Proces algebra

PROCES ALGEBRA

door dr. J. C. M. Baeten

Een inleiding wordt gegeven in de proces algebra, een zich ontwikkelend onderzoeksgebied in de concurrency. Dat wil zeggen dat proces algebra een theorie is over concurrente, communicerende processen. De gebruikte methoden zijn algebraïsch en axiomatisch van aard.

1 PROCESSEN

Proces algebra is een theorie over concurrente, communicerende processen. We willen het begrip proces ruim uitleggen: men kan denken aan het uitvoeren van een programma door een computer of het uitvoeren van een algoritme, maar ook aan het gedrag van een telefooncentrale of koffie-automaat, of aan bepaalde handelingen van een persoon.
We willen processen beschouwen die concurrent of parallel plaatsvinden, en die met elkaar kunnen communiceren, denk bijvoorbeeld aan gedistribueerde systemen of communicatie-protocollen. Een theorie over dergelijke processen zal zich concentreren op bepaalde aspecten en zal abstraheren van andere, zal bijvoorbeeld alleen de handelingen van een persoon beschouwen en niet zijn emoties en gedachten. Een goede theorie moet bruikbaar zijn voor 3 doeleinden, namelijk specificatie, verificatie en ontwerp.
Ten eerste moeten we processen kunnen specificeren of beschrijven in onze theorie, en moeten we ingewikkelde systemen kunnen opbouwen uit eenvoudigere. Daarvoor hebben we een abstracte taal nodig, waarin we kunnen praten over processen, en waarin we processen op diverse manieren kunnen samenstellen.
Ten tweede moeten we kunnen verifiëren of bewijzen dat een beschreven proces het juiste gedrag vertoont, of dat twee specificaties hetzelfde proces beschrijven. Daarvoor hebben we een theorie nodig die zegt wanneer twee processen gelijk zijn, een theorie die zal bestaan uit vergelijkingen, identiteiten tussen processen.
Tenslotte zal een goede theorie een stuk gereedschap zijn bij het ontwerpen van systemen, protocollen of computertalen.
Dit artikel wil duidelijk maken dat proces algebra een goede theorie is over concurrente, communicerende processen. Aan het eind komen wij terug op de zojuist genoemde criteria.

2 MODELLEN

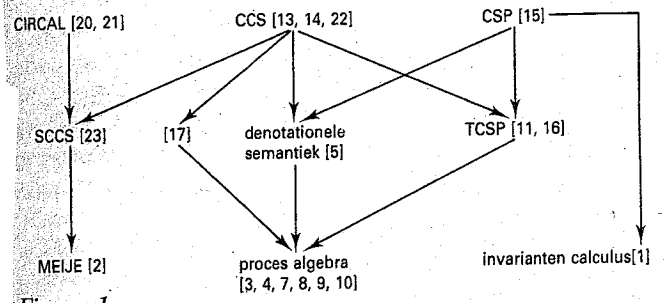
Theorieën over concurrente processen heten in de vaklitteratuur *concurrency* theorieën. Concurrency staat tegenwoordig zeer in de belangstelling; het gebruik van gedistribueerde systemen neemt steeds toe, en er zijn verschillende projecten om machines te bouwen met meerdere processoren, parallelle machines genaamd. Een algemeen geaccepteerde theoretische ondergrond voor dergelijke systemen is nog niet voorhanden, en is het onderwerp van veel discussie. Dit is in tegenstelling tot de niet controversiële en veel eenvoudigere theorie over sequentiële, deterministische machines.

Heden ten dage zijn de belangrijkste theorieën van concurrency CCS (Calculus of Communicating Systems), zie Milner [22], CSP (Communicating Sequential Processes), zie Hoare [15], en Petri net theorie, zie Reisig [24]. Al deze theorieën gaan uit van een *model* van concurrency. Dit wil zeggen dat men door middel van abstractie van bepaalde aspecten van processen komt tot een algebraïsche structuur, een wiskundig model. Zo'n model gaat uit van niet nader gedefinieerde basisobjecten of atomen, hier meestal acties, gebeurtenissen of stappen genaamd, en bevat verder manieren om deze acties samen te stellen tot ingewikkelde processen. Vervolgens gaat men onderzoeken welke gelijkheden gelden in het model, en probeert men algemene wetten te formuleren. Zo is bijvoorbeeld de verzameling natuurlijke getallen (0, 1, 2, 3,...) een algebraïsche structuur om rekenen te beschrijven, met de getallen als atomen, en samenstellende functies (operatoren genaamd) plus en maal. In dit model kan men wetten ontdekken als $x + y = y + x$.
In de concurrency bestaan dus verscheidene modellen, waarvan we er enkele genoemd hebben.
Proces algebra nu gaat niet uit van een bepaald model, maar volgt de axiomatische methode. Dit houdt in dat we beginnen met een verzameling wetten, die in de meeste tot nog toe voorgestelde modellen gelden. Aldus krijgt men een algemene algebraïsche theorie, bij een nieuwe voorgestelde wet kan de klasse modellen bekeken worden waar hij geldt, en de klasse waar hij niet geldt, kan men proberen een bepaald model volledig met wetten te beschrijven, en kan men zien welke operatoren in welke modellen gedefinieerd kunnen worden. Door deze benadering kunnen ook de verschillende theorieën van concurrency met elkaar vergeleken worden, en kunnen bijvoorbeeld de voor- en nadelen van CCS en CSP besproken worden. Zo beschrijft (dezelfde analogie gebruikend als eerder) de algebraïsche groepentheorie alle structuren met een operator die aan bepaalde wetten voldoet, en is de verzameling gehele getallen een bepaalde groep, een model van deze theorie. De wet $x + y = y + x$ geldt alleen in de klasse abelse groepen, niet in alle groepen.

3 HISTORISCHE OPMERKINGEN

Proces algebra is grotendeels het werk van personen verbonden aan het Centrum voor Wiskunde en Informatica te Amsterdam. Daar begonnen in 1982 J. A. Bergstra en J. W. Klop met proces algebra, naar aanleiding van een vraag van De Bakker en Zucker [5]. Vergeleken met andere concurrency theorieën is proces algebra het nauwst verwant aan CCS. We kunnen de volgende stamboom tekenen van CCS, CSP en verwante systemen (zie figuur

1). We hebben ons niet beziggehouden met theorieën die voorafgingen aan CCS en CSP. Bij elke theorie geven we enige verwijzingen naar de litteratuur.



Figuur 1

Als hiermee niet verwante theorieën van concurrency noemen we nog de theorie van Petri netten [24] (Petri netten geven een grafische voorstelling van parallelle processen), temporele logica (predicaten logica verrijkt met operatoren om over tijdstippen en tijdsduren te spreken; hierover is veel litteratuur, in [12] vinden we een relatie met CCS) en trace theorie [18, 19, 25] (de theorie die kijkt naar de traces, de mogelijke volgorden acties van een proces).
Ten aanzien van de presentatie van dit artikel willen we nog het volgende opmerken: het zal de lezer opvallen dat er erg veel Engelse terminologie gehanteerd wordt. In principe verdient het natuurlijk de voorkeur hier Nederlandse equivalenten voor te vinden. Bij een zo recent onderzoeksgebied als dit zijn zulke equivalenten nog niet voorhanden, en dan gebruiken we de oorspronkelijke Engelse frase, liever dan dat we een ad hoc vertaling introduceren.

4 BASIS VOOR PROCES ALGEBRA

In deze sectie presenteren wij een kernsysteem voor proces algebra, dat nog niet parallellisme of communicatie kan beschrijven, maar waarin toch al verscheidene zaken duidelijk gemaakt kunnen worden en vele processen gespecificeerd kunnen worden. We gaan uit van een verzameling van niet nader gedefinieerde objecten, *atomaire acties* of *stappen* genaamd. We beschouwen deze ondeelbare acties meestal als puntvormig in de tijd. Dit is geen wezenlijke beperking: als een gebeurtenis een bepaald tijdsinterval duurt, nemen we als atomaire actie het beginpunt (en soms ook het eindpunt) van die actie. Letters a, b, c zullen steeds atomaire acties voorstellen.
Verder beschouwen we twee binaire operatoren, te weten *sequentiële compositie* (product of ·) en *alternatieve compositie* (som of +) met de volgende betekenis:
a·b is het proces dat eerst a doet, en vervolgens b;
a+b is het proces dat ofwel a doet, ofwel b (maar niet beide).

Door herhaald toepassen van deze operatoren kunnen we processen maken als ab + c, (a + b)c, ac + bc, a(b + c), ab + ac en vele andere. Merk op dat we dezelfde notationele conventies gebruiken als in de middelbareschool-algebra: ab betekent a·b, en ab + c betekent (ab) + c. In het algemeen zullen we hebben dat · het sterkst bindt van alle operatoren, en + het zwakst.
We gebruiken de letters x, y, z voor willekeurige processen (dus atomaire acties of samenstellingen daarvan). Voor de operatoren + en · formuleren we de volgende vijf wetten of axioma's, die te zamen het axiomasysteem BPA (Basis Proces Algebra) vormen (zie tabel 1).

1. $x + y = y + x$
2. $(x + y) + z = x + (y + z)$
3. $x + x = x$
4. $(x + y)z = xz + yz$
5. $(xy)z = x(yz)$

Tabel 1: BPA

De eerste drie axioma's geven alle een eigenschap van +, en zijn verklaarbaar uit het feit dat x + y een *keus* betekent tussen x en y; axioma 1 is de *commutativiteit* van +, 2 is de *associativiteit* en 3 de *idempotentie*. Te zamen zeggen ze dat we op elk moment een *verzameling* keuzen hebben. Axioma 4, de *rechts-distributiviteit* van · over +, zegt dat een keus tussen x en y, gevolgd door z, hetzelfde is als een keus tussen x gevolgd door z en y gevolgd door z. Axioma 5 tenslotte is de associativiteit van ·, en is vanzelfsprekend.

5 NON-DETERMINISME

Sommige lezers denken misschien dat in bovenstaand lijstje nog een axioma ontbreekt, namelijk
$x(y + z) = xy + xz$,

dat volledige distributiviteit zou geven. We willen dit axioma echter niet opnemen in ons kernsysteem BPA, omdat het in de meeste modellen van concurrency niet geldt (het geldt eigenlijk slechts in een model, de zogenaamde *trace* theorie, zie [25]).
In term x(y + z) moet *eerst* x gedaan worden, en *vervolgens* gekozen worden tussen y en z, terwijl in xy + xz *eerst* gekozen moet worden, en *vervolgens* de gekozen term uitgevoerd moet worden. Dus is het keuzemoment in beide termen verschillend.
Een keuze als in xy + xz tussen aanvankelijk gelijke alternatieven heet een *non-deterministische* keuze, en vormt een bron van controversen in de concurrency.
We willen dit punt illustreren met het volgende voorbeeld. Stel dat u in een kamer zit met twee deuren, en u weet dat achter een deur een mensetende tijger zit, en achter de andere een beeldschone prins(es), maar u weet niet, wat er achter welke deur zit. Nu moet u een deur openen. We hebben de volgende atomaire acties:
deur = u doet een deur open;
tijger = u wordt opgegeten door de tijger;
prins(es) = u wordt gekust door de prins(es).
Dan voert u het volgende proces uit:
deur·prins(es) + deur·tijger,
hetgeen toch wel heel iets anders is als
deur(prins(es) + tijger)
waar *na* het openen van de deur nog gekozen kan worden tussen prins(es) en tijger!

6 RECURSIE

Uit de middelbare-schoolalgebra zijn we allemaal vertrouwd met het oplossen van een onbekende uit een vergelijking; zo heeft bijvoorbeeld de vergelijking $x = x^2 - x + 1$ de oplossing $x = 1$.
Iets dergelijks doen wij ook, als we processen definiëren met behulp van recursie. Laten we als voorbeeld de vergelijking

$x = a \cdot x$

nemen (a een bepaalde atomaire actie).
Door ax steeds 'in te vullen' voor x vinden we

$x = ax = aax = aaax = \ldots = aaaaa\ldots,$

zodat de 'oplossing' voor x het oneindige proces is, dat steeds actie a uitvoert. We zijn geïnteresseerd in vergelijkingen die een unieke oplossing hebben, en zeggen dat zo'n vergelijking een proces *specificeert.* Zo specificeert de vergelijking $x = ax$ het proces aaaaaa...
Niet alle vergelijkingen hebben een unieke oplossing (*alle* processen voldoen aan de vergelijking $x = x$), maar we kunnen een ruime klasse vergelijkingen definiëren (de zogenaamde *guarded* vergelijkingen) die unieke oplossingen hebben. Als voorbeeld kunnen we een proces specificeren dat zich gedraagt als een *stack* van nullen en enen, met een stelsel van 4 vergelijkingen met 4 onbekenden (S, T, T_0, T_1):

$S = T \cdot S$
$T = push(0) \cdot T_0 + push(1) \cdot T_1$
$T_0 = pop(0) + T \cdot T_0$
$T_1 = pop(1) + T \cdot T_1$

Hier hebben we de volgende atomaire acties:
push(0) (resp. push(1)): push een 0 (resp. een 1) op de stack;
pop(0) (resp. pop(1)): pop een 0 (resp. een 1) van de stack.
(De oplossing voor) S is de stack, T is de terminerende stack (die ophoudt zodra hij leeg is), en T_0 (resp. T_1) stelt een element 0 (resp. 1) voor, dat zich op een stack bevindt. Zo zegt de eerste vergelijking dat $S = T \cdot T \cdot T \cdot \ldots$ (vgl. $x = ax$). De begintoestand is S, de lege stack. Dan $S = T \cdot S = (push(0) \cdot T_0 + push(1) \cdot T_1) \cdot S = push(0) \cdot T_0 \cdot S + push(1) \cdot T_1 \cdot S$, dus na het uitvoeren van push(0) hebben we toestand $T_0 \cdot S$, die dus de stack met inhoud 0 voorstelt. Na verschillende malen 'invullen' en bepaalde volgorden van acties na te gaan, zien we dat bijvoorbeeld de stack met een 0 op top, met 1 en nog een 1 eronder, voorgesteld wordt door proces $T_0 \cdot T_1 \cdot T_1 \cdot S$, en optreedt na het uitvoeren van acties push(1), push(1), push(0) (in die volgorde).
Men kan zo inzien, dat een stack kan worden gespecificeerd met de operatoren + en ·, maar een *queue* kan met de in dit artikel besproken operatoren niet in eindig veel vergelijkingen gespecificeerd worden (een bewijs hiervoor valt buiten het bestek van dit artikel).

7 MERGE

Om processen te kunnen beschrijven die concurrent of parallel plaatsvinden, voeren we de *merge*-operator $\|$ in. (Het Engelse woord merge betekent verweving of fusie.) Het proces $x\|y$ is het proces dat parallel of concurrent de processen x en y uitvoert, en als zowel x als y bestaan uit een aantal momentane atomaire acties, dan zien we dat deze acties verweven worden in de tijd: op elk moment zien we ofwel de volgende actie van x, ofwel de volgende actie van y gebeuren. Zo zien we in $a\|b$ (a en b twee atomaire acties) ofwel eerst a gebeuren en dan b, ofwel eerst b en dan a. Aldus krijgen we de volgende identiteit:

$a\|b = ab + ba.$

Merk op dat we er ons hier niet over uitlaten, of de ene actie is afgelopen als de andere begint. Dat kan bijvoorbeeld afhangen van de implementatie van een proces (op een sequentiële of een parallelle machine).

Evenzo krijgen we bijvoorbeeld

$(ab)\|c = a(bc + cb) + cab$

(let op het keuze-moment!), en

$(a + b)\|c = ac + bc + c(a + b).$

Merk op dat dus $(a + b)\|c$ en $(a\|c) + (b\|c)$ *niet* hetzelfde zijn. We sluiten hier ook uit dat twee acties gelijktijdig plaatsvinden (gelijktijdig voorkomende acties zien we wel in de sectie over communicatie). Wel kunnen we expliciet maken dat twee acties elkaar in tijd overlappen: als g en h twee gebeurtenissen zijn, die enige tijd duren, kunnen we als atomaire acties nemen *begin(g), eind(g), begin(h)* en *eind(h),* en is een van de mogelijkheden van

$(begin(g) \cdot eind(g)) \| (begin(h) \cdot eind(h))$

gegeven door $begin(g) \cdot begin(h) \cdot eind(g) \cdot eind(h)$.
We merken nog op dat in deze theorie geen expliciete vermelding van tijd gemaakt kan worden, wat bijvoorbeeld wel kan in temporele logica.
Nu moeten we nog een aantal identiteiten geven, die het gedrag van de operator $\|$ volledig beschrijven, net zoals we in het systeem BPA de operatoren + en · beschreven. Het is gebleken dat we, om $\|$ te kunnen beschrijven in eindig veel vergelijkingen, een hulpoperator nodig hebben. Deze hulpoperator is $\lfloor\!\lfloor$, de left-merge of linksmerge. Hierbij betekent $x\lfloor\!\lfloor y$ hetzelfde als $x\|y$, maar met de restrictie dat de eerste stap van x moet komen. In tabel 2 presenteren we het axiomasysteem PA (PA is een afkorting van Proces Algebra, een naam die wij nu reserveren voor de hele algebraïsche theorie van concurrency, waarvan dit axiomasysteem slechts een onderdeel uitmaakt).

1. $x + y = y + x$
2. $(x + y) + z = x + (y + z)$
3. $x + x = x$
4. $(x + y)z = xz + yz$
5. $(xy)z = x(yz)$
6. $x\|y = x\lfloor\!\lfloor y + y\lfloor\!\lfloor x$
7. $a\lfloor\!\lfloor x = ax$
8. $ax\lfloor\!\lfloor y = a(x\|y)$
9. $(x + y)\lfloor\!\lfloor z = x\lfloor\!\lfloor z + y\lfloor\!\lfloor z$

Tabel 2: PA

In deze tabel stellen x, y, z willekeurige processen voor, en stelt de letter a een willekeurige atomaire actie voor. De axioma's 1 - 5 zijn weer de axioma's van BPA (zie sectie 3). Axioma 6 definieert de $\|$ in termen van de $\lfloor\!\lfloor$ (als we x en y mergen, hebben we een merge met ofwel de eerste stap van x, ofwel de eerste stap van y). Axioma's 7 - 9 definiëren tenslotte de $\lfloor\!\lfloor$, al naar gelang de vorm van het linkerproces.
Laten we, om het gebruik van deze axioma's te illustreren, twee eerder genoemde identiteiten eens formeel afleiden:

$ab\|c = ab\lfloor\!\lfloor c + c\lfloor\!\lfloor ab$ (axioma 6)
$= a(b\|c) + c(ab)$ (axioma 8 en 7)
$= a(b\lfloor\!\lfloor c + c\lfloor\!\lfloor b) + cab$ (axioma 6, en eigenlijk ook 5)
$= a(bc + cb) + cab$ (axioma 7); en

$(a + b)\|c = (a + b)\lfloor\!\lfloor c + c\lfloor\!\lfloor (a + b)$ (axioma 6)
$= a\lfloor\!\lfloor c + b\lfloor\!\lfloor c + c\lfloor\!\lfloor (a + b)$ (axioma 9)
$= ac + bc + c(a + b)$ (axioma 7).

Als voorbeeld van het gebruik van de merge-operator in

specificaties specificeren we het gedrag van een koektrommel met twee soorten koekjes, spritsen en kano's, door middel van de volgende vergelijking:

$K = in(sprits)\cdot(K \| uit(sprits)) + in(kano)\cdot(K \| uit(kano))$.

Hier is K de koektrommel (leeg in het begin), *in(sprits)* (resp. *in(kano)*) het erin stoppen van een sprits (resp. kano), en *uit(sprits)* (resp. *uit(kano)*) het eruit halen van een sprits (resp. kano).
Weer vergt het enig werk om zich dit te realiseren; de koektrommel met als inhoud 1 sprits en 2 kano's wordt gegeven door proces

$K \| uit(sprits) \| uit(kano) \| uit(kano)$.

Hier treedt de merge-operator niet op om parallellisme aan te geven, maar om elke volgorde van acties mogelijk te maken. We merken op dat K een voorbeeld is van een queue, waarbij de volgorde van de ingekomen data niet gehandhaafd wordt, maar willekeurig veranderd kan worden. Voorts zien we, dat we zo eenvoudig een teller kunnen specificeren (neem een koektrommel met een soort koekje).

8 DEADLOCK

Stel dat we een machine hebben die het proces $a(b + c)$ (a, b, c atomaire acties) aan het uitvoeren is, maar dat de machine actie c niet kan uitvoeren, deze actie is geblokkeerd in de machine. De machine zal dan a uitvoeren en vervolgens, als de keus komt tussen b en c, c niet kunnen kiezen, dus gedwongen worden te kiezen voor b. Voor deze machine is dus $a(b + c)$ hetzelfde als ab.
Nu vragen we de machine het proces $ab + ac$ uit te voeren. Wordt gekozen voor ab, dan is er niets aan de hand: a gevolgd door b wordt uitgevoerd. Als er echter gekozen wordt voor ac, kan de machine na het uitvoeren van a niets meer doen, dan treedt stagnatie of *deadlock* op. Als we de constante δ invoeren voor deadlock, voor het ontbreken van een alternatief, dan is het gedrag van de machine in dit geval $ab + a\delta$.
Het speciale atoom δ kan nu beschreven worden met de twee wetten in tabel 3.

1. $x + \delta = x$
2. $\delta x = \delta$

Tabel 3: Deadlock

De eerste wet zegt, dat er geen deadlock kan optreden als er nog een alternatief is (de machine vertoonde geen deadlock bij $a(b + c)$), en de tweede, dat als er deadlock is opgetreden, geen enkele actie meer kan volgen.
Verder beschouwen we δ als een atomaire actie, en dus gelden bijvoorbeeld wetten 7 en 8 uit tabel 2 ook voor δ in plaats van a.
Zo krijgen we bijvoorbeeld $a\|\delta = a\lfloor\!\lfloor \delta + \delta\lfloor\!\lfloor a = a\delta + \delta a = a\delta + \delta = a\delta$, en kunnen we in het algemeen bewijzen dat $x\|\delta = x\delta$, met behulp van een techniek die inductie naar de structuur van een term heet.
Verder zal het in de volgende sectie nuttig zijn om een operator te hebben, die kan aangeven dat bepaalde atomaire acties geblokkeerd zijn, niet uitgevoerd kunnen worden. Deze operator zal deze atomaire acties hernoemen in δ, en verder niets doen. We kunnen dit als volgt met wetten formuleren:

zij H een verzameling atomaire acties, dan definiëren we de operator ∂_H met de wetten in tabel 4.
∂_H wordt wel de *encapsulatie*-operator genoemd.

1. $\partial_H(a) = a$ als $a \notin H$
2. $\partial_H(a) = \delta$ als $a \in H$
3. $\partial_H(x + y) = \partial_H(x) + \partial_H(y)$
4. $\partial_H(xy) = \partial_H(x)\cdot\partial_H(y)$

Tabel 4: Encapsulatie

In deze tabel stellen x, y willekeurige processen voor, en a een willekeurige atomaire actie. Als voorbeeld verifiëren we het gedrag van de machine die c niet kan doen, besproken aan het begin van deze sectie, op processen $a(b + c)$ en $ab + ac$.

We nemen dus $H = \{c\}$ en krijgen:
$\partial_H(a(b + c)) = \partial_H(a)\cdot\partial_H(b + c) = a(\partial_H(b) + \partial_H(c)) = a(b + \delta) = ab$, en
$\partial_H(ab + ac) = \partial_H(ab) + \partial_H(ac) = \partial_H(a)\partial_H(b) + \partial_H(a)\partial_H(c) = ab + a\delta$.

9 COMMUNICATIE

Een centraal onderwerp in de concurrency is het modelleren van communicatie. De methode die we hier beschrijven geeft *synchrone* communicatie. Dat wil zeggen dat het optreden van communicatie tussen twee processen het gevolg is van het gelijktijdig uitvoeren van een actie door de twee processen. Zo kan het ene proces bijvoorbeeld een *zend*-actie uitvoeren, en het andere een *ontvang*-actie, en als deze acties gelijktijdig plaatsvinden, is het resultaat een communicatie-actie.
We vermelden nog, dat er in de procesalgebra ook systemen bestaan voor het modelleren van asynchrone communicatie, die bijvoorbeeld het versturen van post kunnen beschrijven. Bij synchrone communicatie heeft elke communicatie dus twee helften. We noteren dit als volgt: als atomaire actie *comm(5)* de communicatie is van atomaire acties *zend(5)* en *ontvang(5)*, schrijven we

$zend(5) | ontvang(5) = comm(5)$.

Formeel zeggen we, dat we een communicatiefunctie | gegeven veronderstellen op de verzameling atomaire acties. Als twee acties niet behoren te communiceren, sluiten we die mogelijkheid uit door hun communicatie gelijk te stellen aan δ, dus bijvoorbeeld

$zend(5) | ontvang(4) = \delta$.

Als we nu de merge van twee processen beschouwen met communicatie, $x\|y$, zijn er niet twee mogelijkheden (zoals in PA, sectie 7), maar drie: ofwel we voeren de eerste actie uit van x ($x\lfloor\!\lfloor y$), ofwel we voeren de eerste actie uit van y ($y\lfloor\!\lfloor x$), ofwel we voeren een communicatie-actie uit, met 'helften' van x en y (deze mogelijkheid noteren we met $x|y$).
We geven $x|y$ een betekenis voor willekeurige processen x en y, al naargelang de vorm van processen x en y. De precieze wetten zijn te vinden in de volgende tabel 5. Merk op dat in dit systeem, ACP, de Algebra van Communicerende Processen, alle tot nog toe besproken wetten terugkeren (met uitzondering van de wet voor merge, die van een nieuwe summand voorzien wordt). In deze tabel stellen x, y, z willekeurige processen voor, a, b, c willekeurige atomaire acties, en is H een verzameling

atomaire acties. Zie voor meer informatie over ACP [6, 9].
In de volgende sectie geven we een uitgewerkt voorbeeld.

1. $x + y = y + x$
2. $(x + y) + z = x + (y + z)$
3. $x + x = x$
4. $(x + y)z = xz + yz$
5. $(xy)z = x(yz)$
6. $x + \delta = x$
7. $\delta x = \delta$

8. $a|b = b|a$
9. $(a|b)|c = a|(b|c)$
10. $\delta|a = \delta$

11. $x\|y = x\llcorner y + y\llcorner x + x|y$
12. $a\|y = x\llcorner x = ax$
13. $ax\|y = x\llcorner y = a(x\|y)$
14. $(x + y)\|y = x\llcorner z = x\|y = x\llcorner z + y\|y = x\llcorner z$
15. $ax|b = (a|b)x$
16. $a|bx = (a|b)x$
17. $ax|by = (a|b)(x\|y)$
18. $(x + y)|z = x|z + y|z$
19. $x|(y + z) = x|y + x|z$

20. $\partial_H(a) = a$ als $a \notin H$
21. $\partial_H(a) = \delta$ als $a \in H$
22. $\partial_H(x + y) = \partial_H(x) + \partial_H(y)$
23. $\partial_H(xy) = \partial_H(x)\partial_H(y)$

Tabel 5: ACP

10 VOORBEELD

Als voorbeeld van het gebruik van het systeem ACP bekijken we een estafetteloop met twee lopers, Ad en Ben. We hebben de volgende atomaire acties en betekenissen:
renA = Ad loopt zijn 100 meter;
renB = Ben loopt zijn 100 meter;
geef = Ad geeft het stokje aan Ben;
pak = Ben pakt het stokje aan van Ad;
stok = het stokje wordt overgedragen van Ad naar Ben.

We hebben de communicatie $geef \mid pak = stok$, en alle andere communicaties resulteren in δ.
Nu voert Ad het proces $renA \cdot geef$ uit, en Ben het proces $pak \cdot renB$. Als we deze processen samenstellen, moeten we ervoor zorgen dat acties *geef* en *pak* niet op zich voorkomen, maar alleen hun communicatie. Dit doen we door middel van encapsulatie: we definiëren $H = \{geef, pak\}$ en beschouwen het proces

$$\partial_H(renA \cdot geef \,\|\, pak \cdot renB).$$

We stellen nu dat dit proces het juiste bedrag vertoont, dat wil zeggen

$$\partial_H(renA \cdot geef \,\|\, pak \cdot renB) = renA \cdot stok \cdot renB.$$

Het bewijs dat dit inderdaad zo is, maakt gebruik van de wetten van ACP. We geven dit bewijs nu in enig detail. Ter geruststelling: het kan zonder bezwaar overgeslagen worden.

$$\begin{aligned}
&\partial_H(renA \cdot geef \,\|\, pak \cdot renB) = \\
&\partial_H(renA \cdot geef \,\llcorner\, pak \cdot renB + pak \cdot renB \,\llcorner\, renA \cdot geef + \\
&\qquad + renA \cdot geef \mid pak \cdot renB) = \\
&\partial_H(renA(geef \,\|\, pak \cdot renB) + pak(renB \,\|\, renA \cdot geef) + \\
&\qquad + (renA|pak)(geef \,\|\, renB) = \\
&\partial_H(renA(geef \,\|\, pak \cdot renB)) + \delta + \delta = \\
&renA \cdot \partial_H(geef \,\llcorner\, pak \cdot renB + pak \cdot renB \,\llcorner\, geef + \\
&\qquad + geef|pak \cdot renB) = \\
&renA \cdot \partial_H(geef \cdot pak \cdot renB + pak(renB \,\|\, geef) + \\
&\qquad + (geef \mid pak)renB) = \\
&renA(\delta + \delta + \partial_H(stok \cdot renB)) = \\
&renA \cdot stok \cdot renB.
\end{aligned}$$

11 ALTERNATING BIT PROTOCOL

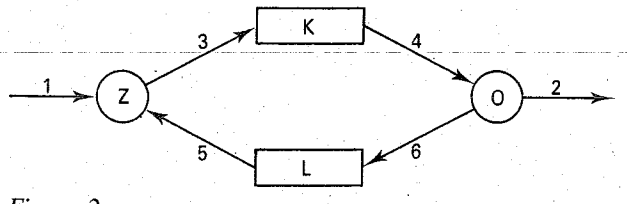
Als een meer praktijkgericht voorbeeld behandelen we een simpele versie van een communicatieprotocol, nl. het zogenaamde alternating bit protocol (zie Bartlett, Scantlebury en Wilkinson [6]). Dit protocol zal toch al behoorlijk ingewikkeld worden, en aan de hand ervan kunnen we enkele geavanceerde onderwerpen in de proces algebra bespreken.
Een communicatieprotocol ziet er meestal als volgt uit: we hebben een eindige verzameling data D, en elementen van D moeten verzonden worden van locatie 1 naar locatie 2 via een *onbetrouwbaar* medium.
We gebruiken de atomaire acties
$l1(d)$: lees bij locatie 1 datum d;
s2(d): schrijf bij locatie 2 datum d.
De specificatie van het gewenste *externe* gedrag van een protocol P is nu

$$P = \sum_{d \in D} l1(d) \cdot s2(d) \cdot P,$$

dat wil zeggen P leest een datum bij 1, voert dan een aantal onzichtbare stappen uit (het verzenden), schrijft dan hetzelfde datum bij 2 en keert terug in zijn begintoestand. Bij de specificatie van P gebruikten we de volgende afkorting: als $X = \{x_1, \ldots, x_n\}$ een verzameling processen is, schrijven we in plaats van $x_1 + x_2 + \ldots + x_n$ ook

$$\sum_{i=1,\ldots,n} x_i \quad \text{of} \quad \sum_{x \in X} x.$$

Het Alternating Bit Protocol implementeert dit als volgt (zie figuur 2).



Figuur 2

De zender Z leest een datum bij locatie 1 in en stuurt een rij d0, d0, d0, ... (kopieën van d met een 0 erachter) naar kanaal K net zo lang tot een ontvangstbevestiging 0 via kanaal L ontvangen wordt. Dan wordt het volgende datum gelezen en met een 1 doorgestuurd; de ontvangstbevestiging is dan 1. Daarna is weer 0 aan de beurt, en zo is de 0, 1 dus het alternerende bit.
K is het datatransmissiekanaal, en stuurt data van de vorm d0, d1 door. Echter, K is onbetrouwbaar in die zin dat hij data onherkenbaar kan maken, in welk geval er e (error) doorgegeven wordt.
O is de ontvanger die data d0, d1 van K ontvangt, de d bij

2 schrijft als deze al niet eerder ontvangen was, en de 0 resp. 1 als ontvangstbevestiging naar L stuurt.
L is het ontvangstbevestigingskanaal, en stuurt een 0 of 1, ontvangen van O, door naar Z. Ook L is onbetrouwbaar, en kan een e doorgeven in plaats van 0 of 1.
In het algemeen beschrijven we een communicatieprotocol in proces algebra als volgt.
Eerst beschrijven we elk der componenten (hier dus Z, K, O, L) door middel van een specificatie, vervolgens stellen we die samen met een merge en een encapsulatie, dan abstraheren we van interne stappen, en tenslotte moeten we dan bewijzen dat we het gewenste externe gedrag hebben. In dit geval moeten we dus specificaties geven voor Z, K, O en L. Om de hoeveelheid formules te beperken, geef ik hier alleen de vergelijking voor L. We hebben

$$L = l6(0)(s5(0) + s5(e))L + l6(1)(s5(1) + s5(e))L,$$

waarbij bijv. $l6(0)$ betekent: lees een 0 bij locatie 6 (afkomstig van O) en s5(e): schrijf een error bij locatie 5 (naar Z).
De communicatiefunctie laat nu elke lees-actie communiceren met de corresponderende schrijf-actie, dus bv. $l6(0)|s6(0) = c6(0)$, waarbij $l6(0)$ uitgevoerd wordt door L en s6(0) door O, en c6(0) betekent dat een 0 gecommuniceerd wordt langs locatie 6. Alle andere communicaties zijn geblokkeerd, gelijk aan δ, dus bijvoorbeeld $l6(0)|s5(0) = \delta$.
De communicatie-'helften' mogen niet los optreden, die moeten we encapsuleren, dus als H bestaat uit alle $ln(d)$, $sn(d)$ voor n = 3,4,5,6 en data d, bekijken we proces

$$\partial_H(Z\|K\|O\|L).$$

Dit proces bevat echter nog interne stappen cn(d) voor n = 3,4,5,6 en data d. Als I de verzameling interne stappen is, en τ_I de abstractie-operator die stappen uit I onzichtbaar maakt, krijgen we proces

$$\tau_I(\partial_H(Z\|K\|O\|L)).$$

Dit proces nu moet het gewenste gedrag vertonen, dus we moeten bewijzen dat

$$\tau_I(\partial_H(Z\|K\|O\|L)) = P,$$

waar P het proces is, dat aan het begin van deze sectie gedefinieerd werd.
Bij het bewijs van deze stelling komen een aantal zaken aan de orde, die nog niet besproken zijn. We stippen deze zaken achtereenvolgens aan.

a. Abstractie. De definitie van abstractie op processen is allerminst eenvoudig, en wij zullen dit onderwerp daarom in dit artikel niet goed kunnen behandelen (zie [8]). In de volgende sectie geven we een indicatie, wat voor problemen er zijn.
b. Fairness. We hebben in het bewijs een aanname nodig dat de kanalen fair zijn. Dit wil zeggen, dat een kanaal niet oneindig vaak achtereen een error mag geven, niet volkomen defectief mag raken. Deze fairness aanname is algebraïsch weergegeven in de 'Koomen's Fair Abstraction Rule'. Deze regel speelt een cruciale rol in het bewijs. Zie voor meer informatie [4, 9].
c. Oneindige processen. De stelling geeft een gelijkheid tussen processen gegeven door middel van vergelijkingen, meestal zijn dat oneindige processen. We hebben een extra bewijsprincipe nodig, het 'Recursive Specification Principle', om te kunnen redeneren over zulke processen. Zie voor meer informatie [4, 9].

Vanwege deze, maar ook andere zaken, valt het bewijs, dat het Alternating Bit Protocol als hier gegeven een goed communicatieprotocol is, buiten het kader van dit artikel. Het volledige bewijs is te vinden in [9].

12 ABSTRACTIE

Zoals in de vorige sectie is beargumenteerd, willen we in staat zijn bepaalde acties weg te abstraheren, onzichtbaar te maken. Hier maken we duidelijk, dat we niet altijd stappen zomaar weg kunnen laten. De reden hiervoor is, dat we willen dat een proces zich voor en na abstractie hetzelfde gedraagt (afgezien van de geabstraheerde acties), en dat dus een proces dat voor abstractie een mogelijkheid tot deadlock heeft, die mogelijkheid ook na abstractie heeft. Zo heeft proces $a + b\delta$ een mogelijkheid tot deadlock, maar als we b weglaten, krijgen we $a + \delta = a$, en dan hebben we geen deadlock. Daarom voeren we een *stille stap* τ in, die we in bepaalde gevallen weg kunnen laten, maar in andere gevallen niet. Abstraheren we nu in $a + b\delta$ van b, dan krijgen we $a + \tau\delta$, en deze τ kan niet weg; abstraheren we in ab van b, dan krijgen we $a\tau$, en hier kunnen we de τ wel weglaten, dit proces is gelijk aan a.
In het algemeen kunnen we drie wetten formuleren (de τ-wetten van Milner), die het gedrag van τ precies beschrijven. We geven deze wetten in tabel 6.

1. $x\tau = x$
2. $\tau x + x = \tau x$
3. $a(\tau x + y) = a(\tau x + y) + ax$

Tabel 6: τ-wetten

Merk op dat we τ niet als atoom kunnen beschouwen (niet alle wetten voor atomaire acties zullen opgaan voor τ). Deze τ-wetten kunnen we nu toevoegen aan het systeem BPA (tabel 1), maar ook, als we de interactie van τ en $\|$ in wetten vastleggen, aan de systemen PA (tabel 2) en ACP (tabel 5).
Dan is de *abstractie-operator* τ_I (met I een verzameling atomaire acties) de operator die alle acties uit I hernoemt in τ, en verder niets doet. Deze operator τ_I heeft wetten analoog die van ∂_H uit tabel 4. Zie [8] voor meer details over stille stap en abstractie.

13 CONCLUSIES

We hebben in het voorgaande gezien, dat wij in proces algebra operatoren kunnen definiëren, die zaken beschrijven als sequentiële compositie, deterministische en non-deterministische keuze, parallelle compositie, deadlock, synchrone communicatie, abstractie van interne stappen en fairness. Er bestaan nog andere systemen in proces algebra waarmee zaken besproken kunnen worden als synchroon parallellisme, asynchrone communicatie, projectie, interrupts, tight regions, divergentie, het alfabet van een proces, algemene herbenoemingen, localisatie, restrictie, en de toestand (state) van een proces (zie bv. [3, 7]).
Met zoveel operatoren is de specificerende kracht van proces algebra enorm. In dit artikel hebben we enkele voorbeelden gezien van specificaties, zoals die van een

stack. Met behulp van de state operator kunnen we zelfs een programma in een hogere programmeertaal rechtstreeks, en op een mechanische manier, vertalen in een specificatie in proces algebra. Daarom voldoet proces algebra aan de specificatie-eis voor een goede concurrency theorie, geformuleerd in de eerste sectie.
Aan de hand van het voorbeeld van het alternating bit protocol hebben we gezien wat een verificatie inhoudt, een verificatie dat een proces waarvan de componenten gegeven zijn door specificaties, extern het juiste, vooraf gespecificeerde gedrag vertoont. Inmiddels zijn in proces algebra ingewikkelder protocollen geverifieerd, met behulp van nieuwe principes, zodat proces algebra op weg is, om ook aan de verificatie-eis voor een goede concurrency theorie te voldoen.
Tenslotte ligt het gebruik van proces algebra bij ontwerp nog in de toekomst, maar lijkt bijvoorbeeld het ontwerp van een nieuwe programmeertaal gebaseerd op proces algebra al bijna binnen bereik te zijn.
Concluderend wil de auteur stellen, dat proces algebra een goede theorie over concurrency is, die naast eerder genoemde andere concurrency theorieën een eigen plaats dient in te nemen.

REFERENTIES.

1. Apt, K. R., N. Frances & W. P. de Roever, *A proof system for communicating sequential processes,* ACM Trans. on Progr. Lang. & Systems 2(3), 1980.
2. Austry, D. & G. Boudol, *Algebre de processus et synchronisation,* Theor. Comp. Sci. 30(1), 1984, blz. 91-131.
3. Baeten, J. C. M., J. A. Bergstra & J. W. Klop, *Syntax and defining equations for an interrupt mechanism in process algebra,* rapport CS-R8503, Centrum voor Wiskunde en Informatica, Amsterdam, 1985.
4. Baeten, J. C. M., J. A. Bergstra & J. W. Klop, *On the consistency of Koomen's Fair Abstraction Rule,* rapport CS-R8511, Centrum voor Wiskunde en Informatica, Amsterdam, 1985.
5. De Bakker, J. W. & J. I. Zucker, *Processes and the denotational semantics of concurrency,* Inf. & Control 54-(1/2), 1982, blz. 70-120.
6. Bartlett, K. A., R. A. Scantlebury & P. T. Wilkinson, *A note on reliable full-duplex transmission over half duplex lines,* C. Assoc. Comp. Mach. 12(5), 1969.
7. Bergstra, J. A. & J. W. Klop, *Process algebra for synchronous communication,* Inf. & Control 60(1/3), 1984, blz. 109-137.
8. Bergstra, J. A. & J. W. Klop, *Algebra of communicating processes with abstraction,* Theor. Comp. Sci. 37(1), 1985, blz. 77-121.
9. Bergstra, J. A. & J. W. Klop, *Verification of an alternating bit protocol by means of process algebra,* rapport CS-R8404, Centrum voor Wiskunde en Informatica, Amsterdam, 1984.
10. Bergstra, J. A. & J. V. Tucker, *Top-down design and the algebra of communicating processes,* Sci. of Comp. Progr. 5(2), 1985, blz. 171-199.
11. Brookes, S., C. A. R. Hoare & W. Roscoe, *A theory of communicating sequential processes,* J. Assoc. Comp. Mach., 31(3), 1984, blz. 560-599.
12. Graf, S. & J. Sifakis, *A modal characterization of observational congruence on finite terms of CCS,* Proc. 11th Int. Colloq. Aut. Lang. & Progr., Antwerpen 1984, J. Paredaens, ed., Springer LNCS 172, blz. 222-234.
13. Hennessy, M., *A term model for synchronous processes,* Inf. & Control 51(1), 1981, blz. 58-75.
14. Hennessy, M., *Synchronous and asynchronous experiments on processes,* Inf. & Control 59(1/3), 1983, blz. 36-83.
15. Hoare, C. A. R., *Communicating sequential processes,* Prentice Hall Int'l Series in Comp. Sci., 1985.
16. Hoare, C. A. R. & E.-R. Olderog, *Specification-oriented semantics for communicating processes,* Proc. 10th Int. Colloq. Aut. Lang. & Progr., Barcelona 1983, J. Diaz eds., Springer LNCS 154, blz. 561-572.
17. Koomen, C. J., *A structure theory for communication network control,* proefschrift T.H. Delft, april 1982.
18. Mazurkiewicz, A., *Traces, histories, graphs: instances of a process monoid,* Math. Found. Comp. Sci., Springer LNCS 176, 1984, blz. 115-133.
19. Meyer, J.-J. Ch., *On merging regular processes by means of fixed point theory,* te verschijnen in Theor. Comp. Sci.
20. Milne, G. J., *Circal: a calculus for circuit description,* Integration, the VLSI Journal 1, 1983, blz. 121-160.
21. Milne, G. J., *Circal and the representation of communication, concurrency, and time,* ACM Trans. on Progr. Lang. & Systems, 7(2), 1985, blz. 270-298.
22. Milner, R., *A calculus of communicating systems,* Springer LNCS 92, 1980.
23. Milner, R., *Calcull for synchrony and asynchrony,* Theor. Comp. Sci. 25, 1983, blz. 267-310.
24. Reisig, W., *Petri nets,* EATCS monograph on Theor. Comp. Sci., Springer 1985.
25. Rem, M., *Partially ordered computations, with applications to VLSI design,* Proc. 4th Advanced Course on Found. Comp. Sci., part 2, eds. J. W. de Bakker & J. van Leeuwen, Tract 159, Mathematisch Centrum, Amsterdam, 1983.