ir. J.M. VAN THIEL

ACP geeft geen definitie van 'proces' maar werkt er alleen mee. Inhoudelijk kan een taak alles voorstellen, van computerhandelingen tot complexe organisaties.

Amsterdamse processen

Fundamenteel onderzoek bij Academie voor Informatica

Het goede nieuws, voor ontwerpers, is dat er veelbelovende wiskundige technieken bestaan om processen mee te beschrijven. Het slechte nieuws is dat die zich vooral nog in het stadium van wetenschappelijk onderzoek bevinden en moeilijk toegankelijk zijn. Aan het Centrum voor Wiskunde en Informatica in Amsterdam is, en wordt, in samenwerking met enkele universiteiten een theorie ontwikkeld die men 'Algebra of Communicating Processes' noemt. Eind april is onder verantwoordelijkheid van de Academie voor Informatica een cursus over dit onderwerp gegeven voor geïnteresseerden uit diverse ondernemingen. Het volgende wil een globale indruk geven van de op de twee lesdagen behandelde algebra. Niet al het besprokene zal hierbij worden vermeld en de streng wiskundige beschrijving blijft onderbelicht.

Op 26 en 27 april jongstleden gaf de Academie voor Informatica (AvI) een cursus procesalgebra aan het Centrum voor Wiskunde en Informatica (CWI) in Amsterdam. Deze academie is een samenwerkingsverband van enkele universiteiten en instellingen dat door middel van korte opleidingen relevant geachte wetenschappelijke kennis onder de aandacht wil brengen van het bedrijfsleven.

De cursus is een van de vier modulen software-algebra, dat zelf weer een van de drie onderwerpen is die gerangschikt worden onder de naam softwaretechnologie. Tijdens de twee dagen stond de theorie *Algebra van Communicerende Processen* (ACP) centraal, die het Centrum voor Wiskunde en Informatica in Amsterdam in samenwerking met enkele universiteiten ontwikkelt. Deze algebra is verwant aan de *Calculus of Communicating Systems* en de *Communicating Sequential Processes* die door Milner respectievelijk Hoare zijn opgezet.

ACP

Het wetenschappelijke collectief – waarin behalve het CWI de Universiteit van Amsterdam en de Rijks Universiteit Utrecht zijn betrokken – is in 1982 begonnen met de ontwikkeling van ACP. Initiatoren waren J.A. Bergstra en J.W. Klop. Hun werk wordt thans voortgezet door de CWI-groep onder leiding van J.C.M. Baeten.

De theorie onderscheidt zich niet alleen inhoudelijk van de andere theorieën. Een belangrijk verschil ligt in de gevolgde methodologie: het opstellen van axioma's en expliciete bewijsregels die als zodanig onafhankelijk zijn van een bepaald domein. Deze benadering heeft een aantal duidelijke voordelen, juist ook voor de eventuele toepassing. ACP gaat uit van atomaire processen of taken die niet nader zijn gedefinieerd en die door middel van operatoren op verschillende manieren kunnen worden samengesteld. Sequentiële compositie, non-deterministische keuze en een parallelle samenvoeging zijn enkele vormen van samenstellingen.

PROCES

In het algemeen wordt de beschrijving van een proces gegeven in de vorm van een specificatie die bestaat uit een stelsel algebraïsche vergelijkingen. Daarbij kan het tevens gaan om communicatie tussen twee andere taken. Door gebruik te maken van recursie kunnen bovendien in een aantal gevallen oneindige processen worden beschreven met behulp van eindige vergelijkingen.

Een andere eigenschap van een taak is dat hij kan zijn geblokkeerd (*deadlock*). Door te blokkeren met behulp van de encapsulatieoperator kunnen zekere restricties in de procesgang worden aangeven. Een speciale abstractieoperator maakt het mogelijk om deeltaken te verbergen. Het grote belang hiervan is dat (soms) berekend kan worden of een inwendige specificatie gelijk is aan de uitwendige, met andere woorden: of een bepaalde taak inderdaad voldoet aan de door de omgeving gestelde eisen.

De aanduiding *proces* kan in principe van alles voorstellen. Zo zijn de toelevering, het intern transport en het vervoer ter verzending uit het fabrieksschema van figuur 1 zeker processen [1a]. Ook de besturingsfuncties en statusmeldingen, zoals de pijlen die weergeven, kunnen gezien worden als taken. Hetzelfde geldt voor de fabricagecellen en het besturingsblok en eveneens voor de in- en uitgaande pijlen, die de betrekkingen met de omgeving voorstellen.

AXIOMA'S

In het toepassingsgebied van de algebra bestaat alles uit een atomaire taak of uit een samenstelling van zulke taken. Bij ACP is een atomair proces niet nader gedefinieerd. Sterker nog, ook aan de operatoren die de samenstellingen representeren wordt geen specifieke betekenis gegeven. De hele theorie van ACP bestaat in wezen uit een stelsel axioma's. Op basis daarvan is het mogelijk om, met behulp van algemene en specifieke bewijsregels, stellingen af te leiden en gelijkheid of ongelijkheid van formules aan te tonen. ACP wordt dus ontwikkeld als een zuiver wiskundige

theorie en het hele instrumentarium, geclassificeerd volgens het schema van tabel 1, is te vinden in de tabellen 2 en 3 (zie ook [1b]).

De genoemde benadering heeft als grote voordeel dat deze in principe volledig onafhankelijk blijft van elke visie van wat nu een proces is. In de praktijk moet deze uitspraak gedeeltelijk terug worden genomen: de axioma's zijn wel degelijk ontworpen met een schuin oog naar domeinen zoals die ook in de *Calculus of Communicating Systems* van Milner en de CSP (*Communicating Sequential Processes*) van Hoare (zie bij voorbeeld [2]) worden bestudeerd. Maar de bewijsvoering en de berekeningen maken hier verder geen gebruik van.

ALGEBRA

Voor de modellen van ACP die in de leerboeken [2,3] worden geïntroduceerd – waaronder term-, projectieve-limiet- en graafmodel – geldt hetzelfde. Ook dit zijn wiskundige constructies en geen empirische entiteiten. Met name het graafmodel kan echter wel goed dienen als representatie van 'echte' processen. Maar men zij gewaarschuwd; ook dan nog is het niet helemaal zoals het lijkt: deze grafen zijn niet de vertrouwde toestandsdiagrammen die het gedrag van eindige-toestandsautomaten vastleggen.
Het belangrijkste cursusmateriaal bestond uit de boeken *Procesalgebra* van Baeten [2] en het meer recente *Process Algebra* van Baeten/Weijland [3]. Bij het practicum werd gewerkt met een implementatie van de formele specificatietaal *PSF*. Dat gebeurde aan de hand van twee artikelen van Mauw en Veltink, *An Introduction to PSF* [4] en *A Process Specification Formalism* [5]. Verder werd een analyse behandeld van een toepassing op een systolisch systeem, een configuratie van identieke elementen die met elkaar communiceren (zie [1c]). Sommige van de daarbij besproken onderwerpen worden in dit artikel slechts even aangestipt.

ATOMAIR

ACP kent alleen niet nader gedefinieerde atomaire taken die intuïtief kunnen worden gerepresenteerd door lijnen (pijlen) van gerichte grafen die de verschillende knooppunten van het systeem verbinden. Aan basis van ACP staan de axioma's *A1* tot *A5* uit tabel 2 die de Basis Proces Algebra (BPA) vormen. Deze kwamen dan ook als eerste aan de orde tijdens de cursus van de Academie voor Informatica. BPA onderscheidt twee samenstellingsmethoden: sequentiële compositie en non-deterministische keuze. Deze worden ook wel aangeduid met de termen vermenigvuldiging en optelling. Overeenkomstig geeft in formules de punt een sequentiële compositie weer – waarbij, zoals gebruikelijk, deze eventueel ook kan vervallen – en indiceert het plus-teken de non-deterministische keuze. Grafisch is een en ander geschetst in figuur 2.
De axioma's *A1* tot *A5* uit tabel 2 bevatten variabelen *x*, *y* en *z* die voor willekeurige processen staan. *A3* betekent dan bij voorbeeld dat een keuze tussen twee gelijke taken hetzelfde is als het ontbreken van een keuze.
A4 verdient een nadere toelichting. Normaliter zullen twee verschillende *x* en *y* in het algemeen het systeem in een verschillende toestand brengen. Desondanks kan een keuze toch in eenzelfde knoop van de graaf uitkomen, namelijk als daar in beide gevallen dezelfde *z* op volgt. Dit is aange- ▷

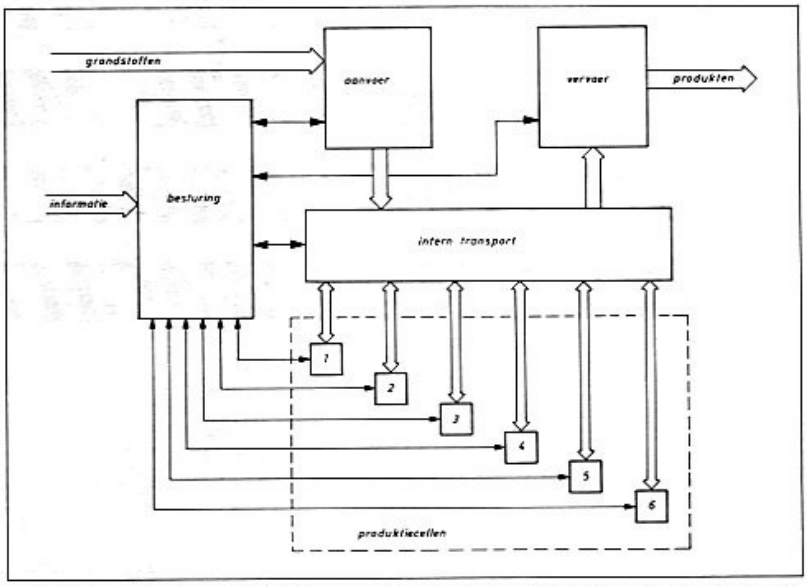

Fig. 1. In bij voorbeeld een fabriek kunnen niet alleen zaken als toelevering, transport, produktie en uitlevering als processen worden gezien maar ook de communicatie- en besturingsfuncties.

Tabel 1. ACP wordt gevormd door een raamwerk van axioma's en bewijsregels die bestaat uit de hier afgebeelde componenten.

A PROCESS SPECIFICATION AND VERIFICATION FRAMEWORK	
Basic Process Algebra	A1-5
Deadlock	A6,7
Communication Function	C1-3
Merge with Communication	CM1-9
Encapsulation	D1-4
Silent Step	T1-3
Silent Step: Auxiliary Axioms	TM1,2; TC1-4
Abstraction	DT; TI1-5
Projection	PR1-6
Hand Shaking	HA
Standard Concurrency	SC
Expansion Theorem	ET
Alphabet Calculus	CA
Recursive Definition Principle	RDP
Recursive Specification Principle	RSP
Weak Approximation Induction Principle	AIP^-
Cluster Fair Abstraction Rule	CFAR

▷geven in het voorbeeld van figuur 3. Zo'n knoop is dus niet zomaar te identificeren met een eenduidige, vooraf bekende systeemtoestand.

RUSSISCH ROULETTE

De analogie van axioma *A4*:

$$x(y + z) = xy + xz$$

geldt niet in BPA en het uitgebreidere ACP (zie figuur 4). Wat ruw, maar wel duidelijk, is dit te motiveren met het volgende. Stel iemand speelt Russische roulette. Er kan natuurlijk al dan niet een kogel in de kamer zitten, maar zodra de trekker wordt overgehaald kan de schutter niet meer bepalen wat er gebeurt. De tweede graaf in figuur 4 geeft die taak wel correct weer, in tegenstelling tot de eerste. In de laatstgenoemde situatie kan immers na afloop van het *x* (het overhalen van de trekker) nog gekozen kan worden tussen *y* en *z*.

Twee taken zijn in het algemeen gelijk als op elk punt in de afhandeling van de ene precies dezelfde processen kunnen volgen als bij de andere en vice versa. Het zal duidelijk zijn dat dit in figuur 3 wel het geval is en in figuur 4 niet. In het graafmodel wordt de genoemde eigenschap uitgedrukt met de term *bisimulatie*. De twee grafen in figuur 3 zijn weliswaar niet hetzelfde maar zij bisimuleren.

EINDIG

De drie grafen uit figuur 5 bisimuleren ook. Zij stellen het continu herhaalde proces *a* voor. Van deze drie is echter van de cyclische graaf geen duidelijke wortel aan te geven, in tegenstelling tot de andere twee. Dit leidt tot een probleem bij het optellen van deze graaf, voorgesteld door $a(\omega)$, en een andere taak *b*.

Om aan te geven dat de keuze tussen een $a(\omega)$ en *b* niet hetzelfde is als een $a(\omega)$ die na elke herhaling met een keuze voor *b* kan worden verlaten, moet de graaf één keer worden 'uitgerold' (fig. 6). Het probleem is dat een lus geen welbepaalde wortel heeft. Uit tweede en derde graaf in figuur 5 valt voor elke knoop op te maken hoe, vanaf het begin, het proces is verlopen en voor de eerste is dat niet mogelijk.

Het aantal stappen dat een taak heeft afgelegd wordt uitgedrukt met de term *projectie* en de kenmerken daarvan zijn precies gedefinieerd in de regels *PR1* tot *PR4* van ta-▷

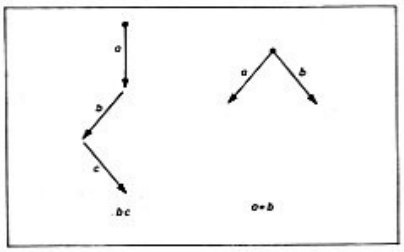

Fig. 2. Sequentiële compositie en non-deterministische keuze.

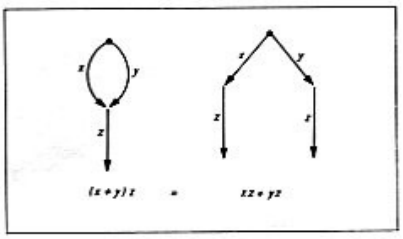

Fig. 3. Twee bisimulerende grafen: hoewel niet hetzelfde representeren zij in alle gevallen dezelfde processen en de volgorde daarvan: (x + y)z = xz + yz.

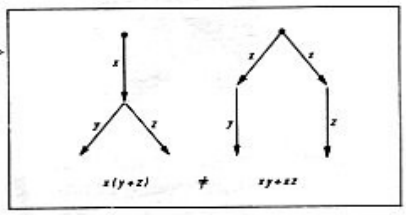

Fig. 4. Russisch roulette in een graaf: na het overhalen van de trekker (x) treedt ofwel y ofwel z op. Na x is kiezen echter niet meer mogelijk zodat alleen de tweede graaf de juiste is: x(y + z) ≠ xy + xz.

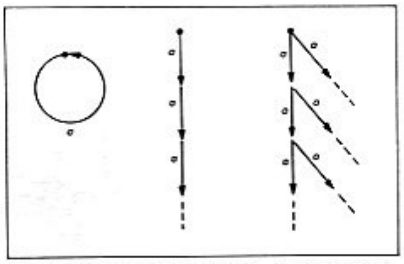

Fig. 5. Herhalende processen. Alleen in b *en* c *kan van elke knoop het eraan voorafgaande traject worden bepaald.*

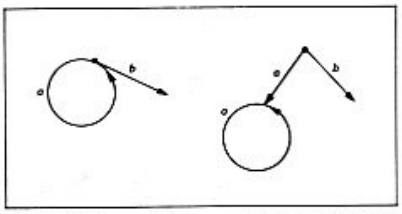

Fig. 6. Het verschil tussen een cyclische graaf en een graaf die een keer is uitgerold komt tot uiting in de recursieve specificaties.

Tabel 2. Bewerkingen en de bijbehorende regels binnen ACP.

ACP_τ			
$x+y=y+x$	A1	$x\tau=x$	T1
$(x+y)+z=x+(y+z)$	A2	$\tau x=\tau x+x$	T2
$x+x=x$	A3	$a(\tau x+y)=a(\tau x+y)+ax$	T3
$(x+y)z=xz+yz$	A4		
$(xy)z=x(yz)$	A5		
$x+\delta=x$	A6		
$\delta x=\delta$	A7		
$a\|b=b\|a$	C1		
$(a\|b)\|c=a\|(b\|c)$	C2		
$\delta\|a=\delta$	C3		
$x\|\|y=x\lfloor y+y\lfloor x+x\|y$	CM1		
$a\lfloor x=ax$	CM2	$\tau\lfloor x=\tau x$	TM1
$ax\lfloor y=a(x\|\|y)$	CM3	$\tau x\lfloor y=\tau(x\|\|y)$	TM2
$(x+y)\lfloor z=x\lfloor z+y\lfloor z$	CM4	$\tau\|x=\delta$	TC1
$ax\|b=(a\|b)x$	CM5	$x\|\tau=\delta$	TC2
$a\|bx=(a\|b)x$	CM6	$\tau x\|y=x\|y$	TC3
$ax\|by=(a\|b)(x\|\|y)$	CM7	$x\|\tau y=x\|y$	TC4
$(x+y)\|z=x\|z+y\|z$	CM8		
$x\|(y+z)=x\|y+x\|z$	CM9	$\partial_H(\tau)=\tau$	DT
		$\tau_I(\tau)=\tau$	TI1
$\partial_H(a)=a$ if $a\notin H$	D1	$\tau_I(a)=a$ if $a\notin I$	TI2
$\partial_H(a)=\delta$ if $a\in H$	D2	$\tau_I(a)=\tau$ if $a\in I$	TI3
$\partial_H(x+y)=\partial_H(x)+\partial_H(y)$	D3	$\tau_I(x+y)=\tau_I(x)+\tau_I(y)$	TI4
$\partial_H(xy)=\partial_H(x)\cdot\partial_H(y)$	D4	$\tau_I(xy)=\tau_I(x)\cdot\tau_I(y)$	TI5

▷bel 3. De eerste projectie van de $a(\omega)$ is a, de tweede aa, de vierde $aaaa$ etcetera. En in samenstellingen wordt dat bij voorbeeld:

$\pi(4)$ van $(a(\omega) + b) = (aaaa + b)$

Met het begrip projectie ontstaat ook een criterium om te bepalen of een proces eindig is of niet. Als, bij voldoend grote n, de n^e projectie gelijk wordt aan het proces zelf, is het eindig en anders niet. De projectieve limiet is, net als de structuur van grafen modulo bisimulatie, een model voor de algebra.

ONEINDIG

De hele filosofie achter BPA en het uitgebreidere ACP is het opstellen van een formalisme met het doel hier processen mee door te kunnen rekenen. Oneindige processen zijn natuurlijk heel belangrijk en het heeft dan ook sterk de voorkeur om die aan te kunnen geven zonder oneindig veel variabelen te hoeven gebruiken. Dit kan, in bepaalde gevallen, met behulp van recursie. Neem bij voorbeeld de vergelijking:

$X = aX$

Bij substitutie van de tweede X door de rechter term ontstaat:

$X = aaX$

En bij de volgende substitutie:

$X = aaaX$

Dit kan oneindig lang doorgaan, met als resultaat steeds een rij taken a waar weer een a op kan volgen, enzovoort. De recursieve vergelijking $X = aX$ heeft dus het oneindige proces $a(\omega)$ als oplossing (zie figuur 5).

Hetzelfde substitutiemechanisme kan worden uitgevoerd op de vergelijking $X = Xa$. Dat levert opnieuw een oneindige rij van sequenties a op, die ditmaal wel ergens eindigt maar die geen gedefinieerd begin heeft.

NIETSZEGGEND

De laatstgenoemde vorm, zonder eenduidige beginsituatie, wordt niet als proces beschouwd, althans in ACP. Elke recursieve variabele in een ACP-specificatie moet worden voorafgegaan door een atomaire taak dan wel tot een dergelijke vorm zijn te herschrijven. Deze voorwaarde wordt omschreven met *gedekt* (*guarded*) en de restrictie voorkomt dat er vergelijkingen ontstaan als $X = X$ die niets zeggen omdat ze altijd gelden.

Met recursie kunnen de grafen uit figuur 6 worden gespecificeerd door middel van respectievelijk:

$X = aX + b$

en

$X = aX$

$Y = X + b$

en nu blijkt duidelijk het verschil in de formules. Overigens is het conventie om recursieve variabelen met een hoofdletter te schrijven.

Bovenstaande bewering, dat de recursieve specificaties de aangegeven processen representeren, is natuurlijk nog slechts gefundeerd op intuïtie. Om dit echt te bewijzen zijn onder andere de bewijsregels *RDP*, *RSP* en *AIP*$^-$ uit tabel 3 nodig.

FORMEEL

Een proces is in het algemeen goed te kenschetsen door het stap voor stap te beschrijven. Met name *AIP*$^-$ legt dan de verbinding tussen zo'n notatie in het projectieve-limietmodel en de recursieve specificatie in ACP.

Het is niet altijd gemakkelijk om de juistheid van zo'n verbinding aan te tonen. Als het al mogelijk is om een bewijs te leveren dan is dit vaak erg lang en zitten er de nodige haken en ogen aan het opstellen van de bewijsregels. Veel onderzoek richt zich dan

Tabel 3. Extra eigenschappen van ACP en een aantal bewijsregels.

REMAINING AXIOMS AND RULES

$\pi_1(ax)=a$	PR1
$\pi_{n+1}(ax)=a\cdot\pi_n(x)$	PR2
$\pi_n(a)=a$	PR3
$\pi_n(x+y)=\pi_n(x)+\pi_n(y)$	PR4
$\pi_n(\tau)=\tau$	PR5
$\pi_n(\tau x)=\tau\cdot\pi_n(x)$	PR6
$x\|y\|z=\delta$	HA
$x\|y=y\|x$	SC1
$x\|\|y=y\|\|x$	SC2
$x\|(y\|z)=(x\|y)\|z$	SC3
$(x\lfloor y)\lfloor z=x\lfloor(y\|\|z)$	SC4
$(x\|ay)\lfloor z=x\|(ay\lfloor z)$	SC5
$x\|\|(y\|\|z)=(x\|\|y)\|\|z$	SC6
$x_1\|\|\cdots\|\|x_n=\sum_{1\le i\le n} x_i\lfloor(\|\|_{1\le k\le n,\,k\neq i} x_k)+\sum_{1\le i<j\le n}(x_i\|x_j)\lfloor(\|\|_{1\le k\le n,\,k\neq i,k\neq j} x_k)\quad(n\ge 3)$	ET
$\alpha(\delta)=\varnothing$	AB1
$\alpha(\tau)=\varnothing$	AB2
$\alpha(a)=\{a\}$ (if $a\neq\delta$)	AB3
$\alpha(\tau x)=\alpha(x)$	AB4
$\alpha(ax)=\{a\}\cup\alpha(x)$ (if $a\neq\delta$)	AB5
$\alpha(x+y)=\alpha(x)\cup\alpha(y)$	AB6
$\alpha(x)=\cup_{n\ge 1}\alpha(\pi_n(x))$	AB7
$\alpha(\tau_I(x))=\alpha(x)-I$	AB8
$\alpha(x)\|(\alpha(y)\cap H)\subseteq H\Rightarrow\partial_H(x\|\|y)=\partial_H(x\|\|\partial_H(y))$	CA1
$\alpha(x)\|(\alpha(y)\cap I)=\varnothing\Rightarrow\tau_I(x\|\|y)=\tau_I(x\|\|\tau_I(y))$	CA2
$\alpha(x)\cap H=\varnothing\Rightarrow\partial_H(x)=x$	CA3
$\alpha(x)\cap I=\varnothing\Rightarrow\tau_I(x)=x$	CA4
$H=H_1\cup H_2\Rightarrow\partial_H(x)=\partial_{H_1}\circ\partial_{H_2}(x)$	CA5
$I=I_1\cup I_2\Rightarrow\tau_I(x)=\tau_{I_1}\circ\tau_{I_2}(x)$	CA6
$H\cap I=\varnothing\Rightarrow\tau_I\circ\partial_H(x)=\partial_H\circ\tau_I(x)$	CA7

RDP	*Every guarded and abstraction-free specification has a solution*
RSP	*Every guarded and abstraction-free specification has at most one solution*
AIP$^-$	*Every process which has an guarded abstraction-free specification is determined by its finite projections*
CFAR	*If E is a guarded recursive specification, and C a finite conservative cluster of I in E, then for each $X\in C$:* $\tau_I(X)=\tau\sum_{Y\in exits(C)}\tau_I(Y)$.

▷ ook op het formuleren van specificaties en correctheidsbewijzen voor uiteenlopende processen (zie bij voorbeeld [1]).
Door de gevolgde methodologie, het definiëren van axioma's en bewijsregels, is het in principe mogelijk om problemen op te sporen, regels te veranderen en die veranderingen weer te toetsen. Alles is immers expliciet en streng wiskundig geformuleerd. Ook uitbreidingen kunnen zo op een gecontroleerde manier worden ingevoerd.

ACP

In de Algebra van Communicerende Processen zijn alle axioma's en bewijsregels van BPA nog steeds geldig. ACP kan ook de situatie aangeven waarin taken wel allemaal worden uitgevoerd, net als bij sequentiële compositie, maar waarbij de volgorde er niet toe doet. Dit wordt aangeduid met de term *vrije samenvoeging* (*free merge* of *interleaved merge*). Deze operatie, aangegeven met ||, staat dus voor een soort parallellisme.
In het grafenmodel komt de samenvoeging overeen met het cartesisch produkt van de twee grafen links en rechts van de || (zie figuur 7). Om de operatie te kunnen definiëren is ook nog een hulpoperator nodig, de zogenaamde *samenvoeging links* (*left merge*) of ||– maar deze speelt verder geen bijzondere rol.
Voor twee atomaire processen *a* en *b* geldt:

$$a \parallel b = ab + ba$$

Er is echter nog een mogelijkheid namelijk die waarbij twee gescheiden taken zodanig aan elkaar zijn gekoppeld dat zij niet onafhankelijk van elkaar kunnen functioneren.

Er ontstaat zo als het ware een nieuw proces uit de twee andere.
Stel bij voorbeeld dat *a* een zender voorstelt en *b* een ontvanger. In ACP wordt dit uitgedrukt met:

$$c = a \mid b$$

waarin | de *communicatie* tussen *a* en *b* voorstelt.
Communicatie wordt ook toegelaten tot de samenvoeging. Zo levert, op basis van axioma *CM1* uit tabel 2, a || b de graaf uit figuur 8ª op:

$$a \parallel b = ab + ba + a \mid b$$

Met *CM1* tot *CM9* zijn dergelijke grafen ook te construeren voor uitgebreidere formules. Als *a* en *b* uitsluitend met elkaar communiceren en niet afzonderlijk of in een andere combinatie kunnen voorkomen, moet dat in het ACP-formalisme kunnen worden aangegeven.

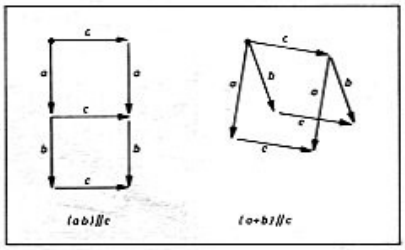

Fig. 7. Samenvoegen zonder communicatie.

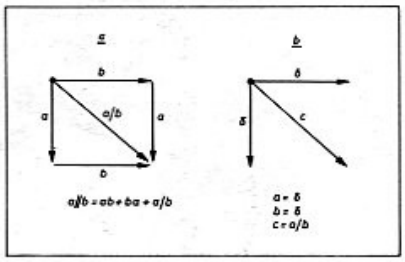

Fig. 8. Communicatie tussen a *en* b *levert een extra proces* c *op (c = a | b). Als* a *en* b *zijn geblokkeerd blijft* c *over.*

VASTLOPEN

De aanduiding δ staat voor een *geblokkeerd proces*. Na δ komt er niets meer en als er gekozen kan worden tussen δ en een andere taak blijft alleen die laatste over. Deze eigenschap wordt uitgedrukt met *A6* en *A7* uit tabel 2.
Zijn nu zowel *a* en *b* uit figuur 8ª geblokkeerd (a = δ, b = δ), terwijl *c* expliciet als communicatie was gepostuleerd (c = a | b), dan gaat de situatie van figuur 8ª over in die van figuur 8ᵇ. Door vervolgens de regel toe te passen die vastgelopen taken uit de graaf haalt (*A6*), blijft alleen de communicatie *c* over. Als de communicatie niet expliciet als proces was gedefinieerd, zou die ook (wegens *C3*) in δ zijn overgegaan.
Voor actief blokkeren is een afzonderlijke bewerking gedefinieerd, *encapsulatie* genaamd en aangeduid met de operator $\delta(H)$, die in de regels *D1* tot *D4* is uitgewerkt. Het komt er op neer dat een verzameling *H* precies die processen als elementen bevat die moeten worden geblokkeerd.
Gelijktijdige communicatie tussen meer dan twee taken is onmogelijk wegens axioma *HA* (*handshaking*) uit tabel 3. Onder die voorwaarde geldt ook de expansiestelling *ET* met behulp waarvan samengestelde samenvoegbewerkingen kunnen worden ontleed. *SC1* tot *SC6* zijn hulpstellingen die onder meer nodig zijn voor het bewijzen van *ET*. De regels *AB* en *CA* uit tabel 3 voor de *alfabetcalculus* dienen om te bepalen welke constanten in variabelen kunnen voorkomen. Zij blijven verder onvermeld.

ABSTRACTIE

Zoals tijdens de cursus bleek, is de algebra gaandeweg en stapsgewijs uitgebreid met nieuwe operaties, constanten, axioma's en bewijsregels. Een van die toevoegingen is de *abstractie*, die als operator τ_I heeft gekregen, waarmee processen kunnen worden 'verborgen'. Geheel analoog aan de encapsulatieoperatie speelt hier een verzameling *I* waarin als elementen de atomaire taken zijn opgenomen die moeten worden geabstraheerd. Deze worden dan hernoemd tot een *stille stap* τ (*silent step*).
Het belangrijkste kenmerk van τ is dat hij in bepaalde gevallen uit formules kan worden weggelaten. De regels hiervoor staan in de rechterhelft van tabel 2 (zie ook tabel 1).
Achterliggende gedachte bij het begrip abstractie is dat in bepaalde gevallen kennis van de precieze aard van sommige deeltaken niet nodig is om het proces te kunnen beschrijven. Het bevordert de overzichtelijkheid wanneer deze details dan ook niet in de specificaties voorkomen.

NEGEREN

Dikwijls is het wel van belang om te weten dat er intern iets gebeurt maar maakt het verder niet uit wat dat is. Een voorbeeld daarvan is de vergelijking X = aX + b, die al eerder aan de orde is geweest (zie figuur 6). Hierin kan *a* zich onbeperkt herhalen maar ligt al vanaf het begin en na elke iteratie een keuze voor *b* vast.
Het voorbeeld van de Russische roulette is ook hier weer bruikbaar. In dit geval wordt de situatie door slechts twee taken gerepresenteerd: de trekker overhalen met een lege kamer *a* of schieten met een volle kamer *b*. Waar komt dit nu op neer als *a* niet interessant is en deze mogelijkheid moet worden geabstraheerd? Wat is het verschil als *a* nu eens niet het schieten zonder kogel voor de loop voor zou stellen maar het drinken van een glas wodka? De intuïtieve conclusie is dat inderdaad, wat *a* ook moge zijn, bij voldoende pogingen *b* toch eens zal optreden.
Dit principe wordt in tabel 3 uitgedrukt in de regel *CFAR* voor *clusterabstractie* (*Cluster Fair Abstraction Rule*) Met *a* in *I* geldt:

$$\tau_I(aX + b) = \tau.b$$

Hier gebeurt dus iets, wat dat ook mag zijn, en daarna volgt *b*. Het is echter ook mogelijk om direct voor *b* te kiezen omdat regel *T2* uit tabel 2 zegt dat:

$$\tau.x = \tau.x + x$$

UITBREIDINGEN

CFAR kan ook worden gebruikt om de *a* weg te werken uit de tweede specificatie van figuur 6. Abstractie van *a* geeft hier:

$$\tau_I(Y) = \tau.\delta + b$$ ▷

▷Wat bovenstaande uitdrukking aangeeft is dat, als voor de stille stap gekozen wordt, er vervolgens niets meer zal gebeuren. Uiteraard draait in werkelijkheid *a* steeds door. Vanuit een extern standpunt echter, waarin alleen het optreden van *b* interessant is, lijkt het net of het totale proces is geblokkeerd.
De hele algebraïsche theorie, zoals geformuleerd in de tabellen 1 tot 3, worden ook wel ACP met τ of *ACP(τ)* genoemd. Deze kan, net zoals met BPA is gebeurd, nog verder worden uitgebreid. Een nieuwe constante ε kan bij voorbeeld worden ingevoerd om het lege proces voor te stellen. Dit heeft zijn nut bij bewerkingen binnen het model van bisimulerende grafen, zoals uitrollen.
Ook is een operatie te definiëren die een taak creëert en kunnen interrupts worden bestudeerd door de taken prioriteiten mee te geven. Verder kan de introductie van een tijdsparameter een real-time algebra doen ontstaan.

PSF-DRAFT

Al deze veranderingen vergen de nodige aanpassingen van de bestaande theorie. Axioma's blijken in de nieuwe situatie soms strijdig te zijn, bestaande modellen hoeven niet meer te voldoen, operaties moeten wellicht anders worden gedefinieerd enzovoort.
Dergelijke problemen hebben zich ook al voorgedaan bij ACP(τ). Het projectievelimietmodel voor BPA, bij voorbeeld, is hierin ongeldig en het grafenmodel moest ook enigszins worden bijgesteld. Eveneens een zorgenkindje is de samenvoegoperatie, die vooral lastig wordt bij real-time algebra en bij interrupts.

Tenslotte moet nog worden opgemerkt dat het bovenstaande alleen een ruwe indruk kan pretenderen te geven. De theorie is niet alleen moeilijk, zowel mathematisch als conceptueel, er zijn ook nog talloze onopgeloste problemen. Een en ander biedt natuurlijk ook veel mogelijkheden en met het *Process Specification Formalism–draft* (*PSF–d*), dat zich op ACP wil baseren, komt bovendien een hulpmiddel of 'tool' in zicht.

Het formalisme voor specificatie dat thans in ontwikkeling is, baseert zich op ACP(τ) en is, zodra ook datastructuren zijn ingevoerd, te vergelijken met een parallelle programmeertaal. De rapporten *A Process Specification Formalism* van Mauw en Veltink [5] en *Real Time Process Algebra* van Baeten en Bergstra [6] geven een uitgebreide beschrijving, een verantwoording op basis van operationele semantiek (zie hiervoor ook [4]) alsmede een vergelijking met de specificatietaal *LOTOS*.

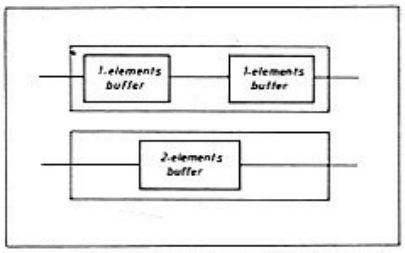

Fig. 9. Het voorbeeld dat ook tijdens het practicum van de cursus aan de orde kwam: de beschrijving van twee een-elements buffers in serie en van een twee-elements buffer.

PRACTICUM

Het volgende probleem kwam op het practicum van de AvI-cursus aan de orde en kan tevens het voorgaande toelichten. Figuur 9 toont boven twee serieel geschakelde databuffers die elk precies één element kunnen bevatten. De linkerbuffer *A* kan een dataeenheid inlezen en er een wegschrijven en hetzelfde geldt voor de rechterbuffer *B*. Listing 1 bevat de specificatie van het geschetste voorbeeld. De buffers *A* en *B* zijn in listing 1 gedeclareerd als *atoms* en hebben de namen *OBB-L* en *OBB-R* gekregen. Het ingelezen element bij *OBB-L* heet *r1*, het weggeschreven *s3* en bij *OBB-R* zijn dat respectievelijk *r3* en *s2*. De specificatie van buffer *A*, de taak *OBB-L*, wordt nu gegeven door de recursieve vergelijking:

OBB-L = r1.s3.OBB-L

De punt voor vermenigvuldiging of sequentiële compositie is hier natuurlijk expliciet nodig.
Dat de vergelijking gedekt is valt eenvoudig te constateren en door substitutie wordt ook duidelijk dat er niets meer gebeurt dan het herhaald inlezen en weer wegschrijven van een dataelement. De rechterbuffer *B* (*OBB-R*), wordt volledig analoog gespecificeerd.
OBB-L en *OBB-R* zijn aan elkaar gekoppeld tot een dubbele één-elements buffer.

Dit gebeurt op een welomschreven manier, namelijk door de communicatie tussen *s3* en *r3*, gerepresenteerd en gedefinieerd als het atomaire proces *c3* (zie listing 1). Koppeling van *A* en *B* is mogelijk door de samenvoegoperator te gebruiken en de onafhankelijke taken *r3* en *s3* te blokkeren. Deze komen dus in de encapsulatieverzameling *H* te staan.
Double-OBB wordt dan:

Double-OBB = encaps(H,OBB-L||OBB-R)

TWEE ELEMENTEN

Wat is nu de specificatie van de twee-elements buffer van figuur 9? Er zijn drie situaties en derhalve ook drie processen. De eerste taak, *TBB*, is een leesoperatie *r1* waarop *TBB'* kan volgen. *TBB'* bestaat uit een leesbewerking *r1* en kan op zijn beurt worden gevolgd door ofwel *TBB''* ofwel ▷

Listing 1. De twee buffers beschreven in de specificatietaal PSF.

```
process module opg3
begin

   atoms
     r1, s2,
     r3, s3, c3

   processes
     Double-OBB, TBB, TBB', TBB'', OBB-L, OBB-R

   sets
        of atoms
             H = { r3, s3 }
             I = { c3 }

   communications
      r3 | s3 = c3

   definitions

     OBB-L = r1 . s3 . OBB-L
     OBB-R = r3 . s2 . OBB-R
     Double-OBB = hide(I,encaps(H, OBB-L || OBB-R ))

     TBB = r1 . TBB'
     TBB' = r1 . TBB'' + s2 . TBB
     TBB'' = s2 . TBB'
```

▷ een schrijfoperatie *s2* waarna *TBB* weer kan beginnen. *TBB''*, tenslotte, omvat louter *s2* en kan alleen worden gevolgd door *TBB'*.
De gehele twee-elements buffer krijgt dus de recursieve specificatie:

TBB
= r1.TBB'.TBB'
= r1.TBB'' + s2.TBB.TBB''
= s2.TBB'

De toestanden van de buffer zijn hierbij hoogstens impliciet gebruikt, de specificatie bevat alleen processen. Het gaat er om dat op *TBB* alleen *TBB'* kan volgen, etcetera.

BEWIJS

Uit figuur 9 valt al op te maken dat de twee-elements buffer niet hetzelfde is als de twee enkelvoudige in serie. Het kan echter zijn dat de specificaties wel overeenkomen als wordt geabstraheerd van de interne communicatie van figuur 8[a]. In dat geval wordt de specificatie, met *c3* in de verzameling *I*:

Double-OBB =
hide(I, encaps(H, OBB-L||OBB-R)

Nu moet, met behulp van de axioma's en bewijsregels uit tabellen 1 en 2 en samen met eventuele daaruit af te leiden hulpstellingen, worden bewezen dat de twee versies inderdaad gelijk zijn. Procesalgebra is eigenlijk ontworpen om zulke bewijzen algebraïsch te leveren, onder meer omdat het graafmodel in wat meer ingewikkelde situaties erg onoverzichtelijk wordt.

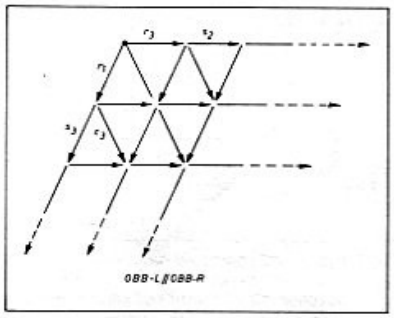

Fig. 10. Resultaat van het samenvoegen van OBB-L *en* OBB-R.

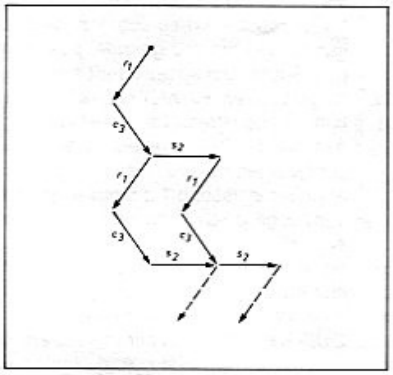

Fig. 11. Encapsulatie levert deze graaf op, waarbij alle geblokkeerde processen zijn weggelaten.

In dit geval is het, informeel, grafisch nog wel aannemelijk te maken dat de specificaties aan elkaar gelijk zijn. Figuur 10 toont de samenvoeging van *OBB-L* en *OBB-R* met communicatie. Het effect van encapsulatie daarop is geschetst in figuur 11, waarbij alle geblokkeerde taken zijn weggelaten. Als nu van *c3* wordt geabstraheerd, ontstaat figuur 12. De stille stap is te elimineren omdat die steeds achter *r1* staat (zie regel *T1* van tabel 2).
De specificatie van de twee-elements buffer komt overeen met de graaf uit figuur 13. Voor elke knoop in de ene graaf is een knoop in de andere graaf te vinden met precies dezelfde processen die daar mogelijk op kunnen volgen. De grafen bisimuleren dus en dat is, in het algebraïsche model, het criterium dat zij gelijke taken representeren.

PROGRAMMATUUR

Het bovenstaande is nog geen mathematische afleiding. In het algemeen zijn echter axiomatische bewijzen erg nuttig. Het probleem is echter dat de zaak, zodra het de analyse van wat ingewikkelder situaties

Academie voor Informatica

Fundamenteel onderzoek kan, haast per definitie, niet meteen in de praktijk worden toegepast. Doorgaans moeten wetenschappelijke ontwikkelingen nog 'rijpen' alvorens ze commercieel bruikbaar zijn. Het risico daarbij is dat te lang wordt gewacht; als echter methoden en technieken onopgemerkt blijven tot de internationale concurrentie er aan begint dan is er duidelijk sprake van gemiste kansen.
De discussie over de rol van fundamenteel onderzoek is, inderdaad, zo oud als de weg naar Rome, maar het is onomstreden dat de afstand met de applicatie zo klein mogelijk moet zijn. De Academie voor Informatica (AvI) stelt zich ten doel om wetenschappelijk onderzoek in de informatica dat aan Nederlandse universiteiten en instituten plaatsvindt onder de aandacht te brengen van het bedrijfsleven. Dit gebeurt door relevante cursussen aan te bieden die vaak worden verzorgd door de betreffende onderzoekers zelf. De filosofie hierbij is dat het kennisaanbod, en niet zozeer de marktvraag, de keuze van onderwerpen bepaalt.

Samenwerking

De AvI is geïnitieerd door prof.dr. J.A. Bergstra (UvA en RUU). Inmiddels bestaat er een gestructureerd samenwerkingsverband van de Universiteit van Amsterdam, de Vrije Universiteit, de Rijksuniversiteit Utrecht, het Centrum voor Wiskunde en Informatica (te Amsterdam) en de Amsterdamse Hogeschool voor de Kunsten. Bovendien is de studierichting 'Automation in Technology' opgezet in samenwerking met de Technische Universiteit Delft. De Academie is ondergebracht in een stichting en het project heeft een startsubsidie ontvangen van het ministerie van Onderwijs en Wetenschappen voor de duur van vijf jaar (tot 1993).
Er zijn zes studierichtingen, waaronder de kerninformatica. Deze bevat weer een afdeling die *Software Technology School* wordt genoemd. Hiertoe behoren op dit moment *software-algebra, programmeeromgevingen/compilerbouw* en *formele ontwerptechnieken*. Deze zijn opgebouwd uit modulen; een- of tweedaagse cursussen waarin dan een bepaald gebied de revue passeert.

Cursus

Onder software-algebra vallen bij voorbeeld de onderwerpen *term-herschrijftechnieken, algebraïsche specificaties, module-algebra* en *procesalgebra*. Laatstgenoemde cursus, die in dit artikel wordt beschreven, omvat het uitgebreide gelijknamige college voor derdejaars studenten aan de UvA, met een practicuminleiding, plus een introductie in real-time procesalgebra en nog een bespreking van een bepaald artikel over een applicatie.
Als voorkennis is volgens de folder 'elementary discrete mathematics' vereist, maar daar valt over te twisten. De benodigde achtergrondtheorie werd weliswaar ter plekke behandeld maar een zekere vertrouwdheid daarmee was echt geen luxe voor deze cursus. Het onderwerp procesalgebra werd belicht als een wiskundige theorie op zich en, behalve in het practicum, nauwelijks als een mogelijke techniek of hulpmiddel. Het was waarschijnlijk interessant geweest om ook verbanden met de 'concurrerende' theorieën CCP en CSP te bespreken.
De essentie is echter dat procesalgebra, gezien als geavanceerde wetenschappelijke theorie, met degelijkheid als zodanig werd behandeld. Of het voor 'de praktijk' gewenst is om dergelijke kennis op te pakken of om nog af te wachten is uiteindelijk een strategische beslissing. Dat de beschikbaarheid van zulke kennis wordt vergroot door een instelling als de Academie voor Informatica lijkt, hoe dan ook, alleen maar gunstig.

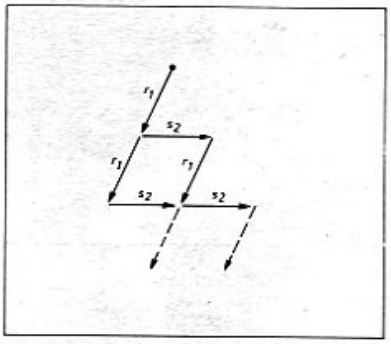

Fig. 12. Abstractie van c, gevolgd door eliminatie van τ.

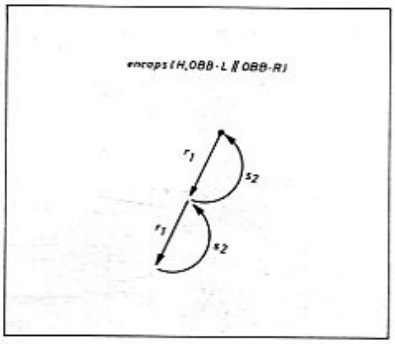

Fig. 13. Deze graaf voor de twee-elements buffer biedt voor elke knoop precies dezelfde keuzeprocessen als de omgewerkte graaf voor de twee een-elements buffers. De twee bisimuleren derhalve.

betreft, snel groot en onoverzichtelijk wordt. Met de momenteel in ontwikkeling zijnde ondersteunende programmatuur moet ook daar verbetering in komen. Met PSF-implementaties, zoals die in listing 1, kunnen de daarin gedefinieerde taken in principe ook worden uitgevoerd (gesimuleerd in de gebruikelijke betekenis). Tevens wordt gewerkt aan de mogelijkheid om door de gebruiker aan te geven termen automatisch te herschrijven. ACP kan namelijk ook worden geformuleerd als een *termreductiesysteem* (*TRS*) en er bestaat een algemene theorie over dergelijke systemen. □

Inl.: Academy for Computer Science, Software Technology School, Van Diemenstraat 86, 1013 CN Amsterdam (020) 205828.

Referenties:

[1] J.C.M. Baeten (red.): *Applications of Process Algebra*, Cambridge University Press, in druk:
[1*] S. Mauw: *Process Algebra as a Tool for the Specification and Verification of CIM-architectures*, pag. 53-80.
[1²] J.A. Bergstra, J.W. Klop: *An Introduction to Process Algebra*, pag. 1-22.
[1³] L. Kossen, W.P. Weijland: *Correctness Proofs for Systolic Algorithms: Palindromes and Sorting*, pag. 89-125.
[2] J.C.M. Baeten: *Procesalgebra*, Kluwer, 1986.
[3] J.C.M. Baeten , W.P. Weijland: *Process Algebra*, Cambridge University Press, in druk.
[4] S. Mauw, J.G. Veltink: *An Introduction to PSF*, rapport P8901 van de vakgroep Programmatuur, Universiteit van Amsterdam, januari 1989.
[5] S. Mauw, J.G. Veltink: *A Process Specification Formalism*, rapport P8814 van de vakgroep Programmatuur, Universiteit van Amsterdam, juli 1988.
[6] J.C.M. Baeten, J.A. Bergstra: *Real Time Process Algebra*, rapport P8916b van de vakgroep Programmatuur, Universiteit van Amsterdam, maart 1990.