat bovenstaande semantische benaderingen ook gegeven hadden kunnen worden op de manier zoals dat in hoofdstuk 3 wordt gedaan.

Men kan bewijzen dat
I en II gelijkwaardig zijn, ook V en VI !
Overigens zijn ze alle niet-gelijkwaardig, hetgeen blijkt uit het volgende schema dat de eigenschap weergeeft van expressies om een tautologie te zijn:

	I, II	III	IV	V, VI
$[[p \supset q] \supset p] \supset p$	ja	nee	nee	nee
$[[[r \supset p] \supset q] \supset [r \supset p]] \supset [r \supset p]$	ja	ja	nee	nee
$[p \supset q] \supset [[q \supset r] \supset [p \supset r]]$	ja	ja	ja	ja
$q \supset [p \supset p]$	ja	ja	ja	ja
$p \supset [q \supset p]$	ja	nee	nee	ja

—//—