Informatica als wetenschap : formele specificatie en wiskundige verificatie

Informatica als wetenschap

Formele specificatie en wiskundige verificatie

INTREEREDE

Prof.dr. J.C.M. Baeten

Technische Universiteit Eindhoven

INTREEREDE

Uitgesproken op vrijdag 30 oktober 1992 aan de Technische Universiteit Eindhoven.

Prof.dr. J.C.M. Baeten

Mijnheer de Rector Magnificus,
Dames en Heren,

Ik wil het met U hebben over de informatica als wetenschappelijke discipline. We hebben de neiging te vergeten, dat het nog maar net 10 jaar geleden is, dat de informatica als studierichting werd ingesteld aan de Nederlandse algemene en technische universiteiten. In die tijd heeft de informatica zich een zodanige plaats verworven aan de universiteiten en ook in ons dagelijks leven, dat het niet zo opvalt, dat informatica als wetenschap nog in de kinderschoenen staat. Ik wil in dit betoog beargumenteren, dat de informatica nog een lange weg te gaan heeft, voordat het zich als volwaardige wetenschap kan presenteren. Mijn eigen onderzoek, het ontwikkelen van logische en wiskundige ondersteuning voor het bouwen van informatiesystemen, wil hieraan een bijdrage leveren.

1. Een vergelijking

Laten wij met behulp van een vergelijking de stand van zaken in de informatica nader bepalen. Omdat wij ons aan een technische universiteit bevinden, wil ik mij beperken tot de informatica als technische wetenschap. Welnu, een belangrijke taak van iemand die aan deze universiteit is afgestudeerd als informatica-ingenieur, is het ontwerpen van informatiesystemen. Laten we een vergelijking maken met een afgestudeerd bouwkundig ingenieur, wiens taak het is bruggen te ontwerpen.

Ik wil in de wetenschap van het bouwen van bruggen drie stadia onderscheiden. In het verste verleden verliep het bruggen bouwen door het uit te proberen, laten we dit het "probeer-maar-wat"-stadium noemen. Verscheidene constructies worden uitgeprobeerd, en de praktijk leert welke meer succesvol zijn dan andere.

Vervolgens komen we – zeg in de middeleeuwen – aan het heuristische stadium. In dit stadium zijn er, door "trial and error" beproefde methoden ontstaan, en kan men door kopiëren en aanpassing van succesvolle exemplaren goede bruggen bouwen. Zelfs is het mogelijk, door de beste technieken en methoden uit het verleden te combineren, langere bruggen te bouwen dan ooit tevoren. Maar dan bevinden we ons op glad ijs, en een middeleeuws bouwmeester zal geen enkele garantie kunnen geven, dat zijn produkt grote natuurkrachten of hoge belasting kan weerstaan.

Het derde stadium wil ik het moderne stadium noemen. Pas in dit stadium kunnen we spreken van de wetenschap van het bouwen van bruggen. Nu begint het bouwen van een brug aan de tekentafel. De brug wordt daar precies beschreven, materiaalkeuze

en vele andere aspecten worden vastgelegd. Vervolgens wordt het ontwerp doorgerekend. De bouwkundig ingenieur hanteert daarbij abstracte begrippen, zoals trekkracht, belasting, enz. Met deze grootheden wordt vervolgens gerekend, hij past een stuk wiskunde toe. Delen van deze wiskunde zijn speciaal ontworpen voor deze ingenieurs-discipline. Hij zal misschien differentiaalvergelijkingen oplossen, Fourier-analyse toepassen, en andere specifieke stukken wiskunde gebruiken. Hij maakt zich daarbij geen zorgen om de fundering of herkomst van deze wiskunde, maar beoordeelt deze louter op bruikbaarheid. Hij kan eventueel programmatuur gebruiken, die delen van de berekening voor hem automatiseert, maar dit zal waarschijnlijk een interaktief proces blijven, dit is alleen ondersteunend. Als de berekeningen kloppen, wordt een blauwdruk gemaakt, en is het werk van de bouwkundig ingenieur gedaan. De wetenschappelijke methode geeft een grotere zekerheid dat de brug later zal blijven staan (maar geen absolute zekerheid), en maakt het ook mogelijk nieuwe ontwerpen te bekijken, en veel langere bruggen te bouwen, in moeilijkere omstandigheden dan voorheen.

We hebben nu drie stadia onderscheiden in de discipline van het bruggen bouwen: het "probeer-maar-wat"-stadium, het heuristische stadium en het moderne stadium.

2. Informatica in het heuristische stadium

Laten wij nu onze aandacht richten op de informatica. Hier kunnen we dezelfde stadia onderscheiden. Ik wil de stelling verdedigen, dat de wetenschappelijke discipline van het bouwen van informatiesystemen nog in het heuristische stadium verkeert, zich nog in de middeleeuwen bevindt. Als zodanig is informatica nog geen echte wetenschap.

Na de tweede wereldoorlog zijn computers geïntroduceerd, en gaandeweg is ons duidelijk geworden, wat je daar zo al mee kunt doen. Met veel uitproberen bleek dat je er eigenlijk heel veel mee kunt doen. De eerste programma's werden in machinecode aan de computer geleverd en meestal deed de computer niet wat er van verwacht werd. Om in termen van de vergelijking te blijven: meestal stortte de brug in. Het gebruik van hogere programmeertalen en veel ervaring hebben daar enorme verandering in gebracht, en tegenwoordig doet een computer meestal wel, wat we er van verwachten. We kunnen zeggen, dat we het "probeer-maar-wat"-stadium te boven zijn gekomen, en er zijn vele methoden en technieken in de handel, om op een gestructureerde wijze tot het ontwerp van een informatiesysteem te komen. Maar dit zijn alle heuristische methoden, we hebben de succesvolle probeersels uit het verleden gevangen in diagrammen, stappenplannen, life cycles, faseringen,

analyses en wat niet al. We zijn hiermee erg ver gekomen. Zelfs is het mogelijk, door de beste technieken en methoden uit het verleden te combineren, grotere en complexere informatiesystemen te bouwen dan ooit tevoren. Maar dan bevindt de informatica-ingenieur zich op glad ijs, en een verstandig ontwerper zou geen enkele garantie moeten geven, dat zijn produkt in alle omstandigheden betrouwbaar blijft werken. Dat was ook de reden voor de Canadees Parnas om zich terug te trekken uit het SDI project (ook wel Star Wars geheten): hij kon geen garantie geven, dat een zo complex informatiesysteem als ter ondersteuning van SDI nodig was, in alle omstandigheden correct zou werken (zie [Par86]). Een ander voorbeeld is het Rotterdamse walradarsysteem, dat niet als één geheel gebouwd kon worden, omdat het informatiesysteem te complex werd. Deze lijst kan moeiteloos uitgebreid worden met andere voorbeelden.

Laten we ons vervolgens richten op wat er nodig is opdat het ontwerpen van informatiesystemen het moderne stadium kan ingaan. Pas dan immers kan de informatica zich een volwaardige wetenschap noemen. Ik wil hier stellen dat hiervoor tenminste twee zaken nodig zijn, te weten formele specificatie en wiskundige verificatie. Op beide vereisten zal ik in het vervolg ingaan.

3. Formele specificatie

Allereerst richten we ons op specificatie. Laten we hiertoe weer de analogie met de bruggenbouwer volgen. Het is nodig, dat we een informatiesysteem aan de tekentafel precies vastleggen. We moeten de componenten van zo'n systeem precies beschrijven, specificeren, op het gewenste niveau van abstractie (en tijdens de ontwerpfase zijn waarschijnlijk verschillende niveaus van abstractie nodig). Natuurlijke taal leent zich niet zo goed als beschrijvingstaal, vanwege inherente vaagheden en dubbelzinnigheden. Maar dit geldt ook voor de meeste tegenwoordig in de praktijk gebruikte beschrijvingstechnieken, die zijn ook niet toereikend. Bekijken we als voorbeeld de bekende dataflow diagrammen (DFD's). De betekenis van zo'n DFD is moeilijk exact te bepalen. Er zijn meerdere systemen, die naar ons gevoel totaal verschillend gedrag hebben, maar toch hetzelfde DFD hebben. Een DFD geeft slechts een beperking aan het communicatiegedrag van de verschillende componenten van een systeem. Ook is een DFD statisch, en kunnen we niet het verloop van acties in de tijd eruit afleiden. Daarom gebruiken sommige methoden naast DFD's nog een andere beschrijving van het te bouwen systeem. Dan rijst meteen de vraag, wat dan wel de relatie is tussen beide beschrijvingsmethoden. Soortgelijke overwegingen kunnen we maken bij de meeste in de praktijk gebruikte beschrijvingsmethoden.

Wat zijn dan de kenmerken van een exacte specificatie? Welnu, we hebben een formele specificatietaal nodig. Zo'n taal moet een formele syntax hebben, dat wil zeggen er moet precies vastliggen wat de geldige expressies in de taal zijn. Verder moet zo'n taal minstens één formele semantiek hebben, dat wil zeggen de betekenis van elke expressie moet eenduidig vastliggen. Deze betekenis zal meestal gegeven worden door middel van een vertaling van de taal-expressies naar een wiskundige structuur. Deze vertaling moet zodanig zijn dat de betekenis van een samengestelde expressie op een éénduidige manier afhangt van de betekenis van zijn onderdelen. In andere termen gezegd: de operatoren van de taal, de taalconstructies, zijn bij voorkeur ontleend aan de (wiskundige) logica. Een taal die aan deze vereisten voldoet, noem ik een formele specificatietaal. Een additionele eis die vaak wordt opgelegd, is de eis dat er enige computerondersteuning voor de taal is, dat er enige programma's zijn die ons helpen met formele specificaties om te gaan. Zo wil men vaak dat een specificatie computerleesbaar is. Dan moet automatisch nagegaan kunnen worden, of een gegeven specificatie voldoet aan de syntactische regels van de taal (type checking). Een scala van andere software tools vindt men beschreven in de literatuur.

Iets wat dan meteen opvalt, is dat de meeste programmeertalen voldoen aan de eisen van een formele specificatietaal. Dit is juist, we kunnen een programmeertaal ook beschouwen als een formele specificatietaal. Een programmeertaal heeft echter nog de extra eis dat hij executeerbaar moet zijn: een specificatie geschreven in een programmeertaal kan uitgevoerd worden door een computer. Een specificatie geschreven in een specificatietaal hoeft niet implementeerbaar te zijn: we kunnen in een specificatietaal een aantal eisen aan een systeem opleggen die niet verenigbaar zijn. Dit moet dan blijken in de volgende ontwerpstappen.

Er wordt al zeker 10 jaar onderzoek naar formele specificatietalen gedaan, en er zijn ook al vele formele specificatietalen voorgesteld, maar deze hebben hun weg naar de praktijk van het ontwerpen van informatiesystemen nog niet gevonden. Het onderzoek naar het ontwerp van specificatietalen beschouw ik niet als onderzoek in de informatica, maar als onderzoek in de toegepaste logica. Zo is ook het ontwerp van programmeertalen het werkterrein van de logicus. Ik wil enkele voorbeelden van formele specificatietalen noemen. De twee specificatietalen die het verst zijn gevorderd op de weg naar de praktijk zijn VDM en Z (zie [Jon90], [Spi88]). Beide talen specificeren een systeem door een definitie van de mogelijke toestanden en een definitie van de operaties die het systeem kan uitvoeren. Verder wordt gewerkt met pre- en postcondities, ofwel invarianten. Een interessante andere formele specifi-

catietaal is de aan het Philips Natuurkundig Laboratorium ontwikkelde ontwerptaal COLD (Common Object-Oriented Language for Design, zie [FeJ92]. COLD is een zogenaamde breed-spectrum specificatietaal, dat wil zeggen dat het hele systeemontwikkeltraject erin beschreven kan worden. In enkele produktdivisies van Philips is met het gebruik van COLD inmiddels ervaring opgedaan. Een nadeel van bovengenoemde talen is dat ze niet erg goed kunnen omgaan met de beschrijving van parallelle of gedistribueerde systemen. Daarom zijn uitbreidingen voorgesteld die erop zijn gericht dit beter te kunnen (zie bijvoorbeeld [Mid90]), maar ook zijn specificatietalen ontwikkeld specifiek gericht op parallelle of gedistribueerde systemen, zoals μ-CRL (Micro Common Representation Language, (zie [GrP90]) ontwikkeld aan het Centrum voor Wiskunde en Informatica in Amsterdam. Voor alle duidelijkheid: met een parallel systeem bedoel ik een systeem waarbij verscheidene processoren gezamenlijk aan een taak werken (zoals in een parallelle computer), met een gedistribueerd systeem bedoel ik een systeem met een aantal zelfstandige componenten, die onderling informatie uitwisselen (communiceren). In [BeR89] (waar een aantal observaties hierboven gemaakt ook te vinden zijn) wordt betoogd dat formele specificatietalen op dit moment al met vrucht industrieel inzetbaar zijn voor de specificatie van informatiesystemen.

Ik heb betoogd dat een formele specificatietaal een vereiste is voor een exacte beschrijving van informatiesystemen. Dit is nodig opdat het ontwerpen van informatiesystemen het moderne stadium kan ingaan. Laten we vervolgens kijken naar de tweede vereiste, wiskundige verificatie.

4. Wiskundige verificatie

Als een informatiesysteem eenmaal gespecificeerd is, moet het doorgerekend worden, moeten we (vóór de bouw) kunnen nagaan of het ontwerp aan de verwachtingen zal beantwoorden. Net zo min als we het van de bouwkundig ingenieur zouden accepteren dat hij de juistheid van een ontwerp voor een brug verifieert door hem eerst te bouwen, en er vervolgens met zwaar materieel over heen te rijden, moeten we het van een informatica-ingenieur accepteren dat hij de verificatie van het ontwerp voor een informatiesysteem verschuift naar een testfase. In de huidige praktijk van het ontwerpen van informatiesystemen gebeurt dit laatste nog al te vaak. Een modernere opvatting op dit vlak is het principe van de prototypering; daarbij worden slechts stukjes van het systeem gebouwd en vervolgens geverifieerd. In situaties waar testen achteraf of onderweg niet mogelijk is, zoals bij het informatiesysteem van een satelliet, gaat men er wel toe over redundantie in te bouwen. Een systeem wordt bijvoorbeeld driemaal uitgevoerd, en de meerderheid

beslist. Het moge duidelijk zijn dat fouten hierbij niet uit te sluiten zijn en we worden dan ook regelmatig geconfronteerd met incorrect werkende systemen. Er zijn twee ontwikkelingen die deze situatie verergeren.

De eerste ontwikkeling is onze toenemende afhankelijkheid van informatiesystemen. Informatiesystemen dringen steeds verder door in ons dagelijks leven. Zo kunnen steeds meer mensen niet werken "als de computer het niet doet". Verder worden informatiesystemen steeds meer ingezet in omgevingen waar het optreden van fouten desastreuze gevolgen heeft, zoals besturingssystemen van vliegtuigen, auto's en kerncentrales. Bovendien worden steeds meer informatiesystemen aan elkaar gekoppeld, zodat fouten grote gevolgen kunnen hebben. Zo kon onlangs een fout in een lokale centrale het gehele interlokale telefoonverkeer van de Verenigde Staten een tijdlang onmogelijk maken.

De tweede ontwikkeling die de situatie verergert is het toenemend gebruik van parallelle en gedistribueerde systemen. Omdat bij dergelijke systemen het totale aantal toestanden van het systeem het product is van de aantallen toestanden van de componenten, wordt het totale aantal toestanden al gauw erg groot. Hierdoor heeft een ontwerper niet meer een overzicht over alle mogelijkheden, en zullen er al gauw situaties optreden die niet voorzien zijn. Men wijt het neerstorten van enkele exemplaren van een nieuw type Airbus aan het optreden van zulke onvoorziene situaties.

Om de bedrijfszekerheid van onze informatiesystemen te verhogen, is een wiskundige verificatie van het ontwerp noodzakelijk. Zoals de bruggenbouwer zijn ontwerp doorrekent en daarbij gebruik maakt van specifieke stukken wiskunde, dient ook de verificatie van een ontwerp voor een informatiesysteem een berekening te zijn, de toepassing van een stuk wiskunde. Zo'n stuk wiskunde, zo'n calculus, moet dan wel eerst ontworpen worden.

Wat zijn dan de vereisten voor een verificatiecalculus? Allereerst moet zo'n calculus gaan over de juiste abstracte begrippen. Zoals de bruggenbouwer berekeningen uitvoert over trekkracht en belasting, zal de informatica-ingenieur uitgaan van basisbegrippen van informatiesystemen. Deze basisbegrippen zijn abstracties, geen tastbare grootheden zoals draden en weerstanden. Het zal gaan over toestanden of acties van een systeem, en verschillende manieren om systeemcomponenten aan elkaar te koppelen. De essentie is natuurlijk dat er met zo'n calculus gerekend moet kunnen worden: we hebben een aantal wetten nodig, een aantal rekenregels. Natuurlijk moet de betekenis van de expressies van de calculus ook hier precies vastliggen, maar hier ligt niet zo'n nadruk op formele syntax en semantiek als in het geval van specifi-

catie. Gebruiksgemak staat hier eerder voorop, en de ingenieur zal zich naar hartelust bedienen van afkortingen, handigheidjes, heuristieken enzovoort. Computerondersteuning voor een verificatiecalculus is ook niet noodzakelijk, maar kan in gevallen dat de berekening zeer omvangrijk wordt wenselijk zijn. Om expressies computerleesbaar te maken is natuurlijk wel een formele syntax vereist.

Naar de ontwikkeling van verificatiecalculi wordt al zeker 10 jaar onderzoek gedaan. Voor de duidelijkheid: onderzoek naar de ontwikkeling van zo een calculus beschouw ik niet als onderzoek in de informatica, maar als onderzoek in de toegepaste wiskunde. Er zijn inmiddels meerdere calculi voorgesteld, maar deze hebben hun weg naar de praktijk van het ontwerpen van informatiesystemen nog niet gevonden. De bekendste calculus voor de verificatie van sequentiële systemen is de Floyd-Hoare logica (zie [Apt81]). Echter, deze calculus toont tekortkomingen als we parallelle of gedistribueerde systemen willen verifiëren. Zoals eerder beargumenteerd, hebben we juist voor dergelijke systemen een grote behoefte aan een verificatiecalculus. Daarom zijn er specifieke stukken wiskunde ontwikkeld voor de verificatie van parallelle en gedistribueerde informatiesystemen, zogenaamde concurrency-theorieën. Sinds 1984 doe ik onderzoek op het gebied van de concurrency, en dat is een reden om er hier iets dieper op in te gaan.

5. Concurrency

Concurrency-theorieën kan men indelen naar aard. Zo zijn er grafische, logische en algebraïsche theorieën. De theorie van Petri-netten is een voorbeeld van een grafische theorie (zie [Rei85]), en de theorie van de temporele logica is een voorbeeld van een logische theorie (zie [MaP92]). In mijn groep aan deze technische universiteit wordt veel belangwekkend onderzoek op het gebied van de temporele logica en andere assertionele methoden gedaan. Het belang van een calculus voor concurrency werd voor het eerst duidelijk onder woorden gebracht door Robin Milner, dit jaar winnaar van de Turing Award, die in 1980 zijn boek over CCS (de Calculus of Communicating Systems) publiceerde [Mil80]. CCS is een algebraïsche concurrency-theorie, ook wel een procesalgebra genoemd. Zoals in de universele algebra de theorie van groepen, ringen, lichamen enz. gegeven wordt door een stel wetten, axioma's, zo is het streven van een algebraïsche concurrency-theorie gericht op het vinden van een vergelijkbaar stel wetten voor concurrente, parallelle processen.

Naast CCS zijn de andere twee meest bekende procesalgebra's CSP (zie [Hoa85]) en ACP. Deze laatste, ACP (de Algebra of Communicating Processes), in 1982 voor het eerst geformuleerd door Jan Bergstra en Jan Willem Klop [BeK82], wordt gekenmerkt door een consequente axiomatische

en algebraïsche aanpak. Hiermee wordt bedoeld dat uitgegaan wordt niet van een semantiek, van de betekenis van expressies in een wiskundige structuur, maar van de rekenregels, van de axioma's. Bij een semantische concurrency-theorie wordt uitgegaan van een bepaald model van parallelle processen. Zo'n model is een wiskundige structuur, en er is een vertaling van de syntax van de taal naar het model. Vervolgens wordt dan vastgesteld welke gelijkheden tussen processen, welke rekenregels gelden in het model. Er is een heel scala van concurrencymodellen, al naar gelang welke aspecten van processen men van belang acht, en van welke aspecten men wil abstraheren. Hier kom ik later nog op terug. De meeste concurrency-theorieën, ook CCS en CSP, zijn semantisch. In tegenstelling hiermee is ACP axiomatisch, dat wil zeggen, dat uitgegaan wordt van een stel axioma's, een stel rekenregels dat moet gelden en dat men daarna vrij is elk model te beschouwen waarin alle axioma's gelden. Deze situatie is te vergelijken met groepentheorie, waar elke structuur die voldoet aan de groepsaxioma's, een groep wordt genoemd.

Twee boeken uit 1990 ([BaW90], [Bae90]) geven een overzicht van de theorie van ACP en een aantal voorbeelden van berekeningen in de calculus ACP. Zo vindt men hierin berekeningen over communicatieprotocollen, een protocol voor wederzijdse uitsluiting ("mutual exclusion"), Computer Integrated Manufacturing, een systolisch systeem, een sorteeralgoritme, een operating system, de semantiek van een programmeertaal. Inzicht in wat een gebruiker van de calculus nodig heeft ter verhoging van het gebruiksgemak, kunnen we immers slechts krijgen door veel voorbeelden door te rekenen.

Met concurrency-theorieën zoals ACP worden berekeningen ter ondersteuning van het ontwerp van informatiesystemen in principe mogelijk. Ik zeg in principe, omdat er nog vele moeilijkheden overwonnen moeten worden, voor dergelijke theorieën ook in de praktijk bruikbaar worden. Een belangrijke moeilijkheid lijkt te zijn, dat deze theorieën werkelijke informatiesystemen niet aankunnen, vanwege hun grootte en complexiteit. De oplossing kan liggen in het optimaliseren van berekeningen, door het toevoegen van afkortingen, handigheidjes en heuristieken, maar van belang is ook het ontwikkelen van een goede computerondersteuning bij het uitvoeren van berekeningen. Dan nog is het volgens mij een heilloze weg te proberen alle mogelijke toestanden van een systeem door te rekenen (het aantal hiervan gaat ook al gauw de capaciteit van een computer te boven), maar moeten we zoeken naar andere methoden. Op dit gebied zijn interessant de PSF toolkit, gebaseerd op ACP, ontwikkeld aan de Universiteit van Amsterdam (zie [MaV93]), maar ook bijvoorbeeld PAM (Process Algebra Manipulator, (zie [Lin91]) ont-

wikkeld aan de Universiteit van Sussex.

Wiskundige berekeningen bij het ontwerp van informatiesystemen moeten het mogelijk maken grotere zekerheid te krijgen over de correctheid van dergelijke systemen, en moeten het mogelijk maken grotere en complexere systemen te bouwen dan tot nu toe. Wellicht ook kan zelfs nieuwe functionaliteit verwezenlijkt worden.

Ik heb in het voorgaande beargumenteerd dat formele specificatie en wiskundige verificatie nodig zijn, voordat het ontwerpen van informatiesystemen het moderne stadium kan ingaan. Pas dan kan de informatica zich manifesteren als volwaardige wetenschap naast de andere ingenieurswetenschappen aan een technische universiteit. Het ontwerpen van formele specificatietalen is een taak van de toegepast logicus, het ontwerpen van een verificatiecalculus is een taak van de toegepast wiskundige. Bij beide activiteiten is een nauw overleg met de praktizerend informatica-ingenieur noodzakelijk. Beide activiteiten kunnen goed uitgevoerd worden door een theoretisch informaticus met een achtergrond in logica en wiskunde. Daarom acht ik onderzoek naar formele specificatietalen en verificatiecalculi een zinvolle invulling van mijn taak als hoogleraar theoretische informatica aan deze universiteit.

Vervolgens wil ik twee onderwerpen behandelen, die voor een algemeen publiek waarschijnlijk minder interessant zijn, maar meer bedoeld zijn voor vakgenoten. Het eerste betreft het verschil tussen logica en wiskunde bij het onderzoek naar ondersteuning van systeemontwerp, het tweede betreft de verschillende concurrency-modellen en de scholenstrijd in de theoretische informatica.

6. Logica en wiskunde

De invloed van logici op het onderzoek naar de ondersteuning van systeemontwerp is de laatste jaren sterk toegenomen, en dit is toe te juichen. Dit zeg ik omdat het ontwerp van programmeertalen en specificatietalen beschouwd kan worden als toegepaste logica. Hierbij is een rigoureus taalontwerp noodzakelijk, en logici met hun ervaring in de analyse van de taal van de wiskunde zijn hiervoor met name geschikt. Ook bij de constructie van computerondersteuning van verificatieberekeningen is een logicus nuttig, omdat hier eveneens een exacte syntax noodzakelijk is.
Anders wordt het als het gaat om de ontwikkeling van wiskunde voor een verificatiecalculus. Een informatica-ingenieur die een systeemontwerp wil verifiëren, wil alle wiskunde gebruiken die voorhanden is, en maakt zich niet druk om de fundering of exacte formulering van zijn berekening, maar alleen om het gemak van de berekeningsmethode en de uitkomst. Hier bedrijven we dus geen logica, maar wiskunde. Het is daarom een misvat-

ting om zich druk te maken over de exacte syntax van een verificatiecalculus, zich te verzetten tegen te veel of te wilde uitbreidingen van een calculus, of te vinden dat het gebruik van reële getallen te ver gaat: de bruggenbouwer gebruikt reële en complexe getallen naar hartelust. Deze misvatting zien we echter opduiken, omdat theoretische informatica in zijn geheel gezien wordt als toegepaste logica. We moeten ons realiseren dat sommige onderdelen van de theoretische informatica geen toegepaste logica zijn maar toegepaste wiskunde. We moeten dan ook bevorderen dat er meer wiskunde in de theoretische informatica komt (zoals in [Ver92]).

7. Verschillen tussen concurrency-theorieën

Het tweede onderwerp waar ik iets over wil zeggen betreft de verschillen tussen de concurrency-theorieën onderling. We hebben verschillende invalshoeken (grafisch, logisch, algebraïsch), maar er bestaat wel grote overeenstemming over wat de juiste abstracte begrippen zijn. Bijna alle concurrency-theorieën gaan uit van een verzameling basiselementen (atomaire acties, observaties of events genaamd) en kennen standaardoperaties om grotere processen op te bouwen uit kleinere, te weten alternatieve compositie (keuze), sequentiële compositie (volgorde) en parallelle compositie (met communicatie). Daarom is het opvallend dat over de rekenregels van deze standaardoperatoren zeer weinig overeenstemming bestaat.

Twee verschillen, die haast tot controversen zijn geworden, wil ik even noemen, namelijk branching time tegenover linear time en interleaving tegenover partial order. In branching time worden de momenten van keuze in een proces bijgehouden, in linear time wordt daarvan geabstraheerd. Een belangrijk argument voor branching time is dat processen die vast kunnen komen te zitten (in een deadlock geraken), onderscheiden kunnen worden van processen die altijd voortgang kunnen boeken. Een belangrijk argument voor linear time is dat er eenvoudiger gerekend kan worden.

In interleaving semantiek kan in veel gevallen parallelle compositie uitgedrukt worden in termen van alternatieve en sequentiële compositie (men zegt: de acties van de componenten worden verweven in de tijd), in partial order semantiek kan dat niet en kunnen we spreken over causaliteit. Een analyse van deze laatste controverse is te vinden in een recent artikel [Bae92].

Om eerder genoemde theorieën te plaatsen: de theorie van Petri-netten is een voorbeeld van een branching time partial order theorie en de procesalgebra's CCS, CSP, ACP zijn branching time interleaving. Een voorbeeld van een linear time theorie is trace theorie, waarmee hier in Eind-

hoven veel successen zijn behaald (zie bijvoorbeeld [Rem87]).

Het is heel natuurlijk dat we bij de ontwikkeling van een nieuw vakgebied verschillende theorieën zien die met elkaar in competitie zijn. De discussie over welke theorieën het best toegepast kunnen worden in welke situaties moet echter zakelijk gevoerd worden, en niet in termen van geloof, of in termen van goed en kwaad. Ik ben het dan ook volledig eens met mijn voorganger Willem Paul de Roever, als hij in zijn intreerede in Kiel zegt dat we meer vergelijkende wetenschappelijke evaluatie nodig hebben [Roe91]. In dit licht bezien is het extra teleurstellend dat de Basic Research Action SPEC waarin hij deelnam, er met name voor gezorgd heeft dat het verenigde Europese concurrency-project ACE niet tot stand is gekomen. Nu dreigen de verschillende theorieën, juist door het bestaan van allerlei externe geldbronnen waar wetenschappers steeds meer van afhankelijk worden (zoals ESPRIT), te snel geïnstitutionaliseerd te worden, te snel tot scholen te verworden. In de vakgroep Informatica van deze universiteit wordt onderzoek verricht naar Petri-netten, trace-theorie, temporele logica en procesalgebra. Daardoor hebben we een uitstekende kans om muren te doorbreken, die tussen verschillende scholen zijn opgericht. We moeten deze kans niet voorbij laten gaan.

8. Onderwijs

Nu wil ik tenslotte nog iets zeggen over het onderwijs in de informatica. Een informatica-ingenieur dient weliswaar enige basiskennis te hebben van programmeertalen, maar we moeten er naar mijn mening op letten dat het leren programmeren maar een klein deel van het curriculum in beslag neemt. Andere opleidingen leiden mensen op tot programmeur. Belangrijker is het, studenten de achterliggende principes van programmeertalen bij te brengen, en de juiste abstracties voor het ontwerpen van informatiesystemen. Daarom is het van belang dat er ruimte in het curriculum is voor zaken als lambda-calculus, als paradigma van functioneel programmeren, en voor resolutie-logica, als paradigma van logisch programmeren. Het lopende curriculum aan deze universiteit is toereikend in deze opzichten.

Verder moge het uit het voorgaande betoog duidelijk zijn, dat er naar mijn mening aandacht in het curriculum dient te zijn voor formele specificatie en wiskundige verificatie. Op dit vlak is er in het basiscurriculum alleen aandacht voor verificatie van sequentiële (dat wil zeggen niet-parallelle) systemen. In het bovenbouwcurriculum komen formele specificatie en verificatie van parallelle systemen in keuze-vakken aan de orde. Ik zou het een goede ontwikkeling vinden als er ruimte in het onderbouwcurriculum zou komen voor formele specificatie en als de

voorgestelde afstudeerrichting "Specificatie en Verificatie" zou worden goedgekeurd. Ook zou ik het een goede zaak vinden, als het college algebraïsche specificaties, dat dit jaar als caputcollege gegeven zal worden, omgezet zou worden in een regulier college.

Zolang het nog niet algemeen aanvaard is, dat het gebruik van formele specificatie en wiskundige verificatie een vooruitgang betekent voor het ontwerp van informatiesystemen, kan ik billijken, dat er in het curriculum plaats blijft voor heuristische methoden ter ondersteuning van het ontwerp van informatiesystemen. Echter, zo gauw er intern op dit punt overeenstemming is bereikt, dient een vooruitstrevende universiteit hier het voortouw te nemen, en zich niet te laten weerhouden door geluiden vanuit de zogenaamde praktijk.

9. Tot besluit

Ik wil deze rede besluiten met het uitspreken van enkele dankwoorden. Ik ben hier zeer terughoudend, en noem niet degenen die mij persoonlijk zeer na staan en erg veel voor mij betekenen. Ik noem alleen enkelen, die voor mij belangrijk waren in mijn ontwikkeling als wetenschapper.

Ik dank het College van Bestuur van de Technische Universiteit Eindhoven voor mijn benoeming, en iedereen in de sectie Theoretische Informatica, in de vakgroep Informatica, en in de faculteit Wiskunde en Informatica die zich daarvoor ingezet heeft. Ik dank iedereen in de sectie, vakgroep en faculteit voor goede samenwerking.

Persoonlijk wil ik noemen Theo Standaert. In de laatste klas van de middelbare school heb je me laten zien dat wiskunde leuk kan zijn, en me gestimuleerd wiskunde te gaan studeren.

Vervolgens wil ik bedanken Henk Barendregt. Je hebt me geleerd wat wetenschap is, en wat wetenschappelijk onderzoek is.

Dank ook aan Jaco de Bakker. Jij hebt me laten zien hoe je op een zeer prettige manier leiding geeft aan een groep individuen, en die naar buiten toe laat optreden als een eenheid.

Dank aan Jan Willem Klop, Peter Weijland, Rob van Glabbeek, Frits Vaandrager en Scott Smolka voor de prettige samenwerking bij het schrijven van artikelen.

Als laatste wil ik noemen Jan Bergstra. Jij bent voor mij steeds van zeer grote waarde geweest, vanaf het moment dat ik je tegenkwam in 1983. Je bent een onuitputtelijke bron van ideeën over het vak. Ik heb ook erg veel van je geleerd op het gebied van management en het omgaan met mensen. Mijn dank.

Ik heb gezegd.

10. Literatuur

[Apt81] K.R. Apt, Ten years of Hoare's logic: a survey part I, ACM TOPLAS 3 (4), 1981, pp. 431-483.
[Bae90] J.C.M. Baeten (ed.), Applications of process algebra, Cambridge Tracts in Theoretical Computer Science 17, Cambridge University Press 1990.
[Bae92] J.C.M. Baeten, The total order assumption, in: Proceedings Workshop "What good is partial order", Sheffield 1992, te verschijnen.
[BaW90] J.C.M. Baeten & W.P. Weijland, Process algebra, Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press 1990.
[BeK82] J.A. Bergstra & J.W. Klop, Fixed point semantics in process algebras, rapport IW 206, Mathematisch Centrum, Amsterdam 1982.
[BeR89] J.A. Bergstra & G.R. Renardel de Lavalette, De plaats van formele specificaties in de softwaretechnologie, Informatie 31 (6), 1989, pp. 480-494.
[FeJ92] L.M.G. Feijs & H.B.M. Jonkers, Specification and design with COLD-K, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press 1992, te verschijnen.
[GrP90] J.F. Groote & A. Ponse, The syntax and semantics of μ-CRL, rapport CS-R9076, Centrum voor Wiskunde en Informatica, Amsterdam 1990.
[Hoa85] C.A.R. Hoare, Communicating sequential processes, Prentice Hall 1985.
[Jon90] C.B. Jones, Systematic software development using VDM, Prentice Hall, second edition 1990.
[Lin91] H. Lin, PAM: a process algebra manipulator, in: Proceedings CAV 91, Aalborg (K.G. Larsen & A. Skou, eds.), Lecture Notes in Computer Science 575, Springer Verlag 1992.
[MaP92] Z. Manna & A. Pnueli, The temporal logic of reactive and concurrent systems, Springer Verlag 1992.
[MaV93] S. Mauw & G.J. Veltink (eds.), Algebraic specification of communication protocols, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press 1993, te verschijnen.
[Mid90] C.A. Middelburg, Syntax and semantics of VVSL - a language for structured VDM specifications, Proefschrift, Universiteit van Amsterdam 1990.
[Mil80] R. Milner, A calculus of communicating systems, Lecture Notes in Computer Science 92, Springer Verlag 1980.
[Par86] D.L. Parnas, Software-aspecten van strategische defensiesystemen, Informatie 28 (3), 1986, pp. 175-186.
[Rei85] W. Reisig, Petri nets, EATCS Monograph on Theoretical Computer Science, Springer Verlag 1985.
[Rem87] M. Rem, Trace theory and systolic computations, in: Proceedings PARLE, Eindhoven (J.W. de Bakker, A.J. Nijman & P.C. Treleaven, eds.), Lecture Notes in Computer Science 258, Springer Verlag 1987, pp. 14-33.
[Roe91] W.P. de Roever, Foundations of computer science: leaving the ivory tower, Bulletin of the EATCS 44, 1991, pp. 455-492.
[Spi88] J.M. Spivey, Understanding Z, Cambridge Tracts in Theoretical Computer Science 3, Cambridge University Press 1988.
[Ver92] C. Verhoef, Linear unary operators in process algebra, Proefschrift, Universiteit van Amsterdam 1992.

Vormgeving en druk:
Reproduktie en Fotografie van de CTD
Technische Universiteit Eindhoven

Informatie:
Academische en Protocollaire Zaken
Telefoon (040-47)2250/4676

Jos Baeten werd in 1954 te Tilburg geboren. Na gymnasium-β te Tilburg studeerde hij in 1978 af in de wiskunde aan de RU Utrecht, afstudeerrichting logica en grondslagen van de wiskunde, op een scriptie over λ-calculus. Tot 1983 deed hij promotie-onderzoek aan de University of Minnesota (USA), afgerond met een proefschrift over definieerbaarheids-theorie. Na een jaar als medewerker wiskunde aan de TH Delft, trad hij in 1984 in dienst van de Afdeling Informatica van het CWI te Amsterdam. Hij werkte daar tot 1985, en weer van 1988 tot 1991, de laatste periode als groepsleider van de onderzoeksgroep Concurrency en Real Time Systemen van de Afdeling Programmatuur. Vanaf 1985 tot 1991 (met een korte onderbreking) was hij in dienst van de Vakgroep Programmatuur van de Universiteit van Amsterdam, waarvan 1 jaar als voorzitter van de vakgroep.
Twee boeken over procesalgebra (een theorie over parallelle communicerende processen) staan op zijn naam. Hij is actief in Europese samenwerkingsprojecten, o.a. als projectleider van de Basic Research Actions CONCUR en CONCUR2. Hij is mede-initiator van de reeks concurrency conferenties CONCUR.
Hij is wetenschappelijk directeur van de onderzoeksschool i.o. EURICS (Eindhoven Utrecht Research Institute for Computing Science).
Per 1 oktober 1991 is hij benoemd tot hoogleraar Theoretische Informatica aan de TUE.