Liber Amicorum

Jaco de Bakker

Centrum voor Wiskunde en Informatica

Jaco de Bakker

Amsterdam, 30 augustus 2002

Editors
Frank de Boer, Marlin van der Heijden, Paul Klint, Jan Rutten

Layout
Jan Schipper

Cover design
Tobias Baanders

Printing and binding
Ponsen & Looijen, Wageningen

Publication date
August 30, 2002

Inhoudsopgave

Voorwoord

Na een dienstverband van 38 jaar heeft Prof.dr. Jaco de Bakker op 1 juli 2002 afscheid genomen van het CWI. Dit is voor ons aanleiding tot een terugblik op zijn carrière en tot het uitdrukken van onze waardering voor zijn bijdragen aan de informatica in het algemeen en het CWI in het bijzonder.

Op 30 augustus 2002 organiseren wij daarom een symposium waarin een viertal sprekers (Jan Bergstra, Paul Klint, Joost Kok, John Tucker) hun persoonlijke visie op Jaco en zijn werk presenteren.

Daarnaast hebben wij persoonlijk getinte bijdragen gevraagd aan collega's in binnen- en buitenland. Het resultaat daarvan ligt nu voor U. Een bonte collectie van persoonlijke herinneringen, anecdotes en een enkele technische bijdrage.

Wij danken allen die een bijdrage geleverd hebben aan de totstandkoming van deze bundel: de auteurs van de bijdragen, Tobias Baanders voor de fraaie vormgeving, Jan Schipper en het CWI Reprocenter voor integratie en productie van dit boek en Mieke Bruné voor organisatorische ondersteuning.

Jaco, we nemen nu afscheid van je en hopen dat als je nog eens door dit boek bladert, je de waardering voor je werk en de vriendschap voor jou opnieuw zult beleven.

Amsterdam, augustus 2002

Frank de Boer, Marlin van der Heijden, Paul Klint, Jan Rutten

Jaco de Bakker: Pionier van de Nederlandse Informatica

1 Inleiding

Prof.dr. Jaco de Bakker kan zonder enige overdrijving één van de pioniers van de Nederlandse informatica worden genoemd. In een tijd dat computers nog een zeldzaamheid waren, en het vakgebied van de informatica nog niet bestond, is De Bakker als één van de allereerste grondleggers begonnen de informatica gestalte te geven en van een gedegen theoretisch fundament te voorzien.

Na in 1964 als wiskundige te zijn afgestudeerd aan de Vrije Universiteit te Amsterdam, trad De Bakker als wetenschappelijk medewerker in dienst van het toenmalige Mathematisch Centrum. Dit onderzoeksinstituut werd kort na de tweede wereldoorlog opgericht met als doelstelling 'de systematische beoefening van de zuivere en toegepaste wiskunde in Nederland te bevorderen'. In 1964 bestond informatica als zelfstandige discipline nog niet, en De Bakker begon zijn wetenschappelijk onderzoek dan ook bij de zogeheten Rekenafdeling. In deze afdeling werden, naast fundamenteel onderzoek, ook rekenopdrachten voor het bedrijfsleven uitgevoerd. De Bakker werkte onder leiding van Prof.dr.ir. A. van Wijngaarden, onder wiens supervisie hij in 1967 de graad van doctor behaalde aan de Universiteit van Amsterdam. Het onderwerp van zijn proefschrift handelde over formele definities van programmeertalen (i.h.b. de taal ALGOL 60, een van de eerste geavanceerde programmeertalen uit de geschiedenis van de informatica), en het specialisme van De Bakker is ook daarna de mathematische semantiek (betekenisleer) van programmeertalen en het redeneren over programmacorrectheid gebleven.

Aan het Mathematisch Centrum maakte De Bakker na zijn aanstelling snel carrière. In 1970 werd hij wetenschappelijk hoofdmedewerker, in 1971 souschef van de Rekenafdeling, en in 1973 werd bij vervolgens chef van diezelfde afdeling, die toen de Afdeling Informatica is gaan heten. In 1985 was de Afdeling Informatica onder zijn leiding zover uitgegroeid, dat een opsplitsing in drie afzonderlijke afdelingen noodzakelijk werd. Vanaf dat moment tot aan 1 januari 2002 is De Bakker afdelingshoofd gebleven van één van deze drie afdelingen, namelijk de Afdeling Programmatuur (vanaf 1997 het Cluster Software Engineering geheten). De voor een groot gedeelte onder de verantwoordelijkheid en leiding van De Bakker tot stand gekomen bloei van de discipline van de informatica binnen het Mathematisch Centrum, heeft in 1984 tot een naamsverandering van het gehele instituut geleid: Centrum voor Wiskunde en Informatica (CWI). Tevens werd de doelstelling verruimd tot: 'de systematische beoefening en toepassing van de wiskunde en informatica'. De Bakker heeft sinds 1973 op het hoogste niveau mede vorm gegeven aan de dagelijkse organisatie en het bestuur van het CWI, ondermeer door zijn lidmaatschap van het Overleg Beleidszaken en later het Management Team. In 1973 werd De Bakker hoogleraar aan de Vrije Universiteit, wat hij tot aan zijn emeritaat in 2001 is gebleven. Hij heeft de grondbeginselen van de informatica gedoceerd aan vele studenten van de Vrije Universiteit. Het belang en de betekenis van de wetenschappelijke, educatieve, en bestuurlijke activiteiten en verdiensten van De Bakker, zijn zowel in Nederland als daarbuiten niet onopgemerkt gebleven, en hebben geleid tot de volgende eervolle benoemingen: lid van de Koninklijke Nederlandse Akademie van Wetenschappen (1989); lid van de Academia Europaea (1990); Fellow van het Centrum voor Wiskunde en Informatica (2002).

2 Wetenschappelijk werk

De Bakker heeft meer dan honderdvijftig wetenschappelijke artikelen en boeken geschreven, veelal gepubliceerd in de meest vooraanstaande wetenschappelijke tijdschriften op het gebied van de theoretische informatica. Dit heeft hem tot een onderzoeker van wereldfaam gemaakt, met alle daaruit voortvloeiende activiteiten van dien: gastlezingen aan buitenlandse universiteiten en instituten, en op vooraanstaande congressen; redacteurschappen van internationale wetenschappelijke tijdschriften en boekenseries; en het organiseren van vele wetenschappelijke bijeenkomsten, congressen, en scholen. Met een aantal publicaties heeft De Bakker school gemaakt. Dit geldt in het bijzonder voor zijn leerboek 'Mathematical Theory of Program Correctness', in 1980 verschenen bij Prentice Hall International, dat voor enkele generaties studenten en onderzoekers de bijbel is geweest voor het redeneren over de correctheid van computerprogramma's. Een ander voorbeeld is het artikel van De Bakker en Prof.dr. J. Zucker 'Processes and the denotational semantics of concurrency', uit 1982, dat voor vele onderzoekers, waaronder een aantal promovendi van De Bakker, uitgangspunt voor verdere vraagstellingen is geweest. Een derde voorbeeld betreft een serie van conferentieverslagen van zes door De Bakker (mede)georganiseerde scholen, symposia en workshops, de zogeheten REX Proceedings, die tot op de dag van vandaag zeer invloedrijk zijn gebleven. (Over de REX serie volgt later in dit stuk nog wat aanvullende informatie.)

3 Wetenschappelijke bijeenkomsten

Het organiseren van congressen is voor de dagelijkse praktijk van de wetenschapsbeoefening van grote betekenis. Naast de feitelijke kennisoverdracht in de vorm van lezingen en gastcolleges, bieden congressen de wetenschapper de kans om collega's te ontmoeten en te spreken, en op die manier een netwerk op te bouwen. Lidmaatschap van het programmacomité van een conferentie, dat tot taak heeft een selectie uit de ingezonden wetenschappelijke bijdragen te maken, is voorbehouden aan de ervaren en gerespecteerde wetenschapper. Het feit dat De Bakker van het programmacomité van zoveel belangrijke internationale conferenties deel heeft uitgemaakt, kan dan ook zonder meer worden gezien als een bevestiging zijn wetenschappelijke reputatie. Van de talrijke nationale en internationale wetenschappelijke bijeenkomsten, congressen, scholen, en symposia, waaraan De Bakker, vaak als organisator, maar ook als voorzitter of lid van een programmacomité, heeft meegewerkt, willen we er hier enkele met name noemen.

De conferentie ICALP (International Colloquium on Automata, Languages and Programming) is tot op de dag van vandaag Europa's meest prestigieuze conferentie op het gebied van de theoretische informatica. De Bakker is tien maal lid en een keer Voorzitter van het programmacomité van deze conferentie is geweest. Bovendien was De Bakker een van de oprichters van de EATCS, de 'European Association for Theoretical Computer Science', waaruit deze conferentie is voortgekomen. Sinds de oprichting in 1972 is hij tot 1982 Vice-president en tot 1988 bestuurslid van de EATCS gebleven.

De Bakker heeft een belangrijke rol gespeeld bij de wetenschappelijke vorming van verscheidene generaties studenten en onderzoekers op het gebied van de theoretische informatica. Dit blijkt niet alleen uit het grote aantal promovendi (zie hieronder), maar ook uit zijn betrokkenheid bij de organisatie van de volgende internationale, invloedrijke cursussen. De Bakker was directeur van de 'Advanced Course on Foundations of Computer Science' (Amsterdam) in 1974, 1976, en in 1978 en 1982 samen met Prof.dr. J. van Leeuwen. In 1984 heeft De Bakker, samen met Prof.dr. W.P de Roever en Prof.dr. G. Rozenberg, het LPC (Landelijk Project Concurrency) opgericht, dat tot 1989 heeft gelopen. De activiteiten van het LPC bestonden, naast het begeleiden van een aantal promovendi, uit het houden van een serie van lezingen door experts uit binnen- en buitenland over de grondslagen van het in die jaren in opkomst zijnde thema van parallelle en gedistribueerde programmatuur. Het LPC kreeg in 1988 een vervolg in de vorm van REX (Research and Education in Concurrent Systems), wederom door De Bakker en voornoemde collega's opgericht. De impact van dit project was zo mogelijk nog groter, mede omdat het zich nadrukkelijk ook internationaal manifesteerde. Er werden in het totaal zes internationale symposia en workshops georganiseerd.

De verslagen van ieder van deze bijeenkomsten zijn gepubliceerd in Springer's LNCS (Lecture Notes in Computer Science) serie. De wereldwijde invloed van deze zes REX bijeenkomsten kan niet snel worden overschat, en is tot op de dag van vandaag voelbaar: Wereldwijd hebben vele onderzoekers in de theoretische informatica een deel van hun basisvorming te danken aan één of meerdere van deze zes bijeenkomsten. Dit laatste wordt onderstreept door de hoge ranking (46), die de serie van REX Proceedings inneemt op de lijst van zogeheten 'Estimated impact of publication venues in Computer Science', welke een onderdeel is van de voor de informatica belangrijke citatie-index van het NEC Research Institute (zie http://citeseer.nj.nec.com/impact.html).

Naast het houden van en luisteren naar lezingen op congressen, zijn vaktijdschriften en boekenseries een cruciaal medium voor de disseminatie van wetenschappelijke kennis. Ingezonden bijdragen worden door een groep van redacteuren beoordeeld en van kritiek voorzien, voordat ze als artikel ook daadwerkelijk kunnen worden gepubliceerd. Meer nog dan de betrokkenheid bij conferenties geldt dat het redacteurschap van een vaktijdschrift een kwestie is van wetenschappelijke prestige. De redacteur moet wetenschappelijk excellent zijn, het vakgebied kunnen overzien, en beschikken over een grote wetenschappelijke ervaring. Dat De Bakker al deze eigenschappen in zich verenigt, blijkt uit het feit dat hij in de loop der jaren als redacteur is gevraagd voor vele vooraanstaande internationale vaktijdschriften en boekenseries.

Gedurende zijn carrière heeft De Bakker talloze cursussen, gastcolleges, en lezingen gehouden op universiteiten, instituten, en conferenties overal ter wereld. We vermelden hier enkele voorbeelden. Hij is gasthoogleraar geweest bij de universiteiten van Carnegie-Mellon (USA) en Bar-Ilan (Israel), en ook bij de Pontificia Universidade Catolica in Rio de Janeiro. Verder heeft hij als gastdocent gedoceerd bij onder andere het Stefan Banach Centre in Warschau, en aan de UNESCO Summer School on Foundations of Computer Science in Turku, Finland. De Bakker maakte ook deel uit van het Seminarium van het International Institute for Software Technology, van de United Nations University, dat in 1991 een serie van lezingen in Azië organiseerde, onder andere in Peking.

4 Internationale samenwerking

Naast zijn betrokkenheid bij internationale vaktijdschriften en wetenschappelijke bijeenkomsten, heeft De Bakker altijd veel waarde gehecht aan internationale wetenschappelijke samenwerking. Hij heeft dan ook in een groot aantal internationale projecten samengewerkt met buitenlandse specialisten en bedrijven. Hieronder noemen we enkele van de belangrijkste daarvan.

Van 1984 tot 1989 was De Bakker lid van de Coördinatiecommissie van ESPRIT project 415: Parallel Architectures and Languages for Advanced Information Processing. Dit met gelden van de Europese Gemeenschap gefinancierde project was het grootste op het terrein van de AIP, begroot op in het totaal 250 manjaren. Tot de partners van dit project behoorden, naast een groot aantal Europese onderzoeksinstellingen en universiteiten, tevens verschillende grote multinationals, waaronder Philips, Nixdorff/Siemens, General Electric, en AEG. Met name de samenwerking met Philips Research in Eindhoven was bijzonder intensief, en leidde onder andere tot een dubbelpromotie (P. America en J. Rutten) op basis van een gemeenschappelijk proefschrift 'A Parallel Object-Oriented Language: design and semantic foundations', over de door Philips ontwikkelde programmeertaal POOL. In diezelfde periode was De Bakker ook Voorzitter van de 'Working group on semantics and proof techniques', een forum waarin alle leden van ESPRIT 415 vertegenwoordigd waren, en dat voor de samenwerking binnen het project van groot belang was.

De Bakker is initiator en projectleider geweest van ESPRIT Basic Research Action 'Integration of functional, logic and object-oriented programming', dat van 1989 tot 1993 heeft gelopen. Hierin werd samengewerkt door een aantal Europese universiteiten, kennisinstellingen, en bedrijven, aan moderne 'high-level' programmeertalen. Tot de partners behoorden, onder anderen, de Ecole Normale Supérieure Paris, de universiteiten van Lissabon en Pisa, Imperial College London, en Philips Research Eindhoven.

5 Bestuurlijke en educatieve activiteiten

Naast zijn wetenschappelijke onderzoekswerk is De Bakker binnen de Nederlandse informatica altijd bijzonder actief geweest op bestuurlijk en educatief terrein. Allerbelangrijkst zijn daarbij geweest zijn bestuursfuncties op het CWI en zijn rol van promotor op de Vrije Universiteit. Deze bestuurlijke en educatieve activiteiten hebben een belangrijke bijdrage geleverd aan de totstandkoming van een groot gedeelte van het huidige kader van Nederlandse hoogleraren informatica. Hieronder volgt een opsomming van Nederlandse hoogleraren in de informatica, die ten minste gedurende enkele jaren onder de directe leiding van De Bakker als groeps- of projectleider op het CWI hebben gewerkt, en die op die manier een wezenlijk gedeelte van hun wetenschappelijke vorming aan hem te danken hebben:

Naam	Jaar van benoeming	huidige affiliatie
Prof.dr. P. Klint	1985	CWI, UvA
Prof.dr. J.W. Klop	1986	CWI, VUA
Prof.dr. L.G.L.T. Meertens	1987	UU, Kestrel Institute (USA)
Prof.dr. J.C. van Vliet	1987	VUA
Prof.dr. D.J.N. van Eijck	1990	CWI, UU
Prof.dr. K.R. Apt	1991	CWI, UvA
Prof.dr. J.C.M. Baeten	1991	TU/e
Prof.dr. F. Vaandrager	1995	KUN
Prof.dr. J.F. Groote	1998	TU/e
Prof.dr. W.J. Fokkink	2001	CWI, VUA
Prof.dr. J.A. La Poutré	2001	CWI, TU/e
Prof.dr. J.J.M.M. Rutten	2001	CWI, VUA

Verder is een aantal van de oud-medewerkers van De Bakker momenteel hoogleraar aan een universiteit in het buitenland: Prof.dr. P.J.F. Lucas (Univ. van Aberdeen), Prof.dr. J. Tucker (Univ. van Swansea), Prof.dr. R. Back (Univ. van Turku), en Prof.dr. J. Zucker (Univ. van Hamilton).

De Bakker heeft, naast zijn werk op het CWI, sinds 1973 ook een deeltijd aanstelling als hoogleraar op de Vrije Universiteit gehad. In die hoedanigheid heeft hij een groot aantal wetenschappers opgeleid, die onder zijn, vaak dagelijkse en altijd intensieve, begeleiding een proefschrift hebben geschreven en bij hem zijn gepromoveerd. Hieronder volgt een overzicht van alle promovendi van De Bakker, onder vermelding van het jaar van promotie en de huidige affiliatie:

Naam	Jaar van promotie	huidige affiliatie
Prof.dr. W.P. de Roever	1973	Univ. van Kiel
Prof.dr. P.M.B. Vitányi	1978	CWI, UvA
Prof.dr. A. Nijholt	1980	UT
Prof.dr. J.J. Meyer	1985	UU
Prof.dr. A. de Bruin	1986	Erasmus Universiteit
Prof.dr. J.N. Kok	1989	UL
Dr. P. America	1989	Philips Research Eindhoven
Prof.dr. J.J.M.M. Rutten	1989	CWI, VUA
Dr. E.P. de Vink	1990	TU/e
Dr. F.S. de Boer	1991	UU
Dr. A. Eliëns	1991	VUA
Dr. Eiichi Horita	1993	NTT, Japan
Prof.dr. F. van Breugel	1994	Univ. van York, Toronto
Dr. D. Turi	1996	Univ. van Edinburgh
Dr. M.M. Bonsangue	1996	UL
Dr. J. den Hartog	2002	TU/e

(Daarnaast is De Bakker betrokken geweest, als externe referent of lid van de promotie-commissie, bij ruim dertig promoties in binnen- en buitenland.) Wat opvalt aan dit lijstje, naast het grote

aantal: zestien in het totaal, is het feit dat de helft van alle promovendi van De Bakker inmiddels zelf hoogleraar is.

6 Epiloog

Het bovenstaande maakt duidelijk dat De Bakker aan de informatica in het algemeen, en aan de Nederlande informatica in het bijzonder, een aantal bijdragen van uitzonderlijk belang heeft geleverd:

- Het totstandbrengen van een bijzonder omvangrijk en invloedrijk wetenschappelijk oeuvre, van fundamenteel belang voor de ontwikkeling van de informatica als een zelfstandige wetenschappelijke discipline.
- Het vormgeven aan en het tot bloei brengen van de informatica in Nederland door middel van zijn werk aan het Centrum voor Wiskunde en Informatica.
- Het opleiden en vormen van verschillende generaties informatici, waaronder een groot aantal huidige hoogleraren informatica.
- Het totstandbrengen, bevorderen, en ondersteunen van internationale samenwerking op het gebied van wetenschappelijk onderzoek en educatie binnen de informatica.

Jaco de Bakker kan, nu hij na 38 dienstjaren afscheid neemt van het CWI (waaraan hij overigens als adviseur nog enige tijd verbonden blijft), terugkijken op een vruchtbare en welbestede wetenschappelijke carrière.

Amsterdam, juli 2002

Frank de Boer, Marlin van der Heijden, Paul Klint, Jan Rutten

Jaco played an important role in my scientific career: twice he offered me a job and twice I was happy to accept. The first time it was in 1975 and the second time in 1987. When I look back at Jaco's career what strikes me most is that he introduced computer science, or offered a job, to a whole array of logicians. I am not sure whether I was the first one, but I recall several ones who followed me: Jeff Zucker, John Tucker, Jan Bergstra, Frank de Boer, Jan Willem Klop, Bart Jacobs. We all settled down in computer science and we all profitted from Jaco's unshaken confidence in logic and logicians. And I think it is fair to say that we all lived up in one way or another to his expectations. In some sense Jaco created for all of us a unique opportunity to enter computer science through the main door and provided us with interesting research problems that helped us to carry out the transition from logic to computer science in a relatively painless way. In the meantime I have visited enough countries to appreciate how unique were these generous possibilities created by Jaco.

To most of us Jaco is perceived as a bastion of reliability and responsibility and it is unimaginable that he could miss a train, let alone a plane. Yet, I am happy to report that it did happen and that I played a role in this event. In 1976, when I got married for the first time, my first wife and me gave a party in our *souterrain* at Binnenkant 42. As a result of this party Jaco missed next day a plane to a conference in Poland.

Jaco worked for almost 40 years at CWI and it is difficult to get used to the fact that he now retires. In fact, a couple of months ago, when I learned that he became a CWI fellow I thought that he will rather use this opportunity to work undisturbed on a new book. But who am I to say this? Who knows what I shall be up to in ten years?

Jaco, please enjoy your free time the way it suits you best. Knowing you I am sure you will dive into some good fiction book or a book on painting. My personal recommendation: The Short Stories of Isaac Babel. Hardly known in the West, and killed on the orders of Stalin, Babel created amazing short stories and theater plays that are truly unique and fascinating portraits of people. They surpass in my opinion all what the Western culture has produced in this genre, including the short stories of Bernard Malamud and theater plays of Tennessee Williams.

Krzysztof Apt
May 7, 2002

Jaco de Bakker:
Personal Impressions of a Professional Colleague

Farhad Arbab

June 2002

There is a story about Pablo Picasso which, as the German saying goes, if not true at least it is well-constructed. Having acquired some fame already, Picasso was sitting alone in a quiet café one day when a gaudy parvenu spotted him. The man approached Picasso and made a scene showing off that he had recognized the great artist. Picasso acknowledged him but tried to deflect his attempts to impose himself upon his solitude. Aching for a trophy to show off his chance encounter with the great artist, the man asked Picasso to do a portrait of him. Picasso refused. The man would not relent; if not a painting, he wanted at least a portrait drawing. To spare himself further agony of his companionship, Picasso finally agreed to do his portrait sketch right there and then. The man was jubilant. Picasso took his pencil and quickly drew a few lines on paper and handed it over to the man. The man was silent and astonished for a moment. Then he started to loudly flaunt his appreciation of the artist's insight for seeing the essence of his being in his face, and his masterful expression of that insight in just a few lines. Picasso had barely managed to salvage his composure through the unwelcome avalanche of his annoying admiration when the parvenu offered to pay him for the drawing. Picasso refused to accept, but the man was relentless. Picasso proposed the man just pays for his drink, but the man insisted that he wanted to actually pay for the work and wanted the artist to name his price. Finally, Picasso said "OK, it's 4000 dollars." The parvenu was shocked; in a surprised tone and a softened volume he said "What? Four thousand dollars for two minutes of your time and four lines!" Picasso grabbed the drawing, tore it up and replied "No, it was 4000 dollars for forty years plus two minutes and four lines."

Experience, skill, and insight are precious commodities whose impacts on a piece of work cannot be rightfully measured by the duration in which they actually touch that work. Reflection distils toil and yields the unguent of experience. Skill is mastered by diligence and refined by dedication to detail over time. Insight is honed by immersion and sharpened by uncompromising desire to discover the intricacies of structures and relationships that underlie all that is sublime. Experience, skill, and insight are the hallmarks of the long and fruitful professional life of Jaco de Bakker as a researcher, teacher, and leader. The breadth of his impact is the extent to which his work has influenced theoretical computer science and the fields of formal semantics and semantics of concurrency. The depth of his impact can best be seen in the professional lives of a long list of colleagues who collaborated and apprenticed with him, or observed him up close perform his craft and lead.

I knew of Jaco de Bakker before I came to CWI in 1990, but I did not know him. Our paths physically crossed a few times in and around the building, and I'm sure in his capacity as a department head he also knew something about me. Yet, we did not really know each other until our professional paths crossed in 1997. The reorganization of CWI landed me in his new Cluster and, more significantly, our professional interests put us both in the same Theme in that Cluster to work on Coordination Models and Languages.

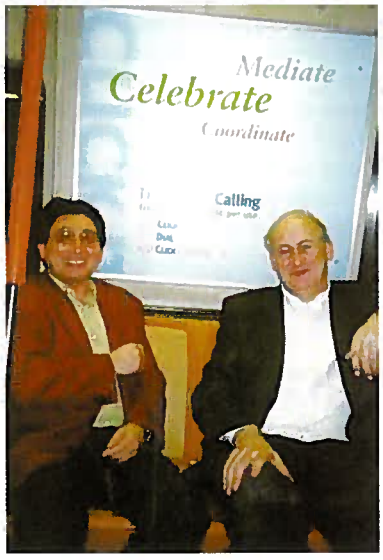
As the focus of Jaco's famous regular Tuesday afternoon ACG meetings slightly shifted and the C of Coordination replaced the former C of Concurrency in its name, my previous work on IWIM and Manifold was presented, discussed, and analyzed in these meetings. These discussions became the feed for a smaller, more focused spin-off group where for the next two years we met almost regularly each week and worked on the semantics of Coordination languages, and specifically, elaborated on the formal semantics of Manifold.

Everything flows. Interests change, topics become fashionable and then fall out of style, hard problems draw attention and effort until they are resolved and the attention and effort then shift toward new emerging topics and new arising problems. Ultimately, the ability to carry and apply insight, skill, and experience to new topics is perhaps even more important than the mastery of any specific topic. I would like to think that I was already mature enough to recognize and appreciate experience, skill, and insight, but these qualities were too prominent in Jaco's scientific work and his professional leadership was too observable throughout our collaborations for anyone to miss. Our work and the ACG meetings introduced me to a new group of bright colleagues, all apprenticed in Jaco's school, whose collaborations I cherish and who have influenced if not shaped my own work in the past five years. I observed good examples of how those valuable constants cultivated through flows of the past apply to the currents of the present and form the flows of the future. Reo. I look forward to the culmination of my current joint activities with this group in the coming years, but I also know that I will enjoy at least as much the actual journey of collaborations and interactions that will get us there. I hope we can at least occasionally still benefit along our way from Jaco's taking a few steps with us.

Jaco, I know I grew professionally in the past few years because we had the opportunity for me not to be an Idealized Worker and for you not to be an Idealized Manager. Thank you for your forty years plus two minutes and four lines.

White hair, the Universe did not bestow upon me for free,
This novelty, I have paid for with the cash of my youth.

Rahi Moayerri
Contemporary Persian lyricist and poet



A Menagerie of Constants in Process Algebra

J.C.M. Baeten

Department of Computer Science, Technische Universiteit Eindhoven,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
josb@win.tue.nl

Abstract. Jaco de Bakker has always been a constant factor in computer science at the CWI. Among constants, there is a great variety. This contribution presents an overview, and tries to find Jaco's place in this constellation.

1 Introduction

We give an inventory of constants that have been introduced in process algebra during the past decades.

- 0 the zero process [3]. This process is characterized by being a true zero, i.e. it satisfies $x + 0 = x$ and $0 \cdot x = x \cdot 0 = 0$. With these laws, it does not constitute a process that can be characterized operationally, in terms of states and transitions. It is useful, though, in calculations, as shown in [3]. We say the process 0 denotes an *inconsistent* or *virtual* state.
- $\emptyset$ the empty graph without entries or exits [9]. The paper cited investigates process states with named entry and exit points. By this means, feedback loops can be specified. The paper also gives the addition of the constant $\emptyset$ to standard ACP, and asks the question how the different constants introduced in ACP over the years relate to each other. This question is answered in the present paper. Being the empty graph, it does not have a state. Abusing words, we will sometimes say it has an inconsistent state.
- *nil* necessary termination [1]. This process is characterized by only being able to terminate (no other option available). It is the only constant of basic CCS [18, 19], where (different from ACP or CSP) there is no distinction between successful termination and unsuccessful termination (deadlock). We use the operational characterization of [1].
- δ inaction [14]. This is the basic constant of standard ACP. Operationally, it is characterized by having no transitions or predicates whatsoever. However, interpreted in a theory with time, it is better named *livelock*, that allows time to pass, but no execution of an action, see [2].
- τ silent action [18, 15]. An action is executed (at an unspecified moment of time), and this can cause a change of state. The action itself is not observable, but the change of state may be.
- Δ divergence [1]. It is possible to execute an unbounded number of silent actions, but it is also always possible to exit the loop and terminate (to start a subsequent process).

- a atomic action [18, 16]. The basic building block of a process algebra. In CCS, this takes the form of a prefix operator, and the process that only executes action a is denoted $a.nil$. In ACP, the actions are constants, and denote the execution of the action followed by immediate termination.
- ϵ termination option [22]. The counterpart of the inaction constant δ. In standard ACP, ϵ stands for successful termination, and acts as a unit element for sequential composition. Interpreted in a theory with time, it allows time to pass, see [2].
- χ chaos [17]. This is the CSP constant denoting the underspecified process, that can show arbitrary behaviour.
- $\perp$ nonexistence: nonexistent state of a process [7]. The paper cited considered a process algebra where to each state is associated a set of propositions that hold in that state. Then, $\perp$ denotes a state where falsum holds. As such, this is a state that cannot be entered from a normal, consistent state.

Next, we consider a number of constants that were introduced in process algebras with timing. In order not to introduce too much overhead, we limit ourselves to discrete and relative timing (so time is divided into slices and time is measured with respect to the previous action). Using references [10–12], this can be extended to theories with dense or absolute timing.

- $\dot{\delta}$ immediate deadlock or having terminated unsuccessfully [4, 5, 10]. This process is the unit element of alternative composition in all theories with timing. Operationally, it does not denote a consistent state of a process, but it can be entered by the execution of an action (for instance in case of inconsistent timing).
- $\dot{\epsilon}$ immediate termination or having terminated successfully [12]. The unit element of sequential composition in all theories with timing.
- $\underline{a}, \underline{\delta}$ action and deadlock in the current time slice [6, 10]. The process $\underline{a}$ does not allow passing to the next time slice, and will execute a in the current time slice, followed by immediate termination. The process $\underline{\delta}$ does not allow passing to the next time slice, and is characterized operationally by having no transitions or predicates.
- $\underline{\sigma}$ passage to the next time slice, followed by immediate termination. See [12, 2].
- $\underline{\epsilon}$ termination in the current time slice [2]. The counterpart of $\underline{\delta}$. Does not allow passing to the next time slice.

2 Algebra

In this section, we consider the interrelations of the different constants. We start by giving tables for the basic operators, viz. alternative, sequential and parallel composition. In these tables, we consider the constants $0, \dot{\delta}, \dot{\epsilon}, nil, \underline{\delta}, \underline{a}, \underline{\sigma}, \underline{\epsilon}, \delta, a, \epsilon, \Delta, \chi, \perp$. We do not show the constant τ, as the results are the same as for the constant a. We will say something about the special laws for τ further on. Further, we do not show the constant $\emptyset$, as the laws in standard ACP as shown

in [9] are the same as the laws for δ in [10]. Thus, we will identify these two constants in the sequel.

We first show the table for alternative composition. As the operator is commutative, only half of the table needs to be filled.

$+$	0	$\dot\delta$	$\dot\epsilon$	nil	$\underline{\underline{\delta}}$	$\underline{\underline{a}}$	$\underline{\underline{\sigma}}$	$\underline{\underline{\epsilon}}$	δ	a	ϵ	Δ	χ	$\bot$
0	0	$\dot\delta$	$\dot\epsilon$	nil	$\underline{\underline{\delta}}$	$\underline{\underline{a}}$	$\underline{\underline{\sigma}}$	$\underline{\underline{\epsilon}}$	δ	a	ϵ	Δ	χ	$\bot$
$\dot\delta$		$\dot\delta$	$\dot\epsilon$	$\underline{\underline{\delta}}$	$\underline{\underline{\delta}}$	$\underline{\underline{a}}$	$\underline{\underline{\sigma}}$	$\underline{\underline{\epsilon}}$	δ	a	ϵ	Δ	χ	$\bot$
$\dot\epsilon$			$\dot\epsilon$	$\underline{\underline{\epsilon}}$	$\underline{\underline{\epsilon}}$	$\underline{\underline{a}}+\underline{\underline{\epsilon}}$	$\underline{\underline{\sigma}}+\underline{\underline{\epsilon}}$	$\underline{\underline{\epsilon}}$	$\delta+\underline{\underline{\epsilon}}$	$a+\underline{\underline{\epsilon}}$	ϵ	Δ	χ	$\bot$
nil				nil	$\underline{\underline{\delta}}$	$\underline{\underline{a}}$	$\underline{\underline{\sigma}}$	$\underline{\underline{\epsilon}}$	δ	a	ϵ	Δ	χ	$\bot$
$\underline{\underline{\delta}}$					$\underline{\underline{\delta}}$	$\underline{\underline{a}}$	$\underline{\underline{\sigma}}$	$\underline{\underline{\epsilon}}$	δ	a	ϵ	Δ	χ	$\bot$
$\underline{\underline{a}}$						$\underline{\underline{a}}$	$\underline{\underline{a}}+\underline{\underline{\sigma}}$	$\underline{\underline{a}}+\underline{\underline{\epsilon}}$	$\underline{\underline{a}}+\delta$	a	$\underline{\underline{a}}+\epsilon$	$\underline{\underline{a}}+\Delta$	χ	$\bot$
$\underline{\underline{\sigma}}$							$\underline{\underline{\sigma}}$	$\underline{\underline{\epsilon}}+\underline{\underline{\sigma}}$	$\underline{\underline{\sigma}}+\delta$	$\underline{\underline{\sigma}}+a$	ϵ	Δ	χ	$\bot$
$\underline{\underline{\epsilon}}$								$\underline{\underline{\epsilon}}$	$\underline{\underline{\epsilon}}+\delta$	$\underline{\underline{\epsilon}}+a$	ϵ	Δ	χ	$\bot$
δ									δ	a	ϵ	Δ	χ	$\bot$
a										a	$a+\epsilon$	$a+\Delta$	χ	$\bot$
ϵ											ϵ	Δ	χ	$\bot$
Δ												Δ	χ	$\bot$
χ													χ	$\bot$
$\bot$														$\bot$

We give some comments to this table. In line 1 (the line of 0), we use that $x+0=x$ for all x. This needs to hold for a true zero. In line 2, we use that $x+\dot\delta=x$ in all theories with timing. In line 3, a termination option is added. This makes no difference for constants that already allow termination ($\underline{\underline{\epsilon}}, \epsilon, \Delta$). Further, in theories with discrete timing, addition of $\dot\epsilon$ to a consistent process is the same as addition of $\underline{\underline{\epsilon}}$, as passage of time *within* the current time slice will be allowed. Throughout, $\bot$ absorbes everything, and χ absorbes everything (except $\bot$). Lines 1-3 consider inconsistent processes. In line 4, we see that nil is absorbed by every consistent process, $x+nil=x$ holds in basic CCS. In the absence of nil, $\underline{\underline{\delta}}$ is the neutral element of the consistent processes in discrete relative time ACP. This is shown in line 5. Line 6 is straightforward, if we notice that a allows execution in any time slice, including the current one. For line 7, notice that passage to the next time slice, and termination there, is also allowed by ϵ and Δ. Line 8: ϵ allows termination in any time slice, including the current one. Line 9: δ is the neutral element in time free ACP. The rest is straightforward.

Next, we consider sequential composition. This operator is not commutative, so we need to fill the entire table. Entry (i,j) below shows the result of $i\cdot j$, sometimes written as ij.

The top line and left-most column are again explained by the fact that 0 is a true zero. The lines for $\dot\delta, \underline{\underline{\delta}}, \delta$ are explained by the fact that these constants do not allow termination. The line and column for $\dot\epsilon$ reflect that this constant is the neutral element for sequential composition. The constant nil is the neutral element for sequential composition for all consistent processes, and changes immediate (inconsistent) termination into termination by nil. The line for $\underline{\underline{a}}$ is

clear, if we realise that sequential composition with $\bot$ disallows execution of the a, as the state $\bot$ cannot be entered. Similarly, the line of $\underline{\sigma}$ is clear, if we realise that δ already allows passage of time. For $\underline{\epsilon}$, we remark that termination of a sequential composition is only possible if both arguments allow termination. Next, we remark that $\epsilon\underline{\sigma} = \underline{\sigma}\epsilon$, in both cases termination is possible starting in the next time slice. Finally, we remark that in theories of timing in [10, 11] we have $\underline{\sigma} \cdot \dot{\delta} = \underline{\delta}$, but this abstraction is not necessary, see the discussion in [5].

$\cdot$	0	$\dot{\delta}$	$\dot{\epsilon}$	nil	$\underline{\delta}$	$\underline{a}$	$\underline{\sigma}$	$\underline{\epsilon}$	δ	a	ϵ	Δ	χ	$\bot$
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\dot{\delta}$	0	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$
$\dot{\epsilon}$	0	$\dot{\delta}$	$\dot{\epsilon}$	nil	$\underline{\delta}$	$\underline{a}$	$\underline{\sigma}$	$\underline{\epsilon}$	δ	a	ϵ	Δ	χ	$\bot$
nil	0	$\underline{\delta}$	nil	nil	$\underline{\delta}$	$\underline{a}$	$\underline{\sigma}$	$\underline{\delta}$	δ	a	ϵ	Δ	χ	$\bot$
$\underline{\delta}$	0	$\underline{\delta}$	$\underline{\delta}$	$\underline{\delta}$	$\underline{\delta}$	$\underline{\delta}$	$\underline{\delta}$	$\underline{\delta}$	$\underline{\delta}$	$\underline{\delta}$	$\underline{\delta}$	$\underline{\delta}$	$\underline{\delta}$	$\underline{\delta}$
$\underline{a}$	0	$\underline{a}\dot{\delta}$	$\underline{a}$	$\underline{a}nil$	$\underline{a\delta}$	$\underline{aa}$	$\underline{a\sigma}$	$\underline{a\epsilon}$	$\underline{a\delta}$	$\underline{a}a$	$\underline{a}\epsilon$	$\underline{a}\Delta$	$\underline{a}\chi$	$\underline{\delta}$
$\underline{\sigma}$	0	$\underline{\sigma}\dot{\delta}$	$\underline{\sigma}$	$\underline{\sigma}nil$	$\underline{\sigma\delta}$	$\underline{\sigma a}$	$\underline{\sigma\sigma}$	$\underline{\sigma\epsilon}$	δ	$\underline{\sigma}a$	$\underline{\sigma}\epsilon$	$\underline{\sigma}\Delta$	$\underline{\sigma}\chi$	$\underline{\delta}$
$\underline{\epsilon}$	0	$\underline{\delta}$	$\underline{\epsilon}$	$\underline{\delta}$	$\underline{\delta}$	$\underline{a}$	$\underline{\sigma}$	$\underline{\epsilon}$	δ	a	ϵ	Δ	χ	$\bot$
δ	0	δ	δ	δ	δ	δ	δ	δ	δ	δ	δ	δ	δ	δ
a	0	$a\dot{\delta}$	a	$a\,nil$	$a\underline{\delta}$	$a\underline{a}$	$a\underline{\sigma}$	$a\underline{\epsilon}$	$a\delta$	aa	$a\epsilon$	$a\Delta$	$a\chi$	δ
ϵ	0	δ	ϵ	δ	δ	a	$\epsilon\underline{\sigma}$	ϵ	δ	a	ϵ	Δ	χ	$\bot$
Δ	0	$\Delta\underline{\delta}$	Δ	Δnil	$\Delta\underline{\delta}$	$\Delta\underline{a}$	$\Delta\underline{\sigma}$	Δ	$\Delta\delta$	Δa	$\Delta\epsilon$	Δ	$\Delta\chi$	$\Delta\underline{\delta}$
χ	χ	χ	χ	χ	χ	χ	χ	χ	χ	χ	χ	χ	χ	χ
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

Next, we consider parallel composition or merge. Actions can interleave or can synchronise (the latter possibility omitted in the following table), but time needs to progress in both components simultaneously. As merge is commutative, only half of the entries is necessary.

$\parallel$	0	$\dot{\delta}$	$\dot{\epsilon}$	nil	$\underline{\delta}$	$\underline{a}$	$\underline{\sigma}$	$\underline{\epsilon}$	δ	a	ϵ	Δ	χ	$\bot$
0	0	$\dot{\delta}$	0	0	0	0	0	0	0	0	0	0	0	0
$\dot{\delta}$		$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$	$\dot{\delta}$
$\dot{\epsilon}$			$\dot{\epsilon}$	nil	$\underline{\delta}$	$\underline{a}$	$\underline{\sigma}$	$\underline{\epsilon}$	δ	a	ϵ	Δ	χ	$\bot$
nil				nil	$\underline{\delta}$	$\underline{a}nil$	$\underline{\sigma}nil$	$\underline{\delta}$	δ	$a\,nil$	δ	Δ	χ	$\bot$
$\underline{\delta}$				$\underline{\delta}$	$\underline{a\delta}$	$\underline{\sigma\delta}$	$\underline{\delta}$	$\underline{\delta}$	$\underline{a\delta}$	$\underline{\delta}$	$\tau^*\underline{\delta}$		χ	$\bot$
$\underline{a}$					$\underline{aa}$	$\underline{a\sigma}$	$\underline{a\epsilon}$	$\underline{a\delta}$	$\underline{a}a + \underline{aa}$	$\underline{a}\epsilon$	$\underline{a}\Delta + \underline{\tau}(\underline{a} \parallel \Delta)$		χ	$\bot$
$\underline{\sigma}$						$\underline{\sigma}$	$\underline{\delta}$	δ	$\underline{a\sigma} + \underline{\sigma}a$	$\underline{\sigma}\epsilon$	Δ		χ	$\bot$
$\underline{\epsilon}$							$\underline{\epsilon}$	$\underline{\delta}$	$a\underline{\epsilon}$	$\underline{\epsilon}$	$\tau^*\underline{\epsilon}$		χ	$\bot$
δ								δ	$a\delta$	δ	$\Delta\delta$		χ	$\bot$
a									aa	$a\epsilon$	$a\Delta + \tau(a \parallel \Delta)$		χ	$\bot$
ϵ										ϵ	$\Delta\epsilon$		χ	$\bot$
Δ											Δ		χ	$\bot$
χ													χ	$\bot$
$\bot$														$\bot$

3 Operational semantics

In this section, we give an SOS specification in Plotkin style ([20]) for all constants considered in the previous section, and the basic operators.

We consider the following relations and predicates.

- $x \xrightarrow{a} x'$ execution of an action; upon execution of a, process x transforms into x'
- $x \overset{1}{\mapsto} x'$ passage to the next time slice, tick; by passing to the next time slice, process x transforms into x'
- $zero(x)$ predicate only holds for 0 process;
- $x\uparrow$ inconsistency, only holds for processes $0, \dot{\delta}, \dot{\epsilon}$;
- $\surd(x)$ necessary termination, only holds for process nil;
- $x\downarrow$ process x has a termination option;
- $ch(x)$ chaos predicate, only holds for process χ;
- $n\xi(x)$ nonexistence predicate, only holds for process $\bot$.

In the rules, we sometimes use negative premises. Thus, $x \overset{a}{\not\rightarrow}$ means that x cannot do an a-step, and $\neg zero(x)$ means that x does not satisfy the $zero$ predicate. Nevertheless, we see that all rules satisfy the *panth* format of [21], and it is not difficult to come up with a stratification, so that bisimulation is a congruence.

First of all, the axioms of the deduction system. See Table 1.

$$zero(0) \quad 0\uparrow \quad \dot{\delta}\uparrow \quad \dot{\epsilon}\uparrow \quad \surd(nil)$$

$$\dot{\epsilon}\downarrow \quad \underline{\underline{\epsilon}}\downarrow \quad \epsilon\downarrow \quad \Delta\downarrow$$

$$\underline{\underline{a}} \xrightarrow{a} \dot{\epsilon} \quad a \xrightarrow{a} \dot{\epsilon} \quad \Delta \xrightarrow{\tau} \Delta$$

$$\underline{\underline{\sigma}} \overset{1}{\mapsto} \dot{\epsilon} \quad \delta \overset{1}{\mapsto} \delta \quad \epsilon \overset{1}{\mapsto} \epsilon \quad a \overset{1}{\mapsto} a \quad \Delta \overset{1}{\mapsto} \Delta$$

$$ch(\chi) \quad n\xi(\bot)$$

Table 1. Axioms for deduction system (a an action)

Next, we consider the rules for alternative composition. See Table 2. Here, we need negative premises in a rule for time steps and a rule for the chaos predicate. We read a rule with multiple premises and multiple conclusions as follows: whenever all of the premises are satisfied, then all of the conclusions follow.

$$\frac{zero(x),\ zero(y)}{zero(x+y)} \qquad \frac{x\uparrow,\ y\uparrow}{x+y\uparrow} \qquad \frac{x\downarrow}{x+y\downarrow,\ y+x\downarrow} \qquad \frac{\surd(x),\ \surd(y)}{\surd(x+y)}$$

$$\frac{x \xrightarrow{a} x'}{x+y \xrightarrow{a} x',\ y+x \xrightarrow{a} x'} \qquad \frac{x \overset{1}{\mapsto} x',\ y \overset{1}{\mapsto} y'}{x+y \overset{1}{\mapsto} x'+y'} \qquad \frac{x \overset{1}{\mapsto} x',\ y \overset{1}{\not\mapsto}}{x+y \overset{1}{\mapsto} x',\ y+x \overset{1}{\mapsto} x'}$$

$$\frac{ch(x),\ \neg n\xi(y)}{ch(x+y),\ ch(y+x)} \qquad \frac{n\xi(x)}{n\xi(x+y),\ n\xi(y+x)}$$

Table 2. Rules for alternative composition (a an action)

Next, Table 3 considers the rules for sequential composition. There are several rules with negative premises.

Finally, Table 4 considers the rules for parallel composition.

$$\frac{zero(x)}{zero(x\cdot y),\ zero(y\cdot x)} \qquad \frac{\surd(x),\ \surd(y)}{\surd(x\cdot y)} \qquad \frac{x\downarrow,\ x\uparrow,\ \surd(y)}{\surd(x\cdot y),\ \surd(y\cdot x)}$$

$$\frac{x\uparrow,\ x\not\downarrow}{x\cdot y\uparrow} \qquad \frac{x\uparrow,\ x\downarrow,\ y\uparrow}{x\cdot y\uparrow} \qquad \frac{x\downarrow,\ y\downarrow}{x\cdot y\downarrow}$$

$$\frac{x\downarrow,\ y \xrightarrow{a} y'}{x\cdot y \xrightarrow{a} y'} \qquad \frac{\surd(x),\ y \xrightarrow{a} y'}{x\cdot y \xrightarrow{a} y'} \qquad \frac{x \xrightarrow{a} x',\ \neg zero(y),\ \neg n\xi(x'\cdot y)}{x\cdot y \xrightarrow{a} x'\cdot y}$$

$$\frac{y \overset{1}{\mapsto} y',\ x \overset{1}{\not\mapsto},\ x\downarrow}{x\cdot y \overset{1}{\mapsto} y'} \qquad \frac{y \overset{1}{\mapsto} y',\ \surd(x)}{x\cdot y \overset{1}{\mapsto} y'} \qquad \frac{x \overset{1}{\mapsto} x',\ x\not\downarrow,\ \neg zero(y),\ \neg n\xi(x'\cdot y)}{x\cdot y \overset{1}{\mapsto} x'\cdot y}$$

$$\frac{x \overset{1}{\mapsto} x',\ x\downarrow,\ y \overset{1}{\mapsto} y'}{x\cdot y \overset{1}{\mapsto} x'\cdot y+y'}$$

$$\frac{ch(x),\ \neg zero(y)}{ch(x\cdot y)} \qquad \frac{x\downarrow,\ ch(y)}{ch(x\cdot y)} \qquad \frac{\surd(x),\ ch(y)}{ch(x\cdot y)}$$

$$\frac{n\xi(x),\ \neg zero(y)}{n\xi(x\cdot y)} \qquad \frac{x\downarrow,\ n\xi(y)}{n\xi(x\cdot y)} \qquad \frac{\surd(x),\ n\xi(y)}{n\xi(x\cdot y)}$$

Table 3. Rules for sequential composition (a an action)

$$\frac{zero(x)}{zero(x\cdot y),\ zero(y\cdot x)} \qquad \frac{\surd(x),\ \surd(y)}{\surd(x\cdot y)} \qquad \frac{x\downarrow,\ x\uparrow,\ \surd(y)}{\surd(x\cdot y),\ \surd(y\cdot x)}$$

$$\frac{x\uparrow,\ x\not\downarrow}{x\cdot y\uparrow} \qquad \frac{x\uparrow,\ x\downarrow,\ y\uparrow}{x\cdot y\uparrow} \qquad \frac{x\downarrow,\ y\downarrow}{x\cdot y\downarrow}$$

$$\frac{x\downarrow,\ y\xrightarrow{a} y'}{x\cdot y\xrightarrow{a} y'} \qquad \frac{\surd(x),\ y\xrightarrow{a} y'}{x\cdot y\xrightarrow{a} y'} \qquad \frac{x\xrightarrow{a} x',\ \neg zero(y),\ \neg n\xi(x'\cdot y)}{x\cdot y\xrightarrow{a} x'\cdot y}$$

$$\frac{y\overset{1}{\mapsto} y',\ x\overset{1}{\not\mapsto},\ x\downarrow}{x\cdot y\overset{1}{\mapsto} y'} \qquad \frac{y\overset{1}{\mapsto} y',\ \surd(x)}{x\cdot y\overset{1}{\mapsto} y'} \qquad \frac{x\overset{1}{\mapsto} x',\ x\not\downarrow,\ \neg zero(y),\ \neg n\xi(x'\cdot y)}{x\cdot y\overset{1}{\mapsto} x'\cdot y}$$

$$\frac{x\overset{1}{\mapsto} x',\ x\downarrow,\ y\overset{1}{\mapsto} y'}{x\cdot y\overset{1}{\mapsto} x'\cdot y+y'}$$

$$\frac{ch(x),\ \neg zero(y)}{ch(x\cdot y)} \qquad \frac{x\downarrow,\ ch(y)}{ch(x\cdot y)} \qquad \frac{\surd(x),\ ch(y)}{ch(x\cdot y)}$$

$$\frac{n\xi(x),\ \neg zero(y)}{n\xi(x\cdot y)} \qquad \frac{x\downarrow,\ n\xi(y)}{n\xi(x\cdot y)} \qquad \frac{\surd(x),\ n\xi(y)}{n\xi(x\cdot y)}$$

Table 3. Rules for sequential composition (a an action)

4 Axiomatization

We proceed to give an axiomatization. We claim that this axiomatization is sound and complete for the model of transition systems modulo strong bisimulation.

We split the axioms into two tables. The first one, Table 5, gives the axioms for alternative and sequential composition, i.e. for basic process algebra. We use the following precedence order on the operators: sequential composition $\cdot$ has highest binding power and alternative composition $+$ has lowest binding power; all other operators are not ranked in between. We have not investigated thoroughly whether some axioms are derivable from the others, so it is possible that some optimization can be achieved.

In the axiomatization, we used that any non-zero process is either equal to nil or has a $\dot{\delta}$ summand (can be written as $x+\dot{\delta}$).

The second table, Table 6, gives the axioms for parallel composition, using the auxiliary operators $\between$, left merge, and $|$, communication merge. As a parameter, the communication function γ is given as a partial, commutative and associative function on the set of atomic actions (see [14, 13]). It is straightforward to give SOS rules and tables as in Section 2 for these auxiliary operators.

In the axiomatization, we used that every consistent process (i.e., every process different from $0, \dot{\delta}, \dot{\epsilon}$) has nil as a summand.

On the basis of the information given, many subtheories can be constructed that contain any subset of the set of constants given, and equational specifications of subalgebras of reduced models investigated (so-called SRM specifications, see [6].

In this paper, we just considered strong bisimulation equivalence, and no special laws for silent steps. For a development of branching bisimulation in the presence of discrete timing, we refer to [8]. Using their theory, we can infer in our setting laws as $\underline{\tau}\cdot\underline{\tau}\cdot(x+nil)=\underline{\tau}\cdot(x+nil)$ (but not $\underline{\tau}\cdot\underline{\tau}=\underline{\tau}$!).

$x + y = y + x$	$x + 0 = x$
$(x + y) + z = x + (y + z)$	$\dot{\epsilon} + \dot{\delta} = \dot{\epsilon}$
$x + x = x$	$nil + \dot{\epsilon} = \underline{\epsilon}$
$(x + y) \cdot z = x \cdot z + y \cdot z$	$nil + \dot{\delta} = \underline{\delta}$
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	$\underline{\epsilon} + \underline{\delta} = \underline{\epsilon}$
$\delta = \underline{\sigma} \cdot \delta$	$\epsilon + \delta = \epsilon$
$\epsilon = \underline{\epsilon} + \underline{\sigma} \cdot \epsilon$	$\chi + \epsilon = \chi$
$a = \underline{a} + \underline{\sigma} \cdot a$	$\bot + \chi = \bot$
$\dot{\epsilon} \cdot x = x$	$\Delta = \tau \cdot \Delta + \epsilon$
$x \cdot \dot{\epsilon} = x$	$\Delta \cdot \Delta = \Delta$
$0 \cdot x = 0$	$\underline{a} \cdot \bot = \underline{\delta}$
$x \cdot 0 = 0$	$\underline{\sigma} \cdot \bot = \underline{\delta}$
	$a \cdot \bot = \delta$
$\underline{\epsilon} \cdot \dot{\delta} = \underline{\delta}$	
$\underline{\epsilon} \cdot (x + nil) = x + nil$	$nil \cdot (x + y) = nil \cdot x + nil \cdot y$
$\epsilon \cdot (x + y) = \epsilon \cdot x + \epsilon \cdot y$	$nil \cdot \dot{\delta} = \underline{\delta}$
$\epsilon \cdot \dot{\delta} = \delta$	$nil \cdot nil = nil$
$\epsilon \cdot nil = \delta$	$nil \cdot \underline{\delta} = \underline{\delta}$
$\epsilon \cdot \underline{\delta} = \delta$	$nil \cdot \underline{a} = \underline{a}$
$\epsilon \cdot \underline{a} = a$	$nil \cdot \underline{\sigma} = \underline{\sigma}$
$\epsilon \cdot \underline{\sigma} = \underline{\sigma} \cdot \epsilon$	$nil \cdot \underline{\epsilon} = \underline{\delta}$
$\epsilon \cdot \underline{\epsilon} = \epsilon$	$nil \cdot \chi = \chi$
$\epsilon \cdot (x + \delta) = x + \delta$	$nil \cdot \bot = \bot$
$\chi \cdot nil = \chi$	$\chi \cdot (x + \dot{\delta}) = \chi$
$\bot \cdot nil = \bot$	$\bot \cdot (x + \dot{\delta}) = \bot$
$\dot{\delta} \cdot nil = \dot{\delta}$	$\dot{\delta} \cdot (x + \dot{\delta}) = \dot{\delta}$
$\underline{\delta} \cdot nil = \underline{\delta}$	$\underline{\delta} \cdot (x + \dot{\delta}) = \underline{\delta}$
$\delta \cdot nil = \delta$	$\delta \cdot (x + \dot{\delta}) = \delta$

Table 5. Axioms of basic process algebra ($a \in A$)

$x \parallel y = x \between y + y \between x + x \mid y$	
$(x+y) \between z = x \between z + y \between z$	$x \mid y = y \mid x$
$\dot{\delta} \between x = \dot{\delta}$	$(x+y) \mid z = x \mid z + y \mid z$
$x \between \dot{\delta} = \dot{\delta}$	$\dot{\delta} \mid x = \dot{\delta}$
$0 \between (x + nil) = 0$	$\dot{\epsilon} \mid x = \dot{\delta}$
$(x + nil) \between 0 = 0$	$0 \mid 0 = 0$
$x \between \dot{\epsilon} = x$	$0 \mid (x + nil) = 0$
$\dot{\epsilon} \between (x + \dot{\epsilon}) = \dot{\epsilon}$	$nil \mid (x + nil) = nil$
$\dot{\epsilon} \between 0 = 0$	$\underline{\delta} \mid (x + nil) = \underline{\delta}$
$\dot{\epsilon} \between (x + y) = \dot{\epsilon} \between x + \dot{\epsilon} \between y$	$\underline{\epsilon} \mid (x + nil) = \underline{\delta}$
$\dot{\epsilon} \between nil = \dot{\delta}$	$\underline{\sigma} \cdot x \mid \underline{a} y = \underline{\delta}$
$\dot{\epsilon} \between \underline{\delta} = \dot{\delta}$	$\underline{\sigma} \cdot x \mid \underline{\sigma} \cdot y = \underline{\sigma} \cdot (x \mid y)$
$\dot{\epsilon} \between \underline{a} x = \dot{\delta}$	$\underline{a} \cdot x \mid \underline{b} \cdot y = \underline{c} \cdot (x \parallel y)$ if $\gamma(a,b) = c$
$\dot{\epsilon} \between \underline{\sigma} \cdot x = \dot{\delta}$	$\underline{a} \cdot x \mid \underline{b} \cdot y = \underline{\delta}$ otherwise
$\dot{\epsilon} \between \chi = \dot{\delta}$	$\chi \mid (x + nil) = \chi$
$\dot{\epsilon} \between \bot = \dot{\delta}$	$\bot \mid (x + nil) = \bot$
$nil \between (x + nil) = nil$	
$\underline{\delta} \between (x + nil) = \underline{\delta}$	$\underline{\epsilon} \between (x + \dot{\epsilon}) = \underline{\epsilon}$
$\underline{a} \cdot x \between (y + nil) = \underline{a} \cdot (x \parallel (y + nil))$	$\underline{\epsilon} \between nil = \underline{\delta}$
$\underline{\sigma} \cdot x \between nil = \underline{\sigma} \cdot (x \parallel nil)$	$\underline{\epsilon} \between \underline{\delta} = \underline{\delta}$
$\underline{\sigma} \cdot x \between \underline{\delta} = \underline{\delta}$	$\underline{\epsilon} \between (\underline{a} \cdot y + z) = \underline{\epsilon} \between z$
$\underline{\sigma} \cdot x \between \underline{\sigma} \cdot y = \underline{\sigma} \cdot (x \between y)$	$\underline{\epsilon} \between (\underline{\sigma} \cdot y + z) = \underline{\epsilon} \between z$
$\underline{\sigma} \cdot x \between (\underline{a} \cdot y + z) = \underline{\sigma} \cdot x \between z$	$\underline{\epsilon} \between \chi = \underline{\delta}$
$\underline{\sigma} \cdot x \between (\dot{\epsilon} + y) = \underline{\sigma} \cdot x \between y$	$\underline{\epsilon} \between \bot = \underline{\delta}$
$\underline{\sigma} \cdot x \between \chi = \underline{\delta}$	$\chi \between (x + nil) = \chi$
$\underline{\sigma} \cdot x \between \bot = \underline{\delta}$	$\bot \between (x + nil) = \bot$

Table 6. Axioms of parallel composition ($a \in A$)

5 Conclusion

We considered many different constants in process algebra, and determined their relationships. The question remains, where Jaco fits into this scheme. After careful consideration, it was determined that none of the constants suffices. The only solution is to add one more constant to the scheme, viz. the $\top$.

References

1. L. Aceto and M. Hennessy. Termination, deadlock and divergence. *Journal of the ACM*, 39:147–187, 1992.
2. J.C.M. Baeten. Embedding untimed into timed process algebra; the case for explicit termination. In L. Aceto and B. Victor, editors, *Proceedings EXPRESS'00*, number 39 in Electronic Lecture Notes in Theoretical Computer Science, pages 45–62, State College, Pa., 2000. Elsevier.
3. J.C.M. Baeten and J.A. Bergstra. Process algebra with a zero object. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings CONCUR'90*, number 458 in Lecture Notes in Computer Science, pages 83–98, Amsterdam, 1990. Springer Verlag.
4. J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.
5. J.C.M. Baeten and J.A. Bergstra. Real space process algebra. *Formal Aspects of Computing*, 5(6):481–529, 1993.
6. J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra. *Formal Aspects of Computing*, 8(2):188–208, 1996.
7. J.C.M. Baeten and J.A. Bergstra. Process algebra with propositional signals. *Theoretical Computer Science*, 177:381–406, 1997.
8. J.C.M. Baeten, J.A. Bergstra, and M.A. Reniers. Discrete time process algebra with silent step. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction (Essays in Honour of Robin Milner)*, Foundations of Computing, pages 535–570. MIT Press, 2000.
9. J.C.M. Baeten, J.A. Bergstra, and Gh. Ştefănescu. Process algebra with feedback. In A. Ponse, Y. Venema, and M. de Rijke, editors, *Modal Logic and Process Algebra*, number 53 in CSLI Lecture Notes, pages 13–37. CSLI Publications, 1995.
10. J.C.M. Baeten and C.A. Middelburg. Process algebra with timing: Real time and discrete time. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 10, pages 627–684. Elsevier Science, Amsterdam, 2001.
11. J.C.M. Baeten and C.A. Middelburg. *Process Algebra with Timing*. EATCS Monographs. Springer Verlag, 2002.
12. J.C.M. Baeten and M.A. Reniers. Termination in timed process algebra. Technical Report CSR 00/13, Eindhoven University of Technology, Computer Science Department, 2000.
13. J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.
14. J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1/3):109–137, 1984.
15. J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.
16. J.A. Bergstra and J.W. Klop. A convergence theorem in process algebra. In J.W. de Bakker and J.J.M.M. Rutten, editors, *Ten Years of Concurrency Semantics*, pages 164–195. World Scientific, 1992.
17. S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.
18. R. Milner. *A Calculus of Communicating Systems*. Number 92 in Lecture Notes in Computer Science. Springer Verlag, 1980.
19. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
20. G.D. Plotkin. An operational semantics for CSP. In D. Bjørner, editor, *Proceedings Conference on Formal Description of Programming Concepts II*, pages 199–225. North-Holland, Amsterdam, 1983.
21. C. Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. In B. Jonsson and J. Parrow, editors, *Proceedings CONCUR'94*, number 836 in LNCS, pages 433–448, Uppsala, 1994. Springer Verlag.
22. J.L.M. Vrancken. The algebra of communicating processes with empty process. *Theoretical Computer Science*, 177:287–328, 1997.

Beste Jaco,

Na een indrukwekkende carrière binnen en buiten het CWI van bijna 40 jaar houd je het hier nu voor gezien en gelijk heb je. Je kwam als jeune premier, je vertrekt nu als elder statesman. Wat me van je functies het meest is bijgebleven is de obscure naam van het eerste door jou geleide subthema (toen heette dat subsectie): Nonnumerieke Research en Nonnumeriek Onderwijs. Het overkoepelend cluster was de Rekenafdeling, het dito thema was de Programmeersectie. Beide namen gaven aan hoe relatief onbekend en weinig omvangrijk de informatica toen nog was: computers werden gebruikt om te rekenen en te programmeren en wat je er nog meer mee kon doen onttrok zich aan bijna ieders waarneming en werd maar nonnumeriek genoemd. Na de kanteling eind 1972 van de Rekenafdeling kreeg je een eigen afdeling met de eigentijdse naam Informatica.

Hoewel ik nooit bij jou op de afdeling heb gezeten (voor mij misschien maar goed ook, want je heette nogal streng te zijn en ik had wat startproblemen op weg naar mijn promotie) kreeg ik toch al gauw met je te maken, omdat we geregeld elkaars post kregen, die we dan netjes bij elkaar terugbrachten. Toen je prof. werd, nam dat wat af, maar de Franse post voor M DE BAKKER krijg ik tot de dag van vandaag in mijn vakje.

Na mijn toetreding tot het Bureau als beleidsmedewerker kreeg ik nog meer met je te maken. Dit verliep in het algemeen plezierig, al kon je af en toe ook wel eens boos worden, als ik naar jouw mening onvoldoende werk afleverde. Maar dat hoort erbij.

Jaco, ik wens je heel veel plezier in de post-MC/CWI-periode. Ik twijfel er overigens niet aan dat je nog vaak op het CWI bent te zien.

Miente Bakker

On Jaco and from MTPC to MTOP

J.A. Bergstra*

June 5, 2002

This short note is written in connection with Jaco de Bakker's end of professional carreer party. Given the fact that science is unbounded and always unfinished, leaving reseach is a big step that marks the qualitative difference between a single person's limited contribution and the ongoing efforts of 'the system'. One then begins to think in terms of someone's contribution to the field and to its management and organization. This will be done by many different people each from their own perspective. So a person doing so is also in the process of making a contribition, in this case to a sort of distributed biography.

What will my contribution be. Jaco has organized many workshops, conferences meetings, projects and so on. And he has been an editor of many publications of various forms. These things I all tend to forget. Then, together with Jaco we have made numerous trips to Brussels, mainly to the Philips laboratory with intruiging people such Pierre Wodon and Jacques Hagelstein. We also visited the Commision staff to be informed, or to survive one of the many assessments of our projects. These trips were always nice.

Then Jaco got me into the CWI. As far as I can see the CWI is indeed the best place to be for research in computing, at least in the Netherlands. Perhaps I left too early, but that is not important now. Then Jaco produced research. The difficulty of scientific writing, and our field may be a problematic example by all standards, is that it takes so much energy to read other people's work. Moreover, given the current style of working, we all write more than our colleagues can (and will) read. Jaco is here no exception. Strange questions emerge: would Jaco have been a stronger researcher had he written twice as many papers? Or is it just the other way around? Is it valid to try to formulate an opinion given that one has seen and read only a part of the work? Can I make sort of an assessment of what Jaco did that had an influence on me? Is that of any use, and moreover, is it reliable? A clear fact is that Jaco has formulated a question on the existence of fixed points in models of the metric process theory that he designed together with Jeff Zucker. This question led Jan-Willem Klop and myself in 1982 to the work on BPA, PA and later ACP. Some five years later Jaco suggsted questions about the topology of process spaces that we have also spent considerable time on.

*University of Amsterdam, Programming Research Group, & Utrecht University, Department of Philosophy, Applied Logic Group, email: Jan.Bergstra@phil.uu.nl

In order to make some progress I will now make some drastic simplification. The model is given by the Britisch research assessment exercise. One paper per researcher per year will be evaluated. I will consider no more than 2 products of Jaco. This would have been unfair to Gödel and to Einstein, but for many other workers it may well be acceptable. Take the best two products and turn these into an 'image' of the scholar under consideration. I do not claim any virtues for this method else than that it actually allows one to reduce the staggering amount of data to manageble proportions. The two pieces of work are very easy to determine: the book: 'Mathematical Theory of Program Correctness' (MTPC) and the paper with Jeff on the Metric Theory of Processes (MTOP). MTPC now stands out as a very clearcut story in classical style on program language semantics. In particular MTPC tells the story of procedural languages, exploiting (then) recent insights on CPO's but allowing the reader to forget about sophisticated models of the lambda calculus (quite reassuring indeed). Till today MTPC stands out as a classical piece of work, which can be used as a reference by anyone who intends to be serious about the history and the bibliography of work in program language semantics. The work is very solid, one need not do it again to get the details right or anything of that sort. Improvements are always possible, but Jaco's MTPC is a very clear statement of what can be achieved along a certain line of thinking. Whoever deviates from that line of thought regarding the same topic should in fact explain why.

Moving on it became apparent that reactive behavior, concurrency and nondeterminism could not so easily be handled in the style of MTPC. That necessitates the search for another domain which has better potential as a semantic carrier for these topics. This led to MTOP. MTOP is one of many process theories that were developed between 1975 and 1990. In fact it is one of the lesser known ones, the competition with CCS, TCSP, Petri-nets amongst other being quite strong indeed. Has MTOP been outdated by newer developments? Does it still carry a message? For me it does. I will try to explain this. MTOP produces a story about processes which is in no way depending on any form of requirements engineering or aprioiri analysis regarding processes that may be useful for computing science. It features, till this day, a maximum of reuse of known and standard mathematical structures and methods. A particular class of hereditary finite sets is equipped with a (known) metric. Then the topological completion is introduced. In that world automatically a large class of fixed point equations can be solved. Solving fixed point equations proves to be of equal importance for the problems of concurrency as it had proved already for the issues of MTPC. No special equivalences, bisimulations, congruences, SOS rules, or special purpose axiom systems are used in MTOP. Instead the fact that ZF set theory has the notion of strong bisimulation of finite trees built in is used. Here we see the 'weakness' of MTOP as well. It came at a time that one cannot anymore rely on the fact that students in computing know some ZF style set theory as well as the construction of the real numbers in set theory, which makes use of exactly the ingredients that MTOP puts to work. Therefore they cannot appreciate how thoroughly classical this work is, and how remarkable it is that a very workable process space can be manufactured using mathematics

which every first year mathematics student may still be supposed to understand immediately.

Let us now consider MTOP in more detail. Like all theories it can be explained in boring detail, allowing anyone who sees it for the first time to ignore it and not to be bothered in the least. Having said that I would say that the topological completion of a metric space still stands out as the one obvious, effective and simple idea that will always bring you somewhere in this field. I am now working on, what I call Program Algebra, and the construction of models for it seems to be easier by means of the MTOP methods than via SOS and the subsequent introduction of process equivalences. Following our work on BPA, PA and ACP, I prefer to work with projective limits, but that has always been known to be an available mathematical alternative.

Having arrived at this point I may say that I would consider it entirely justified to teach all first year students in computing the basic facts of MTOP, be it in a modernized presentation (i.e. using projective limits and a bit more algebra, to make the set theoretic coding details a bit less visible). Moving from MTOP to infinite transition systems is difficult. The topology starts working against one because the phenomenology is far from obvious. This problem disappears at once if the set theoretic coding is given up in favor of an algebraic style and the idempotence of alternative composition is initially ignored. This may still be a very effective basis for process theory. Also the addition of silent steps and the introduction of various sophisticated abstract bisimulations runs counter to the metrical intuitions that underly MTOP. I would claim that it still pays off quite a lot to have these intuitions before one moves towards the theology of silent steps. Mobile features are equally hard to add to MTOP, the set theory giving little semantic intuition about something which is perhaps closer to a lambda calculus. This need not be the definite picture, however. On the contrary, on the basis of recent work with Jos Baeten and Lou Feijs I hold it plausible that mobile features will be fairly easily understood in a metric process space in due time.

From the point of view of computer science the story about the real numbers, using Cauchy sequences and metric spaces is an exercise in semantics. That one tries to explain program semantics to an audience that has not been and will never be exposed to these aspects of mathematics may simply be a temporary anomaly. (It is now widely acknowledged that our teaching programs are too narrow, so a change may be forthcoming soon?) For a population of students who have been thoroughly exposed to the aforementioned Caucy sequences, MTOP is the obvious next step that enables one to see and to appreciate the rich phenomenolgy of tau's, fairness, priorities, infinite state spaces, mobility, causality, readies, failures and so on.

Here is a 'conjecture': on the long run process algebra will either disappear or its main emphasis will be on features that exist within the metric (or projective limit) model. This model will support the long and tedius build up of modular libraries of operators that might turn process algebra into a substantial tool. Features that defeat being modeled in the metric model may simply be lacking the required modularity, so that these do not integrate so well in the long run.

As a conclusion I may say that the MTOP has had just as much influence on my perception of my topics of research as my exposure to the construction of the real numbers had many years ago. It is totally obvious that Jaco need not walk around in person on a daily basis in order to keep such content matter in existence. It will survive entirely by itself for many years to come.

Some Thoughts on Teaching Software Engineering*
Central Rôles of Semantics

Dines Bjørner
Computer Science and Engineering
Informatics and Mathematical Modelling
Building 322, Richard Petersens Plads†
Technical University of Denmark
DK–2800 Lyngby, Denmark
E–Mail: db@imm.dtu.dk

19th of May 2002

Abstract

Somehow fitting the occasion for which this essay is written, I reminisce on a quarter century's teaching of software engineering. I delineate the sciences of computers and computing, and motivate a triptych of software development: From domain descriptions via requirements prescriptions to software architecture and component design. Via characterisations of domain attributes, stake–holder perspectives and facets; domain, interface and machine requirements and attendant domain projections, domain instantiations, domain extensions and domain initialisations, I venture into a series of characterisations of various forms of principles and techniques of abstraction and modelling: Basic ("assembler language–like") and conceptual ("modular, procedural") abstractions and models. From here I go back into issues of semiotics: Pragmatics, semantics and syntax, and to the art, craft and discipline of descriptions.

Throughout — framing in boxes — I risk my reputation by uttering dogmas and prejudices.

As my teaching is based mostly on own research, I will primarily only refer to own recent reports, lecture notes and publications — as well as to those of Jaco de Bakker's which had a deep influence on my work in the 1960s and 1970s.

Contents

*This essay is to appear in a Liber Amicorum in honour of Professor, Dr Jaco W. de Bakker on the occassion of his retirement, 31st of August, 2002.

†Prof. Richard Petersen was the instigator of the development and use of the first Danish stored programme electronic computer

1 Some Software Engineering Dogmas

1.1 From Science via Engineering to Technology

The engineer "walks the bridge" between science and technology: Creates technology based on scientific insight; and, vice–versa, analyses technological artifacts with a view towards understanding their possible scientific contents. Both science and technology; both synthesis and analysis.

In teaching software engineering we must teach both programming methodological, ie. computing science skills, as well as more mundane computer science skills — in order to walk both ways, forwards and "backwards", to, respectively from technology.

Work in the early 1970s, at the IBM Vienna Laboratory [1] on establishing a semantics for the IBM programming language PL/I, as well as that of my groups in the late 1970s and early to mid 1980s in establishing semantics for the CHILL and Ada programming languages [2, 3], clearly, besides a scientific content, had the semantics definition engineers walk the bridge from the technologies of PL/I, CHILL and Ada "back" to science, trying to discover whatever scientific values those languages might have had.

Current work on trying to establish a semantics for UML is not of the same nature. Whereas PL/I, CHILL and Ada could indeed be said, or claimed, to be soundly based on previous good scientific insight into programming language design, it seems that UML missed the boat: 20 years of painstaking programming methodological insight appears not to have been embodied in UML.

In teaching software engineering we are confronted with the recurrent dilemma of being asked to train in current, fashionable technologies, for which there is little scientific merit. My answer is one of almost Dutch Reformed Church pietism and strictness: Don't. Instead I focus on the wonderful, practical theories that ought to have been in those technologies.

1.2 CS ⊕ CS ⊕ SE

Computer science, to me, is the study and knowledge of the artifacts that can "exist" inside computers: Their mathematical properties: Models of computation, and the underlying mathematics itself. Computing science, to me, is the study and knowledge of how to construct those artifacts: (i) Programming languages, their pragmatics, their semantics, including proof systems, their syntax, and the principles and techniques of use; (ii) computing systems such as compilers, operating systems, database management systems, data communication systems, etc.; and (iii) applications — mostly.

The difference, between the computer and the computing sciences, is, somehow, dramatic: One is more contemplative, analytic, appeals to clever school boys. The other more adventurous, daring — in my prejudiced mind.

Software engineering is the art, discipline, craft, science and logic of conceiving, constructing, and maintaining software.

The sciences are those of applied mathematics and computing. I consider myself both a computing scientist and a software engineer.

Many so–called Computer Science departments, for lack of understanding, or because their lecturing cum researcher staff can't agree, or other, "waver" a course of teaching software engineering that sometimes contain too much theoretical computer science courses in relation to too few real programming methodological courses, or, vice–versa: contains too little theoretical computer science courses in relation to too many rather ordinary programming and software technology courses.

My, 'ideal' software engineering candidates have been taught one semester computer science courses in at least (i) automata theory, formal languages and computability, (ii) algorithms and complexity theory, and (iii) the theories underlying algebraic, axiomatic and denotational semantics. They have also been taught one or two semester computing science cum programming methodology courses each in (i) functional, (ii) logic, (iii) imperative, and (iv) parallel programming, in (v) algorithms and data structures (basic, intermediate and advanced), in (vi) real–time, embedded and concurrent systems design, and a (vii–ix) two–three semester course in the kind of software engineering outlined in this essay. After all this come courses in (x) compiler design, (xi) operating systems design, (xii) database management and database system design, (xiii) distributed systems & protocol design, etc. All of these computing science courses are based on the use of formal techniques: formal specification, analysis, verification, etc.

1.3 Informatics

1.3.1 The Quadrant of Informatics

Informatics, such as I see it, is a combination of: Mathematics, computer & computing science, software engineering, and applications. Some "sobering" observation: Informatics relates to information technology (IT) as biology does to bio—technology; $\mathcal{E}$tcetera ! I am somewhat "saddened" at the confusion of our field, of informatics, with that of information technology. The former is based on mathematics: Logic and algebra. The latter on the natural sciences. The former is a universe of intellectual quality: Elegance, beauty, correctness, fit. The latter is a universe of material quantity: Faster, smaller size, lower cost, larger capacity.

Many departments, who call themselves 'Informatics' departments, do not have a precise understanding of what they mean by 'informatics', and many computer science, or computing science (etc.) departments do not make sufficiently clear the relationships between their sciences and those primarily behind information technology.

1.3.2 Informatics of Infrastructures

What makes informatics interesting, to me at least, is its relations with the concept of infrastructures.

The World Bank Concept of Infrastructure: One may speak of a country's or a region's infrastructure.[1] But what does one mean by that ?

According to the World Bank,[2] 'infrastructure' is an umbrella term for many activities referred to as 'social overhead capital' by some development economists, and encompasses activities that share technical and economic features (such as economies of scale and spill-overs from users to non-users).

Our interpretation of the 'infrastructure' concept, see below, albeit different, is, however, commensurate.

[1] Winston Churchill is quoted to have said, during a debate in the House of Commons, in 1946: ... *The young Labourite speaker that we have just listened to, clearly wishes to impress upon his constituency the fact that he has gone to Eton and Oxford since he now uses such fashionable terms a 'infra–structure'* ...

[2] Dr. Jan Goossenarts, an early UNU/IIST Fellow, is to be credited with having found this characterisation.

Concretisations: Examples of infrastructure components are typically: The transportation infrastructure sub–components (road, rail, air and water [shipping]); the financial services industry (banks, insurance companies, securities trading, etc.); health–care; utilities (electricity, natural gas, telecommunications, water supply, sewage disposal, etc.); and perhaps also education, etc. ?

1.3.3 A Software Concept of Infrastructure

At UNU/IIST we took, in the mid 1990s[3], a more technical, and, perhaps more general, view, and saw infrastructures as concerned with supporting other systems or activities.

Software for infrastructures is likely to be distributed and concerned in particular with supporting communication of information, people and/or materials. Hence issues of (for example) openness, timeliness, security, lack of corruption, and resilience are often important.[4]

1.3.4 A Unification Attempt

We shall accept the two characterisations in the following spirit: For a socio–economically well–functioning infrastructure (component) to be so, the characterisations of the intrinsics, the support technologies, the management & organisation, the rules & regulations, and the human behaviour, must, already in the domain, meet certain "good functionality" conditions.

That is: We bring the two characterisations together, letting the latter "feed" the former. Doing so expresses a conjecture: One answer, to the question" *"What is an infrastructure"*, is, seen from the viewpoint of systems engineering, that it is a system that can be characterised using the technical terms typical of computing systems.

The Question and its Background: The question and its first, partial answer, only makes sense, from the point of view of the computer & computing sciences if we pose that question on the background of some of the achievements of those sciences. We select a few analysis approaches. These are aspects of denotational, concurrency, type/value, and knowledge engineering approaches, as well as a computer science approach.

An important aspect of my answer, in addition to be flavoured by the above, derives from the *semiotics* distinctions between: *pragmatics*, *semantics*, and *syntax*.

1.3.5 Discussion

The major challenge in software engineering seems to lie in the successful, and in the believably so, development of trustworthy, very large scale computing systems for world–wide infrastructure components. Such systems embody well–nigh all facets of computing abstractions: denotational as well as computational, concurrent and distributed (hence spatial), real-time (temporal), and, as we shall comment on later, autonomous multi–agencies probably embodying mechanised speech acts.

This is what makes informatics fascinating. And this is why *Unifying Theories of Programming* [4] becomes an overriding theoretical concern.

In understanding infrastructures we shall seek the semantics road.

[3] I write "mid 1990's" since that is what I can vouch for.

[4] The above wording is due, I believe, to Chris George, UNU/IIST.

1.4 A *Triptych* Software Engineering Dogma

Before software can be designed, we must understand the requirements. Before requirements can be expressed we must understand the domain.

Software engineering consists of the engineering of domains, engineering of requirements, and the design of software. In summary, and ideally speaking: We first *describe* the domain: $\mathcal{D}$, from which we *prescribe* the domain requirements; from these and interface and machine requirements, ie. from $\mathcal{R}$, we *specify* the software design: $\mathcal{S}$. In a suitable reality we secure that all these are properly documented and related: $\mathcal{D}, \mathcal{S} \models \mathcal{R}$, when all is done !

In proofs of correctness of software ($\mathcal{S}$) wrt. requirements ($\mathcal{R}$) assumptions are often stated about the domain ($\mathcal{D}$). But, by domain descriptions $\mathcal{D}$ we mean "much more" than just expressing such assumptions.

Domain engineering resembles, but is not the same as knowledge engineering. In the former we seek only understanding of the domain — through establishing precise, mathematical models. In the latter, it seems, knowledge engineers seek immediately computable models.

I find, but this may be just my personal "hang up", that many programming methodology cum software engineering curricula do not sufficiently enunciate the difference between domain and requirements engineering.

2 Formal Techniques vs. "Formal Methods"

A significant characteristics in our approach is that of the use of formal techniques: formal specification, verification & model checking. The area as such is usually — colloquially — referred to as "formal methods". By a method we understand a set of *principles* of analysis and for selecting *techniques* and *tools* in order efficiently to achieve the construction of an efficient artifact.

As such methods cannot be formal: Being carried out by humans whose ingeniousness or lack of same cannot be "straight–jacketed" — cannot be formalised.

3 Some Issues of Domain Engineering

Since, as we claim, domain engineering is rather a novel idea, we shall spend some space enunciating ideas of domain modelling.

3.1 Domain Facets

To understand the application domain we must describe it. We must, I believe, describe it, informally (ie. narrate), and formally, as it is, the very *basics*, ie. the *intrinsics;* the *technologies* that *support* the domain; the *management & organisation* structures of the domain; the *rules & regulations* that should guide human behaviour in the domain; those *human behaviours:* the correct, diligent, loyal and competent works; the absent–minded, "casual", sloppy routines; and the near, or outright criminal, neglect. *&c.*

We have not found these domain facet concepts in the software engineering literature — so perhaps we should advertise, here, their usefulness in directing the software engineering in being systematic about certain aspects of domains.
For the software engineer, to make effective use of the domain facet concepts, principles and techniques for their abstraction and modelling must be provided.

In [5] we present some such principles and techniques.

3.2 Domain Attributes

Michael Jackson, in [6], has convincingly, enunciated a number of attributes of phenomena of domains: Static and dynamic attributes; tangible and intangible attributes — which we further classify into humanly or otherwise physically perceivable tangible attributes as well as only conceptually, hence — in a sense — intangible, conceptual attributes; and zero, one or multi–dimensionality attributes. To these we add: discrete, continuous and chaotic attributes (not necessarily along a time axis); and temporal, spatial and combined time/space attributes.

For the software engineer, to make effective use of the domain attribute concept, principles and techniques for their abstraction and modelling must be provided.

In [11] we present some such principles and techniques

3.3 Domain Perspectives

Software serves many masters: Some are themselves machines, others are humans. Some humans, ie. managers, order software for their enterprise. Others use it daily. yet others "suffer" from such uses. In any software development project it is of interest to try delineate the spectrum of stake–holders: From enterprise owner, via strategic, tactical and operational enterprise management, to "floor" ("blue collar") workers, enterprise clients, providers of such software, IT, etc., regulatory agencies, citizens "at large", and the ever present, talkative politicians who interfere in almost anything: All have a stake in the computing systems, one way or another. The art is now to sufficiently identify this spectrum, to identify their perspective upon the domain, and, hence, to model these perspectives, coherently and consistently.

Too much "softness", too much politically correct talk is, to my taste, connected with the topic of securing proper attention to stake–holder perspectives. It need not be so. There simply is an exciting theory and likewise worthwhile engineering techniques for modelling the stake–holder perspective notion.

In [5] we present some such principles and techniques. In general, Chapter 15 of our lecture notes, [7], cover many principles, techniques and tools of domain abstraction and modelling.

3.4 The Evidence

How are we describing the domain ? We are *rough sketching* it, and *analysing* these sketches to arrive at *concepts*. We establish a *terminology* for the domain. We *narrate* the domain: A concise professional language description of the domain using only (otherwise precisely defined) terms of the domain. And we *formalise* the *narrative*. We then *analyse* the *narrative* and the *formalisation* with the aims of: *validating*, "against" domain stake–holders, and *verifying* properties of, the *domain description*.

Software engineering text books perfunctorily cover the subject of documentation, see next, but rarely enunciate the distinction between rough sketching, terminologisation, narration, and formalisation. We find it of utmost importance that the software engineer be trained in principles and techniques of description: Of expressing oneself in natural, albeit the professional language of the application domain at hand.

3.5 On Documentation in General

In general there will be many documents for each phase[5], stage[6] and step[7] of development: Informative documents: Needs and concepts, development briefs, contracts, *&c.* Descriptive/prescriptive documents: Informal (rough sketches, terminologies, and narratives) and (formal models) analytic documents: Concept formation, validation, and verification. These sets of documents are related, and occur and re–occur for all phases.

3.6 Discussion

We find that current, popular software engineering text books let the students down on making the above distinctions — etcetera !

[8, 9, 10] provide various examples of the application of domain modelling principles and techniques. [11] provides a summary overview.

In describing domains we shall seek the semantics road.

4 Some Issues of Requirements Engineering

We see requirements prescriptions as composed from three viewpoints: Domain, interface and machine requirements. We survey these.

Requirements are about the machine: The hardware and software to be designed.

4.1 Domain Requirements

Requirements that can be expressed solely with reference to, ie. using terms of, the domain, are called *domain requirements*. They are, in a sense, "derived" from the *domain understanding*. Thus whatever vagueness, non–determinism and undesired behaviour in the domain, as expressed by the respective parts of the domain *intrinsics, support technologies, management & organisation, rules & regulations*, and *human behaviour*, can now be constrained, if need be, by becoming requirements to a desirably performing computing system.

We do not find, in the software engineering literature, this distinction between, on one hand doing a sufficiently proper at understanding, including, notably, formalising, models of the domain, and, on the other hand, "deriving", as it were, domain requirements from domain models. So perhaps we should advertise, here, their usefulness in directing the software engineering in being systematic about certain aspects of requirements.

[5] Domain, requirements and software design are three main phases of software development.

[6] Phases may be composed of stages, such as for example the domain requirements, the interface requirements and the machine requirements stages of the requirements phase, or, as another example, the software architecture and the software component stages of the software design phase.

[7] Stages may then consist of one or more steps of development, typically data type reification and operation transformation — also known as refinements.

The development of domain requirements can be supported by principles and techniques of *projection:* Not all of the domain need be supported by computing — hence we project only part of the domain description onto potential requirements; *determination:* Usually the domain description is described abstractly, loosely as well as non–deterministically — and we may wish to remove some of these properties; *extension:* Entities, operations over these, events possible in connection with these, and behaviours on some kinds of such entities may now be feasibly "realisable" — where before they were not, hence some forms of domain requirements extend the domain; and *initialisation:* Phenomena in the world need be represented inside the computer — and initialising computers is often a main computing task in itself, as is the ongoing monitoring of the "state" of the 'outside' world for the purpose of possible internal state (ie. database) updates. There are other specialised principles and techniques that support the development of requirements.

We do not find these domain projection, determination, extension and initialisation concepts in the software engineering literature — so perhaps we should advertise, here, their usefulness in directing the software engineering in being systematic about certain aspects of requirements.

In describing domain requirements we shall seek the semantics road.

4.2 Interface Requirements

Requirements that deal with the phenomena shared between external users (human or other machines) and the machine (hardware and software) to be designed, such requirements are called *interface requirements.* Examples of areas of concern for interface requirements are: Human computer interfaces (HCI, CHI), including graphical user interfaces (GUIs), dialogues, etc., and general input and output (examples are: Process control data sampling (input sensors) and controller activation (output actuator)). Some interface requirements can be formalised, others not so easily, and yet others are such for which we today do not know how to formalise them.

The old "adage": 'User friendliness' has become 'pat'. What we increasingly need, and what we, in fact, increasingly, can also express formally, is what is meant by 'user friendliness', namely that the interface reflects only, and exactly those concepts that are indigenous to the domain — albeit in some, usually diagrammatically rendered form.

4.3 Machine Requirements

Requirements that deal with the phenomena which reside in the machine are referred to as *machine requirements.* Examples of concerns of machine requirements are: performance (resource [storage, time, etc.] utilisation), maintainability (adaptive, perfective, preventive, corrective and legacy–oriented), platform constraints (hardware and base software system platform: development, operational and maintenance), business process re–engineering, training and use manuals, and documentation (development, installation, and maintenance manuals, etc.).

Whereas domain requirements seem formalisable, that is, whereas it seems possible to express domain requirements precisely, such seems not the case, currently with machine requirements: All right, we can mathematically express for example performance and dependability issues, but we seem not to know how to "refine" such expressions into provably related implementations — such as we increasingly know how to do it for domain requirements.

[12] covers notions of domain to requirements "derivation". So does a later [13]. [14] provides a summary overview.

5 Some Issues of Software Design

Once the requirements are reasonably well established software design can start. We see software design as a potentially multiple stage, and, within stages, multiple step process. Concerning stages one can identify two "abstract" stages.

5.1 Software Architectures

The software architecture design stage in which the domain requirements find an computable form, albeit still abstract. Some interface requirements are normally also, abstract design–wise "absolved".

5.2 Software Components

and the software component design stage in which the machine requirements find a computable form. Since machine requirements are usually rather operational in nature, the software component design is less abstract than the software architecture design. Any remaining interface requirements are also, abstract design–wise "absolved".

The views that software architectures emerge from domain requirements, and hence have their "root" in domain models, whereas software component designs emerge from machine requirements, and hence have their root in the possibilities of the machine — those views seem different from that of the prevailing literature. We have had some difficulty, in the past, in getting enthusiastically excited about the seemingly, individually isolated concepts of "software architectures", respectively "software components". We now know why. It is all very simple.

[15] covers notions of domain to requirements to software architecture and component design.

6 Abstraction and Models

Whether we describe domain phenomena, or prescribe requirements, or specify software, we express models. They are just models. They are not the real thing. In expressing models we abstract. Judicious use of abstraction seems more important to software engineers than to, for example, automotive engineers, or to chemical engineers.

[16] provides a condensed overview of what I believe to be pertinent abstraction and modelling techniques.

6.1 Abstraction

To conceive of pleasing and adequate abstractions seems to be an art. But much can be learned from reading beautiful abstractions. In the following we will mention a few ideas.

It seems, to me, that we, in the computer and computing sciences as well as in the software engineering education, still have a lot to communicate to our students: The art, the discipline, the craft, the logic, and the science of abstraction.

In capturing abstractions we shall seek the semantics road.

6.2 Models and Modelling

When describing (as for domains), prescribing (as for requirements) and specifying (as for software designs), we create models. It is therefore important, for the software engineer, to decide which aspects these models are to portray and how: Whether they are analogic, iconic, or analytic models; whether they are prescriptive or descriptive; whether they do so in extension or in intension; and for what purposes the models are established: to gain understanding; and/or to get inspiration and to inspire; and/or to present, educate and train; and/or to assert and predict;and/or to implement. Finally, the principle of modelling, manifested by the problem domain itself, the mathematical structure of the model, and the identification between the two, clearly spells out the importance of the software engineer being conscious about the rôle, *&c*, of models.

It seems that engineers, for example taught control theory or operations research, are made better aware of the above notions of models and modelling — judging, simply, from the software engineering versus the respective literatures of control theory and operations research.

In building models we shall seek the semantics road.

6.3 Basic Modelling Principles & Techniques

Distinctions are made between property and model oriented abstract modelling. Property oriented models are usually algebraically cum axiomatically expressed. Model oriented models are usually expressed in terms of such mathematical entities as Booleans, numbers, sets, Cartesians, lists, maps and functions. The type concept, in the early days of Scott's (and de Bakkers) contributions to mathematical models for the λ–Calculus known as domain theory, is perhaps the finest contribution computer science has made to mathematics. Specification languages such as `OBJ`, `Act One`, `CafeOBJ`, `Maude`, and `CASL` facilitate property oriented specifications. Specification languages such as `VDM-SL`, `Z`, `RAISE's RSL`, `B`, and `ASM` facilitate model oriented specifications. All chronologically listed.

In a proper software engineering education we must make sure, I think, that our candidates know at least one property oriented specification approach, and at least two, reasonably diverse, model oriented approaches.

Both property oriented and model oriented specification work benefit from focusing first on semantics.

6.4 Additional Modelling Principles & Techniques

Over and above the basic modelling principles we find a number of additional modelling paradigms: (i) Non–determinism, internally or externally "chosen", and looseness; (ii) specification programming: Applicative (functional), imperative, logic, parallel, and algebraic programming — composing functions, dealing with references, propositions, composing processes or composing algebras; (iii) hierarchical ("top–down") versus compositional ("bottom–up") development and/or presentation of models; (iv) denotational versus computational semantic models; (v) configurations in terms of a spectrum from the more static contexts (environments) to the more dynamic states (stores); (vi) temporal, spatial and time/space models; *&c.*

The reader, by now, starts to see a picture emerging: One of a sizable variety of abstraction and modelling principles, techniques and tools. From the basics of property and model oriented abstractions, via the one just listed above (i-vi, *&c.*), and the domain attributes, stake–holder perspectives, domain facets, to the domain requirements projects, determinations, extensions and initialisations, and so and and so forth. Software engineering is indeed a universe of intellectual conceptualisations. And we need study and teach it all !

All of them reflecting, at their core, semantics, in one way or another.

7 Complementary Issues

7.1 On the Importance of Semiotics

"It is not for nothing" that Jaco de Bakker has devoted considerable time to the semantics of programming languages. It is, without question, the singlemost important issue of software engineering. Not just as a user of programming languages. But much more because whatever artifact the software engineer is designing, core issues of pragmatics, semantics and syntax enter into the design: The input to any software system, however end–user oriented, constitutes a language. Proper attention to what it is, ie. values of which semantic types, that one wishes to express, is hence of utmost importance. After semantics, in importance, come syntax.

Pragmatics is being neglected as a scientific and engineering topic although it, without question, is the most important topic of the three components of semiotics. Probably because it, by its very nature, cannot be formalised.

The software engineer must be well–versed in semiotics — in particular semantics and syntax — modelling techniques: denotational, operational, axiomatics, etc. And the software engineer must well–trained in making appropriate distinctions as to when a problem is a pragmatic, or is a semantic, or is ("just") a syntactic problem.

7.2 Logics, Agents and Language–based Knowledge Engineering

There is the *knowledge engineering* view. In one, of several, variants of this view — and we shall only cover that variant, albeit ever so briefly — one focuses on *logics*, *agent behaviours* and *speech acts*.

The *logics* area has two facets to it: The classical logics which are part also of the denotational, concurrency, type system, and formal techniques facets described and assumed earlier, and the less classical logics of modal logics. Thus, by *logics* we here mean those of the *epistemic logics* of *knowledge & belief*, the *deontic logics* of *permission & obligation*, the *modal logics* of *possibility & necessity*, *&c*. Other logics are relevant — also when describing domains: *dynamic logics* of action, *defeasible, uncertainty* and *possibilistic logics*, *logics* of *belief revision*, *&c*. These are not just logics of AI and logic programming but also logics of general domain engineering.

We present what may be termed the AI approach to *"agency"*, but intend to "lift" the AI *"agency"* notions to apply, inter alia, to domain engineering as well as to software (requirements and design). Agents *interact* through *communication*. Agents come in groups: *Multi agent "systems"*. Agents perform both *competitive* and *co–operative* tasks. *Open multi agent "systems"* have agents serve different interests, *autonomously* and *heterogeneously*. Just like humans ! Agent *interaction* (alphabetically listed)[8] involves *arguments:* Formation of reasons, drawing of conclusions, and applying these actively; *commitments, conversations, co–ordination, dialogue, negotiation, obligation, planning, &c*. In doing so agents deploy various modal logics, and, as we shall next see: Speech acts.

Speech acts are characterised by: *Locutions* — The physical utterances of speakers; *illocutions* — The intended meaning of speaker utterances; and *perlocutions* — The actions that result from locutions. Wrt. illocutions, speech acts are often classified in the following five *performatives: Assertive*, ie. statements of fact; *directive*, ie. commands, requests or advice; *commissive*, eg. promises; *expressive*, eg. feelings and attitudes; and *declarative* which entail the occurrence of an action in themselves. Obviously speech acts and agents relate strongly.

We see an increasing fusion of software engineering, as it is classically known, with knowledge engineering, as indicated above. We cannot, at all, accept current, separational distinctions between software engineering and AI.

It is surprising to see how very little work has been actually done, based on 30 years of semantics of programming language, to formulate clear and concise semantics descriptions of multi–agencies and speech acts. My student, Mr Hans Madsen Petersen has recently completed a nice MSc Thesis on this topic (20 June, 2002).

7.3 On Description Principles and Techniques

Michael Jackson, in his delightful [6], espouses a "theory" of descriptions based on designations, definitions and refutable assertions — with designations centering around notions of recognition rules and designation sets. Although we do modify Jackson's description theory a bit, we wish here to gratefully acknowledge our debt. Jackson refrains, perhaps wisely so, from providing other than simple propositional logic examples. We advocate, in Chapter 13 of [7], a number of description principles, techniques and tools.

We can not overemphasise the importance of our software engineering candidates becoming far more capable of succinctly mastering their own, national language. One almost wishes 'Rhetoric', as a university discipline back !

[8] The listing is extracted from my MSc student, Hans Madsen Petersen's MSc pre–project report: *Agent Communication Languages and Speech Acts — and their Semantics*, October 2001.

7.4 Towards a Philosophy of Informatics

An issue central to description of domains is *"What can be described ?"* This question borders to, or is an outright philosophical problem. Such philosophy of logic, of language and of mathematics topics as mereology [17], epistemology [18], and ontology, are seen as increasingly important concepts of an emerging philosophy of informatics discipline.

> We can only advocate that every software engineer be given serious courses in Theories of Science, Philosophy of Mathematics, and of Logic, and of Language.

We advice our students to have such books as [19, 20, 21, 22] at their desk for ready consultation.

8 Conclusion

We have surveyed a set of notions of software engineering. Our departure pont, in this discussion, has been that the underlying science of informatics has a strong base in mathematics — as well as in the philosophies of logic, mathematics and language: Epistemology, ontology and mereology, to name a few ingredients. But that it, basically, does not have a basis in the natural sciences. There is still a long way to go for may computer & computing scientists before their university colleagues from the natural sciences and mathematics understand that informatics is a whole new discipline, like theirs are old disciplines.

We have provided the subtitle: *A Rôle for Semantics* to the main title of this essay. Maybe it has not been sufficiently emphasised above, so let it be emphasised here:

> The rôle of semantics, next to that more elusive one of pragmatics, far overshadows that of syntax. The software engineer must be fluent in semantics modelling, covering as wide a spectrum of techniques as possible. Here Jaco de Bakkers work over the last almost 40 years, including his seminal books [23, 24] have played, play, and will continue to play an important research and teaching function.

In a previous paper [25] we outlined more, including technical, details on the issue of teaching software engineering. In [26] I outlined the contents of a "massive" set of lecture notes, perhaps a publishable book [7], for such a set of main courses in formal techniques based software engineering. These lecture notes represent some 25 years of thinking and practice. They are nearing an n^{th} iteration ($n \approx 4$) of completion ! Together they could easily cover 3–4 semesters of teaching !

9 Acknowledgements

In my earlier years, at the IBM Vienna Laboratory, I came across [27], foreboding exciting things to come. And they came: Over the next almost 20 years I studied and enjoyed the contents and the precise style of numerous papers: [28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]. Beyond 1980 the references are too numerous to list: But work on POOL figures among them. I am sure this Liber Amicorum will bring a comprehensive list of all of Jaco's splendrous works. So: Thanks, Jaco, for a lifetime of steadfast insistence on what we now consider a crowning achievement of yours — as well as of our field: The myriad of principles

and techniques of describing and analysing the semantics of a great variety of programming languages.

The reader will be excused: I primarily refer only to my own current reports, lecture notes and recent publications — and to those of Jaco de Bakker's !

References

[1] H. Bekič, Dines Bjørner, W. Henhapl, C.B. Jones, and P. Lucas. A Formal Definition of a PL/I Subset. Technical Report 25.139, Vienna, Austria, 20 September 1974.

[2] P.L. Haff, editor. *The Formal Definition of CHILL.* See [42]. ITU (Intl. Telecomm. Union), Geneva, Switzerland, 1981.

[3] Dines Bjørner and O. Oest, editors. *Towards a Formal Description of Ada*, volume 98 of *LNCS*. Springer-Verlag, 1980.

[4] C.A.R. Hoare and He Ji Feng. Unifying Theories of Programming. Publ.: Prentice Hall, 1997.

[5] Dines Bjørner. *"What is a Method ?" — A Study of Some Aspects of Software Engineering.* IFIP WG2.3. MacMillan, Oxford, UK, 2002. Programming Methodology: Recent Work by Members of IFIP Working Group 2.3. Eds.: Annabelle McIver and Carrol Morgan. To be published.

[6] Michael A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices.* ACM Press. Addison-Wesley Publishing Company, Wokingham, nr. Reading, England; E-mail: ipc@awpub.add-wes.co.uk, 1995. ISBN 0-201-87712-0; xiv + 228 pages.

[7] Dines Bjørner. *Software Engineering: Theory & Practice.* "In publisher's hand !", expected out 2003. Presently these lecture notes of around 1,000 pages are in a fourth phase of rewriting. Earlier phases took place in the mid 1980s, the late 1980s and the mid to late 1990s. A published book version is to be a "cut" version of the lecture notes.

[8] Dines Bjørner. Towards the E–Market: To understand the E–Market we must first understand "The Market". In *Government E–Commerce Development.* Ningbo Science & Technology Commission, Ningbo, Zhejian Province, China, 23–24 April 2001.

[9] Dines Bjørner. Formal Software Techniques in Railway Systems. In Eckehard Schnieder, editor, *9th IFAC Symposium on Control in Transportation Systems*, pages 1–12, Technical University, Braunschweig, Germany, 13–15 June 2000. VDI/VDE-Gesellschaft Mess– und Automatisieringstechnik, VDI-Gesellschaft für Fahrzeug– und Verkehrstechnik. Invited plenum lecture.

[10] Dines Bjørner. What is an Infrastructure ? In *The UNU/IIST 10th Anniversary Symposium.* UNU/IIST, Springer, March 2002. Eds.: Armando Haeberer, Tom Maibaum and Carlo Ghezzi.

[11] Dines Bjørner. Domain Engineering — A Prerequisite for Requirements Engineering — Principles and Techniques. Technical report, Informatics and Mathematical Modelling, Building 322, Richard Petersens Plads, Technical University of Denmark, DK–2800 Kgs.Lyngby, Denmark, 2001. This paper is one of a series of papers currently being submitted for publication: [43, 16, 14, 44, 45, 46, 47, 48, 49].

[12] Dines Bjørner. Domains as Prerequisites for Requirements and Software *&c*. In M. Broy and B. Rumpe, editors, *RTSE'97: Requirements Targeted Software and Systems Engineering*, volume 1526 of *Lecture Notes in Computer Science*, pages 1–41. Springer-Verlag, Berlin Heidelberg, 1998.

[13] Dines Bjørner. From Domains to Requirements — Some Protocol Challenges. In *FORTE: Formal Protocol Description and Verification Techniques*. IFIP WG6.1, Kluwer Press, 2001.

[14] Dines Bjørner. Requirements Engineering — Some Principles and Techniques — Bridging Domain Engineering and Software Design. Technical report, Informatics and Mathematical Modelling, Building 322, Richard Petersens Plads, Technical University of Denmark, DK–2800 Kgs.Lyngby, Denmark, 2001. This paper is one of a series of papers currently being submitted for publication: [43, 16, 11, 44, 45, 46, 47, 48, 49].

[15] Dines Bjørner. Where do Software Architectures come from ? Systematic Development from Domains and Requirements. A Re–assessment of Software Engneering ? *South African Journal of Computer Science*, 1999. Editor: Chris Brink.

[16] Dines Bjørner. Principles and Techniques of Abstract Modelling — Some Basic Classifications. — Towards a Methodology of Software Engineering. Technical report, Informatics and Mathematical Modelling, Building 322, Richard Petersens Plads, Technical University of Denmark, DK–2800 Kgs.Lyngby, Denmark, 2001. This paper is one of a series of papers currently being submitted for publication: [43, 11, 14, 44, 45, 46, 47, 48, 49].

[17] Peter M. Simons. *Foundations of Logic and Linguistics: Problems and their Solutions*, chapter Leśniewski's Logic and its Relation to Classical and Free Logics. New York, 1985. Georg Dorn and P. Weingartner (Eds.).

[18] Jonathan Dancy and Ernest Sosa, editors. *The Blackwell Companion to Epistemology*. Blackwell Companions to Philosophy. Blackwell Publishers, 108 Cowley Road, Oxford OX4 1JF, UK, 1994.

[19] Rober Audi. *The Cambridge Dictionary of Philosophy*. Cambridge University Press, The Pitt Building, Trumpington Street, Cambridge CB2 1RP, England, 1995.

[20] Ted Honderich. *The Oxford Companion to Philosophy*. Oxford University Press, Walton St., Oxford OX2 6DP, England, 1995.

[21] David Crystal. *The Cambridge Encyclopedia of Language*. Cambridge University Press, 1987, 1988.

[22] Nicholas Bunnin and E.P. Tsui-James, editors. *The Blackwell Companion to Philosophy*. Blackwell Companions to Philosophy. Blackwell Publishers, 108 Cowley Road, Oxford OX4 1JF, UK, 1996.

[23] J.W. de Bakker. *Mathematical Theory of Programming Correctness*. Prentice-Hall, 1980.

[24] J.W. de Bakker and Erik de Wink. *Control Flow Semantics*. 608 pages The MIT Press, Cambridge, Mass., USA, April 1, 1996. ISBN: 0262041545

[25] Dines Bjørner and Jorge Cuellar. Software Engineering Education: The Rôle of Formal Specifications and Design Calculi. Annals of Software Engineering, 6 (1998) 365–409.

[26] Dines Bjørner. On Teaching Software Engineering based on Formal Techniques, Thoughts about and Plans for, A Different Software Engineering Text Book. *Journal of Universal Computer Science*, Vol.7, no.8, 2001: Colloquium "Formal Aspects of Software Engineering". Text of a talk given at the *Abschieds–Symposium* in honour of Professor Peter Lucas on the occassion of his retirement from The Technical University of Graz, Austria.

[27] Dana S. Scott and Jaco de Bakker. Approx. title: Notes on a Mathematical Model for the the λ–Calculus Informal, handwritten notes: IBM Vienna Laboratory, Vienna, Austria, 1979.

[28] J.W. de Bakker. Axiomatics of simple assignment statements. *MR94, Math. Centrum, Amsterdam*, pages 1–37, 1968.

[29] J.W. de Bakker. Semantics of programming languages. In *Advances in Information Systems Sciences, 2*, chapter 3, pages 173–227. Plenum Press, 1969.

[30] J.W. de Bakker. *Recursive Procedures*, volume 24. Math. Centre Tracts, Amsterdam, 1971.

[31] J.W. de Bakker. Axiom systems for simple assignment statements. In *[50]*, pages 1–22, 1971.

[32] J.W. de Bakker and W.P. de Roever. A calculus for recursive program schemes. In M. Nivat, editor, *International Colloquium on Automata, Languages and Programming, European Association for Theoretical Computer Science*, pages 167–196. North-Holland Publ.Co., Amsterdam, 1973.

[33] J.W. de Bakker and L.G.L.T. Meertens. On the completeness of the inductive assertion method. *International Journal of Computer and Information Sciences*, 11:323–357, 1975.

[34] J.W. de Bakker. The fixed point approach in semantics: Theory and applications. In J.W. de Bakker, editor, *Foundations of Computer Science*, pages 3–53. Math. Centre Tracts 63, Mathematisch Centrum, 1975.

[35] J.W. de Bakker. Semantics and termination of nondeterministic recursive programs. In S. Michaelson and R. Milner, editors, *International Colloquium on Automata, Languages and Programming, European Association for Theoretical Computer Science*, pages 435–477. Edinburgh Univ. Press, 1976.

[36] J.W. de Bakker. Least fixed points revisited. *Theoretical Computer Science*, 2:155–181, 1976.

[37] K.R. Apt and J.W. de Bakker. Exercises in denotational semantics. In A. Mazurkiewicz, editor, *Mathematical Foundations of Computer Science, Proceedings*, pages 1–11. Lecture Notes in Computer Science, Vol. 45, Springer-Verlag, 1976.

[38] K.R. Apt and J.W. de Bakker. Semantics and proof theory of Pascal procedures. In A. Salomaa and M.Steinby, editors, *International Colloquium on Automata, Languages and Programming, European Association for Theoretical Computer Science*, pages 30–44. Lecture Notes in Computer Science, Vol 52, Springer-Verlag, 1977.

[39] J.W. de Bakker. Semantics and the foundations of program proving. In B. Gilchrist, editor, *IFIP World Congress Proceedings*, pages 279–284. North-Holland Publ.Co., Amsterdam, 1977.

[40] J.W. de Bakker. Recursive programs as predicate transformers. In *[51]*, pages 165–181, 1978.

[41] J.W. de Bakker. A sound and complete proof system for partial program correctness. In J. Bečvář, editor, *Mathematical Foundations of Computer Science, Proceedings*, pages 1–12, Springer-Verlag, 1979. Lecture Notes in Computer Science, Vol. 74.

[42] Anon. *C.C.I.T.T. High Level Language (CHILL), Recommendation Z.200, Red Book Fascicle VI.12.* See [2]. ITU (Intl. Telecmm. Union), Geneva, Switzerland, 1980 – 1985.

[43] Dines Bjørner. Models, Semiotics, Documents and Descriptions — Towards Software Engineering Literacy. Technical report, Informatics and Mathematical Modelling, Building 322, Richard Petersens Plads, Technical University of Denmark, DK–2800 Kgs.Lyngby, Denmark, 2001. This paper is one of a series of papers currently being submitted for publication: [16, 11, 14, 44, 45, 46, 47, 48, 49].

[44] Dines Bjørner. Healthcare Systems. Towards a Domain Theory for Work Flow Systems. Technical report, Informatics and Mathematical Modelling, Building 322, Richard Petersens Plads, Technical University of Denmark, DK–2800 Kgs.Lyngby, Denmark, 2001. This paper is one of a series of papers currently being submitted for publication: [43, 16, 11, 14, 45, 46, 47, 48, 49].

[45] Dines Bjørner. E–Business. Towards a Domain Theory for Work Flow Systems. Technical report, Informatics and Mathematical Modelling, Building 322, Richard Petersens Plads, Technical University of Denmark, DK–2800 Kgs.Lyngby, Denmark, 2001. This paper is one of a series of papers currently being submitted for publication: [43, 16, 11, 14, 44, 46, 47, 48, 49].

[46] Dines Bjørner. Logistics. Towards a Domain Theory for Work Flow Systems. Technical report, Informatics and Mathematical Modelling, Building 322, Richard Petersens Plads, Technical University of Denmark, DK–2800 Kgs.Lyngby, Denmark, 2001. This paper is one of a series of papers currently being submitted for publication: [43, 16, 11, 14, 44, 45, 47, 48, 49].

[47] Dines Bjørner. Projects & Production: Planning, Plans & Execution. Towards a Domain Theory for Work Flow Systems. Technical report, Informatics and Mathematical Modelling, Building 322, Richard Petersens Plads, Technical University of Denmark, DK–2800 Kgs.Lyngby, Denmark, 2001. This paper is one of a series of papers currently being submitted for publication: [43, 16, 11, 14, 44, 45, 46, 48, 49].

[48] Dines Bjørner. Railways Systems: Towards a Domain Theory. Technical report, Informatics and Mathematical Modelling, Building 322, Richard Petersens Plads, Technical University of Denmark, DK–2800 Kgs.Lyngby, Denmark, 2001. This paper is one of a series of papers currently being submitted for publication: [43, 16, 11, 14, 44, 45, 46, 47, 49].

[49] Dines Bjørner. Financial Service Institutions: Banks, Securities Trading, Insurance, *&c.* Towards a Domain Theory for Work Flow Systems. Technical report, Informatics and Mathematical Modelling, Building 322, Richard Petersens Plads, Technical University of Denmark, DK–2800 Kgs.Lyngby, Denmark, 2001. This paper is one of a series of papers currently being submitted for publication: [43, 16, 11, 14, 44, 45, 46, 48, 47].

[50] E. Engeler. *Symposium on Semantics of Algorithmic Languages*, volume 188 of *Lecture Notes in Mathematics*. Springer-Verlag, 1971.

[51] E. Neuhold. *Formal Description of Programming Concepts (I)*. North-Holland Publ.Co., Amsterdam, Proc. of IFIP TC-2 Work.Conf., St. Andrews Canada, Aug. 1977, 1978.

Beste Jaco,

Als een kersverse filosoof kwam ik uit Groningen naar het CWI waar je me had uitgenodigd voor een sollicitatiegesprek voor een promotieplaats op het CWI. Mijn antwoord 'gitaar gespeeld' op je vraag `wat heb je al die tijd gedaan na je middelbare schooltijd en voor je studie' heeft je waarschijnlijk in eerste instantie doen twijfelen.
Je stuurde me in elk geval voor een nader onderzoek door naar John-Jules Meyer en Joost Kok, twee medewerkers van je.
Een andere verse medewerker van je, Jan Rutten, kende ik overigens al als een vriend van een vriendin van een vriendin uit Nijmegen van mijn toenmalige vriendin die zelf ook uit Nijmegen kwam.
Gewogen en uiteindelijk goed bevonden kon ik aan de slag.
Maar het had niet veel gescheeld of ik was in Eindhoven beland bij je eerste promovendus Willem-Paul de Roever waar ik namelijk toen ook op een promotieplaats heb gesolliciteerd.
Na mijn promotie echter was het zo gezegd toch onvermijdelijk dat ik alsnog naar Eindhoven ging.
Lang hield Willem-Paul mijn aanwezigheid daar niet uit en liet zich Kielhalen.

Om kort te gaan, na Eindhoven, en een jaar VU, kwam ik terecht in Utrecht in de groep van John-Jules Meyer.
Daar zit ik nog, maar sinds kort ten dele: voor het merendeel van mijn tijd ben ik weer terug op het CWI. Dankzij jouw inspanningen, die van Jan Rutten, en zeker niet te vergeten die van Gerard van Oortmerssen. Daar ben ik jou in het bijzonder zeer erkentelijk voor.

Inhoudelijk was ik met mijn bewijstheoretische doelstellingen wel een wat vreemde eend in het semantische poel van wat toen nog de Amsterdamse Concurrency Groep heette.
Zelf heb ik van deze bijeenkomsten geleerd bewijstheoriëen te ontwikkelen vanuit een semantisch perspectief.
Dit perspectief is zeer vruchtbaar gebleken.

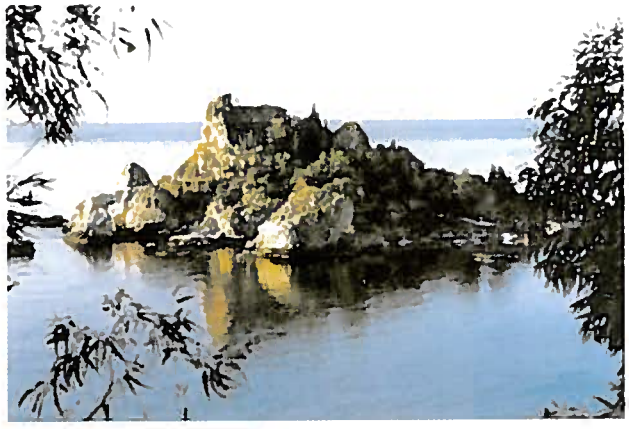
Frank de Boer

Your passion for the islands (by Marcello Bonsangue)

I have been one of your Ph.D. students. It happened almost by sheer chance. In 1991, after my master in Computer Science at the University of Milano, I visited you at CWI to learn more about semantics and concurrency of programming languages. Just before leaving, in 1993, you proposed me a Ph.D. studentship for three years at the Vrije Universiteit in Amsterdam, and you gave me a complete freedom to find my way in the scientific world. Now, in 2002, I still remember many of the conversations we had during all these years. Especially those about islands: the beautiful Capri, the Azores and its whales, the steep coasts of Madeira, the blue sea of Sardinia, and not to forget Cyprus, Isola d'Elba, Corfu, Crete and many others. Many of these islands were chosen for hosting the final meeting of scientific projects in which you have participated, following your mysterious and fascinating idea that projects have to finish on an island.

We never discussed what was behind this idea, but I like to see the coasts of an island as interminable. If we walk along the coastline of an island, each starting point perfectly coincides with the ending point of our previous wandering.

This year your scientific career comes to an end. And like all your projects, also this time it finishes at an island: Sicily. An island that has played a very important role in the history of Punics, Phoenicians, Greeks, Romans, Arabs, Normans, the French, and the Spanish. An Isola Bella, nearby Taormina island that has inspired the work of Giovanni Verga, Luigi Pirandello, Tomasi di Lampedusa and Leonardo Sciascia. A magical island, with a coastline described by Gesualdo Bufalino in "The naked island" as "... a perimeter of wonders, the same which over the centuries has inveigled both merchants and predators, sighting it one morning from the sea.



There rise white cities, there jut aggressive promontories, long tongues of sand surrounding it, damp and smooth, into which the heel voluptuously sinks, unless perchance it should form into shifting heaps, bosoming dunes that the wind moulds and dissolves at will. No rivers of majestic flow disembogue upon it, but lean waters with felicitous names: Simeto, Ippari, Irminio, Salso, Platani, Anapo, Ciane, the estuaries of which seek peace in secret among screens of canes and, in one case, among clumps of half dead papyrus. Elsewhere, between Capo Passero and Noto, the terra firma is bordered with swamps and pools and marshes, where, lovely to see for their flight and their plumage, emigrating birds find refuge. Elsewhere again, between Trapani and Marsala, stretches a chequer-board of salt-pans, and an impressive sight are the motionless flats of waters guarded by Quixotic windmills and flanked by heaps of the finished product.
These are interminable coasts, as is only right in an island whose ins and outs, accurately measured, show as much as 1100 kilometres of shoreline – a considerable proportion of that of the whole of Italy."

To be short, just a perfect place to find an ending point of this period of your life and a perfect occasion to make it coincide with the starting point of your future wanderings.

I wish you all the best,
Marcello

15 februari 2000

De datum boven dit stukje is de dag waarop ik, na mijn komst op het CWI begin 2000, de eerste vergadering van het MT bijwoonde. Keurig op tijd was ik als eerste in M376 aanwezig, zocht een plekje uit en installeerde mijzelf.

Eén voor één kwam men vervolgens binnen en ging zitten op de plaatsen waar men - zoals later bleek- elke keer zat. Bij mijn vorige werkgever waren we bij MT vergaderingen gewoon om elke keer ergens anders te gaan zitten, maar dat was bij het CWI dus niet het geval... Voor het beeld van de lezer ter verduidelijking: vergelijk de tafel in M376 met een klok. Gerard zijn plaats bevindt zich op 6 uur met Coby links van hem, Martin op 10 uur, destijds Hans en nu Jan op 12 uur, Lex op 2 uur en Jaco op 4 uur.

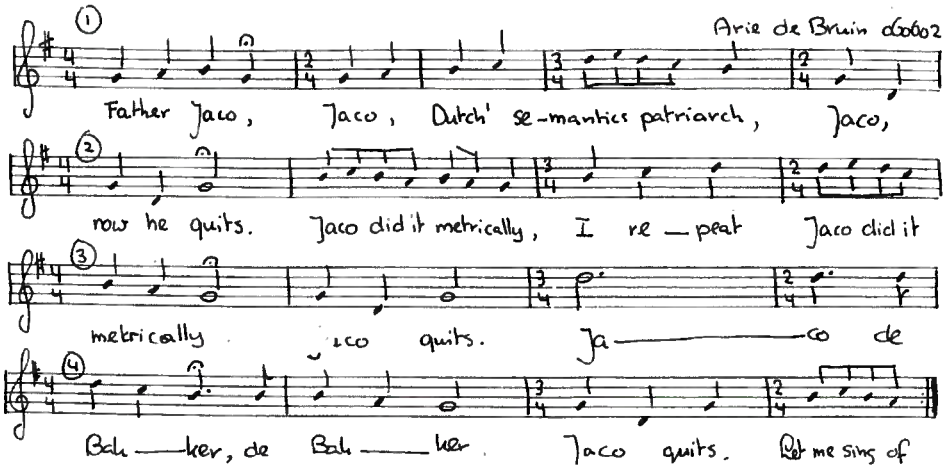
En in al mijn onwetendheid had ik mij precies op de plek van Jaco geïnstalleerd... Het ritueel was mij uiteraard onbekend, maar het leidde tot de nodige hilariteit en een zuinig gezicht bij Jaco. Ik had geen enkel idee wat er nu precies loos was, maar navraag leerde mij nadien hoe het nu precies hoorde en uiteraard ben ik de volgende keer direct naar mijn, 8 uur-positie verhuisd, waar mijn voorgangers Ad en Paul zaten tijdens de MT-vergaderingen. Zo hebben we in M376 gezeten tot en met de laatste vergadering van Jaco, op 12 december 2001.

Ik wil Jaco voor zijn innerlijke rust niet onthouden dat zijn opvolger, Paul, het in januari 2002 helemaal bont heeft gemaakt door maar liefst drie maal achtereen een andere plaats uit te kiezen, waardoor alle andere clusterleiders eveneens hun moment van radeloosheid hebben moeten doorstaan. Het is voorspelbaar dat uiteindelijk ook de mobiliteit van Paul zich kort daarna heeft gestabiliseerd, hij heeft de plaats van Jaco in het MT nu ook letterlijk ingenomen.
Het MT van het CWI is zo weer in een keurig evenwicht beland.

Beste Jaco, je hebt nu al even aan de fase na het CWI kunnen ruiken; ik wens je alle goeds in de toekomst.

Dick Broekhuis.

TO JACO


Arie de Bruin 06 60 02
Father Jaco, Jaco, Dutch' se-mantics patriarch, Jaco,
now he quits. Jaco did it metrically, I re — peat Jaco did it
metrically Jaco quits. Ja — co de
Bak — ker, de Bak — ker Jaco quits. Let me sing of

Jaco's Nieuwe Beelding

Pierpaolo Degano

June 3 2002

The first time I went in Amsterdam was to attend one of these wonderful spring schools on *Mathematical Foundations of Computer Science* that Jaco was used to organize a few years ago. And there I met Jaco for the first time, too. The CWI was still in de Boers Straat, and to go there from the place of a friend of mine who hosted me, I had to pass through one of the most charming parts of the city. So it is natural for me to associate Jaco with Amsterdam. Since then, I often met Jaco here and there in Europe, and I can easily remember the many times he made fun of me: in much less occasions I did the same with him (or even tried to, unbelievable enough!) or I can tell you of when I smiled for the bouquet Jaco just bought and, a couple of years later, he smiled in response of my bunch of flowers; or when I was proud (actually sorry!) when Jaco called me "the only wise man of the group" (I did not join Catuscia, Jan, Joost and Frank going to a discotheque). So it is equally natural for me to associate Jaco with Europe. There is however a special link to Amsterdam, to its *gruchts*, to its buildings.

One of the most charming features of the facades on the canals in Amsterdam I feel is their geometry. It is very different from the classical harmony of the *case* and the *palazzi* I am used to find here in Italy, and certainly not only because windows are so large, suggesting that the interior of the houses is not separated from outside. It is a geometry of lines that intersect forming right angles, disappear for a while and then appear again; of plain figures touching each other, splitting apparently forever, but then they suddenly join together; of a comfortable feeling of well-known drawings that surprise you suggesting a richer space, not yet imagined, through which you can walk with a quite pleasure, safely driven by what you have seen so far - — shouldn't semantics do the same?

Jaco always gave me the same comfortable, solid and neat feeling. Rather than trying to avoid fuzziness — you will forgive me — let me go on by analogy and let me rephrase some statements (taken from an italian translation of an article on *De Stijl*) by Piet Mondrian who I think is also tightly coupled with Amsterdam and its architecture, much in the same way as I presume Jaco is. I found these remarks about the external and the internal aspects of things in pictorial representations fairly applicable to Jaco's mathematical description of computation.

We need to find new tools for expressing [...] the dynamic rhythm

> *of the relationships between objects.*
>
> *Abstraction becomes precise in a mathematical [ly . . . determined] equilibrium among spatial quantities, lines and colors.*
>
> *[. . .] painting may become even more real, less subjective, much more objective when its [specifications] are indeed reified in architecture, so that the artist's craft is coupled with the implementor's.*

These are some of the main reasons for the many things I learnt from Jaco — including *oude jenever* and *nieuwe haring*, too; my fault not catching more, or grasping some only after too long. For them all and for the nice time we had.

Dank U well, Jaco!

Nonnumerieke Research en Onderwijs

door

Dirk Dekker

Bij het afscheid van Jaco de Bakker wil ik graag een bijdrage leveren aan zijn Liber Amicorum. In een korte terugblik sta ik stil bij een deel van zijn wetenschappelijke loopbaan, vooral waar die met de mijne verweven was toen wij samen op het Mathematisch Centrum (MC) waren. Hiervoor zoek ik enige steun bij desbetreffende delen van de serie jaarverslagen van het MC [1].

Het verslag van het jaar 1964 vermeldt:

> 'De volgende wetenschappelijke medewerkers werden in de loop van dit jaar aangesteld:' --- 'drs J.W. de Bakker (per 1 Mei) Rekenafdeling' ---.

Verder in dat jaar over hem niets bijzonders.
In het verslag van 1965 lezen we als eerste activiteit van Jaco:

> '6 - 24 September: Deelname van de heren de Bakker, Mailloux en Nederkoorn aan de Nato Summer School on Programming Languages. Prof. van Wijngaarden hield een serie lezingen over "Formal definition of syntax and semantics of programming languages".'

Dit verslag vermeldt tevens de eerste publicatie van Jaco, MR 74: 'Formal definition of algorithmic languages with an application on the definition of syntax and semantics of ALGOL 60, 98 p.'. Dit was meteen een schot in de roos, want zijn proefschrift twee jaar later zal, op kleine wijzigingen na, dezelfde titel dragen [2].
Hij promoveert aan de Universiteit van Amsterdam (UvA) op 17 mei 1967.
Het jaarverslag van 1967 vermeldt verder:

> '25 Augustus - 2 September: 3rd International Congress for Logic, Methodology and Philosophy of Science te Amsterdam, bijgewoond door Prof. van Wijngaarden, Prof. Kruseman Aretz en de heren Brandt Corstius en Mailloux. Lezing van de heer de Bakker over: "Automatic and Programming Languages".'

Gaf Jaco op het congres een lezing en woonde hij het niet bij? Deze en andere activiteiten leidden tot een reeks rapporten van Jaco in de welbekende MR-reeks (zie ook [3]).

In 1970 begint er stap voor stap een nieuwe fase in Jaco's loopbaan. Ik citeer:

> 'Binnen de Programmeersectie werd op 1 October de subsectie Nonnumerieke Research en Onderwijs gevormd, die onder leiding kwam te staan van dr. J.W. de Bakker'.

Op 1 januari 1971 (de ontwikkelingen gaan snel) wordt deze subsectie gepromoveerd tot 'sectie' van de Rekenafdeling (RA) en wordt Jaco gepromoveerd tot souschef van deze sectie. De RA heeft dan drie secties, namelijk, naast de twee genoemde, de sectie Numerieke Wiskunde (NW) waarvan ik in die tijd souschef was.

Merkwaardige benaming eigenlijk: 'Nonnumerieke Research en Onderwijs'. Elders in het jaarverslag heet het 'Nonnumerieke Research en Nonnumeriek Onderwijs' voor alle duidelijkheid. 'Nonnumeriek' staat blijkbaar tegenover 'Numeriek'. Waren Jaco en ik als souschefs van de onderhavige secties dan te beschouwen als elkaars tegenpolen? En was er dan geen betere karakterisering van het vakgebied te bedenken?
Wat benaming betreft, eveneens merkwaardig is het dat de ouderwetse functienamen 'chef' en 'souschef' zich zo lang op het MC / CWI gehandhaafd hebben.

Op 1 september 1971 gebeurt er van alles (waar Jaco overigens weinig mee te maken had): ik verlaat het MC vanwege mijn benoeming tot hoogleraar aan de UvA, Reind van de Riet wordt souschef ad interim van de sectie NW en tegelijk wordt naast de drie secties in de RA de 'Rekendienst, onderverdeeld in Projectprogrammering en Machinedienst' opgericht onder leiding van 'drs. G.J.R. Förch, hoofd'.
Wat Jaco zelf betreft: in dat jaar verschijnt zijn tweede boek in de MC Tracts serie, namelijk nummer 24 getiteld: 'Recursive Procedures', 108 p. (1971) [4].

In 1971 werd het 'MC-25 Informatica symposium' georganiseerd ter viering van het 25-jarig jubileum van Professor dr ir Aad van Wijngaarden, directeur van het MC en vanaf het begin chef van de RA. Dit symposium vond plaats in 1972, op 6 en 7 januari, doch de gelijknamige publicatie met bijdragen van de negen sprekers, waaronder Jaco, verscheen in 1971 [5]. Het organisatiecomité bestond uit de drie souschefs van de Rekenafdeling: Jaco, Reind van de Riet en ondergetekende. Dit was de belangrijkste activiteit waarin Jaco, Reind en ik hebben samengewerkt.

In 1972 gebeurt er verder nog van alles in de RA. Op 1 mei wordt - na het vertrek van Gert-Jan Förch en vanwege het operationeel worden van SARA - de Machinedienst opgeheven. Op 1 december verlaat Reind van de Riet het MC en worden de secties Systeemprogrammatuur en Nonnumerieke Research samengevoegd tot de sectie 'Informatica' (eindelijk een passende naam), waarvan Jaco souschef wordt.
Vanaf 1 januari 1973 wordt de RA gesplitst in de afdeling Informatica, waarvan Jaco chef wordt, en de afdeling Numerieke Wiskunde (NW), met Piet van der Houwen als chef. Piet krijgt als souschef Herman te Riele. Jaco krijgt geen souschef, hij kan het alleen af; wel wordt voor zijn afdeling Reind gedurende een jaar als adviseur benoemd. In datzelfde jaar, op 1 mei 1973 wordt Jaco buitengewoon hoogleraar aan de Vrije Universiteit. Hierna begint er voor hem een stabiele en zeer vruchtbare periode.

Vanwege mijn vertrek van het MC naar de UvA werden de contacten tussen Jaco en mij minder veelvuldig. Toch bleef er een band tussen ons bestaan, en wel onder auspiciën van Van Wijngaarden. Het zwaartepunt van deze contacten lag niet op het MC, maar op een onverwachte plek, namelijk restaurant 'Kopenhagen', een restaurant aan het Rokin, waar wij, Amsterdamse informatici, onder leiding van Van Wijngaarden zo'n tien keer bijeen gekomen zijn van 1977 tot 1981.

Bij dit gezelschap hoorden later ook Van Emde Boas, Tanenbaum en sommige gastdocenten. Onder het eten van de voortreffelijke vis en het genot van de overheerlijke wijn bespraken wij de informatica in de regio. Dit wordt door Reind van de Riet treffend beschreven in zijn bijdrage aan het Liber Amicorum bij gelegenheid van mijn afscheid van de UvA [6].
Het restaurant was niet goedkoop. Jaco waagde het eens op te merken dat de prijzen aan de hoge kant waren. Ik heb daarop toen gesuggereerd om bij Dorrius bijeen te komen. Die suggestie viel bij Van Wijngaarden niet in goede aarde. Hij zei zoiets als: "Ik heb er geen trek in om raasdonders te eten".
In 1981 kwam er een eind aan onze genoeglijke bijeenkomsten. De studie Informatica ging in dat jaar officieel van start, onder meer bij de beide Amsterdamse universiteiten, die inmiddels met elkaar en met de RU Utrecht een officiële samenwerking waren aangegaan.

Het einde van deze bijeenkomsten bracht met zich mee dat Jaco en ik elkaar sindsdien jarenlang slechts incidenteel ontmoetten. Recentelijk mogen wij ons evenwel verheugen in een nieuw contact met elkaar in het kader van het NWO-project 'Biografie van Van Wijngaarden', uit te voeren door dr Gerard Alberts. We hadden elkaar bijna uit het oog verloren, doch zeker niet uit het hart. Vandaar dat ik hierbij gaarne deze bijdrage lever.

Literatuurverwijzingen

[1] Stichting Mathematisch Centrum: Jaarverslagen;
2e Boerhaavestraat 49, Amsterdam (O).

[2] J.W. de Bakker: Formal definition of programming languages with an applica tion to the definition of ALGOL 60 (24 cm, 207 p.);
Mathematical Centre Tracts 16, Mathematisch Centrum Amsterdam, 1971.

[3] M.C. Publications 1946 - 1971, 1 Bibliography and Author index, 2 KWIC index; Mathematisch Centrum Amsterdam, 1971.

[4] J.W. de Bakker: Recursive procedures (108 p.);
Mathematical Centre Tracts 24, Mathematisch Centrum Amsterdam, 1971.

[5] J.W. de Bakker, G.A. Blaauw, A.J.W. Duijvestijn (co-auteurs: G.A.M. Kamsteeg - Kemper & J.P. Schaap - Kruseman), E.W. Dijkstra, P.J. van der Houwen,
F.E.J. Kruseman Aretz, W.L. van der Poel, M.V. Wilkes & G. Zoutendijk:
MC-25 Informatica symposium; Mathematical Centre Tracts 37,
Mathematisch Centrum Amsterdam, 1971.

[6] R.P. van de Riet: "Kopenhagen", bijdrage in het Liber Amicorum "Is er nog nieuws?", aangeboden aan Prof. Dr Th.J. Dekker, verzameld door P. van Emde Boas, P.W. Hemker, W. Hoffmann, P.J. van der Houwen & P.R. Pfluger; Amsterdam, 27 november 1992.

Afscheid van Jaco

Aan prof.dr. Jaco de Bakker
ter gelegenheid van zijn afscheid van het CWI.

Beste Jaco,

Als je dit leest zit ik op een conferentie in Hongarije, dus deze brief is tevens mijn persoonlijke afscheid van jou. De brief bestaat uit twee delen: een wetenschappelijk deel en een kort persoonlijk naschrift.

De Thue Morse reeks

Mijn wetenschappelijke bijdrage sluit aan bij het stuk van Jan Willem Klop in deze zelfde afscheidsbundel, dat ik van Jan Willem onder embargo te lezen heb gekregen. Je zult je herinneren dat Jan Willem in de CWI lezing ter gelegenheid van zijn eredoctoraat kort refereerde aan de Thue Morse reeks. Noem deze reeks M. Jan Willem gaf de versie die start met 1. Noem het resultaat van omwisselen van nullen en enen in de Thue Morse reeks M'. De reeks M' is wat je krijgt als je het Thue Morse proces start met 0.

Het voorbeeld van de Thue Morse reeks intrigeerde me. De traditionele manier om hiernaar te kijken is recursief, door de reeks te zien als opgebouwd als het resultaat van successieve expansie met behulp van de map

$$\begin{aligned} 0 &\mapsto 01 \\ 1 &\mapsto 10. \end{aligned}$$

Dit geeft:

$$1 \Longrightarrow 10 \Longrightarrow 1001 \Longrightarrow 10010110 \Longrightarrow 1001011001101001 \Longrightarrow \cdots$$

Of voor M':

$$0 \Longrightarrow 01 \Longrightarrow 0110 \Longrightarrow 01101001 \Longrightarrow 0110100110010110 \Longrightarrow \cdots$$

De specificatie laat zich rechtstreeks omzetten in een functioneel programma. Ter illustratie zal ik in dit stuk de taal Haskell gebruiken; deze brief is tevens een proeve van 'literate programming', het is in feite een becommentarieerd programma in Haskell.[1] De programma code is steeds de omkaderde tekst. Hier is het Haskell programma voor benaderingen van de Thue Morse reeks dat gebruik maakt van recursieve expansie:

```
expand ""       = ""
expand ('0':xs) = '0':'1': expand xs
expand ('1':xs) = '1':'0': expand xs

thuemorse 0 = "1"
thuemorse (n+1) = expand (thuemorse n)
```

[1] Zie `www.haskell.org` voor informatie over de taal.

De aanroep `thuemorse n` genereert the eerste 2^n tekens van de Thue Morse reeks M. We krijgen bij voorbeeld:

```
Main> thuemorse 0
"1"
Main> thuemorse 1
"10"
Main> thuemorse 2
"1001"
Main> thuemorse 3
"10010110"
Main> thuemorse 4
"1001011001101001"
Main> thuemorse 5
"10010110011010010110100110010110"
Main> thuemorse 6
"1001011001101001011010011001011001101001100101101001011001101001"
Main>
```

Corecursieve programma's voor M en M'

Dit is aardig, maar het volgt *niet* de uitleg van Jan Willem bij zijn lezing. De manier waarop Jan Willem de Thue Morse recks in zijn praatje introduceerde was anders. De beginstring "1" is gegeven. Daarna kijk je steeds naar wat er al staat, swapt de nullen en enen, en schrijft het resultaat van de swap achter wat er al staat.

Omwisselen van de symbolen in een string van nullen en enen gaat als volgt:

```
swap ""       = ""
swap ('1': xs) = '0': swap xs
swap ('0': xs) = '1': swap xs
```

De procedure die Jan Willem gaf in zijn lezing is geen recursieve procedure maar een corecursieve. En inderdaad, het leidt tot een prachtig corecursief programma:

```
morse xs = swap xs ++ morse (xs ++ swap xs)

tm1 = '1' : morse "1"
```

Dit kan natuurlijk ook door eerst te swappen, en dan de geswapte string als parameter aan de hulpfunctie mee te geven. Dit geeft:

```
thue xs = xs ++ thue (xs ++ swap xs)

tm1a = '1' : thue "0"
```

Dit werkt, en het leuke is dat het *precies* de intuitieve uitleg volgt die Jan Willem gaf. Ik neem dit voorbeeld op als opgave in het boek van Kees Doets en mij over Redeneren en Programmeren met Haskell.[2] Ik ben op het ogenblik bezig met het afmaken en redigeren van de tekst. Het boek besteedt ruim aandacht aan het aanleren van wiskundige redeneertechnieken. In dat kader worden niet alleen inductieve bewijsmethoden geoefend, maar is er ook aandacht voor corecursie en coinductie. Ons boek is bij mijn weten het eerste tekstboek (bedoeld als inleiding redeneren en programmeren voor informatici) waarin recursie en corecursie, en inductie en coinductie, expliciet naast elkaar worden behandeld. Vandaar dat ik nu overal corecursie zie.

Overigens kent deze procedure ook een recursieve versie. De eerste 2^n tekens van M worden ook gegeven door `thuemorse2 n`:

```
thuemorse2 0 = "1"
thuemorse2 (n+1) = thuemorse2 n ++ swap (thuemorse2 n)
```

Jan Willem geeft in zijn *Wonderful Stream* voor jou[3] vier manieren om the Thue Morse reeks te genereren. De uitleg van zijn lezing is wat hij in zijn tekst manier (i) noemt. De definitie met behulp van $1 \to 10, 0 \to 01$ is manier (ii) uit zijn tekst. Als je met behulp van manier (ii) de hele reeks wilt genereren kan dat als volgt:

```
thmo n = thuemorse n ++ thmo (n+1)

tm2 = thmo 0
```

Dit is weer een voorbeeld van corecursief programmeren: de definitie van `thmo` gebruikt een parameter n, maar er is geen basisgeval, zoals bij een recursieve definitie.

Manier (iii) van Jan Willem definieert een functie $b(n)$ als het aantal enen in de binaire expansie van n, modulo 2. $b(n)$ geeft positie n in de reeks M'.

In de implementatie maken we gebruik van een hulpfunctie `binary` die een integer omzet in zijn binaire representatie, beschouwd als een lijst van nullen en enen. De functie $b(n)$ kijkt naar die lijst, filtert de enen eruit, en neemt de lengte van die nieuwe lijst, modulo 2:

[2] Kees Doets en Jan van Eijck, *Reasoning, Computation and Representation using Haskell*, onder submissie bij Cambridge University Press.

[3] Jan Willem Klop, *A Wonderful Stream for Jaco*, deze bundel.

```
binary x = reverse (bits x)
  where bits 0 = [0]
        bits 1 = [1]
        bits n = (rem n 2) : bits (quot n 2)

b x = length (ones x) `mod` 2
  where ones n = filter (==1) (binary n)
```

Dit geeft:

```
Main> map b [0..25]
[0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,0,0,1]
```

Of als je liever een string hebt dan een reeks integers:

```
Main> map (intToDigit . b) [0..25]
"01101001100101101001011001"
```

Hier staat `intToDigit . b` voor de compositie van de functies `b` en `intToDigit`. De functie `intToDigit` zet een cijfer om in het corresponderende ASCII teken.

Haskell is een taal voor lazy list processing, dus je kunt ook de hele stroom genereren. Dit leidt tot de volgende definitie van de Thue Morse reeks (`swap` is nodig om M' om te zetten in M):

```
tm3 = swap (map (intToDigit . b) [0..])
```

Als ik het programma `tm3` op mijn computer opstart blijft het apparaat bits genereren tot ik het proces afbreek.

Manier (iv) van Jan Willem gebruikt de volgende functie ϵ:

$$\begin{aligned} \epsilon_0 &= 1 \\ \epsilon_{2n} &= \epsilon_n \\ \epsilon_{2n+1} &= 1 - \epsilon_n \end{aligned}$$

Hier is de Haskell versie:

```
epsilon 0 = 1
epsilon n | even n = epsilon (div n 2)
          | odd  n = 1 - epsilon (div (n-1) 2)
```

Wat is nu de bijbehorende functie voor M'? Simpelweg de functie ϵ' die je krijgt uit ϵ door de beginwaarde te swappen:

$$\begin{aligned} \epsilon'_0 &= 0 \\ \epsilon'_{2n} &= \epsilon'_n \\ \epsilon'_{2n+1} &= 1 - \epsilon'_n \end{aligned}$$

De implementatie wordt:

```
epsilon' 0 = 0
epsilon' n | even n = epsilon' (div n 2)
           | odd  n = 1 - epsilon' (div (n-1) 2)
```

Dit geeft:

```
Main> map epsilon [0 .. 100]
[1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1,0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,0,1,
1,0,1,0,0,1,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1,0,1,1,0,
1,0,0,1,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1,1,0,0,1,0]
Main> map epsilon' [0 .. 100]
[0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1,1,0,
0,1,0,1,1,0,0,1,1,0,1,0,0,1,0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,1,0,0,1,
0,1,1,0,0,1,1,0,1,0,0,1,0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,0,1,1,0,1]
```

Een en ander leidt tot de volgende programma's voor M en M':

```
tm4 = map (intToDigit . epsilon) [0..]

tm4' = map (intToDigit . epsilon') [0..]
```

Overigens kan dit nog worden omgezet in een expliciete corecursie (vergelijk de opmerking van Jan Willem dat coinductie technieken hetzelfde kunnen klaarspelen als infinitair herschrijven):

```
h n = (intToDigit . epsilon) n : (h (n+1))

tm4a = h 0
```

Ik voeg er hier nog een manier aan toe die het idee van corecursie in (i) combineert met het expansie idee uit (ii). De functie `expand` voor het expanderen van strings nullen en enen is boven gegeven. De Thue Morse reeks M is het resultaat van 1 opschrijven, gevolgd door genereren van een string uit het zaadje "0". Genereren uit een zaadje `xs` geschiedt door dat zaadje op te schrijven, en het te laten volgen door de string die het resultaat is van genereren met zaadje `expand xs`. We krijgen dan het volgende programma:

```
grow xs = xs ++ grow (expand xs)

tm5 = '1': grow "0"
```

Als je de functie `grow` vergelijkt met de functie `thue`, dan zie je dat hier gebruik is gemaakt van het feit dat voor elke eindige string `xs` geldt dat `expand xs` gelijk is aan `xs ++ (swap xs)`.

Maar het kan nog anders. Jan Willem geeft in paragraaf 2 van zijn paper wat hij een *self similar property* van M noemt. Dit valt direct in Haskell te programmeren, en het leidt tot ons zesde programma voor M.

```
tm6 = '1' : '0':  alternate (tail tm6) (tail (swap tm6))

alternate (x:xs) (y:ys) = x: y: alternate xs ys
```

De code voor het in elkaar vlechten van twee oneindige lijsten kan nog iets compacter:

```
alt (x:xs) ys = x: alt ys xs
```

En het hele programma kan nog wat verder worden gestileerd:

```
tm7  = '1' : tm7a
tm7a = '0' : alt tm7a (swap tm7a)
```

Dit is in feite het systeem van bewaakte recursievergelijkingen (*guarded recursive equations*) dat Jan Willem geeft in paragraaf 3 van zijn paper.

De corecursieve programmas `tm1`, `tm2`, `tm5`, `tm6` en `tm7` hebben last van wat bij functioneel programmeren een *space leak* wordt genoemd. Ze maken gebruik van string concatenatie en/of

string parameters, en de lengte van de strings die steeds als parameter worden meegegeven groeit exponentieel. Vandaar dat deze programma's er na enige tijd wegens geheugengebrek de brui aan geven. Na enige honderden regels nullen en enen zie je:

```
Main> tm1
"01101001100101101001011001l0 ...
...
00110100110010110100101100ll
ERROR - Garbage collection fails to reclaim sufficient space
```

Hetzelfde gebeurt met de andere genoemde programma's. Dit euvel is inherent aan de definities, die immers een parameter gebruiken waarvan het geheugenbeslag exponentieel groeit. De definities `tm3` en `tm4` kennen dit probleem niet.

Proces specificaties van M en M'

Maar terug naar de definities en implementaties van M. Wanneer we bedenken dat M' het resultaat is van swappen van de nullen en enen in M, dan zien we dat we M en M' zeer elegant kunnen definiëren met behulp van simultane corecursie, als volgt:

```
m  = '1' : n
m' = '0' : k
n  = '0' : alt n k
k  = '1' : alt k n
```

Dit leidt direct tot een mooie proces-specificatie voor M en M':

$$\begin{aligned} M &= 1 \cdot N \\ M' &= 0 \cdot K \\ N &= 0 \cdot (N \,[]\, K) \\ K &= 1 \cdot (K \,[]\, N) \\ (b \cdot X) \,[]\, Y &= b \cdot (Y \,[]\, X) \end{aligned}$$

Hierbij staat b voor een willekeurige bit en X, Y voor een willekeurig proces.

Terzijde zij hier opgemerkt dat Jan Willem's exercitie om M met behulp van hernoemen en communicatie te definiëren me niet echt overtuigd heeft. Intuitief is de $[]$ operatie op processen immers gemakkelijk met behulp van procesvergelijkingen te specificeren. De interactie tussen $[]$ en $\cdot, +$ wordt dan gegeven door de volgende vergelijkingen (waarvan de eerste twee overbodig zijn als we $[]$ alleen voor oneindige processen definiëren):

$$\begin{aligned} v \,[]\, X &= v \cdot X \\ (v \cdot X) \,[]\, w &= v \cdot w \cdot X \\ (v \cdot X) \,[]\, Y &= v \cdot (Y \,[]\, X) \\ (X + Y) \,[]\, Z &= X \,[]\, Z + Y \,[]\, Z \end{aligned}$$

Ik ben benieuwd wat Jan Willem hiervan vindt. In elk geval denk ik dat het antwoord op zijn vraag 'Kan M worden gespecificeerd in procesalgebra met bewaakte recursievergelijkingen, maar zonder hernoemen?' bevestigend luidt.

De Toeplitz reeks T

In paragraaf 4 van zijn paper bespreekt Jan Willem de Toeplitz reeks T die je krijgt door de verschilreeks te nemen van M. De verschilreeks maken voor een binaire stroom gaat in Haskell als volgt:

```
difseq (x:y:zs) = dif y x : difseq (y:zs)
  where dif '0' '1' = '1'
        dif '1' '0' = '1'
        dif  _   _  = '0'
```

Merk hierbij op dat dit in feite verschilrijen oplevert modulo 2. Wanneer we niet modulo 2 rekenen krijgen we immers $0 - 1 = -1$ versus $1 - 0 = 1$.

De verschilrij van M' nemen modulo 3 levert een kwadraat-vrije (*square-free*) reeks op: er komen geen substrings van de vorm WW in voor. De verschil-operatie wordt nu:

```
difseq' (x:y:zs) = dif y x : difseq' (y:zs)
  where dif '0' '1' = '2'
        dif '1' '0' = '1'
        dif  _   _  = '0'
```

Dit geeft:

$$\begin{aligned} M' &= 01101001100101101001011001101001100101100110100101\ldots \\ \Delta M' \pmod 3 &= 10212010201210212012102010212010201210201021201210\ldots \end{aligned}$$

De reeks $\Delta M' \pmod 3$ is een transliteratie van de reeks

$$\phi M' = 02101202120102101201021202101202120102120210120102\ldots$$

die je krijgt door het morfisme $01 \mapsto 0,10 \mapsto 1,00 \mapsto 2,11 \mapsto 2$ los te laten op M. De implementatie:

```
phi ('0':'1':xs) = '0' : phi ('1':xs)
phi ('1':'0':xs) = '1' : phi ('0':xs)
phi ('0':'0':xs) = '2' : phi ('0':xs)
phi ('1':'1':xs) = '2' : phi ('1':xs)
```

Dit geeft:

```
Main> take 50 (phi m')
"02101202120102101201021202101202120102120210120102"
```

Die reeks is kwadraat-vrij, dus elke transliteratie ervan is dat ook. De transliteratie t die we nodig hebben om uit de verschilrij $\Delta M'$ (mod 3) de rij $\phi M'$ te krijgen waarvan Morse en Hedlund bewijzen dat ze kwadraat-vrij is, is $t = \lambda x(x + 2 \pmod 3)$. Overigens volgt hieruit meteen dat de reeksen ϕM en ΔM (mod 3) ook kwadraat-vrij zijn. Het is gemakkelijk te zien dat ϕM kan worden verkregen uit $\phi M'$ door verwisselen van nullen en enen.

De verschilreeks-definitie van T leidt tot het volgende programma voor T:

```
t1 = difseq m
```

Hiermee krijgen we:

```
Main> take 60 t1
"101110101011101110111010101110101011101010111011101110101011"
```

Uiteraard zijn de verschilreeksen modulo 2 die je krijgt uit M en M' identiek. Laat T' het resultaat zijn van omwisselen van de nullen en enen in T.

De verschilreeksen modulo 3 uit M en M' zijn niet identiek, maar het zijn transliteraties van elkaar: ze zijn in elkaar over te voeren door een swap van tweeën en enen.

De reeks T kan worden beschouwd als het resultaat van het toepassen van een alternatieve expansiefunctie:

```
expand2 ""       = ""
expand2 ('0':xs) = '1':'1': expand2 xs
expand2 ('1':xs) = '1':'0': expand2 xs
```

We hebben nu drie verschillende morfismen geïmplementeerd. Het wordt de moeite waard om een generatie-functie te definiëren met parameters voor het gebruikte morfisme en voor de beginwaarde:

```
generate :: (String -> String) -> String -> Int -> String
generate m init 0 = init
generate m init (n+1) = m (generate m init n)
```

Nu kun je

- M benaderen met behulp van `generate expand "1"`,
- M' benaderen met behulp van `generate expand "0"`,
- T benaderen met behulp van `generate expand2 "1"`.

Jan Willem merkt op dat de recursieve definities `toeplitz0`, `toeplitz1` en `toeplitz2` in de limiet dezelfde stroom opleveren, i.e., je kunt T ook benaderen met `generate expand2 "0"`.

```
toeplitz0 = generate expand2 "0"
toeplitz1 = generate expand2 "1"

toeplitz2 0 = "1"
toeplitz2 (n+1) | even n    = toeplitz2 n ++  "0" ++  toeplitz2 n
                | otherwise = toeplitz2 n ++  "1" ++  toeplitz2 n
```

Een proces specificatie van T

De stroom M heeft de eigenschap dat teken $M(2n)$ steeds verschilt van teken $M(2n+1)$ (wanneer we het eerste teken van de reeks aanduiden met $M(0)$). Dit volgt direct uit definitie (iv) van M. Dat betekent dat de verschilreeks modulo 2 voor M op alle *even* plaatsen een 1 heeft (wanneer we tellen vanaf 0).

M heeft ook de eigenschap dat $M(2n+2) - M(2n+1) = 1 - (M(n+1) - M(n))$. Dit kun je bij voorbeeld halen uit de definitie van M met behulp van ϵ. Maar dat wil zeggen dat $T(2n+1) = 1 - T(n)$. En dat wil weer zeggen dat we, door de oneven posities uit de verschilreeks T te nemen, de reeks T' krijgen.

Wat hieruit volgt is dat T ontstaat door om en om een 1 en een element van T' te nemen. Dit geeft het volgende corecursieve programma voor T:

```
t2 = alt ones (swap t2) where ones = '1' : ones
```

Weer kunnen we dit zonder de swap doen, door T and T' met simultane corecursie te definiëren:

```
t  = alt ones t' where ones = '1' : ones
t' = alt zeros t where zeros = '0' : zeros
```

Het belang van deze simultane corecursie is dat het ons een fraaie proces-specificatie oplevert voor T en T':

$$\begin{aligned} O &= 1 \cdot O \\ Z &= 0 \cdot Z \\ T &= 1 \cdot (T' [] O) \\ T' &= 0 \cdot (T [] Z) \\ (b \cdot X) [] Y &= b \cdot (Y [] X) \end{aligned}$$

Hierbij staat b weer voor een willekeurige bit en X, Y voor een willekeurig proces.

Recursievergelijkingen voor de elementen van T

We kunnen nu een volgende vraag van Jan Willem beantwoorden. Wat zijn de recursievergelijkingen voor de functie τ die de elementen van T levert?

We maken gebruik van de functie ϵ voor M (hierboven geïmplementeerd als `epsilon`), en van rekenen modulo 2. Uit de definitie van T plus de definitie van ϵ krijgen we met rekenen modulo 2:

$$\begin{aligned} \tau(2n) &= \epsilon(2n+1) - \epsilon(2n) \pmod 2 \\ &= (1 - \epsilon(n)) - \epsilon(n) \pmod 2 \\ &= 1 \\ \tau(2n+1) &= \epsilon(2n+2) - \epsilon(2n+1) \pmod 2 \\ &= \epsilon(n+1) - (1 - \epsilon(n)) \pmod 2 \\ &= 1 - (\epsilon(n+1) - \epsilon(n)) \pmod 2 \\ &= 1 - \tau(n) \end{aligned}$$

Dit geeft ook direct antwoord op een andere vraag van Jan Willem. Alle staarten $t^n(T)$ zijn inderdaad verschillend. De reeks T is *niet* uiteindelijk periodiek. Hier is de implementatie van τ:

```
tau n | even n = 1
      | odd n  = 1 - tau (div (n-1) 2)
```

Dit leidt tot de volgende alternatieve implementatie van T:

```
t3 = map (intToDigit . tau) [0..]
```

Uiteraard vormen de recursievergelijkingen voor de functie τ' die T' oplevert het spiegelbeeld van die voor de functie voor T:

$$\begin{aligned} \tau'(2n) &= 0 \\ \tau'(2n+1) &= 1 - \tau'(n) \end{aligned}$$

Dit geeft:

```
tau' n | even n = 0
       | odd n  = 1 - tau' (div (n-1) 2)

t3' = map (intToDigit . tau') [0..]
```

Genererende functies voor M, M', T, T'

Een manier om oneindige reeksen te bestuderen die al sinds jaar en dag zeer vruchtbaar is gebleken is de volgende. Beschouw de elementen uit de reeks als de coëfficiënten $[a_0, a_1, a_2, ..]$ van een machtreeks voor een genererende functie $g(z)$:

$$g(z) = a_0 + a_1 z + a_2 z^2 + a_3 z^3 + \cdots$$

Genererende functies werden al bestudeerd door Pascal, die aantoonde dat $(a+b)^n$ een genererende functie is voor het aantal combinaties van n dingen met k tegelijk genomen.

```
module Galois2

where

data Galois2 = Zero | One deriving (Eq, Ord, Enum, Read, Bounded)

instance Show Galois2
  where show Zero = "0"
        show One  = "1"

instance Num Galois2
  where
  Zero + One    = One
  One  + Zero   = One
  _    + _      = Zero
  One  * One    = One
  _    * _      = Zero
  negate        = id
  abs           = id
  signum        = id
  fromInteger n | even n = Zero
                | odd n  = One
```

Figuur 1: Een module voor GF(2).

In ons geval dienen we te rekenen modulo 2, hetgeen wil zeggen dat we de genererende functies beschouwen over het eindige lichaam (of Galois lichaam) GF(2). Dit is het lichaam van restklassen modulo 2, dus het heeft alleen de elementen 0 en 1, met optellen en vermenigvuldigen modulo 2.

Optellen modulo 2 komt neer op toepassen van de logische XOR functie, terwijl vermenigvuldigen modulo 2 hetzelfde is als de logische AND. Vermenigvuldigen van een bit met -1, modulo 2, verandert niets, want $1 \equiv -1 \pmod 2$, dus rekenkundige negatie is *niet* hetzelfde als logische negatie. Mijn bedoeling is om genererende functies voor M, M', T, en T' af te leiden die kunnen worden geïllustreerd met een implementatie. Daarbij heb ik een module nodig voor het Galois lichaam GF(2): zie Figuur 1. De bewerkingen op machtreeksen die we nodig hebben zijn optellen en vermenigvuldigen modulo 2. De module in Figuur 2 implementeert optellen en vermenigvuldigen op polynomen en machtreeksen.[4] Optellen van machtreeksen komt neer op coëfficiëntsgewijs optellen, vermenigvuldigen van machtreeksen is convolutie van de coëfficiënten: het n-de lid van het product van $[a_0, a_1, a_2, \ldots]$ en $[b_0, b_1, b_2, \ldots]$ wordt gegeven door $\Sigma_{k=0}^{n} a_k b_{n-k}$. Door nu te kijken naar machtreeksen over het Galois lichaam kunnen we rekenen modulo 2 met

[4] Nadere toelichting in het hoofdstuk over corecursie in ons eerder genoemde boek.

```
module Polynomials where

z :: Num a => [a]
z = [0,1]

infixl 7 .*
(.*) :: Num a => a -> [a] -> [a]
c .* []     = []
c .* (f:fs) = c*f : c .* fs

instance Num a => Num [a] where
  fromInteger c   = [fromInteger c]
  negate []       = []
  negate (f:fs)   = (negate f) : (negate fs)
  fs     + []     = fs
  []     + gs     = gs
  (f:fs) + (g:gs) = f+g : fs+gs
  fs     * []     = []
  []     * gs     = []
  (f:fs) * (g:gs) = f*g : (f .* gs + fs * (g:gs))
```

Figuur 2: Een module voor Polynoomreeksen.

machtreeksen implementeren.

We zullen nu zien dat de operatie [] van 'om en om nemen' in GF(2) een zeer natuurlijke interpretatie heeft als een operatie op machtreeksen modulo 2. Merk allereerst op dat kwadrateren modulo 2 gaat volgens de regel $(A+B)^2 = A^2 + B^2$. Immers, de term met $2AB$ valt weg omdat $2 \equiv 0 \pmod 2$. Maar dat betekent dat kwadrateren van een polynoomreeks modulo 2 als volgt gaat:[5]

$$(a_0 + a_1 z + a_2 z^2 + a_3 z^3 + \cdots)^2 = a_0 + a_1 z^2 + a_2 z^4 + a_3 z^6 + \cdots \pmod 2.$$

Met andere woorden: de coëfficiëntenreeks $[a_0, a_1, a_2, a_3, ..]$ wordt omgezet in de reeks

$$[a_0, 0, a_1, 0, a_2, 0, a_3, 0, ..].$$

We kunnen dit illustreren aan de implementatie (in het onderstaande werk ik met een systeem waarin modules Galois2 en Polynomials geladen zijn):

```
TM> [One,One,One,One,One,One]
[1,1,1,1,1,1]
```

[5] Zie ook `http://mathworld.wolfram.com/Thue-MorseSequence.html`.

```
TM> [One,One,One,One,One,One]^2
[1,0,1,0,1,0,1,0,1,0,1]
```

Verschuiven van een polynoomreeks over een positie naar rechts, met invoegen van een 0 aan het begin, geschiedt door vermenigvuldigen met z. Immers:

$$z \cdot (a_0 + a_1 z + a_2 z^2 + a_3 z^3 + \cdots) = 0 + a_0 z + a_1 z^2 + a_2 z^3 + a_3 z^4 + \cdots$$

Een en ander houdt in dat we vervlechten van reeksen A en B over GF(2) (het proces $A[]B$) kunnen modelleren als:

$$A^2 + zB^2.$$

Hier is een voorbeeld:

```
TM> [One,One,Zero,One]^2 + z * [Zero,Zero,One,Zero]^2
[1,0,1,0,0,1,1,0]
```

Met behulp hiervan kunnen we de procesdefinitie voor M en M' nu omzetten in het volgende simultane stelsel van genererende functies:

$$\begin{aligned} M(z) &= 1 + z \cdot N(z) \pmod 2 \\ M'(z) &= z \cdot K(z) \pmod 2 \\ N(z) &= z \cdot (N^2(z) + z \cdot K^2(z)) \pmod 2 \\ K(z) &= 1 + z \cdot (K^2(z) + z \cdot N^2(z)) \pmod 2 \end{aligned}$$

Hetzelfde kunnen we doen voor T en T'. Hierbij moet bedacht worden dat $\frac{1}{1-z}$ de genererende functie is voor de rij $[1, 1, 1, \cdots]$. Om en om nemen van de reeks $\frac{1}{1-z} = [1, 1, 1, \ldots]$ en de reeks $T'(z)$ geschiedt door $(\frac{1}{1-z})^2$ en $z \cdot T'^2(z)$ bij elkaar op te tellen, alles modulo 2. Om en om nemen van de reeks $[0, 0, 0, \cdots]$ en de reeks T geschiedt door $[0, 0, 0, \cdots]$ en $z \cdot T^2$ bij elkaar op te tellen, maar het resultaat hiervan is weer $z \cdot T^2$. We krijgen dus:

$$\begin{aligned} T(z) &= \left(\frac{1}{1-z}\right)^2 + z \cdot T'^2(z) \pmod 2 \\ T'(z) &= z \cdot T^2(z) \pmod 2. \end{aligned}$$

Invullen van de formule voor $T'(z)$ in die voor $T(z)$ en vereenvoudigen met behulp van $(1-z)^2 = 1 - z^2 \pmod 2$ geeft de volgende formule voor de genererende functie van T:

$$T(z) = \frac{1}{1-z^2} + z^3 \cdot T^4(z) \pmod 2.$$

Ter afsluiting geef ik de bijbehorende implementaties van de genererende functies voor M, M', T en T', in de vorm van een module die de modules voor het Galois lichaam en voor rekenen met machtreeksen importeert:

```
module TM where

import Galois2
import Polynomials

m  = One  : n
m' = Zero : k
n  = Zero : (n^2 + (Zero: k^2))
k  = One  : (k^2 + (Zero: n^2))

t  = ones^2 + (Zero : Zero : Zero : t^4) where ones = One : ones
t' = Zero : t^2
```

Dit geeft inderdaad weer de goede reeksen:

```
TM> take 32 m
[1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1,0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0]
TM> take 32 m'
[0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1]
TM> take 32 t
[1,0,1,1,1,0,1,0,1,0,1,1,1,0,1,1,1,0,1,1,1,0,1,0,1,0,1,1,1,0,1,0]
TM> take 32 t'
[0,1,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,1,0,1,0,0,0,1,0,1]
```

Omdat vermenigvuldiging van een genererende functie met $\frac{1-z}{z}$ een genererende functie voor de verschilrij oplevert, kan T ook worden gekarakteriseerd als $\frac{1-z}{z}M(z)$. Het effect van delen door z is dat de coëfficiëntenrij een positie naar links verschuift. T wordt dus gegenereerd als de staart van $(1-z)M(z)$:

```
TM> take 32 (tail ((1-z) * m))
[1,0,1,1,1,0,1,0,1,0,1,1,1,0,1,1,1,0,1,1,1,0,1,0,1,0,1,1,1,0,1,0]
```

Een paar open vragen over $\Delta M \pmod 3$

Laat H de verschilrij $\Delta M \pmod 3$ zijn. Kunnen we nu ook recursievergelijkingen geven voor de elementen van H? Een definitie van een functie h voor de elementen van H in termen van ϵ valt gemakkelijk te leveren:

$$\begin{aligned} h(2n) &= \epsilon(2n+1) - \epsilon(2n) \pmod 3 \\ &= (1-\epsilon(n)) - \epsilon(n) \pmod 3 \\ &= 1 - 2\epsilon(n) \pmod 3 \\ &= 1 + \epsilon(n) \end{aligned}$$

$$\begin{aligned} h(2n+1) &= \epsilon(2n+2) - \epsilon(2n+1) \pmod 3 \\ &= \epsilon(n+1) - (1 - \epsilon(n) \pmod 3 \\ &= 2 + \epsilon(n+1) + \epsilon(n) \pmod 3 \end{aligned}$$

Dit leidt tot de volgende implementatie:

```
ha n | even n = (1 + epsilon (div n 2))
     | odd  n = (2 + (epsilon (m+1)) + (epsilon m)) `mod` 3
        where m = (div (n-1) 2)
```

Het is mij niet bekend of h ook rechtstreeks te definiëren valt, dus zonder gebruik te maken van een functie voor M.

Een andere (voor mij) open vraag is die naar formule voor een genererende functie voor H. Zoals boven reeds vermeld kan een genererende functie voor een verschilrij ΔA in het algemeen worden verkregen uit een genererende functie $g(z)$ voor A door vermenigvuldiging van $g(z)$ met $\frac{1-z}{z}$. Voor een genererende functie voor H dienen we te rekenen over het Galois lichaam GF(3), het lichaam van restklassen modulo 3. Dus, als we een genererende functie $M(z)$ zouden hebben voor M over dit lichaam, dan zou de genererende functie voor H er als volgt kunnen uitzien:

$$\begin{aligned} H(z) &= \frac{1-z}{z} M(z) \pmod 3 \\ &= \frac{1+2z}{z} M(z) \pmod 3. \end{aligned}$$

Het is me echter niet gelukt een genererende functie voor M over GF(3) te vinden. De moeilijkheid is dat de truc om $[a_0, a_1, a_2, \ldots]$ te transformeren in $[a_0, 0, a_1, 0, a_2, 0, \ldots]$ met behulp van kwadrateren alleen werkt in GF(2). In GF(3) hebben we: $(A+B)^2 = A^2 + 2AB + B^2$, en dit is in het algemeen niet gelijk aan $A^2 + B^2$.

Tot slot

Beste Jaco, ik ben aan het eind gekomen van mijn wetenschappelijke beschouwing. Rest een persoonlijk slot. Mijn jaren hier op het CWI kunnen worden onderscheiden in twee perioden: het pre-kantelingstijdvak, waarin ik ressorteerde in een cluster (toen nog 'afdeling') onder jouw bewind, en een periode daarna in het gekantelde tijdperk, waarin ik los van jou door het instituut gezweefd heb. Die tweede periode als vrij atoom heeft me de periode daarvoor retrospectief meer doen waarderen. Het is wijsheid achteraf, maar ik zie nu dat je indertijd buitengewoon goed voor je groepsleiders hebt gezorgd. Mijn besluit om je afdeling te verlaten was in het licht van de omstandigheden op dat moment wellicht onvermijdelijk, maar een wijzer en bezonnener iemand dan ik toen was had die stap misschien niet genomen. Wie zal het zeggen? Weet in elk geval dat ik met grote waardering afscheid van je neem. Het ga je goed.

Met vriendelijke groet,

Jan van Eijck

does wisdom come with the years?

A. Eliëns

2/4/2002

Does wisdom come with the years? Or does it disappear? Sitting across Jaco in the Theeboom, having dinner with the Informatica Visitation Committee, I heard him plea for an Interfaculty of Computer Science, away from Mathematics, away from the exact sciences. Appealing, but surprising. Surprising to hear this from Jaco. But perhaps at this stage of his career ...

Let me go back to the beginning. 1984. Not the year of doom, but the year my first child was born. Out of shere boredom with the offerings at the other university, I decided to take a course on the mathematical theory of program correctness. Indeed, Jaco's course. Old-fashioned style, reading from the book, yawning. Occassionally he made a mistake, which I kindly corrected, also yawning. Not in the intimidating way, but just because of a lack of sleep, due to the restlessness of my new-born child.

Never good at dress-codes, I must have been an odd figure in the class. But Jaco didn't care and invited me to do a thesis with him. This was the time you were selected to do a thesis. So, about one and a half year later I joined Jaco's 'little family' at CWI, with Jan, Joost, Frank and Erik. Then the meaning, or rather effect, of his yawning became more apparent. It gave many people the feeling that what they had to say was not interesting or worthwhile (personal communication). Of course, that may be so. However, still lacking sleep, I simply joined in yawning, which is after all an infecting gesture.

My residence at CWI continued, accepting the offer to work as a research assistent. However, I was in a different project. Altogether different, for that matter. But I became again in close contact with the 'family' when I was considered to be too controversial by representatives of our national bulb industry, and Jaco stood up for me, recognizing my work as valuable, and accepting me as a step-child in his 'family'.

Finishing my thesis, it was again Jaco who ordered me to go after a job at the university where I am currently working. There we met time and again at a variety of occassions, the so-called 'vakgroep' meetings, but also as members of a 'team of wise men' (imagine) to select the next dean.

How is life at an institution? Jaco knows better than me. I still remember his 25th professional anniversary, when I was completing my thesis. He did his 'march through the institutions'. By now, I have diverged a great deal, far away from the family tree, but nevertheless grateful for the support Jaco gave me exactly at the time I needed it.

Jaco is saying goodbye. Not finishing a chapter. It is more like closing a book.

De laatste Opponent

Discipuli, Spectate Picturam.....

Ooit vormde deze zin het eerste stukje Latijn in het boek dat wij op het Lyceum gebruikten. Laten wij de boodschap echter toepassen en de afbeelding bekijken. De foto, een originele Maria Austria opname, vervaardigd op 18 September 1974 bij mijn promotie, toont het college van opponenten. En wat wij zien geeft aanleiding tot treurnis.

Vrijwel alle afgebeelden behoren inmiddels tot de categorieen geëmeriteerd, dood, of beide. Allemaal, op degene na die thans het gezelschap compleet maakt. De emeritus van heden.

De moraal? Welkom in dit illustere gezelschap ? Aldus eindigt de discussie ? We are victorious !!? Of het besef dat deze valbijl evenzeer zweeft boven de hals van de defendens op de voorgrond, die ook nog hoogstens acht jaar te gaan heeft.

Sic Transit Gloria

Peter van Emde Boas
ILLC-FNWI-UvA
Bronstee.com Software & Services B.V.
20020515

Prof. De Bakker, or How I Learned to Stop Worrying and Love the Boss

Wan Fokkink

The name De Bakker came to my ears for the first time as a student mathematics, during topology courses. One of many names of people that were apparently shaping mathematics together, somewhere on Mount Olympus.

February 1991 I started as a PhD student at CWI, and Jaco became a person of flesh and blood. But he remained a distant figure, as the boss of my boss. At that time the financial situation of CWI was not so rosy, and every month, at the end of an AP (Afdeling Programmatuur) scientific meeting, Jaco would remind us of this fact. With his sonorous voice he would recite the financial losses and reductions in personnel at AP. Depressing stuff.

January 1997 I moved to Swansea in Wales, where the head of department John Tucker turned out to be a very good friend of Jaco. They knew each other from a time when John worked at CWI. Jaco and his wife had once visited the Tucker family, an episode which had found its way into the Book of Legendary Visits. John used to say "I am president of the Jaco de Bakker fan club" (after a short visit to The Netherlands he once jokingly added "and possibly the only member...").

From September 2000 onward, as theme leader at CWI, I started to share John's views and learned to fully appreciate Jaco. As my direct boss he was always ready to give full support and sound advice, based on his wide experience and exceptionally good memory. His guidance prevented me from several pitfalls and stimulated me to venture in new directions. Moreover, I found that he is truly interested in personal matters and family affairs. By now for me Jaco is one of Four Fatherly Figures, key persons in my scientific career and development.

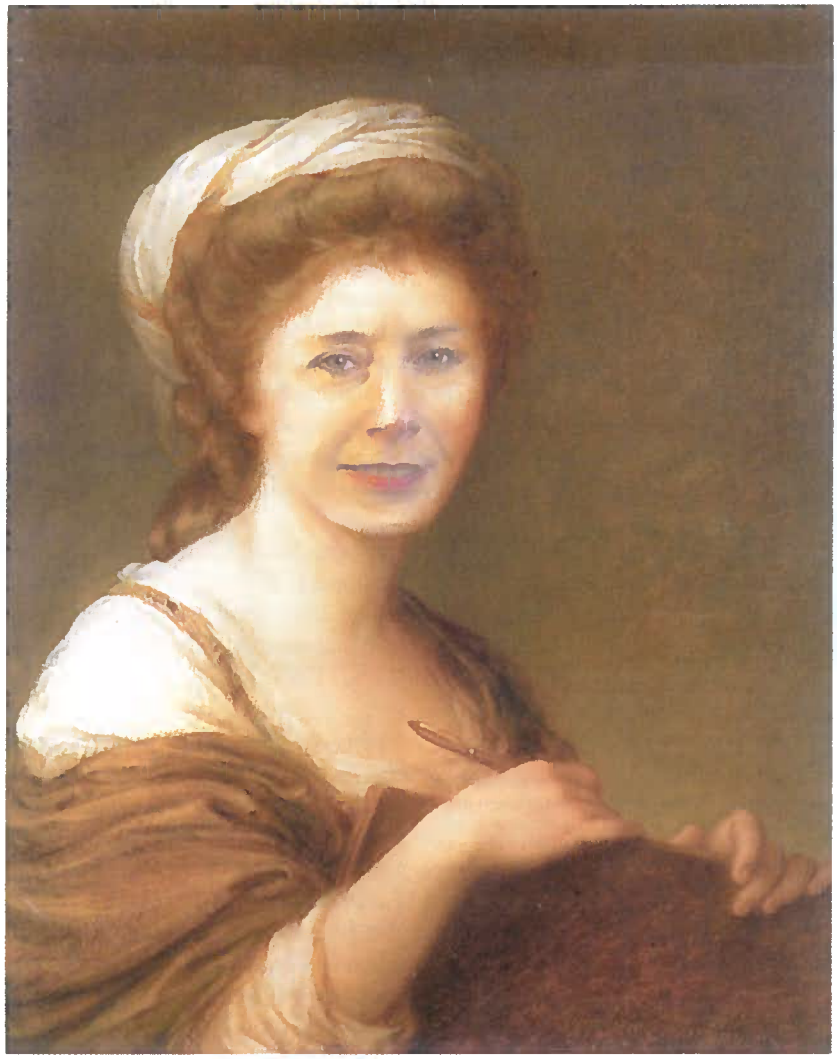
De 'Cicerone' van Rome

Over een bijdrage voor jouw Liber Amicorum moest ik wel nadenken. Waarover zou ik moeten schrijven ? We hadden vanuit onze functies niet zo vaak contact.

Toch is er wel een aantal momenten in de afgelopen tien jaar geweest die een herinnering hebben achtergelaten. Het begon wat mij betreft in mei 1992 toen er bij het CWI, na de reorganisatie in 1991, voor het eerst een facilitair manager werd aangesteld. Er werden toen nog Instituutsraad-vergaderingen gehouden, waarbij wetenschap en ondersteuning vertegenwoordigd waren. Daarbij hoorde ook één IR-dag per jaar bij. Van die beginperiode kan ik me herinneren dat tijdens mijn eerste IR-dag voor mij de wondere wereld van de bètawetenschappers openging. Die eerste dag in 1992 was jij volgens de verslagen niet present, de jaren daarna wel. Ik kan me herinneren dat je op enig moment tijdens die eerste IR-dag waar wij beiden aanwezig waren ('de Wakende Haan' te Abcoude 1993), voorzichtig vroeg of ik zo'n dag wel interessant vond. Je was verbaasd dat ik wel degelijk geïnteresseerd was in het wel en wee van het instituut, ook op het gebied van de wetenschap. Hoewel naar mijn smaak wetenschap alleen niet zaligmakend is voor ons menselijk bestaan, is deze naast politiek en cultuur een belangrijk en interessant onderdeel van onze samenleving.
Inmiddels zijn in de afgelopen jaren de IR-vergaderingen niet alleen verminderd, maar uiteindelijk geruisloos verdwenen en daarmee ook de meeste momenten dat wij elkaar even spraken. Vergaderingen met de combinatie ondersteuning en wetenschap bleken geen succes om de juiste communicatie tussen die twee zo verschillende groepen te krijgen.

Maar nog even terug in de tijd. De publicatiedienst, toen nog onder mijn leiding, onderhield veel contacten met de wetenschappers. Rapporten die niet op tijd verschenen, boeken die uit elkaar vielen, deadlines die altijd werden overschreden en het voortbestaan van de publicatiedienst waren aanleiding tot het instandhouden van onze communicatie. Verder spraken we over een voor het CWI niet onbelangrijk onderwerp: de huisvestingsproblematiek.

Het verbouwen van ons pand en het herhuisvesten van alle medewerkers ten tijde van de 'kanteling' was een periode van veel en behoedzaam overleg. Mijn facilitaire insteek aangaande opruimacties, ook in de werkkamers van wetenschappers, is mij nooit in dank afgenomen, maar was vaak onvermijdelijk om weer ruimte in ons pand vrij te maken. Ons meest recente werkcontact was naar aanleiding van de rol van, 'portefeuillehouder huisvesting', vanuit je MT-lidmaatschap.
Een taak die nooit goed omschreven is, maar die als doel heeft brandjes te blussen als er weer eens vierkante meters tekort zijn. De strijd om vierkante meters meer of minder blijft actueel in ons instituut. Omdat alles in 2001 wel goed liep hebben we sporadisch contact gehad tot aan het begin van dit jaar ook die rol is neergelegd.

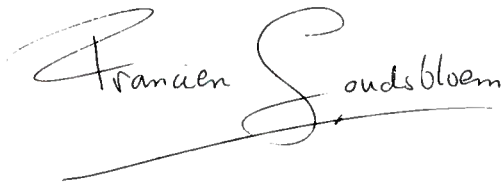


'Cicerone' van Rome

Hoewel we een totaal andere natuur en een geheel andere werkomgeving hebben, zijn de contacten altijd goed geweest en waren deze niet gespeend van humor. Dat het tempo van mijn spraakwaterval niet altijd met de weloverwogen rustige spreekstijl van jou strookt, heeft geen problemen opgeleverd. Zelf vind ik het jammer dat in de afgelopen twee jaar ons contact tot een minimum beperkt is gebleven. We hebben bijvoorbeeld nooit eerder ontdekt dat wij beiden van 18e-eeuwse schilderkunst houden. Er zijn nauwelijks gelegenheden geweest om even bij te praten; dat zal wel horen bij deze hectische tijd. Dus in hoeverre nemen wij afscheid van elkaar ?

Je zult ongetwijfeld contact blijven houden met je wetenschappelijke collega's waarmee de banden veel sterker zijn. Met oud-collega's in de ondersteuning zal het anders zijn en daarom heb ik een andere, ludieke herinnering bedacht.

In 1784 schilderde Angelika Kauffmann dit zelfportret volgens het type van Rafaël en andere Italiaanse renaissanceschilders. Zij was tijdens haar leven een heel succesvolle schilderes met groot talent. Zij wordt omschreven als een zeer welbespraakte gids voor vreemden en krijgt de bijnaam 'Cicerone van Rome'. Met computertechniek is veel mogelijk, dus als je goed kijkt naar dit portret wordt herinnering geactiveerd. Als je dit boek, genietend van je pensioen en in een bui van nostalgie, nog eens doorbladert, zal het geheugen vanzelf zijn werk doen.



Bits, trits, qits, pits of hits

Jan Friso Groote

Faculteit Wiskunde en Informatica, TU/e, Postbus 513, 5600 MB Eindhoven

CWI, Postbus 94079, 1090 GB Amsterdam

Email: J.F.Groote@tue.nl

Het is ruim 14 jaar geleden dat ik op het CWI aankwam. Ongebruikelijk, vanuit Twente, in de hoop de informatica op een wiskundiger manier te leren kennen. Het toeval leidde mij naar de groep AP2, binnen de afdeling programmatuur (AP) die geleid werd door Jaco de Bakker. In de eerste periode van ongeveer 4 jaar op het CWI, had ik, en menig promovendus met mij, het gevoel weinig met Jaco van doen te hebben. Alleen tijdens de maandelijkse AP bijeenkomsten, waar Jaco tevens de wetenswaardigheden over het CWI samenvatte, hadden we met elkaar te maken. En als ik op reis wilde. Dan mocht ik 1 minuut bij Jaco komen, waarbij hij steevast antwoordde dat ik natuurlijk op reis kon, mits ik het geld elders wist te regelen.

Mijn waardering voor Jaco kwam pas toen ik na het CWI bij de faculteit Wijsbegeerte in Utrecht ging werken. Daar maakte ik veel van de bestuurlijke discussies en andere onverkwikkelijkheden mee. Ik realiseerde me hoe heerlijk het is mij op mijn onderzoek te kunnen richten, zonder ook maar een vermoeden te hebben van de bestuurlijke stormen die er woeden. En die stormen waren er indertijd ook op het CWI; een reorganisatie met naar ik meen gedwongen ontslagen.

Na 4 jaar Utrecht vroeg Jaco of ik terug wilde naar het CWI, nu als groepsleider van AP2 (later SEN2). Als groepsleider kan de buitenwereld niet buitengesloten worden. Maar ook hier opereerde Jaco op uitstekende wijze, te kenmerken door twee woorden: abstractie en betrouwbaarheid. Hij sloeg gade wat er in de buitenwereld gebeurde en gaf daarvan door wat relevant was. Een uiterst efficiënte manier van werken. Jaco's betrouwbaarheid zorgde ervoor dat er nooit teveel tijd aan een nieuw plan werd besteed. In een discussie gaf hij onmiddelijk de haalbaarheid aan. Als het Jaco's zegen had, dan werd het plan meestal vlot gerealiseerd, en bleek het CWI nooit een probleem.

Hier in Eindhoven benijd ik de situatie op het CWI, hoewel die natuurlijk zal veranderen met een nieuwe clusterleider.

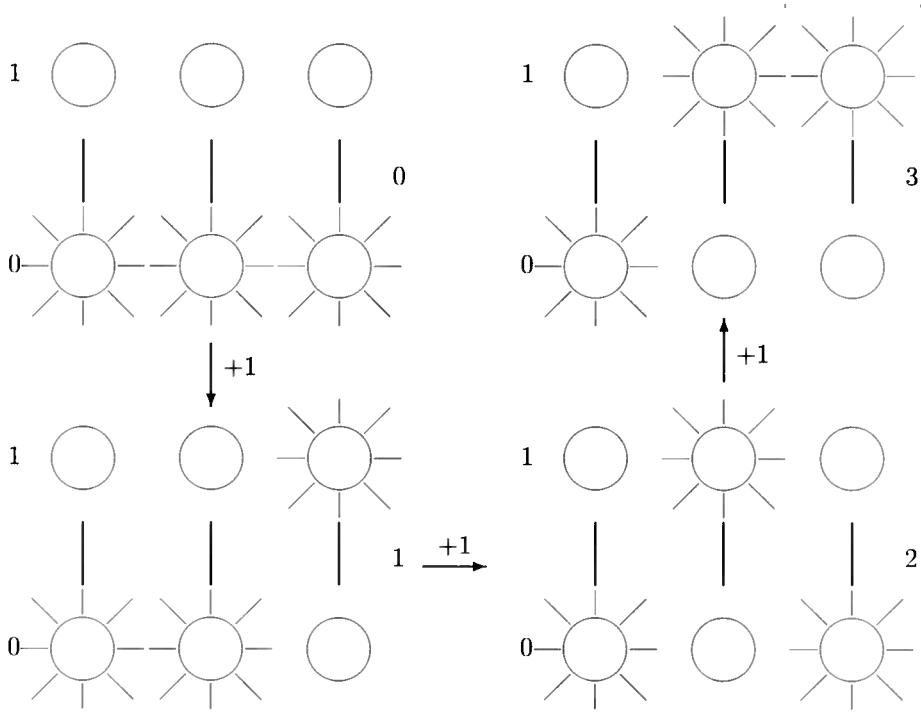
Nu mijn bijdrage. Mijn eerste op het CWI geschreven artikel was zodanig irrelevant, dat ik het vrijwel niemand heb laten lezen. Het heeft 14 jaar in de kast gelegen, maar het lijkt me typisch geschikt voor deze gelegenheid. Het is geschreven in een tijd dat ik mij volledig moest aanpassen aan de denk- en werkwijze op het CWI binnen de groep AP2 die volstrekt anders was dan alles dat ik in Twente had gezien. Het artikel zegt dat als we n-waardige geheugenelementen zouden gebruiken, die op een IC ruimte proportioneel aan

> n in zouden nemen, we het best met driewaardige elementen kunnen werken om het totale ruimtebeslag per opgeslagen informatieeenheid te minimaliseren. Bits zijn vanuit dit perspectief niet de beste keuze. Hier komt het, met enkele redactionele wijzigingen.

De gegevens in het geheugen van een computer worden opgeslagen in tweewaardige geheugenelementen die worden aangeduid met de term *bits*. Een *byte* wordt gevormd door 8 bits, een *word* bestaat uit 16 bits en 8388608 bits is een aardige inschatting van het gemiddelde geheugen van een computer.

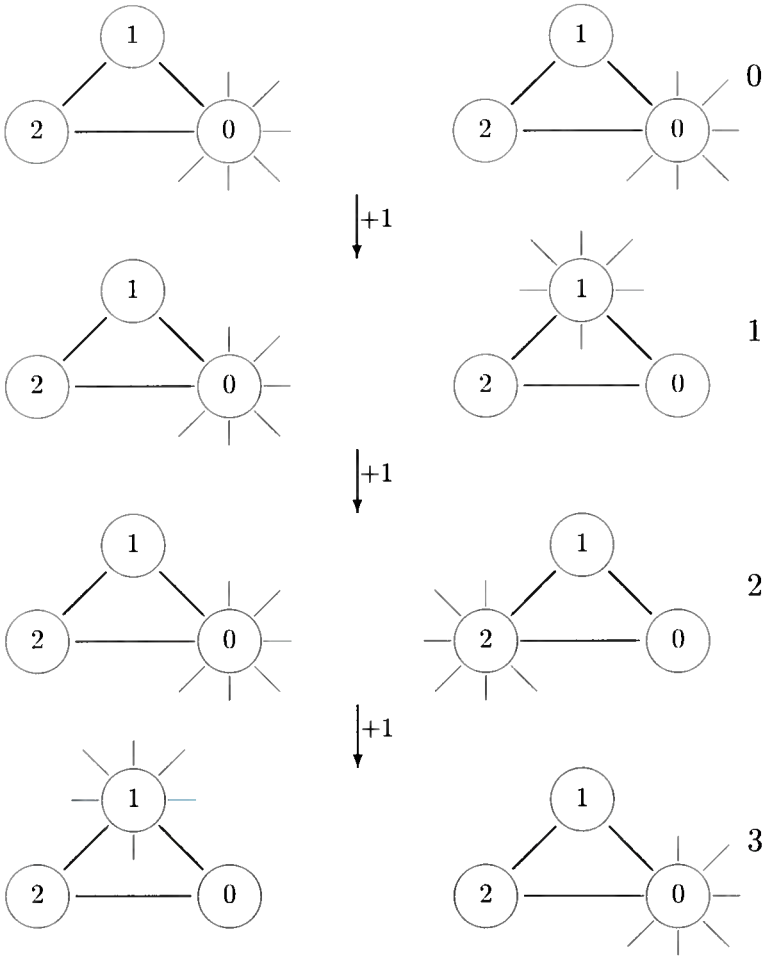
Het begrip bit is zo'n natuurlijk gegeven dat weinigen zich zullen afvragen waarom geheugens met tweewaardige, en niet met drie, vier, vijf of wellicht honderdwaardige basiselementen werken. Men zou geneigd kunnen zijn te denken dat bits zo mooi corresponderen met 0 of 5 volt, of met licht en donker. Maar dit argument is niet zo sterk. Tussen licht en donker kan het in veel gradaties schemeren, en de voltages -5V, 0V, 5V, 10V corresponderen met een vierwaardig systeem. In het vervolg zal blijken dat bits zelfs niet altijd het voordeligst zijn.

We nemen aan dat een bit wordt weergegeven door een toestandsautomaat met twee toestanden. De toestand waarin het systeem zich bevindt wordt weergegeven door een stralenkrans om deze toestand. Door gebruik te maken van meerdere bits kunnen we veel meer toestanden representeren. De gebruikelijke manier van binair tellen staat in de volgende figuur:

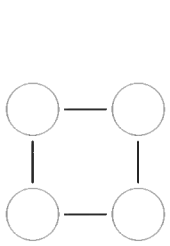


Met 6 toestanden kunnen 3 bits worden gemaakt omdat ieder bit immers twee toestanden vergt. Het drietal bits links boven in de figuur geeft het getal 0 weer. Als er één bij opgeteld wordt, springt het meest rechterbit van 0 naar 1. Weer één optellen zorgt ervoor dat het meest rechterbit weer naar 0 springt. Als dit gebeurt, zal tevens het bit links daarvan verspringen. Zo kan men van 0 tot 8 tellen. Met 8 bits (16 toestanden) wordt het getallenbereik van 0 tot 256 bestreken. Dit komt precies overeen met het aantal tekens in de uitgebreide ASCII tekenset. Met 16 bits (32 toestanden) zijn 65536 verschillende getallen te maken. Ook dit zal voor kenners een bekend getal zijn.

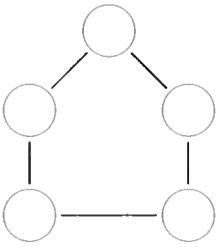
We kunnen in plaats van twee ook drie toestanden per geheugenelement nemen. Laten we dit trits noemen. Een trit kan de waarden 0, 1 en 2 aannemen. Tellen gebeurt op dezelfde wijze als bij bits. Zie hieronder.



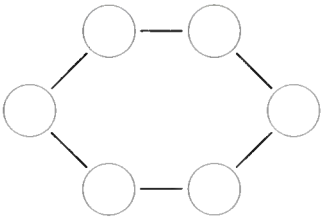
Telkens wanneer er één opgeteld wordt, verspringt de stralenkrans rond een toestand. Als die in toestand 2 staat gaat die weer naar 0 en verspringt bovendien de stralenkrans in de trit links ervan. Met 2 trits kunnen de getallen 0 t/m 8 bereikt worden. Met 11 trits (33 toestanden) komen we al tot 177147. We kunnen dit ook doen voor vier, vijf, zes etc. waardige elementen die ik qits, pits en hits heb genoemd, de laatste twee naar de Griekse getallen pente en hexa die staan voor vijf en zes.



qit



pit



hit

Met welke manier van tellen kan, gegeven een aantal toestanden, het grootste getal worden weergegeven? Het blijken trits te zijn. Dit wordt het best geïllustreerd door de volgende tabel. Horizontaal is het aantal toestanden uitgezet, verticaal de manier van tellen. 102 en 1020 geven aan dat geheugenelemente met 102 en 1020 toestanden zijn gebruikt. In de tabel staat het aantal verschillende bereikbare getallen Een streepje geeft aan dat het totaal aantal toestanden niet deelbaar is door het aantal toestanden per element.

	2	3	4	5	6	12	24	102	1020
bits	2	-	4	-	8	64	4096	2.310^{15}	$3.4\ 10^{153}$
trits	-	3	-	-	9	81	6561	$1.7\ 10^{16}$	$1.7\ 10^{162}$
qits	-	-	4	-	-	64	4096	$2.3\ 10^{15}$	$3.4\ 10^{153}$
pits	-	-	-	5	-	-	-	-	$3.9\ 10^{142}$
hits	-	-	-	-	6	36	1296	$1.7\ 10^{13}$	$1.9\ 10^{132}$
102	-	-	-	-	-	-	-	102	$1.2\ 10^{20}$
1020	-	-	-	-	-	-	-	-	1020

Door enig rekenwerk blijkt dat ook voor nog meer toestanden trits de beste keuze zijn. Mogen we nu de conclusie trekken dat bits hun langste tijd gehad hebben? Dat zou te ver gaan. We hebben immers geoptimaliseerd naar het aantal toestanden. Voor geheugen IC's zou dit betekenen dat iedere toestand een vaste hoeveelheid ruimte in zou moeten nemen. Een trit mag dan 1.5 maal zoveel ruimte innemen als een bit. Een dergelijke schakeling is mij onbekend, maar wellicht weet iemand iets te bedenken.

Toch is het feit dat het gebruik van bits niet helemaal vanzelfsprekend is van noemenswaardig belang. Alle software die uitgaat van het binaire talstelsel zal onbruikbaar zijn wanneer een nieuw talstelsel zijn intrede doet. De software die het enige decennia uit zal moeten houden zou met dergelijke problemen te maken kunnen krijgen. Goede software zou geen enkele aanname moeten maken over de onderliggende hardware. Het is de taak van een compiler en het operating systeem de software zo goed mogelijk te laten passen op de hardware.

Building Metric Structures with the *Meas*-Functor

J.I. den Hartog & E.P. de Vink
Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
e-mail jhartog@win.tue.nl, evink@win.tue.nl

Abstract We introduce the functor *Meas* in the category of complete ultra metric spaces and nonexpansive mapping. The main result of this paper is that *Meas* is a welldefined and locally nonexpansive endofunctor. Therefore the functor fits naturally in the metric approach to programming language semantics. The use of *Meas* in the construction of probabilistic powerdomains, either directly or through the use of domain equations, is illustrated with two examples.

1 Introduction

Recently there has been a growing interest in probabilistic systems and the application of formal methods to probabilistic systems. Probabilistic models are designed to describe systems that are inherently probabilistic, such as a faulty communication channel, or programs where probability is explicitly introduced, i.e. random algorithms. These random algorithms have shown to be successful at solving many problems that deterministic algorithms cannot solve or cannot solve efficiently.

Often the most difficult problems to find and diagnose are intermittent problems which occur only in a fraction of the executions or which occur only under specific circumstances. Many properties of probabilistic programs are intermittent by nature. These probabilistic properties cannot be tested by a single or a couple of executions of a program. Instead one needs to consider the average behavior over many program runs. The difficulty in testing for errors in probabilistic systems, which may be very subtle, makes the use of formal methods even more pressing.

An important part of a formal approach is the semantical modeling of programs to give a precise description of a program and to create a setting in which one can reason about a program. Even when using a syntactical reasoning approach, such as e.g. process algebra [5, 3] or Hoare logic [20] one still needs a semantical model to check the correctness of reasoning rules. A metric approach to defining semantics uses metric spaces as semantical domains. In reasoning about the processes in these domains, the properties of metric spaces can be used. Banach's theorem, for example, which guarantees existence of a unique fixed points for so-called 'contractive functions' on a complete metric space, is a useful tool for definitions and in showing equality. After initial work by

Arnold and Nivat [4, 24], a vast amount of work has been done in this area by the Amsterdam Concurrency Group led by Jaco de Bakker, see e.g. [8]. An overview of the use of metric techniques and of many of the results obtained can be found in [9, 12].

Domain equations play an important role in the specification of semantical domains. Domain equations over metric spaces were pioneered by De Bakker and Zucker [10] in the early eighties. Several types of domain equations can be solved by purely metric means. See e.g. [9]. A categorical approach to solving domain equations was developed by America and Rutten [1], while [27] provides a coalgebraic reformulation of these results. The approach of America and Rutten, translating results of [31] to the metric context, can be used to solve domain equations based on so-called locally contractive functors. Locally nonexpansive functors also play an important role in this setting as the composition of a locally nonexpansive functor and a locally contractive functor is again locally contractive.

Several methods are used to describe probabilistic choices within a semantical domain. For a probabilistic choice between finitely many options, a set or a multiset of options labeled with probabilities can be used, see e.g. [17, 15], or a discrete probability distribution over options [29]. The number of options for a probabilistic choice, however, is not always finite. When modeling the performance of a system, the amount of time needed or the achieved throughput may be probabilistic. As such an interval of real values may be possible as outcomes, at least conceptually. Programs which run indefinitely can also result in a probabilistic choice between infinitely many options even if only finite choices are possible in each step. For probabilistic choices with infinitely many options, the probability of each of the single options is insufficient to find the probability of all events. More information has to be included in the domain. A solution is the use of measures or valuations.

In this paper we will deal with a metric version of measures: The functor *Meas*, which yields an ultrametric space of measures over a given ultrametric space, is used to build metric domains. The functor *Meas* was introduced in [33] and also studied in [32]. An issue, not dealt with in these papers is the completeness of the resulting space which is required for the application of Banach's theorem to obtain unique fixed points. We will show that the functor *Meas* preserves completeness, i.e. $Meas(M)$ is complete for a given complete ultrametric space M. We will also show that the functor *Meas* is locally nonexpansive on the category of complement ultra metric spaces and thus fits in the America-Rutten approach to solving domain equations. This is illustrated in section 4 by building a simple probabilistic branching domain using domain equations.

Van Breugel and Worrell, inspired by work done by Desharnais, Gupta, Jagadeesan and Panangaden [16], introduce in [14, 13] a metric on a domain of measures. A key difference with their approach is that they use the Hutchinson metric to provide quantitative information about the probabilities and in this way a pseudometric rather than an ultrametric is obtained. The processes in their distance have distance zero when they are probabilistically bisimilar and

close when they exhibit approximately the same probabilities. In the metric obtained by using *Meas*, processes that are close have exactly the same probabilities for approximately the same processes. Probabilistic bisimilar programs are assigned the same process. An advantage of the Van Breugel and Worrell approach is the intuitive way of dealing with distances between probabilities; small differences in probabilities lead to a small distance between processes. A disadvantage is that a pseudometric is obtained rather than an ultrametric. Only having a pseudometric instead of a metric means that the uniqueness of fixed points is not guaranteed. Additionally, ultrametricity is often useful in reasoning about processes in a metric domain. A property of ultrametric spaces that is exploited in this setting, for example, is the fact that two open balls are either disjoint or one is contained in the other.

In [14] a real-valued logic is introduced which can be used to characterize the distance between processes. In this paper a comparison is also given with the pseudo metric introduced by Desharnais, Gupta, Jagadeesan and Panangaden in [16], the metric on compact support measures from [33] also used here and with the pseudometric introduced by Norman in his thesis [25, section 6.1].

In [6] Baier and Kwiatkowska present a metric space of valuations and compare it with spaces defined using a set-theoretic and a complete partial order approach. The valuations introduced in [6] are similar to the compact support measures used here.

Slightly different are the valuations used in the thesis work of Jones [21] to introduce probabilistic semantical domains. These valuations are a slight simplification of measures. Not every measure necessarily has a corresponding valuation. In the setting of metric spaces each valuation does have unique extensions to a measure, though not necessarily a compact support measure.

The remainder of this paper is organized as follows: Section 2 collects some metric and measure theoretic definitions and results. Section 3 introduces the functor *Meas*. It is shown that this functor is well-defined and locally nonexpansive on the category of complete ultrametric spaces with nonexpansive mappings. Finally, section 4 illustrates the use of the functor for the construction of a probabilistic semantical domain.

2 Mathematical preliminaries

We assume the reader to be familiar with the basic notions from metric topology, measure theory and category theory. (See, e.g., [9, 30], [18, 26], and [22, 11], respectively.)

Definition 2.1 *A function* $d\colon M \times M \to [0,1]$ *is a 1-bounded ultrametric on the nonempty set* M *if the following conditions are met:*

(i) $d(x,y) = d(y,x)$

(ii) $d(x,y) = 0 \iff x = y$

(iii) $d(x,z) \leq \max\{\, d(x,y), d(y,z) \,\}$

for all $x, y, z \in M$.

Condition (iii) of Definition 2.1 is referred to as the strong triangle-inequality. For a function d to be a metric on M, only the weaker triangle-inequality

$$d(x,z) \leq d(x,y) + d(y,z)$$

is required. An immediate consequence of the strong triangle-inequality is that two ε-balls $B_\varepsilon(x)$ and $B_\varepsilon(z)$ are either disjoint or coincide. For, suppose $y \in B_\varepsilon(x) \cap B_\varepsilon(z)$, then $d(x,z) \leq \max\{ d(x,y), d(y,z) \} < \varepsilon$. Now, if $w \in B_\varepsilon(x)$, then $d(w,z) < \max\{ d(w,x), d(x,z) \} < \varepsilon$. Hence, $w \in B_\varepsilon(z)$. So $B_\varepsilon(x) \subseteq B_\varepsilon(z)$. By symmetry, $B_\varepsilon(x) = B_\varepsilon(z)$ follows. Similarly one can show that $B_\delta(x) \subseteq B_\varepsilon(z)$ if $\delta \leq \varepsilon$ and $x \in B_\varepsilon(z)$.

Clearly, an ultrametric on a set is also an metric on that set. So, an ultrametric induces a topology, viz. the topology $\mathcal{O}$ generated by the open balls. The notation $\mathcal{O}_\varepsilon$ denotes the collection $\{ O \in \mathcal{O} \mid x \in O \Rightarrow B_\varepsilon(x) \subseteq O \}$. It follows that, in an ultrametric space, each $O \in \mathcal{O}_\varepsilon$ is the union of ε-balls, viz. $O = \bigcup\{ B_\varepsilon(x) \mid x \in O \}$, and that the complement $M \setminus O$ is also in $\mathcal{O}_\varepsilon$, hence open, since $M \setminus O = \bigcup\{ B_\varepsilon(x) \mid x \notin O \}$. We use $\mathcal{O}_*$ as shorthand for the set $\bigcup_n \mathcal{O}_{2^{-n}}$.

A metric d on a set M also induces a distance-function on the collection of subsets of M. This is the so-called Hausdorff-distance d_H, given by

$$d_H(X,Y) = \{ \varepsilon > 0 \mid \forall O \in \mathcal{O}_\varepsilon : X \subseteq O \iff Y \subseteq O \}.$$

The Hausdorff-metric enjoys various nice properties. For example, if d is an ultrametric on M, so is d_H on $\mathcal{P}_{cl}(M)$ with $\mathcal{P}_{cl}(M)$ the collection of closed subsets of M. A similar results holds for the hyperspace $\mathcal{P}_{co}(M)$ of compact subsets of M. See, e.g., [10, 9].

Lemma 2.2 *If M with distance function d is a complete ultrametric space, then $\mathcal{P}_{co}(M)$ endowed with the Hausdorff-distance induced by d is also a complete metric space.*

The metric d on M also induces a metric on the collection $A \to M$ of mappings from A to M. This distance is called the distance of pointwise convergence and is given by

$$d_F(f,g) = \sup\{ d(f(a), g(a)) \mid a \in A \}$$

for $f, g : A \to M$. Again, as can be straightforwardly shown, ultrametricity and completeness of d implies ultrametricity and completeness of d_F.

The notation $\mathcal{B}(M)$ for an ultrametric space M denotes the Borel-σ-algebra on M, i.e. the least σ-algebra containing all open sets of M. Recall that a Borel probability measure is a Borel-measure μ such that $\mu(M) = 1$.

From a technical point of view it is attractive when a Borel probability measure 'lives' on a compact set. This will be instrumental in our proof of the preservation of completeness by the functor *Meas* below.

Definition 2.3 *Let M be an ultrametric space. A Borel probability measure μ is said to have compact support if μ vanishes outside some compact set K, i.e. $\mu(M \setminus K) = 0$ or, equivalently, $\mu(K) = 1$. In this case the support* $\mathrm{spt}(\mu)$ *of μ is defined as the smallest compact set K for which $\mu(M \setminus K) = 0$.*

From the definition of a compact support it follows that for any open O in M we have that

$$\mu(O) = 0 \iff \mathrm{spt}(\mu) \cap O = \emptyset, \tag{1}$$

since the intersection of a closed and a compact set is again compact. The property 1 is used in the proof of the following lemma. which stretches σ-additivity of measures to arbitrary additivity of certain collections of balls.

Lemma 2.4 *If μ is a measure on M and $\{ B_\varepsilon(x_i) \mid i \in I \}$ a collection of pairwise disjoint ε-balls in M, then it holds that $\mu(\bigcup\{ B_\varepsilon(x_i) \mid i \in I \}) = \sum\{ \mu(B_\varepsilon(x_i)) \mid i \in I \}$.*

Proof Let $K = \mathrm{spt}(\mu)$. As $\{ B_\varepsilon(x) \mid x \in K \}$ is an open cover of K, it follows from ultrametricity of d and compactness of K, that $K \cap B_\varepsilon(x_i) \neq \emptyset$ iff $i \in I'$, for some finite subset $I' \subseteq I$. Hence $\mu(B_\varepsilon(x_i)) = 0$ for $i \in I \setminus I'$. Likewise, since $K \cap \bigcup\{ B_\varepsilon(x_i) \mid i \in I \setminus I' \} = \emptyset$, we have that $\mu(\bigcup\{ B_\varepsilon(x_i) \mid i \in I \setminus I' \}) = 0$. Therefore,

$$\begin{aligned}
&\mu(\bigcup\{ B_\varepsilon(x_i) \mid i \in I \}) \\
&\quad = \mu(\bigcup\{ B_\varepsilon(x_i) \mid i \in I' \}) + \mu(\bigcup\{ B_\varepsilon(x_i) \mid i \in I \setminus I' \}) \\
&\quad = \mu(\bigcup\{ B_\varepsilon(x_i) \mid i \in I' \}) \\
&\quad = \sum\{ \mu(B_\varepsilon(x_i)) \mid i \in I' \} \\
&\quad = \sum\{ \mu(B_\varepsilon(x_i)) \mid i \in I \}.
\end{aligned}$$

which was to be proven. □

We have the following extension result for measures defined on $\mathcal{O}_*$. A proof, in a more general setting, can be found in [19, Sec. 13].

Theorem 2.5 *If μ is a Borel probability measure on $\mathcal{O}_*$ then there is a unique extension to a measure $\bar{\mu}$ on $\mathcal{B}$.*

The next lemma states that two measures coincide if they agree on the open balls.

Lemma 2.6 *If μ, ν are two Borel probability measures of compact support on an ultrametric space M such that $\mu(B_\varepsilon(x)) = \nu(B_\varepsilon(x))$ for all $\varepsilon > 0$ and $x \in M$, then $\mu = \nu$.*

Proof It is first shown that the two measures coincide on all sets in $\mathcal{O}_\varepsilon = \{ O \in \mathcal{O} \mid x \in O \Rightarrow B_\varepsilon(x) \subseteq O \}$ (for all $\epsilon > 0$). A basic result from measure theory can then be used to show that the two measures are the same.

For a given $\epsilon > 0$ take a finite collection of points $x_1, \ldots x_n$ such that both μ and ν vanish outside $B_\varepsilon(x_1) \cup \ldots \cup B_\varepsilon(x_n)$. Clearly such points exists as the collection of all ϵ-balls is a cover of $\mathrm{spt}(\mu) \cup \mathrm{spt}(\nu)$. This set is compact and consequently has a finite subcover. By ultrametricity, the balls $B_\varepsilon(x_1), \ldots, B_\varepsilon(x_n)$ can be assumed to be pairwise disjoint. Also, for a given set $O \in \mathcal{O}_\varepsilon$, a ball $B_\varepsilon(x)$ is either completely inside O or completely outside O. We now reason as follows:

$$\begin{aligned}
\mu(O) &= \mu(\cup_{i=1}^n B_\varepsilon(x_i) \cap O) \\
&= \mu(\cup\{ B_\varepsilon(x_i) \mid x_i \in O, i = 1, \ldots, n \}) \\
&= \textstyle\sum \{ \mu(B_\varepsilon(x_i)) \mid x_i \in O, i = 1, \ldots, n \} \\
&= \textstyle\sum \{ \nu(B_\varepsilon(x_i)) \mid x_i \in O, i = 1, \ldots, n \} \\
&= \nu(\cup\{ B_\varepsilon(x_i) \mid x_i \in O, i = 1, \ldots, n \}) \\
&= \nu(\cup_{i=1}^n B_\varepsilon(x_i) \cap O) \\
&= \nu(O).
\end{aligned}$$

The two sums are equal because the two measures coincide on balls.

We have now shown that the measures coincide on the ring $\mathcal{O}_* = \bigcup_{\epsilon>0} \mathcal{O}_\varepsilon$. The σ-ring generated by $\mathcal{O}_*$ is the collection of all Borel sets. A basic result from measure theory states that a measure on a ring extends uniquely to a measure on the generated σ-ring (cf., e.g., [18]). Therefore, two measures which coincide on $\mathcal{O}_*$ must also coincide on all Borel sets. Hence $\mu = \nu$. □

The following property claims that if two Borel probability measures of compact support coincide on small balls, the also coincide on all bigger balls.

Lemma 2.7 *Let μ, ν be two Borel-measures of compact support on an ultrametric space M. Let $\delta > 0$ be such that $\mu(B_\delta(x)) = \nu(B_\delta(x))$ for any $x \in M$. Then $\mu(B_\varepsilon(x)) = \nu(B_\varepsilon(x))$ for any $x \in M$ and $\varepsilon \geq \delta$.*

Proof Pick $x \in M$ and $\varepsilon \geq \delta$. We have, by ultrametricity, that

$$B_\varepsilon(x) = \bigcup \{ B_\delta(x') \mid d(x, x') < \varepsilon \}.$$

Moreover, $\{ B_\delta(x') \mid d(x, x') < \varepsilon \}$ is a collection of pairwise disjoint δ-balls (although for some x', x'' the balls $B_\delta(x'), B_\delta(x'')$ may coincide). By applying Lemma 2.4 we obtain

$$\begin{aligned}
\mu(B_\varepsilon(x)) &= \mu(\bigcup \{ B_\delta(x') \mid d(x, x') < \varepsilon \}) \\
&= \mu(\bigcup \{ B_\delta(x') \mid d(x, x') < \varepsilon,\ B_\delta(x') \cap \mathrm{spt}(\mu) \neq \emptyset \}) \\
&= \textstyle\sum \{ \mu(B_\delta(x')) \mid d(x, x') < \varepsilon \}.
\end{aligned}$$

A similar argument holds for the Borel measure ν. We conclude that

$$\begin{aligned}\mu(B_\varepsilon(x)) &= \sum\{\,\mu(B_\delta(x')) \mid d(x,x') < \varepsilon\,\} \\ &= \sum\{\,\nu(B_\delta(x')) \mid d(x,x') < \varepsilon\,\} \\ &= \nu(B_\varepsilon(x)),\end{aligned}$$

since μ and ν coincide on δ-balls. □

Next we give the definition of a locally contractive and locally nonexpansive functor on the category *CUMS* of complete ultrametric spaces and nonexpansive mappings.

Definition 2.8 *Let $\mathcal{F}$: CUMS → CUMS be a functor.*

(a) $\mathcal{F}$ is called locally contractive if, for some α, $0 \leq \alpha < 1$, it holds, for all arrows f, g, that $d_F(\mathcal{F}(f), \mathcal{F}(g)) \leq \alpha \cdot d_F(f,g)$.

(b) $\mathcal{F}$ is called locally nonexpansive if it holds that $d_F(\mathcal{F}(f), \mathcal{F}(g)) \leq d_F(f,g)$ for all arrows f, g.

It is readily checked that, e.g., the composition of a locally contractive and a locally nonexpansive functor is itself locally contractive. In the literature various semantical domains are constructed using different combinations of locally contractive and locally nonexpansive functors, usually given by domain equations. We mention

- $\mathbb{P} = \{p_0\} \cup \mathcal{P}_{cl}((A \cup \{\epsilon\}) \times \mathbb{P}/2$ from [10]
- $\mathbb{P} = \{p_0\} \cup \mathcal{P}_{co}((A \cup \{\tau\}) \times \mathbb{P}/2$ from [7]
- $\mathbb{P} = \{p_0\} \cup (\Sigma \to \mathcal{P}_{cl}(Step))/2$ where $Step = (\Sigma \times \mathbb{P}) \cup Send \cup Answer$ from [2].

The result underpinning these equations, stated in the theorem below, is proven in [1].

Theorem 2.9 *Let $\mathcal{F}$ be a locally contractive functor $\mathcal{F}$: CUMS → CUMS Then the domain equation $\mathbb{P} \simeq \mathcal{F}(\mathbb{P})$ has a unique solution up to isomorphism.*

$$\begin{aligned}\mu(B_\varepsilon(x)) &= \mu(\bigcup\{\,B_\delta(x') \mid d(x,x') < \varepsilon\,\}) \\ &= \mu(\bigcup\{\,B_\delta(x') \mid d(x,x') < \varepsilon,\ B_\delta(x') \cap \mathrm{spt}(\mu) \neq \emptyset\,\}) \\ &= \sum\{\,\mu(B_\delta(x')) \mid d(x,x') < \varepsilon,\ B_\delta(x') \cap \mathrm{spt}(\mu) \neq \emptyset\,\} \\ &= \sum\{\,\mu(B_\delta(x')) \mid d(x,x') < \varepsilon\,\}\end{aligned}$$

since $\mu(\bigcup\{\,B_\delta(x') \mid d(x,x') < \varepsilon, B_\delta(x') \cap \mathrm{spt}(\mu) = \emptyset\,\}) = 0$ and $\mu(B_\delta(x')) = 0$ for $x' \in M$ such that $d(x,x') < \varepsilon$ and $B_\delta(x') \cap \mathrm{spt}(\mu) = \emptyset$. A similar argument holds for the Borel measure ν. We conclude that

3 The functor *Meas*

In this section we introduce the functor *Meas* on the category of *CUMS* of complete ultrametric spaces and nonexpansive mappings. At the object-level *Meas* yields the metric space of all Borel probability measures of compact support; at the arrow-level *Meas* the construction of a measure-along-a-mapping is used. We will show that *Meas* is a well-defined and locally nonexpansive functor, which makes it a useful building block for the definition of metric domains for probabilistic program constructs.

Definition 3.1 *Let M be an ultrametric space with metric d.*

(a) $Meas(M)$ denotes the space of all Borel probability measures on M of compact support.

(b) The distance d_{Meas} on $Meas(M)$ is defined by

$$d_{Meas}(\mu,\nu) = \inf\{\, \epsilon > 0 \mid \forall x \in M : \mu(B_\varepsilon(x)) = \nu(B_\varepsilon(x)) \,\}.$$

In the remainder of this paper the term measure refers to a Borel probability measure with compact support. We also fix an ultrametric space M with distance d. Lemma 3.3 states that d_{Meas} is an ultrametric on $Meas(M)$. For the proof of the lemma we need a technical property first.

Lemma 3.2 *If $\mu, \nu \in Meas(M)$ are such that $d_{Meas}(\mu,\nu) \leq \varepsilon$ for some $\varepsilon > 0$, then $\mu(O) = \nu(O)$ for any $O \in \mathcal{O}_\varepsilon$.*

Proof By ultrametricity of M, any $O \in \mathcal{O}_\varepsilon$ can be represented as a disjoint union of ε-balls. From Lemma 2.7 we have that μ and ν coincide on ε-balls. Therefore we conclude

$$\mu(O) = \sum\{\, \mu(B_\varepsilon(x_i)) \mid i \in I \,\} = \sum\{\, \nu(B_\varepsilon(x_i)) \mid i \in I \,\} = \nu(O)$$

using Lemma 2.4 twice. □

We are now ready to show that d_{Meas} is a distance function on $Meas(M)$.

Lemma 3.3 *For an ultrametric space (M,d) the pair $(Meas(M), d_{Meas})$ is an ultrametric space as well.*

Proof We check the conditions of Definition 2.1. Symmetry is obvious. Clearly $d_{Meas}(\mu,\mu) = 0$ for any $\mu \in Meas(M)$. Suppose $d_{Meas}(\mu,\nu) = 0$. In order to prove that $\mu = \nu$ it suffices, by Lemma 2.6, to check that $\mu(B_\varepsilon(x)) = \nu(B_\varepsilon(x))$ for any $\varepsilon > 0$ and $x \in M$. This is immediate from the definition of d_{Meas}. Finally, in order to verify the strong triangle-inequality $d_{Meas}(\mu,\rho) \leq \max\{\, d_{Meas}(\mu,\nu), d_{Meas}(\nu,\rho) \,\}$ we reason as follows: Suppose that we have that $\max\{\, d(\mu,\nu), d(\nu,\rho) \,\} < \epsilon$. Then both $d_{Meas}(\mu,\nu) < \varepsilon$ and $d_{Meas}(\nu,\rho) < \varepsilon$. Then, by Lemma 2.7, we have that $\mu(B_\varepsilon(x)) = \nu(B_\varepsilon(x))$ and $\nu(B_\varepsilon(x)) = \rho(B_\varepsilon(x))$ for any $x \in M$. Therefore, $\mu(B_\varepsilon(x)) = \rho(B_\varepsilon(x))$ for any ε-ball, thus $d(\mu,\rho) \leq \varepsilon$. So we have

$$\max\{\, d(\mu,\nu), d(\nu,\rho) \,\} < \varepsilon \Rightarrow d(\mu,\rho) \leq \varepsilon$$

and, subsequently, $d(\mu,\rho) \leq \max\{\, d(\mu,\nu), d(\nu,\rho) \,\}$. □

Before addressing the completeness of $Meas(M)$ we need a property relating the Hausdorff-distance and the distance on measures.

Lemma 3.4 *Let $\mu, \nu \in Meas(M)$ and $\varepsilon > 0$. Then it holds that*

$$d(\mu,\nu) \leq \varepsilon \Rightarrow d_H(\mathrm{spt}(\mu), \mathrm{spt}(\nu)) \leq \varepsilon.$$

Proof Pick $\mu, \nu \in Meas(M)$. Put $K = \mathrm{spt}(\mu)$, $L = \mathrm{spt}(\nu)$. If $d_{Meas}(\mu,\nu) \leq \varepsilon$, then, by Lemma 3.2, $\mu(O) = \nu(O)$ for $O \in \mathcal{O}_\varepsilon$. In particular $\forall O \in \mathcal{O}_\varepsilon$: $\mu(O) = 0 \iff \nu(O) = 0$. This is equivalent to $\forall O \in \mathcal{O}_\varepsilon$: $K \cap O = \emptyset \iff L \cap O = \emptyset$. Since, by ultrametricity, $O \in \mathcal{O}_\varepsilon \iff M \setminus O \in \mathcal{O}_\varepsilon$, it follows that $\forall O \in \mathcal{O}_\varepsilon$: $K \subseteq O \iff L \subseteq O$, i.e. $d_H(K,L) \leq \varepsilon$. □

So far we have that $Meas(M)$ is an ultrametric space if $M \in CUMS$. For $Meas(M) \in CUMS$ we need to show the completeness of $Meas(M)$.

Theorem 3.5 *If the ultrametric space M is complete, so is the ultrametric space $Meas(M)$.*

Proof Suppose that M is an ultra-metric space and let $(\mu_i)_i$ be a Cauchy sequence in $Meas(M)$. Let the index i_n be such that $\forall i,j > i_n$: $d_{Meas}(\mu_i,\mu_j) \leq 2^{-n}$. So, by Lemma 3.2 we have that $\forall i,j > i_n$: $\mu_i(O) = \mu_j(O)$ for $O \in \mathcal{O}_{2^{-n}}$. Additionally, we can assume $i_{n+1} > i_n$. Define the mapping $\mu: \mathcal{O}_* \to [0,1]$ by $\mu(O) = \lim_i \mu_i(O)$.

Claim 1: μ is finitely additive. Proof of claim 1: Suppose $O_1, \ldots, O_r \in \mathcal{O}_*$ are disjoint. Pick n such that $O_1, \ldots, O_r \in \mathcal{O}_n$. Note $O_1 \cup \ldots \cup O_r \in \mathcal{O}_n$. Pick $i > i_n$. Then $\mu(O_1 \cup \ldots \cup O_r) = \mu_i(O_1 \cup \ldots \cup O_r) = \mu_i(O_1) + \cdots + \mu_i(O_r) = \mu(O_1) + \cdots + \mu(O_r)$ by additivity of μ_i.

Claim 2: μ has a compact support. Proof of claim 2: Put $K_i = \mathrm{spt}(\mu_i)$. By Lemma 3.4 it holds that $(K_i)_i$ is a Cauchy sequence in the hyperspace $\mathcal{P}_{co}(M)$ of compact subsets of M. By completeness of M, the hyperspace $\mathcal{P}_{co}(M)$ is complete. Therefore $K = \lim_i K_i$ exists and is compact. Let $O \in \mathcal{O}_n$. Pick $i > i_n$. Then

$$\begin{aligned} O \cap K = \emptyset \quad &\iff \quad [\text{lemma 3.4a, } d(K,K_i) \leq 2^{-n}] \quad O \cap K_i = \emptyset \\ &\iff \quad [K_i \text{ support of } \mu_i] \quad \mu_i(O) = 0 \\ &\iff \quad [i > i_n] \quad \mu(O) = 0. \end{aligned}$$

So, $K = \mathrm{spt}(\mu)$ and K is compact.

Claim 3: μ is σ-additive. Proof of claim 3: Let $(O_i)_i$ be a disjoint sequence in $\mathcal{O}_*$ such that $\bigcup_i O_i \in \mathcal{O}_*$. Pick n such that $\bigcup_i O_i \in \mathcal{O}_{2^{-n}}$. Put $O' = M \setminus (\bigcup_i O_i)$. Note that $O' \in \mathcal{O}_{2^{-n}}$. In particular, O' is an open set. We have that $\{ O_i \mid i \in \mathbb{N} \} \cup \{O'\}$ is an open cover of K. Since K is compact, we derive that the index-set $I = \{ i \mid O_i \cap K \neq \emptyset \}$ is finite. Note, by disjointness, $(\bigcup_{i \notin I} O_i) \cap K = \emptyset$, hence $\mu(\bigcup_{i \notin I} O_i) = 0$. Also, for $i \notin I$ we have $\mu(O_i) = 0$. Therefore $\mu(\bigcup_{i \notin I} O_i) = \sum_{i \notin I} \mu(O_i)$. We conclude

$$\mu(\bigcup_i O_i) \quad = \quad \mu(\bigcup_{i \in I} O_i \cup \bigcup_{i \notin I} O_i)$$

$$
\begin{aligned}
&= \quad \text{[finite additivity]} \quad \sum_{i \in I} \mu(O_i) + \mu(\bigcup_{i \notin I} O_i) \\
&= \sum_{i \in I} \mu(O_i) \sum_{i \notin I} \mu(O_i) \\
&= \sum_{i} \mu(O_i).
\end{aligned}
$$

Define $\mu: \mathcal{B} \to [0,1]$, using Theorem 2.5, as the unique extension on $\mathcal{B}$ of μ on $\mathcal{O}_*$. Then we have that $\mu \in Meas(M)$ by the above and $d_{Meas}(\mu, \mu_i) \leq 2^{-n}$ for $i > i_n$, so $\mu = \lim_i \mu_i$. □

We are now in a position to define *Meas* as a functor on the category *CUMS* of complete ultrametric spaces with nonexpansive maps.

Definition 3.6 *The functor Meas: CUMS → CUMS is given as follows:*

(a) The functor Meas assigns the complete ultrametric space Meas(M) to any complete ultrametric space M ∈ CUMS.

(b) The functor Meas assigns to each nonexpansive mapping f: M → N from a complete ultrametric space M to a complete ultrametric space N the mapping Meas(f): Meas(M) → Meas(N) given by

$$Meas(f)(\mu)(B) = \mu(f^{-1}(B)).$$

From Lemma 3.3 and 3.5 we have that indeed $Meas(M) \in CUMS$ for $M \in CUMS$. It can be straightforwardly verified, for $f: M \to N$ nonexpansive, that $Meas(f)(\mu) \in Meas(N)$ using the fact that $f^{-1}(B) \in \mathcal{B}(M)$ for $B \in \mathcal{B}(N)$ and using basic set-theoretic properties of f^{-1}. In fact, $Meas(f)(\mu)$ is referred to as the measure μ along f. So, *Meas* is a well-defined functor on *CUMS*. The last result of this section states that *Meas* is a locally nonexpansive functor as well.

Theorem 3.7 *The functor Meas on CUMS is locally nonexpansive functor.*

Proof Let $M, N \in CUMS$ and $f, g : M \to N$ be two nonexpansive functions. If $d_H(f,g) < \varepsilon$, then for all $y \in N$ we have that $f^{-1}(B_\varepsilon(y)) = g^{-1}(B_\varepsilon(y))$. For, if $x \in f^{-1}(B_\varepsilon(y))$, then $f(x) \in B_\varepsilon(y)$ and, since $d_H(f,g) < \varepsilon$, $d(f(x), g(x)) < \varepsilon$. So, by ultrametricity, $d(y, g(x)) \leq \max\{ d(y, f(x)), d(f(x), g(x)) \} < \varepsilon$, i.e. $g(x) \in B_\varepsilon(y)$. But then $Meas(f)(\mu)(B_\varepsilon(x)) = \mu(f^{-1}(B_\varepsilon(x))) = \mu(g^{-1}(B_\varepsilon(x))) = Meas(g)(\mu)(B_\varepsilon(x))$ or $d_{Meas}(Meas(f)(\mu), Meas(g)(\mu)) < \varepsilon$ for all $\mu \in Meas(M)$. Thus,

$$d(f,g) < \varepsilon \Rightarrow d(Meas(g), Meas(f)) \leq \varepsilon.$$

Since this holds for all $\varepsilon > 0$ it follows that $d(Meas(g), Meas(f)) \leq d(f,g)$. □

Summarizing this section we have established that the functor *Meas* on the category of complete ultrametric spaces and nonexpansive mappings is well-defined and locally nonexpansive.

4 Examples and Conclusions

Above we have discussed the functor *Meas* first introduced in [33] and further studied in [32]. Using *Meas*, one can construct an ultrametric space of measures over a given ultrametric space. More precisely, the space of all compact support Borel probability measures is obtained. We have shown that the functor *Meas* preserves completeness and is locally nonexpansive on *CUMS*, the category of complete ultrametric spaces and nonexpansive mapping.

In definition 3.1 the ultrametric space $Meas(M)$ is defined for any given ultrametric space M. Banach's fixed point theorem, which is often used in the metric approach to guarantee the existence of unique fixed points, requires that the metric space used is complete. It is therefore important that completeness is preserved by *Meas*, i.e. if a space M is complete than so is $Meas(M)$. That *Meas* does preserves completeness has been shown in theorem 3.5. Thus *Meas* can be used to construct suitable metric spaces.

As an example consider the space of finite or infinite action sequences Act^∞ over some action alphabet Act. Equipped with the *Baire metric* this space is a complete ultrametric space. The metric space of all Borel probability measures with compact support $Meas(Act^\infty)$ can be used to describe trace behavior of probabilistic process languages with alphabet Act.

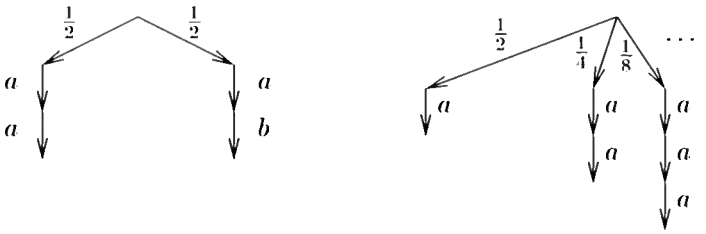
Example 4.1 *Let Act be a given alphabet of action. Let Act^∞ denote the space of finite and infinite sequences of actions with the distance function d_B, the Baire-distance, given by*

$$d_B(w, w') \quad = \quad \inf\{\, \tfrac{1}{2}^n \mid w[n] = w'[n] \,\}$$

for $w, w' \in Act^\infty$. Here $w[n]$ denote the sequence w cut of after n actions. Let the space $\mathbb{P}_l$ be given by

$$\mathbb{P}_l \quad = \quad Meas(Act^\infty).$$

It contains all compact support Borel probability measures over Act^∞ and can be used as a domain to describe behavior of (linear) probabilistic processes. For example the measure Δ_a for some $a \in Act$ is an element of $Meas(Act^\infty)$ that describes that with probability 1 the sequence consisting only of a single action a is executed. The measures Δ_{aa}, Δ_{aaa} and Δ_{a^ω} ($a^\omega = aaaa\ldots$) respectively describe that the sequence aa, aaa or a^ω is executed with probability 1. More precisely, $\Delta_a(B) = 1$ if $a \in B$, $\Delta_a(B) = 0$ otherwise. A measure like Δ_a is referred to as a Dirac measure. Combinations of such measures are also present in $Meas(Act^\infty)$. Finite combinations such as $\frac{1}{2}\Delta_{aa} + \frac{1}{2}\Delta_{ab}$ which chooses between the sequences aa and ab giving equal probability to both, but also infinite combinations such as $\frac{1}{2}\Delta_a + \frac{1}{4}\Delta_{aa} + \frac{1}{8}\Delta_{aaa} + \ldots$ in which any finite number of actions a can be selected according to a geometric distribution.



There are also measures in Meas which cannot be depicted this easily. Define the operation $\bullet/a$ on measures in $\mathbb{P}_l$ by $\mu/a(B) = \mu(\{\, w \mid aw \in B \,\})$ for all Borel sets B, then $\bullet/a$ is contractive function on $\mathbb{P}_l$. That a solution μ for the equation $\mu = \frac{1}{2}\mu/a + \frac{1}{2}\mu/b$ must be present in $Meas(Act^\infty)$ can easily be shown using Banach's fixed point theorem. This measure μ assigns "equal probability" to all infinite sequences of a's and b's. More precisely, with probability 1 some infinite sequence is produced and for any given n and any given sequence of a's and b's of length n the probability of starting with the sequence is $\frac{1}{2}^n$.

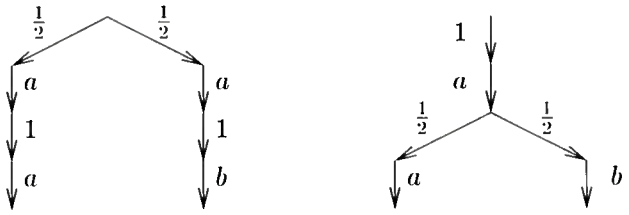
The space $Meas(Act^\infty)$ in the example above is a so-called linear domain; a single probabilistic choice is made to decide the sequence of actions that is produced. One also wants to be able to build branching domains in which choices are made along the way. These branching domains are typically described by domain equations. Definition 3.6 defines *Meas* on nonexpansive functions, the arrows of the category *CUMS*, turning *Meas* into an endofunctor on *CUMS*. The functor *Meas* is shown to be locally nonexpansive in theorem 3.7. As such *Meas* can be used in domain equations in composition with a locally contractive functor. A frequently deployed locally contractive functor is $id_{\frac{1}{2}}(\bullet)$ which simply reduces distances by a half but does not change the space in any other way. The space $Act \times id_{\frac{1}{2}}(\mathbb{P})$, for example can be used to describe processes in $\mathbb{P}$ prefixed with an action in Act.

Example 4.2 *Given an alphabet Act of actions, a branching probabilistic domain $\mathbb{P}_b$ can be specified by the domain equation*

$$\mathbb{P}_b \quad \simeq \quad Meas(Act \times id_{\frac{1}{2}}(\mathbb{P}_b) + Act).$$

We have that $Act \times id_{\frac{1}{2}}(\bullet)$ is locally contractive and so is $Act \times id_{\frac{1}{2}}(\bullet) + Act$ and thus $Meas(Act \times id_{\frac{1}{2}}(\bullet) + Act)$ is also locally contractive by local nonexpansiveness of Meas. This means that the domain equation above must have a unique solution.

The domain $\mathbb{P}_b$ captures branching behavior of probabilistic processes. For readability the examples below use the notation $\rho \cdot x$ for $\rho \cdot \Delta_x$. The process $1 \cdot a$ produces action a with probability 1. The process p_1 given by $p_1 = \frac{1}{2} \cdot \langle a, 1 \cdot a \rangle + \frac{1}{2} \cdot \langle a, 1 \cdot b \rangle$ describes a fair choice between executing a followed by a, and a followed by b. In the process p_2 given by $p_2 = 1 \cdot \langle a, \frac{1}{2} \cdot a + \frac{1}{2} \cdot b \rangle$ the action a is executed before choosing between a and b for the next action.



The domain $\mathbb{P}_b$ contains a process p_3 which satisfies $p_3 = \frac{1}{2} \cdot a + \frac{1}{2} \cdot \langle a, p_3 \rangle$ in other words $p = \frac{1}{2} \cdot a + \frac{1}{2} \cdot \langle a, \frac{1}{2} \cdot a + \frac{1}{2} \cdot \langle a, \ldots \rangle \rangle$. A process p_4, $p_4 = \frac{1}{2} \cdot a + \frac{1}{4} \cdot \langle a, 1 \cdot a \rangle + \frac{1}{8} \cdot \langle a, 1 \cdot \langle a, 1 \cdot a \rangle \rangle + \ldots$, which is similar to process p_2 of example 4.1 is also an element of $\mathbb{P}_b$.

The example p_4 illustrates that structures can be built that are similar to elements of $\mathbb{P}_l$. However, p_1 and p_2 show that distinctions can be made that are not possible in the linear domain. For these examples it is clear which linear process corresponds to the branching process. In general additional structure is required on the domain be able to obtain a linear probabilistic process from a branching probabilistic process [15, chapter 7].

In the metric setting bisimilarity usually corresponds with equality on a metric branching domain [9]. Similarly, in a coalgebraic setting bisimilarity corresponds to equality in the final coalgebra [28]. For the basic probabilistic metric branching domain $\mathbb{P}_b$ in the example above we have that equality on this domain corresponds to the commonly used notion of Larsen-Skou bisimulation [23]. (See [33].) To be more precise: Giving a semantics for a basic probabilistic process language in a standard way using this domain yields an equivalence which is the same as Larsen-Skou bisimulation.

In this paper we have defined the *Meas* functor yielding Borel probability measures with compact support. As can be seen from the examples above (see [15] for a few more examples) the functor *Meas* can be used to build metric structures for modeling probabilistic languages.

References

[1] P. America and J.J.M.M. Rutten. Solving reflexive domain equations in a category of complete metric spaces. *Journal of Computer and System Sciences*, 39:343–375, 1989.

[2] P.H.M. America, J.W. de Bakker, J.N. Kok, and J.J.M.M. Rutten. Denotational semantics of a parallel object-oriented language. *Information and Computation*, 83:152–205, 1989.

[3] S. Andova. Process algebra with probabilistic choice. In J.-P. Katoen, editor, *Proc. ARTS'99*, pages 111–129. LNCS 1601, 1999.

[4] A. Arnold and M. Nivat. Metric interpretations of infinite trees and semantics of nondeterministic recursive programs. *Theoretical Computer Science*, 11(2):181–205, 1980.

[5] J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. *Information and Computation*, 121:234–255, 1995.

[6] C. Baier and M. Kwiatkowska. Domain equations for probabilistic processes (extended abstract). In M. Mislove, M. Nivat, C. Papadimitriou, C. Palamidessi, and J. Parrow, editors, *Proc. Express'97*. Electronic Notes in Theoretical Computer Science 7, 1997.

[7] J.W. de Bakker, J.A. Bergstra, J.W. Klop, and J.-J.Ch Meyer. Linear time and branching time semantics for recursion with merge. *Theoretical Computer Science*, 34:135–156, 1984.

[8] J.W. de Bakker and J.J.M.M. Rutten, editors. *Ten Years of Concurrency Semantics, selected papers of the Amsterdam Concurrency Group*. World Scientific, 1992.

[9] J.W. de Bakker and E.P. de Vink. *Control Flow Semantics*. Foundations of Computing Series. The MIT Press, 1996.

[10] J.W. de Bakker and J.I. Zucker. Processes and the denotational semantics of concurrency. *Information and Control*, 54:70–120, 1982.

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

[12] F. C. van Breugel. *Comparative Metric Semantics for Programming Languages*. Progress in Theoretical Computer Science. Birkhäuser Verlag, Boston, 1998.

[13] F. C. van Breugel and J. Worrell. An algorithm for quantitative verification of probabilistic transition systems. In K.G. Larsen and M. Nielsen, editors, *Proc. 12th CONCUR'01, Aalborg*, pages 336 – 350. LNCS 2154, 2001.

[14] F. C. van Breugel and J. Worrell. Towards quantitative verification of probabilistic transition systems. In F. Orejas, P.G. Spirakis, and J. van Leeuwen, editors, *Proc. 28th ICALP'01*, pages 421 – 432. LNCS 2076, 2001.

[15] J.I. den Hartog. *Probabilistic Extensions of Semantical Models*. PhD thesis, Vrije Universiteit, Amsterdam, 2002.

[16] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labeled Markov systems. In J.C.M. Baeten and S. Mauw, editors, *Proc. CONCUR'99*, pages 258–273. LNCS 1664, 1999.

[17] R.J. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Information and Computation*, 121:59–80, 1995.

[18] P.R. Halmos. *Measure Theory*. Van Nostrand, 1950.

[19] P.R. Halmos. *Measure Theory*, volume 18 of *Graduate Texts in Mathematics*. Springer, reprint edition, 1974.

[20] J.I. den Hartog and E.P. de Vink. Verifying probabilistic programs using a Hoare like logic. *International Journal of Foundations of Computer Science*, 2002. To appear.

[21] C. Jones. *Probabilistic Nondeterminism*. PhD thesis, ECS–LFCS–90–105, University of Edinburgh, 1990.

[22] S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer, 1971.

[23] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991.

[24] M. Nivat. Infinite words, infinite trees, infinite computations. In J.W. de Bakker and J. van Leeuwen, editors, *Foundations of Computer Science III, part 2: Languages, Logic, Semantics*, volume 109 of *Mathematical Centre Tracts*, pages 3–52. Mathematical Centre, Amsterdam, 1979.

[25] G.J. Norman. *Metric Semantics for Reactive Probabilistic Systems*. PhD thesis, University of Birmingham, 1997.

[26] W. Rudin. *Real and Complex Analysis*. McGraw-Hill, 1966.

[27] J. Rutten and D.Turi. On the foundation of final semantics: non-standard sets, metric spaces, partial orders. In *LNCS 666 (Proceedings of the REX Workshop on Semantics: Foundations and Applications)*, pages 477–530. Springer, 1992.

[28] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.

[29] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Massachusetts Institute of Technology, June 1995.

[30] M.B. Smyth. *Handbook of Logic in Computer Science, volume 1, Background: Mathematical Structures*, chapter Topology, pages 641–761. Claredon Press, 1992.

[31] M.B. Smyth and G.D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal on Computing*, 11:761–783, 1982.

[32] E.P. de Vink. On a functor for probabilistic bisimulation and preservation of weak pullbacks. Technical Report IR–444, Vrije Universiteit, Amsterdam, 1998.

[33] E.P. de Vink and J.J.M.M. Rutten. Bisimulation for probabilistic transition systems: a coalgebraic approach. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Automata, Languages and Programming*, pages 460–470. LNCS 1256, 1997. Proc. 24th ICALP'97, Bologna, Italy.

Kan ik je helpen? Wil je tentamen aanvragen?

Jaco ken ik sinds 1975, al heette hij toen voor mij nog "professor De Bakker" of kortweg "prof". Pas in 1985, toen ik was overgestapt naar het CWI, durfde ik hem bij zijn voornaam te gaan noemen. Ach ja, de mores bij de VU zijn anders.
Bij de VU dus, om precies te zijn, in 1975 nog de subfaculteit Wiskunde of het Wiskundig Seminarium geheten; de informatica stond immers nog in de kinderschoenen.
Jaco had er een student rondlopen waarmee ik direct al veel te maken kreeg; Arie de Bruin was ingehuurd als uitzendkracht om te helpen bij de organisatie van het Wiskundig Congres waarvoor de VU in 1976 aan de beurt was. Overigens kreeg diezelfde Arie een tentamenvrijstelling omdat hij het collegedictaat Formele Talen schreef. Jaco was immers buitenwoon (deeltijds) hoogleraar, dus die had daar zelf geen tijd voor.
Ik weet niet hoe het nu is maar destijds waren er heel wat studenten die grote moeite hadden met de vakken die Jaco doceerde. En toch moesten ze een voldoende halen om te kunnen afstuderen. Heel veel mondelinge (her)tentamens moet hij in zijn leven afgenomen hebben.
Studenten konden zo'n tentamen aanvragen met een tentamenbriefje, in 3-voud geloof ik, wat bij de studentenadministratie werd ingeleverd en door diezelfde administratie dan, in het geval van een mondeling tentamen, bij de betreffende docent werd afgeleverd. De docent stelde een datum en tijd vast waarop de student tentamen kon doen en één van de kopieën werd naar het huisadres van de student gestuurd, bij wijze van oproep.

Het zal rond 1978 geweest zijn dat er een nieuwe medewerkster voor de studentenadministratie in dienst kwam, Carla Reuvecamp. Natuurlijk kostte het ook haar enige tijd om alle medewerkers te leren kennen, zeker degenen die in deeltijd werkten en Jaco was destijds veel jonger. Ik geloof dan ook niet dat iemand het haar heeft kwalijk genomen dat ze, toen Jaco enigszins schuchter (alweer een nieuwe medewerkster) aan de balie stond met een tentamenbriefje in z'n hand, aan hem vroeg "Kan ik je helpen? Wil je tentamen aanvragen?".

Marja Hegt

Predestination and Invariants of Specifications

Wim H. Hesselink, 1st May 2002
Dept. of Mathematics and Computing Science, Rijksuniversiteit Groningen
P.O.Box 800, 9700 AV Groningen, The Netherlands
E-mail: wim@cs.rug.nl, Web: www.cs.rug.nl/~wim

Dedicated to Jaco de Bakker, who combined nondeterminism and predestination at the CWI and the Vrije Universiteit.

1 Introduction

It is generally accepted that specifications must be as concise as possible, unbiased by the form of potential implementations. In particular, progress conditions and absence of deadlock can be specified without any regard for the implementation. In the refinement approach to program design, a specification is replaced step by step by more concrete specifications until it is implementable. It follows that an intermediate specification with abstract progress properties may have to implement a more abstract specification. The proof of this implementation relation may require analysis of the states of the intermediate specification, i.e., analysis of its invariants.

There are several concepts of invariants. A *strong invariant* is a property that holds initially and that is preserved in every step of the program. Such a property is sometimes called *inductive* [9]. A program state is called *reachable* iff it can be reached by program steps from an initial state. Let a property that holds in all reachable states be called a *forward invariant*. It is clear that every strong invariant is a forward invariant.

We want to allow specifications with additional clauses that forbid deadlock. Such a specification may have reachable deadlock states and also reachable states in which future deadlock is unavoidable. These states, though reachable, never occur in admissible behaviours of the specification.

We thus define a state to be *occurring* when it occurs in some admissible behaviour of the specification. We define an *invariant* to be a property that holds in all occurring states. Since every occurring state is reachable, every forward invariant is an invariant.

Note that we only exclude states where future deadlock is inevitable, not where future deadlock is possible. In open systems, one may also want to exclude states in which the system cannot prevent the environment to enforce deadlock. This is the point in Dijkstra's classical Banker's Algorithm [2]. In this note we only consider closed systems.

The term predestination in the title thus refers to the fact that some reachable states do not occur since they inevitably lead to forbidden behaviour. Since the use of predestination may seem questionable to many readers, we give tiny toy examples for which soundness can be verified easily.

In [5], we introduced eternity variables as an alternative for the prophecy variables of [1]. Either type of variables have a kind of prescient behaviour. Treating prescient behaviour by assertional means requires the extension of the use of invariants to predestination as we do in this note. Yet, the concept of predestination is independent of any type of auxiliary variables.

We follow Lamport's approach [7] and regard deadlock not as a safety property but as a liveness property. We therefore allow stuttering steps and model deadlock as a continuous sequence of stuttering steps. This has the advantage that deadlock is essentially the same as livelock, a continuous sequence of internal actions, and just a specific way to violate the fairness requirements.

Our investigation was triggered by a specification we are developing for the serializable database interface of [6, 11] and, in particular, by the proof of its implementation.

Overview. Some standard material on relations and lists is given in section 2. In section 3, we introduce specifications following [1] and then define invariants of various flavours. In section 4, we define simulations and characterize invariants by the induced simulations. In section 5, we discuss the question how to find invariants in an abstract setting. The concrete setting of invariants for programs is sketched in section 6. We conclude in section 7.

2 Relations, lists, and properties

We treat a binary relation as a set of pairs. So, a binary relation between sets X and Y is a subset of the cartesian product $X \times Y$. We use the functions *fst* and *snd* given by $fst(x,y) = x$ and $snd(x,y) = y$. A binary relation on X is a subset of $X \times X$. The identity relation 1_X on X consists of all pairs (x,x) with $x \in X$. Recall that a binary relation A on X is called *reflexive* iff $1_X \subseteq A$.

We use lists to represent consecutive values during computations. The elements of a list xs are xs_i for $i \geq 0$. If X is a set, we write X^ω for the set of infinite lists over X. Any binary relation $F \subseteq X \times Y$ induces a binary relation $F^\omega \subseteq X^\omega \times Y^\omega$ between the infinite lists, given by

$$(xs, ys) \in F^\omega \quad \equiv \quad (\forall\, i :: (xs_i, ys_i) \in F) \ .$$

A finite list xs is called *stutterfree* iff every pair of consecutive elements differ. For infinite lists xs and ys, we define xs to be an *unstuttering* of ys, iff xs is obtained from ys by deleting some consecutive duplicates. For example, with $X \supseteq \{a,b,c\}$, the list $(abbc)^\omega$ is an unstuttering of $(aaabbbc)^\omega$.

A subset P of X^ω is called a *property* [1] over X iff, whenever xs is an unstuttering of ys, we have $xs \in P$ if and only if $ys \in P$. We write $\neg P$ to denote the complement (negation) of a property P. We write $Suf(xs)$ to denote the set of infinite suffixes of an infinite list xs. We define $\Box P$ (always P), and $\Diamond P$ (sometime P) as the properties given by

$$\begin{array}{lcl} xs \in \Box P & \equiv & Suf(xs) \subseteq P \ , \\ \Diamond P & = & \neg\Box\neg P \ . \end{array}$$

For $U \subseteq X$ and $A \subseteq X \times X$, the subsets $[\![U]\!]$ and $[\![A]\!]$ of X^ω are defined by

$$\begin{array}{lcl} xs \in [\![U]\!] & \equiv & xs_0 \in U \ , \\ xs \in [\![A]\!] & \equiv & (xs_0, xs_1) \in A \ ; \end{array}$$

$[\![U]\!]$ is always a property; $\Box[\![A]\!]$ is a property when A is reflexive.

3 Specifications and invariants

A *specification* is defined to be a tuple $K = (X, Y, N, P)$ where X is a set, Y is a subset of X, N a reflexive binary relation on X, and P is a property over X. The set X is called the *state space*, its elements are called *states*, the elements of Y are called *initial states*. Relation N is called the *next-state* relation. The set P is called the *supplementary* property.

We define an *execution* of K to be a nonempty list xs over X for which every pair of consecutive elements belongs to N. An execution of K is called *initial* iff $xs_0 \in Y$. We define a *behaviour* of K to be an infinite and initial execution xs of K with $xs \in P$. We write $Beh(K)$ to denote the set of behaviours of K.

It is easy to see that $Beh(K) = [\![Y]\!] \cap \Box[\![N]\!] \cap P$. It follows that $Beh(K)$ is a property. The requirement that relation N is reflexive is imposed to allow stuttering: if xs is a behaviour of K, any infinite list ys obtained from xs by repeating elements of xs or by removing subsequent duplicates is also a behaviour of K.

A subset $D \subseteq X$ is called a *strong invariant* iff $Y \subseteq D$ and $N \cap (D \times X) \subseteq D \times D$ (i.e. $x' \in D$ for every pair $(x,x') \in N$ with $x \in D$). A state is called *reachable* iff it occurs in an initial execution. A subset of X is called a *forward invariant* iff it contains all reachable states. A state is called *occurring* iff it occurs in some behaviour. A subset of X is called an *invariant* iff it contains all occurring states.

It is easy to verify that a strong invariant contains all reachable states and is therefore a forward invariant. Also, since all occurring states are reachable, every forward invariant is an invariant.

Example. Reachable states need not be occurring and an invariant need not be a forward invariant. This is shown by the following program

var x : *Int* := 0 ;
do x = 0 → *choose* x *in Int* ;
[] *true* → x := x − 2 ;
od ;
prop: infinitely often x = 0 .

Formally, the specification is $K = (X, Y, N, P)$ where X is the set of the integers and $Y = \{0\}$. Relation $N \subseteq X \times X$ is given by

$$(x, x') \in N \quad \equiv \quad x = 0 \ \vee \ x' = x - 2 \ \vee \ x' = x \ .$$

The third disjunct serves to allow stuttering. Property P consists of the infinite sequences with infinitely many zeroes, i.e. $P = \Box\Diamond[\![Y]\!]$. Since N allows jumps from 0 to any integer, all integers are reachable. Therefore X itself is the only forward invariant. Since every behaviour contains infinitely many zeroes, the only occurring states are the even natural numbers. So, the even natural numbers form an invariant. The set of the natural numbers is also an invariant. □

Recall that specification K is defined to be *machine closed* [1] iff every finite initial execution of K can be extended to a behaviour of K. Let K be machine closed. Then every reachable state is occurring. Therefore a subset of X is a forward invariant if and only if it is an invariant.

Example. The converse is not true. We give an example of a specification that is not machine closed while every reachable state is occurring.

var x : {0, 1} := 0 ;
do *true* → *choose* x *in* {0, 1} **od** ;
prop: once x decreases, it remains zero.

Formally, we take $K = (X, Y, N, P)$ with $X = \{0, 1\}$ and $Y = \{0\}$ and $N = X \times X$ while P consists of the infinite bit streams x with

$$(\forall\, i, j, k :: i < j < k \ \wedge \ x_i > x_j \quad \Rightarrow \quad x_k = 0) \ .$$

Both states 0 and 1 are occurring, but the initial execution $(0, 1, 0, 1)$ cannot be extended to a behaviour. In this case, X itself is the only invariant. □

4 Simulations and invariants

Invariants are important since they allow the state space to be restricted without hindrance to the behaviours. Restricting the state space is a special case of simulation of one specification by another.

When discussing several specifications, it is useful to denote the components of specification $K = (X, Y, N, P)$ by $\mathit{states}(K) = X$, $\mathit{start}(K) = Y$, $\mathit{step}(K) = N$ and $\mathit{prop}(K) = P$.

Let K and L be specifications. Recall that a relation F between $\mathit{states}(K)$ and $\mathit{states}(L)$ induces a relation F^ω between the sets $(\mathit{states}(K))^\omega$ and $(\mathit{states}(L))^\omega$. Relation F is called a *simulation* $K \twoheadrightarrow L$ iff, for every behaviour $xs \in \mathit{Beh}(K)$, there exists a behaviour $ys \in \mathit{Beh}(L)$ with $(xs, ys) \in F^\omega$. We refer to [5] for some examples of simulations.

The easiest examples of simulations are forward or downward simulations [3, 8], which go back at least to [10]. They are defined as follows.

A relation F between $\mathit{states}(K)$ and $\mathit{states}(L)$ is defined to be a *forward simulation* from specification K to specification L iff

(H0) For every $x \in \mathit{start}(K)$, there is $y \in \mathit{start}(L)$ with $(x, y) \in F$.
(H1) For every pair $(x, y) \in F$ and every x' with $(x, x') \in \mathit{step}(K)$, there is y' with $(y, y') \in \mathit{step}(L)$ and $(x', y') \in F$.
(H2) Every infinite initial execution ys of L with $(xs, ys) \in F^\omega$ for some $xs \in \mathit{Beh}(K)$ satisfies $ys \in \mathit{prop}(L)$.

It is easy to verify and, in fact, well-known that every forward simulation F from K to L is a simulation $K \twoheadrightarrow L$. This justifies the terminology.

Let K be a specification and let D be a subset of $states(K)$. Then we can define the specification K_D by $states(K_D) = D$ and $start(K_D) = D \cap start(K)$ and $step(K_D) = D^2 \cap step(K)$ and $prop(K_D) = D^\omega \cap prop(K)$. Indeed, it is easy to verify that $step(K_D)$ is reflexive and that $prop(K_D)$ is a property. The following result characterizes invariants via simulations.

Lemma. (a) The identity relation 1_D is a simulation $K \twoheadrightarrow K_D$ if and only if D is an invariant.
(b) $1_D \subseteq K \times D$ is a forward simulation $K \twoheadrightarrow K_D$ if and only if D is a strong invariant.

We skip the proof, since it is fairly straightforward and not interesting.

5 How to find invariants

In the remainder of this note, we consider a single specification $K = (X, Y, N, P)$.

How to find invariants of K? First, recall that a set of states D is said to be *stable* iff every step starting in D remains in D, i.e., iff we have $x' \in D$ for every pair $(x, x') \in N$ with $x \in D$. A subset D of X is strong invariant iff it is stable and contains the initial set Y. A subset U of X is called an *attractor* iff $Beh(K) \subseteq \Box\Diamond[\![U]\!]$. A subset D of X is called a *backward invariant* iff D contains an attractor and its complement $X \setminus D$ is stable.

The following result describes the main methods to prove that a set of states is an invariant.

Lemma 1. (a) Every strong invariant is an invariant.
(b) Every set of states that contains an invariant is itself an invariant.
(c) Every intersection of a family of invariants is an invariant.
(d) Every backward invariant is an invariant.

Proof. Part (a) was proved in section 3 immediately after the definition of invariant. Parts (b) and (c) are immediate from the definition of invariant.

Part (d). Let D be a backward invariant that contains the attractor U. We have to prove that D contains all occurring states. So, let x be occurring. Then there is a behaviour xs of K and a number n with $x = xs_n$. Since U is an attractor, there is a number $i \geq n$ with $xs_i \in U \subseteq D$. Now the complement $X \setminus D$ is stable. Therefore, $xs_n \notin D$ would imply $xs_i \notin D$. This proves $x = xs_n \in D$. □

The next result shows that Lemma 1 is in a certain sense a semantically complete method to prove that sets are invariants.

Let specification K be called *history complete* iff every state x has precisely one stutterfree initial execution that ends in x. It is known, [5, 8], that every specification has an essentially equivalent history complete specification, the so-called unfolding.

Theorem. (a) The set of reachable states of K is a strong invariant. Therefore, every forward invariant contains a strong invariant.
(b) Let K be history complete. Then the set of occurring states of K is a backward invariant. Therefore, every invariant contains a backward invariant.

Proof. (a) This is easy and, in fact, well-known.

(b) Write J for the set of occurring states of K. We have to prove that J is a backward invariant, i.e., that J contains an attractor and that its complement is stable. We first observe that $Beh(K) \subseteq \Box[\![J]\!] \subseteq \Box\Diamond[\![J]\!]$. This implies that J is an attractor and, hence, contains itself as an attractor.

It remains to show that the complement of J is stable. Consider $(x, y) \in step(K)$ with $x \notin J$. We have to prove $y \notin J$. Assume $y \in J$. Then $y \neq x$ and $y = ys_n$ for some behaviour ys of K and some $n \in \mathbb{N}$. By unstuttering we can establish that the prefix $(ys_0, \ldots, ys_n)$ is a stutterfree initial execution that ends in y. Since K is history complete, K has a stutterfree initial execution that ends in x. This execution can be extended to y and yields a stutterfree initial execution that ends

in y. Since K is history complete, it follows that $n > 0$ and $x = ys_{n-1}$. This proves that $x \in J$, yielding a contradiction. □

6 Invariants and programming

The theory is most easily formulated in terms of sets of states, but for programming it is more convenient to use predicates, i.e., boolean functions on the state space. We therefore identify a predicate Q with the corresponding set $(Q) = \{x \in X \mid Q(x)\}$. Predicate Q is called stable, invariant, etc., if and only if the set (Q) is stable, invariant, etc. When E is some boolean function on a set Z, we define $Z \models E$ to mean that all elements $z \in Z$ satisfy $E(z)$.

In order to ease our calculations, we use the following notation, due to Pnueli. Recall that $K = (X, Y, N, P)$. For some set Z and a state function $g : X \to Z$, we define $g^0 = g \circ fst : N \to Z$ and $g^+ = g \circ snd : N \to Z$. By convention, the superscript ^{0} is omitted; this may seem questionable, but it is essential to keep applications of e.g. Lemma 2 below manageable.

As a first application, we see that a predicate Q is stable iff $N \models (Q \Rightarrow Q^+)$. It is a strong invariant if, moreover, Q holds initially, i.e., $Y \models Q$.

The classical way to prove that a family of predicates $(J_i)_i$ is a family of invariants (e.g., see [4] 3.1) is to prove

$$Y \models (\forall\, i :: J_i)\ ,$$
$$(\forall\, k :: N \models (\forall\, i :: J_i) \Rightarrow J_k^+)\ .$$

The first condition states that the conjunction $J = (\forall\, i :: J_i)$ holds initially. The second condition implies that J is stable. It follows that J is a strong invariant, so that all conjuncts J_i are (forward) invariants.

As a direct translation of the set-theoretic attractors above, we say that a predicate A is an *attractor of the first kind* iff $A(xs_n)$ holds infinitely often for every behaviour xs. A predicate Q is a backward invariant if and only if Q (contains, i.e.) is implied by some attractor of the first kind and its negation $\neg Q$ is stable.

In practice, we sometimes need a more flexible way to prove invariance. A boolean function A on the step relation N is called an *attractor of the second kind* iff $Beh(K) \subseteq \Box\Diamond[\![A]\!]$.

Lemma 2. Recall that $K = (X, Y, N, P)$. Let J be a predicate on X and let vf be an integer-valued state function on X. Assume that A is an attractor (of first or second kind) and that D is an invariant. Further assume:

(a) $\quad X \models \neg J \ \wedge\ D \ \Rightarrow\ vf \geq 0$,
(b) $\quad \neg J \ \wedge\ D \ \wedge\ vf \leq n$ is stable for every $n \in \mathrm{IN}$,
(c) $\quad N \models A \ \wedge\ \neg J \ \wedge\ D \ \wedge\ vf \leq n \ \Rightarrow\ vf^+ < n$ for every $n \in \mathrm{IN}$.

Then J is an invariant.

Proof. We prove that J holds in every state of any given behaviour. Let xs be a behaviour and k an index with $J(xs_k) = false$. Since D is an invariant, condition (a) implies that $vf(xs_k) \geq 0$. By conditions (a) and (b), the sequence $vf(xs_i)$ with $i \geq k$ is descending and nonnegative. Since A is an attractor, there are infinitely many indices $i \geq k$ with $A(xs_i)$. Therefore, condition (c) implies that the sequence decreases infinitely often and therefore has negative values. This is a contradiction. □

Example. Consider the program

var k : *Nat* := 0 ; m : *Nat* := 0 ;
do k ≤ 2 · m → k := k + 1 ;
[] m + 1 = k → m := m + 1 ;
od ;
prop: k increases infinitely often.

It has the specification $K = (X, Y, N, P)$ where $X = \mathrm{IN} \times \mathrm{IN}$ and $Y = \{(0,0)\}$ and

$$\begin{array}{l}((k,m),(k',m')) \in N \quad \equiv \\ \quad (k \leq 2 \cdot m \quad \wedge \quad k' = k+1 \quad \wedge \quad m' = m) \\ \quad \vee \quad (m+1 = k = k' = m') \quad \vee \quad (k = k' \quad \wedge \quad m = m') \ .\end{array}$$

The third alternative serves to allow stuttering. The liveness property that $\mathtt{k}$ increases infinitely often is expressed in $P = \Box\Diamond[\![\mathtt{k} < \mathtt{k}^+]\!]$.

Intuitively, it is clear that $\mathtt{k}$ can stay growing only when $\mathtt{m}$ follows it closely. We therefore guess the invariant $J : \ \mathtt{k} \leq \mathtt{m}+1$. We prove invariance of J by Lemma 2 with variant function $vf = 2 \cdot \mathtt{m} + 1 - \mathtt{k}$ and invariant $D : \mathtt{k} \leq 2 \cdot \mathtt{m} + 1$. Indeed, D is an invariant and it implies $vf \geq 0$. Outside of J, the value of $\mathtt{m}$ cannot increase; this implies condition (b). Whenever $\mathtt{k}$ increases, vf decreases. We can thus use the attractor (of the second kind) $A : \mathtt{k} < \mathtt{k}^+$. □

7 Conclusion

When considering refinements for specifications with supplementary properties, it is useful to extend the concept of invariant, so that states can be excluded when they cannot occur in admissible behaviours. Traditionally, invariants are proved by inductive (forward) reasoning. The new kind of invariants also need backward reasoning. We have presented a useful proof rule for these invariants, which is based on a variant function such as customary for termination proofs.

References

[1] Abadi, M., Lamport, L.: The existence of refinement mappings. Theoretical Computer Science **82** (1991) 253–284.

[2] Dijkstra, E.W.: Co-operating sequential processes. In: F. Genuys (ed.): Programming Languages (NATO Advanced Study Institute). Academic Press, London etc. 1968, pp. 43–112.

[3] He, J., Hoare, C.A.R., Sanders, J.W.: Data refinement refined. In: Robinet, B., Wilhelm, R. (eds.): *ESOP86* pp. 187–196. Springer Verlag, 1986 (LNCS 213).

[4] Hesselink, W.H.: The verified incremental design of a distributed spanning tree algorithm: extended abstract. Formal Aspects of Computing **11** (1999) 45–55

[5] Hesselink, W.H.: Eternity variables to simulate specifications. Mathematics of Program Construction 2002 (to appear). `www.cs.rug.nl/~wim/pub/whh263t.pdf`

[6] Lamport, L.: Critique of the Lake Arrowhead three. Distributed Computing **6** (1992) 65–71.

[7] Lamport, L.: The temporal logic of actions. ACM Trans. on Programming Languages and Systems **16** (1994) 872–923.

[8] Lynch, N., Vaandrager, F.: Forward and backward simulations, Part I: untimed systems. Information and Computation **121** (1995) 214–233.

[9] Manna, Z., and A. Pnueli: Temporal Verification of Reactive Systems: Safety. Springer V. 1995.

[10] Milner, R.: An algebraic definition of simulation between programs. In: Proc. 2nd Int. Joint Conf. on Artificial Intelligence. British Comp. Soc. 1971. Pages 481–489.

[11] Schneider, F.B.: Introduction. Distributed Computing **6** (1992) 1–3.

Zesendertig Jaar Collega´s

Beste Jaco,

In 1964 kwamen we beiden naar het Mathematisch Centrum in de Boerhaavestraat om onze wetenschappelijke loopbaan te beginnen, jij bij de Rekenafdeling, ik bij de afdeling Toegepaste Wiskunde. Zesendertig jaar lang, tot mijn vertrek in 2000, zijn we collega-onderzoekers geweest, waarvan 24 jaar ook nog collega-afdelingschefs.

In de eerste 4 jaar zullen we elkaar ongetwijfeld wel eens ontmoet hebben, maar ik denk niet dat we elkaar gesproken hebben, hoogstens een groet op de gang. Voor zover ik me kan herinneren vond ons eerste echte gesprek plaats in de zomer van 1968. Dat was een gevolg van de afwezigheid van Dirk Dekker, souschef van de Numerieke sectie van de Reken-afdeling. In de zomer van 1968 ging Dirk voor een jaar naar Amerika. In zijn sectie was een belangrijke taak het draaiende houden van de zogenaamde Winkel. Ik kreeg opdracht gedurende Dirk's afwezigheid op die Winkel te passen.
In de Winkel werden allerlei programmeeropdrachten uitgevoerd, voornamelijk voor derden, maar in een poging om de "klanten" zoveel mogelijk zelf te laten programmeren, werden er ook programma-adviezen verzorgd. Dit werd in het algemeen door de onderzoekers als een corvee werd ervaren en daarom werd de klus, zo vertelde men mij, over alle leden van zowel de Numerieke als de Programmeersectie van de Rekenafdeling verdeeld. Als vervanger van Dirk moest ik een rooster opstellen en daarvoor ging ik de leden van de Rekenafdeling af. Zo kwam ik ook bij jou terecht voor ons eerste echte contact.
Voor ik jouw kamer binnen liep was ik bij je buurman geweest en was daar nogal koel ontvangen, dus ik hield er al rekening mee dat ook jij niet zat te wachten om bij iets als programma-adviezen ingeschakeld te worden. Te meer omdat jij gepromoveerd was en al enige naam in de wetenschap gemaakt had. Ik besloot om je voor de vorm het rooster voor te leggen en als je bezwaren had, me er direct bij neer te leggen. Maar tot mijn verrassing maakte je geen enkel bezwaar. Je vond het vanzelfsprekend je bijdrage aan die klus te leveren!

Zoals ik van dichtbij heb kunnen waarnemen, is dat altijd typerend geweest voor je houding ten opzichte van ons instituut. In 1973 werden de Programmeersectie en de Numerieke sectie zelfstandige afdelingen, de afdelingen Informatica en NumeriekeWiskunde, waar we respectievelijk de afdelingschefs van werden. In die functie heb ik je echt goed leren kennen. Via de vele vergaderingen van de diverse raden, commissies, etc., waar we zitting in hadden, zagen we elkaar regelmatig. In die vergaderingen was het wel en wee van het MC/CWI vaak onderwerp van gesprek. In de discussies voelde je je altijd als een vis in het water en je was in staat, wanneer de discussie verwarrend werd, door een haarscherpe analyse de zaken weer voor iedereen duidelijk te maken. Ik heb je door de jaren heen als een echte instituutsman leren kennen, waarvoor het belang van het hele instituut altijd voorop stond. En niet alleen met woorden, ook met daden. Mede dankzij de talloze extern betaalde onderzoeksprojecten die jij met je afdeling in de wacht wist te slepen, heeft het CWI in moeilijke jaren wetenschappelijk en financieel het hoofd boven water weten te houden.

Ook onze meer persoonlijke contacten dateren uit onze beginjaren als chef van AI en NW. Aly en ik weten het niet meer precies, maar het zal iets van midden zeventiger jaren zijn geweest dat we met enige regelmaat bij elkaar over de vloer kwamen. Ik kan me nog herinneren dat we bij een van de eerste dinertjes een gerecht met artisjokkenhartjes voorgeschoteld kregen dat je zelf bereid had. Je maakte toen nog de opmerking dat het de volgende keer mijn beurt was. Gelukkig ben je er niet op terug gekomen, in de keuken voel ik me niet zo thuis. Mijn aandeel in de etentjes bij ons thuis bestaat uitsluitend uit het kiezen van de wijn.

In de periode zeg 1975 - 2002 hebben we elkanders kinderen leren kennen. Met name jullie jongste zoon en dochter hebben we echt zien opgroeien. We herinneren ons Jacob - nu al student - als een buitengewoon ondernemende kleuter en van Lisa zijn vooral de prachtige tekeningen die ze als klein meisje in Artis maakte ons bijgebleven. Op mijn afscheid kozen jullie ervoor bij onze drie dochters aan tafel te gaan zitten om zo de kennismaking weer te hernieuwen. Dat hebben ze, en wij ook, erg leuk gevonden.

We hadden dit stukje graag willen opsieren met een paar foto's, maar we hebben niet zoveel foto's van jullie en de foto's die we hebben zijn al gebruikt in eerdere Libers. Het enige wat we hebben kunnen vinden is een foto, niet van jullie, maar van de Italiaanse wijnen die we met z'n vieren in de periode 1984 (toen begon ik met mijn wijnhobby) en 1989 hebben gedronken. Die foto heb ik indertijd genomen om een stukje te illustreren dat we voor je 25-jarig Jubileum Liber hadden gemaakt, maar hij is er niet in terecht gekomen, en ik kreeg hem later terug. Hierboven de bewuste foto als een indirecte herinnering aan de vele bijzonder gezellige avonden die we met elkaar doorgebracht hebben. De traditie van de etentjes over en weer moeten we zeker voortzetten.

Jaco en Angeline, waar we meer ervaren in zijn dan jullie, dat is met pensioen zijn. Na twee jaar pensioenervaring kunnen we zeggen dat deze nieuwe fase in het leven veel te bieden heeft en we zijn er ook zeker van dan jullie er volop van zullen genieten.

Aly en Piet van der Houwen.

On combining traditions

J.-M. Jacquet

July 22, 2002

Abstract

While I was finishing my PhD thesis in 1988, I noticed a very interesting paper by Jaco De Bakker and Joost Kok on metric semantics of Concurrent Prolog ([3]). Fascinated by the ideas, I thought that it would be very wise to spend some time on getting familiar with the framework. Thanks to an open position in the ESPRIT project Integration, I had the opportunity to spend 2 years and a half at CWI in Jaco's group, an experience which I appreciate very much 10 years later.

When I arrived at CWI in September 1989, I immediately noticed two cultures: the traditional concurrency community was essentially studying languages based on synchronous communication while the concurrent constraint programming community was promoting asynchronous communication. When you are young, a polite rule is not to discuss foundations proposed by experts. Later, together with L. Brim, D. Gilbert, and M. Kretinsky, I realize that imposing asynchronous communication in concurrent constraint programming is not adequate in all situations.

This paper briefly reports on a new form of communication in concurrent constraint programming. It actually combines both asynchronous and synchronous communication. Of course, being initiated to metric semantics by Jaco, it has been a pleasure to develop a variety of metric semantics for that new framework. Due to the lack of space, I refer the reader to [1, 2] for information on these semantics as well as for a comparison with related work.

1 Concurrency and constraint programming

Concurrent constraint programming has emerged as an important paradigm for concurrent computations. It is based on the idea of computing with partial information placed on a shared space, called the store. Accordingly, concurrent processes communicate through this store by telling pieces of information and by asking whether some piece of information is entailed by the current contents of the store. As a legacy of previous proposals for concurrent logic programming languages (Concurrent Prolog, Parlog, GHC, etc.), communication occurs in an *asynchronous* fashion: tell actions are always allowed to proceed whereas ask operations are blocked when the information on the store is not complete enough to entail the asked constraints.

Following these lines, a natural way of obtaining synchronous communication in concurrent constraint programming is to force the reduction of ask and tell primitives to synchronise. Specifically, our approach considers tell primitives as lazy producers of information and views ask primitives as consumers of this information. From this point of view, a tell operation is reduced when an ask operation requires the told information. Moreover, the reduction of the two primitives is performed simultaneously. However, there is no reason to block ask and tell primitives on information which is already present. Consequently, stress is put on the novelty of the information and hence any `tell`(c) and `ask`(c) operations whose constraint argument c is entailed by the current store are reduced without partners.

This framework, called `Scc`, has been introduced in [1] and its expressiveness has been demonstrated through the coding of a variety of examples. It has been argued that one advantage over related work such as [5, 6, 8], which introduce synchronisation by special operators and not by altering the behaviour of tell and ask primitives, is that `Scc` permits the specification of on *what* information the synchronisation should be made, rather than with *whom*. Synchronisation in `Scc` is thus data-oriented as opposed to process-oriented.

2 A need for novel treatments

In order to motivate its interest and to substantiate the need for novel treatments, it is worth stressing the behavioural difference of `Scc` with, on the one hand, traditional concurrent constraint programming, as exemplified in the `cc` family of languages ([8]), and, on the other hand, traditional concurrent programming models, as exemplified by CCS ([7]).

It has been argued in [4] that the main difference between CCP and CCS is that complementary actions do not synchronise in CCP. This property is due to the fact that telling a constraint never suspends in cc. In contrast, the action of telling a constraint may suspend until an ask can make use of it. A synchronisation similar to that in CCS is thus produced. However, this synchronisation does not hold in `Scc` in the case that the told or asked constraints are entailed by the current contents of the store. A novel kind of synchronisation is thus achieved.

Major differences appear between the three frameworks. It is to be expected that these differences call for new treatments as well. In order to formalise our reasoning somewhat, let us turn to the example given in [4]. There CCS and CCP are compared by interpreting the action a as telling the constraint $x = a$, and the co-action $\overline{a}$ as asking the constraint $x = a$. To keep our notations consistent, we shall use "+" for the non-deterministic choice operator and ";" for the sequential composition operator.

Example 1 (Differentiating CCP and CCS (from [4])) *Let* $A_1 = (\overline{a}; \overline{b}) + (\overline{a}; \overline{c}) + (\overline{a}; \overline{d})$ *and* $A_2 = (\overline{a}; \overline{b}) + (\overline{a}; (\overline{c} + \overline{d}))$. *In any compositional semantics for CCS these two processes must be distinguished. Indeed, they behave differently*

under the context $A = a;(b + c)$. The process A_1 can deadlock, by choosing the third alternative of the choice, while A_2 cannot. However, in cc, *both A_1 and A_2 have the same behaviour. The process A_2 can deadlock by choosing the second alternative, because A can independently decide to produce $y = b$ (after $x = a$).*

Example 2 (Differentiating CCP and Scc) *Using the processes A, A_1, A_2 of the above example, the processes A_1 and A_2 are also distinguished by A in* Scc *for the same reason as in CCS.*

This example illustrates the difference between Scc and cc. Stated in other terms, in the CCP paradigm, since the tell operation is asynchronous, the choice guarded by $tell(a)$ is a *local choice* whereas, since the tell operation is synchronous in Scc, this choice is *global* in Scc.

Nevertheless, synchronisation is only forced in Scc in the case that a process tries to tell information which is not already entailed by the store. Otherwise, it can proceed asynchronously. This fact is used subsequently to differentiate CCS and Scc.

Example 3 (Differentiating CCS and Scc) *Using again the above processes A, A_1, A_2, let $B_1 = \bar{b}; A_1$ and $B_2 = \bar{b}; A_2$. In CCS, these two processes can be distinguished by the process $B = b; A$ for the reasons exposed in Example 1. However, in* Scc, *both processes have the same behaviour. The process B_2 can now deadlock by choosing the second alternative because A can now independently proceed by the first alternative as $y = b$ is already entailed by the store.*

The distinction between Scc and CCS thus appear to be more subtle than the distinction between CCP and CCS. The choice guarded by `tell` is actually a "mixture" of global and local choice. The choice depends upon actions performed and upon the results of the past behaviour of the system, i.e. upon the constraint contained in the store.

References

[1] L. Brim, D. Gilbert, J.-M. Jacquet, and M. Křetínský. A Process Algebra for Synchronous Concurrent Constraint Programming. In M. Hanus and M. Rodriguez-Artalejo, editors, *Proceedings of the 5^{th} Conference on Algebraic and Logic Programming*, volume 1139 of *Lecture Notes in Computer Science*, pages 165–178. Springer Verlag, 1996.

[2] L. Brim, D. Gilbert, J.-M. Jacquet, and M. Křetínský. Multi-agent Systems as Concurrent Constraint Processes. In L. Pacholski and P. Ruzika, editors, *Proc. SOFSEM 2001: Theory and Practice of Informatics*, volume 2234 of *Lecture Notes in Computer Science*, pages 201–210. Springer-Verlag, 2001.

[3] J.W. de Bakker and J.N. Kok. Uniform abstraction, atomicity and contractions in the comparative semantics of concurrent prolog. In *Proc. Fifth Generation Computer Systems (FGCS 88)*, pages 347–355. Ohmsa ltd and Springer Verlag, 1988.

[4] F. S. de Boer and C. Palamidessi. A Fully Abstract Model for Concurrent Constraint Programming. In S. Abramsky and T.S.E. Maibaum, editors, *Proc. of TAPSOFT/CAAP91*, Lecture Notes in Computer Science, pages 296–319. Springer-Verlag, 1991.

[5] M. Falaschi, G. Levi, and C. Palamidessi. A Synchronization Logic: Axiomatics and Formal Semantics of Generalized Horn Clauses. *Information and Control*, 60:36–69, 1994.

[6] J.-M. Jacquet and L. Monteiro. Communicating Clauses: Towards Synchronous Communication in Contextual Logic Programming. In Krzysztof Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 98–112, Washington, USA, 1992. The MIT Press.

[7] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[8] V. Saraswat. *Concurrent Constraint Programming.* The MIT Press, 1993.

A Scientist's Life

a spring well
a frog jumps in
the sound of water

the summer river
where is the bridge?
wet feet

autumn leaves
the storm blows away
the words

a frozen pond
hibernating fish
gazing at the moon

In all seasons Jaco de Bakker stimulated computer science at CWI. I dedicate this to him on the occasion of his retirement. Paul Klint

A wonderful stream

for Jaco

Jan Willem Klop

This note is written for Jaco de Bakker, in the hope that it may entertain him, and by way of thanks for all the years that I was working in his department or cluster, in a stimulating and productive environment created by Jaco's calm and effective leadership.

As I noticed recently, Jaco is well aware of the existence of the stream figuring in this note, namely the Thue-Morse sequence, to be called M henceforth. So the main aspects of this sequence mentioned below will not surprise him. One feature is maybe not well-known, namely the plane tiling that the stream M induces, and of which a figure is included. Otherwise the sequence M is frequently discovered, and studied in a variety of contexts, including formal languages, combinatorics on words, group theory and symbolic dynamics. Browsing around through the literature, it is amazing how widely and deep this sequence is studied. The references included in this note also contain CWI-connected authors; the current president of ERCIM; and the former dutch chess-world champion Max Euwe. It has even been used to generate some minimal music, see Figure 1, given in Allouche and Johnson [96]. Some quite heavy mathematics is devoted to it. As a disclaimer, it should be said that this note does not add to that more serious matter. But the sequence is a delightful example to illustrate in class-room various notions in term rewriting, process algebra, and algebraic data types. In fact this note arose out of a search for some simple example to treat in a process algebra class, in order to show a certain expressivity result - see below. There are several ways to introduce the sequence M.

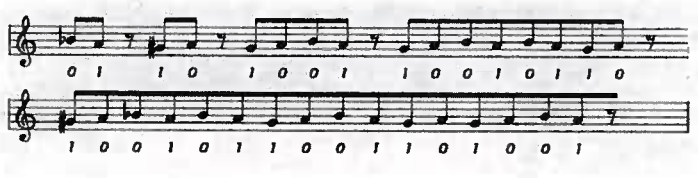


Figure 1. Thue-Morse music

1. Four definitions of M, and some properties. The sequence which is the subject of this note was discovered in 1912 by Axel Thue, one of the founding fathers of the theory of formal languages. It was rediscovered in 1917 by Marston Morse. Actually it occurred already in Prouhet [1851]. My fascination with this sequence started by playing around with symbolic expressions:

α
$\alpha - \alpha$
$(\alpha - \alpha) - (\alpha - \alpha)$
$(\alpha - \alpha) - (\alpha - \alpha) - ((\alpha - \alpha) - (\alpha - \alpha))$
$(\alpha - \alpha) - (\alpha - \alpha) - ((\alpha - \alpha) - (\alpha - \alpha)) - ((\alpha - \alpha) - (\alpha - \alpha) - ((\alpha - \alpha) - (\alpha - \alpha)))$
....,

so in each step subtracting the result of the previous step. Working out the brackets, the +'s and -'s follow a pattern that is in fact the Thue-Morse sequence. It is much easier to write 1 and 0 instead of + and -, and so we obtain the first definition of M:

(i) Start with 1 and append in each generation step the 'negative' of the sequence obtained thus far, where 'negative' means changing a 1 into 0 and 0 into 1. We get

1
10
1001
10010110
1001011001101001
....

and the limit is the infinite stream known as the Thue-Morse sequence M. From this definition it is easy to see that M is, let's call it, an *infinitary palindrome*: each initial part can be extended to a possibly larger segment that is a palindrome. Or otherwise said, M is the limit of palindromes.

(ii) The second definition: M is the result of iterating the morphism $1 \rightarrow 10, 0 \rightarrow 01$, starting with 1. In another terminology (see Saloma [81]) the rules of such a morphism (together with mention of the alphabet Σ and a starting word) form a *DOL system*.

(iii) The third definition: count the number of 1's in the binary representation of n, and take this modulo 2; then we get the negative of M; see Table 1.

(iv) The fourth definition gives the n-th entry in the sequence M, call it ε_n, by the recurrence equations

$$\varepsilon_0 = 1$$
$$\varepsilon_{2n} = \varepsilon_n$$
$$\varepsilon_{2n+1} = 1 - \varepsilon_{2n}$$

The definitions are easily proved equivalent. Now some properties of M, other than the palindrome property noted above. The main property is that M is *cube-free*: it does not contain a subword of the form www. A detailed proof is in Saloma [81]. As explained in Saloma [81], a stronger statement is

true: the Thue-Morse sequence does not even contain a subword of the form wwa, where a is the first symbol of the word w. This is called *strongly cube-free*. Strongly cube-free in turn is equivalent to *overlap-free*, meaning that there is no subword x having two overlapping occurrences.

From the cube-freeness it follows immediately that M is *not eventually periodic*.

M is *self-similar*: each finite subword occurs infinitely many often in the sequence. Also this is easily proved.

0	0	0	0
1	01	1	1
2	10	1	1
3	11	2	0
4	100	1	1
5	101	2	0
6	110	2	0
7	111	3	1
8	1000	1	1
9	1001	2	0
10	1010	2	0
11	1011	3	1

Table 1

Of course M is not *square-free*; a square-free stream of two symbols 0, 1 does not exist. But with three symbols 0,1,2 there are square-free streams, and we obtain one from the negative of M, 0110100110010110... as follows: 01 yields 0, 10 yields 1, 00 yields 2 and 11 yields 2, where 01 yields 0 means that we write a 0 under the first symbol of 01, etc. Thus we obtain:

0110100110010110...
021012021....

and this sequence 021012021.... is according to Morse and Hedlund [1944] square-free. (A proof is in Saloma [81].)

Much more sophisticated properties are in Allouche and Shallit [99], where also the following curious way to obtain the sequence M is mentioned. Let A be the lexicographically smallest set of integers that starts with 0, 1 and for $x \geq 1$, if $x \in A$ then $2x \notin A$. So A =

0, 1, 3, 4, 5, 7, 9, 11, 12, 13, 15, 16, 17, 19, 20, 21, 23, ...

of which the sequence of differences is

1 2 1 1 2 2 2 1 1 2 1 1 2 1 1 2, ...

Using these differences as exponents of alternatingly 0 and 1 yields

$$0^1 1^2\ 0^1 1^1 0^2\ 1^2\ 0^2\ 1^1 0^1 1^2\ 0^1 1^1 0^2 1^1 0^1 1^2 \ldots$$

which indeed is the ubiquitous stream M:

0 11 0 1 00 11 00 1 0 11 0 1 001 0 11...

Table 2 gives the first 256 digits of M, by reading the 16-digits lines consecutively. As this table suggests, we can also consider 'two-dimensional DOL-systems', and consider the result of iterating the morphism with rules

$$1 \rightarrow \begin{matrix} 10 \\ 01 \end{matrix} \qquad\qquad 0 \rightarrow \begin{matrix} 01 \\ 10 \end{matrix}$$

starting from a single 1. We will return to the 'two-dimensional stream' thus obtained later.

2. M and rewriting. It is a nice exercise to define the stream M by rewriting. Actually we need *infinitary rewriting*. There are several solutions to this simple exercise in functional programming.

(i) For the first solution we need auxiliary symbols $\underline{1}, \underline{0}$. A finite word is denoted by x. Now consider rules

$$1x \rightarrow \underline{1}x10,$$
$$0x \rightarrow \underline{0}x01.$$

Then we have $\underline{1}\,0 \rightarrow \underline{1}\,0\,01 \rightarrow \underline{1}\,0\,0\,1\,0\,1 \rightarrow \underline{1}\,0\,0\,1\,0\,1\,1\,0 \rightarrow \underline{1}\,0\,0\,\underline{1}\,0\,1\,1\,0\,0\,1 \rightarrow \ldots$

This works, but the two rewrite or reduction rules do not yet constitute a proper term rewriting system (TRS). However, this can be remedied by conceiving the 0, 1, $\underline{0}$, $\underline{1}$ as unary symbols, employing a constant nil, and the function append, for which rewrite rules are easy to give. Then e.g. the first rule reads

$$1(x) \rightarrow \underline{1}(\text{append}(x, 1(0(\text{nil}))$$

which has the proper TRS format. We note that the infinite reduction sequence of which the first four steps are displayed, satisfies the fundamental requirement in infinitary rewriting, namely that the depth of the contracted redexes tends to infinity. In other words, the reduction sequence is strongly convergent, which guarantees the existence of the infinite limit term.

M			
1 0 0 1	0 1 1 0	0 1 1 0	1 0 0 1
0 1 1 0	1 0 0 1	1 0 0 1	0 1 1 0
0 1 1 0	1 0 0 1	1 0 0 1	0 1 1 0
1 0 0 1	0 1 1 0	0 1 1 0	1 0 0 1
0 1 1 0	1 0 0 1	1 0 0 1	0 1 1 0
1 0 0 1	0 1 1 0	0 1 1 0	1 0 0 1
1 0 0 1	0 1 1 0	0 1 1 0	1 0 0 1
0 1 1 0	1 0 0 1	1 0 0 1	0 1 1 0
0 1 1 0	1 0 0 1	1 0 0 1	0 1 1 0
1 0 0 1	0 1 1 0	0 1 1 0	1 0 0 1
1 0 0 1	0 1 1 0	0 1 1 0	1 0 0 1
0 1 1 0	1 0 0 1	1 0 0 1	0 1 1 0
1 0 0 1	0 1 1 0	0 1 1 0	1 0 0 1
0 1 1 0	1 0 0 1	1 0 0 1	0 1 1 0
0 1 1 0	1 0 0 1	1 0 0 1	0 1 1 0
1 0 0 1	0 1 1 0	0 1 1 0	1 0 0 1

Table 2. First 256 digits of Thue-Morse stream M.

(ii) The second solution begins with finding a TRS for the function ε in the recurrence rules in 1.(iv) above. Let E be the symbol defining ε, so $E(\mathbf{n}) \twoheadrightarrow \boldsymbol{\varepsilon}_{\mathbf{n}}$. (Here **n** is the numeral corresponding to n, and $\twoheadrightarrow$ denotes a finite reduction.) Next, we define the stream $E(\mathbf{0}) : E(\mathbf{1}) : E(\mathbf{2}) : \ldots$ where : is the usual prefix operation, as follows: $H(x) \rightarrow E(x) : H(succ(x))$. Then H(0) has as infinite normal form the desired sequence.

It is interesting that instead of employing the mechanism of infinitary rewriting and infinitary normal forms, we can obtain the same sequence with an appeal to *coinductive techniques*.

(iii) The third solution employs a self-similarity property of M, namely

$$M = 10(t(M) \square i(t(M))).$$

Here t is the *tail* operation that removes the first element of a stream, and i is the operation *'invert'* that takes the 'negative' of a 0-1-stream. Further, $\square$ is the *zip* operation that alternates elements from its left argument with elements from its right argument. (Many more of these properties are mentioned

in Table 6.)

A definition of M exploiting this equation can now easily be given; in Table 3 it is rendered as a functional program in Clean (with thanks to Peter Achten).

```
Start = thue_morse
where
thue_morse = [1,0:zipp (tl thue_morse) (map inv (tl thue_morse))]
inv 0 = 1; inv 1 = 0
zipp [a:as] bs = [a:zipp bs as]
```

Table 3: Definition in Clean of Thue-Morse stream

3. M and process algebra. We can also view the stream M as a *process* performing steps (or actions) 0 and 1. It is an infinite state process; it is not hard to prove that all the tails $t^n(M)$, arising by removing the first n elements, are different. Since equality on streams is the same as bisimilarity, we have that the process M proceeds through infinitely many different states. How can we define M in process algebra, ACP? A theorem in Bergstra and Klop [1984] shows that we cannot define M in PA, that is, without communication. Namely, that theorem states that a process having an infinite branch, and recursively definable in PA (so only with operators +, ., $\|$ and $\lfloor\!\lfloor$) must have a branch which is eventually periodic. Since M has only one branch, which is not eventually periodic, it follows that M is not PA-definable. We do need communication. Indeed, it is not hard to define M in *ACP with renaming*, as follows. We start again from the self-similarity equation above. This yields the guarded system of recursion equations

$$X = 1.Y$$
$$Y = 0.(Y \square i(Y))$$

Next, we express $\square$ by a 'zip process' defined by $\square = blue.red.\square$, and we color (i.e., rename) the left and right argument to be zipped in order to avoid confusion; we take along the operation i (invert) in one stride:

$$X = 1.Y$$
$$Y = 0.(Y^{blue} \square Y^{invred})$$

where we have renamings and communications as follows:

$$blue(0) = 0_{blue}, blue(1) = 1_{blue},$$
$$invred(0) = 1_{red}, invred(1) = 0_{red},$$

$$\text{blue} \mid 0_{\text{blue}} = 0$$
$$\text{blue} \mid 1_{\text{blue}} = 1$$
$$\text{red} \mid 0_{\text{red}} = 0$$
$$\text{red} \mid 1_{\text{red}} = 1$$

Finally:

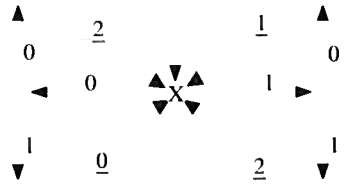
$$X = 1.Y$$
$$Y = 0.\partial_H(Y^{\text{blue}} \parallel \Box \parallel Y^{\text{invred}})$$

where H contains the communication actions.The resulting process X is just M.

There is an interesting question arising here (the answer is not known to the author): can M be defined *in ACP with handshaking communication*? (So without renaming.) We can eliminate the renamings in favour of some more communicating processes, but the catch is that in this way ternary communications arise, while handshaking communication is binary.

A next exercise is to define in process algebra the square-free stream 021012021.... obtained above by applying the "stream transforming rules" 01 yields 0, 10 yields 1, 00 yields 2 and 11 yields 2, on the negative of M. We can capture these rules, also depicted in Figure 2(a), by the process

$$X = 0.(0.\underline{2} + 1.\underline{0})X + 1.(0.\underline{1} + 1.\underline{2})X$$ (see Figure to the right).



where 0 means 'read 0', and $\underline{0}$ means 'write 0', likewise for 1. Now it is easy to obtain the square-free sequence sqf(M) by letting i(M) communicate with X, and applying abstraction:

$$\text{sqf}(M) = (\tau_I \circ \partial_H)(X \parallel i(M))$$

with communications $0|0 = 0$', $1|1 = 1$', and abstracting the communication results 0', 1' away into τ.

Again, it is not known to the author whether we can avoid using abstraction.

4. The Toeplitz stream T. Another interesting stream originates from M by taking the difference sequence (see Figure 2(c). It is called the Toeplitz stream, or the 'period doubling sequence'. We will refer to it as T.

M = 1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 ...
T = 1 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 0 1 0 1 0 1 1 1 0 1 0 1 0 1 1 1 0 1 .....

It can also be generated by iterating a morphism: $1 \to 10, 0 \to 11$, starting from 1.

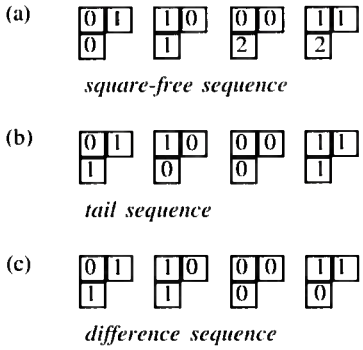


Figure 2: stream transfomation rules

Clearly, T is not cube-free. But it is also self-similar in the sense that every finite part of it is repeated infinitely many times in the sequence. A question that arises is whether the Toeplitz stream is also an infinitary palindrome. Indeed the prefixes of length 1, 3, 7, 15, 31 are palindromes:

1
1 **0** 1
1 0 1 **1** 1 0 1
1 0 1 1 1 0 1 **0** 1 0 1 1 1 0 1
1 0 1 1 1 0 1 0 1 0 1 1 1 0 1 **1** 1 0 1 1 1 0 1 0 1 0 1 1 1 0 1

The midpoints of these palindromes (in boldface) are 1, 0,1,0,1, ... Let us establish that T is indeed the limit of palindromes. The simple inductive proof is suggested by considering the generation tree of T as in Figure 3. Let the morphism ϕ be defined on the set of non-empty 0,1-words $\{0,1\}^+$ by $\phi(1) = 10$, $\phi(0) = 11$, $\phi(uv) = \phi(u)\phi(v)$. Define words α_n, β_n by

$$\alpha_0 = 1,\ \alpha_{n+1} = \phi(\alpha_n)$$
$$\beta_0 = 0,\ \beta_{n+1} = \phi(\beta_n).$$

Then $\alpha_{n+1} = \alpha_n\beta_n$. Further, define words γ_n $(n \geq 1)$:

$$\gamma_{2n} = \gamma_{2n-1}0\gamma_{2n-1} \qquad (n \geq 1)$$
$$\gamma_{2n+1} = \gamma_{2n}1\gamma_{2n} \qquad (n \geq 1)$$

so the words γ_n are palindromes. Now we prove for all $n \geq 1$:

$\alpha_{2n} = \gamma_{2n}1$ $\beta_{2n} = \gamma_{2n}0$

$\alpha_{2n+1} = \gamma_{2n+1}0$ $\beta_{2n+1} = \gamma_{2n+1}1$.

It follows that the limit of the α_n is also the limit of the palindromes γ_n, QED.

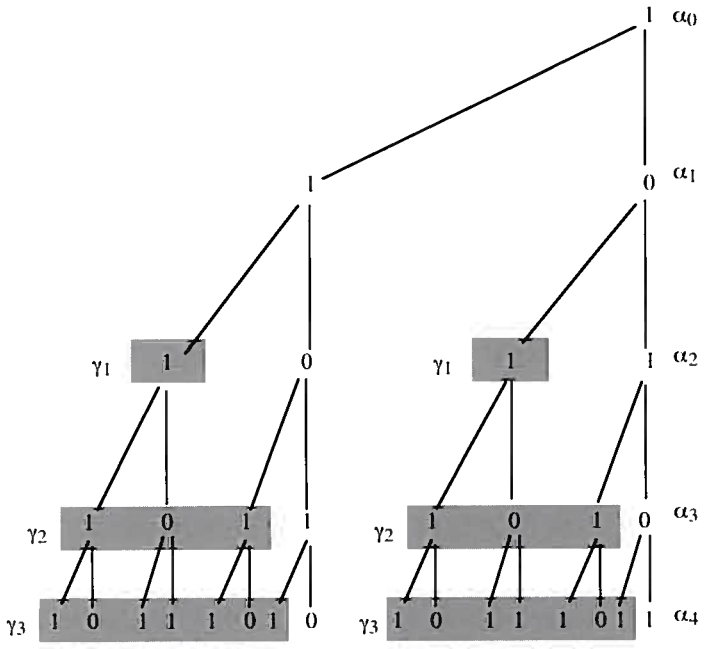


Figure 3. Generation tree of Toeplitz stream

We continue with establishing another property of T. Unzipping T yields

odd(T) = 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 .....

even(T) = * 0 * 1 * 0 * 0 * 0 * 1 * 0 * 1 * 0 * 1 * 0 * 0 * 0 * 1 * 0 * 0 * 0 * 1 * 0 * 0 * 0 * 1 * 0 * .....

T = 1 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 ...

So we observe that T = **1** □ i(T), where **1** is the stream of ones and i is as before the operation inverting 0,1 to 1,0, and □ is the zip operation also introduced earlier.

Question. Are all the the tails $t^n(T)$, different? In other words, is T an infinite state process, like M? What are recurrence equations for the entries of T?

Remark. For M we can determine the analogous generation tree. In the one for T, we have that right branches (i.e. branches taking the right successor of a node each time) alternate 1 and 0, and left branches are constant 1. In the analogous generation tree for M, we have that left branches starting in a 0-node are constant 0, left branches starting in a 1-node are constant 1, while right branches alternate.

Note that the generation tree for T is in fact a regular tree, defined by the recursive equations

$$\alpha = 1(\alpha, \beta)$$
$$\beta = 0(\alpha,\alpha)$$

where we conceive 0,1 as binary operators, α, β as recursion variables. Analogously for the generation tree of M.

$o(t^2(x)) = t(o(x))$
$e(t^2(x)) = t(e(x))$
$o(x) \square e(x) = x$ $o(x \square y) = x,$ $e(x \square y) = y$ $t(x \square y) = y \square t(x)$ $t(x) \square t(y) = t^2(x \square y)$

Table 4. Algebra of streams

5. The algebra of M and T. It is also interesting to look at *algebraic* aspects of the stream M. ('Algebraic' as in algebraic data types, or abstract data types.) To start with, there is the set of all streams of natural numbers. Here we have unary operations *tail*, removing the first element of the stream; *odd*, taking the entries at odd places 1,3,5,...; *even*, taking the entries at even places 0,2,4,...; the binary operation *zip*, taking two streams and zipping them up alternatingly to become one stream. We abbreviate the operations tail, odd, even by their first letter, and write $x \square y$ for zip(x,y). Some obvious equations holding in this algebra are in Table 4.

$x + x = 0$
$x + 0 = x$
$x + 1 = i(x)$
$i(i(x)) = x$
$d(0) = 0$, $d(1) = 0$
$d(0 \square 1) = 1$
$i(1) = 0$,
$t(1) = 1$, $t(0) = 0$,
$0 = 0.0$
$1 = 1.1$
$i(x \square y) = i(x) \square i(y)$
$i(0.x) = 1.i(x)$
$i(1.x) = 0.i(x)$
$i(t(x)) = t(i(x))$
$t(x) = x + d(x)$
$d(x) = x + t(x)$
$d(i(x) = d(x)$
$t(x + y) = t(x) + t(y)$
$d(x+y) = d(x) + d(y)$
$(x + x') \square (y + y') = (x \square y) + (x' \square y')$
$x \square (y+z) = (x \square y) + (0 \square z)$
$d(x \square y) = t(x \square x) + (y \square y)$
$t(d(x)) = d(t(x))$

Table 5. Algebra of boolean streams

Second, we consider the subalgebra of *boolean streams*: infinite sequences of 0's and 1's. In addition to the operations for the whole stream algebra, we now have operations +, adding two streams element-wise modulo 2; *invert*, replacing 0 by 1 and vice versa; the constants 0 (the stream of 0's) and 1, (the stream of 1's). We abbreviate invert by its first letter. Another interesting operation is the operation *dif*, giving for a stream x the stream of differences (modulo 2) of consecutive elements of x. Also this operation is denoted by its first letter. So in fact, T = d(M). Now we have in addition to the equations for all streams, the following equations for the boolean streams, in Table 5. It is just a handful of obvious equations, without any attempt for completeness in whatever sense.

Third, we take the subalgebra of the boolean stream algebra generated by the Thue-Morse sequence M. Some of the extra equations that hold in this algebra are in Table 6. The equations for the difference streams give in fact recurrence equations for these streams, which were observed

empirically from Table 7. The analogous recurrence equations for the tails of M, $t^n(M)$, are easily derived algebraically from the equations mentioned in the tables. They can be checked in Table 8.

$T = d(M)$
$M = M \square i(M)$
$T = 1 \square i(T)$
$M = 10(M \square i(M))$
$t^{2n}(M) = t^n(M) \square t^n(i(M))$
$t^{2n+1}(M) = t^n(i(M)) \square t^{n+1}(M)$
$d^{2n}(M) = d^n(M) \square d^n(M)$
$d^{2n+1}(M) = 0 \square d^{n+1}(M)$

Table 6: Some equations for the streams generated by M

Questions. How about the word problem, complete axiomatizations, ω-complete axiomatizations? M and T are infinitary palindromes. How about the other $d^n(M)$? M and T can be obtained by iterating a morphism. How about the other $d^n(M)$? Some of the sequences $d^n(M)$ are homomorphic images of each other. For which i,j is $d^i(M)$ a homomorphic image of $d^j(M)$? It seems (empirically) that the only homorphisms that we have are those given by the recurrence equations for the $d^n(M)$, namely $d^n(M) \rightarrow d^{2n}(M)$ by the homomorphism $0 \rightarrow 00$, $1 \rightarrow 11$, and $d^{n+1}(M) \rightarrow d^{2n+1}(M)$ by the homomorphism $0 \rightarrow 00$, $1 \rightarrow 01$, for all $n \geq 1$.

6. Plane tilings for M and T: connecting the dots. Each 0,1-stream induces in a natural way a tiling of the plane, or rather, of a quadrant of the plane. For two 0,1-words x,y we define a kind of product that we denote with $x \otimes y$, as follows. The word x is written horizontally, the word y is written vertically. See Figure 3, with x = 10101010 and y = 110110110. Now we construct a matrix of 0,1's by copying x in each row where y has an entry 1, and taking the word i(x) (in the notation employed earlier, so the 'negative' of x) in a row where y has an entry 0. Next, we connect the 0's that are *adjacent* by connection lines, disregarding the 1's. We can do this for finite but also for infinite words x and y. Experiment shows that we will often get a tiling built from some basic tiles that are easily identified. However, in order to get a tiling for $x \otimes y$ from these basic tiles, we have to impose the restriction on x and y that they do not contain 00 or 11 as a subword, or equivalently, that in x and y each 1 is eventually followed by a 0 and vice versa. Otherwise we may get some degenerate

'tilings', e.g. for $1110^{\omega} \otimes \mathbb{1}$, just resulting in a single point, or $(10)^{\omega} \otimes \mathbb{1}$, consisting of vertical lines. Another esthetic detail is that we do not connect adjacent 0's when the connection would be a diagonal of a unit square tile.

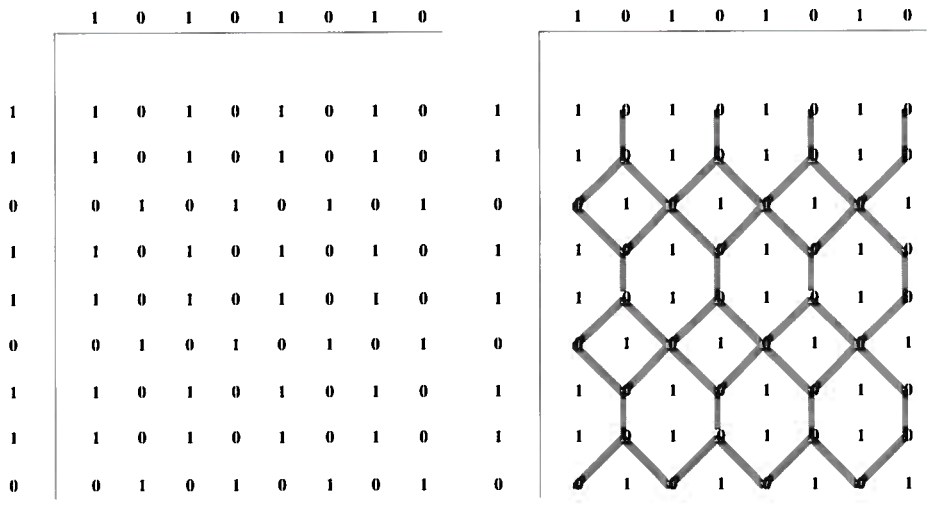


Figure 3. Connecting the dots.

In the figures we have taken the products of (an initial segment of) M with itself, and likewise for T. Thus Figure 4 contains the tiling $(M)_{64} \otimes (M)_{64}$, where $(M)_{64}$ is the prefix of M of length 64, a palindrome. Figure 5 contains an initial part of the tiling for the stream T, namely $(T)_{31} \otimes (T)_{31}$. As we saw $(T)_{31}$ is also a palindrome. The two tilings are coloured so that their structure is more easily seen. Even for these small initial parts of the tilings one can clearly see something of the self-similarity of M and T, now in a "two-dimensional way".

Note that the total tilings $M \otimes M$ and $T \otimes T$ are also self-similar in the sense that each finite part is present infinitely many times in the plane tiling.

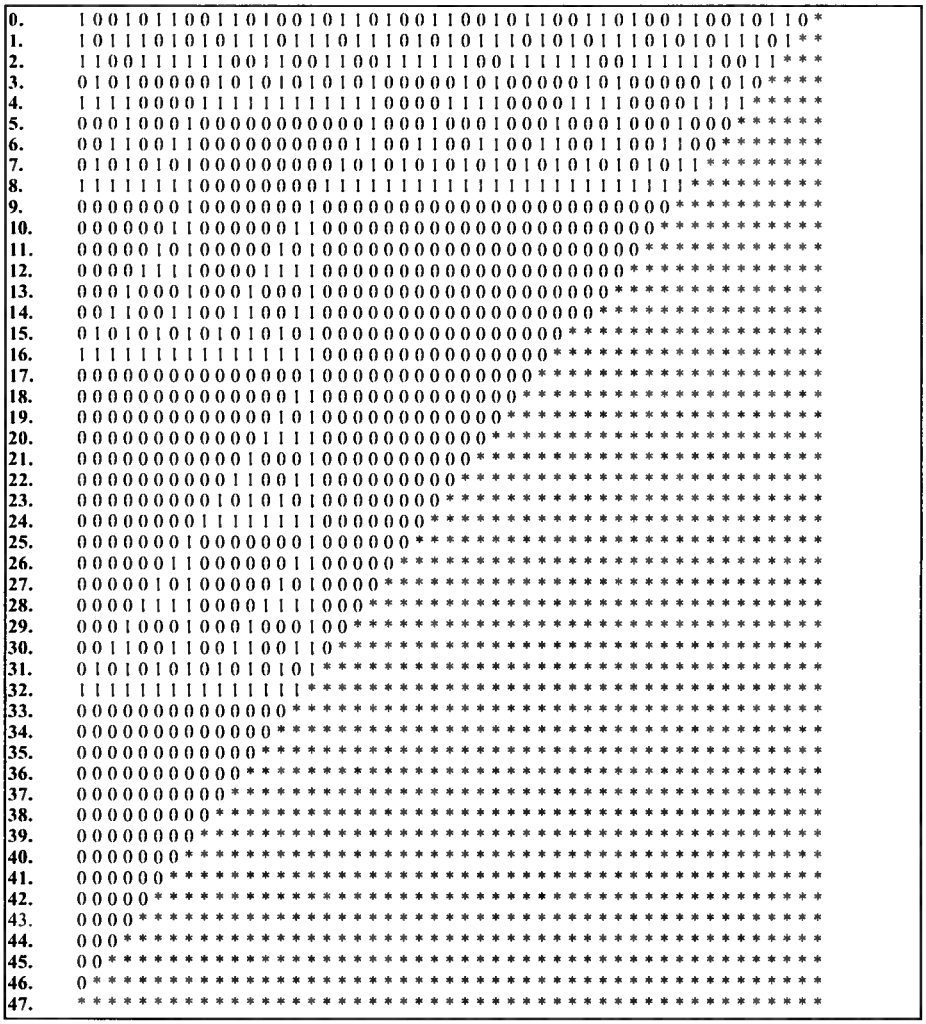


Table 7: Difference streams $d^n(M)$ of the Thue-Morse sequence M

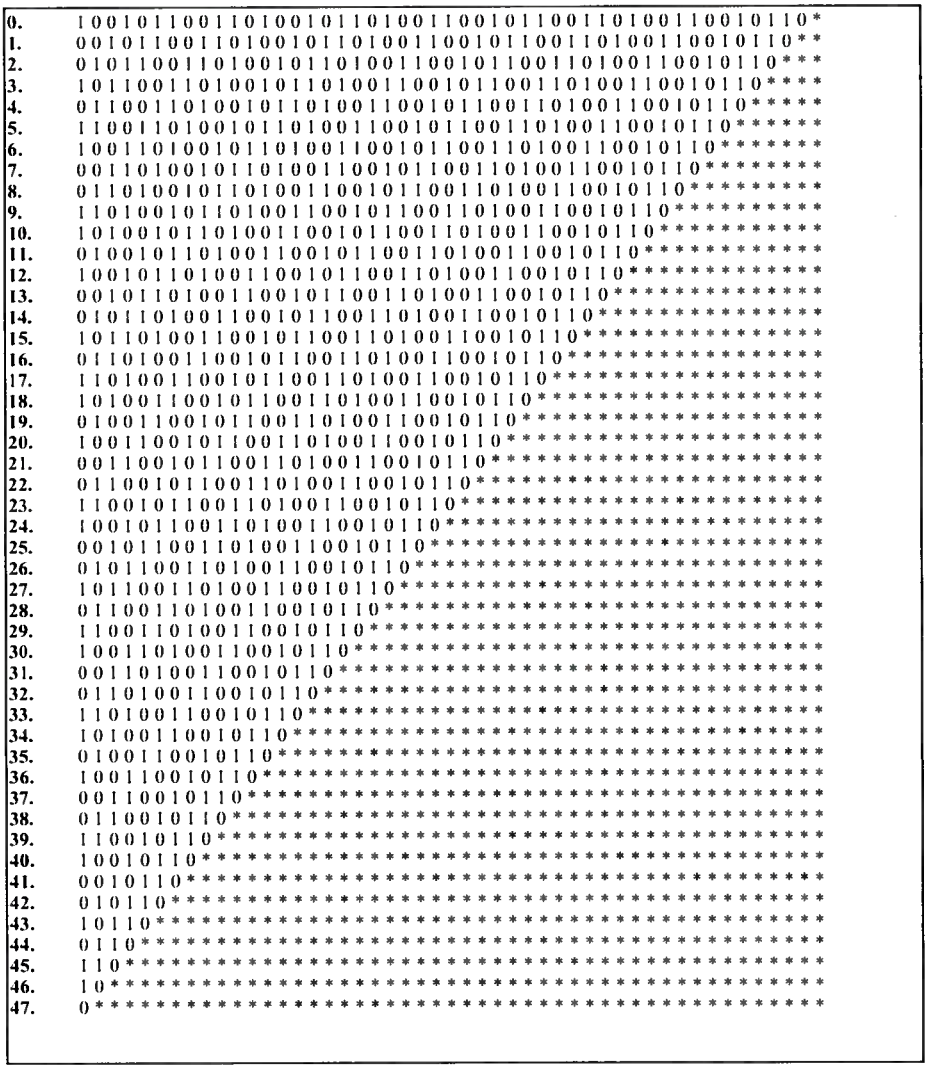


Table 8: Tail streams $t^n(M)$ of the Thue-Morse sequence M

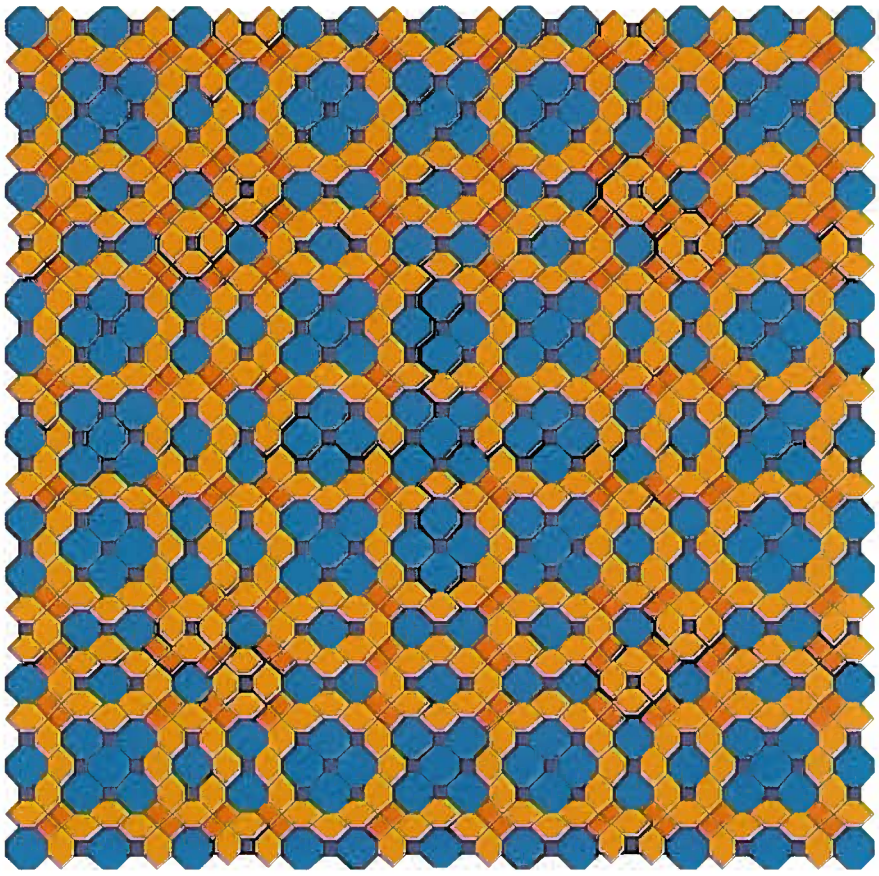


Figure 4: Plane tiling for the Thue-Morse sequence

Finally, we mention another intriguing stream, described by M. Keane in Alberts and van Zwet [2002] and called there the Mephistowals, the result of iterating the morphism $0 \to 001$, $1 \to 110$, starting with 0:

001 001 110 001 001 110 110 110 001 ...

It would be nice to see the plane tiling of this stream. It would be even nicer if someone developed an automated way of rendering these tilings graphically.

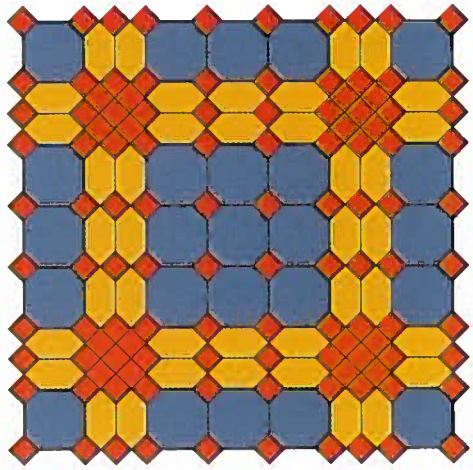


Figure 5: Tiling for the Toeplitz sequence

We conclude with wishing Jaco many hours of reflection at some wonderful stream!

References

G. Alberts and W. van Zwet, Een benoeial met smaak, interview met Mike Keane, Nieuw Archief voor Wiskunde 5/3, nr. 2, juni 2002, p.141-146.

J.-P. Allouche and J. Shallit, *The ubiquitous Prouhet-Thue-Morse sequence*, in: C. Ding, T. Helleseth, and H. Niederreiter editors, *Sequences and their applications, Proceedings of SETA'98*, pages 1-16. Spinger-Verlag, 1999.

J.A. Bergstra and J.W. Klop, *The algebra of recursively defined processes and the algebra of regular processes*, in: Proc. 11th ICALP (ed. J. Paredaens`), Springer LNCS 172, 1984, 82-94. Extended version in: A. Ponse, C. Verhoef, S.F.M. van Vlijmen, eds., Algebra of Communicating Processes, Utrecht 1994, Workshops in Computing, Springer-Verlag 1995, 1-25.

J.-P. Allouche and T. Johnson, *Narayana's cows and delayed morphisms*. In *3èmes Journées d'Informatique Musicale (JIM '96), Ile de Tatihou*, pages 2-7, May 1996.

Jean Berstel, Maxime Crochemore and Jean-Eric Pin, *Thue-Morse sequence and p-adic topology for the free monoid*, July 2000.

M. Euwe, *Mengentheoretische Betrachtungen über das Schachspiel*, Proc. Konin. Akad. Wetenschappen, Amsterdam 32 (1929), 633-642.

M. Keane, *Generalized Morse sequences*, Z. Wahrscheinlichkeitstheorie Verw. Geb. 10 (1968), 335-353.

Marston Morse, *Recurrent geodesics on a surface of negative curvature*. *Trans. Amer. Math. Soc.*, 22:84-110, 1921.

Morse, M. and Hedlund, G. A. *Unending Chess, Symbolic Dynamics, and a Problem in Semigroups*, Duke Math. J. **11**, 1-7, 1944.

J. Tromp and J. Shallit, *Subword complexity of a generalized Thue-Morse word*, Info. Proc. Letters 54 (1995), 313-316.

E. Prouhet, *Mémoire sur quelques relations entre les puissances des nombres*, C.R. Acad. Sci. Paris Sér. I 33 (1851), 225.

A. Saloma, *Jewels of formal language theory*, Pitman 1981.

Sloane, N. J. A. Sequences A010060 in "The On-Line Encyclopedia of Integer Sequences." http://www.research.att.com/~njas/sequences/.

Axel Thue, *Über die gegenseitige Lage gleicher Teile gewisser Zeichenreihen, Norske Vid.-Akad. Oslo Mat.-Natur. Kl. Skr. (N.S.)*, (1), 1912.

Knowledge, Abilities, Results and Opportunities

John-Jules Meyer
Utrecht University
Institute of Information and Computing Sciences
Intelligent Systems Group
Postbus 80.089
3508 TB Utrecht

June 3, 2002

Abstract

In this short paper dedicated to Jaco de Bakker we describe the area of agent technology, and in particular we focus on our work on the KARO (Knowledge, Abilities, Results, Opportunities) framework for describing the attitudes of intelligent agents, and the way we try to realise these agents by means of agent-oriented programming (languages).

1 Intelligent Agents

In the last decade our group has worked on the subject of 'intelligent agents', which is becoming ever more central to the field of artificial intelligence [19]. (Intelligent) agents are software (or hardware) entities that display a certain degree of autonomy while operating in an environment (possibly inhabited by other agents) that is not completely known by the agent and typically is changing constantly. Agents possess properties like reactiveness, proactiveness and social behaviour, often thought of as being brought about by mental or cognitive attitudes involving knowledge, beliefs, desires, goals, intentions,..., in the literature often referred to as '*BDI attitudes*' (for beliefs, desires, intentions).

The area of agent technology covers the foundations as well as the design, implementation and application of intelligent agents, both stand alone and within (the context of) multi-agent systems. The foundations concern agent theories and in particular agent logics as a basis for agent architectures and the specification and verification of agent programs written in agent programming) languages. Design and implementation of agents involve the study of agent architectures and agent programming languages by means of which agents can be realised. Applications of agent technology are numerous, and range from intelligent personal assistants in various contexts to cognitive robots and e-commerce, for example.

Especially important is the study of multi-agent systems, which incorporates such issues as communication, coordination, negotiation and distributed reasoning/problem solving and task execution (e.g. distributed / cooperative planning and resource allocation). As such the area is part of the field of distributed AI. An important subfield is that of the investigation of agent communication languages that enable agents to communicate with other agents in a standardized and high-level manner.

2 Logics for Intelligent Agents

In order to describe and specify the behaviour (and the BDI attitudes, more in particular) of intelligent agents a number of logics have been proposed. The most well-known are those of Cohen and Levesque [1] and Rao and Georgeff [27]. The former is based on a linear-time temporal logic and is augmented with modal operators for belief and goals, and a possibility to express the performance of actions. On this basis Cohen and Levesque define intentions as certain (persistent) goals. In fact, the formalism is 'tiered' in the sense that it contains

about languages for programming agents (such as AgentSpeak(L) [26]). I'll return to this in the next section.

and its many variants: here programs written in that language are employed to guide a theorem prover to find a program (plan) to reach the goals of an agent.

Our approach to overcoming the gap between agent specification and realisation is a more traditional one: distinguish between (logical) specification language and (procedural) programming language. We have devised a number of agent programming languages (3APL, ACPL, GOAL, GrAPL) to write agent programs, each with an emphasis on a particular aspect. For, example 3APL [13, 14, 12] is particularly suited for programming BDI-like agents by means of a mixture of constructs from imperative and logic programming. This language contains constructs involving beliefs and (procedural) goals. From our range of languages, this language is developed the furthest, both theoretically and implementation-wise. ACPL [4, 5, 3, 7, 6, 8] is a language for multi-agent systems with constructs for communication, using constructs based on speech act theory, with which one can program agents that coordinate and negotiate. GOAL [15] is a language with declarative goals ('goals-to-be' rather than the procedural 'goals-to-do' that are present in 3APL). GrAPL [30] contains constructs for initiating group activity (group coordination and execution of tasks).

The idea is to have assertion languages associated with these programming languages in which specification and verification of agent programs can be done, much in the same vain as in traditional software engineering. However, of course in order to be useful these assertion languages should also contain sufficient expressiveness to express the BDI-like notions that are used in the programming languages. So here a kind of mix is needed of formal methods from 'traditional' software engineering and formal methods from AI and knowledge representation. Typically one could think of temporal or Hoare-like logics augmented with BDI modalities.

For (fragments of) the languages 3APL and GOAL we have already obtained first versions of such assertion languages [16, 15].[1]

Of course, the availability of such assertion languages is only a first step towards what one can call *agent-oriented software engineering*.[2] In an NWO-project we try to develop the programming language 3APL further, on the one hand by extending the language with extra features (like communication), on the other by providing a method(ology) to aid the 3APL programmer and the tools associated with this. We are also aiming at a good interface with JAVA so that our language may be widely used, at least in principle. As an application we want to be able to program advanced ('cognitive') robots by means of the language 3APL. Although we are not yet near reaching this aim, it became apparent to us that for programming a BDI-like agent such as the ones programmed in 3APL, the programmer should have much more control of the basic loop of the interpreter (realising the [sense-]reason-act cycle) than we have realised in the present implementation, and we have some preliminary thoughts on how to improve this [2].

5 The KARO Framework Applied

In this section I will apply the KARO framework on the 'agent' Jaco de Bakker as a tool to assess his great value to me as his former student.

Knowledge

Jaco's knowledge of semantics was (and still is, I'm sure) vast, and influenced me profoundly. Jaco gave me a true appreciation of the subject. I think I have always remained a semanticist

[1] In order to do this we need of course also a precise semantics of the languages involved, including the programming languages. This was another problem with AGENT0, for which there was no such semantics available. For languages like 3APL, ACPL and GrAPL we have devised precise operational semantics based on Plotkin-style transition systems, where rules are given for transitions of configurations. In our agent-oriented languages these configurations consist typically of an agent program and the agent's mental state, since agent programs can primarily be thought of as 'mental state transformers': they change the mental (BDI) state, beliefs, desires/goals, intentions of an agent.

[2] Note that this term is ambiguous, and is also used ambiguously in the field. It may refer to the engineering of agent-oriented software, as we mean it here, but it may also refer to the much more ambitious aim of engineering general (in principle) software in an agent-oriented way. Here one faces the extremely difficult task of specifying and designing a program written in an arbitrary programming language, say JAVA, by means of agent concepts and logics. In my opinion this may be too ambitious, since I believe that there should be some natural relation between the concepts (e.g. BDI ones) that are used in the specification and the programming languages.

at heart since my training days with Jaco. Also in my current work in AI I always emphasize conceptual modelling which is very much related to semantics, and also more in particular in the context of agent-oriented programming languages I always stress the importance of the semantics of these languages. (One of the flaws in the otherwise so inspiring work of someone like Yoav Shoham is that he proposed a language, AGENT-0, without a proper semantics, so that one had to experiment with the interpreter to see what the constructs in the language did exactly.) I'm very proud of work done by my (former) students Rogier van Eijk, Koen Hindriks and Wieke de Vries where they have investigated several semantical issues in the area of (multi) agent systems and agent programming.

Abilities

I have always admired Jaco's abilities to combine managerial and research attitudes. Even in the biggest organisational crisis he remained a researcher to me in the sense that he always was interested in research questions and technical details. Another thing I really envied was his ability to write things in a remarkably clear way. For me his book "The Mathematics of Program Correctness" was a kind of bible that mixed clarity with precision! (That is, for me... I know that some might disagree, especially some students at the Free University whom we taught from the book, and perhaps some in Nijmegen and Utrecht, where I used several chaptersof the book for my own lectures.) Jaco could write delightfully clear and systematic surveys and overviews of the work of himself and the people in his group. I've tried to do something in the same vein in this article, with an emphasis on 'tried'.

Results

I can say that from the collaboration with Jaco a lot has resulted. We have written a dozen papers together, also with other people. I've great memories of these collaborations or just acquaintences with the people associated with the Amsterdam Concurrency Group (CWI/VUA) and the people working in Jaco's group at the CWI or just people that were sometimes around, like Jan Bergstra, Jan Willem Klop, John Tucker, Ralph-Johan Back, Jeff Zucker, Ernst-Rüdiger Olderog, Joost Kok, Jan Rutten, Erik de Vink, Eike Best, to mention a few... Let me also take this opportunity to mention a *non*-result that has haunted me for a few years. I remember that we (Jaco, Jan Willem and I) were working on a paper on function procedures (I believe it was about semantics and proof theory for these language constructs). It was difficult stuff. We managed to write an extended abstract that was accepted for the "Logics of Programs" workshop, a predecessor of the later well-known LICS conferences. During summer especially Jan Willem and I tried to extend the paper into a full journal paper and I believe we had some partial results. However, then Jaco came back from the States where he together with Jeff Zucker had invented the semantics of concurrent processes (which later had Bergstra & Klop's well-known process algebra as a kind of offspring). It was clear: Jaco had lost all interests in function procedures and in fact the process (or should I say Jaco & Jeff's processes) influenced Jan Willem and me to the extent of dropping the paper on function procedures altogether. We later regretted this (also because our German colleague Werner Dam asked us about a full paper), and at some point we tried to put it together after all. However, then we discovered how volatile ideas sometimes are: we could not recover everything, and the only thing left are the cryptic notes in Jan Willem's diary...

Opportunities

This is the most personal part. Jaco offered me many opportunities. To begin with, after a job interview in which he lost a tooth which ended the interview abruptly, he gave me the opportunity to start as a PhD at the Free University. As a PhD student I enjoyed the stimulating and internationally oriented environment he had created (Amsterdam Concurrency Group, REX, etc.), which gave me an idea of how science should be done. Later I often referred to the REX seminars as a model of how to organise national research groups. And we worked very intensively together, from which I learned a lot. I remember that Jaco also showed a remarkable tolerance with respect to the scientific 'hobbies' I developed while being a PhD student, such as the subject of deontic logic, which certainly at that time was a bit obscure, even within philosophical logic.

I remember that in the library of the CWI I came across a paper that discussed the so-called Anderson's reduction of deontic logic to alethic modal logic, and in particular all the problems (in the form of paradoxes) associated with it. Actually, by the very study of programming logics such as dynamic logic, I saw a solution to these problems by a reduction to dynamic rather than alethic modal logic. But, as I was not sure about what Jaco would think about it, I worked on it as a weekend activity and kept it a secret till I finished the paper. I showed it to Jaco, and to my relief he was enthusiastic and attended even a first presentation of the work at a seminar at the philosophy department of the University of Amsterdam, which I appreciated very much. Later he supported my career at the Free University, from PhD student till professor (bijzonder hoogleraar), which gave me the opportunity to develop these other interests further resulting in a strong interest in the area of logic for AI that I have kept till this day. The work with co-workers such as Wiebe van der Hoek and Bernd van Linder yielded the KARO logic for specifying intelligent agents...

Jaco, thanks a lot for your 'KARO attitudes'! They have given me knowledge, abilities, results and opportunities to pursue my BDI attitudes in the field of AI and agent technology in particular, as I have outlined in this short paper.

I conclude this paper with a bibliography and take this *opportunity* for a shameless advertisement for the work we have done in the past years in Utrecht on the foundations of agent technology.

References

[1] P.R. Cohen & H.J. Levesque, Intention is Choice with Commitment, *Artificial Intelligence* 42, 1990, pp. 213–261.

[2] M. Dastani, F. de Boer, F. Dignum, W. van der Hoek, M. Kroese & J.-J. Ch. Meyer, Programming the Deliberation Cycle of Cognitive Robots, in Proceedings AAAI Workshop on Cognitive Robotics (CogRob'02), 2002, to appear.

[3] R.M. van Eijk, Programming Languages for Agent Communication, PhD. Thesis, Utrecht University, 2000.

[4] R.M. van Eijk, F.S. de Boer, W. van der Hoek & J.-J. Ch. Meyer, A Language for Modular Information-Passing Agents, *CWI Quarterly* 11(2,3), 1998, pp. 273–297.

[5] R.M. van Eijk, F.S. de Boer, W. van der Hoek & J.-J. Ch. Meyer, Information-Passing and Belief Revision in Multi-Agent systems, in: *Intelligent Agents V (Agent Theories, Architectures, and Languages)* (J.P. Mller, M.P. Singh & A.S. Rao (eds.), LNAI 1555, Springer, Berlin, 1999, pp. 29–45.

[6] R.M. van Eijk, F.S. de Boer, W. van der Hoek & J.-J. Ch. Meyer, Open Multi- Agent Systems: Agent Communication and Integration, *Intelligent Agents VI, Proc. of 6th Int. Workshop on Agent Theories, Architectures, and Languages (ATAL'99)* (N.R. Jennings & Y. Lesp´erance,eds.), LNAI 1757, Springer, Heidelberg, 2000, pp. 218–232.

[7] R.M. van Eijk, F.S. de Boer, W. van der Hoek & J.-J. Ch. Meyer, Operational Semantics for Agent Communication Languages, in: *Issues in Agent Communication* (F. Dignum & M. Greaves, eds.), LNAI 1916, Springer, Berlin, 2000, pp. 80–95.

[8] R.M. van Eijk, F.S. de Boer, W. van der Hoek & J.-J. Ch. Meyer, Generalised Object-Oriented Concepts for Inter-Agent Communication, in: *Intelligent Agents VII* (C. Castelfranchi & Y. Lesp´erance,eds.), LNAI 1986, Springer, 2001, pp. 260-274.

[9] M. Fisher, A Survey of Concurrent METATEM – The language and Its Applications, in: *Temporal Logic – Proc. of the First Int. Conf.* (D.M. Gabbay & H.J. Ohlbach, eds.), LNAI 827, Springer, Berlin, 1994, pp. 480–505.

[10] M.P. Georgeff & A.L. Lansky, Reactive Reasoning and Planning, in: Proc. 3rd Nat. Conf. on Artif. Intell. (AAAI-87), Seattle. WA, 1987, pp. 677-682.

[11] D. Harel, Dynamic Logic, in: D. Gabbay & F. Guenthner (eds.), *Handbook of Philosophical Logic, Vol. II*, Reidel, Dordrecht/Boston, 1984, pp. 497–604.

[12] K.V. Hindriks, Agent Programming Languages: Programming with Mental Models, PhD. Thesis, Utrecht University, 2001.

[13] K.V. Hindriks, F.S. de Boer, W. van der Hoek & J.-J. Ch. Meyer, Formal Semantics for an Abstract Agent Programming Language, in: *Intelligent Agents IV* (M.P. Singh, A. Rao & M.J. Wooldridge, eds.), LNAI 1365, Springer, 1998, pp. 215–229.

[14] K.V. Hindriks, F.S. de Boer, W. van der Hoek & J.-J. Ch. Meyer, Agent Programming in 3APL, *Int. J. of Autonomous Agents and Multi-Agent Systems* 2(4), 1999, pp.357–401.

[15] K.V. Hindriks, F.S. de Boer, W. van der Hoek & J.-J. Ch. Meyer, Agent Programming with Declarative Goals, in: Intelligent Agents VII (C. Castelfranchi & Y. Lesp´erance, eds.), LNAI 1986, Springer, 2001, pp. 228–243.

[16] K.V. Hindriks, F.S. de Boer, W. van der Hoek & J.-J. Ch. Meyer, A Programming Logic for Part of the Agent Language 3APL, in: (Proc. First Goddard Workshop on) Formal Approaches to Agent-Based Systems (FAABS 2000) (Rash, J.L., Rouff, C.A., Truszkowski, W., Gordon, D. & Hinchey, M.G. eds.),, LNAI 1871, Springer, Berlin/Heidelberg, 2001, pp. 78-89.

[17] W. van der Hoek, B. van Linder & J.-J. Ch. Meyer, A logic of capabilities, in: *Proceedings of the Third International Symposium on the Logical Foundations of Computer Science (LFCS'94)* (A. Nerode & Yu. V. Matiyasevich, eds.), LNCS 813, Springer-Verlag, Berlin/Heidelberg, 1994, pp. 366–378.

[18] W. van der Hoek, B. van Linder & J.-J. Ch. Meyer, An Integrated Modal Approach to Rational Agents, in: *Foundations of Rational Agency* (M. Wooldridge & A. Rao, eds.), Applied Logic Series 14, Kluwer, Dordrecht, 1998, pp. 133–168.

[19] N.R. Jennings & M.J. Wooldridge, *Agent technology: Foundations, Applications, and Markets*, Springer, Berlin, 1997.

[20] B. van Linder, Modal Logics for Rational agents, PhD. Thesis, Utrecht University, 1996.

[21] B. van Linder, W. van der Hoek & J.-J. Ch. Meyer, Actions that Make You Change Your Mind: Belief Revision in an Agent-Oriented Setting, in: *Knowledge and Belief in Philosophy and Artificial Intelligence* (A. Laux & H. Wansing, eds.), Akademie Verlag, Berlin, 1995, pp. 103–146.

[22] B. van Linder, W. van der Hoek & J.-J. Ch. Meyer, Formalising Motivational Attitudes of Agents: On Preferences, Goals and Commitments, in: M. Wooldridge, J.P. Müller & M. Tambe (eds.), *Intelligent Agents II – Agent Theories, Architectures, and Languages*, LNAI 1037, Springer-Verlag, 1996, pp. 17–32.

[23] B. van Linder, W. van der Hoek & J.-J. Ch. Meyer, Seeing is Believing (And So Are Hearing and Jumping), *Journal of Logic, Language and Information* 6, 1997, pp. 33–61.

[24] J.-J. Ch. Meyer, W. van der Hoek & B. van Linder, A Logical Approach to the Dynamics of Commitments, *Artificial Intelligence* 113, 1999, 1–40.

[25] R.C. Moore, A Formal Theory of Knowledge and Action, in: J.R. Hobbs & R.C. Moore (eds.), *Formal Theories of the Commonsense World*, 1985, Ablex, Norwood, New Jersey, pp. 319–358.

[26] A.S. Rao, AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language, in: *Agents Breaking Away* (W. van der Velde & J. Perram, eds.), LNAI 1038, Springer, Berlin, 1996, pp. 42–55.

[27] A.S. Rao & M.P. Georgeff, Modeling Rational Agents within a BDI-Architecture, in *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)* (J. Allen, R. Fikes & E. Sandewall, eds.), Morgan Kaufmann, 1991, pp. 473–484.

[28] Y. Shoham, Agent-Oriented Programming, *Artificial Intelligence* 60(1), 1993, pp. 51–92.

[29] W. de Vries, F.S. de Boer W. van der Hoek & J.-J. Ch. Meyer, A Truly Concurrent Model for Interacting Agents, in: Proceedings of the 4th Pacific Rim International Workshop on Multi-Agents (PRIMA 2001) (Soe-Tsyr Yuan & Makoto Yokoo, eds.), LNAI 2132, Springer, 2001, pp. 16–30.

[30] W. de Vries, F.S. de Boer, K.V. Hindriks, W. van der Hoek &. J.-J. Ch. Meyer, A Programming Language for Coordinating Group Actions, in: From Theory to Practice in Multi-Agent (B. Dunin-Keplicz & Edward Nawarecki, eds.), LNAI 2296, Springer, 2002, pp. 313–321.

Jan van Mill

%%

Van de Divisie Wiskunde en Informatica.

Van 1956 tot 1964 was De Bakker student wiskunde aan de VU. Hij volgde ondermeer colleges bij de hoogleraren Mullender, Koksma, Grosheide, Sizoo, en Jonker. Van 1961 tot 1964 was hij assistent C bij de toenmalige subfaculteit Wiskunde en assisteerde hij Professor Mullender. Dat had natuurlijk nog niets met Informatica te maken. De Bakker deed zijn promotieonderzoek aan het CWI. Per 1 mei 1973 werd hij naast zijn baan aan het CWI aan de Vrije Universiteit benoemd tot buitengewoon hoogleraar Informatica bij de toenmalige Subfaculteit Wiskunde. Zijn taak was onderwijs te geven en onderzoek te doen op het gebied van Programmeertheorie (een onderdeel van de Theoretische Informatica). In samenwerking met zijn collegae Van de Riet en Tanenbaum werd in 1984 de zelfstandige opleiding Informatica in het leven geroepen. Het illustere drietal De Bakker, Van de Riet en Tanenbaum was gedurende vele jaren het gezicht van de Informatica aan de VU. Waar Van de Riet de meer praktische Informatica voor zijn rekening nam, zoals databanken, en Tanenbaum de Informatica rond computersystemen, werd De Bakker verantwoordelijk voor de Theoretische Informatica. Zijn specialisme, het gebied Programmeertheorie, werd aangeduid als „Studie van theoretische grondslagen voor algoritmen". De Bakker had een groot aantal promovendi, waarvan velen nu een hoogleraarspositie bekleden in Nederland of daarbuiten. In het bijzonder denken we aan promovendi die ook bij onze subfaculteit in dienst waren, zoals Anton Nijholt, Willem Paul de Roever en John-Jules Meyer.

De Bakker heeft een belangrijke rol gespeeld binnen de Informaticagroep aan de VU. Hij heeft zich een grote internationale naam verworven en het wetenschappelijk niveau van de Informatica aan de VU op grote hoogte gebracht. Topwetenschappers kwamen mede door hem graag op bezoek. Ook de goede naam die de VU-Informaticaopleiding binnen Nederland heeft, is mede aan De Bakker te danken.

De Bakker was aimabel en een stabiele factor binnen de Informatica. Het ging niet altijd precies zoals De Bakker het zich had voorgesteld. De reorganisatie van onze (oude) faculteit in de nieuwe Faculteit der Exacte Wetenschappen had niet zijn instemming. Dat de Theoretische Informatica soms stiefmoederlijk bedeeld werd in zijn ogen, veroorzaakte ook wel eens heftige discussies. Maar nooit verstoorden die discussies de goede verhoudingen. De Bakker was naast een briljant onderzoeker, een alom gerespecteerd bestuurder die in verschillende centrale gremia medebepalend is geweest voor het beleid op het gebied van de Informatica binnen en buiten de toenmalige Faculteit Wiskunde en Informatica.

Zijn plaats wordt thans door twee nieuwe hoogleraren ingevuld: Wan Fokkink en Jan Rutten. Het feit dat alom het belang van de Theoretische Informatica wordt ingezien is mede aan De Bakker te danken.

Jaco het ga je goed in je nieuwe periode van pensioen. Geniet ervan samen met je vrouw Lien.

De Divisiedirectie Wiskunde en Informatica

Vrije Universiteit, 29-05-02

From Pisa to Amsterdam and Back: A Fruitful Path for European Computer Science: Reviving 35 years of Collaboration in Honour of Professor Jaco de Bakker
by Ugo Montanari, Dipartimento di Informatica, University of Pisa

When I arrived at Pisa in February 1968 from Milan - attracted by the plans for a computer science department, which was actually founded a year later - I heard of a conference which had just taken place in Pisa, organized by the late Professor Alfonso Caracciolo di Forino, a pioneer of the area of formal definition of programming languages. In particular I noticed the key contribution of Professor Jaco De Bakker, titled "Formal Definition of Programming Languages - with an Application to the Definition of ALGOL 60". This was the first time I heard of Jaco, already well known in the area. The conference was an important milestone in the development of theoretical computer science.

In the seventies I met Jaco several times in various occasions, but only more recently our interactions became more frequent, while my research interests evolved, through intermediate stages, from artificial intelligence to semantics of concurrency. In the eighties tight collaborations were developed between Jaco and my department in Pisa at the time of the first Esprit projects. However the Pisa counterpart of Jaco was Giorgio Levi - and Catuscia Palamidessi, at that time a PhD student. An important project my research group shared with Jaco's was MASK, Mathematical Structures in Semantics for Concurrency, led by Jan Rutten. It lasted from 1992 to 1995 and included also Philippe Darondeau, Mila Majster-Cederbaum, Lutz Priese and Furio Honsell. In particular I remember the first meeting in Udine, in October 1992, with a very pleasant dinner at Hotel Astoria. We organized the last meeting on May 1995, in Alghero, Sardinia, since Jaco always preferred the last workshops of (successful) projects to be organized in pleasant islands!

Jaco and myself had other important occasions of interaction. For instance I remember Jaco as reviewer of CEDISYS, a second generation ESPRIT project I coordinated from 1989 to 1992. In particular the first review took place in March 1990 in San Miniato, during the 2nd Workshop on Concurrency and Compositionality. We were both members for several years of the expert group on Mathematics and Information Sciences of the Human Capital and Mobility program. We shared HCM project EuroFocs - European Institute in the Logical Foundations of Computer Science - led by Gordon Plotkin, 1994-1996. In this occasion I had the pleasure to host in Pisa for eighteen months an excellent former student of Jaco, Franck van Breugel. More recently, during our collaboration within the Esprit working Group COORDINA - From CoordinationModels to Applications - led by Antonio Porto, we organized two specific Amsterdam-Pisa bilateral meetings, one in Pisa, on January 1999 and the other in Amsterdam, on January 2000. I remember them as very interesting from both the scientific and the human viewpoint. Finally in autumn 2000 - spring 2001, Jan Rutten and Jaco hosted a very gifted student of mine, Matteo Coccia, who unfortunately passed away later in dramatic contingencies.

As a final comment I would like to emphasize that, while remarkable and very well known, the scientific contributions of Jaco are not the only important results he achieved in his career. The educational and organizational merits are not of lesser relevance. In fact entire generations of Dutch students and scientists owe him their formation and orientation, and I am glad to have close relations with several of them. Also European scientists are especially indebted to him, both for his important contributions to the creation of a European approach to computer science and for his organizational role in many European research projects, in several of which both the University of Pisa and the CWI have been jointly involved.

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

Prof. Ugo Montanari	Phone: +39 050 2212721
Dipartimento di Informatica	Fax: +39 050 2212726
Universita' di Pisa	Both changed recently.
Corso Italia, 40	Email: ugo@di.unipi.it
I-56100 Pisa, Italy	http://www.di.unipi.it/~ugo/ugo.html

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

Jaco de Bakker (1976-1980): Best Wel Aardig

Anton Nijholt
Universiteit Twente
Faculteit Informatica
Postbus 217
7500 AE Enschede

Het moet in 1974 geweest zijn dat ik afstudeerde in Delft, Leo Verbeek volgde naar Twente om daar als onderzoeksassistent mijn afstudeerverslag om te zetten naar een tijdschriftartikel en een artikel voor een ACM conferentie. Halverwege 1975 werd er bij mij op aangedrongen te kijken naar de periode daarna. Blijkbaar was er op de Vrije Universiteit bij ene Jaco de Bakker een positie als promovendus. Ik wist weinig van de Vrije Universiteit. Wat ik ervan wist was de aanwezigheid van ene Reind van der Riet, van wie ik een aantal artikelen over formule-manipulatie had gelezen in het tijdschrift Informatie. Dat moet eind van de jaren '60 geweest zijn. Die artikelen hadden op mij indruk gemaakt en tijdens mijn kennismaking met de computer, daarna tijdens een werkperiode als programmeur bij TNO, voorafgaande aan mijn studententijd aan de TU Delft, heb ik dan ook geprobeerd om soortgelijke interessante dingen te doen.

Verder wist ik van de Vrije Universiteit dat men daar een of andere levensovertuiging aanhing of althans wenste uit te dragen waar ik me ver van verwijderd achtte. Wellicht niet zozeer van die levensovertuiging als wel van de wijze van uitdragen ervan. Van de Bakker had ik nooit gehoord en tijdens het sollicitatiegesprek dat ik mocht voeren met Cor Baaijen en Jaco zal mijn näiviteit ook wel opgevallen zijn. Er zal ook sprake geweest zijn van een vrij grote tegenstelling met mijn illustere voorganger als promovendus bij de Bakker aan de VU, namelijk Willem-Paul de Roever.

Aan de andere kant, ook mijn mening dat men daar blijkbaar wist waar men het over had als het ging over het uitdragen van een op christelijke grondslag uitgedragen levensvisie moest worden aangepast. Er moest een vraag gesteld worden tijdens het sollicitatiegesprek over het wel of niet accepteren of onderschrijven van die grondslag. Mijn tegenvraag waaruit die grondslag dan precies bestond leidde tot enige verwarring, het zoeken in een bureaula en het gezamenlijk tot de conclusie komen dat het wellicht toch niet zo belangrijk was voor het vervullen van de open promotieplaats. Ik zal ook duidelijk gemaakt hebben dat ik die grondslag best zou kunnen gedogen, een opvatting die in ieder geval bij de Bakker wel goed scheen te vallen.

Ik werd aangenomen, werd geacht aan semantiek te werken en ging vanaf 1976 mijn eigen weg door (voornamelijk) verder te gaan met syntaxis. Waarom geen semantiek? Moeilijk te zeggen. Het zal te maken hebben met de acceptatie van een aantal artikelen van mijn hand voortgekomen uit mijn afstudeerwerk, de niet alledaagse aanwezigheid van Jaco op de Vrije Universiteit, het niet willen opboksen tegen tal van gerenommeerde semantiekonderzoekers in de omgeving van Jaco en wellicht vooral het op eigen benen willen en kunnen staan. Dat laatste ook vanwege een langer studie- en werktraject voorafgaande aan de promotiepositie dan bij een reguliere pas afgestudeerde student.

Op de Vrije Universiteit en het Mathematisch Centrum kreeg ik te maken met tal van wetenschappers waar ik tegen op keek. Aan de andere kant, mijn eigen zaken werden, waar dan ook, ook geaccepteerd en het reizen naar workshops, conferenties en zomerscholen kon beginnen en Jaco was daarbij duidelijk stimulerend en accepteerde dat het daarbij niet zozeer tot ondersteuning van zijn eigen onderzoekslijn leidde. Desondanks, want zoveel conferenties op het gebied van theoretische informatica waren er ook nog niet, kwamen we elkaar regelmatig in het buitenland tegen. ICALP en MFCS waren wel de belangrijkste ontmoetingsgelegenheden voor theoretische informatici in de jaren zeventig van de vorige eeuw. Zoals het daarbij hoorde, leerde ik ook Jaco kennen van kanten die in Nederland niet zo naar voren traden. Niet negatief hoor. Een bescheiden ijdelheid, een belangstelling voor vrouwelijke conferentiedeelnemers en angst voor het onbekende. Alhoewel, aan de ene kant een voor mij volstrekt onbegrijpelijke geremdheid om een hotel te verlaten en een (communistische) stad te verkennen, aan de andere kant toch ook wel bereid om gedurende een langere periode in een onbekend land als Brazilie te willen verblijven.

Het was natuurlijk uiterst prettig dat Jaco de Bakker wat betreft mijn werkzaamheden die tot een promotie moesten leiden eigenlijk nimmer twijfels uitsprak. Daar heb ik van geleerd en ik heb geprobeerd het geleerde toe te passen bij de vele promovendi die ik tot nu toe begeleid heb. Als iemand zijn eigen gang wil gaan bij het doen van wetenschappelijk onderzoek dan mogen er geen eventuele persoonlijke belangen van een promotor in de weg staan. Het 'persoonlijke' tijdens de promotieperiode vond ik wel erg de moeite waard. Gezamenlijke lunches, contacten tijdens workshops in het buitenland, en bezoeken thuis: in Oud-Zuid met Els (ik woonde nauwelijks een paar honderd meter van Jaco vandaan), later op een gracht waar Jaco tijdelijk een appartement had, gevolgd door een etentje in de rosse buurt, en nog later bij zijn nieuwe gezin in een woning in de buurt van de Amsterdamse RAI.

In 1980 mocht ik bij Jaco de Bakker promoveren. Mijn houding tegenover de promotie en de promotie-plechtigheid kwam vooraf niet bij iedereen even overtuigend over. Een iets te laconiek gedrag (dat heb ik nooit verleerd) wat betreft het wel op tijd beschikbaar zijn van het proefschrift leidde tot enige paniek bij het departement. Gelukkig was er de immer bij een ieder vertrouwen inboezemde Peter Apers die een dag en zijn auto beschikbaar stelde om de proefschriften uit Meppel op te halen, er was een College van Bestuur van de Vrije Universiteit die me op de dag van de promotie om acht uur 's ochtends liet opdraven om me aannemelijk te laten maken dat ik toch echt niet in officiele promotiekleding kon promoveren, er was de kapper van het Hiltonhotel (ja, inderdaad, dat van Herman Brood) die mijn veel te lange haren wilde kortwieken en er waren zeer op prijs gestelde commissieleden (Michael Harrison, Susan Graham, Jan van Leeuwen, Peter van Emde Boas en anderen) die tijdens de promotie de vriendelijkheid zelf waren. Het zoontje van de voorzitter van de faculteit maakte zich verdienstelijk door een fotoreportage te maken. Nog bedankt, Sape!

Het promotiefeest 's avonds bij me thuis was zeer de moeite waard. Meer dan vriendschappelijke vriendinnen hadden een buffet verzorgd, Jaco probeerde een ervan (een vriendin van de tweelingzus van mijn toekomstige vrouw; ik was op alledrie wel een beetje verliefd en sommigen ook op mij) te versieren (althans dat is wat ze mij vertelden) en er werd een pak vla door het open raam gegooid (die grotendeels opgevangen werd door de trui van Paul Vitanyi). Springer wilde mijn proefschrift in haar Lecture Notes opnemen (tja, met zo'n promotor) en er lag een uitnodiging om een poos in Canada research-fellow te spelen bij Derick Wood. Jaco had inmiddels ook al een nieuwe promovendus op de VU weten te regelen: John Jules Meyer.

Dear Jaco,

You have played a very important role in my life, in particular in my early years as a young scientist based at the University of Kiel, Germany.

Starting from 1984 you invited me for several longer visits to the CWI in Amsterdam. We worked on process theory, and I could bring my insights gained in a two years' research stay in Oxford into the fruitful collaboration with you, John-Jules Meyer and Jeff Zucker resulting in papers abbreviated BMOZ and BMO, but also with Jan Bergstra and Jan-Willem Klop. With Krzysztof Apt I happily collaborated on a book on program verification.

In my files I found the attached letter of 1985 communicating details in the development of BMOZ and BMO, and reflecting somewhat the excitement of those days. I take the opportunity of this Liber Amicorum to look back with pleasure at these visits to Amsterdam and the friendly atmosphere in the CWI.

Thank you very much and all the best for the coming years!

Ernst-Rüdiger

Oldenburg, May 2002

INSTITUT FÜR INFORMATIK
UND PRAKTISCHE MATHEMATIK
DER CHRISTIAN-ALBRECHTS-UNIVERSITÄT
E.-R. Olderog

2300 KIEL 1, den 9 July 1985
Olshausenstraße 40, Haus S 21
Fernruf: Geschäftszimmer: (0431) 880-4461
Bearbeiter: (0431) 880-

Institut für Informatik, 2300 Kiel 1, Olshausenstraße 40

Prof. Dr. J.W. de Bakker
CWI
Kruislaan 413
1098 SJ Amsterdam
NIEDERLANDE

Dear Jaco,

First of all thanks you once again for providing the nice opportunity of lecturing at the summer school in Noordwijkerhout. I enjoyed the time there very much.

Today I have some additional material resp. comments on our papers "BM02" and "BM10".

ad BM02: - Enclosed please find a proposal for a better explanation of global nondet. (In Noordwijkerhout we agreed that I should rewrite the remark on p. 4.3.) The new explanation has three additional references which are on sheet R3. I hope the remark is now satisfactory.

Note that I dropped the previous sentence "that global nondet. is more restrictive than local nondet." Because this statement would require a more careful formalization than presently on the old p. 4.3. In fact such a comparison makes sense only on the level of operational semantics $\mathcal{O}_i$.

- John had a question concerning the sentence "If the other path of s_2 (...) is finite, we may assume w.l.o.g. ..." on p. 3.11. I believe it is needed to find a word $v \in \mathcal{O}_1^* [\![s_2]\!]$ with $v \in \{u\} \| ^{\mathcal{B}_1} \{v\}$ in that case.

- 2 -

ad BM0: - p. 2, bottom line: it should read "... is proved fully in sections 5 and 7."

- general: If a smaller symbol $\sqsubseteq$ is too difficult to produce, let us use $\subseteq$ throughout, perhaps with an additional comment to the reader/printer that $\subseteq$ stands for $\sqsubseteq$!
- p. 10, line 7: $(P_{sat}(Obs(A)), \supseteq, Obs(A))$
- p. 10, middle: $\Psi(X) = \bigcup_{w \in X} \Psi(w)$
- p. 20, line 7: "... no infinite chain

$$\ldots \to x_2 \to x_1 \to x_0$$

of elements $x_i \in W$."

- p. 21: reference BM02 has appeared.

I hope the best for our BM02: Jeff told me that John will take care of the final integration and convergence of the corrections.

Best regards,

Ernst-Rüdiger Olderog

P.S. A copy of the new pages p. 4.3/4.3a and R3 will be mailed to Jeff at Buffalo.

P.P.S. Starting from 20 July I stay for three weeks at IBM.

Beste Jaco,

Nu de dag nadert dat je van de FPU gebruik wil maken wil ik vanuit de bibliotheek vaststellen dat je al vrij vroeg een belangrijke bijdrage hebt geleverd aan de opbouw van de collectie. Nader onderzoek heeft namelijk uitgewezen dat je aan het begin van je carrière bij het CWI, voorheen het MC, al nauw was betrokken bij de opbouw van de bibliotheek. In de zestiger jaren heb je voor de bibliotheek reeds belangrijke werkzaamheden verricht, nl. het beoordelen, adviseren en classificeren t.a.v. de aanschaf van nieuwe publicaties in het destijds nog vrij nieuwe vakgebied informatica.

In de tussentijd heeft dit vakgebied een grote ontwikkeling meegemaakt ook voor het CWI. Het heeft ertoe geleid dat de bibliotheekaanwinsten tegenwoordig voor tenminste de helft uit informatica literatuur bestaat en het heeft zelfs tot de naamswijziging van het instituut geleid.

Vanuit je verschillende functies heb je verder altijd grote betrokkenheid met de bibliotheek getoond. Mede namens alle medewerkers wil ik je graag bedanken voor je bijdrage aan de bibliotheek. Ik hoop dat je hierna zult genieten van de vrije tijd en het kunnen doen en laten wat je gewoon leuk vindt.

Ay Ong

Het Verhaal van het CWI

Net als iedere organisatie heeft het CWI zijn eigen "verhaal", zijn eigen geschiedenis. Dat verhaal is meer dan een verzameling feiten. Het is bepalend voor de identiteit van de organisatie. Het vertelt wie we zijn, wat onze waarden zijn, wat ons drijft, waarom we zijn wie we zijn. Daarom moeten we dat verhaal koesteren en doorvertellen. Collega's die lange tijd op het instituut doorbrengen zijn belangrijk voor dat verhaal. Ze schrijven als het ware zelf een belangrijk deel van het verhaal, maar ze vormen ook het geheugen van de organisatie. Daarom verliezen we met de pensionering van Jaco de Bakker niet alleen een uitstekend wetenschapper maar ook een stukje geheugen van het instituut. Jaco heeft 38 jaar op het instituut doorgebracht. In die tijd is er heel veel gebeurd en veel veranderd.

Jaco heeft de pioniertijd van de informatica meegemaakt, de snelle ontwikkeling en het volwassen worden van het vakgebied. Ook de ontwikkeling van de bestuurlijke inbedding, het ontstaan van SION, de groeistuipen en de spanningen die zo nu en dan optraden tussen instituut en landelijke activiteiten.

Maar ook op het CWI is veel veranderd.
Dat blijkt bijvoorbeeld als je de oude jaarverslagen doorleest.
Wat allereerst opvalt is dat de manier van verslaglegging enorm veranderd is. De vorm: nu een fraai geillustreerd PR-document in kleur. In de tijd dat Jaco bij de SMC in dienst kwam bestond het verslag uit een droge tekst zonder plaatjes. Maar vooral de inhoud is veranderd. Waar we nu veel waarde hechten aan transparantie, aan het duidelijk maken wat de output van het instituut is en met welke input die tot stand is gekomen, geven de oude verslagen een onvolledige opsomming van details zonder een duidelijk beeld van het geheel.
Soms worden feiten vermeld die misschien wel aardig zijn maar niet zo erg relevant. In het jaarverslag over 1964 is bijvoorbeeld te lezen dat "de directiesecretaresse i.v.m. haar huwelijk haar werkzaamheden tot een half-time functie wenste te beperken."
Terwijl zaken die je wel zou willen weten juist ontbreken, zoals wat de algemene ontwikkelingen waren, wat de strategie van het instituut was en hoe die paste in de omgeving. Hoe groot het budget was en hoe de financieringsstructuur in elkaar zat.
Het verslag over het jaar 1971 geeft voor het eerst wat financiële cijfers. Het budget was toen Fl 5 miljoen. Maar opmerkelijker is waar de inkomsten vandaan kwamen. ZWO, de voorloper van NWO, leverde slechts de helft van de inkomsten. Niet minder dan 25% werd betaald door de beide Amsterdamse universiteiten, en de rest bestond uit overige subsidies, cursusgelden en vergoedingen voor verrichte diensten.

Het CWI is in elk geval flink gegroeid tijdens de loopbaan van Jaco. Toen hij in 1964 bij het toenmalige Mathematisch Centrum kwam, waren er 85 medewerkers. Nu zijn het er 185, gedetacheerden niet meegerekend.
Ook in het werk is er veel veranderd. Het werk vindt meer en meer in de vorm van projecten plaats. Er wordt meer en intensiever samengewerkt. Natonaal, met universiteiten, onderzoekscholen, Telematica Instituut, WTCW, maar ook in Europees verband.

Onderzoeksmanagement stelt andere eisen dan vroeger. Er moet meer geanticipeerd worden op externe ontwikkelingen. Financiering moet verworven worden in competitie met vele anderen, op basis van goede plannen. Ook Human Resource Management is steeds belangrijker geworden voor het succes van het instituut. Bij al deze ontwikkelingen is Jaco de afgelopen jaren als leidinggevende en lid van het managementteam nauw betrokken geweest.

Het onderzoek is meer multidiciplinair geworden, we werken bijvoorbeeld aan toepassingen in de levenswetenschappen, en ook meer toepassingsgericht .
Er is veel aandacht voor de exploitatie van resultaten van onderzoek, we hebben een actief spin-off beleid dat jaarlijks een nieuw bedrijf oplevert en samenewrking met bedrijven is belangrijk.
Ja, er is veel veranderd. Toch? Hoewel.....Als ik dan het jaarverslag van 1964 lees ga ik twijfelen. Want in het hoofdstuk "Gebouw" kan men lezen: "Door het vertrek van Electrologica in het voorjaar is vrijwel het gehele gebouw wederom ter beschikking gekomen van het Mathematisch Centrum". Ook toen al was er sprake van een spin-off bedrijf.
En de Afdeling Mathematische Statistiek deed in 1964 onderzoek op allerlei toepassingsgebieden, zoals:
- de relatie huisarts-patient
- straftoemeting bij havendiefstal
- lengten van padden
- aspirine en bloedstolling
- de cyclus van vrouwen

Ook toen dus al multidisciplinair onderzoek, en zelfs al levenswetenschappen! Er lijkt niets nieuws onder de zon, maar misschien stuiten we hier wel op een paradox. Het CWI verandert voortdurend, maar er is ook een onderstroom, die zorgt voor de continuïteit: we doen vernieuwend onderzoek van uitstekende kwaliteit, maar wel vanuit een maatschappelijke betrokkenheid en in interactie met onze omgeving. Dat is onze identiteit. Dat is ons verhaal, een mooi verhaal, en dat verhaal gaat door.

Gerard van Oortmerssen

Title: L'influenza di Jaco nel mio sviluppo professionale

By: Catuscia Palamidessi, Penn State University.

Ho conosciuto Jaco nel 1986, in veste di iniziatore e leader del progetto Esprit 415. Tale progetto comprendeva una decina di siti di vari paesi europei, fra cui il gruppo di Logic Programming dell'Universita` di Pisa. A quei tempi io ero una dottoranda in tale gruppo, e i miei interessi erano vicini ai temi sviluppati nell'ambito di Esprit 415, per cui divenne naturale la mia partecipazione ai meetings del progetto come una rappresentante del gruppo pisano.

Di tutte le esperienze scientifiche e professionali fatte in quegli anni, direi che tali meetings sono stati in assoluto i piu` stimolanti a importanti per il mio sviluppo professionale. Ricordo con molta nostalgia le presentazioni di nuovi risultati da parte dei vari partners, e le discussioni che ne seguivano. C'era, palpabile, la sensazione di stare costruendo qualcosa di importante, di nuovo, e l'entusiasmo e lo spirito di collaborazione che percepivo ebbero un peso enorme nello sviluppo del mio interesse per la ricerca, che determino`, al termine del dottorato, la mia scelta per la carriera accademica. Penso che il progetto abbia avuto un'influenza analoga su molti altri dei giovani partecipanti.

Dopo tutti questi anni spesi nella ricerca, e dopo aver lavorato in molti altri progetti, e` facile per me riconoscere che il successo dei meetings e del progetto Esprit 415 fu in grandissima parte dovuta alle capacita` direttive di Jaco. Il suo senso di finalita`, il suo apprezzamento per l'eccellenza scientifica, il suo giudizio sicuro, costituivano per tutti noi motivo di stimolo e un sistema di riferimento prezioso.

Quando, dopo la fine del dottorato, scelsi di andare al CWI per un periodo di Postdoc, Jaco accolse con benevolenza il mio desiderio di partecipare ai meetings settimanali del suo gruppo. E` stato grazie a tali meetings, oltre alla collaborazione scientifica con alcuni dei membri del gruppo (Frank de Boer, Joost Kok e Jan Rutten), che e` nato e si e` consolidato il mio interesse per la Teoria della Concorrenza, un interesse a cui devo la maggior parte dei miei successi e delle mie gratificazioni professionali.

Penso spesso a Jaco con molta gratitudine ed affetto, e vorrei cogliere questa occasione per esprimergli un caloroso "grazie", per aver creato le condizioni ottimali per accrescere il mio background scientifico e affinare le mie capacita`. Se oggi mi sento realizzata professionalmente, lo devo in grandissima parte a lui.

Grazie di tutto, Jaco, e tanti, tantissimi auguri.
-Catuscia

GESPREKSFORMULIER FUNCTIONERINGSGESPREK

Naam medewerker: **Jaco de Bakker**
Naam gesprekspartners: **Personeelsdienst**
Datum gesprek: **29 augustus 2002**

In het afgelopen jaar gewijzigde functie-elementen:

- *Overdragen van werkzaamheden*
- *Nadenken over vrijetijdsbesteding*

Stand van zaken m.b.t. afspraken uit vorige gesprekken:

- voorstellen c.q. wensen voor loopbaanperspectief:
  - o *geniet van je nieuwe vrije tijd*
  - o *neem afstand*
  - o *laat je werkzaamheden geleidelijk aan los*
- afspraken nagekomen:
  Ja, want we vieren jouw afscheid op 30 augustus 2002
- nieuwe acties te nemen door:
  Jaco
  Opmerking: lees vooral aandachtig verder

Werkzaamheden waar momenteel het accent op valt:

- *Onthaasten, loslaten en ontkoppelen*

Opmerkingen t.a.v. de functievervulling:

- *Wij hebben je leren kennen als een zorgzame, vriendelijke en accurate leidinggevende wetenschapper met oog voor detail ten aanzien van je medewerkers en de organisatie*

Opmerkingen t.a.v. de samenwerkingsrelatie vanuit de gesprekspartners:

- *Hou in je vrije tijd dezelfde goede samenwerkingsrelatie met je collega's als nu*
- *Wij gaan je missen*

Opmerkingen/meningen over omstandigheden die het functioneren beïnvloeden of hebben beïnvloed:

- *Jouw opgebouwde expertise is van onschatbare waarde en moeilijk te vervangen*

Ontwikkelingsmogelijkheden in je nieuwe levensfase buiten de eenheid:

- opleiding/studiefaciliteiten:
  - o *volg de cursus 'zicht op pensioen'*
  - o ons advies:
    - > *raadpleeg voor algemene informatie:* SeniorWeb.nl
    - > *raadpleeg voor een leuke opleiding:* hovoutrecht.nl
- loopbaanperspectief (vanuit werkgever):
  - o *reizen maken t.b.v kennisuitwisseling*
  - o *opzetten meldpunt 'hoogleraren met emeritaat'*
  - o *schrijven van een thriller a la 'het Bureau'*

Opmerkingen t.a.v. andere aspecten:

- *Geniet van je nieuwe vrije leven samen met je geliefden*
- *Overweeg de aanschaf van een hond, ter bevordering van de lichaamsbeweging en van nieuwe sociale contacten*

WENSEN

> *genieten van de festiviteiten rondom je afscheid*
> *een actieve nieuwe levensfase in goede gezondheid*

Gesprekspartners Personeelsdienst:

José Koster

Katelijn Arnold

Piet Beertema

Miriam Gravemaker

Marja Hegt

Marlin van der Heijden

Een wijs woord en een nieuw idee.

Han La Poutré

Ik ben bijna vijf jaar geleden begonnen bij het CWI, als themaleider van een nieuwe pilotgroep. Jaco de Bakker en ik hadden daar al enige gesprekken over gevoerd, en in september 1997 was het zover: ik begon aan mijn nieuwe missie. De nieuwe groep, die "Evolutionary Systems and Applied Algorithmics" ofwel SEN4 zou gaan heten, was een nieuw idee binnen het CWI, en in het bijzonder binnen het cluster van Jaco.

Jaco had zich sterk gemaakt voor deze "geheel nieuwe pilot", die voor een groot deel nog verder moest worden vormgegeven. Bij mijn start waren Cees van Kemenade en Michiel van Wezel al aan het werk op het beoogde onderwerp van de groep, met Joost Kok als adviseur.

In de pilotfase van de groep heeft Jaco zich als de ervaren "sparring partner" laten zien. Hierbij kwam de zinsnede "even een wijs woord", met daaropvolgend een reeks van ervaringen en inzichten, veelvuldig voor. In de begintijd was dat vooral tijdens het overleg dat we hadden. Na enige tijd werd dat natuurlijk minder frequent, waarbij Jaco zich meer en meer op de achtergrond hield. Desalniettemin bleven de wijze woorden niet uit. Niet zozeer omdat Jaco mij die opdrong, als wel omdat ik de ervaringen en wijsheid steeds meer ging waarderen en ik altijd beroep op hem kon doen.
Ondertussen had ik al die tijd alle vrijheid van handelen en beslissen. Dat zeker. Maar met een schat aan wijze woorden....

Het nieuwe idee was er niet alleen bij de start van de groep SEN4. Al meteen vanaf mijn start bij het CWI had ik interesse in economie als applicatiegebied, in het bijzonder de agent-based computational economics of evolutionaire economie. Later groeide dit uit tot e-business applicaties en economische software agenten. Hierbij waren met name economische, sociale en maatschappelijke disciplines en ontwikkelingen van belang; iets dat binnen de reguliere informatica nog maar weinig aandacht had gehad. In al deze ontwikkelingen heeft Jaco op stimulerende wijze grote interesse getoond en zich open gesteld voor het nieuwe idee en het nieuwe initiatief. Hij bleek de exploratie van nieuwe disciplines vanuit de SEN4-technieken zeer toe te juichen en had vertrouwen in de toekomst...

Blij dat dit kon. Vernieuwingszin in combinatie met ervaring.

Het wijze woord en het nieuwe idee.

Wiskunde van het programmeren

Martin Rem
Embedded Systems Institute
Technische Universiteit Eindhoven
m.rem@tue.nl

De voorganger van de VSNU was de Academische Raad. De Academische Raad kende secties, waarvan de Sectie Informatica – de ARSI onder leiding van Alexander Verrijn Stuart – er een was. Begin 1980 liet de minister van onderwijs aan de ARSI weten dat hij overwoog om in Nederland vier studierichtingen informatica te stichten, waarvan er een in Twente zou worden gevestigd. Daartoe had hij het land in vier gebieden verdeeld, ruwweg gegroepeerd rond de technische universiteiten en Amsterdam. De studierichtingen zouden er alleen komen als de universiteiten tot een taakverdeling in het wetenschappelijk informatica-onderzoek konden besluiten.

Ik herinner me levendig de discussies in de ARSI, waar zowel Jaco als ik in die tijd deel van uitmaakten. Eindhoven had lange tijd op het standpunt gestaan dat informatica eigenlijk een tak van wiskunde was en claimde dan ook het gebied "mathematische informatica", naar analogie van de mathematische statistiek. Jaco vond dat hij en anderen rond hem toch ook op een wiskundige wijze de informatica bedreven en keurde de term als zijnde te breed af. Hij ondervond hierin vrij algemene steun. Ik denk dat verzet tegen de Eindhovense brief van de "bende van vier" ook nog een rol speelde in de besluitvorming van veel ARSI-leden. Jaco was degene die met de oplossing kwam. Toen uit de discussie bleek dat Eindhoven vooral wilde werken aan wiskundige methoden om het ontwerpproces te ondersteunen, opperde hij de term "wiskunde van het programmeren".

Zo is het ook in het rapport van de ARSI terechtgekomen. Ik weet niet of de minister de taakverdeling ooit serieus heeft genomen; hij heeft het niet laten merken. Ik weet wel dat het niet bij vier studierichtingen is gebleven; al gauw waren het er negen. Wat meer is beklijfd, is Jaco's term "wiskunde van het programmeren". Het werd bijvoorbeeld de naam van de leerstoel van Roland Backhouse, en in zijn Engelse vertaling (Mathematics of Program Construction) werd het de naam van een succesvolle congresserie.

Reduction steps in partitions

Gerard R. Renardel de Lavalette, Wim H. Hesselink*

Dedicated to Jaco de Bakker, at the occasion of his goodbye to CWI

Abstract

This short note starts with some personal words to Jaco de Bakker. They are followed by two problems in recreational mathematics about the length of reduction paths, consisting of partitions of some natural number n. One of these problems came up in an attempt to prove the famous conjecture P = NP. Solutions are given in the Appendix.

Dear Jaco,

we start with some personal words, from each of us.

Wim: we first met in February 1986 when I gave a talk at the CWI on unbounded nondeterminacy of recursive pocedures. Some years later, you encouraged and stimulated me to write a book on that subject for Cambridge University Press, where you were on the Editorial Board.

Gerard: I first met you on 28 April 1989, at the celebration of your 25-year jubilee at CWI. I had entered Computer Science a few years before, with a background in mathematical logic. You were (and still are) for me one of the legendary examples of mathematicians with great impact on computer science. As a logician, I appreciate your groundbreaking work for semantics: it relates computer science in a seminal way to logic and linguistics.

This contribution to your *Liber Amicorum* is about two related problems of a recreational-mathematical nature. The first problem arose from an attempt by A.D. Plotnikov to prove P = NP, via an efficient algorithm for the minimal clique partition problem. The second problem is a mutation of the first, created by an error in copying it. Solutions are given in the Appendix.

Reduction steps in partitions

$P(n)$ denotes the collection of partitions of natural number n. Partitions will be represented by weakly decreasing sequences. So we have, e.g.

$$P(5) = \{5, 41, 32, 311, 221, 2111, 11111\}$$

For reasons of mathematical smoothness, we make these sequences infinite by adding an infinite tail of zero's. The lexicographical order $<_L$ is defined as usual: $a <_L b$

*University of Groningen, Department of Computing Science, Groningen, the Netherlands, email {grl,wim}@cs.rug.nl

iff, for some k, $a_k > b_k$ and $a_i = b_i$ for all $i < k$. In addition, we define the *tail ordering* $\geq_T$ by

$$a \geq_T b \ =_{\text{def}} \ \forall n \ a_n \geq b_{n+1}$$

i.e. a is pointwise greater than or equal to the tail of b. With $<_L$ and $\geq_T$, we define two reduction relations on $P(n)$:

$$a \to b \ =_{\text{def}} \ (a <_L b \wedge a \geq_T b)$$

$$a \Rightarrow b \ =_{\text{def}} \ (a <_L b \wedge b \geq_T a)$$

We observe that $\to$ nor $\Rightarrow$ is transitive:

$$1111 \to 211 \to 22, \text{ but } 1111 \not\to 22$$

$$222 \Rightarrow 321 \Rightarrow 411, \text{ but } 222 \not\Rightarrow 411$$

Now let $p(n)$ be the number of partitions of n, $r_1(n)$ the maximal length (i.e. number of nodes) of a $\to$-chain in $P(n)$, and $r_2(n)$ the maximal length of a $\Rightarrow$-chain in $P(n)$. We computed the first values of these functions:

n	1	2	3	4	5	6	7	8	9	10	11
$p(n)$	1	2	3	5	7	11	15	22	30	42	56
$r_1(n)$	1	2	3	5	7	11	15	19	25	33	42
$r_2(n)$	1	2	3	5	7	10	14	19	25	33	43

Observe that, for $n \leq 5$, the one-step lexicographical order in $P(n)$ equals both $\to$ and $\Rightarrow$. In $P(6)$ this symmetry is broken by the fact that $222 \not\Rightarrow 3111$; in $P(8)$, we have $2222 \not\to 3111111$, $332 \not\to 41111$ and $44 \not\to 5111$.

We have $\log(p(n)) = \Theta(\sqrt{n})$ (see [2, p. 278]), so the asymptotic behaviour of $p(n)$ is subexponential. The question is: what is the asymptotic behaviour of r_1 and r_2?

Problem 1. Is the length of the maximal $\to$-reduction path polynomial in n, i.e. is r_1 polynomial?

Problem 2. Is the length of the maximal $\Rightarrow$-reduction path polynomial in n, i.e. is r_2 polynomial?

History. In 2000, the paper *An Efficient Algorithm for the Minimum Clique Partition Problem* by Anatoly D. Plotnikov was available on www via the address `www.busygin.dp.ua/clipat.html`. It claimed to contain a polynomial-time algorithm for the Minimum Clique Partition Problem (MCPP). This is the problem of finding a partition of a finite graph into the minimum number of cliques. (It is equivalent to the Minimum Graph Coloring Problem for the complementary graph.) Since MCPP is NP-hard, this would provide a positive answer on the long-standing open question P =? NP.

In a review of the paper (then available at the same www-address), Stas Busygin wrote:

> In general, the paper correctly describes a new conception on graphs (so-called vertex-saturation) and on the basis of it presents a new NP-complete problem (the MPP problem) heavily related to the minimum clique partition problem. However, the author fails not only to show polynomial time solvability of the MPP problem (that is doubtful after all) but also to clarify its polynomial equivalence to the considered NP-hard problem.

The second author (Hesselink) studied Plotnikov's paper, with the intention to transform the good ideas into correct algorithms (see [1]). In the analysis of the time complexity of one of the subalgorithms, he came upon the definition of the reduction relation $\rightarrow$ and the function r_1, and conjectured that $r_1(n)$ was bounded by n^2. He discussed $\rightarrow$ and r_1 with the first author (Renardel), and the latter tried to answer the question of the polynomiality of r_1. However, in doing so he inadvertedly transformed it into another problem, viz. the polynomiality of r_2 based on $\Rightarrow$! So when he presented the solution to Hesselink, it became clear that the wrong problem had been solved. Renardel tried again, and finally succeeded in solving the original problem, too.

In the meantime, Plotnikov's paper mentioned above is no longer available at www. The address `www.busygin.dp.ua/clipat.html` now (May 2002) refers to a thorough revision of the paper, under the title *Four NP-Complete Problems*. From Busygin's comment (at the same address), we quote:

> Now, after about 1.5 years, the author has reworked a significant part of his manuscript – where the flaws had been found – and extended the proposal to the maximum independent set problem as well. Definitely, the publication should NOT be taken as the final P=NP conclusion and each of its claims is subject to a thorough questioning and discussion. Moreover, I completely understand those who will abstain from spending own time and energy on analyzing the proposal on the basis of its probable falseness. [...] However, I do consider the work of Anatoly Plotnikov on the famous NP-hard graph problems interesting enough to be presented here and invite everyone working in the computational complexity field to participate in its investigation.

References

[1] W.H. Hesselink, *The borderline between P and NP*. To be obtained from `www.cs.rug.nl/~wim/pub/mans.html`

[2] H. Rademacher, *Topics in Analytic Number Theory*. Springer-Verlag, Berlin, 1973.

Appendix

Solution of Problem 1

We claim that, for all $n \geq 1$:

there is a reduction path of length 2^n in $P((3n^2 + n)/2)$

So $r_1(n)$ is not polynomial: like $p(n)$, it is of order $2^{\sqrt{n}}$.

The construction of the reduction path is based on three ideas. Firstly, we observe that the path consisting of the first 2^n binary numbers $0 \ldots 0$ to $1 \ldots 1$ is in lexicographical order. For $n = 3$:

000 001 010 011 100 101 110 111

However, these sequences are not weakly decreasing and the path is not in tail order. We apply the second idea: add, to every sequence in the path, pointwise the sequence $(n-1)\ (n-2)\ \ldots\ 1\ 0$. The resulting sequences are weakly decreasing, and the resulting path is in lexicographical order and in tail order, e.g.:

210 211 220 221 310 311 320 321

But the sequences in the path do not represent partitions of the same number. The third idea resolves this: add, at the end of every sequence in the path, the number of zeroes of the binary number that was used in its definition; moreover, add n to the other numbers in the sequence in order to keep it weakly decreasing. This leads to a $\rightarrow$-reduction path, e.g. for $n = 3$ in $P(15)$:

$$5433 \rightarrow 5442 \rightarrow 5532 \rightarrow 5541 \rightarrow 6432 \rightarrow 6441 \rightarrow 6531 \rightarrow 654$$

In the general case, each sequence is a partition of $(\Sigma_{i=0}^{n-1} i) + n^2 + n = 3(n^2+n)/2$.

Solution of Problem 2

We claim that, for all n:

there is a $\Rightarrow$-reduction path of length 2^n in $P((n^2+n+2)/2)$

So $r_2(n)$ is not polynomial: like $p(n)$, it is of order $2^{\sqrt{n}}$.

We find it convenient to reverse the order here and to work with $\Leftarrow$. The proof is based on the following observation

$$x\ (m+1)\ 1^p \Leftarrow x\ m\ m\ 1^{p-m+1}$$

where x is some finite weakly decreasing sequence of numbers $\geq m+1$, and 1^p denotes the sequence of p 1's. With induction over m we can prove:

if k is sufficiently large, then there is a $\Leftarrow$-reduction path
of length $2^m + 1$ from $x\ (m+2)\ i^k$ to $x\ (m+1)\ 1^{k+1}$ (1)

The induction step is proved by first using

$$x\ (m+2)\ 1^k \Leftarrow x\ (m+1)\ (m+1)\ 1^{k-m}$$

followed by applying the induction hypothesis $m-1$ times, on $(x\ (m+1)\ p\ 1^{k-p+1})$, where $m+1 \geq p \geq 2$. Concatenate the reduction paths obtained this way, and we get the desired reduction path. For an example, we take x empty, $m = 2$ and $k = 3$:

$$4111 \Leftarrow 331 \Leftarrow 322 \Leftarrow 3211 \Leftarrow 31111$$

Now apply (1) n times with x the empty sequence and m running from $n-1$ to 0, and concatenate the reduction paths. This yields a reduction path of length 2^n from $(n+1)\ 1^k$ to 1^{k+n+1}, and we only have to determine the 'sufficiently large' value of k for which this all works well in $P(k+m+1)$. It is quite obvious that the critical position in the path, where the exponent of 1 takes its lowest value, is the sequence $n\ (n-1)\ (n-2) \ldots 3\ 2\ 2\ 1^{k-(n^2-n)/2}$. So $k = (n^2-n)/2$ is just large enough: then all sequences are in $P(k+n+1) = P((n^2+n+2)/2)$. For $n = 3$ this yields the reduction path

$$4111 \Leftarrow 331 \Leftarrow 322 \Leftarrow 3211 \Leftarrow 31111 \Leftarrow 22111 \Leftarrow 211111 \Leftarrow 1111111$$

van Herman te Riele

Bij het afscheid van een sympathieke collega Amsterdam, 27 mei 2002

Beste Jaco,

Gedurende vele jaren zijn jij en ik overburen geweest in de oude locatie van het CWI aan de 2e Boerhaavestraat. We groetten elkaar altijd heel vriendelijk, maakten wel eens een praatje, en toen je hoogleraar werd, werden Toke en ik gastvrij bij jou thuis uitgenodigd om dat mee te komen vieren. Toch zijn onze contacten altijd wat aan de oppervlakte gebleven, maar dat kan goed verklaard worden uit het feit dat wij op verschillende gebieden van de informatica resp. wiskunde werken.

Op de nieuwe locatie aan de Kruislaan zijn we elkaar een beetje uit het oog verloren, behalve dan dat ik je regelmatig op de fiets het WCW-terrein zag binnenkomen of verlaten, terwijl ik met hetzelfde bezig was.
Jij woont in Amsterdam-Zuid en ik fiets meestal naar/van station Diemen-Zuid, dus onze fietswegen vallen maar een heel klein stukje samen, wat de kans op contact ook al niet groter maakt.

Onlangs kreeg ik via de bibliotheek een exemplaar van jouw proefschrift in handen met een harde kaft: toentertijd was het de gewoonte om enkele exemplaren van het proefschrift speciaal met zo'n harde kaft uit te geven, bv. voor de promotor. Voorin dit exemplaar staat de handtekening van A. van Wijngaarden dus ik neem aan dat dat het exemplaar van jouw promotor is... Kopen? ;-)

Stelling IX daarin luidt:

"Logische fouten in programma's die gesteld zijn in een programmeertaal
"zoals ALGOL 60 of FORTRAN hebben vaak tot gevolg dat de waarde van een
"variabele wordt gevraagd voordat een waardetoekenning aan deze variabele
"heeft plaatsgevonden. Voor testdoeleinden is het daarom van belang dat
"men over de faciliteit beschikt om een programma op zodanige wijze uit
"te voeren dat dergelijke situaties worden gedetecteerd en een geschikte
"foutmelding wordt gegeven.

Om eens te kijken of de suggestie die jij in deze stelling hebt gedaan is opgevolgd, heb ik het volgende simpele FORTRAN programmaatje geschreven:

```
program jacoIX
  print *, i
  print *, j
  stop
end
```

Dit wordt zonder foutmelding gecompileerd bv. met f77 en met f90 op het medusa systeem van SGI, en met g77 op een SUN workstation.
Na compilatie met f77 op de medusa krijg ik de uitvoer:

```
      0
 183999856
```

en na compilatie met f90 op de medusa, en g77 op mijn SunBlade 100 workstation:

```
0
0
```

Geen foutmeldingen dus, en ik heb niet kunnen ontdekken, ook niet na raadpleging van Dik Winter, of er opties zijn om dit soort situaties tijdens de uitvoering van het programma te detecteren en dan een foutmelding te genereren.

Ligt hier misschien nog een taak voor je voor na je terugtreden?

Hoe dan ook: ik wens je nog vele gezonde en goede jaren toe met al degenen die je dierbaar zijn.

Herman te Riele

Datum: augustus 2002

Promoties, Geloof en onze Verhouding

Door: Reind van de Riet

Beste Jaco en Angeline,

Bij je afscheid van de VU heb ik, tijdens een gezellig diner, je toegesproken. Bij je afscheid van het CWI is mijn bijdrage tot dit tweede Liber Amicorum, LB2, gebaseerd op die toespraak. Uiteraard is het niet hetzelfde, zo zijn er foto's toegevoegd en is de tekst hier en daar aangepast.

Het is niet moeilijk om onderwerpen te bedenken waarover ik het zou kunnen hebben, want we hebben bijna 40 jaar samen gewerkt en we kennen elkaar van de collegebanken sinds 1956, dat is dus al 45 jaar geleden! Alle reden om een verhaal op te hangen rond het thema: "Weet je nog wel oudje?". Aan dat thema kan ik natuurlijk niet ontkomen, maar ik wilde het ook over andere dingen hebben, namelijk promoties, het geloof en onze verhouding. Je ziet ik schuw breedheid niet terwijl ook de diepte aandacht geschonken krijgt, uiteraard een drietal thema's, vanwege het getal drie waar ik twee jaar geleden aandacht aan besteedde (overigens toen ging het om vier thema's met elk drie subthema's). Mijn gereformeerde achtergrond zegt dat een goede preek altijd uit drie punten bestaat.

Allereerst iets over de foto's waar dit verhaal mee begint. Op de linker foto sta jij afgebeeld in een karakteristieke pose (linker hand in de zij), terwijl je over mijn proefschrift kritische vragen stelt, tijdens mijn promotie op 7 februari 1968 in de Lutherse kerk, op dat moment de aula van de Universiteit van Amsterdam en de rechter foto, gemaakt door Andy Tanenbaum, tijdens het diner dat mij werd aangeboden ter gelegenheid van mijn afscheid. Dat was dus een goede dertig jaar later. Om de vergelijking makkelijker te maken zijn de koppen boven elkaar gezet. Opvallende verschillen zijn, afgezien van de kleur, de hoeveelheid haar op je kop: toen een behoorlijke haardos, nu vrijwel kaal, nu een bril toen zonder. Je bent van een knappe jonge vent veranderd in een man waar het succes van afstraalt aan het eind van een zeer succesvolle carrière.

Het materiaal waarop ik dit verhaal voor LB2 baseer bestaat uit twee libers amicorum, nu ja dit meervoud is vast niet OK, maar ik heb ook maar een HBS opleiding gehad. Bij mijn afscheid was jij zo vriendelijk een mooi verhaal over mij in de beginjaren bij het MC en de VU te schrijven, bij jouw 25-jarig dienstverband bij het CWI was ik degene die zo'n verhaal in elkaar zette. Dat doet mij herinneren aan iets pijnlijks: mijn 25-jarig dienstverband bij de VU werd namelijk vergeten. Ik kan daar nu niet meer boos over worden, maar wil een kleine bekentenis doen. Wat was het geval: met mij was Jim van Keulen 25 jaar aan de VU verbonden. Ik heb toen, als voorzitter van de vakgroep, een groots festijn met receptie en kadoo voor hem georganiseerd. Hem in een mooie toespraak geprezen en onze prima samenwerking geroemd. Ik herinner me dat enkele aanwezigen zich realiseerden dat dit ook mijn 25-jarig jubileum was en daar besmuikt een opmerking over maakten.

Maar ja toen was het te laat. Ik heb het mijn collega's alleen maar een beetje kwalijk genomen, en bij mijn afscheid hebben ze het weer helemaal goed gemaakt! Zo was jouw Liber er eerder dan de mijne. Hetzelfde geldt voor onze promoties: daar was je mij een slag voor. Zo zelfs dat jij vragen mocht stellen over mijn proefschrift, maar andersom kon dat dus niet. In een aantal andere zaken was ik jou weer voor: ik studeerde eerder af, was eerder bij het MC aangesteld en was ook eerder bij de VU als hoogleraar aangesteld.

In jouw verhaal in mijn Liber noem je deze zaken ook. Is er misschien toch sprake geweest van enige rivaliteit tussen ons? Het geeft iets weer over onze verhouding. Die begon op de collegebanken in 1956; we waren beiden jong, maar jij was een jaar ouder. Verder was jij lid van een gerenommeerd studenten dispuut, onderdeel van het Corps en ik was nihilist, dwz geen lid van het corps of dispuut. Wel was ik lid van clubs zoal J&E, Jeugd en Evangelie en bezocht samenkomsten van Stromen van Kracht. Ik was erg gelovig, veranderde in mijn studententijd van Gereformeerd naar Baptist. Jouw religieuze belangstelling ging richting agnostiek. Later toen je religieuze achtergrond tegen het licht gehouden moest worden bij je sollicitatie als kandidaat voor je positie bij de VU, speelde dat nog een belangrijke rol; je verwijst er naar in je verhaal in mijn Liber: je spreekt over "langdurige verwikkelingen". Inderdaad, dat was niet gemakkelijk. In die tijd was de VU nog een Gereformeerde universiteit, hoewel veranderingen al wel op komst waren. Ik herinner me dat ik zelf bij het bestuur van de VU moest komen bij mijn benoeming. Ik had me geprepareerd op een verhoor over mijn overgang van Gereformeerd naar Baptist en wist precies uit te leggen wat de "drie formulieren van enigheid" waren, maar de voorzitter van dat bestuur de heer Meinen, tevens voorzitter van de Raad van Bestuur van Aku (Nu Nobel-Akzo), vertelde dat hij vaak bij vrienden in Amerika kwam en die waren Baptist, met mij zat het dus toen ook wel goed. Zo makkelijk ging het niet met jouw benoeming. Wij van de subfaculteit moesten laten zien dat het merendeel van de staf de Gereformeerde of eveneens gerespecteerde denominaties als de Baptistische beginselen aanhingen, dus er was een lijst met o.a. Maarten Maurice, Rien Kaashoek, Gerke Nieuwland, Rikus Kok en mijn naam. Zo ging dat toen, maar jouw benoeming kwam er door, dat was het belangrijkste. Het zou interessant zijn na te gaan hoe de grondslag van de VU en de daaruit voort vloeiende doelstelling hebben gefunctioneerd in die jaren en hoe nu, waar de theologische faculteit bijna een Iman/m opleiding binnen haar poorten had gekregen. Jammer dat dit niet is doorgegaan, Imam/Geestelijk Raadsman Haselhoef had er veel kunnen leren.

Terug naar jouw verhaal in mijn Liber: over promovendi en het contact dat we onder leiding van van Wijngaarden hadden in het restaurant Copenhagen over drie informatica groepen in Amsterdam en twee opleidingen. Je maakt een opmerking over een discussie over de merites van het proefschrift van mijn eerste promovendus Frank Teer: "Hieruit citeren zou niet passen in een Liber Amicorm" schrijf je. Typisch jouw stijl: op bedekte wijze aangeven van kritiek. Uiteraard zal ik nu niet via dit verhaal met je gaan discussiëren over de juistheid van die kritiek, wel wil ik opmerken dat jij met succes als promotor wist op te treden van medewerkers die met mij samen werkten in het onderwijs, die later hoogleraar werden, dit waren: Willem Paul de Roever, Arie de Bruin en Anton Nijholt.

In het begin van mijn verhaal heb ik aangegeven dat ik in een aantal aspecten eerder was dan jij, maar er is een aspect waar je mij veruit de baas was: wetenschappelijke status. Het is o.a. te zien aan de twee Libers: jouw liber bevat 31 wetenschappelijke artikelen, terwijl de mijne er 7 bevat, waarvan overigens wel één over de zonen van Johann Sebastian Bach. Dat jij benoemd werd tot lid van de KNAW moet hier natuurlijk ook vermeld worden, en dat is wetenschappelijk gezien van een veel hoger niveau dan mijn status van voorzitter van de vakgroep, van SION of van SIKS. Dat jij in jouw bijdrage aan mijn Liber niet alleen mijn bestuurlijke kwaliteiten prijst, maar ook mijn wetenschappelijke heb ik erg gewaardeerd.
Graag wil ik op mijn beurt hier melden dat je op bestuurlijk gebied heel veel gedaan hebt; ik noem in het bijzonder de oprichting van de werkgemeenschap Theoretische Informatica als voorloper van SION.
Aan een behoorlijk intensieve samenwerking is nu een eind gekomen, en dat vind ik jammer want het was erg plezierig. Ik wens jullie, en nu spreek ik ook Angeline toe, een aangename en gezonde tijd toe als gepensioneerde(n). Geniet van je welverdiende pensioen, ga op reis en doe (net als wij: Dinie en ik) dingen waar je vroeger niet aan toe kwam; bijvoorbeeld gezellig eten, zoals tijdens het diner op mijn afscheid waar Andy (alweer onze hoffotograaf) een mooie foto van maakte die hier wordt afgedrukt:

Het ga je goed

Jaco de Bakker:
Herinneringen van Willem-Paul de Roever uit 1969-1993

Met Jaco de Bakker maakte ik kennis in 1969; hij verklaarde zich toen bereid mijn promotieonderzoek te begeleiden. En in 1993 vond de laatste door Jaco, Gregorz Rozenberg en mij georganiseerde REX workshop plaats, na tien jaar met elkaar samengewerkt te hebben. Uit de tussenliggende periode wil ik nu enige herinneringen ophalen.

Jaco de Bakker's rol was in die tijd van invloed op een breed gebied van de Nederlandse Informatika door zijn samenwerking met Gregorz en mij in het door ons gezamelijk georganiseerde Landelijk Project Concurrency en het eropvolgende REX project.

Dit blijkt onder andere uit de ook nu nog (in 2002, dus 9 jaar na publicatie van het laatste REX deeltje!) zeer hoge plaatsing op de internationale citation index van de REX proceedings (ver voor LICS en CAV). Uit de vele Nederlandse onderzoekers van naam, nu industriele of universitaire medewerkers en hoogleraren, waarvan de carriere toendertijd door de vanuit die twee projecten georganiseerde aktiviteiten richting is gegeven, zoals America, Brinksma, de Boer, de Bruin, Engelfriet, Gerth, Hooman, Huizing, Klein, Kok, Kuiper, Rutten, Vaandrager, en Zwiers. En uit het internationale netwerk waar deze mensen door de vele, in het kader van deze projekten uitgenodigde en op het terrein van concurrency onderzoek werkzame internationale beroemdheden mee in aanraking kwamen, zoals Berry, Clarke, Emerson, Hoare, Jonsson, Lamport, Lynch, Mazurkiewicz, Milner, Plotkin, Pnueli, Sifakis, Stirling en Wolper (een volledige lijst dekt dit gebied gedurende die periode af - allen bezochten zij Nederland). Vele van deze beroemde internationale toponderzoekers op het gebied van Concurrency in de Informatika zijn naderhand in Nederland regelmatige gasten geworden, en beinvloeden de Nederlandse Informatika nog immer.

Dat deze twee landelijke projekten überhaupt op zulk een hoog wetenschappelijk niveau en naar een voor die tijd ambitieuze formule georganiseerd, en door het ZWO/NWO gefinanceerd, konden worden - zij bestonden uit langdurige verblijven aan Nederlandse universiteiten van internationaal gerenormeerde toponderzoekers, vele nieuwe medewerkersplaatsen, regelmatig georganiseerde landelijke concurrency dagen, een ruim bemeten reiskostenpot en workshops waar de internationale top persoonlijk ontmoet kon worden en voordrachten gaf - is grotendeels aan het inzicht en de visie van Jaco de Bakker te danken.

Hij voelde dat de tijd daarvoor rijp was. Dat de financiele speelruimte bij de Nederlandse overheid daarvoor aanwezig was. En dat er op een topformule voor zulk een landelijk samenwerkingsverband gemikt kon worden. Dit inzicht dankt hij aan een nagenoeg onfeilbaar gevoel voor wetenschappelijke kwaliteit, gepaard aan een grote belangstelling voor wat beheers- en bestuursmatig haalbaar is.

In zijn eigen wetenschappelijk onderzoek is dit tot uiting gekomen in een groot aantal publicaties, grotendeels op het gebied van formele methoden in de semantiek van programmertalen. Want dat was het gebied waar zijn wetenschappelijke hart lag en nog steeds ligt. Het zij toegegeven dat dit een beperkt gebied der Informatika betreft. Dat het alleen indirect, als basis voor de konstruktie van elektronische verifikatiewerktuigen, praktisch relevante uitstraling heeft. Doch tegelijkertijd dient te worden beseft dat de huidige bloei van modelcheckers en semi-automatische theorem provers niet zonder een solide fundering in denotationele en operationele semantiek denkbaar is - het terrein waar Jaco's belangstelling in eerst instantie naar uitging. Deze belangstelling heeft hij geerfd van zijn promotor, de Nederlandse informatika-pionier par excellence Prof.Dr.Ir. Aad van Wijngaarden, een der voormalige steunpilaren en directeuren van het eerste uur van het Mathematisch Centrum (waaruit het huidige CWI uit voortgekomen is).

Als promotor stond Jaco de Bakker eveneens garant voor een zeer hoog wetenschappelijk gehalte, waarbij het naderhand een karakteristieke trek van hem bleek te zijn dat hij na een aanvankelijke intensieve begeleiding, nadat zijn toekomstige promovendus de door hem gestelde kwaliteitstoets doorstaan had, die promovendus de vrije hand liet om het, nu verder eigen, onderzoek van laatstgenoemde zelf in te vullen. Ik verwacht dat zijn vele promovendi deze karaktertrek van hem in deze bundel zullen bevestigen.

Was dan alles rozengeur en maneschijn bij Jaco in deze periode? Neen. Het is waarschijnlijk een bewijs van mijn eigen naiviteit in de omgang met mijn medemensen dat ik pas in de omgang met Jaco de Bakker geleerd heb dat het zinvoller is begaafde medemensen niet in termen van wit of zwart, maar in termen van zorgvuldig genuanceerde schakeringen van grijs te beoordelen.

Daartegenover staat de indrukwekkende lijst van begaafde onderzoekers die allen van hem hun stiel geleerd hebben, een brede lijst van veelbelovende wetenschappelijke initiatieven die hij een kans gegeven heeft, zoals recentelijk het onderzoek naar de grondslagen van component-based software architecture op het CWI en het onderzoek op het gebied van coordinatietalen. Maar evenzo het al in 1985 aangevangen zeer succesvolle onderzoek naar de semantiek, specificatie en verifikatie van in de object georienteerde programmeertaal POOL geschreven concurrente systemen, in het kader van het industriele ESPRIT project 415, waarvan Loek Nijman de begaafde coordinator bij Philips was, en Jaco de Bakker wetenschappelijk adviseur.

POOL was het geesteskind van Pierre America. (Nooit is mij duidelijk geworden waar POOL voor stond: voor Pierre's Own Object-Oriented Language of Philips O.-O. Language?) Jan Rutten en Pierre schreven het eerste Nederlandse dubbelproefschrift in de Informatika over de semantiek van POOL, met Jaco als promotor. Frank de Boer promoveerde bij Jaco op een geldige en relatief volledige bewijstheorie voor POOL, en leverde daarmee een internationale pioniersarbied van de eerste rang, voortbouwend op door mijzelf ontwikkelde ideeen. Deze lijst wordt in deze bundel zeker voortgezet.

Deze willekeurig gekozen greep uit onder Jaco's leiding gepromoveerdeonderzoekers-getuigt van Jaco's nagenoeg feilloze intuitie voor wetenschappelijke begaafdheid bij de keuze van zijn promovendi. Zij zijn door hem in de gelegenheid gesteld er toe bij te dragen de brandende fakkel van het onderzoek in de Nederlandse Informatika op hoog wetenschappelijk niveau verder te reiken.

Ik ben hem hiervoor, en voor de zich over vele jaren uitstrekkende wetenschappelijke samenwerking met hem, ten zeerste dankbaar, en ben er zeker van dat ik hierin, uit naam van al zijn promovendi, spreek.

Willem-Paul de Roever, hoogleraar Software Technologie aan de Christian-Albrechts-Universität zu Kiel, voorheen hoogleraar aan de Katholieke Universiteit Nijmegen en de Technische Universiteit Eindhoven.

Kiel, 25 Mai 2002

Faculty of Mathematics and Natural Sciences

LIACS
Leiden Institute of Advanced Computer Science

Prof. Dr. G. Rozenberg
Head Theoretical Computer Science Group
http://www.liacs.nl/~rozenber/

Dear Jaco,

I have much enjoyed our cooperation over the past three decades and certainly profited from it in many ways. We have worked together in many projects, both national and international. Sometimes the cooperation involved also our families - for example, planning sessions for the LPC, and later REX, meetings that took place at our homes and often ended with dinners, sometimes sprinkled with a little bit of magic.

The *semantics* of your life will be quite different now, but there are many advantages in retirement. You will enter a life with fewer deadlines and obligations. On the other hand, you can pursue lines of interest that you really like. I wish you much success in doing so.

Together with Maja we wish you many years of enjoying the new phase of your life, and we hope that we will still meet from time to time.

Grzegorz

Bilthoven, May 2002

Niels Bohrweg 1
2333 CA Leiden
The Netherlands
Telephone +31-71-5277061
Fax +31-71-5276985

Leiden University

Bij het Afscheid van Jaco de Bakker

Beste Jaco,

Nu je afscheid neemt van het CWI, wil ik in dit stukje stilstaan bij de rol die je in mijn leven hebt gespeeld. Daarmee wordt dit dus geen algemene beschouwing over jouw wetenschappelijke carrière, over het feit dat je veel belangrijke artikelen hebt geschreven, en vele promovendi en hoogleraren informatica hebt opgeleid. Neen, het wordt een betrekkelijk persoonlijk stukje, voor zover dat past in een semi-publiek medium als een Liber Amicorum, over ons. Dat is hopelijk voor jou interessant, voor mij is het dat zeker, en voor derden kan het mogelijk dienen als een illustratie van jouw persoon en werkwijze.

Op voorspraak van Willem-Paul de Roever, bij wie ik als student wiskunde in Nijmegen een college 'semantiek van programmeertalen' volgde, vond onze eerste kennismaking plaats in de vorm van een sollicitatiegesprek, ergens aan het eind van 1984 of het begin van 1985. Gelukkig maakte je daarbij op mij een onvergetelijke indruk, zodat ik deze dan ook hier kan navertellen: Je was, en bent, een enorme persoonlijkheid, gezag en autoriteit uitstralend, over de wetenschap maar eigenlijk ook al het andere. De basis daarvan wordt ongetwijfeld gevormd door het feit dat je veel weet, en beschikt over een grote wetenschappelijke expertise. Maar er zijn ook andere factoren in het spel: kalme houding en postuur, koele en analytische, afwachtende en wegende blik. Met daarbij een heldere en zorgvuldige manier van formuleren, zonder dikdoenerij of een teveel aan woorden, en meteen terzake.

Eenmaal begonnen als promovendus op het CWI, werd ik op een no-nonsense manier begeleid bij het begin van mijn wetenschappelijke werk. Ik kreeg een aantal artikelen te lezen, alsmede het boek waarmee je in die tijd school hebt gemaakt: 'Mathematical theory of program correctness'. Maar aangezien ik was ingehuurd om een mathematische semantiek voor de door Philips ontwikkelde programmeertaal POOL te construeren, en programmacorrectheid was uitbesteed aan een ander, te weten Frank de Boer, vertelde je er meteen bij dat ik alleen het eerste deel van het boek hoefde te bestuderen, over semantiek, en niet het tweede gedeelte, dat over bewijstheorie handelde. Je hebt aan deze inperking van mijn studie-opdracht later nog veelvuldig gerefereerd, veelal met woorden als ' ... maar dat is iets wat je waarschijnlijk nooit echt hebt bestudeerd, dat staat in het gedeelte van mijn boek dat je nooit hebt gelezen'. Lui, gezagsgetrouw, en al snel geabsorbeerd door allerhande semantische kwesties als ik was, heb ik dat tweede deel inderdaad nooit echt bestudeerd.

Gedurende de volgende fase van mijn promotie-onderzoek, zijn we, samen met Joost Kok, veelvuldig naar het Philips Natuurkundig Laboratorium in Eindhoven getogen, om daar met Pierre America (de bedenker van POOL) en Frank van der Linden, te werken aan semantische modellen voor POOL. Aanvankelijk werden we altijd door een Philips-taxi van het station opgehaald, en lunchten we in een soort van directiezaal, waarbij we aan tafel een warme maaltijd kregen geserveerd door vriendelijke Eindhovense dames. Later werd dat minder, maar dat kan niet aan onze wetenschappelijke prestaties hebben gelegen: Naarmate we langer aan de semantiek van POOL werkten, werden de modellen mooier en beter. We maakten daarbij gebruik van het formalisme van de metrische semantiek, dat in de jaren daarvoor door Jeff Zucker en jou ontwikkeld was. Als een interessant bij-produkt van deze semantische exercities ontstond naast de modellen ook nog een elegante, metrische domein-theorie, waarin de methode van Jeff en jou voor het oplossen van recursieve domeinvergelijkingen wat systematischer en algemener werd uitgewerkt.

Jouw werk over metrische semantiek, waarvoor je, enigzins ten onrechte, in de wetenschappelijke wereld al met al minder waardering en erkenning hebt gekregen dan voor je eerdere werk over

programma-correctheid, heeft zowel in het begin als later een voor mijn eigen onderzoek fundamentele rol gespeeld. Zo hebben Marcello Bonsangue, Franck van Breugel, en ik verder gewerkt aan generalisaties van metrische domeintheorie (van quasi-metrische ruimten). Daarnaast onstond, zo ongeveer ten tijde van jouw vijfentwintig-jarig jubileum op het CWI, het inzicht dat een van de door Jeff en jou gedefinieerde metrische domeinen, precies bisimulatie-equivalentie karakteriseert. Samen met het toen net verschenen werk van Peter Aczel over niet-welgefundeerde verzamelingen, vormde dat de basis en motivatie om de notie van coalgebra te gaan bestuderen. Zoals je weet is dat vervolgens een beetje uit de hand gelopen, en ben ik er, gelukkig, nog steeds niet klaar mee. Hoewel de basiselementen van de theorie van coalgebra reeds allen aanwezig waren in het voornoemde werk van Peter Aczel, ben ik ervan overtuigd dat zonder jouw werk aan metrische domeinvergelijkingen en het gebruik daarvan bij de semantiek van programmeertalen, het nog heel lang geduurd zou hebben voordat coalgebra tot een onderzoeksonderwerp op zich zou zijn geworden.

Dat in het kort over jouw invloed op de inhoud van mijn eigen wetenschappelijke werk. Ook over de vorm ervan wil ik nog wat zeggen. Onze bovenvermelde bezoeken aan Philips werden op het CWI altijd grondig voorbereid, in de vorm van lange en intensieve besprekingen, die jij met Joost en mij hield. De manier waarop je mij tijdens mijn promotie-onderzoek, en nog lange tijd daarna, hebt begeleid, kenmerkte zich, meer in het algemeen, door intensiteit en zorgvuldigheid. Je stelde daarbij altijd bijzonder hoge eisen aan de kwaliteit van het geschreven woord, en was een meester in het hanteren van 'Occam's Razor', immer gebrand op het elimineren van overbodige noties, en altijd op zoek naar de meest heldere formuleringen. Ook daarmee heb je een wezenlijke en bepalende bijdrage geleverd aan mijn wetenschappelijke vorming.

Het heeft, vreemd genoeg, betrekkelijk lang geduurd, voordat ik me ten volle bewust ben geworden van jouw invloed op mijn eigen onderzoek. Dat heeft waarschijnlijk te maken met het feit dat ik wiskunde heb gestudeerd. Daardoor was ik, in het algemeen, niet al te snel onder de indruk van het wetenschappelijke gehalte van de theoretische informatica. Mijn leermeesters, dat waren toch vooral de wiskundigen in Nijmegen, die mij de wonderen der wiskunde hadden getoond. Pas later in de tijd ben ik ook jou als leermeester gaan beschouwen. Voor zover ik de kunst van het wetenschappelijk onderzoek beheers, heb ik die in belangrijke mate van jou geleerd.

Ook in je rol van bestuurder en organisator heb je mij veelvuldig tot voorbeeld gediend. Je bent in die hoedanigheden uitzonderlijk succesvol geweest, zowel binnen het CWI (en daarvoor het Mathematisch Centrum) als daarbuiten. Ik geloof dat, naast de al eerder genoemde kwaliteiten waarover je beschikt, ook de volgende eigenschappen in hoge mate aan dat succes hebben bijgedragen: je vermogen en, minstens zo belangrijk, je bereidheid om je te verplaatsen in de zienswijze van anderen. In de loop van de jaren ben je me langzaam maar zeker gaan betrekken bij de aanvraag en organisatie van vele europese en nederlandse onderzoeksprojecten, die ik hier maar niet allemaal ga opnoemen (bovendien weet ik zeker dat jij ze sneller op een rij weet te zetten dan ik). Ook ben je gaandeweg meer beleidsmatige zaken met me gaan bespreken, eerst als groeps- en later als clusterleider. En ook daarbij heb je me in de gelegenheid gesteld, de kunst een beetje af te kijken.

Naast al deze voor mijn werk als wetenschapper en werkleider belangrijke zaken, heb ik nog vele andere dingen van je geleerd. Een van de belangrijkste daarvan is het hebben van respect voor de persoon en zienswijze van de ander, en, aansluitend bij wat ik boven al noemde, het vermogen en de wil je in de ander te verplaatsen. (Talloze keren heb ik het je horen opnemen voor de afwezige, die tijdens een gesprek door een van de aanwezigen werd bekritiseerd.) Bij dat alles hoort een grote voorzichtigheid, die je er in de loop der jaren steeds vaker van heeft weerhouden om met de vuist op tafel te slaan. En natuurlijk heb ik dat laatste ook wel eens gemist, bij voorbeeld ten tijde van de fameuze 'kanteling', halverwege de jaren negentig. Maar tegelijkertijd ben ik je grote behoedzaamheid, die gebaseerd is op respect en, hier nog niet eerder genoemd, oprechte bescheidenheid, steeds meer gaan waarderen.

Toen we het lang geleden eens over mijn plannen voor de toekomst hadden, heb je opgemerkt dat het voor mijn carrière het beste zou zijn, als ik me danwel geografisch danwel thematisch van je zou verwijderen. Het was zeker geen suggestie, eerder een nuchtere observatie. Ik prijs me zeer gelukkig dat ik geen van beide opties ooit bewust heb nagestreefd. (Je hebt er overigens voor

gezorgd, dat ik desondanks over het verloop van mijn carrière geen reden tot klagen heb gehad: Op jouw initiatief ben ik in 1997 themaleider geworden, en op voorspraak van Jan Willem Klop en jouzelf ben ik onlangs benoemd als bijzonder hoogleraar aan de VU.) Zo zijn we al die jaren, ruim zeventien in getal, zeer naaste en elkaar vertrouwde collega's geweest, eerst in de groep AP1 en later in het thema SEN3. De laatste jaren spraken we elkaar veel, regelmatig en vaak, over wetenschap, steeds meer ook over beleid, en over al het andere.

Voor dit alles, Jaco, wil ik je heel hartelijk danken. Ik zal je missen. Gelukkig blijf je nog enige tijd aan het CWI verbonden als adviseur, zodat we elkaar als collega's af en toe zullen blijven ontmoeten. En laten we, daarnaast, zo nu en dan nog eens een stukje gaan varen.

Amsterdam, juni 2002

Jan Rutten

De dingen die voorbijgaan

Nico Temme

Beste Jaco,

de dagen, weken, maanden, jaren gaan sneller voorbij dan tijdens de jonge jaren, en we hollen als het ware naar het einde van deze betrekking. Blijft er genoeg tijd over om na te denken over wat er komen gaat?

Als je ergens nieuw in functie komt, of student wordt, etc., zijn er medewerkers of ouderejaars die je het klappen van de zweep leren kennen. Als je aan het einde van de rit bent gekomen zijn er, vooral bij het CWI, nauwelijks mensen die je kunnen voorbereiden op de nieuwe periode. Aangezien ik zelf zo langzamerhand de oudste medewerker dreig te worden krijg ik het gevoel dat uitwisseling van ervaringen voor de nieuwe periode totaal onmogelijk lijkt. Ieder voor zich, bij uittreden.

Hoewel ik me niet kan voorstellen dat ik me zal vervelen na vertrek van het CWI blijft toch altijd bij mij op de achtergrond de vraag spelen: "Wat doet men eigenlijk na vertrek van het CWI?" Nieuwe hobbies starten of oude verder cultiveren lijkt me het beste.

Ik weet van je eigen privéactiviteiten te weinig af om hier verder in te adviseren, vandaar dat ik meer vanuit mezelf ga redeneren om je wat ideeën voor die tijdsbesteding te geven. Ik zou aardig wat tijd gaan besteden aan het lezen van boeken, literatuur vooral, maar ook wat zich verder aandient.

Het boek *De Puinhopen van zoveel jaar CWI* is nog niet verschenen, en ik denk dat we daar ook lang op zullen moeten wachten. Het NIOD rapport is van een geweldige omvang, je zou er een tijdje zoet mee zijn, maar ik heb het zelf niet gelezen en kan het je dan ook niet echt aanraden.

Enkele recente ervaringen, die ik wel kan aanbevelen, zijn de volgende boeken.

Portret van een jonge man (Engelse titel: *Youth*) van J.M. Coetzee, een Zuidafrikaanse schrijver die zijn jonge jaren in Londen beschrijft; zijn vurigste wens is dichter te worden, maar hij moet in zijn onderhoud voorzien als programmeur bij IBM; hij heeft wiskunde gestudeerd.

Ingenieurs van de ziel van Frank Westerman, over de rol van schrijvers, en andere intellectuelen, tijdens de Stalinjaren in de USSR. De rol van Gorki hierbij, en in een andere zin, die van Paustovskij, geven interessante en aangename leeservaringen. Stalin heeft de sovjetschrijvers destijds de "ingenieurs van de ziel" genoemd en ze voor het bewind willen gebruiken bij het opheffen van de megalomanische waterbouwwerken. Bijvoorbeeld, Moskou moest een zeehaven krijgen.

Het Kaartenhuis van Mark Z. Danielewski (in het Engels *The House of Leaves*). Een "spannend" boek; ruim 700 pagina's. Ik heb de Nederlandse vertaling maar gelezen omdat ik wist dat het een zeer complex boek is. Je waant je vaak in het complexe vlak, met imaginaire grootheden (een huis dat meer inwendige ruimte bezit via geheime deuren en gangen dan in werkelijkheid mogelijk is). Ontregelend veel voetnoten en voetnoten binnen voetnoten (die nieuwe verhaallijnen introduceren); tekst die in andere fonts gezet wordt, op z'n kop of in spiegelschrift, etc. Het houdt je een tijdje bezig, zal ik maar zeggen, en het is een fikse uitdaging. Voor web info: http://www.houseofleaves.com/ waar ook discussie gaande is.

En tussen de bedrijven door naar de film: *A beautiful mind.* John Nash was natuurlijk zo gek nog niet, en de makers van de film ook niet.

Beste Jaco, onze contacten op het CWI waren voornamelijk in verband met managementzaken en ik heb deze contacten altijd erg plezierig gevonden. Je hebt het CWI uitstekende diensten bewezen. Het ga' je goed, en ik wens je veel ontspannende momenten toe in deze nieuwe fase van je leven.

Nico.

Origins of our Theory of Computation on Abstract Data Types at the Mathematical Centre, Amsterdam, 1979-80

J V Tucker
Department of Computer Science, University of Wales Swansea,
Singleton Park, Swansea, Wales, SA2 8PP

J I Zucker
Department of Computing and Software, McMaster University,
Hamilton, Ontario, Canada, L8S 4L7

To Jaco de Bakker
With gratitude, admiration and affection

1. Introduction

In 1979 the authors (hereafter JVT and JIZ) began our work together on developing a theory of computation that works for *any* data. We were members of Jaco de Bakker's research group at the Mathematical Centre. There we learnt about the semantics of programs and specifications, and about proof systems for program verification. There we found the mathematical form of our theory and its first application in computing. We will recall those times, describe the beginnings of our research programme, record some of Jaco's influence, and mention some present developments.

Our theory is about computation over *arbitrary* data types. It combines the algebraic theory of data with different programming language constructs and their semantics. Data is modelled using many sorted universal algebras and specified using equations or conditional equations. Programming constructs of many kinds control and extend the operations and tests of the algebra to compute sets and functions on the data. The theory is a generalisation of the now classical theory of computable functions on natural numbers or strings.

In 1979 the theories of programming language semantics and data types were both young. From the beginning, we have aimed to create a comprehensive mathematical theory for computing functions and sets, and to develop plenty of applications. In fact we have aimed to match, and in some areas supersede, classical computation theory based on natural numbers or strings. How much we have succeeded is for another occasion to judge. Something of a progress report was contained in *J W de Bakker, 25 Jaar Semantiek: Liber Amicorum* in 1989. We include our current bibliography for interest. We are still working on both theoretical and applied problems and there are many

problems in our research plans awaiting their turn. For the past few years we have been particularly focussed on computational problems on discrete and continuous data types and the analysis of that interface.

At the heart of our work is our theory of computable functions and semicomputable sets on many sorted algebras. Our theory is *abstract* in the sense that computations are independent of the representations of data and so are isomorphism invariants. It is *general* in the sense that it covers all data types - since all data can be modelled by many sorted algebras. Thus, from our theory of computable functions on algebras we can derive computability theories on particular classical data types, like number systems (e.g., natural numbers, real numbers, and complex numbers) and syntax (e.g., strings, terms). We can also derive computability theories for mathematical constructions (e.g., spaces of continuous functions) and general axiomatic classes of data types (e.g., semigroups, groups, rings, fields, vector spaces, Banach and Hilbert spaces). We can add storage structures (e.g., arrays, stacks and queues) and notions like time (e.g., timed data streams and waveforms) and location (e.g., addresses and co-ordinate systems). Over the years we have considered all of these examples of many sorted algebras.

Recently, we published a 200 page survey (Tucker and Zucker [2000]) of our theory, based on one of the simplest models of computation that we have studied in depth, namely the **while** programming language and its extensions. This survey contains a detailed history of the subject and account of the principal theorems. It also has a taste of our current research problems, namely to use the general theory to create a theory of computation on topological data and apply it in modelling, specification and programming.

Our first publication was *Program correctness over abstract data types with error-state semantics* (Tucker and Zucker [1988]). It was a research monograph based on Jaco's book *The Mathematical Theory of Program Correctness* (De Bakker [1980]). It generalised and extended Jaco's work on semantics and proof systems for program correctness. Its last chapter was on computable functions on many sorted algebras.

2. At The Mathematical Centre 1978-79

2.1 Background

In 1978, the Mathematical Centre, or MC, was at the Tweede Boerhaavestraat, next to the old Amstel Brewery. Its Director was Aad van Wijngaarden. A lot of excellent people in Computer Science had worked there almost from its inception in 1946. The Director had joined the MC in 1947 and begun work on the construction of the first Dutch computers (the ARRA, ARMAC and X1). Among his 15 doctoral students and former colleagues at the MC were most of the Dutch professoriate in the period. In 1978 three of his students were at the MC: Jaco, Hans van Vliet and Dick Grune. Peter van Emde Boas had recently moved to Amsterdam University but was frequently to be met at the MC. In 1978, the MC had a mature and strong computer science research culture.

Jaco joined the MC in 1964 and was awarded his doctorate under van Wijngaarden in 1967. Jaco was a theoretical computer scientist with a fine international reputation as a pioneer in the field of programming language semantics. He was also experienced in organising and encouraging this new scientific field through the foundation of the *European Association for Theoretical Computer Science* (EATCS), journals like Elsevier's *Theoretical Computer Science*, and international summer schools at the MC. He was Head of the Computer Science Department, which was small but first-rate.

While doing research, Jaco was also engaged in writing his graduate textbook and research monograph *The Mathematical Theory of Program Correctness*. This book was to give a self-contained account of the semantics of imperative programming language constructs. These constructs were introduced one by one, in small languages designed to focus on the computational ideas behind the construct. The operational and denotational styles of semantics were introduced and proved equivalent. Furthermore, for the constructs and their languages, a Floyd-Hoare logic was given to prove partial correctness assertions. Each logic was proved sound and complete with respect to its semantics. All computations, specifications and proof systems involved natural numbers and Booleans *only*.

At the time defining the meaning of programming constructs and languages was in some circles an important methodological problem. Program verification had to seen against the backcloth of axiomatic semantics. This was the idea that one could specify the meaning of a programming language by postulating axioms and rules for a logic to prove the correctness of programs. This was a high-level approach to defining the meaning of a programming language and one centred on the user of the language.

Euclid had axiomatised geometry circa 300 BC, and axiomatic foundations for the rest of mathematics had been created in the 19th and early 20th centuries. The conception was simple enough, but the implementation was exciting and hugely ambitious. The possibility of an axiomatic semantics for a programming language first appears in Floyd [1967] and is beautifully explored in Hoare [1968]. Axiomatic approaches to other programming issues had appeared: for example, van Wijngaarden [1966] for computer arithmetic, and De Bakker [1968]. The whole enterprise would now be seen as part of the search for verification methods for software (Jones [1994]).

In the 1970s proof rules were being invented for constructs that had no independent semantics. Faulty rules were not uncommon. The problem of correctness was recognised as a fundamental and pervasive problem and was being attacked by formal methods, but the first formal methods lacked theoretical foundations. Jaco's book was part of the research effort into providing these foundations.

Assisting Jaco at the MC was Arie de Bruin, who was working on his PhD thesis under Jaco's supervision (De Bruin [1986]). Former colleagues in semantics at the MC had moved on but were in regular contact, including Willem Paul de Roever (at Utrecht) and Krzysztof Apt (at Rotterdam).

Among the cognoscenti in 1979, research in concurrency was recognised as a large, exciting and *open* area. Willem Paul had visited Tony Hoare in Belfast and become interested in CSP. The paper Owicki and Gries [1976] was taken up in new work by Willem Paul and Krzysztof. Their seminars and pre-prints were attracting a great deal of attention locally and internationally (Apt, Francez, and De Roever [1980]). Much later they were to write their own books Apt and Olderog [1991], Francez [1992], De Roever et al [2001]

Although Jaco's book was completed at the time concurrency research was taking off semantically, he did not attempt to include parallelism. In Chapter 7 (page 258), on *Nondeterminsitic Statements*, Jaco gives as the first of three reasons for the study of nondeterministic choice

$$S_1 \cup S_2$$

its role in the arbitrarily interleaving semantics of parallel execution. However, he noted that "... the present status of the research on the semantics and proof theory of parallelism is, in our opinion, not yet such that it justifies a treatment on the textbook level..."

Indeed, it all looked awfully difficult to JIZ and JVT at the time.

2.2 Our Collaboration

JIZ joined Jaco's group in September 1978, moving from Dirk van Dalen's logic group at Utrecht. JIZ was an experienced logician and proof theorist, and was settled in the Netherlands having held appointments in Amsterdam and Eindhoven. JIZ's colleagues during his three years with Dirk's group in Utrecht (1975-1978) had included two of Dirk's doctoral students, Jan Willem Klop and Jan Bergstra, who would later have a profound impact on Dutch computer science.

JVT joined in January 1979 from Oslo where he had been a member of the group of Jens Erik Fenstad on generalised recursion theory. JVT's colleagues during his two years with Jens Erik included Dag Normann, Viggo Stoltenberg-Hansen and Johan Moldestad. He had written to Jaco describing his research and inquiring if there were any visiting posts now or in the future. It turned out that Krzysztof was leaving the MC for Rotterdam and there was a vacancy. Jaco invited JVT to come for a year. They had not yet met.

JIZ and JVT shared the end office in the little "temporary" building in the courtyard of the MC. Accommodation at the Tweede Boerhaavestraat was tight. From the ground floor of the main building, you passed into the courtyard, beside the windows of the punch card room, and into a single storey prefab with offices on one side of a corridor only, stocked with computer scientists. Our office was at the far end. There were two grey metal desks in the centre of the room, at which we sat facing one another. This was ideal for conversation. JIZ sat with his back to the window, his papers and notes surrounded by 3B Staedtler pencils and erasers.

The collaboration got off to an excellent start. JVT was impressed by JIZ's wide intellectual interests, especially his knowledge of logic, philosophy and linguistics, partly represented by the excellent collection of books which filled the grey metal bookcases in our office. JIZ was impressed, then and later, by JVT's facility for social interaction, based on a natural curiosity about other people, and an ability to draw them out about their own ideas, and discern connections with his work - a talent which led to a number of fruitful collaborations with different researchers.

JIZ and JVT had a lot in common. We were devoted to logic, loved literature, and delighted in conversation. JIZ was deeply committed to music and JVT enthralled by fine art. We were both at the MC to learn, too. Our logical expertise had given us a close connection with theoretical computer science. From different directions – proof theory and computability theory - we were now simply moving into the subject. Jaco's department was the first computer science department either of us had worked in. We were by no means the first or the last logicians to enter the subject through joining Jaco's department!

Our immediate next door neighbours were Hans van Vliet and Dick Grune who were involved in the later stages of the Algol 68 project. They kept a coffee percolator in their room. They were frequently interrupted when in residence by serious coffee drinkers like us, and by the noise of our animated conversations. Hans and Dick both worked for long periods on the computers in a terminal room in the main building. Their neighbours were Paul Klint and Lambert Meertens, also often in the terminal room. JIZ was, probably, the only inhabitant of the courtyard who did not smoke at that time.

Other theoreticians at the MC were Paul Vitanyi and Peter J Asveld, who were working on complexity theory and formal languages in the main building. High up in the main building was Jaco's office, where all good bosses should be.

Our working days had a pattern. JIZ arrived early and JVT arrived late, though always with time to do something before the morning visit of Mien, the tea lady, which signalled the start of our daily conversation. JVT spent at least an hour exploring the library every day. For the pleasure of browsing, no library compared with the MC's. Each day we went out to lunch in a small sandwich and coffee shop in the Weesperplein, often with Paul and Peter. The talk was largely of theoretical computer science. The afternoon was very much for conversation.

In January 1979 we started to discuss two subjects of interest to us, a little each day. The topics seemed quite independent.

1. *Logics for program verification* and the semantics of imperative constructs, based on Jaco's approach, and the soundness and completeness of Floyd-Hoare logics, based on Steve Cook's approach.

2. The *generalisation of computability theory* from natural numbers to arbitrary algebraic and relational structures based on various models, notably the finite algorithmic procedures of Harvey Friedman, which was a generalised register machine approach.

It took until Spring 1979 to see the natural connections between these subjects and ask some obvious questions.

JIZ was pursuing various research topics in logic, including work on phenomenology and logic with Robert Tragesser. He was also working closely with Jaco on the later stages of his book. Through reading the book closely, he became interested in the expressiveness of the assertion language in the proofs of completeness. He was working out a self-contained proof of the expressiveness of first order logic for pre- and post-conditions for computation by recursive procedures on the natural numbers, to be incorporated in the book as an appendix. This was quite an exercise and gave him the opportunity to dig into the technicalities of the relevant semantical and logical issues. It demanded more frequent wanderings up and down the corridor than many of the problems we have worked on since.

JVT was already working on generalising computability theory. At Oslo he had written some papers on computability theory on algebraic structures. Theorems in Moldestad, Stoltenberg-Hansen and Tucker [1980a,1980b] bridged the gap between abstract register machine computations (a la Harvey Friedman and John C Shepherdson) and (i) axiomatic computation theories and (ii) fixed point methods in higher type recursion. The results had been included in the monograph Fenstad [1980]. Recently, in Fenstad [2002], Jens Erik has reflected on that period of generalised recursion theory, both historically, and in the light of some of our more recent results.

JVT had been introduced to Harvey Friedman's paper in his first year of graduate studies at Bristol, in a course given by John Shepherdson in January 1974. However, his PhD had been on computable universal algebras, groups and fields under John P Cleave. Computable algebras of various kinds had been studied by M O Rabin and A I Mal'cev. JVT had started to work on John Cleave's ideas for a method of analysing the computability of uncountable topological algebras (specifically the classical matrix groups) using generalised enumerations from Baire space, but had found the Mal'cev theory for countable algebras under-developed. Cleave did not publish his approach and, quite independently, Klaus Weihrauch later created an extraordinary theory of computation for topological spaces using enumerations from Baire and Cantor space starting in Kreitz and Weihrauch [1985] and recently described in Weihrauch [2000].

In the first half of 1979 he was finishing a paper with Viggo Stoltenberg-Hansen on the undecidability of the roots of unity problem in computable fields. He was also actively learning about programming language theory and semantics. In the lunchtime meetings with Paul and Peter the subject of complexity theory often featured. JVT and Peter started to work together on complexity results for abstract models of computation on universal algebras (Asveld and Tucker [1982]).

In February 1979, JVT began regular meetings with Jan Bergstra. They had become friends through Jan's visits to Oslo, when he was active in higher type recursion theory. Discussions were focussed on algebraic specification methods for data types and logical foundations for program specification and correctness. As their research developed, the pattern was established of JVT visiting Leiden or Utrecht, usually on a Tuesday, to discuss a variety of subjects, and, especially, the week's progress on their investigations.

Algebraic methods for the specification of data types had been developed in the programming methodology literature by Barbara Liskov and Stephen Zilles, and by John Guttag. Extensive case studies showed that there were many ways to use many sorted equations, conditional equations, and other formulae to axiomatise the operators of data types and this raised many questions as to which were best. Not surprisingly, some methods were expressed through examples only. The algebraic specification techniques were first analysed mathematically by the ADJ Group, as part of their general research programme on initial algebra semantics (Goguen, Thatcher, Wagner and Wright [1977]). The study of the scope and limits of the methods led to a deep collaboration between JVT and Jan that uncovered intimate connections between computability, specification and verification. The first of many MC Reports they devoted to these subjects appeared in 1979 (see Bergstra and Tucker [1979 a, b, c]. The theory of data types had taken root in The Netherlands.

The algebraic approach to data types was also discussed at length in the office at the MC. It gave a new perspective to the search for a general theory of computable functions on algebras and raised the question what is the theory of specification and correctness on data types other than the natural numbers. JIZ began to actively study abstract computability theory. He wondered if the whole of Jaco's approach could be generalised to arbitrary structures. He had met many sorted structures through the interest of his supervisor, Sol Feferman, at Stanford. The many sortedness was exactly right for modelling data. The computability theory showed that the assertion languages could no longer be first order because they could not express even simple computational conditions, but that a weak second order assertion language (allowing finite sequences or arrays) would be adequate. JIZ suggested we write an MC Tract to see how the theory looked. He also suggested we might look at errors. We agreed and proposed the idea to Jaco who was enthusiastic.

We planned out our monograph keeping closely to the pattern of the first five chapters of Jaco's book. JIZ would write the three correctness chapters and JVT would write the introduction to the project and a final chapter on computability on many sorted algebras. It was an extension of Jaco's approach, designed to allow applications to any data.

At the end of September 1979, JIZ left The Netherlands for Bar Ilan University in Israel. Our plans were clear and it was agreed that he would make return visits to Amsterdam to work on the end of one book (Jaco's) and the start of another (our own). JVT was left working on a paper on generalised computability theory and its applications to abstract algebra. It was the published version of his invited lecture at the Association for

Symbolic Logic Summer Meeting at Leeds in 1979. This was the last of the register machine work that he had started in Oslo (Tucker [1980]).

Ralph Johan Back (Helsinki) took JIZ's desk and new conversations started up, this time on programming methodology, refinement and logic. JVT learnt a great deal of contemporary thinking on software construction from Ralph. Whist Ralph introduced him to several research problems JVT was too slow a pupil to write to a joint paper.

The following year, 1980, saw the move of the MC from the Tweede Boerhaavestraat to the Kruislaan. At lunch this was longed for by Ralph (used to the quality and modernity of Scandinavian interiors) and bemoaned by Paul and JVT (used to the MC). One lunchtime we upped and went to see the new building before its completion to prove our respective points. However the debate descended into speculations on how to get there, where the sandwich shops were, and where our offices might be. When the MC moved we still managed to get there each day, there was a canteen, and Ralph and JVT were in the same office.

That summer brought the seventh *International Colloquium on Automata, Languages and Programming* (ICALP) to Noordwijkerhout, organised by Jaco and others. With it came visitors to Amsterdam, including Jim Thatcher who had been working out the mathematical theory of data types with Joe Goguen, Eric Wagner and Jesse B Wright in one fine paper after another for several years. On September 1, came the retirement of Aad van Wijngaarden, and changes in the organisation of the MC. Ralph returned to Helsinki but the conversations continued, JVT visiting him in Helsinki in 1981. Subsequently, Ralph has influenced the development of formal methods for software through his deep study of refinement, starting in his thesis and MC Tract, Back [1980], and most recently described in his book, Back and von Wright [1998]. He has also had a strong effect on Finnish computer science, of course.

How was our *Program correctness over abstract data types with error-state semantics* written? Slowly. We received a lot of patient encouragement and support from Jaco over the nine years it took to publish the book in 1988. We were occupied with other research problems: for example, JIZ and Jaco were working on concurrent processes (see below), and JVT and Jan Bergstra continued their work on data type specifications and Hoare logic. There were disruptions. In January 1981, JVT returned for a brief period to Bristol University in the UK. In October 1981 he settled at Leeds University. After three years, JIZ moved from Bar Ilan to SUNY at Buffalo, New York.

However, technology, too, moved on. The IBM typewriters that had produced Jaco's book were passing into history. Our book was produced by JIZ using the Ditroff text processor under UNIX with a QMS Lasergrafix 1200, the first laser printer to be used at Buffalo's computer science department, funded by his grant from National Science Foundation. From that time to this day, we write `tt` and `ff` for Booleans, in fond memory of Jaco's typewriter notation for $\mathtt{t}$ and $\mathtt{f}$ in his book (using 3B Staedtler pencils, of course).

3. After Jaco's Book

When Jaco's book was finished, there was an obvious interesting question for everyone in his circle. What was Jaco going to do next? There was a definite sense of achievement in the completion and appearance of the book in 1980. In it the basic technical issues of semantic modelling, and its connection with logics for program verification, were answered clearly, rigorously and in detail. For imperative programming, the book sorted out technical issues that had been causing problems since the emergence of semantic modelling and formal verification a decade earlier. This sense of achievement can also be found in Krzysztof's survey paper *Ten years of Hoare Logic* (Apt [1981]). The basic approach to semantics went something like this:

- design constructs that require semantic models
- design syntax-directed logics for the constructs
- prove soundness and completeness.

If you wanted to do this properly then you had know Jaco's book.

In terms of research, as the book came to completion, it seemed to us there were some obvious avenues for further theoretical research on semantics and verification:

1. The mathematical investigation of Hoare logic, since little was known about its proof theory and model theory. This was taken up in a series of papers in the 1980s at the hands of a few authors including Ed Clarke, Ernst-Rüdiger Olderog, Jan Bergstra and JVT.

2. The investigation of proof systems for communication and concurrency. This had started in earnest was being taken up by many researchers.

3. The extension of Hoare logic with abstract data types, modules, classes and other encapsulation and architectural ideas, and with errors and exceptions. This was taken up by several authors including us.

Jaco was primarily interested in denotational semantics, and he abandoned Hoare logic. He had followed the research on concurrency and was intrigued by the abstract process approach of Robin Milner, newly expounded in Milner [1980]. The process analysis of concurrency was exciting. Uninterpreted actions could be manipulated by operations and defined by fixed point formulae, but the method of making a calculus such as CCS appealed less. Jaco had also been taken with the work of Maurice Nivat on infinite trees and strings (Nivat [1979]). It contained a beautiful metric space treatment of the idea of specifying a set of strings or trees, and was related to Nivat and Arnold's metric space theory of nondeterministic programs. Jaco had already considered trees in De Bakker [1977]. Trees and metric spaces were central to Ruurd Kuiper's first work on semantics at the MC, written under Jaco's guidance (Kuiper [1981]).

The idea that a process is "like" an infinite tree was a starting point for applying metric techniques to the specification and semantics of concurrent systems. This seemed a new

and promising project: it combined simple abstract notions of process with established mathematical tools. Surely it would lead to a denotational semantics for concurrency and new insights into concurrent processes? It certainly did.

For many years after JIZ had left Amsterdam he visited the Netherlands every summer to work with Jaco at the CWI. Sometimes JIZ and JVT would meet at the CWI; sometimes he would travel on to Leeds. In addition, Jaco and JVT were guests of JIZ at Bar Ilan University and of the Weizmann Institute in Israel in the summer of 1981, following the eighth ICALP meeting at Akko that year. It was a relaxed, pleasant and productive visit. We continued working on our book, and Jaco and JIZ developed the ideas leading to the paper De Bakker and Zucker [1982], which helped to lay the foundation for the broad subject of concurrent process theory.

On subsequent visits by JIZ to the MC and CWI, this subject of topological process theory was developed much further by Jaco, JIZ and other colleagues, including Joost Kok, John-Jules Meyer, Ernst-Rüdiger Olderog, and Jan Rutten, resulting in the papers De Bakker and Zucker [1983 a,b], De Bakker, Meyer and Zucker [1983], De Bakker, Kok, Meyer, Olderog and Zucker [1985], De Bakker, Meyer, Olderog and Zucker [1988], and Rutten and Zucker [1992].

The origins of this subject is discussed van Breugel [1999] and its effects recorded in the books De Bakker and Rutten [1992] and De Bakker and De Vink [1996].

4. Concurrency, Domain Representability and Computability on Topological Data Types

The metric space theory of processes was influential in the development of a popular concrete approach to the study of computability on topological algebras, that of using domains to represent the algebras.

Jan Bergstra and Jan Willem Klop started working on process algebra after a lecture by Jaco in Utrecht in June 1982. They tackled the open problem he posed of solving unguarded recursion equations in the topological model of De Bakker and Zucker [1982]. Their solution was this: in the case of a finite set of atomic actions, they created the axiomatic system *Process Algebra* PA for processes. The theory PA had an initial algebra A_ω and a system of projections A_n that modelled the execution of processes for n steps, for $n = 1, 2, \ldots$. These projections were also models of PA and the algebras formed an inverse sequence with inverse or projective limit A_∞, which was again a model of PA. They proved that all recursion equations have solutions in all the A_n and so in the A_∞. Since the A_∞ can be embedded in the De Bakker-Zucker model of processes, the problem was solved.

The theory PA was extended to the axiomatic system called *Algebra of Communicating Processes* ACP, and it was soon recognised that these axiomatisations were independent

and important new approaches to the theory of concurrent processes. Early on, in 1983, Jan Bergstra made one of his regular visits to JVT on holiday at his home in Ogmore-by-Sea in Wales, and he explained the theory in some detail. It was immediately exciting: mathematically, the theory combined the abstract conception of processes, the beautiful algebraic axioms of PA and ACP which were algebraic specifications of processes, and applications of mathematical techniques from the huge range available in universal algebra and topology. It was clear there could be many operators and axiom systems. Therefore, it seemed to be important that the subject follow general algebraic principles, i.e., it should use the basics of universal algebra, equational axiomatisations, homomorphisms, etc. This first encounter led to a few joint papers, including Bergstra and Tucker [1985] which attempted to give a clean account of the ideas in a strict algebraic style; added axioms (e.g., standard concurrency); proved a Milner expansion theorem for ACP; and attempted to use homomorphisms to model top-down design. JVT was sold on process algebra in one afternoon, between lunch and high tea, and has followed with great pleasure the development of the subject ever since.

In 1983, JVT was also continuing his longstanding collaboration with Viggo Stoltenberg-Hansen (now in Uppsala) on computable algebra. They wanted to develop a smooth and general treatment of computability in topological algebras, especially rings and fields, about which they knew a lot (Stoltenberg-Hansen and Tucker [1999a]). This problem had been encountered by JVT when working with John Cleave in 1974. Unknown to them, Klaus Weihrauch was also seeking a general method of analysing computations. In fact, Klaus considered a connection between metrics and partially ordered spaces in Weihrauch [1981], but abandoned this direction in favour of his theory of Baire space enumerations.

At their next meeting in Leeds, Viggo explained how he had been attracted to the study of local rings and their completions. There a local ring R with a maximal ideal M is used to create a inverse sequence of rings R/M^n and an inverse limit R_ω. This inverse limit was an uncountable ultrametric space and, in particular, a completion. What can be computed in local rings and their completions? We thought about computing with Cauchy sequences of rationals and with the Baire space of all functions on the natural numbers.

Viggo had also taken the brave step (for a mathematician) of offering a course on domain theory at Uppsala. There was a lot to learn about the connections between order and topology, fixed-point theory, and, of course, effective domains. But, by taking a select path into the subject, it did appear to be a generalisation of the key elements of higher type recursion theory.

On visits like this Viggo and JVT simply talked most of the time. Conversations about local rings and domains were interleaved many times each day and began to converge. They worked out how to build representations of complete local rings using total elements in algebraic domains. This allowed them to apply the theory of effective domains to analyse computability in the ring automatically.

At the same time JVT was keen to introduce a third new subject to their repertoire. He explained the algebraic theory of processes based on ACP with its use of equations, initial models and inverse limit model, which was another ultrametric space and completion. They saw that the methods for representing the completion of the local rings also applied to the inverse limit model of ACP and to many other sorts of completions, including infinite trees.

They wrote out the domain representation method for topological universal algebras, and formulated the general approach of analysing the effective content of topological algebras. They formulated the general problem of finding (computable) solutions of equations in topological algebras, inspired by the idea of giving equational specifications of (computable) processes. How were the domain theoretic and metric space based methods related for topological algebras in general, and for process algebras in particular?

Viggo and JVT embarked on a research programme to represent different sorts of topological algebras using domains. The theoretical starting point was this:

inverse limits of algebras were ultrametric algebras, and conversely,

and our domain representation methods worked beautifully for ultrametric algebras. In fact, the typical situation was exactly like that of the inverse limit model of processes: there was an initial algebra and a family of congruences $\equiv_n$ that led to a countably indexed inverse limit.

Viggo and JVT ultimately wrote up the work on the local rings and the general approach in Stoltenberg-Hansen and Tucker [1985] and circulated it widely as the first preprint of the newly launched *Centre for Theoretical Computer Science* at Leeds University. It was later published as Stoltenberg-Hansen and Tucker [1988]. They also wrote up their investigations in papers on ultrametric universal algebras and the solution of finite and infinite systems of equations using domains and fixed points (Stoltenberg-Hansen and Tucker [1991,1993]). The idea was to show that

(i) there was an equivalence of approaches to concurrency based on process algebras; process calculi; metric space methods and, indeed, domain theory; and
(ii) these equivalences could be derived from the principles of universal algebra.

They gave applications to process algebra and infinite synchronous concurrent algorithms. Domain representability, and results such as the fact that the Banach Contraction Mapping Theorem was derivable from the Least Fixed Point Theorem via a domain representation, were included in Viggo's book on domains (Stoltenberg-Hansen, Lindstrom and Griffor [1994]).

Now, although the real numbers were invaluable in exploring the abstract idea of completion they do not themselves an inverse limit. Viggo and JVT broadened the general domain representation method and showed that the functions on the real numbers that were effective in a natural domain representation of the real numbers were the computable functions of Grzegorczyk and Lacombe defined in the 1950s (see below). This result was published later in Stoltenberg-Hansen and Tucker [1995].

The domain representation method for topological universal algebras is now widely known and used when looking at computation in analysis and its applications (see Edelat [1997]). It is interesting to note the power of the oral tradition in the origins of this domain approach to computable topological data types. The general forms of domain representations for universal algebras were revealed through conversations on concurrency theories between Jaco and JIZ, Jan and Jan Willem, Jan and JVT, and JVT and Viggo.

5. A Taste of our Current Research

Notions of computability on the natural numbers and strings have long been known to agree. Since 1936, many models of computation have been developed and proved to be equivalent. The theory of computable functions on natural numbers is stable. It is largely independent of data representations (e.g., computability on binary numbers is equivalent to that on decimal numbers) and programming constructs. It possesses many elegant and efficient models with which to work on applications. And, thanks to the theory of the arithmetic hierarchy, the connection between computation and specification in first order logic is clear and beautiful. It is truly worthy of its name Classical Computability Theory.

Our research has sought to analyse the concepts and mathematical ideas in classical computability by examining them in the more general setting of an arbitrary many sorted algebra and applying them in particular cases, such as algebras of real numbers. Currently, computing with topological algebras is our main occupation. Here is a taste emphasising the real numbers.

5.1 Computability on topological algebras

There are many approaches to defining computability on topological and metric algebras that are technically different and have different agendas. Until recently, there was no sign of a stable theory, only a rather confusing range things to do and ways to do them. The rather basic question,

What are the computable functions on topological algebras?

did not have a general, widely agreed and understood answer, even for the real numbers; an elegant solution to this fundamental problem seemed even further away.

The computability theories in the literature may be divided usefully into two kinds:

1. *abstract computability theories*, in which computations are independent of data representations; and
2. *concrete computability theories*, in which computations depend on some chosen data representations.

Abstract computation theories are based on "programs" that use the basic operations and tests of the algebra only. They may have different control or specification constructs, such as **goto**, recursion or parallelism. The programming languages that are used in semantical studies, like Jaco's book, are all examples of abstract models of computation. One simply has to ask the question: What does this language compute?

As we showed in Chapter 4 of Tucker and Zucker [1988], many abstract models of computation of different kinds have been defined and shown to be equivalent. Subsequently, we analysed several more and found a family of abstract models better suited to specification rather than computation. Thus, the theory of abstract computation is quite stable. For a recent comprehensive introduction to abstract computation, including a new survey of its origins in the 1950s and principal literature, see our Tucker and Zucker [2000]. There we used **while-array** programs over these algebras, the primary mathematical model of imperative programming. Abstract computation theories are designed to compute on *all* many sorted algebras and so can be used to develop computability theories for particular algebras such as *rings and fields of real numbers* (see, e.g., Blum et al [1998]) and *topological and metric algebras* (Tucker and Zucker [1999a]).

Concrete computation theories are designed to analyse computability in terms of classical recursion theory on natural numbers via chosen representations of data and spaces. Some general approaches are:

Effective metric spaces	Moschovakis [1964];
Computable sequence structures for Banach spaces	Pour El and Richards [1989],
Type 2 enumerations	Weihrauch [2000],
Algebraic domain representations	Stoltenberg-Hansen and Tucker [1988, 1995],
Continuous domain representations	Edalat [1995],
Numbered spaces	Spreen [1998].

The equivalence of most of these concrete approaches is proved for certain topological algebras in Stoltenberg-Hansen and Tucker [1999b]. Whilst the study of concrete computability is not so well understood, it seems to be stable on the real numbers. The general concrete models have all been shown to define the *Grzegorczyk-Lacombe (GL) computable functions* on the real numbers, first formulated in the 1950s and the basis of early studies in computable analysis.

It is the connection between the abstract and concrete theories that has been a problem. Concrete models can compute a lot more or a lot less, depending on the selection of the algebraic operations. Only recently, it has been shown that, surprisingly, notions of

(i) continuity and partiality,
(ii) limit processes and approximation, and
(iii) nondeterminism and multivaluedness,

are necessary to bridge the gap between them for general classes of metric algebras: see Brattka [1996, 1999] and Tucker and Zucker [1999, 200?]). Let us amplify these points in turn.

Ad (i): Abstract computability theories are dependent entirely on the operations and tests of the algebra. Since the choice of operations is unlimited, it is possible to choose operations that are not concretely computable. A typical mistake is to allow tests like

$$= \text{ or } <$$

as total functions in the algebra. These operations are discontinuous. It follows from conceptual analysis and different versions of Ceitin's Theorem that concrete computable functions are always continuous on the real numbers. Hence, such basic tests can only be introduced as *partial operations*.

Ad (ii): Abstract models compute outputs that are contained in the subalgebra generated by the input. However, in a metric algebra they can computably approximate data outside this subalgebra. For example, on the field of real numbers, $\sqrt{x}$ cannot be computable by **while-array** programs because $\sqrt{x}$ is not in the subfield $<2> = Q$, the rationals. It is, however, computably approximable.

Ad (iii): Concrete models can compute single-valued selection functions that are continuous in the topology of the *representations* of the real numbers, but *not* in the topology of the real numbers themselves. Abstract models cannot compute such single-valued functions.

In our Tucker and Zucker [200?a] we extended the language of **while-array** programs with the *non-deterministic assignment statement*

$$x := \textbf{choose } z\text{: } \textbf{nat} \mid b(z, y).$$

where b is a Boolean-valued procedure. This produces *countable* nondeterministic choice. This construct escaped Jaco's book but was caught in Apt and Olderog [1991]. We first encountered the idea in our old office in the MC, where Ralph Back was looking at nondeterministic assignments, in the early days of his extensive theory of program refinement (Back [1980a, 1980b, 1998]).

Let us describe some recent extensions to our recent work in this area Tucker and Zucker [1999, 2002, 200?a]

In computing with the real numbers we often use the assumption of global uniform continuity, as it simplifies considerably technical definitions of the computability of functions on spaces. In metric algebras, compactness implies that continuous functions are uniformly continuous. We weaken this assumption by considering the broader class of functions that are uniformly continuous in pieces, by localising the uniformities, necessary for computability, using families of open sets

$$(U, V_0, V_1, V_2 \ \ldots) \quad \text{such that } U = \cup V_i$$

called *open exhaustions* This leads to the notion of *effective local uniform continuity*. Exhaustions are an obvious and standard way of extending computability notions via localisation. The resulting theorems have applications to the study of functions on *all* of n-space $\mathbf{R}^n \to \mathbf{R}^m$, rather than on a compact cube $[a, b]^n \to \mathbf{R}^m$.

We extend our various general notions to the localised notion using exhaustions. We can show that for connected exhaustions, on certain algebras, **while** approximation, **while-array** approximation, and polynomial approximation are equivalent. On choosing a *particular* total algebra of real numbers A_{real}, these three notions can be shown to coincide with standard GL-computability on **R** (c.f. Tucker and Zucker [1999]). Next we extend our main bridging theorem in Tucker and Zucker [200?a] to the localised case. We can show that for effectively locally uniformly continuous functions, and a wide class of metric algebras, approximation by **while-array** programs with countable choice is equivalent to a simple general notion of effectively trackable in a concrete Moschovakis-like computational model. Combining these results in the special case of real numbers we obtain:

Theorem *Let* f: $\mathbf{R}^n \to \mathbf{R}^m$ *be a function that is effectively locally uniformly continuous on an exhaustion. Then the following are equivalent:*

1. f *is GL computable on* **R**.
2. f *is "effectively trackable" on* **R**.
3. f *is locally polynomial approximable on* A_{real} .
4. f *is locally* **while array** *approximable on* A_{real} .
5. f *is locally* **while array** *with countable choice approximable on a partial algebra* B_{real}

This, together with other equivalences between concrete models and GL-computability, gives us a stable foundation for the idea of a locally computable function based on exhaustions. Next we consider the specification of these functions.

5.2 Specifications

In the theory of data, abstract data types are modelled by many sorted algebras and homomorphisms, and are specified axiomatically by equations and conditional equations. Most of the theory has been developed for data types that are discrete and countable, since they are the data types for which exact digital computation is possible. Jan Bergstra and JVT began a investigation that revealed surprising equivalences between computability, algebraic specification methods, and term rewriting; see, for instance Bergstra and Tucker [1983,1987,1995], Meseguer, Moss and Goguen [1992], Khoussainov [1998]; and surveys such as Meseguer and Goguen [1985] and Stoltenberg-Hansen and Tucker [1995].

Now, data types containing continuous data can be modelled by *topological* many sorted algebras and continuous homomorphisms, or - to take a more restricted class, closer to examples - by many sorted topological algebras that are also *metric spaces*. There are two questions we need to answer:

What methods exist to axiomatically specify functions on topological algebras?

and

Can all computable functions be specified?

The theory of topological data types is in its infancy. As we have seen there are *many* approaches to computability theory on general and specific spaces, and *few* approaches to specification theory. In Tucker and Zucker [2002] we have studied these questions.

Algebraic specification methods characterise functions as the solutions of systems of algebraic formulae that are unique in some sense. By algebraic formulae, we mean equations

$$t(X) = t'(X)$$

or conditional equations

$$t_1(X) = t_1'(X) \wedge \ldots \wedge t_k(X) = t'_k(X) \rightarrow t(X) = t'(X),$$

or other formulae, based on these, and enjoying some algebraic properties or customised to the particular algebraic context. For example, in working in metric algebras, we have as standard the sort of real numbers, so we will adapt the formulae to include (i) inequalities between reals and (ii) localisation via exhaustions. Taking (i) and (ii) together, we form specifications using *localised conditional equations and inequalities*. We can then prove that algebraic specifications can specify all computable functions on metric algebras. We say that the algebraic specification $(\Sigma^+, C(n))$ is a *universal specification* if as n varies over the natural numbers the specification defines all the computably approximable functions over all metric Σ algebras. In the case of the real numbers we can derive:

Theorem *There is a finite universal specification* $(\Sigma^+, C(\mathbf{n}))$, *consisting of conditional equations and inequalities over* Σ^+, *that defines all the locally GL computable functions on* $\mathbf{R}$.

From this it is easy to prove technical results with the informal meaning:

Theorem *If a deterministic finite dimensional dynamical system has a model that can be simulated to any degree of accuracy by an algorithm then there exists an algebraic specification that uniquely defines that algorithmic model. Indeed, for each n, there is an algebraic specification that uniformly captures all algorithmic models with n-dimensional state spaces.*

These results will appear in Tucker and Zucker [200?b].

6. Conclusion

Semantics, Verification and Process are some of the Big Ideas to emerge in Computer Science. Jaco has played an important pioneering role in the development of the modern field of Semantics of Computation from its earliest days, and has made big contributions to fields of Verification and Processes. His work will be studied with profit in the years to come.

Data is another Big Idea. The idea of working out a computability theory for data in general, and applying it, seemed a basic task in 1979 and is more so even now. Jaco's ideas and techniques in semantics shaped our joint research and, of course, our research with others. Topological data types are fundamental in modelling physical systems and are the characteristic data types of analogue processing. What is the relationship between computations with continuous data and with discrete data? The semantical theory is developing, but little is known.

Nor are the books over! We intend to write a comprehensive graduate textbook and monograph on abstract computability theory. Some of basic ideas and approaches of both Jaco's and our book have found their way in to the undergraduate textbook Tucker and Stephenson [200?].

The Netherlands is one of the most important countries in the world for research in Computer Science. Amsterdam is the most important city in the world for the theoretical research on programming and specification. Jaco is the primary architect of this city's outstanding reputation though his research, leadership and organisation at the MC and CWI for the past 38 years.

In our own case we worked with Jaco in our formative years as computer scientists and met with him many times in the years that followed. Through regular visits to Amsterdam and the hospitality of Jaco and Angeline, at home and on excursions, we have become firm and loyal friends. We owe him a great deal. There are so many scientists who owe the key steps in their progress to Jaco.

7. References

K R Apt, N Francez, W-P de Roever, A proof system for communicating sequential processes, *ACM Transactions on Programming Languages and Systems*, 2 (1980) 359-385.

K R Apt and E-R Olderog, Verification of sequential and concurrent programs, Springer Verlag, New York, 1991.

P R J Asveld and J V Tucker, Complexity theory and the operational structure of algebraic programming systems, *Acta Informatica,* 17 (1982), 451-476.

C A R Hoare, An axiomatic basis for computer programming, *Communications of the ACM*, 12 (1969), 576-580, 583.

R J R Back, Semantics of unbounded nondeterminism, in J W de Bakker and J van Leeuwen (eds.) *Automata, Languages and Programming, Seventh Colloquium, Noordwijkerhout, 1980,* Springer Lecture Notes in Computer Science 81, Springer Verlag, Berlin, 1980, 51–63. a

R J R Back, *Correctness Preserving Program Refinements: Proof Theory and Applications*, Mathematical Centre Tracts 131, Mathematical Centre, Amsterdam, 1980. b

R J R Back and J von Wright, *Refinement Calculus: A Systematic Introduction*, Graduate Texts in Computer Science, Springer Verlag, 1998.

J W de Bakker, Axiomatics of simple assignment statements, Report 94, Mathematish Centrum, Amsterdam, 1968.

J W de Bakker, *Recursive Procedures*, Tract 24, Mathematish Centrum, Amsterdam, 1973.

J W de Bakker, Semantics of infinite processes using generalised trees, in J Gruska (ed.), *Mathematical Foundations of Computer Science, 6th Colloquium*, Lecture Notes in Computer Science, 53, Springer, Berlin, 1977, 240-252.

J W de Bakker, *The Mathematical Theory of Program Correctness*, Prentice Hall International, London,1980.

J W de Bakker and J J M M Rutten, *Ten years of concurrency semantics*, World Scientific, Singapore Amsterdam, 1992.

J W de Bakker and E de Vink, *Control flow semantics*, MIT Press, 1996.

J W de Bakker and J I Zucker, Processes and the denotational semantics of concurrency, *Information and Control*, 54 (1982), 70-120. Reprinted, with errata, in De Bakker and Rutten [1992], 28-80.

J W de Bakker and J I Zucker, Compactness in semantics for merge and fair merge, in E Clarke and D Kozen (ed.), *Logics of Programs. Workshop, Carnegie-Mellon University,*

Pittsburgh, PA, June 1983, Lecture Notes in Computer Science 164, Springer Verlag, 1983, 18-33. a

J W de Bakker and J I Zucker, Processes and a fair semantics for the ADA rendez-vous, in J Diaz (ed.), *Automata, Languages and Programming: Proceedings of the 10th International Colloquium on Logic, Automata and Programming, Barcelona, Spain, July 1983*, Lecture Notes in Computer Science 154, 1983, Springer Verlag. b

J W de Bakker, J-J Ch Meyer and J I Zucker, On infinite computations in denotational semantics, *Theoretical Computer Science*, 26 (1983), 53-82.

J W de Bakker, J N Kok, J J Ch Meyer, E R Olderog and J I Zucker, Contrasting themes in the semantics of imperative concurrency, in J W de Bakker et al. (eds.) *Current Trends in Concurrency (Overviews and Tutorials): Proceedings of the ESPRIT/LPC Advanced School in Concurrency, Noordwijkerhout, The Netherlands, June 1985*, Lecture Notes in Computer Science 224, Springer Verlag, (1985), 51-121.

J W de Bakker, J-J Ch Meyer, E R Olderog and J I Zucker, Transition systems, metric spaces, and ready sets in the semantics of uniform concurrency, *Journal of Computer and System Sciences*, 54 (1988), 158-224.

J A Bergstra, J Tiuryn and J V Tucker, Correctness theories and program equivalence, Stichting Mathematisch Centrum. Informatica, IW 119/79, Amsterdam 1979, 31 pp.

J A Bergstra and J V Tucker, On the adequacy of finite equational methods for data type specification, *ACM-SIGPLAN Notices*, 14.11 (1979) 13-18.

J A Bergstra and J V Tucker, Algebraic specifications of computable and semi-computable data structures, Stichting Mathematisch Centrum Informatica, IW 115/79, Amsterdam 1979, 24pp.

J A Bergstra and J V Tucker, A characterisation of computable data types by means of a finite, equational specification method, Stichting Mathematisch Centrum Informatica, IW 124/79, Amsterdam, 1979, 23 p.

J A Bergstra and J V Tucker, A natural data type with a finite equational final semantics specification but no effective equational initial semantics specification, *Bulletin of the European Association for Theoretical Computer Science*, 11 (1980) 23-33.

J A Bergstra and J V Tucker, A characterisation of computable data types by means of a finite equational specification method, in J W de Bakker and J van Leeuwen (eds.) *Automata, Languages and Programming, Seventh Colloquium, Noordwijkerhout, 1980,* Springer Lecture Notes in Computer Science 81, Springer Verlag, Berlin, 1980, pp. 76-90.

J A Bergstra and J V Tucker, The completeness of the algebraic specification methods for data types, *Information and Control,* 54 (1982) 186-200.

J A Bergstra and J V Tucker, Initial and final algebra semantics for data type specifications: two characterisation theorems, *SIAM Journal on Computing*, 12 (1983) 366-387.

J A Bergstra and J V Tucker, Top-down design and the algebra of communicating processes, *Science of Computer Programming*, 5 (1985) 171-199.

J A Bergstra and J V Tucker, Algebraic specifications of computable and semi-computable data types, *Theoretical Computer Science,* 50 (1987) 137-181.

J A Bergstra and J V Tucker, Equational specifications, complete term rewriting systems, and computable and semicomputable algebras, *Journal of ACM*, 42 (1995) 1194-1230.

V Brattka, Recursive characterisation of computable real-valued functions and relations, *Theoretical Computer Science*, 162 (1996), 45-77.

V Brattka, *Recursive and comptable operations over topological structures*, PhD thesis, Fachbereich Informatik, FernUniversität Hagen, 1999.

J Blanck, V Stoltenberg-Hansen and J V Tucker, Streams, stream transformers and domain representations, in B Möller and J V Tucker (eds.), *Prospects for hardware foundations*, Springer Lecture Notes in Computer Science, Vol 1546, 1998, 27-68.

J Blanck, V Stoltenberg-Hansen and J V Tucker, Domain representations of partial functions, with applications to spatial objects and constructive volume geometry, *Theoretical Computer Science*, 284 (2002), 207-240.

L Blum, F Cucker, M Shub and S Smale, *Complexity and real computation*, Springer Verlag, New York, 1998.

F van Breugel, De Bakker-Zucker Processes Revisited (Dedicated to Jaco de Bakker on the occasion of his 60th birthday.) Report CS-1999-05, York University, Toronto, November 1999. To appear in *Information and Computation*.

A Edalat, Domains for computation in mathematics, physics and exact real arithmetic, *Bulletin of Symbolic Logic*, 3 (1997) 401-452.

J E Fenstad, *Generalised recursion theory*, Springer Verlag, Berlin, 1980.

J E Fenstad, Computability theory: structure or algorithms, in W Sieg, R Somer, C Talcott (eds.), *Reflections on the foundations of mathematics: Essays in honour of Solomon Feferman*, Lecture Notes in Logic, volume 15, Association for Symbolic Logic, 2002, 188-213.

C B Jones The Search for Tractable Ways of Reasoning about Programs, Manchester University, UMCS-92-4-4, 1994.

R Kuiper, An operational semantics for bounded nondeterminism equivalent to a denotational one, J W de Bakker and J C Van Vliet (eds.), *Algorithmic languages*, North-Holland, 1981, 373-398.

R W Floyd, Assigning meanings to programs, *Proceedings AMS Symposium in Applied Mathematics* 19 (1967) 19-31.

N Francez, *Program verification*, Addison Wesley, 1991.

J A Goguen, J W Thatcher, E G Wagner and J B Wright, Initial algebra semantics and continuous algebras, *Journal ACM* 24 (1977), 68-95.

B Khoussainov, Randomness, computability, and algebraic specifications, *Annals of Pure and Applied Logic*, 91 (1998) 1-15.

C Kreitz and K Weihrauch, Theory of representations, *Theoretical Computer Science* 38 (1985) 35-53.

J Moldestad, V Stoltenberg-Hansen and J V Tucker, Finite algorithmic procedures and inductive definability, *Mathematica Scandinavica*, 46 (1980) 62-76. a

J Moldestad, V Stoltenberg-Hansen and J V Tucker, Finite algorithmic procedures and computation theories, *Mathematica Scandinavica*, 46 (1980) 77-94. b

K Meinke and J V Tucker, Universal algebra, in S Abramsky, D Gabbay and T Maibaum (eds.) *Handbook of Logic in Computer Science. Volume I: Mathematical Structures*, Oxford University Press, 1992, pp.189-411.

J Meseguer and J A Goguen, Initiality, induction and computability, in M Nivat and J Reynolds (eds.), *Algebraic methods in semantics*, Cambridge University Press, 1985.

J Meseguer, L Moss and J A Goguen, Final algebra, cosemicomputable algebras and degrees of unsolvability, *Theoretical Computer Science*, 100 (1992) 267-302.

M Nivat, Infinite words, infinite trees, infinite computations, in J W de Bakker and J van Leeuwen (ed.), *Foundations of Computer Science III, Part 2: Languages, Logic, Semantics*, Mathematical Centre Tracts vol.109, Mathematical Centre, Amsterdam, 1979, 3-52.

J J M M Rutten and J I Zucker, A semantic approach to fairness, *Fundamenta Informaticae*, 16(1992), 1-38.

D Spreen, On effective topological spaces, *Journal of Symbolic Logic* 63 (1998) 185 – 221.

D Spreen, Representations versus numberings: On the relationship of two computability notions, *Theoretical Computer Science* 263 (2001), 473-499.

D Spreen and H Schulz, On the equivalence of some approaches to computability on the real line, in Keimel, K et al., (eds.) *Domains and Processes*, Proc. 1st Intern. Symp. on Domain Theory, Shanghai, China, 1999, Kluwer, Boston, 2001, 67-101.

W-P de Roever, F de Boer, U Hannemann, J Hooman, Y Lakhnech, M Poel, and J Zwiers, *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*, Cambridge University Press, 2001.

V Stoltenberg-Hansen, I Lindstrom and E R Griffor, *Mathematical Theory of Domains*, Cambridge University Press, 1994.

V Stoltenberg-Hansen and J V Tucker, Computing roots of unity in fields, *Bulletin of the London Mathematical Society*, 12 (1980) 463-471.

V Stoltenberg-Hansen and J V Tucker, Complete local rings as domains, *Journal of Symbolic Logic*, 53 (1988) 603-624.

V Stoltenberg-Hansen and J V Tucker, Algebraic equations and fixed-point equations in inverse limits, *Theoretical Computer Science*, 87 (1991) 1-24.

V Stoltenberg-Hansen and J V Tucker, Infinite systems of equations over inverse limits and infinite synchronous concurrent algorithms in J W de Bakker, G Rozenberg, and W P de Roever (eds.) *Semantics - Foundations and applications*, Springer Lecture Notes in Computer Science 666, Springer Verlag, 1993, 531-562.

V Stoltenberg-Hansen and J V Tucker, Effective algebras, in S Abramsky, D Gabbay and T Maibaum (eds.) *Handbook of Logic in Computer Science. Volume IV: Semantic Modelling* , Oxford University Press, 1995, pp.357-526.

V Stoltenberg-Hansen and J V Tucker, Computable rings and fields, in E Griffor (ed.), *Handbook of Computability Theory*, Elsevier, 1999, pp.363-447. a

V Stoltenberg-Hansen and J V Tucker, Concrete models of computation for topological algebras, *Theoretical Computer Science*, 219 (1999) 347-378. b

J V Tucker, Computing in algebraic systems, in F R Drake and S S Wainer (eds.) *Recursion Theory, its Generalisations and Applications*, London Mathematical Society Lecture Note Series 45, Cambridge University Press, Cambridge, 1980, pp. 215-235.

J V Tucker, Applications of computability theory over abstract data types, in J W Klop (ed.) *J W de Bakker: 25 Jaar Semantiek. Liber Amicorum,* CWI Amsterdam, 1989, 421-432.

J V Tucker, Theory of computation and specification over abstract data types and its applications, in F L Bauer (ed.), Proceedings of NATO Summer School 1989 at Marktoberdorf, in *Logic, algebra and computation*, Springer, 1991, pp.1-39.

J V Tucker and K Stephenson, *Data, syntax and semantics*, in preparation.

J V Tucker and J I Zucker, *Program correctness over abstract data types with error-state semantics*, CWI Tract 6, North-Holland, Amsterdam, 1988.

J V Tucker and J I Zucker, Horn programs and semicomputable relations on abstract structures, in G Ausiello, M Dezani-Ciancaglini, S Ronchi Della Rocca (eds.) *Automata, Languages and Programming, Sixteenth Colloquium, Stresa, 1989,* Springer Lecture Notes in Computer Science 372, Springer Verlag, Berlin, 1989, pp.745-760.

J V Tucker and J I Zucker, Toward a general theory of computation and specification over abstract data types, in S G Akl, F Fiala, and W W Koczkodaj (eds.), *Advances in computation and information, May 1990,* Canadian Scholars Press, 1990, pp.101-102. Also book republished in Springer Lecture Notes in Computer Science 468, Springer Verlag, Berlin, 1990, 129-133.

J V Tucker and J I Zucker, Examples of semicomputable sets of real and complex numbers, in J P Myers Jr and M J O'Donnell (eds.), *Constructivity in computer science*, Springer Lecture Notes in Computer Science 613, Berlin, pp.179-198.

J V Tucker and J I Zucker, Projections of semicomputable relations on abstract data types, *International J of Foundations of Computer Science* 2 (1991) 267-296.

J V Tucker and J I Zucker, Deterministic and nondeterministic computation, and Horn programs, on abstract data types, *Journal of Logic Programming*, 13 (1992) 23-55.

J V Tucker and J I Zucker, Theory of computation over stream algebras, and its applications, in I M Havel and V Koubek (eds.) *Mathematical Foundations of Computer Science 1992, 17th International Symposium, Prague*, Springer Lecture Notes in Computer Science 629, Berlin, 62-80.

J V Tucker and J I Zucker, Provable computable selection functions on abstract structures, in P Aczel, H Simmons and S S Wainer (eds.) *Proof theory*, Cambridge University Press, 1993, 277-306.

J V Tucker and J I Zucker, Computable functions on stream algebras, in H Schwichtenberg (ed.), Proceedings of NATO Advanced Study Institute, International

Summer School 1993 at Marktoberdorf, in *Proof and Computation*, Springer, 1994, 341-382.

J V Tucker and J I Zucker, Computation by while programs on topological partial algebras, *Theoretical Computer Science*, 219 (1999) 379-421.

J V Tucker and J I Zucker, Computable functions and semicomputable sets on many sorted algebras, in S Abramsky, D Gabbay and T Maibaum (eds.) *Handbook of Logic for Computer Science Volume V*, Oxford University Press, 2000, 317-523.

J V Tucker and J I Zucker, Infinitary initial algebraic specifications for stream algebras, in W Sieg, R Somer, C Talcott (eds.), *Reflections on the foundations of mathematics: Essays in honour of Solomon Feferman*, Lecture Notes in Logic, volume 15, Association for Symbolic Logic, 2002, 234-253.

J V Tucker and J I Zucker, Abstract computability and algebraic specification, *ACM Transactions on Computational Logic*, 3 (2002) 279-333.

J V Tucker and J I Zucker, Abstract versus concrete models of computation on partial metric algebras, *ACM Transactions on Computational Logic*, accepted, 200?a.

J V Tucker and J I Zucker, Computable total functions, algebraic specification and dynamical systems, submitted, 200?b.

S S Wainer, J V Tucker and J I Zucker, Provably computable functions on abstract data types, in M Patterson (ed.) *Automata, Languages and Programming, Seventeenth Colloquium, Coventry, 1990,* Springer Lecture Notes in Computer Science 443, Springer Verlag, 1990, pp.660-673.

K W Weihrauch, *Computable analysis*, Springer Verlag, 2000.

A van Wijngaarden, Numerical analysis as an independent science, *BIT* 6 (1966), 66-81.

Onbekende Havens

I have had the privilege of having Jaco as "meta-supervisor" during the years I spent at the CWI (1991-96).

Extraordinary was his understated ability to exert, through few measured strokes, a much needed steadying influence on the rocking boat that was my thesis. His natural sensitivity made him repeatedly choose what proved the best course of action, putting the success of the enterprise above anything else.

But when I think of Jaco the first image that comes to my mind is his smile. When I was still at the CWI I could not made full sense of that intriguing smile, where ever changing shades would as much cover as express the mobile spirit living behind it. It was only at a Rembrandt exhibition years later that I found myself feeling that I suddenly understood both these classic paintings and Jaco's smile.

Now, just few days ago, chance brought me back in contact with Jaco after all these years. And, clearer than ever, I saw his smile emerge from the lines he wrote me back, als de geheime stoet voorbijtrekte...

Daniele Turi
Edinburgh, May 2002

Voor mij blijft het allemaal Grieks

Hans van Vliet
Vrije Universiteit Amsterdam
Department of Mathematics and Computer Science
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
e-mail: hans@cs.vu.nl

6 juni 2002

Ik ben in 1967 in dienst getreden bij het CWI, toen nog Mathematisch Centrum geheten, en gehuisvest in een oud schoolgebouw in de Tweede Boerhaavestraat. Ik ben daar aangenomen door Kruseman Aretz, hoofd van de Rekenafdeling. Na het vertrek van Kruseman Aretz naar Eindhoven werd Reind van de Riet mijn baas, en na diens vertrek naar de VU dus Jaco. Inmiddels had ik zelf ook promotie gemaakt, van computer operator tot programmeur. Ik zat samen met enkele collega-programmeurs op een kamer op de eerste verdieping, recht onder de huisdrukkerij. Midden in onze kamer stond een heipaal, om te voorkomen dat het plafond op ons zou neerstorten onder druk van de machines en stapels papier boven ons. Later werd er op de binnenplaats een barak neergezet. Ik verhuisde, samen met een flink deel van de juniores van de Rekenafdeling, naar deze barak. Dat was een hele verbetering; het stonk er altijd naar verschaald bier, de zon zag je nooit, en als het glad was gleed je uit over algen en mos als je van de barak naar het hoofdgebouw wilde.

Zoals gebruikelijk waren de werkelijk belangrijke mensen, zoals de directeur, en ook Jaco, gehuisvest op de bovenste verdieping van het gebouw. Dat was ook een soort barak, die al eerder boven op het dak van het schoolgebouw was geplaatst. Wij kwamen daar niet zo vaak. Jaco was een chef op afstand. Letterlijk en figuurlijk, want ik sprak hem netjes aan met U, en meneer. Toen ik mijn doctoraal haalde mocht ik je en jij gaan zeggen. Dat kostte me nog enige tijd moeite.

Zoals gezegd bemoeide Jaco zich inhoudelijk niet zo veel met ons werk. Ik was inmiddels betrokken geraakt bij het zogenaamde ALGOL 68 project, waarin een groep mensen jarenlang geprobeerd heeft informatica-onderzoek te combineren met het schrijven van een compiler voor ALGOL 68. Andy Tanenbaum heeft nog een blauwe maandag in dit project gewerkt, en onder andere het concept-contract opgesteld dat wij met onze opdrachtgever wilden afsluiten; zijn Engels was beter dan dat van ons. Dick Grune deed onderzoek naar een taal om compilers in te schrijven, en promoveerde daarop. Lambert Meertens werkte tussen al zijn andere hobbies door mee in dit project. Zo rond 1980 zagen wij de futiliteit in van ons streven, en besloten het project stop te zetten. Jaco steunde ons daarin volledig. Inmiddels echter had de afdeling een heuse externe adviescommissie. Die was helaas minder gecharmeerd van onze beslissing, en wenste een nette afronding middels nog wat extra publikaties. En zo ploeterden we nog een tijdje voort. Het ALGOL 68 project heeft mij heel nadrukkelijk wel op het pad van de software engineering gezet.

In 1973 ben ik naast mijn werk bij het Mathematisch Centrum wiskunde gaan studeren aan de VU. Aan de VU, omdat ze daar een afstudeerrichting informatica hadden. Ik 'volgde' daar vakken als datastructuren bij Reind van de Riet, computerorganisatie bij Andy Tanenbaum, en formele talen alsmede semantiek van programmeertalen bij Jaco. Ik schrijf 'volgde' omdat ik naast mijn werk weinig gelegenheid had om colleges te volgen, en dus in ieder geval de informaticavakken probeerde te halen door gewoon het boek of de syllabus te bestuderen. Dat deed ik ook voor de twee vakken die Jaco verzorgde. Voor het vak Formele talen en automaten gebruikten we het bekende boek van Hopcroft & Ullman, *Formal Languages and their Relation to Automata*, aangevuld met een syllabus die geschreven was door Arie de Bruin. Voor het vak Semantiek was er ook een syllabus, die later uitgroeide tot een heus boek, *Mathematical Theory of Program Correctness*.

Dit waren zeer geleerde syllabi en boeken, die vol stonden met Griekse symbolen. Aangezien ik slechts

simpel de hbs had gedaan, had ik daar de nodige moeite mee. Ik kende wel de namen van een aantal Griekse letters, maar wist niet goed welke symbolen daarbij hoorden. Ik moest op enig moment mondeling tentamen doen bij Jaco. Gewoon op de bovenste verdieping van het Mathematisch Centrum. Jaco stelde een vraag, en ik gaf antwoord, en schreef tegelijkertijd op het bord. Dat werd een nachtmerrie, voor ons allebei. Bij het vak Formele talen viel het nog wel mee, omdat we daar niet veel verder kwamen dan de eerste drie letters van het alfabet. Daar moest ik dingen leren als (definitie 2.1.3):

$\alpha \overset{\Rightarrow}{G} \beta$ (α produceert β onder G in één stap), dan en slechts dan als er α_1, α_2 uit V^*, γ uit V^+ en N uit V_N bestaan, zodanig dat $\alpha = \alpha_1 N \alpha_2, \beta = \alpha_1 \gamma \alpha_2$ en $(N, \gamma) \in P$.

Maar wat dan te denken van Stelling 3.36 uit Jaco's eigen boek:

Let $\pi = \mathcal{T}(\mathrm{b}), \pi' = \mathcal{T}(\mathrm{p}), \phi = \bigsqcup_i \phi_i$, with ϕ_i as in definition 3.22, $\pi_0 = \lambda\tau \bullet \mathrm{ff}$, $\pi_{i+1} = \pi' \vee \{(\pi \wedge \pi_i) \leftarrow \mathcal{M}(S)\}$. Then

$$\pi' \leftarrow \phi = \neg\pi \wedge \bigsqcup_{i=0}^{\infty} \pi_i.$$

(Het was voor mij ook al een tijdje geleden dat ik dit las, dus ik zag ook niet meteen dat dit de semantische karakterisering van de sterkste postconditie van een while-loop is.) Bij de bespreking van deze stelling zei ik dan bijvoorbeeld zoiets als "met psi sub i zoals in definitie 3.22", en schreef op het bord "met ϕ_i als in ...". Jaco onderbrak me dan regelmatig om me een lesje Grieks te geven: "Hans, dat symbool heet phi, en niet psi". In mijn herinnering ging het dat hele uur om het verbeteren van mijn Grieks. Wonder boven wonder kreeg ik ook nog een voldoende.

Grieks en programmacorrectheid zijn geen van beide echt hobbies van me geworden. Ik ken inmiddels wel de eerste vijf letters van het Griekse alfabet, en heb (nog steeds) enige notie van het begrip programmacorrectheid. Waarvoor, naast vele andere zaken, dank.