Abstracte Datatypen en Categorie-Theorie

Maarten Fokkinga, Universiteit Twente, fac INF
Postbus 217, 7500 AE Enschede, e-mail: `fokkinga@cs.utwente.nl`

30 december 1994

Sommige datatypen kunnen volledig met axioma's gekarakteriseerd worden, zonder op enigerlei wijze voor te schrijven hoe de elementen van het datatype er uit zien; we spreken dan van *abstract* datatype. We laten de karakterisering en enige stellingen en bewijzen in detail zien voor het *cartesisch product* en *disjoint union* van twee verzamelingen.

De begrippen en methoden die een rol spelen komen uit de categorie-theorie.

1 Inleiding. Datatypen spelen in het programmeren een grote rol. Wanneer alle eigenschappen van een datatype zijn vastgelegd die "van buitenaf waarneembaar" zijn, zonder een implementatie voor te schrijven, dan spreken we van een *abstract* datatype. We laten zo'n karakterisering in detail zien voor het cartesisch product en de disjoint union van twee verzamelingen. Van te voren geven we aan wanneer we een stel eigenschappen een karakterisering noemen, en we bewijzen dat de gegeven eigenschappen een karakterisering zijn. We geven ook een stelling en bewijs waarin de karakteriseringen van beide datatypen een rol spelen.

We kiezen het cartesisch product en de disjoint union ter illustratie, en niet zoiets als lijsten of bomen of natuurlijke getallen, ten eerste omdat de formules nu erg eenvoudig blijven, en ten tweede omdat de karakterisering —zelfs voor zulke alledaagse datatypen als het cartesisch product en de disjoint union— nog wel interessante en verrassende feiten oplevert.

De aanpak en methode die we volgen is typerend voor de categorie-theorie. We leggen uit wat categorie-theorie is, en hoe alles wat we gedaan hebben daarin past.

We besluiten met een heel korte schets van het verband met "echte" datatypen zoals lijsten, en met een suggestie voor een syntaxis van datatypen waaruit de karakterisering mechanisch afgeleid kan worden.

We zullen in het vervolg 'cartesisch product' een datatype noemen, hoewel strict genomen alleen ieder 'cartesisch product van twee verzamelingen' een datatype is, en 'cartesisch product' zelf beter een datatype-constructor genoemd kan worden of een geparameteriseerd datatype. Net zo voor 'disjoint union'.

2 Functie-nivo versus punt-nivo. De uiteindelijke karakterisering, en de daarop gebaseerde stellingen en bewijzen, formuleren we op "functie-nivo" in plaats van "punt-nivo"; dat wil zeggen, we gebruiken in die formuleringen uitsluitend functies en functie-samenstellingen, en géén individuele elementen en toepassingen van functies op elementen. De reden hiervoor is tweeërlei. Ten eerste blijkt dat de formules, en dus stellingen en bewijzen, wat mooier worden; zie bijvoorbeeld paragrafen 9, 10 en 15. Het zou te veel ruimte in beslag nemen om de formuleringen op functie-nivo in detail te vergelijken met de formuleringen op punt-nivo; dus dat laten we achterwege. Ten tweede blijkt dat de formules veel algemener worden; dat leggen we uit in paragrafen 16–18. Kortheidshalve zullen we steeds zo snel mogelijk naar het functie-nivo toewerken, ook al is dat ter plekke soms niet nodig.

In de *voorbeelden*, *implementaties* en sommige *toelichtingen* gebruiken we wél individuele elementen en passen we wél functies toe op die elementen.

Functie-compositie wordt met $\circ$ genoteerd; ter herinnering, voor functies f en g is $f \circ g$ de functie die zijn argument aan g onderwerpt en het resultaat daarvan aan f, dus $(f \circ g)(x) = f(g(x))$. Operatie $\circ$ bindt zwakker dan andere operatoren, dus bijvoorbeeld $f \circ g \vartriangle h = f \circ (g \vartriangle h)$ en *niet* $(f \circ g) \vartriangle h$. Functie id is de identiteit: $id(x) = x$.

Cartesisch product

Het begrip 'cartesisch product' speelt een grote rol in de wiskunde en informatica; je komt het overal tegen. We zullen hier ter herinnering de standaarddefinitie geven, daarna enige alternatieven noemen, en dan de centrale probleemstelling formuleren.

3 Standaard definitie. Gewoonlijk is voor twee verzamelingen A en B het cartesisch product van A en B (notatie: $A \times B$) gedefinieerd door:

$$A \times B \quad = \quad \{(a,b) \mid a \in A,\ b \in B\} \quad .$$

Belangrijke functies gerelateerd aan cartesische producten zijn de zogenaamde *extracties* exl en exr; hiermee kun je de A- en B-component weer terugkrijgen:

$$\begin{aligned} exl((a,b)) &= a \\ exr((a,b)) &= b \quad . \end{aligned}$$

Belangrijk is ook de *tupling* operator $\vartriangle$ die twee functies tot één combineert; hiermee kun je elementen van het cartesisch product produceren:

$$(f \vartriangle g)(x) \quad = \quad (f(x),\ g(x)) \quad .$$

De typering van deze functies luidt:

$$\begin{aligned} exl &: A \times B \to A \\ exr &: A \times B \to B \end{aligned}$$

$$f \vartriangle g \quad : \quad C \to A \times B, \quad \text{voor } f\colon C \to A \ \text{ en } \ g\colon C \to B \quad .$$

In programmeertalen zijn cartesische producten te herkennen in *records* (in Pascal en Modula, bijvoorbeeld) of *tuples* (in Miranda). Records zijn een syntactische verrijking van cartesische producten: de extracties worden bij records genoteerd met zelf verzonnen veldnamen. Bijvoorbeeld, het record-type **record** *links: A; rechts: B* **end** staat voor $A \times B$ waarbij $exl(x)$ genoteerd wordt door $x.links$, en $exr(x)$ door $x.rechts$. In Miranda heten de extracties *fst* en *snd*. Behoudens deze syntactische verschillen zijn er semantisch geen verschillen: wat je met records of tuples kan doen, kan je ook met cartesische producten doen, en omgekeerd.

4 Alternatieve implementaties. Bovenstaande definitie bevat enige willekeur: wanneer je de A-elementen als rechter-component neemt, en de B-elementen als linker-component, dan is er geen verschil te bemerken als je systematisch elementen in het cartesisch product van A en B vormt en gebruikt middels aangepaste functies exl, exr en $\vartriangle$. We zullen die definitie daarom een *implementatie* noemen. We hadden net zo goed de volgende implementatie kunnen nemen:

$$\begin{array}{lcl}
A \times B & = & \{(b,a) \mid a \in A,\ b \in B\} \\
exl((b,a)) & = & a \\
exr((b,a)) & = & b \\
(f \vartriangle g)(x) & = & (g(x), f(x)) \quad .
\end{array}$$

Nog een andere implementatie, in geval A en B verzamelingen van natuurlijke getallen zijn, is de volgende:

$$\begin{array}{lcl}
A \times B & = & \{2^a 3^b \mid a \in A,\ b \in B\} \\
exl(n) & = & \text{het aantal factoren 2 in de ontbinding van } n \\
exr(n) & = & \text{het aantal factoren 3 in de ontbinding van } n \\
(f \vartriangle g)(x) & = & 2^{f(x)} 3^{g(x)} \quad .
\end{array}$$

Er is geen verschil te merken met de oorspronkelijke implementatie (definitie) *wanneer je maar systematisch de elementen van $A \times B$ vormt en gebruikt middels de functies exl, exr en $\vartriangle$*. Uiteraard is niet iedere implementatie acceptabel. Merk op dat we "$A \times B$" nu, en verderop, gebruiken als *naam* van een verzameling; dat hoeft dus niet meer een verzameling van tweetallen te zijn.

5 Probleemstelling. Het bestaan van meerdere, intuitief acceptabele, implementaties roept de volgende vraag op: wat zijn de essentiële eigenschappen van 'cartesisch product', zónder daarbij een specifieke implementatie vast te leggen? We zoeken dus eigenschappen die zich uitspreken over een verzameling $A \times B$ en functies exl, exr en $f \vartriangle g$.

Voordat we deze vraag kunnen beantwoorden, moeten we weten wanneer een eigenschap de essentie van het cartesisch product vast legt en de implementatie volkomen vrij laat.

Daarvoor zullen we het volgende criterium gebruiken. Gesteld dat we het er over eens zijn wanneer twee implementaties *even acceptabel* zijn, dan noemen we een eigenschap E een *karakterisering* wanneer het volgende geldt:

- Implementaties die aan E voldoen, zijn alle even acceptabel.
- Implementaties die even acceptabel zijn, voldoen alle aan E indien één van hen er al aan voldoet.

Het probleem is nou dus een eigenschap te vinden die, volgens dit criterium, een karakterisering is, en die bovendien geldt voor de gegeven standaard implementatie.

Voor het intuïtieve, informele begrip 'even acceptabel' zullen we het formele begrip 'isomorfie' gebruiken (gedefinieerd in paragraaf 6); de correctheid van de overgang van deze informele intuïtie naar het formeel gedefinieerde begrip kunnen we uiteraard niet bewijzen. Hierover kan en mag verschil van mening zijn; bij een andere keuze krijg je mogelijk een andere karakterisering.

6 Isomorfie. Laat $(A \times B, \mathit{exl}, \mathit{exr}, \vartriangle)$ en $(A \times' B, \mathit{exl}', \mathit{exr}', \vartriangle')$ twee implementaties zijn. We vinden ze even acceptabel wanneer de systematische vervanging van de één door de ander nooit een verschil in uitkomst geeft in een programma (voor zover $A \times B$ en $A \times' B$ niet voorkomen in het type van de uitkomst). Dit is nogal lastig te formaliseren; er is onder andere een syntaxis van programma's voor nodig. Daarom nemen we een sterkere maar makkelijker te formaliseren voorwaarde. We zullen de twee implementaties even acceptabel vinden wanneer er functies $\varphi\colon A \times B \to A \times' B$ en $\varphi'\colon A \times' B \to A \times B$ bestaan waarmee we de ene implementatie in de andere kunnen uitdrukken, en omgekeerd. Formeel, de twee implementaties heten *isomorf* via φ, φ' wanneer:

$$\varphi \circ \varphi' = \mathit{id} \qquad \text{en} \qquad \varphi' \circ \varphi = \mathit{id}$$

en

$$\begin{array}{lcl@{\qquad\qquad}lcl}
\mathit{exl} & = & \mathit{exl}' \circ \varphi & \mathit{exl}' & = & \mathit{exl} \circ \varphi' \\
\mathit{exr} & = & \mathit{exr}' \circ \varphi & \mathit{exr}' & = & \mathit{exr} \circ \varphi' \\
f \vartriangle g & = & \varphi' \circ f \vartriangle' g & f \vartriangle' g & = & \varphi \circ f \vartriangle g \quad .
\end{array}$$

Het is niet moeilijk om aan te tonen dat, vanwege de eerste regel, de laatste drie gelijkheden volgen uit de voorgaande drie, en omgekeerd. In een bewijs van isomorfie hoeven we dus maar alleen de eerste regel en één drietal daaronder te bewijzen. (Oef.: 25.)

Merk tevens op dat de eerste regel zegt dat er een één-op-één verband bestaat tussen de verzamelingen $A \times B$ en $A \times' B$: element $x \in A \times B$ correspondeert met element $\varphi(x) \in A \times' B$, en element $y \in A \times' B$ correspondeert met $\varphi'(y) \in A \times B$, en $\varphi'(\varphi(x)) = x$ en $\varphi(\varphi'(y)) = y$. Dus $A \times B$ en $A \times' B$ zijn gelijkmachtig (hebben evenveel elementen).

De gegeven standaard en alternatieve implementaties zijn alle isomorf. De bwijzen zijn welhaast triviaal, en laten we achterwege. (Oef.: 26.)

7 Intuïtie. Om eigenschappen te vinden die tezamen het cartesisch product karakteriseren, gaan we uit van de volgende intuïtie:

- Voor elk element in A en elk element in B bestaat er een element in het cartesisch product van A en B dat die gegeven elementen als linker- en rechtercomponent heeft.

- Een element in het cartesisch product van A en B is volkomen bepaald door zijn linker- en rechtercomponent; met andere woorden, twee elementen in het cartesisch product van A en B zijn hetzelfde, wanneer ze dezelfde linker- en rechtercomponenten hebben.

Deze intuïtie had al vóór paragraaf 4 gegeven kunnen worden! Het is ook niet moeilijk te verifieren dat deze beweringen, of de hieronderstaande formaliseringen, gelden voor ieder van de gegeven implementaties. (Oef.: 27.) Zoals aangekondigd in paragraaf 2 streven we bij de formalisering naar het gebruik van functies in plaats van individuele elementen.

De eerste bewering formaliseren we als volgt. Laat $f(x)$ en $g(x)$ twee elementen in A en B aanduiden. Dan is $(f \vartriangle g)(x)$ dat bedoelde element in het product, dat wil zeggen:

$$exl((f \vartriangle g)(x)) = f(x) \quad \text{en} \quad exr((f \vartriangle g)(x)) = g(x) \quad .$$

Het 'bestaan' van dat element is gegarandeerd doordat functie $f \vartriangle g$ bestaat.

Nu de tweede bewering van de intuïtie. Laat $h(x)$ een willekeurig element in het product aanduiden, en beschouw ook het element $(f \vartriangle g)(x)$ in het product. Wanneer ze gelijke linker- en rechtercomponent hebben, dus —gezien de vorige bewering— $f(x)$ respectievelijk $g(x)$, dan zijn ze gelijk:

$$exl(h(x)) = f(x) \quad \text{en} \quad exr(h(x)) = g(x) \quad \Rightarrow \quad h(x) = (f \vartriangle g)(x) \quad .$$

Merk op dat we deze bewering ook als volgt kunnen lezen: bij gegeven $f\colon C \to A$ en $g\colon C \to B$ kun je een functie $h\colon C \to A \times B$ *definieren* door vast te leggen wat de exl van $h(x)$ is, namelijk $f(x)$, en wat de exr van $h(x)$ is, namelijk $g(x)$; die functie h is dan $f \vartriangle g$.

Op functie-nivo uitgedrukt worden bovenstaande formules wat mooier. Zij luiden, respectievelijk:

$$\begin{aligned} h = f \vartriangle g \quad &\Rightarrow \quad exl \circ h = f \quad \text{en} \quad exr \circ h = g \\ h = f \vartriangle g \quad &\Leftarrow \quad exl \circ h = f \quad \text{en} \quad exr \circ h = g \quad . \end{aligned}$$

Tesamen:

$$h = f \vartriangle g \quad \equiv \quad exl \circ h = f \quad \text{en} \quad exr \circ h = g \quad .$$

We zullen in paragraaf 10 zien dat alle implementaties $(A \times B, exl, exr, \vartriangle)$ die hieraan voldoen, isomorf zijn aan elkaar, en omgekeerd, dat isomorfe implementaties alle hieraan voldoen als één er al aan voldoet. Hiermee is dan de gezochte karakterisering van 'cartesisch product' gevonden. Om een mogelijk misverstand met de "standaard implementatie" te voorkomen, zullen we over 'product' spreken; het cartesisch product van A en B is een van de mogelijke producten van A en B.

Het is wellicht leerzaam om op te merken dat de eerste bewering alléén niet voldoende is om het product te karakteriseren. Bijvoorbeeld, neem, voor getalverzamelingen A en B:

$$
\begin{array}{lll}
A \times B & = & \{0,1,2,\ldots\} \\
exl(n) & = & \text{het aantal factoren 2 in de ontbinding van } n \\
exr(n) & = & \text{het aantal factoren 3 in de ontbinding van } n \\
(f \vartriangle g)(x) & = & 2^{f(x)}3^{g(x)} \quad .
\end{array}
$$

De eerste bewering, namelijk $exl((f \vartriangle g)(x)) = f(x)$ en net zo met exr, is nu wel vervuld. Maar de hier gedefinieerde $A \times B$ is in het algemeen niet van gelijke grootte als die bij de standaard implementatie. Dus bestaan de functies φ, φ' niet die nodig zijn voor een isomorfie van deze implementatie met de standaard.

De tweede bewering van de intuïtie is bij deze implementatie inderdaad *niet* vervuld: de elementen 2^a3^b en $2^a3^b5^{10}$ hebben gelijke linker- en rechtercomponent, maar zijn niet dezelfde; ofwel, op functie-nivo, de functies $h(x) = 2^{f(x)}3^{g(x)}$ en $h'(x) = 2^{f(x)}3^{g(x)}5^{10}$ hebben beide gelijke exl en exr-componenten, maar zijn niet gelijk. (Oef.: 28.)

We vatten nu het bovenstaande samen in een definitie.

8 Definitie: product. Laat A en B gegeven zijn. Veronderstel voorts dat functies exl: $C \to A$, exr: $C \to B$ en operator $\vartriangle$ voldoen aan de de volgende eigenschap:

$$h = f \vartriangle g \qquad \equiv \qquad exl \circ h = f \ \text{ en } \ exr \circ h = g \quad . \qquad \text{prod-CHARN}$$

Dan noemen we het viertal $(C, exl, exr, \vartriangle)$ en ook C alleen een *product* van A en B, en schrijven we ook $A \times B$ voor C. (De naam 'prod-CHARN' is een afkorting van 'product Characterisation'.) (Oef.: 29.)

Als we 'het cartesisch product' zoals bij de standaard implementatie een *concreet datatype* noemen, dan is 'product' zoals hier gedefinieerd een *abstract datatype*. 'Een product van A en B' kunnen we ook 'een implementatie van *hét* product van A en B' noemen.

We laten nu een stel eigenschappen ("wetten") van exl, exr en $\vartriangle$ zien, die volgen uit de karakterisering prod-CHARN. Die eigenschappen zullen we verderop gebruiken in de bewijzen van een paar stellingen.

9 Wetten voor product. Laat A en B gegeven zijn, en veronderstel voorts dat $(A \times B, exl, exr, \vartriangle)$ een product van A en B is. Dan geldt ook:

$$exl \circ f \vartriangle g = f \ \text{ en } \ exr \circ f \vartriangle g = g \qquad \text{prod-SELF}$$

$$exl \vartriangle exr = id \qquad \text{prod-ID}$$

$f \vartriangle g \circ h = (f \circ h) \vartriangle (g \circ h)$ prod-FUSION

Wet prod-SELF volgt uit prod-CHARN door $h = f \vartriangle g$ te nemen; de linkerkant van prod-CHARN is dan waar, dus de rechterkant ook: prod-SELF. De naam 'SELF' is zo gekozen omdat $f \vartriangle g$ *zelf* een oplossing is voor h in de vergelijkingen $exl \circ h = f$ en $exr \circ h = g$. Wet prod-ID volgt uit prod-CHARN door $f, g, h = exl, exr, id$ te nemen; de rechterkant van prod-CHARN is dan waar, dus de linkerkant ook: prod-ID. Wet prod-FUSION wordt als volgt bewezen:

$$
\begin{array}{ll}
 & f \vartriangle g \circ h = (f \circ h) \vartriangle (g \circ h) \\
\equiv & \quad \text{prod-CHARN met } f, g, h := (f \circ h),\ (g \circ h),\ f \vartriangle g \circ h \\
 & exl \circ f \vartriangle g \circ h = f \circ h \ \text{ en } \ exr \circ f \vartriangle g \circ h = g \circ h \\
\Leftarrow & \quad \text{Leibniz (dat wil zeggen: } \cdots x\text{---} = \cdots y\text{---} \text{ volgt uit } x = y \text{)} \\
 & exl \circ f \vartriangle g = f \ \text{ en } \ exr \circ f \vartriangle g = g \\
\equiv & \quad \text{prod-SELF} \\
 & true \quad .
\end{array}
$$

De naam 'FUSION' is zo gekozen omdat de wet laat zien dat een compositie van een $\vartriangle$-samenstelling met een functie h kan *samensmelten* tot één $\vartriangle$-samenstelling. (Oef.: 30, 31, 32 .)

Zoals aangekondigd laten we nu zien dat prod-CHARN een karakterisering is.

10 Stelling: uniciteit van product. Laat A en B gegeven zijn. Veronderstel dat $(C, exl, exr, \vartriangle)$ een product van A en B is, evenals $(C', exl', exr', \vartriangle')$. Dan zijn ze isomorf.
Bewijs. Eerst construeren we functies φ: $C \to C'$ en φ': $C' \to C$ via welke de implementaties isomorf zijn. Op grond van het type voor $\vartriangle'$ kiezen we $\varphi = \ldots \vartriangle' \ldots$, en op grond van het type van exl en exr kiezen we zelfs $\varphi = exl \vartriangle' exr$. Dan is inderdaad φ: $C \to C'$. Net zo kiezen we $\varphi' = exl' \vartriangle exr'$: $C' \to C$.

Dat $\varphi \circ \varphi'$ de identiteit is wordt als volgt aangetoond:

$$
\begin{array}{ll}
 & exl \vartriangle' exr \ \circ \ exl' \vartriangle exr' \\
= & \quad \text{prod-FUSION met } h = exl' \vartriangle exr' \\
 & (exl \circ exl' \vartriangle exr') \vartriangle' (exr \circ exl' \vartriangle exr') \\
= & \quad \text{prod-SELF in het linker- en rechterargument van } \vartriangle' \\
 & exl' \vartriangle' exr' \\
= & \quad \text{prod-ID} \\
 & id \quad .
\end{array}
$$

Het bewijs van $\varphi' \circ \varphi = id$ gaat net zo.

Rest nog aan te tonen dat $exl, exr, \vartriangle$ met behulp van φ, φ' uitgedrukt kunnen worden in $exl', exr', \vartriangle'$. De eerste gelijkheid, $exl = exl' \circ \varphi$, is welhaast triviaal (van rechterlid naar linker):

$$
\begin{array}{ll}
 & exl' \circ exl \vartriangle' exr \\
= & \quad \text{prod-}\textsc{Self} \\
 & exl \quad .
\end{array}
$$

Even eenvoudig gaat het voor $exr = exr' \circ \varphi$. Voor de gelijkheid $f \vartriangle g = \varphi' \circ f \vartriangle' g$ luidt de calculatie (weer van rechterlid naar linker):

$$
\begin{array}{ll}
 & \varphi' \circ f \vartriangle' g \\
= & \quad \text{definitie } \varphi' \\
 & exl' \vartriangle exr' \circ f \vartriangle' g \\
= & \quad \text{prod-}\textsc{Fusion} \text{ met } h = f \vartriangle' g \\
 & (exl' \circ f \vartriangle' g) \vartriangle (exr' \circ f \vartriangle' g) \\
= & \quad \text{prod-}\textsc{Self} \text{ in linker- en rechterlid} \\
 & f \quad \vartriangle \quad g \quad .
\end{array}
$$

Hiermee is aan de bewijsverplichtingen voldaan.

We laten het aan de lezer over om te bewijzen dat een implementatie die isomorf is aan één die aan prod-CHARN voldoet, zelf ook aan prod-CHARN voldoet. (Oef.: 33, 34, 35.)

Disjoint Union

Om het niet bij één voorbeeld te laten behandelen we ook nog de disjoint union. Het zal blijken dat er veel analogie is tussen disjoint unions en cartesische producten, hoewel dat niet duidelijk is uit de gebruikelijke definities van de concrete datatypen. De analogie wordt in de sectie over categorie-theorie geformaliseerd.

11 Disjoint union. Het begrip 'disjoint union', ofwel 'onderscheiden vereniging', is minder bekend dan cartesisch product, maar komt toch ook veel voor. Een gebruikelijke wiskundige formulering is als volgt. Voor verzamelingen A en B is de disjoint union van A en B (notatie: $A + B$) gedefinieerd door:

$$A + B \quad = \quad \{(0, a) | \; a \in A\} \cup \{(1, b) | \; b \in B\} \quad .$$

Dus, informeel gezegd, $A + B$ is een soort vereniging van A en B, maar zó dat aan de elementen in die vereniging te zien is of ze uit de A dan wel B komen; daartoe zijn de elementen in die vereniging gemerkt met 0 respectievelijk 1. Andere merktekens in plaats van 0 en 1, zoals 1 en 2 of '*li*' en '*re*' of '*true*' en '*false*', zijn net zo goed. Merk op dat A en B dezelfde verzameling mogen zijn; aan het merkteken in de elementen van $A + A$ is toch te zien of het element in het linker dan wel rechter deel zit. Dit is het verschil tussen de gewone union en de disjoint union.

Belangrijke functies voor de disjoint union zijn de zogenaamde *injecties* en de *case*-operator. De injecties *inl* en *inr* leveren de elementen in de disjoint union, door hun argument van een merkteken te voorzien. De case-operator $\triangledown$ combineert twee functies f en g tot één; deze combinatie, *gevalsonderscheid* genoemd, werkt op de disjoint union, inspecteert de merktekens van zijn argument en past f dan wel g toe op het gemerkte element. In formules:

$$\begin{array}{lcl} \mathit{inl}(a) & = & (0,a) \\ \mathit{inr}(b) & = & (1,b) \\ (f \triangledown g)((0,a)) & = & f(a) \\ (f \triangledown g)((1,b)) & = & g(b) \quad . \end{array}$$

De typering luidt dus:

$$\begin{array}{lcl} \mathit{inl} & : & A \to A+B \\ \mathit{inr} & : & B \to A+B \\ f \triangledown g & : & A+B \to C \quad \text{voor } f\colon A \to C \text{ en } g\colon B \to C \quad . \end{array}$$

Door systematisch de injecties en het gevalsonderscheid te gebruiken, hoeven de merktekens *nergens* expliciet genoemd te worden (behalve in de definities hierboven).

In Pascal en Modula verschijnt de onderscheiden vereniging onder het mom van de zogenaamde variant record. Net als bij het cartesisch product zijn de elementen zelf benoemd met een veldnaam; het merkteken mag ook met een veldnaam benoemd worden. Bijvoorbeeld, een representatie in Modula van $Integer + Char$ is:

$$\textbf{record}\ \ \textbf{case}\ merkteken\colon Integer\ \textbf{of}\ 0\colon inl\colon Integer \ \ |\ \ 1\colon inr\colon Char\ \textbf{end}\ .$$

Hierbij zijn $merkteken$ en inl en inr vrij te kiezen veldnamen. In Miranda zijn het de algebraïsche typen die een onderscheiden vereniging ingebouwd hebben. Bijvoorbeeld, voor de disjoint union van num met $char$:

$$numchar \quad ::= \quad Inl\ num \quad | \quad Inr\ char \quad .$$

Hier zijn Inl en Inr de injecties; de gebruiker is vrij om ze anders te noemen. Een veelvoorkomende toepassing van disjoint union is in de weergave van persoonsgegevens: deze bestaan bijvoorbeeld uit een naam, een Boolean 'volwassen', en afhankelijk van de volwassenheid een SOFI-nummer dan wel geboortejaar:

$$\begin{array}{l} \textbf{record} \\ \quad naam\colon String; \\ \quad \textbf{record}\ \ \textbf{case}\ volw\colon Boolean\ \textbf{of} \\ \qquad\quad true\colon sofi\colon Integer \ \ |\ \ false\colon gebjr\colon Integer \\ \quad \textbf{end} \\ \ \textbf{end} \quad . \end{array}$$

12 Alternatieve implementaties. Zoals al eerder opgemerkt is, doet de keuze van de merktekens nauwelijks terzake; er zijn alternatieve implementaties van de disjoint union. Bovendien kan, voor verzamelingen van natuurlijke getallen, de disjoint union ook op de volgende manier geïmplementeerd worden:

$$\begin{array}{lcll} A+B & = & \{2a \mid a\in A\}\cup\{1+2b \mid b\in B\} & \\ \mathit{inl}(a) & = & 2a & \\ \mathit{inr}(b) & = & 1+2b & \\ (f \triangledown g)(x) & = & f(x/2), & \text{als } x \text{ even is} \\ & = & g((x-1)/2), & \text{als } x \text{ oneven is} \quad . \end{array}$$

Er geen verschil te merken met de andere implemenaties *als je systematisch elementen van de disjoint union vormt en gebruikt middels de functies* inl *,* inr *en de operatie* $\triangledown$.

13 Karakterisering. Net als bij het cartesisch product willen we een karakterisering van de 'disjoint union' van A en B , zonder een specifieke implementatie voor te schrijven. Om misverstanden met de "standaard" implementatie van de disjoint union te voorkomen zullen we spreken over 'som' in plaats van disjoint union. Het hele verhaal gaat analoog aan dat van het product. We zullen het daarom kort houden. Desgewenst kan de lezer al onze beweringen uitwerken tot de gedetailleerheid van de redenering bij het product.

Om eigenschappen te vinden die tezamen de som karakteriseren, gaan we uit van de volgende intuïtie:

- Voor elk element in A bestaat er een element in de som van A en B dat volgens het gevalsonderscheid het gegeven element voorstelt; net zo voor elk element in B .
- Er zijn geen andere elementen in de som van A en B dan degene die door de injecties opgeleverd worden; met andere woorden, een functie h op de som ligt volkomen vast wanneer voorgeschreven is wat zijn uitkomst is op argumenten van de vorm $\mathit{inl}(a)$, en wat de uitkomst is op argumenten van de vorm $\mathit{inr}(b)$.

Deze intuïtie had al vóór paragraaf 11 gegeven kunnen worden! (Oef.: 36.)

Voor de formalisering op functie-nivo van deze intuïtie veronderstellen we dat $f\colon A\to D$ en $g\colon B\to D$ willekeurige functies zijn. De eerste bewering luidt dan als volgt. Voor willekeurige $a\in A$ en $b\in B$ zijn $\mathit{inl}(a)$ en $\mathit{inr}(b)$ de bedoelde elementen in de som; het gevalsonderscheid $h=f\triangledown g$ op hun toegepast levert $f(a)$ respectievelijk $g(b)$:

$$h=f\triangledown g \quad\Rightarrow\quad \forall a,b :: \quad h(\mathit{inl}(a))=f(a) \ \text{ en } \ h(\mathit{inr}(b))=g(b) \quad .$$

Voor de tweede bewering vinden we de omgekeerde implicatie; als het rechterlid hierboven waar is, dan is h volkomen bepaald, en dus volgens de eerste bewering gelijk aan $f\triangledown g$:

$$h=f\triangledown g \quad\Leftarrow\quad \forall a,b :: \quad h(\mathit{inl}(a))=f(a) \ \text{ en } \ h(\mathit{inr}(b))=g(b) \quad .$$

De twee beweringen tezamen geven een equivalentie; die blijkt een karakterisering te zijn (en geldig voor de disjoint union van A en B).

14 Definitie en stellingen voor som. Laat A en B gegeven zijn. Veronderstel dat functies $inl\colon A \to C$ en $inr\colon B \to C$ en operator $\triangledown$ voldoen aan de volgende eigenschap:

$$h = f \triangledown g \quad \equiv \quad h \circ inl = f \text{ en } h \circ inr = g \quad . \qquad \text{som-CHARN}$$

Dan noemen we het viertal $(C, inl, inr, \triangledown)$ en ook C alleen een *som* van A en B, en schrijven we ook $A + B$ voor C. Er geldt dan ook:

$$f \triangledown g \circ inl = f \text{ en } f \triangledown g \circ inr = g \qquad \text{som-SELF}$$

$$inl \triangledown inr = id \qquad \text{som-ID}$$

$$f \circ g \triangledown h = (f \circ g) \triangledown (f \circ h) \quad . \qquad \text{som-FUSION}$$

Het bewijs gaat net zo als bij de wetten voor het product van A en B.

Voorts is de som uniek, op isomorfie na. Beschouw $(C, inl, inr, \triangledown)$ en $(C', inl', inr', \triangledown')$. Als beide een som zijn van A en B, dan zijn ze isomorf. En omgekeerd, als beide isomorf zijn en één is een som van A en B, dan is de ander dat ook. Ook hier gaat het bewijs net zo als in het bewijs van Stelling 10. (Overigens vereist deze bewering nog wel dat het begrip isomorfie van implementaties voor een som formeel gedefinieerd wordt; ook dat gaat analoog aan het product.)

15 Toepassing. Als toepassing van de definities van product en som, en de daaruit afgeleide wetten, presenteren we de volgende stelling:

$$(f \triangledown g) \vartriangle (h \triangledown j) \quad = \quad (f \vartriangle h) \triangledown (g \vartriangle j) \quad .$$

Wanneer we $f \triangledown g$ schrijven als $f|g$ en $f \vartriangle g$ als $\frac{f}{g}$, dan komt de structuur van deze stelling beter tot uiting:

$$\frac{f|g}{h|j} \quad = \quad \frac{f}{h}\bigg|\frac{g}{j} \quad .$$

De intuïtive betekenis van deze stelling is dat de operatoren $|$ en $-$, dus $\triangledown$ en $\vartriangle$, geen onderlinge interactie hebben: het maakt niet uit of je eerst $|$ en dan $-$ doet, of andersom. Het bewijs gaat bijvoorbeeld als volgt:

$$\begin{array}{cl}
 & (f \triangledown g) \vartriangle (h \triangledown j) = (f \vartriangle h) \triangledown (g \vartriangle j) \\
\equiv & \quad \text{som-CHARN met de substitutie } f, g, h := f \vartriangle h,\ g \vartriangle j, \text{ linkerlid} \\
 & (f \triangledown g) \vartriangle (h \triangledown j) \circ inl = f \vartriangle h \quad \text{en} \quad (f \triangledown g) \vartriangle (h \triangledown j) \circ inr = g \vartriangle j \\
\equiv & \quad \text{prod-FUSION (op twee plaatsen)} \\
 & (f \triangledown g \circ inl) \vartriangle (h \triangledown j \circ inl) = f \vartriangle h \quad \text{en} \quad (f \triangledown g \circ inr) \vartriangle (h \triangledown j \circ inr) = g \vartriangle j \\
\equiv & \quad \text{som-SELF (op vier plaatsen)} \\
 & f \vartriangle h = f \vartriangle h \quad \text{en} \quad g \vartriangle j = g \vartriangle j \\
\equiv & \quad \text{gelijkheid} \\
 & true \quad .
\end{array}$$

Er zijn vele andere varianten van dit bewijs. (Oef.: 37, 38.)

Categorie theorie

Categorie-theorie is een vrij jonge tak van wiskunde, ontstaan in de '50-er jaren, die ook in de informatica steeds meer toepassingen vindt. Aan de hand van wat we hierboven gedaan hebben, leggen we uit wat categorie-theorie is, en geven we een paar belangrijke ideeën uit de categorie-theorie.

16 Terugblik. Laten we nog eens kijken hoe het begrip product gekarakteriseerd is. (Al het volgende gaat analoog voor som). In feite is er in de definitie van product niets gezegd over de "interne" structuur van een product, maar alleen iets over de onderlinge relaties van de functies die er op werken: exl , exr en $\vartriangle$. Dit is de aanpak in categorie-theorie: de bestudeerde dingen worden niet "intern" vastgelegd, maar alleen "extern" door "wat ermee gedaan kan worden". In de categorie-theorie zijn er allerlei wiskundige begrippen op zo'n manier gekarakteriseerd, en aan de hand van de karakterisering op hun eigenschappen onderzocht.

17 Categorie. Omdat er in de definitie van product niets gezegd is over de "interne" structuur van een product van A en B, is het in feite niet nodig dat A en B verzamelingen zijn: nergens is van het lidmaatschap $\in$ gebruik gemaakt in de gepresenteerde definities, stellingen en bewijzen (maar wel in de voorbeelden en implementaties). Evenmin is het nodig dat $\mathit{exl}, \mathit{exr}, f, g, h, f \vartriangle g$ functies zijn: nergens wordt een van hen toegepast op een argument, in de gepresenteerde definities, stellingen en bewijzen (maar wel in de voorbeelden en implementaties). Laten we daarom overal het woord 'verzameling' vervangen door 'object', en het woord 'functie' door 'morfisme'. Het enige wat er over objecten en morfismen verondersteld wordt, in de gepresenteerde definities, stellingen en bewijzen, is dit:

- morfismen zijn getypeerd met objecten, bijvoorbeeld $f\colon A \to B$,
- compositie $\circ$ is een operatie op morfismen die associatief is: $f \circ (g \circ h) = (f \circ g) \circ h$, en id als neutraal element heeft: $\mathit{id} \circ f = f = f \circ \mathit{id}$.

Zo'n stel 'objecten met morfismen' heet *categorie*. 'Verzamelingen met functies' zijn dus een voorbeeld van een categorie. Wil je op categorie-theoretische manier programmeren, dan moet je alles op "functie-nivo" uitdrukken met uitsluitend functie-samenstellingen, in plaats van op "punt-nivo" waar je met expliciete argumenten kunt werken. Zo op het eerste gezicht lijkt categorie-theorie daarom een keurslijf; maar soms (vaak?) worden de formules en bewijzen er wel eleganter door. (Oef.: 39, 40.)

18 Algemeenheid. Het begrip product blijkt dus zinvol te zijn voor iedere categorie: er is precies gedefinieerd wanneer een object een product is van twee objecten A en B. Dus ook voor, bijvoorbeeld, de categorie van 'algebra's met homomorfismen' ligt vast wat een product is van twee algebra's. En sterker nog, de stellingen en bewijzen die we hierboven gepresenteerd hebben, zijn geldig in iedere categorie!! Ga maar na: we hebben geen andere eigenschappen verondersteld dan die voor objecten en morfismen gelden (en ze zijn ook *allemaal* gebruikt). De categorie-theorie is zeer algemeen, en spreekt zich uit over veel velden binnen de wiskunde (waaronder informatica), en onderlinge verbanden worden zodoende duidelijk en formeel gemaakt. (Oef.: 41.)

Het voert te ver om alle ideeën uit de cateorie-theorie hier te bespreken. We beperken ons tot drie: de universal property, dualiteit, en het begrip functor.

19 Universal property. Kijk nog eens naar prod-CHARN. In feite staat er dat de vergelijkingen $exl \circ h = f$ en $exr \circ h = g$ precies één oplossing hebben voor onbekende h (bij gegeven f en g); die oplossing kan dan geschreven worden met een notatie waarin f en g voorkomen, in dit geval $f \vartriangle g$. Welnu, we spreken van *universal property* wanneer een stel vergelijkingen precies één oplossing heeft. Veel begrippen uit het dagelijkse leven (van de wiskundige en informaticus) blijken in de categorie-theorie een karakterisering te hebben middels een universal property. De begrippen product en som hebben we als voorbeeld behandeld. Ook de meeste andere datatypen uit programmeertalen kunnen zo gekarakteriseerd worden. (Oef.: 42.)

20 Dualiteit. Dualiseren is een handeling met formules, namelijk het systematisch omwisselen van linker- en rechter-operand van composities en het omwisselen van linker- en rechterlid in typen. Bijvoorbeeld, de volgende (fantasie)formules ontstaan uit elkaar door dualisatie:

$$\begin{array}{ll} \forall B\ \ \forall f\colon A \to B\ \ \exists!\, g\colon C \to B :: & f = g \circ h \\ \forall B\ \ \forall f\colon B \to A\ \ \exists!\, g\colon B \to C :: & f = h \circ g \quad . \end{array}$$

Het is niet moeilijk aan te tonen dat dualisatie de geldigheid van categorie-theoretische stellingen en bewijzen behoudt. (Want de categorie-theoretische axioma's gaan door dualisatie over in de axioma's zelf.) Niet alleen is dualisatie een formeel spel met formules, maar ook blijkt dat de door dualisatie verkregen 'duale begrippen' meestal praktische betekenis hebben. Een goed voorbeeld hiervan zijn de begrippen product en som: zij zijn elkaars duale, zoals we nu zullen laten zien.

Herinner je de definitie van product:
een viertal $(C,\ exl\colon C \to A,\ exr\colon C \to B,\ \vartriangle)$ heet product van A en B wanneer:

$$h = f \vartriangle g \qquad \equiv \qquad exl \circ h = f \ \text{ en } \ exr \circ h = g \quad .$$

Welnu, laten we deze definitie dualiseren. We gebruiken tevens het woord 'som' in plaats van 'product', en de symbolen $inl, inr, \triangledown$ in plaats van de symbolen $exl, exr, \vartriangle$. Systematische omwisseling van de operanden van $\circ$ en van $\to$ in bovenstaande definitie levert dan de volgende:
een viertal $(C,\ inl\colon A \to C,\ inr\colon B \to C,\ \triangledown)$ heet som van A en B wanneer:

$$h = f \triangledown g \qquad \equiv \qquad h \circ inl = f \ \text{ en } \ h \circ inr = g \quad .$$

Dit is precies de eerder gegeven definitie! En omdat dualiseren de geldigheid van categorie-theoretisch bewezen stellingen behoudt, kunnen we zonder verdere bewijsverplichting concluderen dat ook de som van twee objecten op isomorfie na uniek bepaald is. Het uitvoeren van de analoge bewijzen, zoals aangegeven in paragraaf 14 is dus niet eens nodig. (Oef.: 43, 44.)

21 Functor. We weten inmiddels dat er voor ieder tweetal verzamelingen A en B een product van A en B bestaat. Definieer nu, zoals we al eerder gesuggereerd hebben:

$$A \times B \quad = \quad \text{een of ander willekeurig, maar vast gekozen, product van } A \text{ en } B\,.$$

Dan is $A \times B$ een verzameling waarvan de "interne structuur" niet bekend is; er kunnen immers heel verschillende implementaties bestaan, en bovenstaande definitie verklapt niet welke er gekozen wordt. Toch kunnen we de "product-structuur" van $A \times B$ in zekere zin exploiteren. Het belangrijkste voorbeeld daarvan is wellicht een functie die de product-structuur intact laat en alleen de onderdelen van een gestructureerd argument wijzigt. Dat kunnen we als volgt bereiken. We definiëren, bij gegeven $f\colon A \to A'$ en $g\colon B \to B'$, een functie $f \times g$ die f toepast op het "linkerlid" van het argument en g op het "rechterlid", en daarbij de "product-structuur" van het argument behoudt:

$$f \times g \quad = \quad (f \circ \mathit{exl}) \vartriangle (g \circ \mathit{exr}) \quad : A \times B \to A' \times B' \quad .$$

Er geldt nu dat deze operatie $\times$ en de compositie $\circ$ geen interactie hebben:

$$(f \times g) \circ (f' \times g') \quad = \quad (f \circ f') \times (g \circ g') \quad ,$$

en dat operatie $\times$ de identiteit behoudt:

$$\mathit{id} \times \mathit{id} \quad = \quad \mathit{id} \quad .$$

De bewijzen, op grond van de prod-wetten, laten we over aan de lezer: calculaties van een paar regels. Merk op dat, alweer, nergens in de definitie en de bewijzen gebruik is gemaakt van lidmaatschap of applicatie. De definitie en bewijzen zijn geldig in iedere categorie.

Operatoren zoals bovenstaande $\times$ die op objecten en morfismen werken, en geen interactie hebben met compositie, heten *functoren*; zij spelen een grote rol in categorie-theorie, onder andere doordat zij structuur van objecten "extern waarneembaar" vastleggen. Hier is een voorbeeld van het gebruik van de $\times$-functor:

$$\begin{aligned} f \circ \mathit{exl} \quad &= \quad \mathit{exl} \circ f \times g \\ g \circ \mathit{exr} \quad &= \quad \mathit{exr} \circ f \times g \quad . \end{aligned}$$

Het bewijs is weer niet moeilijk: een calculatie van een paar regels. (Oef.: 45, 46, 47.)

Slotopmerkingen

22 Andere datatypen. Er zijn veel meer datatypen dan het cartesisch product en de disjoint union. Bijvoorbeeld, lijsten, bomen, queues, stacks, enzovoorts. Veel van hen kunnen op soortgelijke wijze gekarakteriseerd worden als product en som. Wanneer daarbij het begrippen-arsenaal van de categorie-theorie gebruikt wordt, kunnen onderlinge verbanden duidelijk en formeel gemaakt worden.

Het voert te ver om dat allemaal in detail aan te tonen. Toch willen we niet nalaten twee voorbeelden te geven; we verwachten niet dat alles glashelder is, maar hopen wel enige belangstelling op te wekken. Hoewel de definities in iedere categorie geïnterpreteerd kunnen worden, zullen wij ze hier toelichten aan de hand van 'verzamelingen met totale functies'. (Aan de programmeertaal Miranda ligt niet deze categorie ten grondslag, maar die van 'verzamelingen met partiële functies'). De voorbeelden zijn: stromen (oneindige rijen) en lijsten (eindige rijen). In beide gevallen is er maar één parameter A, in plaats van het tweetal A en B zo als bij product en som. De rol van de infix operatoren $\times$ en $+$ wordt nu overgenomen door prefix operatoren $Strm$ en Lst.

Allereerst het datatype van lijsten over A. Deze wordt analoog aan som gekarakteriseerd: in plaats van de *constructoren* inl en inr hebben we nu nil en $cons$, en in plaats van operator $\triangledown$ schrijven we nu $\_\mathrm{foldr}\_$ (in infix-notatie). De definitie luidt als volgt. Een viertal $(C,\ nil\colon () \to C,\ cons\colon A\times C \to C,\ \mathrm{foldr})$ is (een implementatie van) het datatype *lijst* over A wanneer:

$$h = f \,\mathrm{foldr}\, g \qquad \equiv \qquad h \circ nil = f \;\text{ en }\; h \circ cons = g \circ id \times h \quad .$$

In dat geval schrijven we $Lst\,A$ voor C. Het rechterlid van deze equivalentie geeft op functie-nivo een definitie van $h\colon Lst\,A \to \ldots$ met inductie naar de $nil, cons$-opbouw van het argument:

$$h(nil(\,)) = f() \qquad \text{en} \qquad h(cons(x,xs)) = g(x, h(xs)) \quad .$$

Zo'n h, kortweg genoteerd met $f\,\mathrm{foldr}\,g$, heeft hetvolgende effect:

$$h(cons(a_0, cons(a_1, \ldots cons(a_{n-1}, nil())))) \quad = \quad g(a_0, g(a_1, \ldots g(a_{n-1}, f()))) \quad .$$

Definieer voorts operatie Lst op functies zó dat $Lst\,f\,(xs)$ de functie f op elk element in xs toepast, en de lijst-structuur behoudt. Dus $Lst\,f$ is de bekende $map\,f$:

$$Lst\,f \circ nil = nil \qquad \text{en} \qquad Lst\,f \circ cons = cons \circ f \times Lst\,f$$

ofwel

$$Lst\,f\,(nil()) = nil() \qquad \text{en} \qquad Lst\,f\,(cons(x,xs)) = cons(f(x), Lst\,f\,(xs)) \quad .$$

Dan is Lst een functor:

$$Lst\,f\colon\ Lst\,A \to Lst\,B, \qquad \text{voor } f\colon\ A \to B$$
$$Lst\,f \circ Lst\,g \;=\; Lst(f \circ g)$$

$$Lst\, id \;=\; id \quad .$$

(Oef.: 48, 49, 50, 51.)

Ten tweede het datatype van stromen over A. Deze wordt analoog aan product gedefinieerd: in plaats van de *destructoren* exl en exr hebben we nu $head$ en $tail$, en in plaats van operator $\vartriangle$ schrijven we nu $\_\Box\_$ (bij gebrek aan een betere notatie). De definitie luidt als volgt. Een viertal $(C,\ head\colon C \to A,\ tail\colon C \to C,\ \Box)$ is (een implementatie van) het datatype *stroom* over A wanneer:

$$h = f \Box g \qquad \equiv \qquad head \circ h = f \ \text{ en } \ tail \circ h = h \circ g \quad .$$

In dat geval schrijven we $Strm\, A$ voor C. Het rechterlid van deze equivalentie geeft op functie-nivo een definitie van $h\colon \ldots \to Strm\, A$ met inductie naar de $head, tail$-afbraak van het resultaat:

$$head(h(x)) = f(x) \qquad \text{en} \qquad tail(h(x)) = h(g(x)) \quad .$$

Zo'n h, kortweg genoteerd met $f \Box g$, heeft hetvolgende effect:

$$h(x) \quad = \quad [f(x), f(g(x)), f(g(g(x))), \ldots, f(g^i(x)), \ldots] \quad ,$$

waarbij het rechterlid een notatie is voor een stroom waarvan voor $i = 0, 1, 2, \ldots$ de uitkomsten onder $head \circ tail^i$ staan opgesomd.

Definieer voorts operatie $Strm$ op functies zó dat $Strm\, f\ (xs)$ de functie f op elk element in xs toepast, en de stroom-structuur behoudt:

$$head \circ Strm\, f = f \circ head \qquad\qquad \text{en} \qquad tail \circ Strm\, f = Strm\, f \circ tail$$

ofwel

$$head(Strm\, f\ (xs)) = f(head(xs)) \qquad \text{en} \qquad tail(Strm\, f\ (xs)) = Strm\, f\ (tail(xs)).$$

Dan is $Strm$ een functor:

$$\begin{array}{l} Strm\, f\colon\ Strm\, A \to Strm\, B, \qquad \text{voor } f\colon\ A \to B \\ Strm\, f \circ Strm\, g \;=\; Strm(f \circ g) \\ Strm\, id \;=\; id \quad . \end{array}$$

(Oef.: 52, 53.)

23 Syntaxis voor datatypen. Huidige programmeertalen hebben een heel verschillende syntaxis voor het product en de som, ondanks het feit dat die elkaars duale zijn, en ze verschaffen ook weer een heel andere syntaxis voor de overige datatypen. Een syntaxis die alle op gelijke voet behandeld, is al in 1988 door G.C. Wraith gesuggereerd:

datatype $A \times B$ **with** $\_\vartriangle\_$ **has destructors**

$\quad exl\colon\ A \times B \to A$

$exr\colon\ A \times B \to B \quad .$

datatype $A + B$ **with** $\_ \triangledown \_$ **has constructors**

$inl\colon\ A \to A + B$

$inr\colon\ B \to A + B \quad .$

Op een vrij eenvoudige manier, die we hier niet verder uitleggen, kan uit deze tekst het type van de **with**-operatoren $\vartriangle$ en $\triangledown$ afgeleid worden, alsmede de karakterisering, en de definitie van $f \times g$ en $f + g$. De datatypen van natuurlijke getallen, lijsten en stromen zien er met deze syntaxis als volgt uit:

datatype nat **with** $\_\mathrm{loop}\_$ **has constructors**

$zero\colon\ () \to nat$

$succ\colon\ nat \to nat \quad .$

datatype $Lst\,A$ **with** $\_\mathrm{foldr}\_$ **has constructors**

$nil\colon\ () \to Lst\,A$

$cons\colon\ A \times Lst\,A \to Lst\,A \quad .$

datatype $Strm\,A$ **with** $\_\Box\_$ **has destructors**

$head\colon\ Strm\,A \to A$

$tail\colon\ Strm\,A \to Strm\,A \quad .$

Ook hier kan het type van loop, foldr, $\Box$ en de karakterisering, en de definitie van $Lst\,f$ en $Strm\,f$ mechanisch afgeleid worden uit de tekst. (Oef.: 54.)

De vorm van karakterisering die voor het product en de som geldig is, is onvoldoende voor datatypen met beperkingen, zoals "gesorteerde lijsten" (lijsten beperkt tot degene die gesorteerd zijn), en voor datatypen met wetten, zoals "lijsten met een commutatieve lijstvormer" (zodat de plaats van een element in een lijst niet terzake doet). Voor de karakterisering van dit soort datatypen heeft de categorie-theorie ook weer begrippen en methoden die een hulp bieden.

24 Over categorie-theorie. Categorie-theorie is heel algemeen; veel stelsels van "objecten" en "morfismen" zijn een categorie (omdat er een associatieve compositie met identiteit is). Door z'n algemeenheid zijn de stellingen ook heel zwak; maar wel veel voorkomend en heel bruikbaar! Het formalisme van de categorie-theorie vergt enige gewenning. Wij zijn vaak gewend ons op punt-nivo uit te drukken, met expliciete argumenten, in plaats van op functie-nivo met alleen samenstellingen van functies om nieuwe functies te vormen, zoals in de categorie-theorie nodig is.

Voor geïnteresseerden volgt zodadelijk nog een korte literatuurlijst voor categorie-theorie en datatypen. In [3, 5] wordt de karakterisering van datatypen in het algemeen onderzocht; de daaruit verkregen wetten blijken nuttig te zijn voor transformationeel programmeren. De overige literatuur richt zich niet speciaal op datatypen.

Dankbetuiging. Jan Kuper's commentaar heeft tot een aanzienlijke verbetering van paragraaf 5 en 7 geleid.

[1] M. Barr and C. Wells. *Category Theory for Computing Science.* Prentice Hall, 1990.

[2] M.M. Fokkinga. Calculate categorically! *Formal Aspects of Computing*, 4(4):673–692, 1992.

[3] M.M. Fokkinga. *Law and Order in Algorithmics.* PhD thesis, University of Twente, Dept INF, Enschede, The Netherlands, 1992.

[4] R. Goldblatt. *Topoi — the Categorial Analysis of Logic*, volume 98 of *Studies in Logic and the Foundations of mathematics.* North-Holland, 1979. Alleen hoofdstuk 2, 3 en 9.

[5] G. Malcolm. Data structures and program transformation. *Science of Computer Programming*, 14(2–3):255–280, September 1990.

[6] B.C. Pierce. *Basic Category Theory for Computer Scientists.* MIT Press, Cambridge, Ma, 1991.

[7] D. Roorda. Boekbespreking van [6]. *Informatie*, jaargang 35, nr 10, 637–638, oktober 1993.

Oefeningen

Onderstaande oefeningen helpen wellicht bij de bestudering van de tekst (ze vragen namelijk om gestelde beweringen te bewijzen, ze laten verdere gevolgen zien, of ze tonen onderliggende moeilijkheden), en vergroten bovendien de vaardigheid in het werken op functie-nivo (wat zo typerend is voor categorie-theoretische beschouwingen).

25 Ad §6. Bewijs de bewering dat "vanwege de eerste regel, de laatste drie gelijkheden volgen uit de voorgaande drie, en omgekeerd".

26 Ad §6. Bewijs dat de in paragraaf 3 en 4 gegeven implementaties van het product alle isomorf zijn.

27 Ad §7. Ga na of bewijs dat de gestelde intuïtie (twee beweringen) gelden voor ieder van de implementaties uit paragraaf 3 en 4.

28 Ad §7. Geef een implementatie die wel aan de tweede bewering van de intuïtie voldoet maar niet aan de eerste, en laat zien dat die implementatie niet isomorf is met de standaard implementatie.

29 Ad §8. In de formele definitie van 'product' komen variabelen f, g, h voor. Leid uit de formules af wat de typen van deze variabelen zijn.

30 Ad §9. Bewijs de volgende wet voor een product $(A \times B, exl, exr, \vartriangle)$ van A en B:

$$exl \circ f = exl \circ g \;\text{ en }\; exr \circ f = exr \circ g \qquad \Rightarrow \qquad f = g \qquad\qquad \text{prod-}\textsc{Uniq}$$

Laat voorts zien dat deze wet een directe formalisatie is van de tweede bewering van de intuïtie over het (cartesisch) product van A en B.

31 Ad §9. Laat zien dat prod-CHARN equivalent is met het tweetal wetten prod-SELF en prod-UNIQ.

32 Ad §9. Formuleer (en bewijs) wet prod-FUSION op punt-nivo.

33 Ad §10. Laat zien dat uit de gelijkheden:

$$\varphi \circ \varphi' = id \qquad \text{en} \qquad \varphi' \circ \varphi = id$$

en

$$\begin{array}{lcllcl} exl & = & exl' \circ \varphi \qquad & exl' & = & exl \circ \varphi' \\ exr & = & exr' \circ \varphi & exr' & = & exr \circ \varphi' \end{array}$$

volgt dat ook:

$$f \vartriangle g \;=\; \varphi' \circ f \vartriangle' g \qquad\qquad f \vartriangle' g \;=\; \varphi \circ f \vartriangle g \quad .$$

Het is dus niet nodig (maar wel voor de hand liggend) om in de definitie van isomorfie op te nemen dat $f \vartriangle g$ en $f \vartriangle' g$ in elkaar uitgedrukt kunnen worden.

34 Ad §10. Bewijs dat $\varphi, \varphi' = exl \vartriangle' exr,\; exl' \vartriangle exr'$ de enige functies zijn via welke de producten $(C, exl, exr, \vartriangle)$ en $(C, exl', exr', \vartriangle')$ isomorf zijn. Dat wil zeggen, toon aan dat als ze ook isomorf zijn via ψ, ψ', dan $\varphi, \varphi' = \psi, \psi'$.

35 Ad §10. Bewijs dat een implementatie die isomorf is aan één die aan prod-CHARN voldoet, zelf ook aan prod-CHARN voldoet.

36 Ad §13. Het is mogelijk een intuïtie te geven omtrent de disjoint union die, veel duidelijker dan de intuïtie van paragraaf 13, de analogie (dualiteit — zie paragraaf 20) met het cartesisch product weergeeft. Probeer die intuïtie te achterhalen.

37 Ad §15. Leid de types af van f, g, h en rechter- en linkerlid in:

$$(f \triangledown g) \vartriangle (h \triangledown j) \quad = \quad (f \vartriangle h) \triangledown (g \vartriangle j) \quad .$$

Druk de gelijkheid uit op punt-nivo; maak eventueel de keuze $f = g = exl$ en $h = j = exr$.

38 Ad §15. Geef een alternatief bewijs van $(f \triangledown g) \vartriangle (h \triangledown j) = (f \vartriangle h) \triangledown (g \vartriangle j)$, waarbij de eerst gebruikte wet prod-CHARN is, in plaats van som-CHARN.

39 Ad §17. (Om enige vaardigheid te ontwikkelen in het werken op functie-nivo.) Definieer, bij gegeven operator $\oplus$, operator $*$ op functies door:

$$(f * g)(x) \quad = \quad ((a \oplus b), z) \text{ where } (a, y) = f(x),\ (b, z) = g(y) \quad .$$

Geef een alternatieve maar gelijkwaardige definitie "$f * g = \ldots$" geheel op functie-nivo; dus geheel in termen van id en $\circ$ en de functies en operatoren van het product. (Wenk: gebruik en definieer een hulpfunctie $assoc$: $(A \times (B \times C)) \to ((A \times B) \times C)$.)

Bewijs vervolgens op functie-nivo dat operatie $*$ associatief is, wanneer $\oplus$ dat is.

40 Ad §17. Laat een gerichte graaf gegeven zijn: een verzameling van knopen $(A, B, \ldots)$ en kanten (een kant is een paar van knopen). Een pad in de graaf is een rij "aansluitende" kanten. Definieer nu objecten, morfismen, en compositie en identiteiten als volgt:

een object	is	een knoop in de graaf
een morfisme	is	een pad in de graaf
$f \circ g$	$=$	het pad bestaande uit pad g voortgezet met pad f
id_A	$=$	het lege pad (bij knoop A) .

Laat zien dat hierdoor een categorie gedefinieerd wordt, dus bewijs de associativiteit van de compositie, en de neutraliteit van id voor de compositie. Hoe zou de typering gedefinieerd moeten worden, dat wil zeggen, hoe definieert u f: $A \to B$ voor een pad f en knopen A, B? Speelt de typering een rol in de definitie van de compositie, of zou die daar een rol in moeten spelen?

41 Ad §18. Ga na dat in de formele bewijzen *nergens* andere eigenschappen van compositie $\circ$ en identiteit id zijn gebruikt dan associativiteit en neutraliteit van id voor $\circ$. Ga na dat deze beide eigenschappen *ergens* zijn gebruikt.

42 Ad §19. (Om vaardigheid te krijgen in het calculationeel redeneren, in het bijzonder aan de hand van een universal property.) Laat A, B en p: $A \to B$ gegeven zijn, alsmede een unaire operatie $\_/p$ op morfismen (functies). Beschouw de universal property:

$$g = f/p \quad \equiv \quad f = g \circ p \quad . \qquad \text{CHARN}$$

Bewijs dat uit Charn ook de volgende wetten volgen:

$$f = f/p \circ p \qquad \textsc{Self}$$

$$id = p/p \qquad \textsc{Id}$$

$$f \circ p = g \circ p \quad \Rightarrow \quad f = g \qquad \textsc{Uniq}$$

$$f \circ (g/p) = (f \circ g)/p \quad . \qquad \textsc{Fusion}$$

Bewijs ook dat Self en Uniq samen Charn impliceren. Zie ook oefening 43.

43 Ad §20. Dualiseer oefening 42. Dit levert iets als: "Laat Y, Z en $q\colon X \to Z$ gegeven zijn, alsmede een unaire operatie $q\backslash\_$ op morfismen (functies). Beschouw de universal property ... ".

Bewijs ook de volgende wet: $(q\backslash f)/p = q\backslash(f/p)$. Vergelijk uw bewijs met dat van paragraaf 15.

44 Ad §20. Laat A een verzameling zijn en veronderstel dat:

$$\forall B \exists! f :: \qquad f\colon A \to B \quad .$$

Hierbij betekent $\exists!$ "er bestaat precies één". Toon aan dat A de lege verzameling is. (Wenk: Identificeer een functie f met zijn graaf $\{(x, f(x)) |\ x \in \mathrm{dom}(f)\}$. De graaf van een functie is leeg wanneer het domein van de functie leeg is.)

Beschouw nu de dualisering ervan; dus, laat A een verzameling zijn en veronderstel dat:

$$\forall B \exists! f :: \qquad f\colon B \to A \quad .$$

Toon aan dat A een singleton is (een verzameling met één element), en dat iedere singleton A hieraan voldoet. Dus "A is *de* enige lege verzameling" is duaal aan "A is *een* van de vele singletons".

45 Ad §21. Bewijs dat de volgende twee eigenschappen van $\times$ tezamen de operatie volledig definiëren:

$$\begin{aligned} exl \circ f \times g &= f \circ exl \\ exr \circ f \times g &= g \circ exr \quad . \end{aligned}$$

46 Ad §21. Bewijs dat $\times$ een functor is, dat wil zeggen, operatie $\times$ heeft geen interactie met compositie en behoudt de identiteit.

47 Ad §21. Het is wellicht beter om operatie $\times$ een *twee-plaatsige* functor te noemen, omdat het een twee-plaatsige operatie is. Definieer wanneer een n-plaatsige operatie F een n-plaatsige functor is. Hoe luidt deze definitie voor het geval dat $n = 1$? Hoe wordt, bij $n = 1$, de eigenschap "F heeft geen interactie met compositie" gewoonlijk genoemd?

Definieer $I\!I$ op verzamelingen en functies door:

$$\begin{array}{rcl} I\!I(A) & = & A \times A \\ I\!I(f) & = & f \times f \quad . \end{array}$$

Bewijs dat $I\!I$ een 1-plaatsige functor is. Merk op dat bovenstaande definities zinvol zijn in willekeurige categorie.

48 Ad §22. Bewijs (met inductie naar n) dat voor $h = f \,\mathrm{foldr}\, g$:

$$h(\mathit{cons}(a_{n-1}, \ldots \mathit{cons}(a_0, \mathit{nil}()))) \quad = \quad g(a_{n-1}, \ldots g(a_0, f())) \quad .$$

Merk op dat het effect van functie $f \,\mathrm{foldr}\, g$ als volgt omschreven kan worden:

> Definieer de *canonieke termen* van type $\mathit{Lst}\,A$ als de termen van de vorm $\mathit{cons}(a, \ldots\ \mathit{cons}(a', \mathit{nil}()))$. Dan is het effect van $f \,\mathrm{foldr}\, g$ op een Lst-waarde xs dat in de canonieke term voor xs de nil vervangen wordt door f en elke cons door g.

49 Ad §22. Druk functie $\mathit{Lst}\,f$ uit met behulp van foldr (zonder nog recursie te gebruiken). Wenk: de gegeven recursieve definitie van $\mathit{Lst}\,f$ heeft de vorm van het rechterlid van de karakterisering van lijsten over A (waarin ook foldr gedefinieerd wordt).

50 Ad §22. Druk de volgende functies uit met behulp van foldr (en $\mathit{Lst}\,\_$), zonder recursie te gebruiken:

sum	:	de som van een lijst van getallen
prod	:	het product van een lijst van getallen
concat	:	de concatenatie van een lijst van lijsten
and	:	de conjunctie van een lijst van waarheidswaarden
or	:	de disjunctie van een lijst van waarheidswaarden
lth	:	de lengte van een lijst
rev	:	de omgekeerde van een lijst
inits	:	de (een) lijst van alle initiële delen van een lijst .

Wenk: vind geschikte substituties voor f en g zó dat $g(a, \ldots, g(a', f()))$ de gewenste uitkomst is bij lijst $\mathit{cons}(a, \ldots, \mathit{cons}(a', \mathit{nil}()))$; dan is $f \,\mathrm{foldr}\, g$ de gewenste functie. Een andere manier is om recursievergelijkingen op te stellen voor de gewenste functie, zeg h, die de vorm hebben van het rechterlid van de karakterisatie van lijsten; het linkerlid van die karakterisatie geeft dan de foldr-vorm voor h.

Druk nu ook de volgende functies uit op functie-nivo:

tails	:	de (een) lijst van alle staartstukken van een lijst
segs	:	de (een) lijst van alle segmenten van een lijst
filter p	:	de lijst van alle elementen van een lijst die aan p voldoen .

Gebruik zonodig de al eerder gegeven functies.

51 Ad §22. Bewijs:

$$\mathit{nil} \text{ foldr } \mathit{cons} = \mathit{id}$$
$$f \text{ foldr } g \circ \mathit{Lst}\, h = f \text{ foldr } (g \circ h \times \mathit{id}) \quad .$$

Welke eigenschappen worden in de bewijzen gebruikt? (Merk op dat er geen inductie nodig is!) Bewijs ook:

$$\begin{array}{lll} h(\mathit{nil}(\,)) = f() & \text{en} & h(\mathit{cons}(x, xs)) = g(x, h(xs)) \quad \text{en} \\ h'(\mathit{nil}(\,)) = f() & \text{en} & h'(\mathit{cons}(x, xs)) = g(x, h'(xs)) \\ \Rightarrow & & \\ h = h' \quad . & & \end{array}$$

Ook hier is inductie niet expliciet nodig.

52 Ad §22. Laat *iterate* f de functie zijn met:

$$\mathit{iterate}\, f(x) \quad = \quad [x, f(x), f(f(x)), \ldots] \quad .$$

Druk *iterate* f uit met behulp van $\Box$.
Laat predikaten D_k gegeven zijn, met $D_k(x) \equiv$ x is deelbaar door k . Druk nu uit op functie-nivo: de lijst van alle priemgetallen vanaf n . Gebruik zonodig de functies uit de vorige oefeningen.

53 Ad §22. Dualiseer oefeningen 48 en 49.

54 Ad §23. Probeer te achterhalen hoe de karakterisering van een datatype wordt bepaald door de declaratie ervan.
Probeer de karakterisering van het datatype *nat* op te stellen. Wat is de uitkomst van $f$loop$g$ op argument $\mathit{succ}(\mathit{succ}(\mathit{succ}(\mathit{zero}())))$?
