# Simplex sigillum veri : een liber amicorum voor prof.dr. F.E.J. Kruseman Aretz aangeboden op 15 december 1995 ter gelegenheid van zijn afscheidscollege aan de Technische Universiteit Eindhoven

# Simplex Sigillum Veri

## Een Liber Amicorum voor prof.dr. F.E.J. Kruseman Aretz

Samenstelling:

E.H.L. Aarts
H.M.M. ten Eikelder
C. Hemerik
M. Rem

Technische Universiteit tu/e Eindhoven

# Simplex SigillumVeri

Een Liber Amicorum
voor prof.dr. F.E.J. Kruseman Aretz

# Simplex SigillumVeri

Een Liber Amicorum
voor prof.dr. F.E.J. Kruseman Aretz

Aangeboden op 15 december 1995
ter gelegenheid van zijn afscheidscollege
aan de Technische Universiteit Eindhoven

Samenstelling:

E.H.L. Aarts
H.M.M. ten Eikelder
C. Hemerik
M. Rem

Technische Universiteit tu/e Eindhoven

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Simplex

Simplex Sigillum Veri: een liber amicorum voor prof.dr. F.E.J. Kruseman Aretz aangeboden op 15 december 1995 ter gelegenheid van zijn afscheidscollege aan de Technische Universiteit Eindhoven / samenst.: E.H.L. Aarts ... [et. al.]. - Eindhoven: Technische Universiteit Eindhoven
Teksten in het Nederlands en Engels.
ISBN 90-386-0197-2
Trefw.: informatica.

Uitgegeven door: Technische Universiteit Eindhoven
Druk: Universitaire drukkerij, TU Eindhoven

ISBN 90-386-0197-2

# Inhoudsopgave

Prof.dr. F.E.J. Kruseman Aretz

# Voorwoord

Deze bundel is een liber amicorum dat wij Prof.dr. F.E.J. Kruseman Aretz aanbieden ter gelegenheid van zijn afscheidsrede aan de Technische Universiteit Eindhoven. Het is een rijk geschakeerd geheel geworden. De bundel bevat een aantal persoonlijke herinneringen — onder meer nog uit de Amsterdamse tijd van Frans — en ook veel verhandelingen over onderwerpen waarvan we weten dat ze Frans na aan het hart liggen. Het samenstellen van dit liber hebben we met veel plezier gedaan; we zijn er van overtuigd dat ook Frans en Loes Kruseman Aretz vele prettige uren met dit boek zullen doorbrengen.

Hoewel ieder van de auteurs zijn of haar eigen herinneringen heeft aan Frans Kruseman Aretz, kennen wij hem allen vooral als iemand die zich, al sinds de jaren zestig, met bijzonder veel toewijding heeft ingezet voor het onderwijs in de informatica. Hij laat zich daarbij leiden door de informatica als een ontwerp-discipline, een discipline die haar artefacten schept op basis van effectieve, wiskundige redeneringen. In zijn onderwijs, ook als dat in de bedrijfsmatige context van een multinational plaatsvindt, streeft Frans altijd naar het zo eenvoudig mogelijk overdragen van de quintessens van het vak. In zijn ogen moet je als docent belangrijke zaken zo helder en scherp mogelijk formuleren. Wij hebben daarom de lijfspreuk van Hermanus Boerhaave „Simplex Sigillum Veri" als titel van dit liber gekozen. Het feit dat Frans zijn eerste schreden op het pad van de informatica zette aan de 2$^{e}$ Boerhaavestraat te Amsterdam is slechts een toevalligheid.

Het verschijnen van deze bundel was natuurlijk niet mogelijk geweest zonder de inspanningen van alle individuele auteurs. De samenstellers zijn hen daar zeer erkentelijk voor. Het omzetten van alle bijdragen tot een uniform geheel bleek een tijdrovende bezigheid, die gelukkig enigszins verlicht werd door de hulp van Gerard Zwaan, Pascal Coumans en Margret Philips. De hulp van Loes Kruseman Aretz was onmisbaar bij het verzamelen van foto's. Wij danken ook de Stichting Instituut voor Informatica te Eindhoven, welke zonder enige aarzeling bereid bleek de drukkosten voor haar rekening te nemen.

Wij spreken de hoop uit dat Frans en Loes Kruseman Aretz en alle andere lezers van deze bundel veel plezier en voldoening mogen beleven aan deze ode aan de schoonheid van de eenvoud.

De samenstellers,
Prof.dr. E.H.L. Aarts
Dr.ir. H.M.M. ten Eikelder
Dr.ir. C. Hemerik
Prof.dr. M. Rem

# Deel I

# Over Frans Kruseman Aretz

# Levensloop

Frans Kruseman Aretz werd geboren in Malang (Oost-Java) op 15 april 1933. Hij volgde zijn middelbare-schoolopleiding aan het Vossius-Gymnasium te Amsterdam van 1946 tot 1952 en behaalde in 1952 het eindexamen gymnasium-beta.

Van 1952 tot 1960 studeerde hij aan de Universiteit van Amsterdam. In 1955 slaagde hij cum laude voor het candidaatsexamen d (wiskunde en natuurkunde met scheikunde) en in mei 1960 cum laude voor het doctoraal examen theoretische natuurkunde met bijvakken wiskunde en mechanica. In mei 1962 promoveerde hij aan de Universiteit van Amsterdam tot doctor in de wiskunde en natuurwetenschappen op een proefschrift getiteld „Moment Expansions in the Theory of Cooperative Phenomena". Promotor was Prof.Dr. J. de Boer, coreferent en begeleider van het promotieonderzoek was Dr. E.G.D. Cohen. Tijdens zijn promotiewerk was hij in dienst van de Stichting voor Fundamenteel Onderzoek der Materie.

In september 1962 trad hij in dienst van de Stichting Mathematisch Centrum om, onder leiding van Prof.Dr. A. van Wijngaarden en Dr. J.A. Zonneveld, werkzaam te zijn op de rekenafdeling van het gelijknamige instituut te Amsterdam (nu CWI geheten). Hij werkte daar ondermeer aan de ALGOL 60 vertaler voor de EL X8 en aan operating systems voor die machine. In 1967 werd hij bovendien benoemd aan de Universiteit van Amsterdam tot bijzonder hoogleraar vanwege de Stichting voor Hoger Onderwijs in de Toegepaste Wiskunde. Zijn leeropdracht was de toegepaste wiskunde. Hij vervulde die functie tot 1972 en in die periode gaf hij onderwijs in programmeren, programmeertalen en implementatietechnieken.

In augustus 1969 kwam hij naar het Natuurkundig Laboratorium van Philips. In zijn werkzaamheden bij Philips kunnen twee perioden onderscheiden worden. Van 1969 tot 1981 was hij werkzaam op het rekencentrum van het laboratorium. Hij werkte daar wederom aan vertalers en operating systems. Zo ontstonden MILLI voor de EL X8, een ALGOL 60-vertaler voor de P1400 en P1800 en werkte hij mee aan het MILLI- en LEO-systeem voor de P1800. Daarnaast hielp hij onderzoekers bij het oplossen van reken- en programmeerproblemen. In de tweede periode, van april 1981 tot aan zijn pensioenering eind april 1993, was hij afwisselend werkzaam in de groepen computer architectuur en software design. Hij hield zich daar bezig met fundamenteel onderzoek op het gebied van programma-ontwerp en parsing, en had verder een adviserende taak. Gedurende zijn hele Philips-tijd is hij actief geweest op het gebied van onderwijs en interne opleidingen. Naast diverse cursussen Pascal en LISP gaf hij mede vorm aan cursussen als CSO (cursus software-ontwerp), CPO (cursus programma-ontwerp) en SEM (software-educatie voor managers). Als lid van de Kwalificatie-Commissie OIT ondersteunde hij ook de feitelijke invoering van het

OIT-programma.

Vanaf januari 1972 tot april 1995 was hij aan de Technische Universiteit Eindhoven verbonden als buitengewoon hoogleraar voor een dag in de week. Zijn leeropdracht was hier de informatica. Als zijn hoofdtaak heeft hij van meet af het geven van onderwijs gezien, daarbij inbegrepen het begeleiden van afstudeerders en promovendi. In totaal begeleidde hij 12 wiskunde-studenten met specialisatie informatica, 15 informatica-studenten en 2 studenten electrotechniek bij hun afstuderen. Hij trad 7 maal op als eerste promotor en 5 maal als tweede promotor, en hij was lid van 29 andere promotiecommissies. Zijn colleges waren onder meer gewijd aan programmeertalen, compilers, denotationele semantiek, het ontwerpen van parallelle programma's met behulp van monitoren, functioneel programmeren, het ontwerp van een file-systeem en parsing. Steeds lag daarbij de nadruk op die ontwerpprincipes die leiden tot elegante en eenvoudige oplossingen voor ingewikkelde problemen.

Naast zijn onderwijstaken heeft hij op de TUE ook vele organisatorische bijdragen geleverd. Dit was in de eerste jaren bijna onvermijdelijk gezien de uiterst kleine staf van informatici, en dat is later, na de instelling van een studierichting informatica en de daarmee sterk groeiende omvang van de vakgroep, eigenlijk steeds zo gebleven. Zo heeft hij een duidelijk stempel gezet op de vormgeving van het curriculum, zowel voor de onderbouw als voor de bovenbouw. Landelijk is hij betrokken geweest bij diverse ZWO-adviescommissies en is hij lid geweest van de Nederlandse Commissie voor de Wiskunde.

# Promovendi

## Eerste promotor

12-02-82 H.B.M. Jonkers
Abstraction, specification and implementation techniques: with an application to garbage collection

30-03-82 P. Klint
From SPRING to SUMMER: Design, definition and implementation of programming languages for string manipulation and pattern matching

15-05-84 C. Hemerik
Formal definitions of programming languages as a basis for compiler construction

23-02-90 L.M.G. Feijs
A formalisation of design methods: A $\lambda$–calculus approach to system design with an application to text editing

07-10-93 A. Augusteijn
Functional programming, program transformations and compiler construction

05-10-94 E. Poll
A programming logic based on type theory

15-09-95 B.W. Watson
Taxonomies and toolkits of regular language algorithms

## Tweede promotor

12-10-76 M. Rem
Associons and the closure statement

16-03-84 E.O. de Brock
Database models and retrieval languages

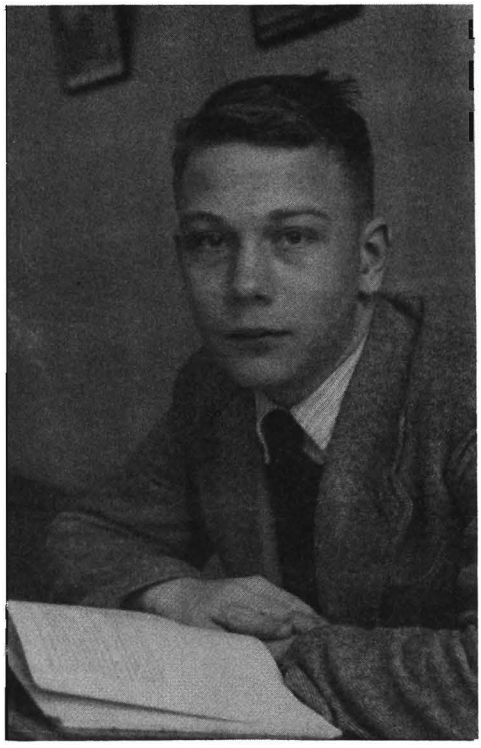
06-05-86 A. Kaldewaij
A formalism for concurrent processes

20-12-88 A.J.M. van Gasteren
On the shape of mathematical arguments

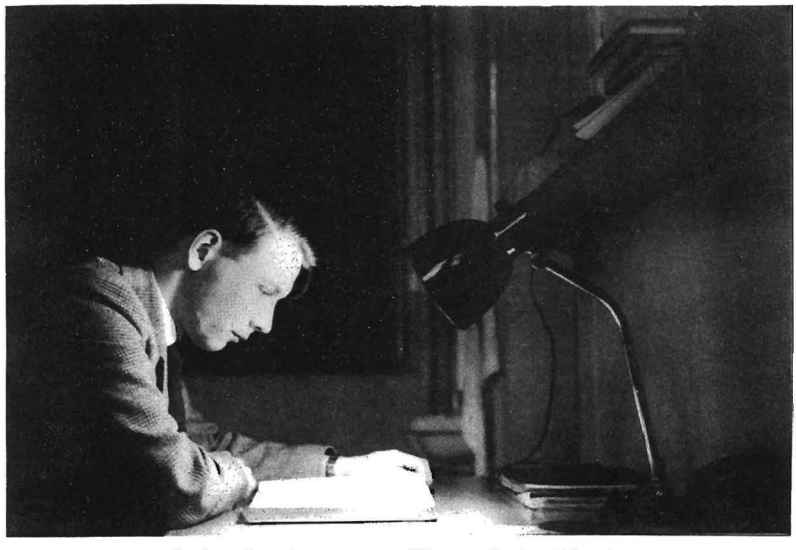
19-12-89 R.R. Hoogerwoord
The design of functional programs: a calculational approach

# Foto's

Fransje aan de wandel (1), 19 maanden oud

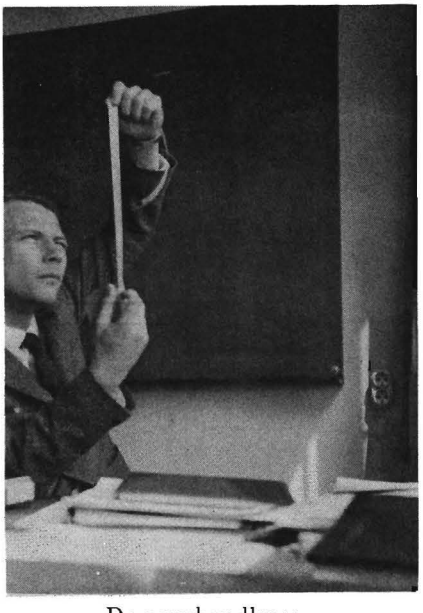


Op het gymnasium

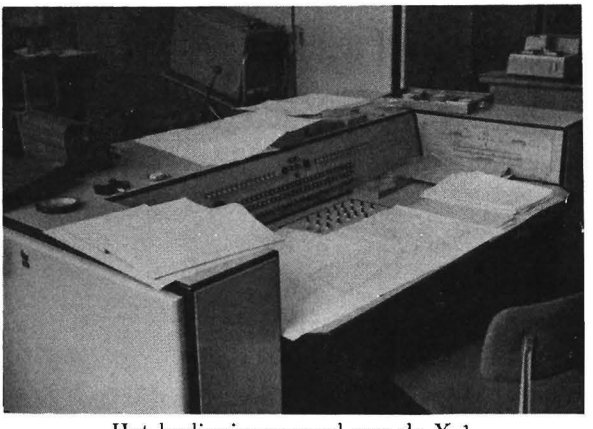


In het Instituut voor Theoretische Physica

„Het Centrum"



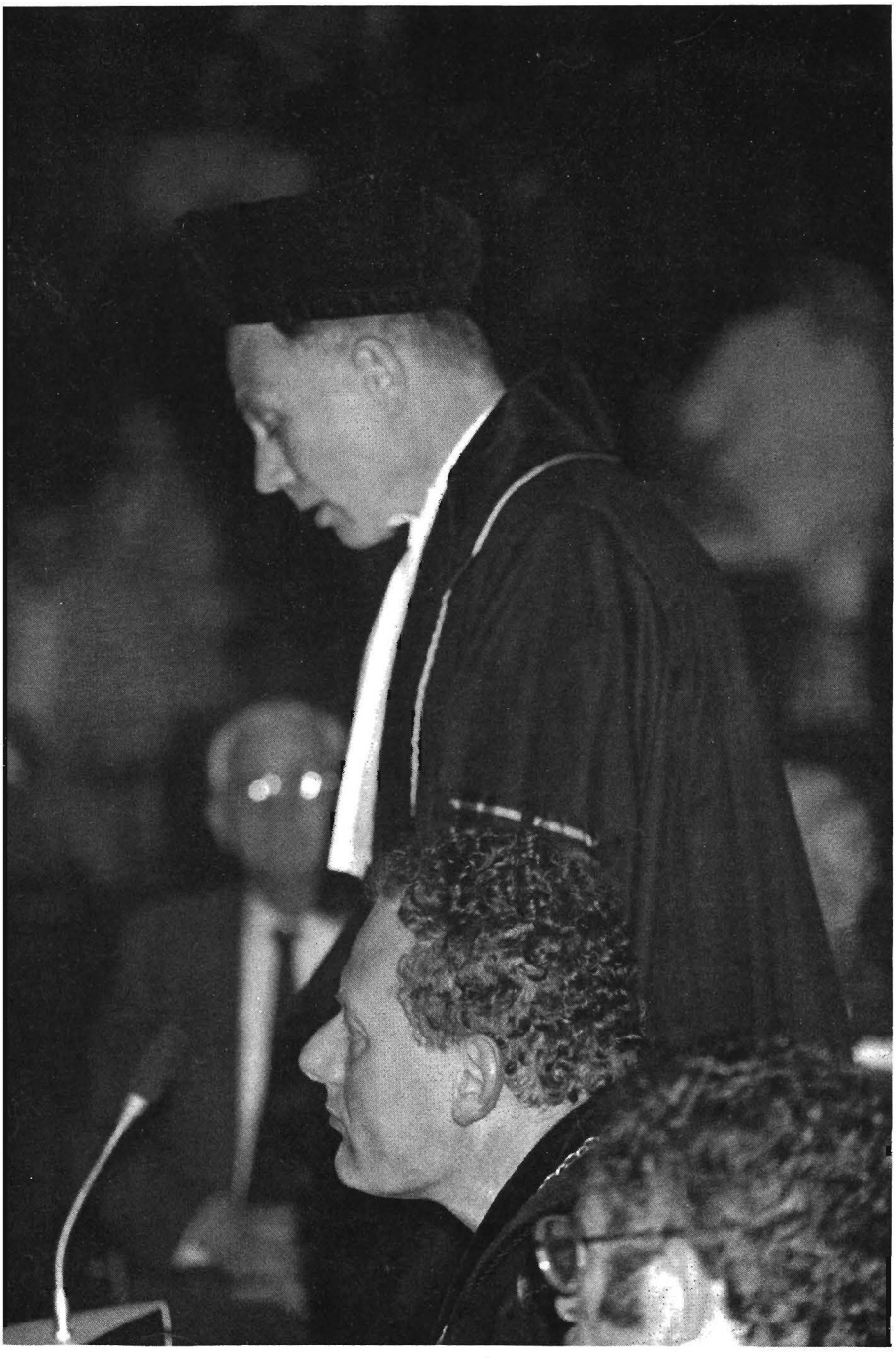
De ponsbandlezer



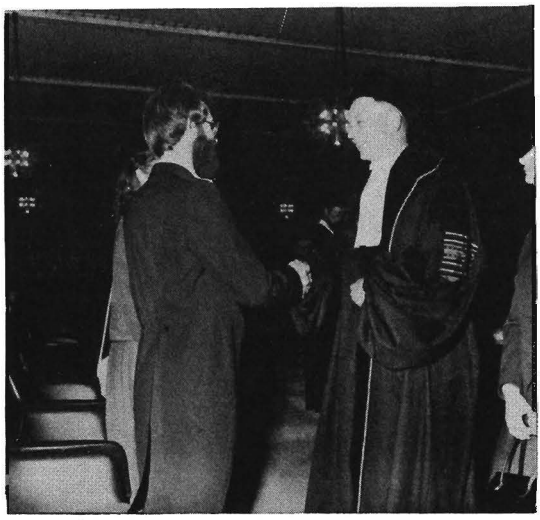
Het bedieningspaneel van de X-1

Bij de promotie van R.P. van de Riet

Ten tijde van zijn benoeming aan de Technische Hogeschool Eindhoven

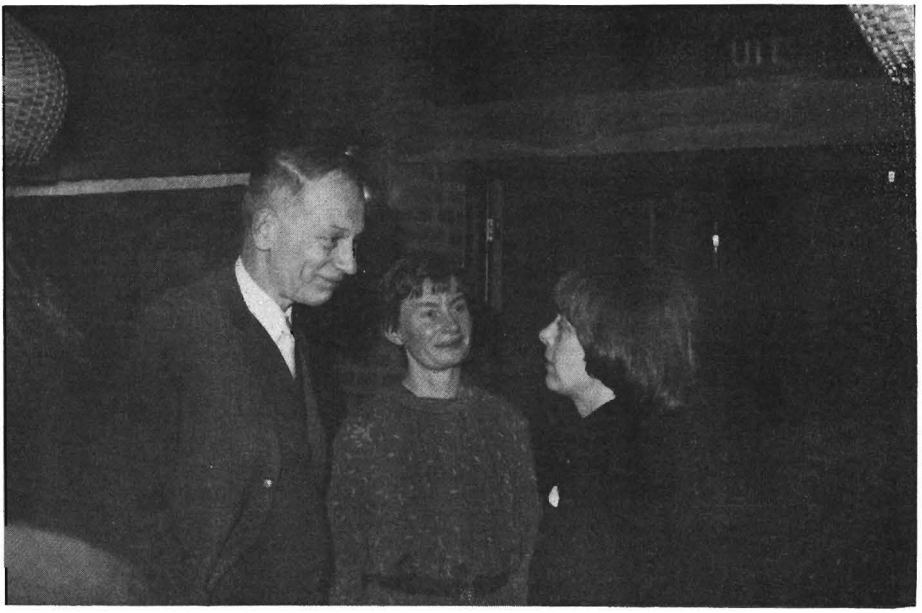


Met Stan Ackermans tijdens een promotie



Bij de promotie van Jan van de Snepscheut

Bij de promotie van Bruce Watson

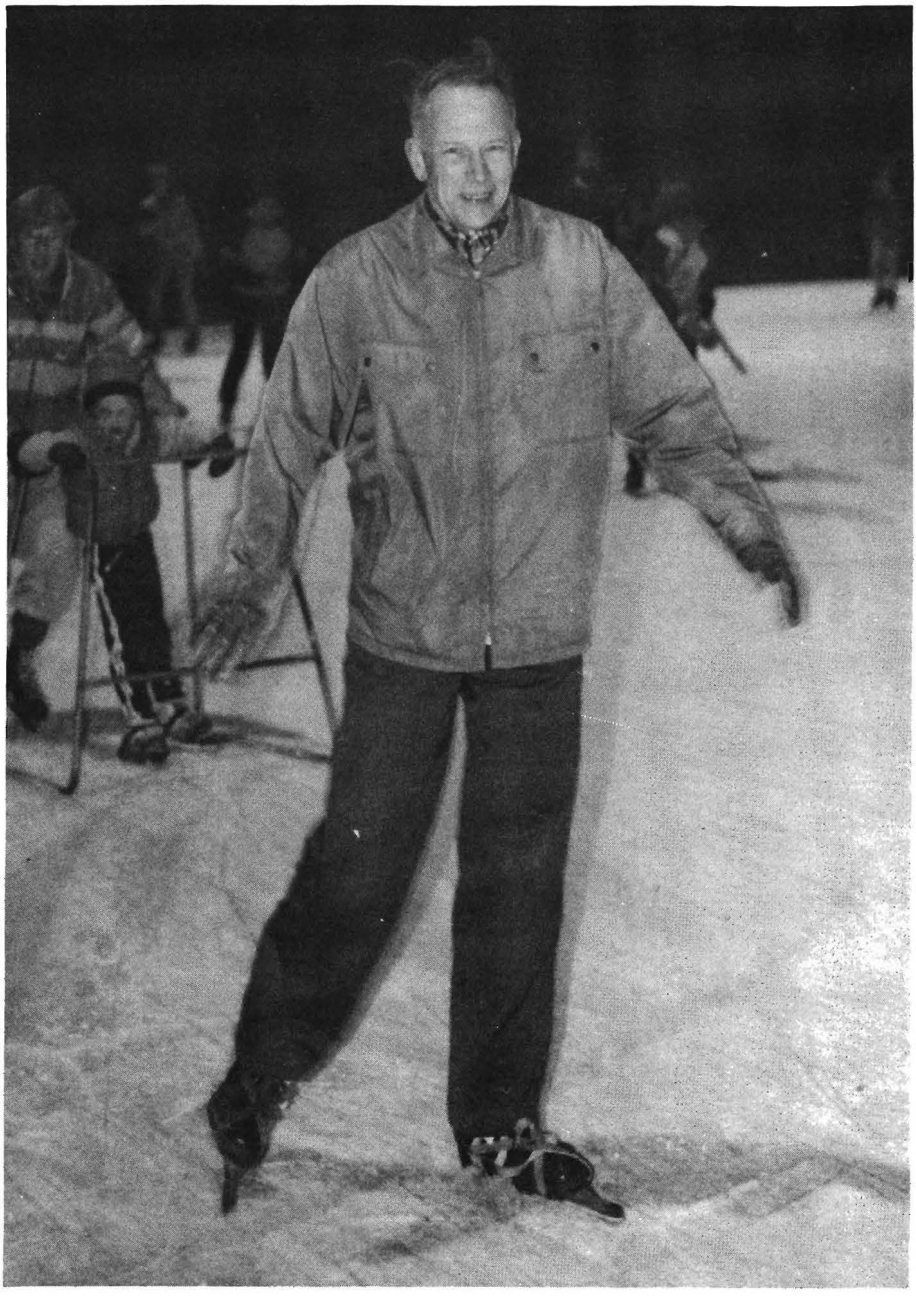


Met Loes en Netty van Gasteren

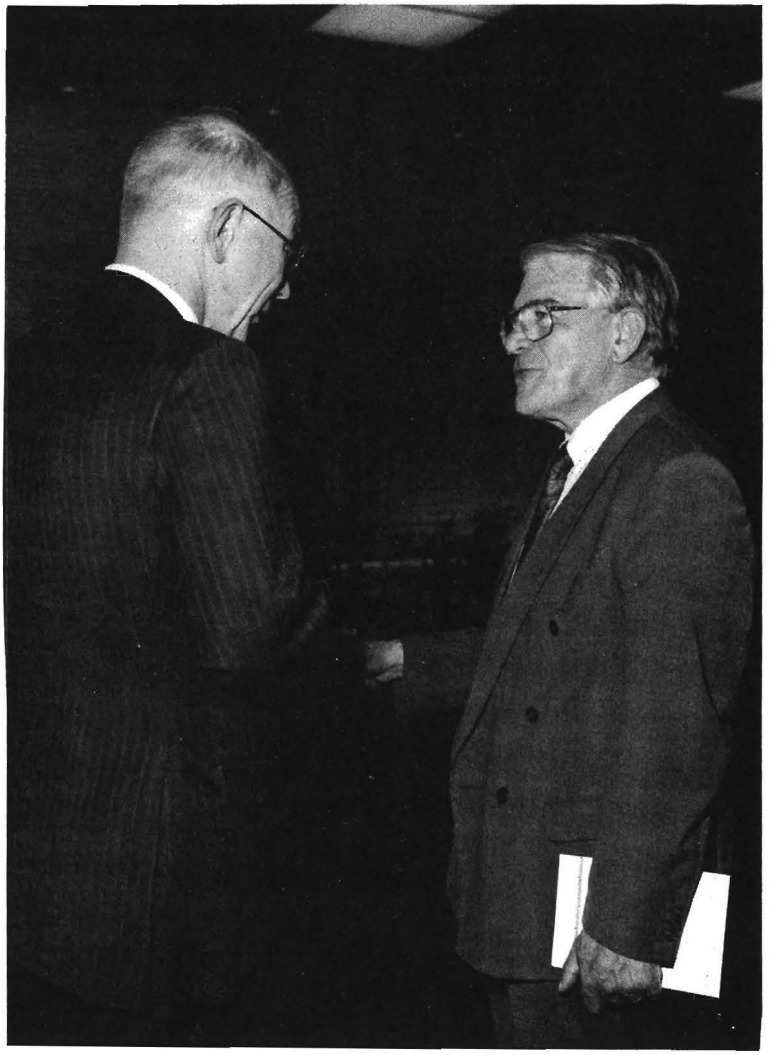
Met de sectie PI

Fransje aan de wandel (2) met Loes

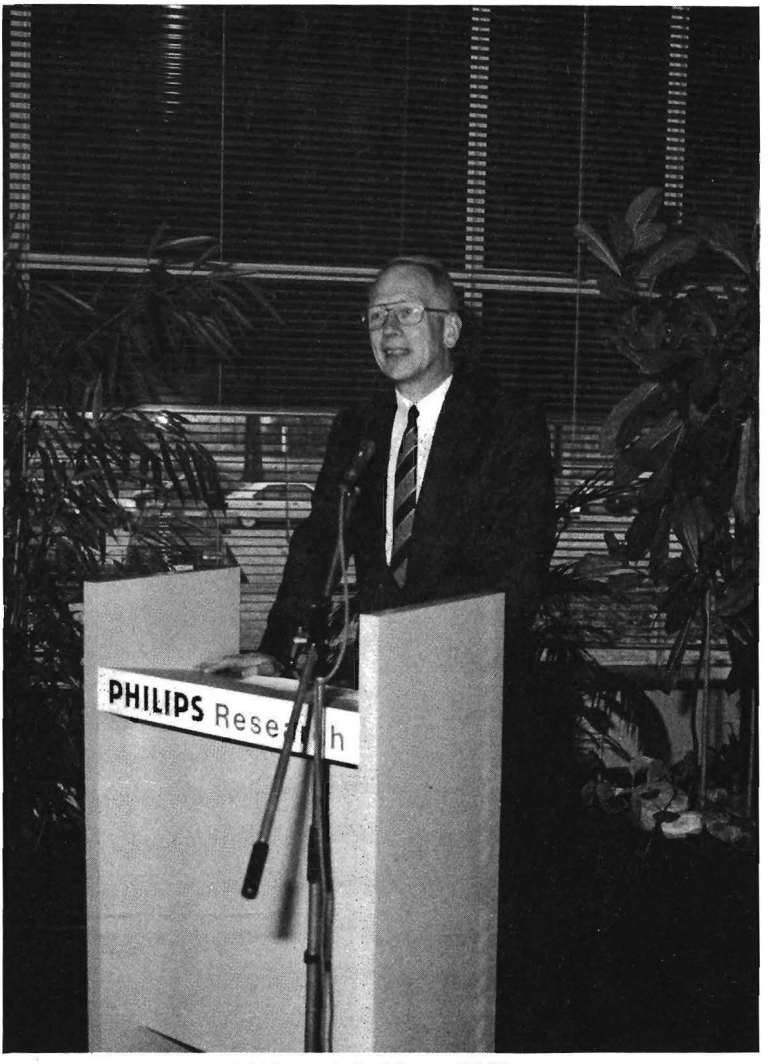
Fransje aan de wandel (3)



Frans rijdt een scheve schaats



Met Frans Remmen




Bij het afscheid van Philips

# Deel II

# Voor Frans Kruseman Aretz

# Bekentenissen over het boek van een vriend

Emile Aarts

Mijn bijdrage aan jouw vriendenboek, beste Frans, bevat een tweetal bekentenissen die betrekking hebben op een boek dat jij me als vriend ooit hebt geleend. Ik ben er nooit toe gekomen om jou de gang van zaken rond deze leen te vertellen en dit lijkt mij een prima gelegenheid om dat alsnog te doen. Het gaat hierbij niet om een grootse zaak; het betreft gewoon iets kleins, dat je misschien leuk vindt om te weten.

**Kennismaking**

Onze eerste ontmoeting dateert uit 1983, het jaar waarin ik als jong-gepromoveerd fysicus de gelederen van het Philips Natuurkundig Laboratorium mocht proberen te versterken. Ik werd toegevoegd aan de informatica sector, die in die tijd geleid werd door Henk Bosma, en de groep waarin ik terecht kwam hield zich bezig met het ontwikkelen van formele methoden voor VLSI-ontwerp. Jijzelf zat toen in een groep die onder leiding stond van Loek Nijman en waarin aan ambitieuze onderwerpen gewerkt werd, zoals parallelle computersystemen, natuurlijke-taalverwerking, compiler constructie en expert-systemen. Als nestor van de informatica op het laboratorium had jij je sporen al verdiend door in nauwe samenwerking met Carel Scholten en Edsger Dijkstra gestalte te geven aan de vroege ontwikkeling van de informatica in Nederland. Ik moest het meeste van de informatica toen nog leren en gelukkig was jij bereid me daarbij te helpen.

**Een eerste bekentenis**

Tijdens een van onze leergesprekken wees jij me op het bestaan van het boek *Computers and Intractability: A Guide to the Theory of NP-Completeness* van Michael Garey & David Johnson [1979], dat in de wandelgangen beter bekend staat als de *Garey & Johnson* (G&J). Je raadde me aan het boek te bestuderen en je leende me daartoe je eigen exemplaar. De eerste bekentenis die ik wil afleggen is dat ik niet onmiddellijk nadat je me het boek geleend hebt met de bestudering ervan ben begonnen. Het heeft zeker twee jaar geduurd voordat ik er ook maar een letter in gelezen heb, en dat is tevens de reden waarom ik het boek zo lang in mijn bezit gehouden heb.

Natuurlijk heb ik jouw raadgeving ter harte genomen en ben ik uiteindelijk toch begonnen met de bestudering van het boek. Dat was mijn eerste kennismaking met de complexiteitstheorie en al snel bleek dat de stof bijzonder fascinerend is. Ik had er spijt van dat ik er niet eerder in begonnen was. G&J en de complexiteit van berekeningen hebben sindsdien een belangrijke rol in mijn werk gespeeld. Waarschijnlijk zou ik via andere wegen ook met G&J in aanraking zijn gekomen, maar ik heb het altijd als een speciale gebeurtenis gezien dat mijn eerste kennismaking met het boek en het onderwerp via jou liep.

### Garey & Johnson

Historisch gezien moet de uitgifte van de eerste druk van G&J geplaatst worden in de tijd dat Cook [1971] het bewijs leverde de het probleem SATISFIABILITY NP-volledig is en Karp [1972] een hierarchie van polynomiale-tijdreducties opstelde waarmee de eerste 21 NP-volledige problemen geïdentificeerd werden, waaronder het handelsreizigersprobleem. Voor velen heeft het boek vooral bekendheid verworven door de geannoteerde lijst met ruim 200 NP-volledige problemen die vaak als uitgangspunt voor nieuwe reducties dienen, maar even zo vaak foutief aangehaald worden als het er om gaat het probleem-van-de-dag lastig te praten op grond van een vermeende gelijkenis met enig probleem uit de lijst.

Het werkelijk rijke van de theorie die in G&J beschreven wordt ligt in het feit dat het mogelijk is op grond van het eenvoudige berekeningsmodel van Alan Turing, dat bekend staat als de *Turing machine*, uitspraken te doen over het wel of niet bestaan van enig efficiënt algoritme voor een gegeven probleem zonder over die algoritmen zelf te praten, waarbij efficiënt slaat op het feit dat de rekentijd polynomiaal-begrensd is in de grootte van de invoer van het probleem. Tevens heeft dit geleid tot de definitie van twee complexiteitsklassen: de klasse $\mathcal{P}$ bevat problemen die efficiënt oplosbaar en de klasse $\mathcal{NP}$ bevat problemen waarvoor een compact certificaat bestaat dat efficiënt geverifieerd kan worden; en algemeen wordt aangenomen dat deze klassen ongelijk zijn. Voor het gemak noemen we de problemen in $\mathcal{P}$ gemakkelijk en die in $\mathcal{NP}\backslash\mathcal{P}$ NP-lastig.

### Vroege onwikkelingen

Natuurlijk was er voor 1970 ook al veel bekend over de complexiteit van berekeningen en er werd heftig gespeculeerd over het verschil tussen *tractable* en *intractable*, *feasible* en *infeasible* en over de vraag: $\mathcal{P} = \mathcal{NP}$? Zo meldt Hartmanis [1989] dat recentelijk een brief ontdekt is waarin Gödel aan Von Neumann vragen stelt over het bestaan van efficiënte primaliteitstesten en over het vermoeden dat Gödel had dat het in het algemeen mogelijk zou moeten zijn de $N$ stappen die nodig zijn om simpelweg alle mogelijkheden bij een probleem af te gaan – Gödel noemt dit *dem blossen Probieren* – te reduceren tot $\log N$ of $(\log N)^2$ stappen. Helaas is het niet bekend of Von Neumann deze brief beantwoord heeft. Wellicht heeft het feit dat Von Neumann reeds ernstig ziek was tegen de tijd dat hij de brief ontving hem ervan weerhouden in te gaan op de uitdaging waarvoor Gödel hem stelde. Von Neumann ontving de brief minder dan een jaar voordat hij op 8 februari 1957 op 51-jarige leeftijd aan kanker overleed.

Het is het vermelden waard dat Gödel's optimisme in zijn tijd al niet gedeeld werd door een groep Russische cybernetici, die zichzelf ervan overtuigd hadden dat er problemen bestonden die alleen oplosbaar waren door impliciet of expliciet alle mogelijkheden af te gaan; iets wat zij *Perebor* noemden, en dat vrij-vertaald brute kracht betekent. Verder waren zij ook al bezig met het leveren van het bewijs dat Perebor niet verwijderd kon worden bij sommige problemen [Trakhtenbrot, 1984].

### Columns

Na het verschijnen van G&J heeft het gebied van de complexiteit van berekenbaarheid een grote ontwikkeling doorgemaakt en de belangrijkste resultaten zijn terug te vinden in

een tweetal columns. David Johnson heeft het catalogiseren van complexiteitsresultaten voortgezet in wat bekend staat als *The NP-Completeness Column: an Ongoing Guide.* Het betreft hier een column in *The Journal of Algorithms*, waarvan er, sinds de eerste in 1981 werd gepubliceerd, reeds 23 verschenen zijn. De laatste dateert echter al weer uit 1992, maar een hervatting is aangekondigd [Johnson, 1995]. Een tweede column staat onder redactie van Hartmanis en verschijnt sinds 1987 onder de titel *The Structural Complexity Column* met enige regelmaat in *The Bulletin of the European Association of Theoretical Computer Science.* In de column worden historische en recente ontwikkelingen in de complexiteitstheorie gepresenteerd door middel van essays bewerkt door Hartmanis zelf of door anderen op uitnodiging.

**Open problemen**

In de eerste druk van G&J staat een lijst met twaalf open problemen. Daarvan is inmiddels aangetoond dat vier problemen lastig en vier problemen gemakkelijk zijn [Johnson, 1990]. LINEAR PROGRAMMING is wellicht het meest bekende open probleem waarvan bewezen is dat het gemakkelijk is [Khachian, 1979]. Van de resterende vier open problemen zijn GRAPH ISOMORPHISM en PRIMALITY TEST het meest bekend. Van het laatst genoemde probleem bestaat het vermoeden dat het gemakkelijk is, omdat zowel dit probleem als ook het complementaire probleem COMPOSITE NUMBER tot $\mathcal{NP}$ behoren. Dit co-lidmaatschap geeft namelijk aanleiding tot het vermoeden dat een probleem gemakkelijk is.

**Meer dan lastig**

Natuurlijk gaat het er in de toepassing van de complexitietstheorie niet alleen om te bewijzen dat een probleem gemakkelijk of lastig is. Naast de klassen $\mathcal{P}$ en $\mathcal{NP}$ bestaan er inmiddels zo'n tweehonderd complexiteitsklassen die een zinvolle betekenis hebben. In Volume A van Jan van Leeuwens *Handbook of Theoretical Computer Science* geeft Johnson [1989] een catalogus van deze klassen, waarbij van alles aan de orde komt. Het meeste heeft betrekking op de complexiteit van parallelle berekeningen, maar daarnaast worden ook resultaten gepresenteerd die betrekking hebben op randomisatie, benadering en het gemiddelde gedrag van algoritmen.

Fascinerend zijn ook de vele pogingen om het complexiteits-probleem $\mathcal{P} = \mathcal{NP}$? eindelijk eens uit de wereld te helpen. De geleverde bewijzen zijn veelal zeer complex en het is vaak moeilijk „de denkfout" op te sporen. In Column 20 geeft Johnson [1987] een aardig overzicht van de resultaten die over dit onderwerp zijn gepubliceerd.

**Meer belangrijk werk**

Het meest serieuze en belangwekkende werk van de laatste tijd betreft de complexiteitsanalyse van benaderingsproblemen op basis van het werk van Feige & Lovász [1992] aan interactieve bewijssystemen. Een basiselement in de analyse van benaderingsalgoritmen voor NP-lastige optimaliseringsproblemen is de vraag hoever de kosten van de oplossingen verkregen met een benaderingsalgoritme af liggen van de optimale waarde. Deze maat, die bekend staat als de *worst-case ratio*, is voor een aantal problemen bekend. Bijvoorbeeld, voor het handelsreizigersprobleem waarbij de afstanden voldoen aan de driehoeksongelijkheid bestaat er een polynomiaal algoritme met een worst-case ratio van 3/2 [Christofides,

1976]. Lund & Yannakakis [1993] hebben bewezen dat voor sommige problemen zo'n maat niet kan worden begrensd. Meer specifiek laten ze zien dat voor GRAPH COLORING er een $\epsilon > 0$ bestaat zodanig dat GRAPH COLORING niet kan worden benaderd met een worst-case ratio $n^{\epsilon}$, tenzij $\mathcal{P} = \mathcal{NP}$. Anders gezegd betekent dit dat er mag worden aangenomen dat er geen efficiënt algoritme bestaat dat voor iedere willekeurige graaf een kleuring vindt met een aantal kleuren dat een constante factor groter is dan het minimum aantal kleuren dat nodig is om de graaf te kleuren. Ofwel, nog korter: voor GRAPH COLORING is benaderen net zo moeilijk als oplossen.

**Nog meer opwinding**

Een opwindend recent resultaat werd gepubliceerd door Adleman [1994] die met behulp van DNA-moleculen een niet-deterministische Turing machine simuleerde voor een instantie van HAMILTON CYCLE met zeven knopen en zodoende aantoonde dat het mogelijk moet zijn NP-lastige problemen op te lossen in polynomiale tijd onder de voorwaarde dat er genoeg moleculen in het heelal zijn om de benodigde hoeveelheid DNA-strengen te coderen. Het is niet duidelijk waartoe dit resultaat kan leiden, maar Lipton [1995] heeft er uit geconcludeerd dat met dit soort moleculaire computers de *Data Encryption Standard* (DES) gekraakt kan worden. Dit zou een zeer verrassend resultaat zijn, aangezien algemeen wordt aangenomen dat deze coderingstechniek die veel gebruikt wordt om geheime gegevens te versleutelen tegen kraken bestand is.

**Een tweede bekentenis**

Uit het voorafgaande, Frans, moge blijken dat het lenen van jouw boek voor mij belangrijk is geweest. Ik heb er veel van geleerd en het heeft er toe geleid dat ik de ontwikkelingen op dit gebied ben gaan bijhouden. Ik moet echter nog bekennen dat ik je het originele exemplaar dat ik van je te leen kreeg nooit heb teruggegeven. Wat je na ruim twee jaren wachten hebt teruggekregen is een nieuw exemplaar uit de herdruk van 1984. Het verschil tussen beide drukken is de twee pagina's omvattende *Update for the Current Printing*. Daarin staat onder andere dat van de dertien open problemen uit de eerste druk er zes zijn opgelost en dat er grote vooruitgang is geboekt ten aanzien van nog twee andere problemen. Het is onduidelijk waarom er sprake is van dertien problemen, aangezien de eerste druk melding maakt van slechts twaalf problemen. Het exemplaar dat ik van jou te leen had was inmiddels zodanig opgebruikt dat ik het niet met goed fatsoen aan je terug kon geven. Ik beschouw het nu als een van mijn meest waardevolle verzamelaarsexemplaren, waaraan ik veel waarde ben gaan hechten doordat het voor mij een tijd van grote ontwikkeling symboliseert. De essence komt natuurlijk van het feit dat het jouw origineel is.

**Bibliografie**

ADLEMAN, L.M. [1994], Molecular computation of solutions to combinatorial problems, *Science* **266**, 1021–1024.

CHRISTOFIDES, N. [1976], *Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem*, Technical Report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburg, PA.

COOK, S.A. [1971], The complexity of theorem-proving procedures, *Proceedings of the 3rd ACM Symposium on the Theory of Computing*, pp. 151–158.

FEIGE, U., AND L. LOVÁSZ [1992], Two-prover one-round proof systems: their power and their problems. *Proceedings of the 24th ACM Symposium on the Theory of Computing*, pp. 733–744.

GAREY, M.R., AND D.S. JOHNSON [1979], *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, (first printing).

HARTMANIS, J. [1989], The structural complexity column: Gödel, von Neumann and the P =? NP problem, *Bulletin of the European Association of Theoretical Computer Science* **38**, 101–107.

JOHNSON, D.S. [1985], A catalogue of complexity classes, in: J. van Leeuwen (ed.) *Handbook of Theoretical Computer Science, Volume A*, 67–161.

JOHNSON, D.S. [1987], The NP-completeness column: An ongoing guide Column 20: Announcements, updates, and greatest hits, *Journal of Algorithms* **8**, 438–448.

JOHNSON, D.S. [1990], The NP-completeness column: An ongoing guide Column 22: The story so far, *Journal of Algorithms* **11**, 144–151.

JOHNSON, D.S. [1995], Personal communication.

KARP, R.M. [1972], Reducibility among combinatorial problems, in: R.E. Miller and J.W. Thatcher (eds.), *Complexity of Computer Computations*, Plenum Press, New York, 85–103.

KHACHIAN, L.G. [1979], A polynomial algorithm in linear programming, *Soviet Mathematics Doklady* **20**, 191-194.

LIPTON, R.J. [1995], *Breaking DES Using a Molecular Computer*, preprint.

LUND, C., AND M. YANNAKAKIS [1993], On the hardness of approximating maximization problems, *Proceedings of the 25th ACM Symposium on the Theory of Computing*, pp. 286–293.

TRAKHTENBROT, B.A. [1984], A survey of Russian approach to Perebor (brute-force) algorithms, *Annals of the History of Computing* **6**, 384–400.

# Opleiden in het gebruik van formele methoden

Jan van Amstel

## 1 Inleiding

Van de contacten die Frans Kruseman Aretz en ik bij Philips hadden, ging er een aantal over het gebruik van formele methoden door software-ontwikkelaars in de praktijk. Het eerste zakelijke contact van Frans en mij ging over de Cursus Software Ontwerp (CSO). Frans is een van de schrijvers van het rapport ([1]) waarin de CSO gespecificeerd was en ik was verantwoordelijk voor het ontwikkelen en voor het geven van de cursus. In de CSO wordt het gebruik van formele technieken sterk gepropageerd. Bij het ontwikkelen en het onderhouden van de cursus is ook altijd het artikel "Aard en wezen van software"([2]) van Frans als leidraad gebruikt. Frans en ik hebben samen een intern memorandum geschreven over hoe het IST zou moeten omgaan met formele methoden. In dit memorandum komen vragen aan de orde als: Moet Philips een eigen ontwikkeling hebben op het gebied van formele methoden, technieken en talen? Moet het IST een sterke positie hebben op het gebied van talen en gereedschappen rond formele methoden? Wat kan het IST doen om ervoor te zorgen dat formele methoden voor de praktijk van software-ontwikkeling profijtelijk worden? Op de derde plaats wil ik de bijdrage noemen die Frans geleverd heeft bij de ontwikkeling - door de groep Education & Training van het IST - van de cursus "Formeel Specificeren en COLD". In dit artikel wil ik nog eens stil staan bij het gebruik van formele methoden in de praktijk.

## 2 Formele methoden in de praktijk

In 1996 wordt een aantal symposia en workshops georganiseerd waar het gebruik van formele methoden in de praktijk van de software-ontwikkeling centraal staat. In de aankondigingen van deze evenementen wordt gesteld dat na 20 jaar research formele methoden met vrucht in de praktijk gebruikt kunnen worden. Dat is natuurlijk ook zo, denk maar aan programmeertalen, generatoren, enz. Waar het echter om gaat is of formele methoden inzetbaar zijn in een vroeg stadium van het software-ontwikkelproces. In een recent rapport voor de Amerikaanse overheid [3] wordt geconcludeerd dat formele methoden weliswaar in een aantal aspecten de volwassenheid nog niet bereikt hebben, maar dat ze steeds meer in de industrie worden toegepast, serieus en met succes. Wel blijkt er een groot verschil te bestaan tussen datgene waar de industrie behoefte aan zegt te hebben en datgene wat onderzoekers zeggen aan te bieden met formele methoden. Tijdens de IFIP conferentie van 1992 werd in een forum gesteld:

- de vraag vanuit de industrie is: ① kortere ontwikkeltijd, ② evoluerende systemen, ③ hulp bij de constructie van complexe systemen, ④ incrementele uitbreidingen van bestaande methoden, ⑤ betere ontwerpers, ⑥ systeemfamilies en ⑦ garanties met betrekking tot prestaties, betrouwbaarheid, enz.

- het aanbod vanuit de research is: ❶ correcte software, ❷ exacte specificaties, ❸ hulp bij de constructie van complexe algoritmen en datastructuren, ❹ nieuwe methoden voor software-ontwikkeling, ❺ mensen die wiskunde kunnen toepassen in het ontwikkelproces, ❻ goed-gedefinieerde systemen en ❼ garanties met betrekking tot de functionaliteit van systemen.

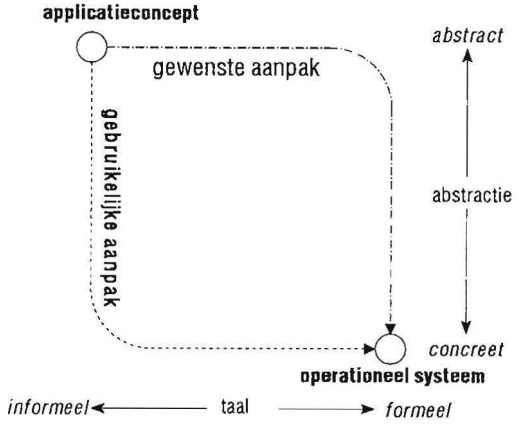
Het is opvallend dat zowel in de vraag als in het aanbod enige referentie ontbreekt aan gereedschap (automatische hulpmiddelen). Dit mag opvallend genoemd worden, omdat in de praktijk het niet beschikbaar zijn van gereedschap vaak gezien wordt als een extra drempel voor het toepassen van formele methoden. Ook in [1] wordt gezegd dat deze extra drempel niet bestaat. Bovendien wordt aangegeven dat het al dan niet beschikbaar zijn van gereedschap niet van belang is voor het succes van het toepassen van formele methoden.

## 3 Gebruik formele methoden: waarom? wanneer? door wie?

**Waarom formele methoden gebruiken?**
In het rapport [1] dat ten grondslag ligt aan de CSO, de cursus die sinds het begin van de tachtiger jaren binnen Philips wordt gegeven voor software-ontwikkelaars, wordt gesteld dat complexiteit een van de meest opvallende eigenschappen van software is. In allerlei vakgebieden worden abstractie en wiskunde gebruikt als hulpmiddelen bij het overwinnen van complexiteit. Waarom zou dat in de software-ontwikkeling anders zijn? In [2] wordt een aantal argumenten voor het gebruik van formele technieken op een rijtje gezet. Zo wordt gesteld dat het resultaat van een softwareproject niet alleen de code is, maar in feite bestaat uit een hele verzameling documenten, die samen bepalend zijn voor de structuur van het ontwikkelproces en van het resulterende softwaresysteem. Al deze documenten betreffen abstracties en formele systemen: abstracties van objecten en relaties uit de werkelijkheid, abstracties van eigenschappen van computersystemen, en formele systemen die theorien en modellen beschrijven. De documenten moeten onderling consistent zijn en de uiteindelijke code moet afgeleid zijn uit deze documenten. Automatisering kan hier behulpzaam zijn. Maar dan moeten de documenten niet alleen gaan over formele systemen, ze moeten zelf ook geformaliseerd zijn, ze moeten een object zijn in een formeel systeem, waarbij de inhoud van en de relaties tussen de documenten moeten voldoen aan zekere formele regels. Formele methoden worden vaak in verband gebracht met kritische toepassingen, zoals bijvoorbeeld systemen voor process control, systemen rond kerncentrales en systemen in vliegtuigen, maar ook systemen rond sociale voorzieningen, systemen in

de medische wereld en systemen in het bankwezen. Het zijn dan eisen als veiligheid, betrouwbaarheid en integriteit die aanleiding geven tot het gebruik van formele methoden. Het effectief en efficiënt bouwen van complexe systemen is op zich echter al reden genoeg om gebruik te maken van formele methoden.

**Wanneer inzetten in ontwikkeltraject?**
In figuur 1 (gebaseerd op [4]) zijn mogelijke wegen voor de ontwikkeling van een softwaresysteem aangegeven. Er zijn twee assen gebruikt: de taal waarin we ons uitdrukken (ïnformeel - formeel") en de abstractie ten opzichte van het computersysteem (äbstract - concreet"). De ontwikkeling, die start met het applicatieconcept (informeel / abstract), moet leiden tot een operationeel systeem (formeel / concreet).




figuur 1

Welke aanpak we ook kiezen om van het applicatieconcept te komen tot een operationeel systeem, er is altijd een moment waarop we ons voor het eerst moeten bedienen van een notatie met een formele syntaxis. Zo is er ook altijd een moment waarop we niet langer kunnen volstaan met een impliciet begrip van het (te realiseren) produkt. Voor veel software-ontwikkelaars vallen de twee overgangen pas samen op het implementatieniveau. Formele specificatietalen geven ons de mogelijkheid om de twee, pragmatisch noodzakelijke stappen in de ontwikkeling van een softwaresysteem (de introductie van een formele notatie en het afstappen van een impliciete betekenis) te laten samenvallen op het specificatieniveau.

**Door wie kunnen formele methoden gebruikt worden?**
Als een software engineer al te maken krijgt met analyse-activiteiten (ten behoeve van domein of applicatie), dan zal dat zijn als een partner van de informatiekundige. Zijn rol zal dan niet zijn dat hij een deskundige is op het terrein van het applicatiegebied (wat

de informatiekundige wel is), maar hij zal de informatiekundige helpen bij het formuleren van de formele specificaties. Dit betekent dat de software engineer moet begrijpen wat het betekent om over te gaan van de informele wereld van het applicatiegebied naar de formele wereld van software-systemen. Een probleem dat optreedt bij de introductie van formele methoden in de praktijk is dat het gebruik veelal beperkt blijft tot een kleine kring van direct betrokkenen. Zo is er in het recente verleden binnen Philips aan gedacht om een speciaal consultancy team op te stellen voor het gebruik van formele methoden. Deze methoden zouden dan alleen bedoeld zijn voor de zeer complexe, bijna onmogelijke problemen en het gebruik ervan zou dan ook tot een kleine elitegroep beperkt blijven. Later vond men dat de expertise niet van buiten moest komen en dat in elke groep van ontwikkelaars één of twee experts op het gebied van formele methoden aanwezig moesten zijn. Beide aanpakken werken echter niet. Formele methoden moeten gebruikt kunnen worden door alle goede software engineers in een ontwikkelteam. Ze kunnen niet bedoeld zijn voor die ene zeer goede ontwikkelaar. Maar het opbouwen van kennis en kunde om met formele methoden te kunnen werken verloopt erg moeizaam binnen de organisaties die experimenteren met formele methoden. Het stelt dan ook eisen aan de methoden met betrekking tot de wijze waarop de software-ontwikkelaars hiermee om moeten kunnen gaan.

## 4 Formele *methoden* ?

In het voorgaande is steeds gesproken over formele methoden. Maar is er wel sprake van methoden? Een methode bestaat uit drie componenten:

- een manier van denken (de "filosofie"; de achterliggende gedachte);
- een manier van werken (welk model en welke aanpak worden gehanteerd);
- een manier van noteren (de gebruikte taal).

Eigenlijk hebben we het, als we spreken van formele methoden, over talen en technieken. Zoals in het verleden het leren programmeren equivalent leek te zijn aan het leren van een programmeertaal (en dit is eigenlijk nog steeds zo, denk maar aan OO en C$^{++}$), zo lijkt het leren gebruiken van formele methoden nu gelijk gesteld te worden aan het leren van een of andere formele specificatietaal. Maar zijn er geen gemeenschappelijke basisbegrippen achter de verschillende formele talen? Van de huidige formele methoden wordt vaak gezegd dat deze zouden moeten aansluiten bij een reeds gehanteerde werkwijze [5]. Dit zou ook de invoering van formele methoden vereenvoudigen. Deze redenering is naar mijn mening niet correct. Een notatie moet passen bij een werkwijze, die op zijn beurt gebaseerd moet zijn op een denkwijze. Laten we de situatie eens vergelijken met "gestructureerd programmeren". De nadruk leggen op notatie heeft daar geleid tot het uitbannen van de 'goto'. Dit ging zover dat men automatische hulpmiddelen bedacht om

een bestaand programma "goto-vrij" te maken. De kern van gestructureerd programmeren, de filosofie, is echter de statische beschouwingswijze van programma's. Dit leidt tot een werkwijze waarbij predikaten worden gebruikt voor de beschrijving van constructies. En dan blijkt er - op het niveau van notatie - geen plaats te zijn voor een 'goto'. Wat is nu het methodische aan de formele methoden? Wat zijn de denkwijze en de werkwijze?

## 5 Formeel en rigoureus

Het kunnen gebruiken van formele methoden betekent het kunnen omgaan met formele systemen. Dit houdt in dat software engineers vertrouwd moeten zijn met de wiskundige theorie en met de wiskundige technieken rond formele systemen (zie [2] en [6]). Deze verschillen van wat gewoonlijk wordt onderwezen in de opleiding van een ingenieur. Bij deze laatste opleiding gaat het onder andere om analyse, Fouriertransformaties en differentiaalvergelijkingen; voor de software engineer zijn onderwerpen als functies en relaties, afbeeldingen, verzamelingen, logica, algebra's, lambda calculus en formele systemen belangrijk. Deze onderwerpen stellen de software engineer in staat om theorieën en modellen te formuleren die van nut zijn bij de specificatie, het ontwerp en de implementatie van softwaresystemen. Er wordt wel onderscheid gemaakt tussen formeel en rigoureus werken. Een formele wijze van werken is een aanpak waarin alle technieken en het gereedschap formeel zijn; dat wil zeggen dat deze een wiskundige inhoud hebben en dat het gebruik ervan ook wiskundig gerechtvaardigd is. De aanpak van software-ontwikkeling is rigoureus als we wel de formalismen gebruiken, maar niet op elk moment alle bewijsverplichtingen nakomen. Deze wijze van werken kan vergeleken worden met de manier waarop een elektrotechnicus de wiskunde gebruikt. Zo zal een elektrotechnicus de volgorde van integreren en sommeren verwisselen, als hem dat goed uitkomt. Een wiskundige zal eerst bewijzen dat de betreffende functie de eigenschappen heeft waaronder deze verwisseling mag plaatsvinden. De software engineer moet met wiskunde kunnen omgaan zoals een elektrotechnicus of een werktuigbouwer dat doen. De software engineer is geïnteresseerd in de wiskunde voor zover het hem in staat stelt effectiever en efficiënter meer betrouwbare systemen te bouwen.

## 6 Opleidingen

Als formele methoden in de praktijk van software-ontwikkeling een rol (gaan) spelen, moeten ontwikkelaars leren hoe ze met deze technieken kunnen omgaan en wat de achterliggende begrippen zijn. Maar de huidige formele methoden leveren wat dit betreft problemen op. Enerzijds doordat bij veel ontwikkelaars de voorkennis ontbreekt, anderzijds doordat deze methoden typisch het produkt zijn van en voor wiskundigen en wiskundig georiënteerde informatici. De terminologie en de notatie zijn afgestemd op de wetenschapper en passen niet bij de denkwijze en de werkwijze van de software-ontwikkelaar. Zo was

het motto van de Tweede Landelijke Specificatie Dag in Eindhoven "Formele Specificatie: Tussen Theorie en Praktijk". Van de deelnemers was, schat ik, meer dan 80 % afkomstig van universiteiten en researchinstituten. Het onderwerp kennistransfer (tussen theorie en praktijk) kwam niet ter sprake. Informatici presenteerden voor elkaar hun werk op het gebied van formele specificaties. En de conferentie van Formal Methods Europe, die als motto had "Industrial Benefit of Formal Methods", was al niet anders. Op de eerder genoemde IFIP-conferentie werd gesteld dat er te weinig onderscheid wordt gemaakt tussen "fundamenten van de software engineering" enerzijds en "formele methoden in de software-ontwikkeling" anderzijds. Concepten uit de wereld van de formele methoden moeten uitgelegd kunnen worden aan de hand van hun gebruik in de softwareconstructie en niet alleen vanuit hun theoretische achtergrond. Als voorbeeld van de slechte aansluiting kunnen we kijken naar het terrein van de algebraïsche specificaties. De keuze tussen initiële en finale semantiek bijvoorbeeld wordt steeds beargumenteerd vanuit de theorie, niet vanuit de software-constructiewereld. En het begrip "conservatieve uitbreiding" wordt met veel theorie omgeven, maar een relatie naar het goed gebruik van inheritance wordt niet gelegd. Steeds weer wordt gezegd dat formele methoden ook voor de praktijk zo belangrijk zijn, maar er gebeurt weinig om ze voor de software-constructeur begrijpelijk en aantrekkelijk te maken. De formele methode zou de software-constructeur in zijn werkzaamheden moeten ondersteunen. Om in de praktijk de acceptatie van formele methoden gemakkelijker te maken, zal aangetoond moeten worden dat het inzetten van deze technieken en talen de produktiviteit en de kwaliteit in de software-ontwikkeling vergroten, zonder dat de kosten significant stijgen. Om dit te bereiken zal er ten minste aandacht moeten worden besteed aan de volgende punten (zie ook [3] en [5]):

- Zorg ervoor dat formele methoden ook in constructieve zin de software-ontwikkelaar van nut kunnen zijn. De methoden moeten de constructeur aanwijzingen geven voor zijn ontwerp.
- Zorg ervoor dat de theorie "verteerbaar" wordt voor de software-ontwikkelaar. Een elektrotechnicus kan leren differentiëren en integreren zonder dat hij zich de achterliggende wiskundige theorie volledig eigen hoeft te maken. Er moet meer effort gestoken worden in het gemakkelijker maken van de overdracht van formele methoden naar een groter publiek.
- Begin nu met het opnemen van de basisprincipes van formele methoden in het onderwijsprogramma voor toekomstige software-constructeurs. Dat zal hen beter voorbereiden op de toekomst en tegelijkertijd de acceptatiegraad van formele methoden vergroten.

## Referenties

[1] R.H. Bourgonjon, F.E.J. Kruseman Aretz, P. Medema, G, van der Munnik, B.L.A. Waumans, A proposal for a programme for software education, Philips, Eindhoven, 1981.

[2] F.E.J. Kruseman Aretz, Aard en wezen van software, in: Informatie, Vol.27 Nr.4 (april 1985).

[3] D. Craigen, S. Gerhart, T. Ralston, An International Survey of Industrial Applications of Formal Methods, Volume 1. Purpose, Approach, Analysis, and Conclusions, U.S. Department of Commerce, Gaithersburg, 1993.

[4] W. Hesse, A systematics of software engineering structure, terminology and classification of techniques in: P. Pepper Program Transformation and Programming Environments Springer, Berlijn, 1984 (NATO ASI Series, Vol. F8).

[5] M.J. Lutz, Formal Methods and the Engineering Paradigm, in: C. Sledge Software Engineering Education (SEI Conference 1992) Springer, Berlijn, 1992 (LNCS 640).

[6] J. Woodcock, M. Loomes, Software Engineering Mathematics, Pitman, Londen, 1988.

# Classification of sorting algorithms by their pattern of recursion

Lex Augusteijn

**Abstract.** Sorting algorithms can be classified in many different ways. The way presented here is by expressing the algorithms as functional programs and to classify them by means of their recursion patterns. These patterns on their turn can be classified as the natural recursion patterns that destruct or construct a given data-type, the so called *cata-* and *anamorphisms* respectively. We show that the selection of the recursion pattern can be seen as the major design decision, in most cases leaving no more room for more decisions in the design of the sorting algorithm. It is also shown that the use of alternative data structures may lead to new sorting algorithms.

## 1 Introduction

In this contribution we present several well known sorting algorithms, namely *insertion sort*, *straight selection sort*, *bubble sort*, *quick sort*, *heap sort* and *merge sort* (see e.g. [Knu73, Wir76]) in a non-standard way. We express the sorting algorithms as functional programs that obey a certain pattern of recursion. We show that for each of the sorting algorithms, the recursion patterns forms the major design decision, often leaving no more space for additional decisions to be taken. We make these recursion patterns explicit in the form of higher-order functions, much like the well-known `map` function on lists.
In order to reason about recursion patterns, we need to formalize that notion. A formalization is already available, based on a category theoretical modeling of recursive data types as can e.g. be found in [Fok92, Mei92]. There is no need to present that theory here. The results that we need can be understood by anyone having some basic knowledge of functional programming, hence we repeat only the main results here. These results show how to each recursive data type a number of morphisms is related, each capturing some pattern of recursion which involve the recursive structure of the data type. Of these morphisms, we use the so called *catamorphism*, *anamorphism*, *hylomorphism* and *paramorphism* on linear lists and binary trees. The value of this approach is not so much in obtaining a nice implementation of some algorithm, but in unravelling its structure.
With respect to sorting, we assume that all sorting operations transform a list `l` into a list `s` that is a permutation of `l`, such that the elements of `s` are ascending w.r.t. to a

total ordering relation <. Moreover, we assume the existence of an equivalence relation == on the elements, such that for all elements a, b, either a<b, a==b or b<a.
We express the sorting algorithms in the functional language Gofer [Jon93], which is a dialect of Haskell [HWA+92]. We assume that the reader is familiar with, but not necessarily an expert in, functional programming.
This paper is organized as follows. In section 2 we present the morphisms on the list data type and show that insertion sort, selection sort and bubble sort can be expressed in terms of these morphisms. In section 3 we present the binary tree data type with its morphisms, which are used to express both quick sort and heap sort. In section 4 we present the leaf tree data type and show how merge sort can be expressed as a morphism over that data type. In section 5 paramorphisms on lists are presented, which can be used to express the recursion pattern of several auxiliary functions used by the different sorting algorithms. We show in section 6 that rose trees form a generalization of lists, binary trees and leaf trees. This fact reveals a novel generalization of quick sort. It also opens the door for a taxonomy of algorithms, based on a hierarchy of data structures and on recursion patterns over those data structures. Section 7 presents the conclusions of this paper.

## 2 Morphisms on lists

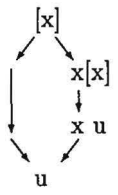
The list data type can be described by the following pseudo data type definition in Haskell.

```
data [x] = []
         | x : [x]
```

In this section we present three recursion patterns over this data type, and show how *insertion sort*, *selection sort* and *bubble sort* can be expressed by means of these recursion patterns.

### 2.1 The list catamorphism

A function of type [x]->u that performs 'natural' recursion over the list structure, consists of two parts, corresponding to the two forms of the list. The first is a part that maps the empty list onto a u. This is just a constant a. A non-empty list can be destructed into a head and a tail. The tail is recursively mapped onto a u and then combined with the head by means of some function f, which forms the second part.
We can present this structure in a diagram where the nodes form the types of the (intermediate) results and the edges the mappings between them.



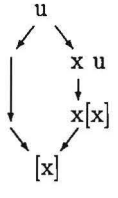

Recursive functions over lists that have this structure can be written by means of a higher order function of `(a,f)` that captures this recursive patterns. As this recursion pattern is generally called *catamorphism* ($\kappa\alpha\tau\alpha$ means downwards), we call this higher order function `list_cata`. As descibed above, it returns `a` on the empty list and applies `f` to the head and the recursive call on the tail.

```
> list_cata :: (u, x -> u -> u) -> [x] -> u
> list_cata (a,f) = cata where
>   cata []     = a
>   cata (x:l) = f x (cata l)
```

It can be observed that this catamorphism replaces the empty list by `a` and the non-empty list constructor `(:)` by `f`. This is what a catamorphism does in general: replacing the constructors of a data type by other functions.

## 2.2 The list anamorphism

Apart from a recursion pattern that traverses a list, we can specify a converse one that *constructs* a list. The structure of this pattern can be obtained by inverting the diagram above.




We need a predicate `p` on `u` that tells us whether to map `u` onto the empty list, or to destruct it, be means of some function `f` into the head of the resulting list `x` and another `u`. This other `u` can then recursively be mapped onto a list, which forms the tail of the final result.
Again, this pattern of recursion can be captured in a higher order function. As this recursion pattern is generally called *anamorphism* ($\alpha\nu\alpha$ means upwards), we call this higher order function `list_ana`.

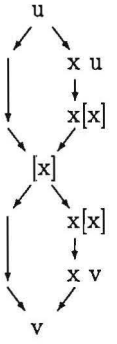
```
> list_ana :: (u -> Bool, u -> (x,u)) -> u -> [x]
> list_ana (p,f) = ana where
>   ana u | p u       = []
>         | otherwise = x : ana l where (x,l) = f u
```

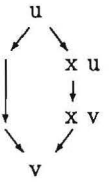
### 2.3 The list hylomorphism

Given a general way to construct a list and to destruct one, we can compose the two operations into a new one, that captures recursion *via* a list. For some philosophical reason, this recursion patterns is called *hylomorphism* ($\acute{\upsilon}\lambda\eta$ means matter). It can be defined as

```
list_hylo (a,f) (p,g) = list_cata (a,f) . list_ana (p,g)
```

This straightforward composition forms an intermediate list, which appears to be unnecessary as the following diagrams exhibits.



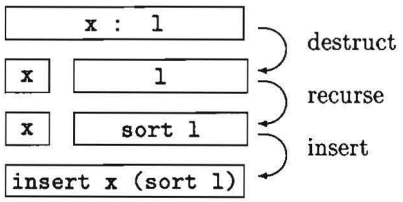

Instead of construction the list in the middle, we can immediately map the `x u` onto the `x v` by recursively applying the hylomorphism.

The implementation is obtained from the `list_ana` by replacing the empty list by `a` and the list constructor `(:)` by `f`.

```
> list_hylo :: (v, x -> v -> v) -> (u -> Bool, u -> (x,u)) -> u -> v
> list_hylo (a,f) (p,g) = hylo where
>   hylo u | p u       = a
>          | otherwise = f x (hylo l) where (x,l) = g u
```

## 2.4 Insertion sort

We can use the `list_cata` recursion pattern to sort a list. In order to analyse the structure of such a sorting algorithm, we first draw a little diagram.



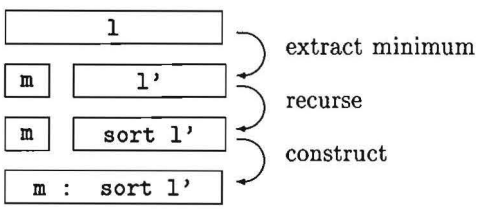

The catamorphism must destruct a list `x:l` into a head `x` and a tail `l`, by means of its recursive structure. Next it is applied recursively to the tail, which in this case means: sorting it. So we are left with a head `x` and a sorted tail. There is only one way left to construct the full sorted list out of these: insert `x` at the appropriate place in the sorted tail. We see that apart from the recursion pattern, we are left with no more design decision: the resulting algorithm is an insertion sort.

```
> insertion_sort l = list_cata ([],insert) l where
>   insert x [] = [x]
>   insert x (a:l) | x < a     = x:a:l
>                  | otherwise = a : insert x l
```

Observe that `insert x` is a recursive function over lists as well. As it does not correspond to the recursive structures introduced so far, we postpone its treatment until section 5.2.

## 2.5 Selection sorts

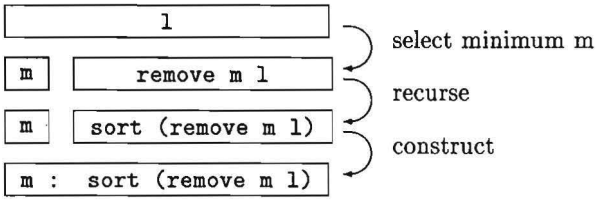
In applying the anamorphism recursion pattern to sorting, we are faced with the following structure.



First of all, the unsorted list `l` is mapped onto the head `m` and a remainder `l'`. This remainder is sorted by recursion and serves as the tail of the final result. From this structure, we can deduce that `m` must be the minimum of `l` and `l'` should equal some permutation of `l` with this minimum removed from it. It is the permutation which gives us some additional design freedom. Two ways to exploit this freedom lead to a straight selection sort and a bubble sort respectively.

### 2.5.1 Straight selection sort

When we first compute the minimum of `l` and then remove it from `l`, maintaining the order of the remaining elements, we obtain a straight selection sort. It has the following recursive structure.



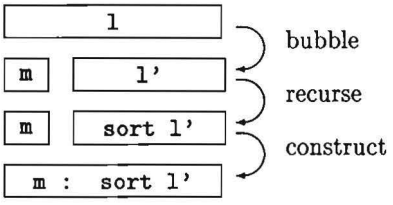

Its implementation as an anamorphism is as follows.

```
> selection_sort l = list_ana (null,select) l where
>   select l = (m, remove m l) where m = minimum l

> remove x [] = []
> remove x (y:l) | x == y    = l
>                | otherwise = y : remove x l
```

Observe that `remove x` is a recursive function over lists as well. As it does not correspond to the recursive structures introduced so far, we postpone its treatment until section 5.3, where we give an implementation of `minimum` as a catamorphism as well.

### 2.5.2 Bubble sort

Selection sort seems a little too expensive as `select` traverses the list twice, once for obtaining the minimum and once for removing it. We can intertwine these to operations to let the minimum 'bubble' to the head of the list by exchanging elements and then split the minimum and the remainder.




```
> bubble_sort l = list_ana (null,select) l where
>   select l = (x,l') where x:l' = bubble l
>   bubble [x]   = [x]
>   bubble (x:l) = if x < y then x:y:l' else y:x:l' where y:l' = bubble l
```

Observe that `bubble` is a recursive function over lists as well. It appears to be a catamorphism, as the following definition shows.

```
> bubble_sort' l = list_ana (null,select) l where
>   select l = (x,l') where x:l' = list_cata ([],bubble) l
>   bubble x [] = [x]
>   bubble x (y:l) = if x < y then x:y:l else y:x:l
```

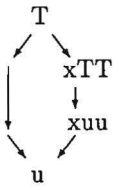
## 3 Binary trees

The sorting algorithms that can be described by list-based recursion patterns all perform linear recursion and as a result behave (at least) quadratically. The $O(n \log n)$ sorting algorithms like quick sort and merge sort use at least two recursive calls per recursion step. In order to express such a recursion pattern we need some binary data structure as a basis for the different morphisms. In this section we concentrate on binary trees with the elements in their branches. The next section treats the so called leaf-trees.
The binary tree data type is defined as follows.

```
> data BinTree x = Tip
>                | Branch x (BinTree x) (BinTree x)
```

### 3.1 The tree catamorphism

The structure of the tree catamorphism is completely analogous to that of the list catamorphism. First destruct the tree, recurse on the sub-trees and combine the element and the recursive results. This corresponds to the following diagram, where T stands for `BinTree x`.

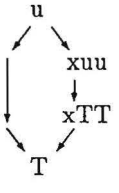



Capturing the recursion pattern in a higher order function `bintree_cata`, gives the following definition.

```
> bintree_cata :: (u, x -> u -> u -> u) -> BinTree x -> u
> bintree_cata (a,f) = cata where
>   cata Tip            = a
>   cata (Branch x l r) = f x (cata l) (cata r)
```

### 3.2 The tree anamorphism

The structure of the tree anamorphism is completely analogous to that of the list anamorphism. First decide by means of a predicate `p` between the tree constructors to be used (`Tip` or `Branch`). In the latter case, destruct into an element and two remaining objects which are recursively mapped onto two trees. Then combine the element and these subtrees. This procedure corresponds to the following diagram.



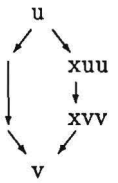

Capturing the recursion pattern in a higher order function `bintree_ana`, gives the following definition.

```
> bintree_ana :: (u -> Bool, u -> (x,u,u)) -> u -> BinTree x
```

```
> bintree_ana (p,f) = ana where
>   ana t | p t       = Tip
>         | otherwise = Branch x (ana l) (ana r) where (x,l,r) = f t
```

### 3.3 The tree hylomorphism

The tree hylomorphism can be obtained by composing the ana- and the catamorphism. Again the binary tree in the middle need not be constructed at all as the following diagram illustrates. We can apply recursion to map the two u's into v's, without constructing the trees.

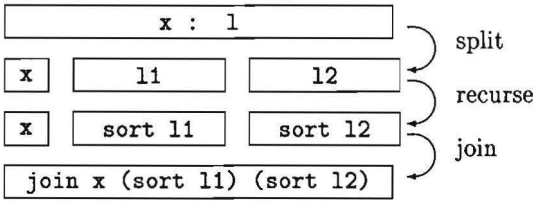



The implementation of the binary tree hylomorphism is obtained from the anamorphism by replacing `Tip` by `a` and the `Branch` constructor by `f`.

```
> bintree_hylo :: (v, x -> v -> v -> v) ->
>                 (u -> Bool, u -> (x,u,u)) ->
>                 u -> v
> bintree_hylo (a,f) (p,g) = hylo where
>   hylo t | p t       = a
>          | otherwise = f x (hylo l) (hylo r) where (x,l,r) = g t
```

### 3.4 Quicksort

We can apply the binary tree hylomorphism recursion pattern to sorting by sorting a list via binary trees (which are never really constructed of course). The following diagram exhibits the recursion pattern.

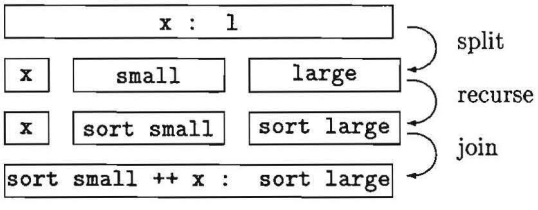


A list `x:l` is split into an element `x` and two other lists `l1` and `l2`. These lists are sorted by recursion. Next `x` and the sorted sub-lists are joined. We are left with a two design decisions here.

- The choice of `x`. Sensible choices here are the head of the list, using the structure of the list, or the minimum or median of the list, exploiting the ordering relation. If we take the minimum, we obtain a variant of heap sort. A derivation of heap sort is given in section 3.5. For quicksort, we choose to take the head of the list. Taking the median is left to the reader.

- The choice of the two sub-lists. An essential choice here is to make them dependent on the element `x` or not. If not, there seems to be no particular reason to separate `x`. If we do not use the head `x` at all, the algorithm obeys a different recursion pattern, which we treat in section 4.
  The remaining option, making the sub-lists depend on `x`, still leaves some choices open. The most natural one seems to let them consists of the elements that are smaller than, respectively at least `x`, exploiting the ordering relation. This can be done for `x` being either the head or the median of the list, where the latter gives a better balanced algorithm with a superior worst case behaviour. We will take the head for simplicity reasons here.

Given the decisions that we take `x` to be head of the list and split the tail into small and large elements w.r.t. to `x`, the only way in which we can combine the sorted sub-lists with `x` is to concatenate them in the proper order.




The final result is an implementation of *quicksort* as a hylomorphism from lists, via trees, onto lists.

```
> quick_sort l = bintree_hylo ([],join) (null,split) l where
>   split (x:l) = (x,s,g) where (s,g) = partition (<x) l
>   join x l r  = l ++ x : r
```

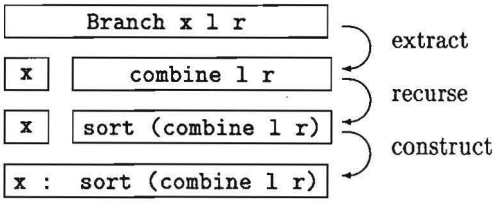
The function `partition` which splits a list into two lists w.r.t. to some predicate p appears to be a list catamorphism:

```
> partition p l = list_cata (([],[]),f) l where
>   f x (a,b) = if p x then (x:a,b) else (a,x:b)
```

### 3.5 Heap sort

In this section we analyse the recursion pattern of heap sort. This algorithm operates by constructing a binary tree that has the so called *heap property*, which means that for such a tree `Branch m l r`, `m` is the minimum element in the tree. The two sub-trees `l` and `r` must also have the heap property, but are not related to each other.

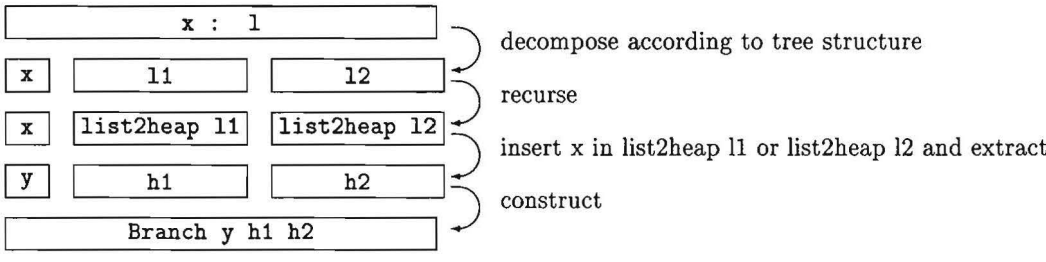
Such a tree is transformed into a sorted list in the following way, where `combine l r` combines two heaps into a new one. It is clearly a list anamorphism.




Thus, heapsort can be implemented in the following way, leaving the function `list2heap`, which transforms an unsorted list into a heap, to be specified. The `b@(Branch x l r)` construction in Gofer matches an argument to the pattern `Branch x l r` as usual, but also binds the argument as a whole to `B`.

```
> tip Tip = True
> tip _   = False

> combine :: Ord a => BinTree a -> BinTree a -> BinTree a
> combine t Tip   = t
> combine Tip t   = t
> combine b@(Branch x l r) c@(Branch y s t)
>     | x < y     = Branch x l (combine r c)
>     | otherwise = Branch y (combine b s) t

> heap_sort l = (list_ana (tip,extract) . list2heap) l where
>   extract (Branch x l r) = (x, combine l r)
```

Three recursion patterns that could be applicable to the function `list2heap` are a list catamorphism, a tree anamorphism, or a hylomorphism over some additional data type. Let us analyse the recursion pattern of such a function, where we assume that we use `list2heap` recursively in a binary tree pattern (note that this a design decision).




From this diagram, it can be seen that the step before the recursive application is not a tree destruction, and the step behind it is not a tree construction (although it contains one). Hence, we must conclude that it is a binary tree hylomorphism [1].

The decomposition is simple: just split the tail into two halves and apply recurion to obtain two heaps `l` and `r` from them. Inserting `x` in either of the two sub-heaps `l` and `r` goes as follows. Optionally swap the heaps, resulting in `h3` and `h2` respectively, such that `h3` contains the smallest top element of the two. Insert `x` into `h3`, resulting in a new heap `h1`, and an element `y` that is smaller than all the elements in both `h1` and `h2`.

```
> list2heap l = bintree_hylo (Tip, heaps_insert) (null,decompose) l where
>   decompose (x:l) = (x,l1,l2) where (l1,l2) = split l

> heaps_insert :: Ord a => a -> BinTree a -> BinTree a -> BinTree a
> heaps_insert x l r = Branch y h1 h2 where
>  (h3,h2) = swap_tree l r
>  (y,h1)  = heap_insert x h3

> heap_insert :: Ord a => a -> BinTree a -> (a,BinTree a)
> heap_insert x (Branch y l r) | y < x = (y, heaps_insert x l r)
> heap_insert x b = (x,b)

> swap_tree b@(Branch x _ _) c@(Branch y _ _)
>   | x < y       = (b,c)
>   | otherwise   = (c,b)
> swap_tree Tip b = (b,Tip)
```

[1] We leave it as an excercise to the reader to show that it is also a list catamorphism.

```
> swap_tree b c    = (b,c)
```

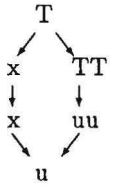
## 4 Leaf trees

Another form of binary trees are so-called *leaf-trees*. These trees hold their elements on their leaves instead of their branches. The leaf-tree data type is given by:

```
> data LeafTree x = Leaf x
>                 | Split (LeafTree x) (LeafTree x)
```

### 4.1 The leaf-tree catamorphism

The leaf-tree catamorphism is a bit more interesting than the previously presented ones, since it needs a function on the element, rather than a constant, to construct its non-recursive result. The recursion pattern diagram is:



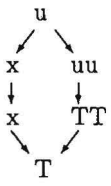

Capturing the recursion pattern in a higher order function `leaftree_cata`, gives the following definition (again, just replace the tree constructors `Leaf` and `Split` by other functions, `fl` and `fs` respectively).

```
> leaftree_cata :: (x -> u, u -> u -> u) -> LeafTree x -> u
> leaftree_cata (fl,fs) = cata where
>   cata (Leaf x)    = fl x
>   cata (Split l r) = fs (cata l) (cata r)
```

### 4.2 The leaf-tree anamorphism

The leaf-tree catamorphism is also a bit more interesting than the previously presented ones, since the non-recursive branch needs a destructor function. Hence, we must parameterize the `leaftree_ana` function with a triple `(p,fl,fs)` where `p` is a predicate that

chooses between applying the `Leaf` or `Split` constructor, `fl` and `fs` are the destructors to be applied to these two cases respectively.
The diagram is obtained by reversing the previous one.

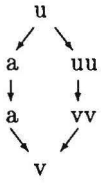



Capturing the recursion pattern in a higher order function `leaftree_ana`, gives the following definition.

```
> leaftree_ana :: (u -> Bool, u -> x, u -> (u,u)) -> u -> LeafTree x
> leaftree_ana (p,fl,fs) = ana where
>   ana t | p t       = Leaf (fl t)
>         | otherwise = Split (ana l) (ana r) where (l,r) = fs t
```

### 4.3 The leaf-tree hylomorphism

The leaf-tree hylomorphism should be straightforward now. We present only its diagram

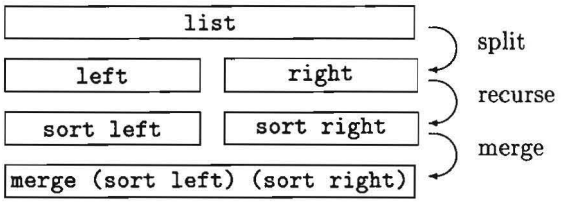



and its implementation with no further comment:

```
> leaftree_hylo :: (x -> v, v -> v -> v) ->
>                  (u -> Bool, u -> x, u -> (u,u)) ->
>                  u -> v
> leaftree_hylo (fl,fs) (p,gl,gs) = hylo where
>   hylo t | p t       = fl (gl t)
>          | otherwise = fs (hylo l) (hylo r) where (l,r) = gs t
```

### 4.4 Merge sort

The leaf-tree hylomorphism can be used to sort lists via leaf-trees. The recursion pattern can be depicted as follows.




In the recursive case, the list is split into two sub-list, which are sorted, and then combined. The main choice left here is to make the sub-lists dependent or independent of the elements of the other sub-lists.
When we assume independence, the combination of the recursive results must merge two unrelated sorted lists, and we obtain merge sort.
The choice of two sub-lists which are dependent on each other does not give us much. If we assume that we can only apply an ordering and an equality relation to the elements, we can't do much more than separating the elements into small and large ones, possibly w.r.t. to the median of the list (which would yield quicksort). We do not pursue this way of sorting any further here.
The implementation of merge sort as an hylomorphism from lists, via leaf-trees, onto lists is given below. The non-recursive case deals with lists of one element. The empty list is treated separately from the hylomorphism. Observe the definition of `split` as a list catamorphism.

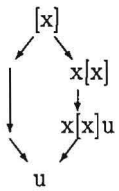
```
> merge_sort [] = []
> merge_sort l  = leaftree_hylo (single,merge) (is_single,elem,split) l
>  where
>   single x = [x]
>   merge (x:xs) (y:ys) | x < y     = x : merge xs (y:ys)
>                       | otherwise = y : merge (x:xs) ys
>   merge [] m = m
>   merge l [] = l
>   is_single (x:l) = null l
>   elem [x] = x

> split = list_cata (([],[]),f) where
>   f x (l,r) = (r,x:l)
```

# 5 Paramorphisms on lists

Several sub-functions, like `insert` and `remove` used above where almost catamorphisms on lists. They deviate by the fact that in the construction of the result, they do not only use the recursive result of the tail, but also the tail itself. This recursion pattern is known as *paramorphism*, after the Greek word $\pi\alpha\rho\alpha$, which among other things means 'parallel with'.
Paramorphisms can be expressed as catamorphisms by letting the recursive call return its intended result, tupled with its argument. Here, we study them as a separate recursion pattern however.

## 5.1 The list paramorphism

The list paramorphism follows the recursion patterns of the following diagram.

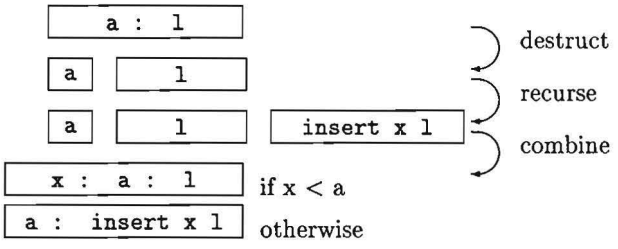



Its implementation is straight-forward, just supply the tail `l` as an additional argument to the constructor function `f`.

```
> list_para :: (a, b -> [b] -> a -> a) -> [b] -> a
> list_para (a,f) = para where
>   para []     = a
>   para (x:l) = f x l (para l)
```

## 5.2 Insert as paramorphism

The insertion operation `insert x` of insertion sort can be expressed as a paramorphism. First we analyse its recursive structure.

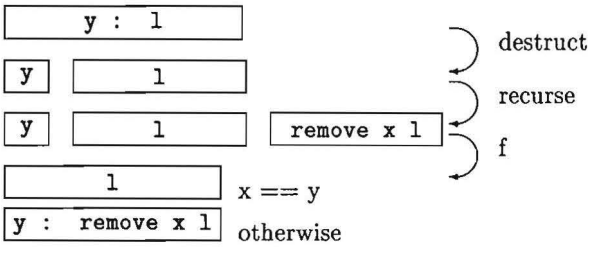



The list `a:l` is split into head `a` and tail `l`. Recursively, `x` is inserted into `l`. Depending on where `x < a`, we need to use the original tail, or the recursive result.
Although it may seem inefficient to construct the recursive result and then optionally throw it away again, laziness comes to help here. If the recursive result is not used, it is simply not computed.

```
> insertion_sort' l = list_cata ([],insert) l where
>   insert x = list_para ([x],combine) where
>     combine a l rec | x < a     = x : a : l
>                     | otherwise = a : rec
```

### 5.3 Remove as paramorphism

The selection operation `remove x` of the straight selection sort can be expressed as a paramorphism as well. First we analyse its recursive structure.




It destructs the list into the head `y` and tail `l`. It recursively removes `x` from `l`. Next, it chooses between using the non-recursive tail `l` (when `x == y`, it suffices to remove `y`), or, in the other case, to maintain `y` and use the recursive result.

Below, we give the paramorphic version of the algorithm. Observe that `minimum` has been written as a catamorphism.

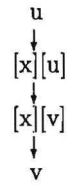
```
> selection_sort' l = list_ana (null,select) l where
>   select l = (m, remove m l) where m = minimum l
>   minimum (x:l) = list_cata (x,min) l
>   remove x = list_para ([],f) where
>     f y l rec | x == y    = l
>               | otherwise = y : rec
```

## 6 Generalizing data structures

The previous sections have shown that the use of patterns of recursion gives rise to a classification of sorting algorithms. One can obtain a refined taxonomy of sorting algorithms by introducing yet another level of generalization. This extra level is the generalization of data types. We illustrate this by generalizing the binary tree data type to the rose tree data type.

```
> data RoseTree a = RoseTree [a] [RoseTree a]
```

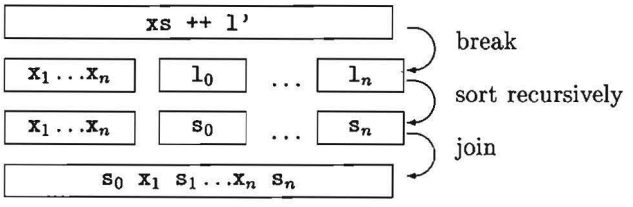
A binary tree has one element and two branches, a rose tree $n$ elements and $m$ branches. The empty rose tree is represented by the $m = n = 0$ case. Since quicksort is a hylomorphism on binary trees, a hylomorphism on rose trees is expected to be a generalization of quicksort. The rose tree hylomorphism is given by the following diagram and definition.




```
> rosetree_hylo :: ([x] -> [v] -> v) -> (u -> ([x],[u])) -> u -> v
> rosetree_hylo f g = hylo where
>   hylo t = f x (map hylo l) where (x,l) = g t
```

The generalization of quicksort can be obtained by using $n$ pivots, rather than 1. These $n$ pivots are sorted, e.g. by `insertion_sort` for small $n$, and the remaining elements are grouped into $n + 1$ intervals w.r.t. the pivots. I.e. the first internal contains all elements less than the first pivot, the second internal contains all remaining elements less than the second pivot, etc., while the $n + 1$-th interval contains all elements not less than the

$n$-th pivot. These intervals are sorted recursively, and the intervals and pivots are joined together. The following diagram illustrates this process.




The implementation is relatively straight-forward after this design. The `l==[]` case needs to be treated specially to ensure termination. First, `l` is split into its first (at most) $n$ elements `xs` and the remainder `l'`. Next `xs` is sorted to obtain `sx`. Then `l'` is split w.r.t. to `sx`.

```
> rose_sort n l = rosetree_hylo join break l where
>   join x []     = x
>   join x (s:l) = s++concat (zipWith (:) x l)
>   break [] = ([],[])
>   break l  = (sx,split sx l') where
>     (xs,l') = take_drop n l
>     sx = insertion_sort xs
>     split p l = list_cata ([l],f) p where
>       f x (a:l) = s:g:l where (s,g) = partition (<x) a

> take_drop 0 l = ([],l)
> take_drop n [] = ([],[])
> take_drop n (x:l) = (x:a,b) where (a,b) = take_drop (n-1) l
```

Experiments show that this algorithm behaves superiour to the `quick_sort` function when applied to random list of various sizes. The optimal value of `n` appears to be independent of the length of the list (it equals 3 in this implementation). A decent analysis of the complexity of this algorithm should reveal why this is the case. The split size can be adapted to the length of the list $L$ by replacing `n` by some function of $L$. It is an open problem which function will give the best behaviour.

Since rose trees can be viewed as a generalization of linear lists, binary trees and leaf trees together, the other sorting algorithms generalize as well. E.g. the two-way merge sort becomes a $k$-way merge sort. We leave these generalizations as exercises to the reader.

## 7 Conclusions

We have shown that it is possible to express the most well-known sorting algorithms as functional programs, using fixed patterns of recursion. Given such a pattern of recursion there is little or no additional design freedom left. The approach shows that functional programming in general and the study of recursion patterns in particular form a powerful tool in both the characterization and derivation of algorithms. In this paper we studied the three data types linear lists, binary trees and binary leaf trees. Generalizing these data types to rose trees revealed a generalization of quick-sort, which is, as far as the author knows, a novel sorting algorithm. It may well be that other data types, and their corresponding recursion patterns, can be used to derive even more sorting algorithms.

By distinguishing a hierarchy of data structures on the one hand, and different patterns of recursions on the other hand, a taxonomy of algorithms can be constructed. It would be nice to compare this with other techniques for constructing algorithm taxonomies as e.g. presented in [Wat95].

Of course, other algorithms than sorting can be classified by means of their pattern of recursion. See e.g. [Aug93], where similar techniques were used to characterize parsing algorithms. The subject of deriving and classifying algorithms by means of their recursion pattern is just skimmed by this paper.

The question whether the presentation of the algorithms as such is clarified by their expression in terms of morphisms has not been raised yet. When we compare the catamorphic version of insertion sort with the following straight implementation, the latter should be appreciated over the first.

```
insertion_sort []     = []
insertion_sort (x:l) = insert x (insertion_sort l) where
  insert x [] = [x]
  insert x (a:l) | x < a      = x :a:l
                 | otherwise = a : insert x l
```

The value of this approach is not so much in obtaining a nice presentation or implementation of some algorithm, but in unravelling its structure. Especially in the case of heap sort, this approach gives a very good insight in the structure of the algorithm, compared for instance to [Knu73] or [Wir76].

**Acknowledgements**
Above all, I wish to thank Frans Kruseman Aretz for the patient and careful supervision during the writing of my thesis, of which this contribution is a natural continuation. Also the discussions about many different subjects and the pleasant time when we were roommates give me pleasant memories. I am also grateful to Doaitse Swierstra and Tanja Vos for stimulating discussions about the different morphisms, to Erik Meijer for his illuminating thesis and to Herman ter Horst for his refereeing.

## References

[Aug93] Lex Augusteijn. *Functional Programming, Program Transformations and Compiler Construction.* PhD thesis, Eindhoven Technical University, October 1993.

[Fok92] Maarten M. Fokkinga. *Law and order in algorithmics.* PhD thesis, Twente University, 1992.

[HWA+92] Paul Hudak, Philip Wadler, Arvind, Brian Boutel, Jon Fairbairn, Joseph Fasel, Kevin Hammond, John Hughes, Thomas Johnsson, Dick Kieburtz, Rishiyur Nikhil, Simon Peyton Jones, Mike Reeve, David Wise, and Jonathan Young. Report on the Programming Language Haskell, A Non-Strict, Purely Functional Language, Version 1.2. *ACM SIGPLAN Notices*, 27(5):Section R, 1992.

[Jon93] Mark P. Jones. *Release notes for Gofer 2.28.*, February 1993. Included as part of the standard Gofer distribution, `http://www.cs.nott.ac.uk:80/Department/Staff/mpj/`.

[Knu73] Donald E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, 1973. Sorting and Searching.

[Mei92] Erik Meijer. *Calculating Compilers.* PhD thesis, Utrecht State University, Utrecht, the Netherlands, 1992.

[Wat95] Bruce W. Watson. *Taxonomies and Toolkits of Regular Language Algorithms.* PhD thesis, Eindhoven University of Technology, 1995.

[Wir76] Niklaus Wirth. *Algorithms + Data Structures = Programs.* Prentice Hall, 1976.

# The Schröder-Bernstein theorem

Roland Backhouse

Mathematics contains many theorems whose proof involves a non-trivial programming problem. The goal of this paper is to demonstrate the use of the calculational method in the *construction* of a proof of such a theorem. The theorem (which seems to be known by various names) is as follows.

> Let $X$ and $Y$ be sets and suppose there exist one-to-one maps $f \in X \rightarrow Y$ and $g \in Y \rightarrow X$ . Then there exists a bijective map $h$ from $X$ onto $Y$ .

Let us first note that constructing a proof of the theorem is indeed a programming problem. We are given two functions with particular properties and we are required to construct a third satisfying a particular property. Computer programmers should be well equipped to solve this problem. All proofs I have seen, however, begin by postulating the value of $h$ (sometimes via a lemma called "Banach Decomposition" fished out of the blue) and then verify that it meets the required conditions.

In order to construct $h$ we need to begin by expressing the problem in terms of a suitable calculus. Relation algebra suggests itself because the four properties of being one-to-one, functional, onto and total are expressed so concisely and symmetrically in the algebra. Specifically, the problem is expressed in relation algebra in the following way. Given are two relations $f$ and $g$. Both are functional and injective, and the range of the one is contained in the domain of the other. That is[1],

(1) $\quad f^{\cup} \circ f = \mathsf{rng}.f \subseteq \mathsf{dom}.g = g \circ g^{\cup}$

and

(2) $\quad g^{\cup} \circ g = \mathsf{rng}.g \subseteq \mathsf{dom}.f = f \circ f^{\cup}$ .

(The condition $f^{\cup} \circ f = \mathsf{rng}.f$ expresses that $f$ is functional —from left to right—, the condition $\mathsf{dom}.g = g \circ g^{\cup}$ that $g$ is injective —i.e. functional from right to left—.)

Required is to construct two relations $h$ and $k$ such that

(3) $\quad h = k^{\cup}$

[1] See the appendix for a summary of the notation used.

(4) $h \circ k = \mathsf{dom}.f$

(5) $k \circ h = \mathsf{dom}.g$ .

This then is the problem. Apart from reformulating it in the algebra of relations I have also introduced the function $k$ for the simple reason that by doing so the symmetry between $f$ and $g$ remains intact.

Using (2) we can rewrite (4) in the form

(6) $h \circ k = f \circ f^{\cup}$ .

Similarly, (5) can be rewritten in the form

(7) $k \circ h = g \circ g^{\cup}$ .

The assignments $h := f$ and $k := f^{\cup}$ satisfy (3) and (6) but not necessarily (7). Dually, the assignments $k := g$ and $h := g^{\cup}$ satisfy (3) and (7) but not necessarily (6). The solution would thus seem to be to assign to $h$ some combination of $f$ and $g^{\cup}$, and to $k$ some combination of $f^{\cup}$ and $g$. A well-trained programmer would immediately think of a conditional statement as the obvious way to "combine" the two functions. Let us therefore introduce guards $A$ and $B$ and postulate

(8) $h = A \circ f \cup \neg A \circ g^{\cup}$

and

(9) $k = B \circ g \cup \neg B \circ f^{\cup}$ .

(Note that the form of these postulates has been carefully chosen so as to retain the symmetry in the problem.)

Now we try to solve (3) through (9). We begin with (3).

$$
\begin{array}{ll}
 & h = k^{\cup} \\
\equiv & \quad \{ \quad \text{(8) and (9), converse} \quad \} \\
 & A \circ f \cup \neg A \circ g^{\cup} \;=\; g^{\cup} \circ B \cup f \circ \neg B \\
\Leftarrow & \quad \{ \quad \text{Leibniz and commutativity of } \cup \quad \} \\
 & A \circ f = f \circ \neg B \;\wedge\; \neg A \circ g^{\cup} = g^{\cup} \circ B \\
\equiv & \quad \{ \quad \text{converse} \quad \} \\
 & A \circ f = f \circ \neg B \;\wedge\; B \circ g = g \circ \neg A \quad .
\end{array}
$$

Thus we demand

(10) $A \circ f = f \circ \neg B$

and

(11) $B \circ g = g \circ \neg A$ .

The follows-from step above may seem to be very coarse, but it is justified by the fact that the only known relationship between $f$ and $g$ is the relationship between their domains.

In the next step we use some elementary domain calculus with goal the creation of an equation with a bare "$A$" or a bare "$B$" on one side.

$$
\begin{array}{ll}
 & A \circ f = f \circ \neg B \\
\equiv & \quad \{ \quad \text{domain translation (exploiting } \mathsf{dom}.f = f \circ f\cup) \quad \} \\
 & f \circ \mathsf{rng}.(A \circ f) = f \circ \neg B \\
\Leftarrow & \quad \{ \quad \text{Leibniz} \quad \} \\
 & \mathsf{rng}.(A \circ f) = \neg B \\
\equiv & \quad \{ \quad \text{negation is an involution} \quad \} \\
 & \neg(\mathsf{rng}.(A \circ f)) = B \ .
\end{array}
$$

Symmetrically,

$$B \circ g = g \circ \neg A \Leftarrow \neg(\mathsf{rng}.(B \circ g)) = A \ .$$

We therefore replace (10) and (11) by

(12) $\neg(\mathsf{rng}.(A \circ f)) = B$

(13) $\neg(\mathsf{rng}.(B \circ g)) = A$ .

The key step is to observe that (12) and (13) do have (simultaneous) solutions in the unknowns $A$ and $B$. Specifically, by eliminating $B$ we obtain the requirement on $A$

$$\neg(\mathsf{rng}.(\neg(\mathsf{rng}.(A \circ f)) \circ g)) = A \ .$$

But the function $A \mapsto \neg(\mathsf{rng}.(\neg(\mathsf{rng}.(A \circ f)) \circ g))$ is monotonic (since rng is monotonic and $\neg$ is anti-monotonic). Moreover, the relations (of a given type) form a complete lattice under the usual subset ordering. The function thus has a least fixed point – courtesy of the Knaster-Tarski theorem. Taking this fixed point as the solution for $A$ and substituting this in (12), we obtain a solution to the two requirements (12) and (13). (The rolling rule should be used to verify that this is indeed the case.)

It remains to see whether (12) and (13) automatically guarantee (4) and (5). By symmetry it is sufficient to check (4). Recalling that (12) and (13) are implied by (10) and (11), we have:

$$
\begin{array}{ll}
 & h \circ k \\
= & \quad \{ \quad \text{(8) and (9)} \quad \} \\
 & (A \circ f \cup \neg A \circ g^{\cup}) \circ (\neg B \circ f^{\cup} \cup B \circ g) \\
= & \quad \{ \quad \text{distributivity} \quad \} \\
 & A \circ f \circ \neg B \circ f^{\cup} \cup A \circ f \circ B \circ g \cup \neg A \circ g^{\cup} \circ \neg B \circ f^{\cup} \cup \neg A \circ g^{\cup} \circ B \circ g \\
= & \quad \{ \quad \text{(10) and (11)} \quad \} \\
 & A \circ f \circ (A \circ f)^{\cup} \cup f \circ \neg B \circ B \circ g \cup g^{\cup} \circ B \circ \neg B \circ f^{\cup} \cup (B \circ g)^{\cup} \circ B \circ g \\
= & \quad \{ \quad \text{domain calculus and (2); guards;} \\
 & \qquad \text{guards; range calculus and (2)} \quad \} \\
 & \mathsf{dom}.(A \circ f) \cup \perp\!\!\!\perp \cup \perp\!\!\!\perp \cup \mathsf{rng}.(B \circ g) \\
= & \quad \{ \quad \text{calculus} \quad \} \\
 & \mathsf{dom}.(A \circ f) \cup \mathsf{rng}.(B \circ g) \\
= & \quad \{ \quad \text{domain calculus} \quad \} \\
 & A \circ \mathsf{dom}.f \cup \mathsf{rng}.(B \circ g) \\
= & \quad \{ \quad \mathsf{rng}.(B \circ g) \subseteq \{ \text{ monotonicity } \} \ \mathsf{rng}.g \subseteq \{ \text{ (2) } \} \ \mathsf{dom}.f \quad \} \\
 & A \circ \mathsf{dom}.f \cup \mathsf{rng}.(B \circ g) \circ \mathsf{dom}.f \\
= & \quad \{ \quad \text{(13) and calculus} \quad \} \\
 & \mathsf{dom}.f \ .
\end{array}
$$

This completes the proof. Compared to proofs in mathematical texts it may seem relatively long. But, since verification is always easier than construction, that was to be expected. The final calculation, although the longest of all, is the most straightforward. (One might even contemplate feeding it to a computer.) The two critical steps in the construction of $h$ are the use of Leibniz in the second step (a weakening is always critical and deserves careful thought) and the subsequent use of domain translation to replace

$A \circ f$ by $f \circ \mathsf{rng}.(A \circ f)$. The latter is a rule that few would care to commit to memory. But there is no need to do so: at that point in the calculation it is necessary to shift "$f$" either from the left to the right of a composition or, vice-versa, from the right to the left. And, as we all know, necessity is the mother of invention!

## Appendix

This appendix summarises the notation used in the paper.

A relation is a set of pairs. The converse of relation $R$, denoted $R^{\cup}$ is the set of pairs $(x, y)$ such that the pair $(y, x)$ is an element of $R$. The composition $R \circ S$ of relations $R$ and $S$ is the set of pairs $(x, z)$ such that, for some $y$, the pair $(x, y)$ is an element of $R$ and $(y, z)$ is an element of $S$. The union $R \cup S$ of two relations is the usual union of two sets and $\perp\!\!\!\perp$ denotes the empty set.

A *guard* is a subset of the identity relation. The negation of guard $A$, denoted $\neg A$, is the guard containing exactly those pairs $(x, x)$ that are not elements of $A$. The domain and range of a relation are defined to be guards; $\mathsf{dom}.R$ is the set of pairs $(x, x)$ such that $(x, y)$ is an element of $R$ for some $y$, and $\mathsf{rng}.R$ is the set of pairs $(y, y)$ such that $(x, y)$ is an element of $R$ for some $x$.

# Some simple calculations in relative discrete time process algebra

J.C.M. Baeten and J.A. Bergstra

**Abstract**

We do some simple calculations involving buffers in discrete time process algebra with relative timing.

*Note:* Dedicated to prof.dr. F.E.J. Kruseman Aretz, on the occasion of his 'afscheidscollege'.

## 1 Introduction

Any formal language should enable formal analysis. As Kruseman Aretz already emphasised in [6], it is essential to do precise calculations, perform mathematics, with a formal language, and a well-designed language should facilitate such calculations. In this note, we do some simple calculations in the formal language of relative discrete time process algebra.

The process algebra ACP [3] describes the main features of imperative concurrent programming without explicit mention of time. Implicitly, time is present in the interpretation of sequential composition: in $p \cdot q$ the process $p$ must be executed before $q$. A quantative view on the relation between process execution and progress of time is absent in ACP, however. In recent years, process algebras have been developed that provide standardised features to incorporate a quantative view on time. On the one hand, we can represent time by means of non-negative reals, and have time stamps on actions. On the other hand, we can divide time in slices indexed by natural numbers, have an implicit or explicit time stamping mechanism that provides each action with the time slice in which it occurs and have a time order within each time slice only. We use the phrase *discrete time process algebra* if an enumeration of time slices is used.

In [1] ACP was extended to a discrete time process algebra. Here, we use discrete time process algebra with *relative timing*, where timing refers to the execution of the previous action. We use the so-called *two-phase version*, where the passage of time and the execution of actions is separated. Another version of discrete time process algebra uses *absolute timing*, where all timing refers to an absolute clock. Finally, we have discrete process algebra with *parametric timing*, where absolute and relative timing are integrated (see [1]).

## 2 Discrete Time Process Algebra with Relative Timing

We present axioms for discrete time process algebra with relative timing. We base ourselves on [1], but use the slightly optimised presentation of [2].

### 2.1 Basic Process Algebra

We assume we have given a (finite) set of atomic actions $A$. This set is a parameter of the theory to be presented.

With $\underline{a}$ we denote the process that will execute atomic action $a$ in the current time slice, the time slice in which it is initialised. So, if $\underline{a}$ is enabled during slice 7, then $a$ will be performed in the course of time slice 7. With the operator $\sigma_{\mathrm{rel}}$ processes can be delayed one time slice. So if the process $\sigma_{\mathrm{rel}}(\sigma_{\mathrm{rel}}(\underline{a}))$ is initialised in slice 5, then $a$ will be executed in slice 7.

The signature of $\mathrm{BPA}_{\mathrm{drt}}$ has constants $\underline{a}$ (for $a \in A$), denoting $a$ in the current time slice, and $\underline{\delta}$, denoting a deadlock at the end of the current time slice. Furthermore, we have the immediate deadlock constant $\dot{\delta}$, denoting immediate and catastrophic deadlock. The operators are alternative ($+$) and sequential composition ($\cdot$), and the *relative discrete time unit delay* $\sigma_{\mathrm{rel}}$. The process $\sigma_{\mathrm{rel}}(x)$ will start after one clock tick, i.e. in the next time slice. Note that the operator $\sigma_{\mathrm{rel}}$ is a constructor: the term $\sigma_{\mathrm{rel}}(\underline{a})$ cannot be reduced. Furthermore, we have the auxiliary operator *current time slice time out* $\nu_{\mathrm{rel}}$. The current time slice time out operator disallows an initial time step, it gives the part of a process that starts with an action in the current time slice. This operator will prove useful when we extend $\mathrm{BPA}_{\mathrm{drt}}$ with parallel composition in the next subsection.

Within a time slice, there is no explicit mention of the passage of time, we can see the passage to the next time slice as a clock tick. Thus, the $\underline{a}$ can be called *non-delayable* actions: the action must occur before the next clock tick. In order to add delayable actions to the theory, we have the operator $\lfloor\ \rfloor^{\omega}$, the *unbounded start delay* operator: $\lfloor x \rfloor^{\omega}$ can start the execution of $x$ in the current time slice, or delay unchanged to the next time slice. The defining axiom for this operator takes the form of a recursive equation. The unbounded start delay operator is defined easily in terms of the current time slice time out operator. A delayable action $a$ (atomic action $a$ in any time slice) is then defined as the unbounded start delay of the non-delayable action $\underline{a}$ (axiom ATS). In order to be able to work with recursive equations, e.g. if we want to prove identities concerning the unbounded start delay operator, we need to extend the algebra with a proof principle. We will look at such a principle next.

The axioms of $\mathrm{BPA}_{\mathrm{drt}}$ are given in Table 1. The axiom DRT1 is the *time factorization* axiom: it says that the passage of time by itself cannot determine a choice. However, it is possible that passage of time disables the execution of an action $a$. This is illustrated by the example $\underline{a} + \sigma_{\mathrm{rel}}(\underline{b})$. If a clock tick occurs the process evolves into $\underline{b}$. Thereby, it has

become impossible to execute the action $a$: it is disabled by the occurrence of the clock tick. The axiom DCS3 for the current time slice time out operator clearly expresses that all alternatives which cannot perform an action in the current time slice are replaced by $\underline{\underline{\delta}}$.

| | | | |
|---|---|---|---|
| $x+y=y+x$ | A1 | $\sigma_{\mathrm{rel}}(x)+\sigma_{\mathrm{rel}}(y)=\sigma_{\mathrm{rel}}(x+y)$ | DRT1 |
| $(x+y)+z=x+(y+z)$ | A2 | $\sigma_{\mathrm{rel}}(x)\cdot y=\sigma_{\mathrm{rel}}(x\cdot y)$ | DRT2 |
| $x+x=x$ | A3 | $\sigma_{\mathrm{rel}}(\dot{\delta})=\underline{\underline{\delta}}$ | DRT3 |
| $(x+y)\cdot z=x\cdot z+y\cdot z$ | A4 | $\underline{\underline{a}}+\underline{\underline{\delta}}=\underline{\underline{a}}$ | DRT4 |
| $(x\cdot y)\cdot z=x\cdot(y\cdot z)$ | A5 | | |
| $x+\dot{\delta}=x$ | A6ID | $\nu_{\mathrm{rel}}(\dot{\delta})=\dot{\delta}$ | DCS1 |
| $\dot{\delta}\cdot x=\dot{\delta}$ | A7ID | $\nu_{\mathrm{rel}}(\underline{\underline{a}})=\underline{\underline{a}}$ | DCS2 |
| | | $\nu_{\mathrm{rel}}(\sigma_{\mathrm{rel}}(x))=\underline{\underline{\delta}}$ | DCS3 |
| $\lfloor x\rfloor^{\omega}=\nu_{\mathrm{rel}}(x)+\sigma_{\mathrm{rel}}(\lfloor x\rfloor^{\omega})$ | USD | $\nu_{\mathrm{rel}}(x+y)=\nu_{\mathrm{rel}}(x)+\nu_{\mathrm{rel}}(y)$ | DCS4 |
| $a=\left\lfloor\underline{\underline{a}}\right\rfloor^{\omega}$ | ATS | $\nu_{\mathrm{rel}}(x\cdot y)=\nu_{\mathrm{rel}}(x)\cdot y$ | DCS5 |

Table 1: Axioms of $\mathrm{BPA}_{\mathrm{drt}}$ ($a\in A\cup\{\delta\}$).

## 2.2 Recursion

A *recursive specification* is a set of equations $E=\{X_i=t_i(X_1,\ldots,X_n):1\leq i\leq n\}$ where $n\geq 1$, the $X_i$ are (formal) variables and the $t_i$ are terms (over $\mathrm{BPA}_{\mathrm{drt}}$), possibly containing variables $X_i$. A sequence of processes in a certain model of $\mathrm{BPA}_{\mathrm{drt}}$ is called a *solution* of $E$ if all equations of $E$ are valid in the model, if we interpret the variables by the corresponding process.

We call a recursive specification $E$ *guarded* if all occurrences of variables $X_i$ in all right-hand sides $t_j$ are guarded, i.e. preceded by an atomic action or a delay. To be precise, an occurrence of a variable $X$ in term $t$ is guarded if $t$ has a subterm $s$ containing this occurrence of $X$, and $s$ has one of the forms $\underline{\underline{a}}\cdot s'$, $a\cdot s'$ or $\sigma_{\mathrm{rel}}(s')$ for some $a\in A$.

The *Recursive Specification Principle RSP* says that a guarded recursive specification has at most one solution. This principle was introduced in [5] and has proven to be very useful in system verifications. Specialised to the recursive equation defining the unbounded start delay, RSP instantiates as shown in Table 2. This special form of RSP will be called RSP(USD) in the sequel.

## 2.3 Structured Operational Semantics

We refer to [1] for an operational semantics for $\mathrm{BPA}_{\mathrm{drt}}$. The set of bisimulation equivalence classes of closed terms then gives a model for $\mathrm{BPA}_{\mathrm{drt}}$, thereby showing consistency of the

| $y = \nu_{\mathrm{rel}}(x) + \sigma_{\mathrm{rel}}(y) \Rightarrow y = \lfloor x \rfloor^{\omega}$ RSP(USD) |
|---|

Table 2: RSP for Unbounded Start Delay.

theory. We can also provide action rules in order to deal with recursion. Then, we obtain a model in which every recursive specification has a solution, and in which every guarded recursive specification has a unique solution (thus, satisfying RSP). For further details, see [7].

**Proposition 2.1** The following identities are derivable from $\mathrm{BPA}_{\mathrm{drt}}$ + RSP(USD):

1. $\sigma_{\mathrm{rel}}(x) = \sigma_{\mathrm{rel}}(x) + \underline{\delta}$
2. $\left\lfloor \dot{\delta} \right\rfloor^{\omega} = \delta$
3. $\lfloor x + y \rfloor^{\omega} = \lfloor x \rfloor^{\omega} + \lfloor y \rfloor^{\omega}$
4. $\lfloor x \cdot y \rfloor^{\omega} = \lfloor x \rfloor^{\omega} \cdot y$
5. $\lfloor \sigma_{\mathrm{rel}}(x) \rfloor^{\omega} = \delta$
6. $a = \underline{a} + \sigma_{\mathrm{rel}}(a)$
7. $\nu_{\mathrm{rel}}(a) = \underline{a}$
8. $\lfloor a \rfloor^{\omega} = a$
9. $a + \delta = a$
10. $\delta \cdot x = \delta$.

**Proof** 1. $\sigma_{\mathrm{rel}}(x) = \sigma_{\mathrm{rel}}(x + \dot{\delta}) = \sigma_{\mathrm{rel}}(x) + \sigma_{\mathrm{rel}}(\dot{\delta}) = \sigma_{\mathrm{rel}}(x) + \underline{\delta}$; 2. Use 1 and RSP(USD); 3. Use DCS4 and RSP(USD); 4,5. Likewise, with DCS5,3; 6. By definition; 7. Use 6; 8. Use 7; 9. $a + \delta = \left\lfloor \underline{a} \right\rfloor^{\omega} + \left\lfloor \underline{\delta} \right\rfloor^{\omega} = \left\lfloor \underline{a} + \underline{\delta} \right\rfloor^{\omega} = \left\lfloor \underline{a} \right\rfloor^{\omega} = a$; 10. Like 9.

### 2.4 Parallel Composition

We extend the theory $\mathrm{BPA}_{\mathrm{drt}}$ with parallel composition, with or without communication, as in [2]. The additional syntax has binary operators $\|$ (merge), $\lfloor\!\lfloor$ (left merge) and $|$ (communication merge), and unary operators $\partial_H$ (encapsulation operator, for $H \subseteq A$). The process $x \| y$ interleaves the behaviours of $x$ and $y$ (with possible synchronisation on communication actions), but it is forced to synchronise on time steps, i.e. a clock tick is executed by both processes at the same time, or if this is not possible, no clock tick can occur. We assume given a partial, commutative and associative communication function $\gamma : A \times A \to A$. The additional axioms are given in Table 3. We obtain a model satsifying RSP for the extended theory $\mathrm{ACP}_{\mathrm{drt}}$ along the same lines as before.

In the sequel, we will only use the so-called *standard communication function.* Suppose we have given two finite sets, the set of messages or data $D$, and the set of ports $P$. For each $d \in D$ and $i \in P$, we have atomic actions $ri(d)$, $si(d)$, $ci(d)$ (denoting *receive, send* and *communicate* $d$ along $i$) and the only defined communications are $\gamma(ri(d), si(d)) = \gamma(si(d), ri(d)) = ci(d)$.

**Proposition 2.2** We derive the following identities for time free atoms in $\mathrm{ACP}_{\mathrm{drt}}$+ RSP(USD):

| | | | |
|---|---|---|---|
| $x \parallel y = x \between y + y \between x + x \mid y$ | CM1 | $\underline{a} \mid \underline{b} = \underline{\underline{\gamma(a,b)}}$ if $\gamma$ defined | DRTCF1 |
| $\underline{a} \between (x + \underline{\delta}) = \underline{a} \cdot (x + \underline{\delta})$ | DRTCM2 | $\underline{a} \mid \underline{b} = \underline{\delta}$ otherwise | DRTCF2 |
| $\underline{a} \cdot x \between (y + \underline{\delta}) = \underline{a} \cdot (x \parallel (y + \underline{\delta}))$ | DRTCM3 | $\dot{\delta} \between x = \dot{\delta}$ | LMID1 |
| $(x + y) \between z = x \between z + y \between z$ | CM4 | $x \between \dot{\delta} = \dot{\delta}$ | LMID2 |
| $\underline{a} \cdot x \mid \underline{b} = (\underline{a} \mid \underline{b}) \cdot x$ | DRTCM5 | $\dot{\delta} \mid x = \dot{\delta}$ | CMID1 |
| $\underline{a} \mid \underline{b} \cdot x = (\underline{a} \mid \underline{b}) \cdot x$ | DRTCM6 | $x \mid \dot{\delta} = \dot{\delta}$ | CMID2 |
| $\underline{a} \cdot x \mid \underline{b} \cdot y = (\underline{a} \mid \underline{b}) \cdot (x \parallel y)$ | DRTCM7 | $\partial_H(\dot{\delta}) = \dot{\delta}$ | DID |
| $(x + y) \mid z = x \mid z + y \mid z$ | CM8 | $\partial_H(\underline{a}) = \underline{a}$ if $a \notin H$ | DRTD1 |
| $x \mid (y + z) = x \mid y + x \mid z$ | CM9 | $\partial_H(\underline{a}) = \underline{\delta}$ if $a \in H$ | DRTD2 |
| $\sigma_{\mathrm{rel}}(x) \between (\sigma_{\mathrm{rel}}(y) + \nu_{\mathrm{rel}}(z)) = \sigma_{\mathrm{rel}}(x \between y)$ | DRT5 | $\partial_H(x + y) = \partial_H(x) + \partial_H(y)$ | D3 |
| $\sigma_{\mathrm{rel}}(x) \mid (\sigma_{\mathrm{rel}}(y) + \nu_{\mathrm{rel}}(z)) = \sigma_{\mathrm{rel}}(x \mid z)$ | DRT6 | $\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$ | D4 |
| $(\sigma_{\mathrm{rel}}(x) + \nu_{\mathrm{rel}}(y)) \mid \sigma_{\mathrm{rel}}(z) = \sigma_{\mathrm{rel}}(x \mid y)$ | DRT7 | $\partial_H(\sigma_{\mathrm{rel}}(x)) = \sigma_{\mathrm{rel}}(\partial_H(x))$ | DRT8 |

Table 3: Additional axioms for $\mathrm{ACP}_{\mathrm{drt}}$ ($a \in A \cup \{\delta\}$).

1. $a \between \lfloor x \rfloor^\omega = a \cdot \lfloor x \rfloor^\omega$
2. $a \cdot x \between \lfloor y \rfloor^\omega = a \cdot (x \parallel \lfloor y \rfloor^\omega)$
3. $a \cdot x \mid b = (a \mid b) \cdot x$
4. $a \mid b \cdot x = (a \mid b) \cdot x$
5. $a \cdot x \mid b \cdot y = (a \mid b) \cdot (x \parallel y)$.
6. $\partial_H(a) = a$ if $a \notin H$
7. $\partial_H(a) = \delta$ if $a \in H$
8. $a \mid b = c$ if $\gamma(a,b) = c$
9. $a \mid b = \delta$ otherwise

Moreover, the following identities are useful in the calculations to come:

10. $\underline{a} \cdot x \between \sigma_{\mathrm{rel}}(y) = \underline{a} \cdot (x \parallel \sigma_{\mathrm{rel}}(y))$
11. $\underline{a} \cdot x \mid \sigma_{\mathrm{rel}}(y) = \underline{\delta}$
12. $\sigma_{\mathrm{rel}}(x) \between \underline{a} \cdot y = \underline{\delta}$
13. $a \cdot x \between \underline{b} \cdot y = \underline{a} \cdot (x \parallel \underline{b}y)$
14. $a \cdot x \mid \underline{b} \cdot y = (\underline{a} \mid \underline{b}) \cdot (x \parallel y)$
15. $\sigma_{\mathrm{rel}}(x) \parallel \sigma_{\mathrm{rel}}(y) = \sigma_{\mathrm{rel}}(x \parallel y)$.

**Proof** 1. $a \between \lfloor x \rfloor^\omega = (\underline{a} + \sigma_{\mathrm{rel}}(a)) \between \lfloor x \rfloor^\omega = \underline{a} \between \lfloor x \rfloor^\omega + \sigma_{\mathrm{rel}}(a) \between \lfloor x \rfloor^\omega = \underline{a} \cdot \lfloor x \rfloor^\omega + \sigma_{\mathrm{rel}}(a) \between (\nu_{\mathrm{rel}}(x) + \sigma_{\mathrm{rel}}(\lfloor x \rfloor^\omega)) = \nu_{\mathrm{rel}}(\underline{a} \cdot \lfloor x \rfloor^\omega) + \sigma_{\mathrm{rel}}(a \between \lfloor x \rfloor^\omega)$ so $a \between \lfloor x \rfloor^\omega = \lfloor \underline{a} \cdot \lfloor x \rfloor^\omega \rfloor^\omega = \lfloor \underline{a} \rfloor^\omega \cdot \lfloor x \rfloor^\omega = a \cdot \lfloor x \rfloor^\omega$. The other identities are equally straightforward, we just display 12: $\sigma_{\mathrm{rel}}(x) \between \underline{a} \cdot y = \sigma_{\mathrm{rel}}(x) \between (\underline{a} \cdot y + \underline{\delta}) = \sigma_{\mathrm{rel}}(x) \between (\nu_{\mathrm{rel}}(a \cdot y) + \sigma_{\mathrm{rel}}(\dot{\delta})) = \sigma_{\mathrm{rel}}(x \between \dot{\delta}) = \sigma_{\mathrm{rel}}(\dot{\delta}) = \underline{\delta}$.

## 3 Some Simple Calculations

In time free process algebra, there is the following standard specification of a one-item buffer with input port $i$ and output port $j$:

$$B^{ij} = \sum_{d \in D} ri(d) \cdot sj(d) \cdot B^{ij}$$

A straightforward calculation (see e.g. [3], page 106) shows that the composition of two such buffers in sequence gives a two-item buffer. In the following, we consider three timed versions of this buffer. In each case, only one input per time slice is possible.

### 3.1 No Delay

We can define a channel that allows one input in every time slice, and outputs with no delay, with input port $i$ and output port $j$ by the following recursive equation:

$$C^{ij} = \sum_{d\in D} ri(d) \cdot \underline{\underline{sj(d)}} \cdot \sigma_{\mathrm{rel}}(C^{ij})$$

Using parts 2,3,5 of Proposition 2.2 we see $C^{ij} = \lfloor C^{ij} \rfloor^{\omega}$. With $H = \{r2(d), s2(d) : d \in D\}$ we can derive:

$$\partial_H(C^{12} \parallel C^{23}) = \partial_H(C^{12} \between C^{23}) + \partial_H(C^{23} \between C^{12}) + \partial_H(C^{12} \mid C^{23}) =$$

$$= \sum_{d\in D} \partial_H\left(r1(d) \cdot \underline{\underline{s2(d)}} \cdot \sigma_{\mathrm{rel}}(C^{12}) \between \lfloor C^{23} \rfloor^{\omega}\right) + \sum_{d\in D} \partial_H\left(r2(d) \cdot \underline{\underline{s3(d)}} \cdot \sigma_{\mathrm{rel}}(C^{23}) \between \lfloor C^{12} \rfloor^{\omega}\right) +$$

$$+ \sum_{d,e\in D} \partial_H\left(r1(d) \cdot \underline{\underline{s2(d)}} \cdot \sigma_{\mathrm{rel}}(C^{12}) \mid r2(e) \cdot \underline{\underline{s3(e)}} \cdot \sigma_{\mathrm{rel}}(C^{23})\right) =$$

$$= \sum_{d\in D} r1(d) \cdot \partial_H\left(\underline{\underline{s2(d)}} \cdot \sigma_{\mathrm{rel}}(C^{12}) \parallel C^{23}\right) + \sum_{d\in D} \delta \cdot \partial_H\left(\underline{\underline{s3(d)}} \cdot \sigma_{\mathrm{rel}}(C^{23}) \parallel C^{12}\right) +$$

$$+ \sum_{d,e\in D} \delta \cdot \partial_H\left(\underline{\underline{s2(d)}} \cdot \sigma_{\mathrm{rel}}(C^{12}) \parallel \underline{\underline{s3(e)}} \cdot \sigma_{\mathrm{rel}}(C^{23})\right) =$$

$$= \sum_{d\in D} r1(d) \cdot \left(\partial_H\left(\underline{\underline{s2(d)}} \cdot \sigma_{\mathrm{rel}}(C^{12}) \between C^{23}\right) + \partial_H\left(C^{23} \between \underline{\underline{s2(d)}} \cdot \sigma_{\mathrm{rel}}(C^{12})\right) +\right.$$

$$\left. + \partial_H\left(\underline{\underline{s2(d)}} \cdot \sigma_{\mathrm{rel}}(C^{12}) \mid C^{23}\right)\right) + \delta =$$

$$= \sum_{d\in D} r1(d) \cdot \left(\underline{\underline{\delta}} + \sum_{e\in D} \partial_H\left(r2(e) \cdot \underline{\underline{s3(e)}} \cdot \sigma_{\mathrm{rel}}(C^{23})\right) \between \underline{\underline{s2(d)}} \cdot \sigma_{\mathrm{rel}}(C^{12}) +\right.$$

$$\left. + \sum_{e\in D} \partial_H\left(\underline{\underline{s2(d)}} \cdot \sigma_{\mathrm{rel}}(C^{12}) \mid r2(e) \cdot \underline{\underline{s3(e)}} \cdot \sigma_{\mathrm{rel}}(C^{23})\right)\right) =$$

$$= \sum_{d\in D} r1(d) \cdot \left(\underline{\underline{\delta}} + \underline{\underline{\delta}} + \underline{\underline{c2(d)}} \cdot \partial_H\left(\sigma_{\mathrm{rel}}(C^{12}) \parallel \underline{\underline{s3(d)}} \cdot \sigma_{\mathrm{rel}}(C^{23})\right)\right) =$$

$$= \sum_{d\in D} r1(d) \cdot \underline{\underline{c2(d)}} \cdot \underline{\underline{s3(d)}} \cdot \partial_H\left(\sigma_{\mathrm{rel}}(C^{12}) \parallel \sigma_{\mathrm{rel}}(C^{23})\right) =$$

$$= \sum_{d \in D} r1(d) \cdot \underline{\underline{c2(d)}} \cdot \underline{\underline{s3(d)}} \cdot \sigma_{\mathrm{rel}}\left(\partial_H\left(C^{12} \parallel C^{23}\right)\right).$$

Using a theory of abstraction as outlined in [2], we can derive that after hiding the $\underline{\underline{c2(d)}}$ actions, the composition behaves again as a no-delay channel, with input port 1 and output port 3. This fact was also stated in [4], where this no-delay channel (called minimal stream delayer) acts as the identity in a data flow algebra.

### 3.2 Delay 1, Capacity 1

It is interesting to see what happens if we change the previous specification slightly. Consider

$$D^{ij} = \sum_{d \in D} ri(d) \cdot \sigma_{\mathrm{rel}}\left(\underline{\underline{sj(d)}} \cdot D^{ij}\right).$$

This specification describes a buffer with capacity one and delay between input and output of one time unit. The composition $\partial_H(D^{12} \parallel D^{23})$ (with $H$ as above) satisfies the following recursive specification:

$$X = \sum_{d \in D} r1(d) \cdot \sigma_{\mathrm{rel}}\left(\underline{\underline{c2(d)}}\right) \cdot X_d$$

$$X_d = \sum_{e \in D} \underline{\underline{r1(e)}} \cdot \sigma_{\mathrm{rel}}\left(\underline{\underline{s3(d)}}\right) \cdot \underline{\underline{c2(e)}} \cdot X_e + \sigma_{\mathrm{rel}}\left(\underline{\underline{s3(d)}} \cdot X + \sum_{e \in D} \underline{\underline{r1(e)}} \cdot \underline{\underline{s3(d)}} \cdot \sigma_{\mathrm{rel}}\left(\underline{\underline{c2(e)}}\right) \cdot X_e\right)$$

(for each $d \in D$). The composition denotes a buffer with capacity two and delay of two time units. This is similar to the result obtained if all timing is left out (see [3], page 106).

### 3.3 Delay 1, More Capacity

Finally, we drop the restriction in the previous specification that output must occur before the next input. We obtain the following specification:

$$E^{ij} = \sum_{d \in D} ri(d) \cdot \sigma_{\mathrm{rel}}\left(\underline{\underline{sj(d)}} \parallel E^{ij}\right).$$

Now, the composition satisfies the following recursive specification:

$$Y^0 = \sum_{d \in D} r1(d) \cdot \sigma_{\mathrm{rel}}(Y^1_d)$$

$$Y_d^1 = \underline{\underline{c2(d)}} \cdot Y_d^2 + \sum_{e \in D} \underline{\underline{r1(e)}} \cdot \underline{\underline{c2(d)}} \cdot \sigma_{\mathrm{rel}}(Y_{de}^3) \quad \text{for} \quad d \in D$$

$$Y_d^2 = \sigma_{\mathrm{rel}}\left(\underline{\underline{s3(d)}} \cdot Y^0 + \sum_{e \in D} \underline{\underline{r1(e)}} \cdot \underline{\underline{s3(d)}} \cdot \sigma_{\mathrm{rel}}(Y_e^1)\right) + \sum_{e \in D} \underline{\underline{r1(e)}} \cdot \sigma_{\mathrm{rel}}(Y_{de}^3) \quad \text{for} \quad d \in D$$

$$Y_{de}^3 = \underline{\underline{s3(d)}} \cdot Y_e^1 + \underline{\underline{c2(e)}} \cdot \left(\underline{\underline{s3(d)}} \cdot Y_d^2 + \sum_{f \in D} \underline{\underline{r1(f)}} \cdot \underline{\underline{s3(d)}} \cdot \sigma_{\mathrm{rel}}(Y_{ef}^3)\right) +$$

$$+ \sum_{f \in D} \underline{\underline{r1(f)}} \cdot \left(\underline{\underline{c2(e)}} \parallel \underline{\underline{s3(d)}}\right) \cdot \sigma_{\mathrm{rel}}(Y_{ef}^3) \quad \text{for} \quad d, e \in D$$

Again, the delay between input and output is two time units, but now, it is possible that three data elements are present in the system at the same time.

## References

[1] J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra. Technical Report CSR 95-09, Eindhoven University of Technology, Computing Science Department, 1995. To appear in Formal Aspects of Computing.

[2] J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra with abstraction. Technical Report CSR 95-17, Eindhoven University of Technology, Computing Science Department, 1995. To appear in Proceedings FCT'95, Dresden.

[3] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[4] J.A. Bergstra and Gh. Ştefănescu. Network algebra for synchronous and asynchronous dataflow. Technical Report LGPS 122, Utrecht University, Department of Philosophy, 1994.

[5] J.A. Bergstra and J.W. Klop. Verification of an alternating bit protocol by means of process algebra. In W. Bibel and K.P. Jantke, editors, *Math. Methods of Spec. and Synthesis of Software Systems '85*, number 215 in LNCS, pages 9–23. Springer-Verlag, 1986.

[6] F.E.J. Kruseman Aretz. *Vallen en opstaan (de computer in de wiskunde).* Inaugural Address. Scheltema en Holkema, Amsterdam, 1967.

[7] M.A. Reniers and J.J. Vereijken. Completeness in discrete time process algebra. draft, 1995.

# An application of program derivation techniques to 18th-century mathematics

dedicated to Prof. Dr. F.E.J. Kruseman Aretz

A. Bijlsma

## 1 Introduction

This note presents a derivation of one of the proofs given by Euler [4] of the famous theorem that every prime congruent to 1 modulo 4 is the sum of two squares—an observation attributed to Girard [5] in [2]. The proof seems to be of methodological interest for two reasons: first, because it demonstrates that the techniques for program derivation developed by computing science have matured to a level where a proof that originally took many years to find may now be constructed with a minimum of invention; secondly, because the proof furnishes a counterexample to Dijkstra's remark [3] that all known methods of factorization are nonconstructive. (It does so because writing a prime as $x^2 + y^2$ is equivalent to factorizing it as $(x + i \cdot y) \cdot (x - i \cdot y)$ in $\mathbf{Z}[i]$.)

In what follows, we shall depart from established mathematical notation in one respect. The assertion that $x$ and $y$ differ by a multiple of $k$ will be written as

$$x =_k y$$

rather than

$$x \equiv y \,(\mathrm{mod}\, k) \quad .$$

There are two reasons for this decision: the classical notation is too confusing in the presence of logical equivalence, and equational reasoning demands an infix operator for the equivalence relation involved. The use of an equality sign is acceptable since a weak form of Leibniz' rule is present: if $f$ is a polynomial over the integers,

$$x =_k y \;\Rightarrow\; f.x =_k f.y \tag{1}$$

for integer $x, y$ and positive integer $k$.

## 2 First approximation

We are looking for a constructive proof: given a prime $p$ with $p =_4 1$, we want to find $x, y$ such that

$$x^2 + y^2 = p \quad . \tag{2}$$

Our first approximation is produced by the most commonly used technique in program derivation: we replace a constant by a variable, and consider

$$P0: \quad x^2 + y^2 = k \cdot p$$

as invariant of a repetition to be constructed. The chosen constant is acceptable because is easy to see when $P0$ implies the desired postcondition (2), namely when $k = 1$; and because $P0$ is easily initialized, namely by

$$x,\ y,\ k := p,\ p,\ 2 \cdot p \ .$$

Progress towards the postcondition will be made by decreasing $k$. In order to be able to take $k$ as the variant function, we strengthen the invariant by

$$P1: \quad k \geq 1 \ .$$

## 3 Invariance of $P0$

To be able to guarantee invariance of $P0$ as $k$ is decreased, we need a method to transform a sum of two squares into a smaller one. Here our starting point is the observation that a sum of two squares constitutes the square of the absolute value of a complex number. This gives

$$\begin{array}{ll} & (x^2 + y^2) \cdot (u^2 + v^2) \\ = & \quad \{\text{introduce complex numbers}\} \\ & |x + i \cdot y|^2 \cdot |u + i \cdot v|^2 \\ = & \quad \{\text{distribution}\} \\ & |(x + i \cdot y) \cdot (u + i \cdot v)|^2 \\ = & \quad \{\text{multiplication}\} \\ & |(x \cdot u - y \cdot v) + i \cdot (x \cdot v + y \cdot u)|^2 \\ = & \quad \{\text{eliminate complex numbers}\} \\ & (x \cdot u - y \cdot v)^2 + (x \cdot v + y \cdot u)^2 \ , \end{array}$$

so

$$(x^2 + y^2) \cdot (u^2 + v^2) = (x \cdot u - y \cdot v)^2 + (x \cdot v + y \cdot u)^2 \ . \tag{3}$$

To number theorists, the occurrence of this identity does not come as a surprise: in fact, it constitutes the proof of the theorem that the set of sums of two squares is closed under multiplication, which logically and chronologically precedes Girard's theorem.

Inspired by (3), we investigate an assignment of the form

$$x,\ y := x \cdot u - y \cdot v,\ x \cdot v + y \cdot u \ .$$

For any $k'$, we have

$$
\begin{array}{ll}
 & P0(k,\ x,\ y := k',\ x \cdot u - y \cdot v,\ x \cdot v + y \cdot u) \\
\equiv & \quad \{\text{substitution}\} \\
 & (x \cdot u - y \cdot v)^2 + (x \cdot v + y \cdot u)^2 = k' \cdot p \\
\equiv & \quad \{(3)\} \\
 & (x^2 + y^2) \cdot (u^2 + v^2) = k' \cdot p \\
\equiv & \quad \{P0\} \\
 & k \cdot p \cdot (u^2 + v^2) = k' \cdot p \\
\equiv & \quad \{\} \\
 & k' = k \cdot (u^2 + v^2) \ .
\end{array}
$$

We conclude that, for arbitrary $u, v$, predicate $P0$ is invariant under

$$k,\ x,\ y := k \cdot (u^2 + v^2),\ x \cdot u - y \cdot v,\ x \cdot v + y \cdot u \ .$$

Because this assignment must decrease $k$, we are led to the condition

$$u^2 + v^2 < 1 \ . \tag{4}$$

Equation (4) does not have any interesting solutions in integers, since $u, v = 0, 0$ is hopeless in view of $P1$. However, there is no need for $u$ and $v$ to be integer: the derivation given above works equally well for rational $u$ and $v$, provided the expressions on the right hand side of the assignment are integers. Let us formalize this condition: putting $u = a/c$ and $v = b/c$, we find that the values assigned to $x, y, k$ are integers if

$$c \mid x \cdot a - y \cdot b \ , \tag{5}$$

$$c \mid x \cdot b + y \cdot a \ , \tag{6}$$

$$c^2 \mid k \cdot (a^2 + b^2) \ . \tag{7}$$

(The symbol $\mid$ is pronounced as 'divides'.) In terms of $a, b, c$, equation (4) may be reformulated as

$$a^2 + b^2 < c^2 \ . \tag{8}$$

To ensure invariance of $P0$ while decreasing $k$, it suffices to find a solution to (5) through (8) and then to perform the assignment

$$k,\ x,\ y := k \cdot (a^2 + b^2)/c^2,\ (x \cdot a - y \cdot b)/c,\ (x \cdot b + y \cdot a)/c \ .$$

First we look at (7). If $c$ and $k$ were to be relatively prime, it would follow that

$$
\begin{array}{ll}
 & (7) \\
\equiv & \quad \{\} \\
 & c^2 \mid k \cdot (a^2 + b^2) \\
\equiv & \quad \{c \text{ and } k \text{ relatively prime}\} \\
 & c^2 \mid a^2 + b^2 \\
\equiv & \quad \{(8)\} \\
 & a^2 + b^2 = 0 \ ,
\end{array}
$$

and we have already rejected this solution in view of $P1$. Hence $c$ and $k$ must have a nontrivial factor in common. But since we know nothing of the multiplicative structure of $k$, and indeed $k$ may well be prime, the obvious way to achieve this is to take $c = k$. With this choice, we have

$$
\begin{array}{ll}
 & (7) \\
\equiv & \quad \{c = k\} \\
 & k \mid a^2 + b^2 \\
\equiv & \quad \{\} \\
 & a^2 + b^2 =_k 0 \\
\equiv & \quad \{P0, \text{ as the only relevant information on } k \text{ we have}\} \\
 & a^2 + b^2 =_k x^2 + y^2 \\
\Leftarrow & \quad \{(1)\} \\
 & a =_k y \ \wedge \ b =_k x \ .
\end{array}
$$

In the last line, we might equally well have decided to associate $a$ with $x$ and $b$ with $y$, as is suggested by the order of the alphabet. However, consideration of (5) shows why we have made the right choice.

$$
\begin{array}{ll}
 & (5) \\
\equiv & \quad \{c = k\} \\
 & k \mid x \cdot a - y \cdot b \\
\equiv & \quad \{\} \\
 & x \cdot a - y \cdot b =_k 0 \\
\equiv & \quad \{\} \\
 & x \cdot a - y \cdot b =_k x \cdot y - y \cdot x \\
\Leftarrow & \quad \{(1)\} \\
 & a =_k y \ \wedge \ b =_k x \ .
\end{array}
$$

And we are in luck, for also

$$
\begin{array}{ll}
 & (6) \\
\equiv & \quad \{c = k\} \\
 & k \mid x \cdot b + y \cdot a \\
\equiv & \quad \{\} \\
 & x \cdot b + y \cdot a =_k 0 \\
\equiv & \quad \{P0\} \\
 & x \cdot b + y \cdot a =_k x^2 + y^2 \\
\Leftarrow & \quad \{(1)\} \\
 & a =_k y \ \wedge \ b =_k x \ .
\end{array}
$$

The choice $c = k$ thus allows us to dispense with (5) through (7), provided we take $a$ and $b$ such that $a =_k y$ and $b =_k x$. We are left with (8).

$$
\begin{array}{ll}
 & (8) \\
\equiv & \quad \{c = k\} \\
 & a^2 + b^2 < k^2 \\
\Leftarrow & \quad \{\text{dividing the obligation equally between } a \text{ and } b\} \\
 & |a| < k/\sqrt{2} \ \wedge \ |b| < k/\sqrt{2} \ .
\end{array}
$$

We conclude that equations (5) through (8) are satisfied if $a$ and $b$ are chosen such that

$$a =_k y \ \wedge \ |a| < k/\sqrt{2} \ , \tag{9}$$
$$b =_k x \ \wedge \ |b| < k/\sqrt{2} \ . \tag{10}$$

An explicit solution of (9) and (10) is easy to find: for instance, one of several solutions of (9) is given by

$$a = \begin{cases} y \bmod k & \text{if } y \bmod k < k/2 \ , \\ y \bmod k - k & \text{if } y \bmod k \geq k/2 \ . \end{cases}$$

However, we have no need for a definition that is any more specific than (9).

The conclusion of the preceding calculations may be formulated as follows: $k$ is decreased and $P0$ is kept invariant by the statements

$$
\begin{array}{l}
\quad a: \ \ a =_k y \ \wedge \ |a| < k/\sqrt{2} \\
; \ b: \ \ b =_k x \ \wedge \ |b| < k/\sqrt{2} \\
; \ k, \ x, \ y := (a^2 + b^2)/k, \ (x \cdot a - y \cdot b)/k, \ (x \cdot b + y \cdot a)/k \ .
\end{array}
$$

## 4 Invariance of $P1$

For $a$ and $b$ chosen as in the preceding section, we have

$$k \mid a^2 + b^2 \ , \tag{11}$$
$$a =_k y \ \wedge \ b =_k x \ . \tag{12}$$

Under assumption of (11), (12), and $k \neq 1$, we have

$$
\begin{array}{ll}
 & P1(k := (a^2 + b^2)/k) \\
\equiv & \quad \{\text{substitution}\} \\
 & (a^2 + b^2)/k \geq 1 \\
\equiv & \quad \{(11)\} \\
 & \neg(a^2 + b^2 = 0) \\
\equiv & \quad \{\} \\
 & \neg(a = 0 \ \wedge \ b = 0) \\
\Leftarrow & \quad \{(12)\}
\end{array}
$$

$$
\begin{array}{ll}
 & \neg(k \mid x \ \wedge \ k \mid y) \\
\Leftarrow & \quad \{\} \\
 & \neg(k^2 \mid x^2 + y^2) \\
\equiv & \quad \{P0\} \\
 & \neg(k^2 \mid k \cdot p) \\
\equiv & \quad \{\} \\
 & \neg(k \mid p) \\
\Leftarrow & \quad \{k \neq 1 \text{ and } p \text{ is prime}\} \\
 & k \neq p \ .
\end{array}
$$

Since it is very difficult to ensure invariance of $k \neq p$ under $k := (a^2 + b^2)/k$, we decide to strengthen the invariant of the repetition by

$$P2: \quad k < p \ .$$

As we have already shown $k$ to decrease in each iteration, invariance of $P2$ is trivial. However, the original initialization does not establish $P2$ and we are forced to replace it.

## 5 Initialization

The standard way to construct an initialization is to choose an 'easy' value, say 0 or 1, for one of the variables, and to calculate the corresponding values of the other variables. In the case under consideration, choosing $k = 1$ is useless, since that takes us back to the original postcondition. The problem being symmetric in $x$ and $y$, it suffices to consider start values for $y$.

Initialization of $P0..2$ with $y = 0$ turns out to be impossible, as the diligent reader can easily check. So we consider $y = 1$. This leads to

$$
\begin{array}{ll}
 & P0(y := 1) \\
\equiv & \quad \{\text{substitution}\} \\
 & x^2 + 1 = k \cdot p \\
\equiv & \quad \{\} \\
 & x^2 =_p -1 \ \wedge \ k = (x^2 + 1)/p \ .
\end{array}
$$

Hence $k := (x^2 + 1)/p$ establishes $P0(y := 1)$ provided $x$ is chosen such that $x^2 =_p -1$. This value of $k$ is obviously positive, so $P1$ is established as well. As to $P2$, we have

$$
\begin{array}{ll}
 & (k < p)(k := (x^2 + 1)/p) \\
\equiv & \quad \{\text{substitution}\} \\
 & x^2 + 1 < p^2 \\
\Leftarrow & \quad \{(p-1)^2 + 1 < p^2 \text{ since } p \geq 2\} \\
 & 1 \leq x < p \ .
\end{array}
$$

Our final proof obligation is the existence of an $x$ satisfying

$$1 \leq x < p \ \wedge \ x^2 =_p -1 \ . \tag{13}$$

Here our derivation could end, for it so happens that the existence of an $x$ satisfying (13) is a very famous theorem in number theory, known as the First Supplement to the Law of Quadratic Reprocity. But for completeness' sake, we supply a proof—one that was originally devised by Lagrange [7].

When we look at the proof obligation (13), the property $(=_p -1)$ brings to mind *Wilson's theorem:* for prime $p$,

$$(p-1)! =_p -1 \ . \tag{14}$$

(An easy proof can be found in, for instance, [1, theorem 5.24].) Taking Wilson's theorem as our point of departure, we have for odd $p$, with $n$ short for $(p-1)/2$,

$$\begin{array}{ll}
 & -1 \\
=_p & \quad \{(14)\} \\
 & (p-1)! \\
= & \quad \{\text{definition of factorial}\} \\
 & (\Pi\, j : 1 \leq j < p : j) \\
= & \quad \{\text{domain split}\} \\
 & (\Pi\, j : 1 \leq j \leq n : j) \cdot (\Pi\, j : n < j < p : j) \\
= & \quad \{\text{factorial} \parallel \text{dummy transformation } j := p - j\} \\
 & n! \cdot (\Pi\, j : 0 < j < p - n : p - j) \\
= & \quad \{n = p - n - 1\} \\
 & n! \cdot (\Pi\, j : 1 \leq j \leq n : p - j) \\
=_p & \quad \{p - j =_p -j\} \\
 & n! \cdot (\Pi\, j : 1 \leq j \leq n : -j) \\
= & \quad \{\text{distribution of } - \text{ over } \Pi\} \\
 & n! \cdot (-1)^n \cdot (\Pi\, j : 1 \leq j \leq n : j) \\
= & \quad \{\text{factorial}\} \\
 & (-1)^n \cdot n!^2 \\
= & \quad \{\text{provided } p =_4 1\} \\
 & n!^2 \ .
\end{array}$$

Notice that the last step in the derivation is the first time that the condition $p =_4 1$ plays a role.

We conclude that (13) is established by the assignment

$$x := ((p-1)/2)! \ \mathbf{mod} \ p$$

for $p$ a prime with $p =_4 1$. Inserting this into the algorithm, we obtain the following constructive proof:

$$
\begin{array}{l}
\quad x := ((p-1)/2)! \ \mathbf{mod}\ p \\
;\ k := (x^2+1)/p \\
;\ y := 1 \\
;\ \{\mathrm{inv}\ P0: \quad x^2+y^2 = k\cdot p \quad , \\
\qquad\ P1: \quad k \geq 1 \quad , \\
\qquad\ P2: \quad k < p \quad ; \\
\quad \mathrm{bd}\ k \\
\quad \} \\
\quad \mathbf{do}\ k \neq 1 \rightarrow \quad a: \quad a =_k y \ \wedge\ |a| < k/\sqrt{2} \\
\qquad\qquad\qquad\quad ;\ b: \quad b =_k x \ \wedge\ |b| < k/\sqrt{2} \\
\qquad\qquad\qquad\quad ;\ k,\ x,\ y := (a^2+b^2)/k,\ (x\cdot a - y\cdot b)/k,\ (x\cdot b + y\cdot a)/k \\
\quad \mathbf{od}\ .
\end{array}
$$

**Acknowledgement** The ETAC read a previous version of this note and suggested several improvements in the presentation.

## References

[1] T.M. Apostol, *Introduction to analytic number theory*. Springer, New York, 1976.

[2] L.E. Dickson, *History of the theory of numbers*, vol. II. Carnegie Institute, Washington, 1919. Repr. Chelsea, New York, 1966.

[3] E.W. Dijkstra, *A derivation of a proof by D. Zagier*. EWD1154, August 1993.

[4] L. Euler, 'Novae demonstrationes circa resolutionem numerorum in quadrata'. *Acta Eruditorium Lipsiae* (1773), 193.

[5] A. Girard (ed.), *L'Arithmétique de Simon Stevin*. Leiden, 1625.

[6] G.H. Hardy & E.M. Wright, *An introduction to the theory of numbers*. 4th ed. Oxford University Press, 1960.

[7] J.L. Lagrange, 'Démonstration d'un théorème nouveau concernant les nombres premiers', *Nouv. Mém. Acad. Roy. Berlin* **2** (1773), 125–337.

[8] D. Zagier, 'A one-sentence proof that every prime $p \equiv 1 \pmod 4$ is a sum of two squares'. *Amer. Math. Monthly* **97** (1990), 144.

# Over de oplossing van een potentiaalprobleem door conforme afbeelding

J. Boersma

Bij het onderwerp Wiener–Hopf techniek, onderdeel van mijn college Toegepaste Analyse 2, gebruik ik vaak als voorbeeld (of als tentamenopgave) het volgende potentiaalprobleem voor de functie $\Phi(x,y)$:

$$(1)\qquad \begin{cases} \Delta\Phi = \Phi_{xx} + \Phi_{yy} = 0, \quad -\infty < x < \infty,\ 0 < y < \pi\ ; \\ \Phi(x,0) = 1,\ x > 0; \quad \Phi_y(x,0) = 0,\ x < 0; \quad \Phi(x,\pi) = 0,\ -\infty < x < \infty\ . \end{cases}$$

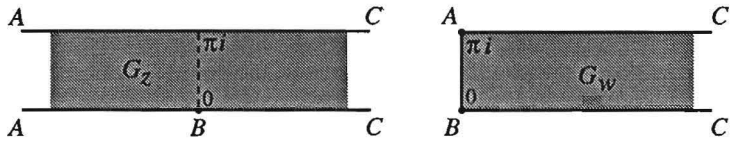
Het voorbeeld is ontleend aan Noble [1, p. 135, Ex. 3.17(i)] waar de randvoorwaarde $\Phi(x,0) = 1$, $x > 0$, vervangen wordt door $\Phi(x,0) = e^{-\varepsilon x}$, $x > 0$, met $\varepsilon > 0$. Het probleem is op te lossen met behulp van Fourier–transformatie en Wiener–Hopf techniek, waarna de oplossing in de vorm van een Fourierintegraal nog is uit te werken tot een reeks van residuen. In de limiet voor $\varepsilon \downarrow 0$ gaat deze reeksoplossing over in

$$(2)\qquad \Phi(x,y) = \begin{cases} 1 - \dfrac{y}{\pi} - \dfrac{1}{\pi^{3/2}} \displaystyle\sum_{n=1}^{\infty} \frac{\Gamma(n+\frac{1}{2})}{n!n}\, e^{-nx} \sin(ny), & x > 0,\ 0 \le y \le \pi\ , \\ \dfrac{1}{\pi^{3/2}} \displaystyle\sum_{n=0}^{\infty} \frac{\Gamma(n+\frac{1}{2})}{n!(n+\frac{1}{2})}\, e^{(n+\frac{1}{2})x} \cos(n+\frac{1}{2})y, & x < 0,\ 0 \le y \le \pi\ . \end{cases}$$

Men kan gemakkelijk verifiëren dat deze uitkomst voldoet aan de vergelijking van Laplace (althans voor $x \neq 0$) en aan de randvoorwaarden in (1). Bezwaar van de oplossing is dat de reeksen in (2) langzaam convergeren in de buurt van $x = 0$. Voorts is niet duidelijk of de twee uitdrukkingen voor $\Phi(x,y)$ continu differentieerbaar aansluiten over $x = 0$. Om deze bezwaren te ondervangen zullen we het potentiaalprobleem nog eens oplossen met de methode van conforme afbeelding. Hierbij wordt het probleem (1) door een geschikte conforme afbeelding overgevoerd in een potentiaalprobleem dat elementair is op te lossen.

Voer in de complexe variabele $z = x+iy$. We zoeken nu een conforme afbeelding $w = f(z)$ van de strook $G_z : 0 <$ Im $z < \pi$, op de halve strook $G_w :$ Re $w > 0$, $0 <$ Im $w < \pi$, zodanig dat de punten $z = \infty, 0, \infty$ op de rand van $G_z$ overgaan in resp. $w = \pi i, 0, \infty$ op de rand van $G_w$; deze punten zijn aangegeven door resp. $A,B,C$ in fig. 1.

De afbeelding $w = f(z)$ wordt bepaald door eerst $G_z$ en $G_w$ beide op een bovenhalfvlak af te beelden als volgt:

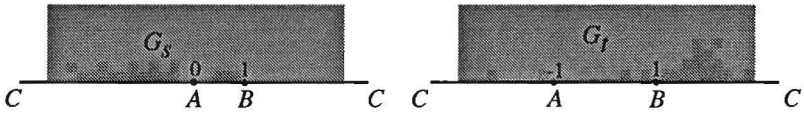



Figuur 1: Conforme afbeelding $w = f(z)$ van $G_z$ op $G_w$.

i) $s = e^z$ beeldt $G_z$ conform af op $G_s$ : Im $s > 0$, waarbij de punten $A,B,C$ overgaan in resp. $s = 0, 1, \infty$.

ii) $t = \cosh w$ beeldt $G_w$ conform af op $G_t$ : Im $t > 0$, waarbij de punten $A,B,C$ overgaan in resp. $t = -1, 1, \infty$. De inverse van deze afbeelding wordt gegeven door $w = \log(t + \sqrt{t^2 - 1})$.

De halfvlakken $G_s$ en $G_t$ worden vervolgens verbonden door de afbeelding $t = 2s - 1$, waarbij gelijknamige punten $A,B,C$ in elkaar overgaan; zie fig. 2.




Figuur 2: Conforme afbeelding $t = 2s - 1$ van $G_s$ op $G_t$.

Door samenstelling van de voorgaande afbeeldingen

$$w = \log(t + \sqrt{t^2 - 1}), \quad t = 2s - 1 = 2e^z - 1 \ ,$$

vinden we de gezochte afbeelding

$$(3) \qquad w = f(z) = \log(2e^z - 1 + 2\sqrt{e^{2z} - e^z}) = 2\log(e^{z/2} + \sqrt{e^z - 1}) \ ,$$

waarin de hoofdwaarde van de wortel en van de logaritme te nemen is.

Schrijf $w = u + iv$, met $u \in \mathbb{R}$, $v \in \mathbb{R}$. Onder de conforme afbeelding $w = f(z)$ gaat het probleem (1) over in het volgende potentiaalprobleem voor de functie $\tilde{\Phi}(u, v) = \Phi(x, y)$:

$$(4) \qquad \begin{cases} \Delta\tilde{\Phi} = \tilde{\Phi}_{uu} + \tilde{\Phi}_{vv} = 0, \quad u > 0,\ 0 < v < \pi \ ; \\ \tilde{\Phi}(u, 0) = 1,\ u > 0; \quad \tilde{\Phi}_u(0, v) = 0,\ 0 < v < \pi; \quad \tilde{\Phi}(u, \pi) = 0,\ u > 0 \ . \end{cases}$$

De oplossing van het laatste probleem is elementair, namelijk

$$\tilde{\Phi}(u,v) = 1 - \frac{v}{\pi} = 1 - \frac{1}{\pi}\,\mathrm{Im}\; w\;.$$

Terugvertaald naar het complexe $z$-vlak volgt de oplossing van het oorspronkelijke potentiaalprobleem (1):

$$(5)\qquad \Phi(x,y) = 1 - \frac{1}{\pi}\,\mathrm{Im}\; f(z) = 1 - \frac{2}{\pi}\,\mathrm{Im}\;\log(e^{z/2} + \sqrt{e^z - 1}),\quad (z = x + iy)\;.$$

Deze vorm van de oplossing is meer expliciet dan de eerdere vorm (2), en daardoor geschikter voor analytische en numerieke bewerking. Zo is gemakkelijk in te zien dat $\Phi(x,0) = 1$, $x > 0$, en $\Phi(x,\pi) = 0$, $-\infty < x < \infty$, in overeenstemming met de randvoorwaarden in (1), terwijl voor $x < 0$ geldt

$$\Phi(x,0) = 1 - \frac{2}{\pi}\,\mathrm{Im}\;\log(e^{x/2} + i\sqrt{1 - e^x}) = 1 - \frac{2}{\pi}\arccos(e^{x/2}) = \frac{2}{\pi}\arcsin(e^{x/2}),\quad x < 0\;.$$

Met iets meer moeite is ook $\Phi(0,y)$ uit te werken tot

$$\Phi(0,y) = \frac{\pi - y}{2\pi} + \frac{2}{\pi}\arctan\left(\frac{\sin\frac{1}{4}(\pi - y)}{\sqrt{2\sin\frac{1}{2}y} + \cos\frac{1}{4}(\pi - y)}\right)\;,\quad 0 < y < \pi\;.$$

Voorts is uit (5) een analoge expliciete voorstelling voor grad $\Phi$ af te leiden. Met gebruik van de differentiaalvergelijkingen van Cauchy–Riemann voor de analytische functie $f(z)$ vinden we

$$\Phi_y(x,y) + i\Phi_x(x,y) = -\frac{1}{\pi}f'(z) = -\frac{1}{\pi}e^{z/2}(e^z - 1)^{-1/2}\;,$$

waaruit de afgeleiden $\Phi_x$ en $\Phi_y$ zijn af te lezen als het imaginaire resp. reële deel van $-f'(z)/\pi$.

Tenslotte zullen we nog verifiëren dat de oplossing (2) verkregen met behulp van de Wiener–Hopf techniek, overeenstemt met de oplossing (5). Schrijf daartoe de oplossing (2) in de vorm, analoog aan (5),

$$(6)\qquad \Phi(x,y) = \begin{cases} 1 - \mathrm{Im}\; F(z), & \mathrm{Re}\; z = x > 0,\ 0 < \mathrm{Im}\; z = y < \pi\;, \\ 1 - \mathrm{Im}\; G(z), & \mathrm{Re}\; z = x < 0,\ 0 < \mathrm{Im}\; z = y < \pi\;. \end{cases}$$

Hierin worden de functies $F(z)$ en $G(z)$ gegeven door

$$(7)\qquad F(z) = \frac{z}{\pi} - \frac{1}{\pi^{3/2}}\sum_{n=1}^{\infty}\frac{\Gamma(n+\frac{1}{2})}{n!n}\,e^{-nz}\;,\quad \mathrm{Re}\; z > 0,\ 0 < \mathrm{Im}\; z < \pi\;,$$

$$(8)\qquad G(z) = i - \frac{i}{\pi^{3/2}}\sum_{n=0}^{\infty}\frac{\Gamma(n+\frac{1}{2})}{n!(n+\frac{1}{2})}\,e^{(n+\frac{1}{2})z}\;,\quad \mathrm{Re}\; z < 0,\ 0 < \mathrm{Im}\; z < \pi\;.$$

Differentiatie van de voorgaande reeksen leidt tot een binomiaalreeks die te sommeren is, zodat

$$F'(z) = \frac{1}{\pi^{3/2}} \sum_{n=0}^{\infty} \frac{\Gamma(n+\frac{1}{2})}{n!} e^{-nz} = \frac{1}{\pi}(1-e^{-z})^{-1/2}\,, \qquad \text{Re } z > 0,\ 0 < \text{ Im } z < \pi\,,$$

$$G'(z) = -\frac{i}{\pi^{3/2}} \sum_{n=0}^{\infty} \frac{\Gamma(n+\frac{1}{2})}{n!} e^{(n+\frac{1}{2})z} = -\frac{i}{\pi} e^{z/2}(1-e^{z})^{-1/2}\,, \qquad \text{Re } z < 0,\ 0 < \text{ Im } z < \pi\,,$$

waarin de hoofdwaarde van de wortels $(1-e^{\mp z})^{-1/2}$ te nemen is. Men kan gemakkelijk inzien dat $F'(z)$ en $G'(z)$ elkaars analytische voortzetting zijn met de gemeenschappelijke voorstelling

$$F'(z) = G'(z) = \frac{1}{\pi} e^{z/2}(e^z-1)^{-1/2} = f'(z)\,.$$

Bepaal nu $F(z)$ en $G(z)$ door integratie van $F'(z) - 1/\pi$ en $G'(z)$ vanuit $\text{Re } z = +\infty$ resp. $\text{Re } z = -\infty$. Uit (7) en (8) volgen de benodigde "beginwaarden": $F(z) - z/\pi \to 0$ als $\text{Re } z \to +\infty$; $G(z) \to i$ als $\text{Re } z \to -\infty$. Aldus vinden we

$$F(z) - \frac{z}{\pi} = \frac{1}{\pi} \int_{z}^{\text{Re } z=+\infty} [1 - e^{\zeta/2}(e^{\zeta}-1)^{-1/2}]d\zeta$$

$$= \frac{2}{\pi} \int_{e^{z/2}}^{\infty} \left[\frac{1}{t} - \frac{1}{\sqrt{t^2-1}}\right] dt = -\frac{2}{\pi}\log 2 - \frac{z}{\pi} + \frac{2}{\pi}\log(e^{z/2} + \sqrt{e^z-1})\,,$$

$$G(z) - i = \frac{1}{\pi} \int_{\text{Re } z=-\infty}^{z} e^{\zeta/2}(e^{\zeta}-1)^{-1/2}d\zeta$$

$$= \frac{2}{\pi} \int_{0}^{e^{z/2}} \frac{dt}{\sqrt{t^2-1}} = -i + \frac{2}{\pi}\log(e^{z/2} + \sqrt{e^z-1})\,.$$

Hieruit is af te lezen dat

$$\text{(9)} \qquad \text{Im } F(z) = \text{ Im } G(z) = \frac{1}{\pi} \text{ Im } f(z)\,,$$

zodat de oplossing (2), herschreven in de vorm (6), en de oplossing (5) overeenstemmen.

**Referentie**

[1] B. Noble, Methods based on the Wiener–Hopf technique for the solution of partial differential equations, Pergamon Press, New York, 1958.

# An old pentomino problem revisited

C.J. Bouwkamp

## 1 Introduction

Pentominos are the twelve combinations of five squares connected edge-to-edge as depicted in Fig.1 along with their alphabetical names. Before Golomb coined them "pentominoes", they were known as "five-pieces" or even "fives". The number of pentomino fans is very large, and so is the number of different problems posed by them, solved and unsolved.

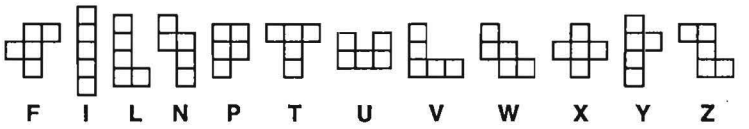



Figure 1: The twelve pentominos and their alphabetical names.

Sets of pentominos are for sale all over the world, in wood, hard or foam plastic, or other, and often in colour. If you want to follow the constructions in this paper, you cannot use these fancy pentominos, because ours are going to be folded on the surface of a cube. So, ordinary paper cuttings will do.

It was F. Hansson from Gothenburg, Sweden, who posed the following problem:

> Using the 12 five-pieces of unit squares, cover the outside of TWO root-five edge cubes simultaneously.

This was printed February 1947. Two months later, Hansson's own solution was published, and he was the only solver. That solution reads as follows:

> Set a8b789c8, def8df9, de7ef6f5, c6cde5e4, f4def3e2, f2g1234. Put a root-5 line from junction of ef12 to junction of gh23. Construct a column of 4 root-5 squares on the base with one more to the right of third and one to left of fourth. The squares fold up into one of the covered cubes. For the other: b10b9abc8, bcd7d89, bcdef6, cde5bc4, d1234e1, ef4efg5. Root-5 base line at de12 to fg23, column of 4 squares on base, squares right and left of third.

Of course, the solution is a bit cryptic, but in conformity with the coding system in The Fairy Chess Review. A square of a chess board is identified by a letter and a number. Assume the origin of the board at bottom-left. The columns are marked a, b, c, ... from left to right, the rows are labeled 1, 2, 3 ... from bottom to top. Then the six codes for the first cube set X, U, W, Z, F, and Y, leaving holes at c7 and d6. For the second cube, the first five codes set T, V, I, N, and L. The very last code, ef4efg5, defines pentomino P, but it would overlap N. However, that is due to a mere printer's error: ef4efg5 should have been ef4efg3.

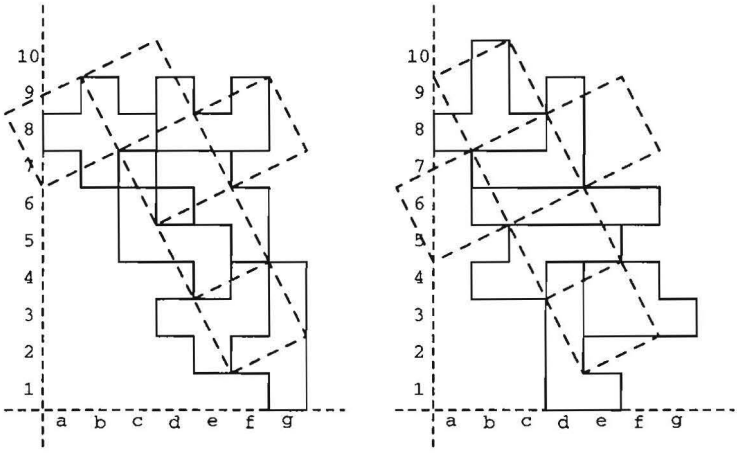



Figure 2: Hansson's cryptogramic solution decoded.

With the diagrams shown in Fig.2, it is easy to actually materialize the two cubes. It should be emphasized that the diagrams are only a means to define the cubes, and because of rotational symmetry of a cube there are 23 other diagrams possible, let alone its 24 reflections, and that holds for each of the pair of diagrams in our figure. For sure, the number of 4*24*24 = 2304 possible figures for one pair of cubes is asking for some standardization, in order to be able to compare or identify solutions.

So far as I know, there is no record of a second solution. None of common books on tilings and dissections mentions the problem. However, there exists a compilation of such problems from The Fairy Chess Review by G.P. Jelliss and a figure like our Fig.2 is given. Apparently, Jelliss was unable to cope with the printer's error in Hansson's solution. Anyhow, the diagram at right is in error: pentomino P is missing while pentomino W is duplicated.

So far so good, and one might be inclined to believe that Hansson's solution is the one and only one possible. However, that is far from true. Then comes the question, whether Hansson's problem is really hard to solve. That is not true either, as I will explain in the next section.

## 2 How to construct a cube covered by six out of the twelve pentominos

In a toy shop, or perhaps on the market as I did, you can get for little money a lot of wooden cubes, perhaps a bit rounded at corners and edges, as required by law for children's protection. Alternatively, you can make them or have them produced for you, unrounded and at little cost. You will need only two of these cubes.

On one face of the cube you start with drawing a straight line from the top-left corner to the centre of the opposite edge at right, and repeat this cyclicly for the remaining corners, so as to obtain Fig.3, and there you detect in the middle your own unit square. Perform this construction for all remaining faces of the cube, and you get 30 cells of unit squares distributed in a nice symmetric way about all six faces of the cube.

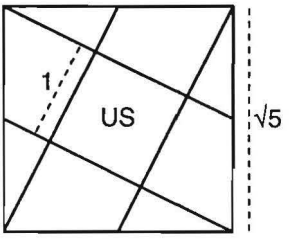



Figure 3: The Unit Square inside a root-5 square.

At this place we start our standardization. We unfold the cube surface onto the plane in the form of a cross, so that all faces become visible, although some are turned about. If the cube stands before you on the table, the upper square of the cross represents the back, the next three squares represent left, top, and right, while the remaining faces are front and bottom. For ease of reference I number the 30 cells, not completely at random but in a rather odd way as in Fig.4.

Pentomino X plays a special role in our problem. One of the cubes must contain X, and that cube is called the X-cube. First we want X to lie on top of the X-cube. Then look for the possible locations of the very central square of X. If it covers cell 5, we have a four-fold symmetry about the vertical axis. In this case the X-cube is said to be of class 2.

If the central square of X covers one of the cells 2, 4, 6, 8, we can force it to cover cell 4 by proper rotation about the vertical axis, and in that case the X-cube is said to be of class 1.

Finally, the central square of X may cover one of the four cells 1, 3, 7, 9 partly visible on the top face, but we then better take a neighbour face as top face, by rotation about a horizontal axis, and see that the resulting X-cube is nothing but of class 1.

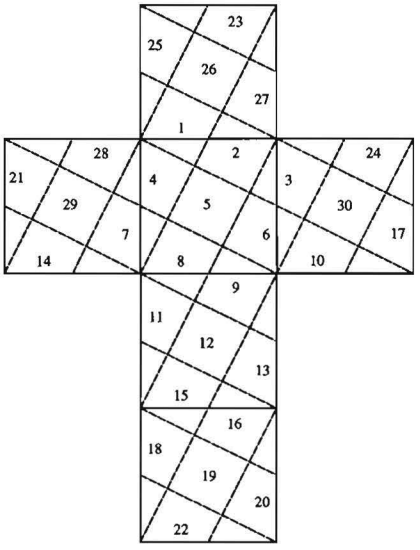



Figure 4: Cell numbering on the cube, here unfolded onto the plane.

What about mirror images of our cubes. We need not bother about that. The reflections of Figs 3 and 4 will never be used. The mirror image of a cube covered by pentominos is obtained, as it were, by refolding its unfold onto the plane in the wrong way, inside out.

It will be understood that you have made the 12 pentominos on the basis of your own unit square. Start experimenting and fold pentomino I on the cube such hat 5 cells are fully covered by I. Take any other pentomino, and do the same thing. You will surely discover a property of pentominos on the cube that does not hold in the plane.

To wit, two different pentominos in the plane can never cover the same set of cells in a rectangular pattern, but they can do so in the pattern on the cube. Cover the three cells at a corner and two other cells by a pentomino. You will then see that the pentomino gets, what I call, an internal edge. It is a bit deformed and not always easy to identify. Further observe that P, U, V can cover the same set of cells. This also holds for L, N, Y. Most queer is the F: it can cover the same cells as N, T, W, X, or Z. Finally, observe that W (which is symmetric in the plane), and Y (which is not) can cover the same cells.

In order to place and hold more than two pentominos on the cube, you need more fingers than you have on one hand. So, I drilled tiny holes at the centre of the cells, and with tiny nails, as used in fasten asphalt paper on wood, I can stick the pentominos to the cube. It works quite well, but you may need to renew your set of paper pentominos now and again!

Both cubes are easily covered by trial and error, or more systematically by backtracking.

The energetic puzzler can find tens, nay hundreds of solutions with the holes-and-nails gadget as indicated.

A few hints for the potential solver are perhaps in order. The pentomino of highest symmetry is X, and as we have seen, it can be placed essentially in only two ways on the first cube (the X-cube), modulo rotation: either folded around a corner with an internal edge (solutions of class 1) or wholly symmetrical on a face (solutions of class 2).

For class 1 the pentomino X occupies the cells 1, 4, 5, 7, and 28. Its internal edge is common to cells 1 and 28. In class 2, pentomino X fills cells 2, 4, 5, 6, and 8.

After setting X, choose the other five pentominos such that at least two pentominos can be placed and interchanged. Further, save P as long as possible for the second cube (which is called the complement cube).

My first example is class 1 with X+FINUV and its complement with pentominos LPTWYZ. For the first cube the two pentominos (X,F) can be interchanged, and so can the two (U,V). That makes 4 different covers for the first cube. A second cube can be easily made and it shows pentomino P folded around a corner. This P can be interchanged with either U or V in the first cube, so that the final result is 12 different solutions for Hansson's problem, all obtained in about seven minutes of puzzling!

A second example is of class 2 with X+FLTUV in the X-cube and INPWYZ in the complement cube. In the first cube the two pentominos (X,F) (no internal edges at start) can be interchanged (one with internal edge after the interchange), and so can (U,V) (both with internal edge before and after interchange). In the complement cube we can interchange (N,Y) and even (W,P) (in the last case, no internal edges before and after the interchange, and undeformed as in the plane), and so we have even 16 different solutions of Hansson's problem in one stroke.

Just for the fun of it I wanted to find a third example but without internal edges, so that all pentominos are undeformed. It took me quite a time (some hours) to obtain the pair, although I easily found more than one cover of the X-cube.

Hansson's solution and my own are shown and commented upon in the next section.

## 3 Diagrams enabling one to actually manufacture the cubes

The X-cube of Hansson involves pentominos X, Z, F, Y, W, and U in the order of the "first free cell". They successively cover the cells

| | | | | | |
|---|---|---|---|---|---|
| 1 | 4 | 5 | 7 | 28 | (X) |
| 2 | 3 | 10 | 27 | 30 | (Z) |
| 6 | 8 | 9 | 12 | 13 | (F) |
| 11 | 14 | 15 | 16 | 29 | (Y) |
| 17 | 20 | 23 | 24 | 26 | (W) |
| 18 | 19 | 21 | 22 | 25 | (U) |

The corresponding pentomino names are shown (Fig.5), and these names are placed in the cells with lowest cell number, henceforth called the origin of the pentomino in question. So, 1 is the origin of X, ... , 18 is the origin of U.

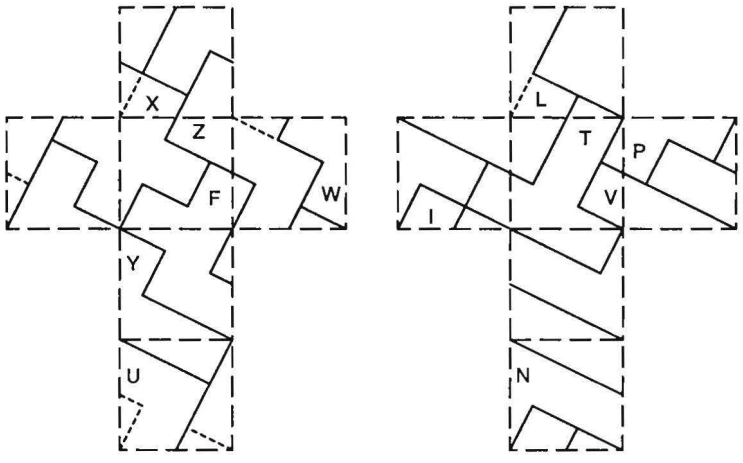



Figure 5: Hansson's cubes unfolded onto the plane. They are of type FUWYZ. The X-cube, of class 1, has four pentominos with internal edge (X,U,W,Z). The complement has one (L). Internal edges are indicated by dotted lines.

It is recommended that you fill the 30-element array of cells step by step so as to get the succession of letters ("letter code")

X Z Z X X F X F F Z Y F F Y Y Y W U U W U U W W U W Z X Y Z

and to convince yourself that the X-cube is covered without hole or overlap indeed.

The complement cube contains pentominos L, T, P, V, I, N which cover the cells

| | | | | | |
|---|---|---|---|---|---|
| 1 | 4 | 22 | 25 | 28 | (L) |
| 2 | 5 | 7 | 8 | 9 | (T) |
| 3 | 23 | 24 | 26 | 27 | (P) |
| 6 | 10 | 11 | 12 | 13 | (V) |
| 14 | 15 | 16 | 17 | 30 | (I) |
| 18 | 19 | 20 | 21 | 29 | (N) |

The letter code of this complement cube is

L T P L T V T T T V V V V I I I I N N N N L P P L P P L N I

and, once more, take the trouble to check it on correctness. As a bonus, you might detect a possible way to represent a solution in computer terms, and even discover an effective procedure of backtracking with the computer along a very large number of 5-cell pentomino patterns and positions on the cube surface.

For my own first example refer to Fig.6 and its legend. The origins of the pentominos are not now indicated, but I will use the figure to illustrate a way of identification and describing a solution in terms of letters arranged in an array of two dimensions.

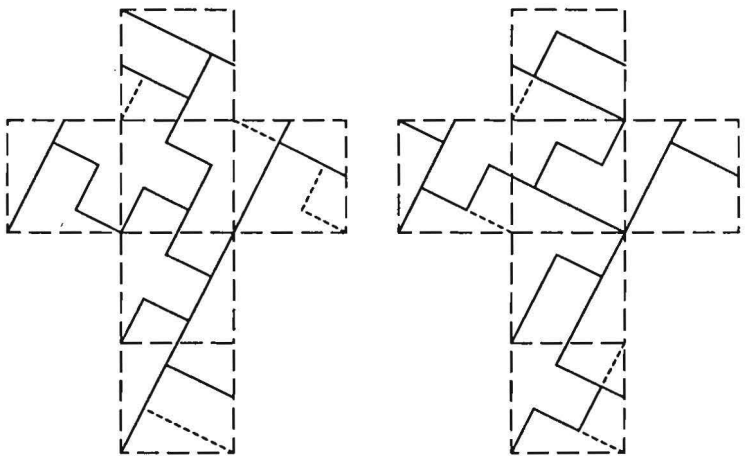


Figure 6: Two simultaneous cubes of type FINUV. The X-cube, of class 1, has four pentominos with internal edge (X,F,U,V) and its complement has also four (P,T,W,Z). In the first cube X and F can be interchanged, and the same holds for U and V. In addition, P in the complement can be interchanged with either V or U in the X-cube. Thus we have 12 pairs of simultaneous cubes: 4 of type FINUV, 4 of type FINPU, and 4 of type FINPV, and their X-cubes are all class 1.

On the base of the letter codes introduced above, it is easy to place all pentomino names in the appropriate cells as shown by Fig.4. The result for Fig.6 is

```
    V                 Z
  I I N   V         Z L L   Z
    X N N U U         W W L P P
  X X X N U         W W L L P
I F X F N         Y W T T T
  F   F F U         T   T Y P
        I U             Y P
        I V V           Y Y Z
          V               Z
```

where, as in Fig.2, the layout of the cross (implied) is somewhat inclined.

For further comments on my other examples, let it suffice to refer to the extensive legends of Figs 7 and 8.

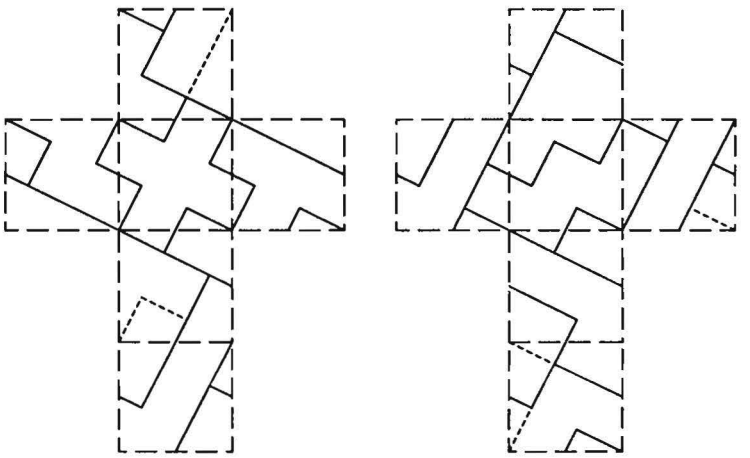


Figure 7: Two simultaneous cubes of type FLTUV. The X-cube, of class 2, has four pentominos with internal edge (X,F,U,V), and its complement has also four (N,Y,P,W). In the X-cube X and F can be interchanged, and so can U and V. In its complement N and Y can be interchanged, while the block (W,P) can be flipped over. This leads to 16 solutions of Hansson's problem. They are all type FLTUV and class 2.

## 4 Invoking the computer

At the time when I learned about Hansson's problem and his one solution, my puzzle became finding the number of different solutions, by computer of course. Surprisingly, Hansson's solution turns out to be one amongst millions, because the number of solutions modulo rotation and reflection is 5,626,985. So, don't expect me to publish a catalogue. Instead, let me indicate some details of my computer approach.

First of all, I should mention that the backtrack process is not in order of alphabetical names. I use the order X, U, T, V, Z, W, I, F, N, Y, L, P. As explained above, pentomino X has only two positions fixed on the X-cube, and will be left alone. Pentomino U can be placed in 96 different ways on an empty cube surface. These are numbered from 1 to 96. Similarly, the T has 120 ways, numbered from 97 to 216, ..., the P has 168 ways, numbered from 1549 to 1716. All this information is stored in a matrix of 5 columns and 1716 rows of cell labels, ordered to increasing value both horizontally and vertically.

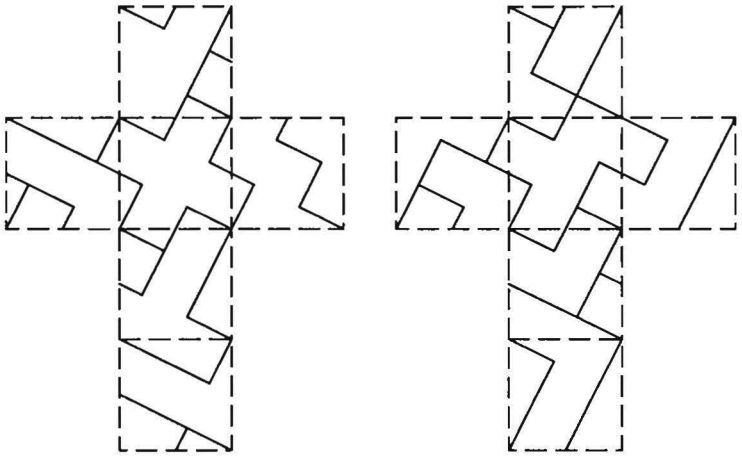


Figure 8: Example of two simultaneous cubes without internal edge, with X-cube necessarily of class 2. It is of type (X+)LPTWY and its complement contains FINUVZ. It is a real beauty with no deformed pentominos. They are just wrapped over the cube and their unit squares show the same neighbouring and connectivity as they do in the plane.

It is quite a job to construct this matrix, but the computer can help us. If a pentomino is placed at one position on one face of the cube, 23 other positions can be determined by rotation. To obtain all codes we have to start from a judiciously chosen set of, in general, five or ten hand-made codes depending on wether the pentomino is or is not invariant under reflection. Pentominos U and P need only four and seven, respectively. Every pentomino has its own idiosyncrasies, so to say, and that is what you learn in the process of constructing the matrix.

In the usual pentomino-packing problems, the codes are more or less computed on the fly, because the shifts occur in the directions of a rectangular grid. That would be very time consuming in our problem, if possible at all, and so we make the codes statically, that is, before the backtrack begins. The matrix is stored on disk in five files of 1716 integers each and ready for input in our programmes.

I now come to the representation of a solution in computer terms. For the X-cube we use an ordered set of five integers and for its complement we use an ordered set of six integers, all numbers in the range 1 - 1716. For example, Hansson's solution is represented and identified by

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 362 | 733 | 1239 | 1285 | 87 | | 1338 | 126 | 1600 | 275 | 628 | 1068 |
| Z | F | Y | W | U | | L | T | P | V | I | N |

where we have added the corresponding pentomino names, and class 1 is implied.

Programmes were written in BASIC (compiled). One of the first determined all 34061 solutions of class 1, which were stored on hard disk, and this in less than six minutes time on my 486 DX2-66. The second run for class 2 found 13044 cubes but the number of different cubes is 13044/4 = 3261 mod rotation. By sorting, merging, skipping and other operations, a table was generated of the number c1, c2 of X-cubes of class 1, 2; the number c3 of their complements; the number [(c1+c2)*c3] of simultaneous cubes; all this in 462 lines ordered alphabetically according to the type, that is, the string of 5 pentominos of the X-cube beyond X.

As the table reveals, for any possible partition of the twelve pentominos into two sets of six there are a lot of solutions. This holds even on the restriction that the X-cube is of class 1, and mind you, 462 is the combinatorial "11 over 5", the number of ways you can take 5 pentominos out of 11. Some of the numbers c2 are zero, so that X-cubes of class 2 are not always possible for a given partition.

The lowest number of simultaneous cubes is 2408, of type FIPUV with X-cube of class 1 (7 solutions) and its complement (344 solutions). Class 2 of this type is empty. The other extreme is type TZWFN: 177 cubes of class 1, 10 cubes of class 2, and 336 of their complements. They represent 62832 solutions of Hansson's problem.

I wrote programmes for counting and selecting upon the number of pentominos with internal edges, especially with no such edges at all. The latter can only occur in class 2, because the X has an internal edge in class 1. I found 687 solutions. Most interesting are the types for which the X-cube and its complement are both unique. They are very hard to find, if you do not know the partition, as I suggested in the case of my third example. On the other hand, I was very lucky to obtain just one of these scarce uniques. There are 18 others. I will give you all 19 types, so that you may find the solutions. It is not too difficult by doing so without the computer.

The beauties are the types

**FILPT, FILYZ, FIPTY, FIPVY, FLNPW, FLTUY, FLTYZ, FPTVZ, FPVWY, ILNTU, LNPWZ, LNTUY, LPTUZ, LPTWY, LTUVY, NPTUW, NPVWY, NPVYZ, NTVYZ.**

Lastly, I made programmes for demonstration purposes, with each pentomino its own colour in movie or still pictures similar to the unfold of the cubes in Figs 5 to 8.

## References

Solomon W. Golomb, Polyominoes, Charles Scribner's Sons, New York, 1965.

F.H. Hansson, Problem/Solution No. 7124. The Fairy Chess Rev. 6, 71/81, 1947.

G.P. Jelliss, Dissection Problems in PFCS/FCR, Summary of results in date order, published by the author, St Leonards on Sea, U.K., 1986.

# Patronen van priemgetallen in kwadratische lichamen

Th.J. Dekker

Beste Frans,

Bij gelegenheid van je afscheid van de TUE wil ik in de eerste plaats mijn en jouw gedachten laten teruggaan naar de tijd dat wij samen op de Rekenafdeling van het MC werkzaam waren. Ik wil hierbij vooral aandacht schenken aan die gelegenheden waarin wij samen werkten of ons werk althans een enigszins nauw verband had.
Verder geef ik een voorbeeld van een programma en enige ermee geproduceerde patronen van priemgetallen in kwadratische lichamen. Zoals je weet, is dit de laatste tijd een hobby van me.

## 1 Terugblik

Ik ben in oude jaarverslagen van het MC gedoken om mijn geheugen op te frissen. In jaargang 1962 staat vermeld dat je op 1 september werd aangesteld op de Rekenafdeling van het MC. Ik lees in het wetenschappelijk verslag het volgende:

> "De heer Kruseman Aretz begon na zijn aanstelling met het ontwerpen van programma's voor multilengte-arithmetiek in ALGOL-60. Bovendien onderzocht hij o.a. de mogelijkheid de vertaler een compacte gecodeerde objectband te laten produceren."

Dit was een bescheiden begin. Het betreft dan ook een periode van slechts vier maanden. In 1963 wordt onze eerste samenwerking gemeld:

> "De heer Dekker vervaardigde verder enige machinecode-procedures voor het ponsen via de snelle bandponser van getallen en teksten in Flexowriter-code, analoog aan de bestaande typroutines. Zij werden voor gebruik gereed gemaakt in samenwerking met de heer Kruseman Aretz."

Ik kan mij nog herinneren hoe ik hiervoor de door Edsger Dijkstra bedachte typcodes voor getallen bekeek en constateerde dat er gelukkig nog ergens een bit over was, waarmee ik iets van belang voor het ponsen kon aangeven. Verder vermeldt deze jaargang onder meer:

> "Laatstgenoemde medewerker slaagde erin de bestaande in gebruik zijnde ALGOL 60-vertaler voor de X1 verder te ontwikkelen en te automatiseren. Een van de resultaten, de zgn. universele vertaler ALDS-7, werd in mei operationeel. De nieuwe versie verschilde met de oude in principe uitsluitend in de code van de objectbanden die met name door het gebruik van de zgn. Fano-code en het benutten van zeven sporen op de band, voor minder dan de helft konden worden ingekort, waardoor aan vertaaltijd belangrijk kon worden gewonnen.

De vertaaltijd werd blijkbaar voor een belangrijk deel bepaald door de tijd die voor het ponsen nodig was. De zinsnede "voor minder dan de helft" betekent letterlijk dat *meer dan de helft* aan lengte over bleef, terwijl "konden worden ingekort" eerder de indruk wekt dat bedoeld is dat een inkorting *tot minder dan de helft* bereikt kon worden. Misschien kun jij, Frans, hierover nog uitsluitsel geven.

> "Bovendien bood deze universele vertaler de mogelijkheid voor omschakeling op de langzame bandponser en de snelle ponser (resp. 25 en 300 symbolen / sec.)."

Hier wordt weer de snelle bandponser genoemd. Deze belangrijke aanwinst heeft blijkbaar ons beiden, doch vooral jou, bezig gehouden. Verder lees ik nog:

> "In het najaar kwam de zgn. load-and-go ALGOL 60-vertaler gereed. De verwerking van ALGOL programma's kon hierdoor belangrijk worden vereenvoudigd, omdat het object-programma (resultaat van de vertaalarbeid) zonder eerst te zijn uitgeponst, nu direct voor uitvoering in het geheugen kon worden opgeslagen."

Dit was dus wel een vruchtbaar jaar. Het is ontroerend te merken, hoe de redacteur van het jaarverslag probeert de zaak voor niet-ingewijden duidelijk te maken, zoals bijvoorbeeld de uitleg: "objectprogramma (resultaat van de vertaalarbeid)".

In jaarverslag 1964 is sprake van een interne werkgroep, bestaande uit de heren Zonneveld, Kruseman Aretz, Nederkoorn, Van de Laarschot en Barning, die zich uitvoerig bezig hield met de verdere ontwikkeling van een ALGOL 60-systeem voor de toekomstige X8-rekeninstallatie. Deze werkzaamheden bestonden o.a. in: het programmeren van standaardfuncties voor de X8.
Dit brengt je op numeriek terrein. Ik herinner mij dat je hiermee bezig was en dat we van gedachten hebben gewisseld over optimale numerieke benaderingen van standaardfuncties. Verder lees ik:

> "De heer Kruseman Aretz vervolmaakte de load-and-go vertaler voor de X1, die in het najaar van 1963 voor gebruik gereed was gekomen. Hij verbeterde een aantal complexroutines, zodat nu bijv. de indexer op lezen en schrijven buiten array-grenzen controleert."

Deze vervolmaking had een onverwacht gevolg waar ik mee te maken had. De (door mij geschreven) ALGOL 60 procedure 'inprod' maakte namelijk op slimme wijze gebruik van de indexer om beginadres en adresverschil van opeenvolgende elementen van de arrays te bepalen. Dit zou in de vervolmaakte vertaler in strijd (kunnen) komen met de controle op array-grenzen. Toen ik in augustus 1964 terug kwam uit Amerika, werd mij daarom gevraagd om zo spoedig mogelijk de procedure inprod aan te passen, zodat deze veilig met de genoemde array-grenzen controle te gebruiken was.
Dat je deze vervolmaking van de vertaler belangrijk vond, en wij allemaal eigenlijk, blijkt uit de treffende passage die je in je inaugurele rede hieraan hebt gewijd. Ik citeer het laatste deel van deze passage:

> "Ontelbare programma's sneuvelden bij de nieuwe toets, waaronder menig programma dat steeds tot volle tevredenheid van de gebruiker gewerkt had. Het regende dan ook protesten in plaats van dankbetuigingen, en tot onze stomme verbazing kwamen er verzoeken binnen tot vergunning het oude systeem te mogen blijven gebruiken."

Jaargang 1965 vermeldt de volgende, voor ons beiden belangrijke wijziging:

> "De sous-chef dr. J.A. Zonneveld, aan het Mathematisch Centrum verbonden sedert 1948, verliet per 1 augustus de dienst wegens het aanvaarden van een functie in de industrie. Met ingang van deze datum werden dr. Th. J. Dekker en dr. F.E.J. Kruseman Aretz benoemd tot sous-chef van respectievelijk de numerieke sectie en de programmeersectie van de afdeling."

Ik herinner mij dat wij ook in die tijd als collega's sous-chefs heel prettig hebben samengewerkt. Verder lees ik omtrent jouw werk in dat jaar:

> "Van de vertaler voltooide de heer Kruseman Aretz de versie geschreven in ALGOL 60. Deze versie kon uitvoerig getest worden met een bestaand ALGOL 60-systeem voor de X1 (de MC II vertaler, geschreven door de heren Nederkoorn en Van de Laarschot)."

Dit was, denk ik, een belangrijke mijlpaal in jouw werk op gebied van vertalers.
Jaargang 1966 deelt mede:

> "Dr. F.E.J. Kruseman Aretz, sous-chef van de programmeersectie, werd per 1 september benoemd tot bijzonder hoogleraar aan de Universiteit van Amsterdam vanwege de Stichting voor Hoger Onderwijs in de Toegepaste Wiskunde."

Verder blijkt dat het ALGOL 60-systeem voor de X8 in april werd afgeleverd en in de loop van mei in bedrijf kwam. Ook hier wordt een stukje samenwerking van ons gemeld:

> "Samen met Prof. Kruseman Aretz onderzocht de heer Dekker in hoeverre dubbele lengte-optelling en -vermenigvuldiging op de X8 gerealiseerd konden worden. Het bleek mogelijk dit te doen met behulp van floating-point operaties, zodat de dubbele lengte-bewerkingen (voor het test-stadium) in ALGOL 60 geformuleerd konden worden."

De eerste toepassing hiervan was, zoals je je wellicht herinnert, een dubbele-lengte procedure voor het berekenen van inwendige producten.

In jaargang 1967 wordt medegedeeld dat je op 24 april het ambt aanvaardde van bijzonder hoogleraar in de programmeringsmethoden in de Numerieke Wiskunde met het uitspreken van een rede, getiteld: "Vallen en opstaan".

In jaargang 1968 is sprake van een nieuw samenwerkingsproject met de Rijksuniversiteit Utrecht. Hierover wordt verder gezegd:

> "Besloten werd tot het oprichten van een gezamenlijke werkgroep, waarin van het Mathematisch Centrum naast Prof. Kruseman Aretz de heren Van der Grinten, Ten Hagen, Van der Velden en, incidenteel, Koksma deelnamen. In deze werkgroep was ook het Natuurkundig Laboratorium van NV Philips Gloeilampen-fabrieken vertegenwoordigd."

De verlokkingen van het bedrijfsleven kon je na deze contacten blijkbaar niet langer weerstaan. Jaargang 1969 meldt dan ook:

> "De programmeersectie stond tot 1 augustus onder leiding van Prof. dr. F.E.J. Kruseman Aretz, die een betrekking aanvaardde in het bedrijfsleven, en daarna van dr. R.P. van de Riet."

Hiermee kwam een einde aan een interessante periode in jouw wetenschappelijke loopbaan, een periode waarin wij als collega's heel plezierig hebben samengewerkt.

## 2 Patronen van priemgetallen in kwadratische lichamen

Ik geef een voorbeeld van een programma dat (normen van) priemgetallen van een kwadratisch lichaam berekent en een patroon van priemgetallen tekent. Voor definities, eigenschappen, stellingen, voorbeelden en andere bijzonderheden verwijs ik naar [1]. Belangrijkste verschil met programma en voorbeelden in genoemde publicatie is dat het hier gegeven programma werkt voor willekeurig gegeven radicand $d$ waarvoor lichaam $Q(\sqrt{d})$ eenduidige factorisatie heeft (hier eenvoudigheidshalve beperkt tot het geval dat $d$ priem en congruent 1 modulo 4 is).

Mijn keuze viel op de programmeertaal Pascal, omdat deze op Macintosh computers beschikbaar is en het MacPascal systeem met de QuickDraw library ons in staat stelt de plaatjes te tekenen. Bovendien zijn set-types in Pascal heel geschikt om de zeef-algoritme te formuleren.

Er zijn negen complexe (dwz. $d$ is negatief) kwadratische lichamen met eenduidige factorisatie. Vijf van deze zijn (norm-) Euclidisch, de vier overige niet. Voor zes van deze complexe lichamen zijn plaatjes (in andere vorm) opgenomen in [1]. Hier worden plaatjes

afgedrukt voor de vier niet-Euclidische lichamen met $d = -19, -43, -67, -163$; de laatst genoemde drie worden hier voor het eerst gepubliceerd.

Voor goed begrip van de plaatjes merk ik nog het volgende op. De afbeelding van een kwadratisch lichaam in het platte vlak is zodanig dat rationale getallen op de x-as en rationale veelvouden van $\sqrt{d}$ op de y-as terecht komen. De plaatjes zijn hier begrensd tot getallen met een norm kleiner dan een gegeven grens 'maxnorm', en hebben daardoor een ellipsvorm.

[1] T.J. Dekker, Prime numbers in quadratic fields; *CWI Quarterly*, vol 7, nr 4 (*Special issue on Computational Number Theory*), pp. 367 - 394;
ook verschenen als: Technical Report CS-94-10, Department of Computer Systems, Faculty of Mathematics and Computer Science, University of Amsterdam, June 1994.

```
program QuadraticPrimesPicture (output);  { Th. J. Dekker, 1 August, 1995.  }
{ Program on the occasion of the retirement of F.E.J. Kruseman Aretz at TUE }
{ On input one must give a prime radicand congruent 1 modulo 4 such that    }
{ the corresponding quadratic field has the unique factorization property.  }
{ This program then calculates, (the norm of) primes of this field by means }
{ of a sieve algorithm, and draws a picture of these primes.                }

 const
  maxnorm = 16383;                  { norms are in subrange 0..maxnorm    }
  maxperiod = 400;                  { maximal period quadratic character  }

 type
  intrange = 0..maxnorm;
  intset = set of intrange;
  introw = array[0..maxperiod] of integer;

 var
  radicand, period, c2, bmax : integer;
  qstart, qchars, primenorms : intset;
  deltac : introw;
```

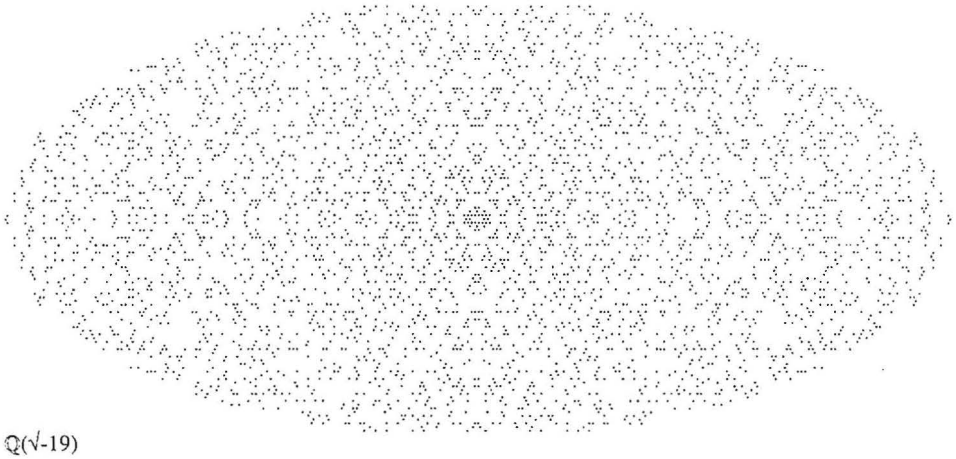
```
{ Program QuadraticPrimesPicture (continued), Th.J. Dekker, 1 August, 1995, }
{ on the occasion of the retirement of F.E.J. Kruseman Aretz at TUE.        }

 procedure sieve (max : intrange;
         start : intset;
         function next (v : integer) : integer;
         var prinorms : intset);
{ This procedure calculates a set containing the norms <= max of primes of  }
{ a quadratic field having the unique factorization property. The method     }
{ used is a sieve algorithm applied to the set containing the elements of    }
{ start and the numbers in [2..max] obtained by repeatedly applying next,    }
{ starting from 1. This should generate the odd numbers of quadratic         }
{ character +1, whereas start should contain the prime divisors of           }
{ discriminant D and number 2 if D mod 8 = 1 or number 4 if D mod 8 = 5.     }
{ The result is delivered in prinorms.                                       }
  var
   i, m, maxim : integer;
 begin  { Form superset of prime norms }
  prinorms := start;
  i := next(1);
  repeat
   prinorms := prinorms + [i];
   i := next(i)
  until i > max;
{ Now prinorms is ready as starting set for the sieve process. }
  i := next(1);   { Start sieve process }
  maxim := max div i;
  repeat
   if i in prinorms then    { delete multiples of i: }
    begin
     m := i;
     repeat
      prinorms := prinorms - [i * m];
      m := next(m)
     until m > maxim        { i.e. until i * m > max }
    end;
   i := next(i);
   maxim := max div i
  until i > maxim;           { i.e. until i * i > max }
 end; { sieve }
```

```
 procedure quadres (p : integer;
         var qset : intset);
{ For given odd prime p, qset := set of odd quadratic residues modulo 2p.   }
  var
   pdouble, x : integer;
 begin
  pdouble := 2 * p;
  qset := [1];
  for x := 2 to (p - 1) div 2 do
   qset := qset + [sqr(2 * x - 1) mod pdouble]
 end; { quadres }

 procedure quadcharpos (d : integer;
         var start, qcharset : intset);
{ Given: square-free integer d, radicand of unique factorization field.     }
{ For simplicity, radicand d is assumed to be a prime congruent 1 modulo 4. }
{ start:= set to start sieve, which contains abs(d) and number 2 or 4,      }
{ depending on wether d is congruent 1 or 5 modulo 8, respectively;         }
{ qcharset:= set of numbers modulo 2|d| of positive quadratic character.    }
  var
   a, even : integer;
 begin
  a := abs(d);
  quadres(a, qcharset);
  even := abs((d - 5) mod 8 div 2 + (d - 1) mod 8);   { 2 or 4 }
  start := [even, a];
 end;{ quadcharpos }

 procedure makedelta (qcharset : intset;
         var deltac : introw);
{ deltac:= differences between successive odd numbers of positive character }
{ to be used by qnext                                                       }
  var
   c, cnext : integer;
 begin
  cnext := period + 1;
  for c := period downto 1 do
   if c in qcharset then
    begin
     deltac[c] := cnext - c;
     cnext := c
    end
 end; { makedelta }
```

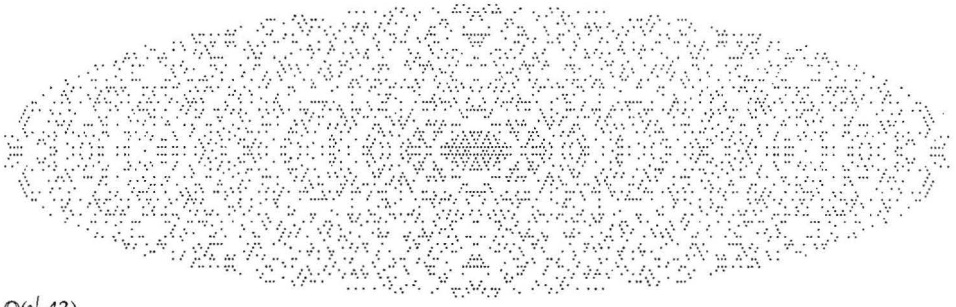
```
{ Program QuadraticPrimesPicture (continued), Th.J. Dekker, 1 August, 1995, }
{ on the occasion of the retirement of F.E.J. Kruseman Aretz at TUE.        }

 function qnext (k : integer) : integer;
{ returns next odd number of positive character for the field considered.   }
 begin
  qnext := k + deltac[k mod period]
 end; { qnext }

 function qnorm (a, b : integer) : longint;
{ returns the norm of quadratic number a + b * (1 + sqrt(radicand)/2.       }
 begin
  qnorm := abs(sqr(a) + a * b + c2 * sqr(b))
 end; { qnorm }

 procedure drawprime (a, b : integer);
{ Draws a block of size dx by dy on a position determined by a and b        }
  const
   x0 = 256;            { origin for pixel coordinates    }
   y0 = 160;
   hx = 1;              { number of pixels in x-direction }
   hy = 2;              { number of pixels in y-direction }
  var
   x, y : integer;
 begin
  x := a * hx + x0;
  y := b * hy + y0;
  MoveTo(x, y);
  LineTo(x, y);
 end; {drawprime}

 procedure drawpicture (var primes : intset);
{ Draws picture of primes in region bounded by given constants.             }
  const
   abmax = 255;         { width in number of half steps }
  var
   a, b : integer;
 begin
  for b := -bmax to bmax do
   for a := -(abmax + b) div 2 to (abmax - b) div 2 do
    if qnorm(a, b) in primes then
     drawprime(2 * a + b, b);
 end; { primespicture }
```

```
 procedure setrects;
  var
   r : Rect;
 begin
  SetRect(r, 0, 20, 600, 400);
  SetTextRect(r);
  SetDrawingRect(r);
 end;   { setrects }

begin
{ Calculates and drasw picture of primes in field Q (sqrt (radicand) )     }
 setrects;
 ShowText;
 writeln;
 write('  radicand   =      ');
 read(radicand);                  { read radicand and print parameters    }
 period := 2 * abs(radicand);
 c2 := (1 - radicand) div 4;                        { coefficient for qnorm }
 bmax := trunc(2 * sqrt(maxnorm / abs(radicand))); { picture height        }
 quadcharpos(radicand, qstart, qchars);             { quadratic character   }
 makedelta(qchars, deltac);                         { make array for qnext  }
 sieve(maxnorm, qstart, qnext, primenorms);         { calculate prime norms }
 ShowDrawing;                                       { shows drawing window  }
 drawpicture(primenorms);                           { draws primes picture  }
 SaveDrawing('zd_pict');
end.
```

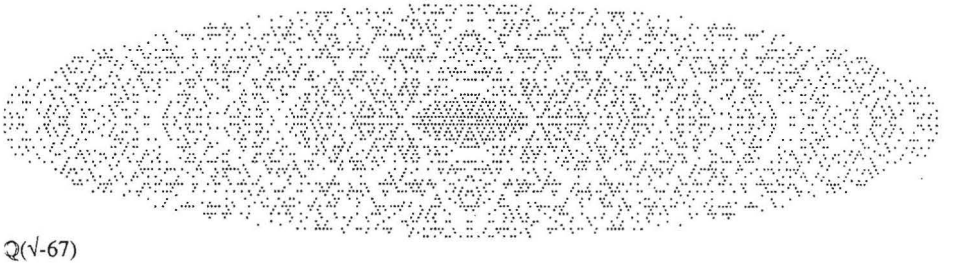
**Priemgetallen in complexe kwadratische lichamen** norm < 16384



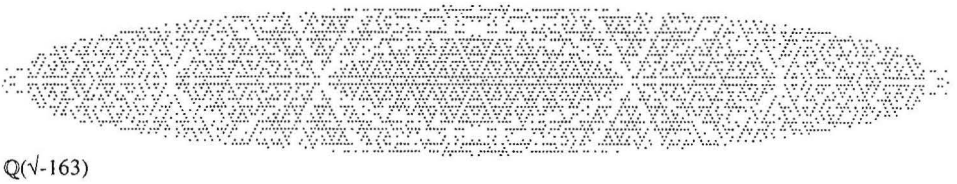
$\mathbb{Q}(\sqrt{-19})$



$\mathbb{Q}(\sqrt{-43})$



$\mathbb{Q}(\sqrt{-67})$



$\mathbb{Q}(\sqrt{-163})$

# On some very binary patterns

Edsger W. Dijkstra

In his last manuscript "A collection of nice problems", Jan L.A. van de Snepscheut described the following one-person game. There is a table with four coins in the positions north, east, south and west, and the player's goal is to get them all in the same orientation, i.e. all heads or all tails. The player never sees the coins but will be informed as soon as he has reached his goal. A move consists of the player mentioning one or more positions in which the coins will then be turned over; the complication is that, prior to each move, the table is rotated by a multiple of 90°; the multiple remains unknown to the player and may vary from move to move.

The version of the problem that Rutger M. Dijkstra told me after the 3rd International Conference on the Mathematics of Program Construction at Kloster Irsee, Germany —where it had apparently circulated— had been sanitized in that the player's goal had been simplified to "all coins in a given orientation, say heads". Moreover, Rutger had generalized the number of coins from 4 to any power of 2, a generalization that was extremely helpful because it was an irresistible invitation to look for a nicely recursive solution.

There are three reasons for writing this note. Firstly, when I had designed a winning strategy, I did not know how to present it nicely. Secondly, my solution's correctness implied a theorem (of sorts) that I did not yet know how to formulate decently. Finally, there is my hope that Frans E.J. Kruseman Aretz, to whom this note is dedicated, will appreciate my efforts.

* * *

In order to avoid all ambiguity, let us formulate the game as a program of which termination has to be guaranteed.

We introduce the type BV of bit vectors of length $n$ ($n \geq 1$; in Jan's original problem, $n = 4$), and two constants ZEROS and ONES of type BV (viz. the sequence of $n$ 0s and the sequence of $n$ 1s respectively). On arguments of type BV, we denote the bit-wise addition modulo 2 (also known as "the carry-less sum") by an infix $\dot{+}$; as any sum, it is symmetric and associative. Finally, we introduce a function

$$\text{rot}: \quad \text{int} \to (\text{BV} \to \text{BV})$$

of which we know that rot.$i$.bv is some rotation of bv. We have no further knowledge how rot.$i$ depends on $i$ (which will act below as move number).

In the following program block, the state of the table with its coins is represented by an initialized global variable

$$t : \quad \mathrm{BV} .$$

The player's strategy is represented by a global function

$$\mathrm{ps}: \quad \mathrm{int} \rightarrow \mathrm{BV} ,$$

where the 1s in bit vector ps.$i$ identify the positions in which the coins are turned over in the $i$-th move.

In the original problem we were interested in the termination of

(0)
```
|[ var i : int; i := 1
 ; do t ≠ ZEROS ∧ t ≠ ONES →
        t := rot.i.t ∔ ps.i; i := i + 1
   od
]|
```

but in the rest of this note we deal with the "sanitized" version

(1)
```
|[ var i : int; i := 1
 ; do t ≠ ZEROS →
        t := rot.i.t ∔ ps.i; i := i + 1
   od
]| .
```

**Remark.** Note that in simplifying the guard we have weakened it. Hence, termination of the sanitized version (1) implies termination of the original program (0). (End of Remark.)

The challenge now is to design such a function ps that (1) terminates independently of the initial value of $t$ and independently of how rot.$i$ depends on $i$.

The purpose of our next massaging is to facilitate keeping track of the identity of the coins so as to ease tracing the influence of a change in initial value of $t$.

We observe that as far as the identity of the coins turned over in a move is concerned, the effect of how far the table has been turned clockwise can also be obtained by the corresponding anti-clockwise turn of the player's move. More precisely: because

(i) rotation distributes over bit-wise addition

(ii) ZEROS is a fixpoint of any rotation, and

(iii) we have to guarantee termination whatever rot's dependence on its first argument,

our concerns about termination of (1) are equivalent to the concerns about termination of

(2)
$$
\begin{array}{l}
[\![\ \textbf{var}\ i : \text{int};\ i := 1 \\
\quad ;\ \textbf{do}\ t \neq \text{ZEROS} \rightarrow \\
\qquad t := t \dot{+} \text{rot}.i.(\text{ps}.i);\ i := i + 1 \\
\quad \textbf{od} \\
]\!]\ ,
\end{array}
$$

viz. for all rot, and for all initial $t$. The gain of this rewriting is that it makes it easier to separate these two universal quantifications.

Consider now a terminating computation of (2) and let $fi$ be the final value of $i$. Because $\dot{+}$ is associative and ZEROS has the property

$$u \dot{+} v = \text{ZEROS} \ \equiv\ u = v\ ,$$

we can conclude that the carry-less sum of $\text{rot}.i.(\text{ps}.i)$ for $1 \leq i < fi$ equals the initial value of $t$. Termination of (2) for *any* initial value of $t$ is thus equivalent to stating that during execution of

(3)
$$
\begin{array}{l}
[\![\ \textbf{var}\ i : \text{int},\ s : \text{BV};\ i, s := 1, \text{ZEROS} \\
\quad ;\ \textbf{do}\ i \neq N \rightarrow \\
\qquad s := s \dot{+} \text{rot}.i.(\text{ps}.i);\ i := i + 1 \\
\quad \textbf{od} \\
]\!]
\end{array}
$$

with sufficiently large $N$, variable $s$ successively takes on all possible values of type BV. Because type BV comprises $2^n$ distinct values, "sufficiently large" is at least $2^n$. As we will show, we don't need a larger value; for the rest of this note we choose

$$N = 2^n \ .$$

Now our challenge is to define $N-1$ values ps.$i$ $(1 \leq i < N)$ such that, in the execution of (3), for any rotation function rot, variable $s$ successively takes on all its $N$ possible values. We shall solve the problem for $n = 1$, and shall show how, from a solution ps for $n$, we can construct a solution ps' for $n'$, provided $n' = 2n$. We thus solve the problem for $n$ an arbitrary power of 2.

**Remark.** Since —as the reader may verify— the problem is unsolvable for $n = 3$, there is probably no point in trying to be more ambitious. (End of Remark.)

For $n = 1$, $N = 2$ and ps.1 $= 1$ does the job: rotating a bit vector of length 1 leaves it unchanged and in the one and only step of the execution of (3), the single bit $s$ is inverted.

For $n' = 2n$, type BV' comprises the $N'$ bit vectors of length $n'$ with $N' = 2^{n'}$ or $N' = N^2$. Variable $s'$ is of type BV' and function ps' : int $\rightarrow$ BV' is to be defined under the hypothesis that ps solves the problem for bit vectors of length $n$.

For the purpose of our discussion we introduce variables $u, v$ : BV such that

$$s' = u +\!\!+ v$$

(where $+\!\!+$ denotes catenation). The above relation implies that $s'$ determines the pair $(u, v)$ and vice versa. We observe that this also holds for $s'$ and the pair $(u \dot{+} v, v)$. (This observation captures the core of our solution.) In our solution, ps is used at two levels, so to speak. At one level —see below at "$r = 0$"— ps is used to generate all possible values for $u \dot{+} v$; at the other level —see below at "$r \neq 0$"— ps is used to generate, for each value of $u \dot{+} v$ and under invariance of it, all possible values for $v$. The two levels thus generate all $(u \dot{+} v, v)$ pairs, i.e. all $s'$-values. More precisely, with $1 \leq q * N + r < N^2 \wedge 0 \leq r < N$, ps' is defined by

$$\begin{aligned} \text{ps}'.(q * N + r) \ = \ & \textbf{if} \ \ r = 0 \rightarrow \text{ZEROS} +\!\!+ \text{ps}.q \\ & [] \ \ r \neq 0 \rightarrow \text{ps}.r +\!\!+ \text{ps}.r \\ & \textbf{fi} \quad . \end{aligned}$$

We have to check that the rotations in (3') of bit vectors of length $2n$ can be interpreted as rotations of bit vectors of length $n$, so as to make ps applicable as generator of all BV

values. On the first level, the symmetry of $u \dot{+} v$ takes care of that, on the second level the symmetry of $\mathrm{ps}.r \dot{+} \mathrm{ps}.r$ does so.

For the sake of illustration, we tabulate ps for small values of $n$ in Figure 0. (For $n = 8$, we give only the beginning of the table, which has 255 entries.)

| $n=1$ | ps | gr |
|---|---|---|
| | 1 | |

| $n=2$ | ps | gr |
|---|---|---|
| | 11 | $y$ |
| | 01 | $x$ |
| | 11 | $y$ |

| $n=4$ | ps | gr |
|---|---|---|
| | 1111 | $yy$ |
| | 0101 | $yx$ |
| | 1111 | $yy$ |
| | 0011 | $xy$ |
| | 1111 | $yy$ |
| | 0101 | $yx$ |
| | 1111 | $yy$ |
| | 0001 | $xx$ |
| | 1111 | $yy$ |
| | 0101 | $yx$ |
| | 1111 | $yy$ |
| | 0011 | $xy$ |
| | 1111 | $yy$ |
| | 0101 | $yx$ |
| | 1111 | $yy$ |

| $n=8$ | ps | gr |
|---|---|---|
| | 11111111 | $yyy$ |
| | 01010101 | $yyx$ |
| | 11111111 | $yyy$ |
| | 00110011 | $yxy$ |
| | 11111111 | $yyy$ |
| | 01010101 | $yyx$ |
| | 11111111 | $yyy$ |
| | 00010001 | $yxx$ |
| | 11111111 | $yyy$ |
| | 01010101 | $yyx$ |
| | 11111111 | $yyy$ |
| | 00110011 | $yxy$ |
| | 11111111 | $yyy$ |
| | 01010101 | $yyx$ |
| | 11111111 | $yyy$ |
| | 00001111 | $xyy$ |
| | ↓ | ↓ |
| | etc. | |

**Figure 0.**

In the column marked "ps" we have tabulated the ps-values as bit vectors of length $n$. In the column marked "gr" we have tabulated the corresponding "generators", which are more compact characterizations of ps-values. Starting with a bit vector of length 1, consisting of a single 1, we can construct any ps-value by successively subjecting our bit vector to operation $x$ or to operation $y$:

$x$ : double its length by prefixing zeros

$y$ : double its length by catenating a copy.

The generator records in order from right to left the sequence of operations that construct the ps-value. (Hence 01010101 has generator $yyx$, while 00001111 has generator $xyy$). Note that each $y$ in the generator doubles the number of 1s in the bit vector.

For the sake of completeness we give a recursive definition, which is completely analogous to that of ps.

For $n = 1$, gr.1 = "the empty string"; for $n'$, with $1 \leq q * N + r < N^2 \ \wedge \ 0 \leq r < N$

$$\begin{aligned} \mathrm{gr}'.(q * N + r) \ = \ & \textbf{if} \ \ r = 0 \rightarrow x\cdot \mathrm{gr}.q \\ & [] \ \ r \neq 0 \rightarrow y\cdot \mathrm{gr}.r \\ & \textbf{fi} \end{aligned}$$

where $x\cdot$ and $y\cdot$ denote prefixing by the indicated singleton.

The advantage of generators is that they are nonredundant: not every bit vector is a ps-value, but every $x/y$-sequence is a generator. Generators, more precisely, the function gr, will be explored for its own sake in the next section.

* * *

The tabulated generators admit an alternative interpretation, viz. as identifiers of the bits whose inversions generate what is known as the "Gray Code", e.g. for $n = 4$:

| | $xx$ | $xy$ | $yx$ | $yy$ |
|---|---|---|---|---|
| | 0 | 0 | 0 | 0 |
| invert bit $yy$: | 0 | 0 | 0 | 1 |
| invert bit $yx$: | 0 | 0 | 1 | 1 |
| invert bit $yy$: | 0 | 0 | 1 | 0 |
| invert bit $xy$: | 0 | 1 | 1 | 0 |
| invert bit $yy$: | 0 | 1 | 1 | 1 |
| invert bit $yx$: | 0 | 1 | 0 | 1 |
| invert bit $yy$: | 0 | 1 | 0 | 0 |
| invert bit $xx$: | 1 | 1 | 0 | 0 |
| | ↓ | | ↓ | |

etc.

**Figure 1.**

That the Gray-Code table of width $n$ and height $N$, as generated above, contains all $N$ values, can be demonstrated by showing that any two entries in the table differ.

To this end we define for generators of equal length a total order $<$ by the alphabetic order, i.e. with $P$ and $Q$ generators of equal length

(4a) $x \cdot P < y \cdot Q$

(4b) $x \cdot P < x \cdot Q \equiv P < Q$

(4c) $y \cdot P < y \cdot Q \equiv P < Q$

and now we can formulate

**Lemma 0.** In the sequence gr.$i$, $h \leq i < j$ with $1 \leq h < j < N$, the alphabetic minimum of the values in the sequence occurs exactly once.

**Proof sketch.** In view of the recursive definition of gr, the proof is by mathematical induction on the length of the generator, with a case analysis in the induction step.

Either some value in the sequence starts with an $x$, or all values in the sequence start with $y$.

In the first case, (4a) tells us that we can confine our attention to the values of the form $x \cdot$ gr.$q$; the $q$-values being consecutive, (4b) thus reduces the demonstrandum for gr$'$ to the demonstrandum for gr.

In the second case we have a sequence of values of the form $y \cdot$ gr.$r$ with consecutive $r$-values; this time, (4c) performs the reduction. (End of Proof sketch.)

Lemma 0 implies the absence of duplicates in the table we generated for the Gray Code: in the sequence of transitions that transforms the one entry into the other, the unique minimum identifies a position in which one inversion takes place, and in which therefore the entries differ.

* * *

After the above interlude about the Gray Code, we return to our original problem. We have shown earlier that all the $s$-values generated by (3) are distinct, whatever rotations for the ps-values are chosen, but now the obvious challenge is to prove this as well in a way analogous to our demonstration of the absence of duplicates in the Gray Code.

To this end, we associate with each generator its "patterns" and its "partition"; we discuss the patterns first. The patterns are the corresponding ps-value in its various rotations. For instance, generator $yxy$ yields ps-value

| | |
|---|---|
| 00110011 | (0,1,4,5) |

and by rotation the further patterns

| | |
|---|---|
| 01100110 | (1,2,5,6) |
| 11001100 | (2,3,6,7) |
| 10011001 | (3,4,7,0) . |

Next to each pattern we have recorded —numbering them in the patterns from right to left —the bit positions of its 1s. They are of interest because, modulo $n$, their differences are unaffected by rotation. Note that the positions of the 1s in the ps-value are obtained by substituting in the generator 0 for $x$, and for each $y$ a 0 or a 1, and reading the result as a binary number. For instance, in the above example, generator $yxy$ yields 1s in positions $000(=0)$, $001(=1)$, $100(=4)$ and $101(=5)$.

The partition associated with a generator partitions the bit positions and has as many cells as an associated pattern has 1s. Each cell contains the values whose binary representations coincide in the $y$-positions of the generator. For instance, generator $yxy$ yields the partition

| | | | | |
|---|---|---|---|---|
| $[0,2]$ | $[1,3]$ | $[4,6]$ | $[5,7]$ | or |
| 0?0 | 0?1 | 1?0 | 1?1 | |

**Lemma 1.** An arbitrary pattern associated with a generator has exactly one 1 in each cell of the partition associated with that generator.

**Proof.** Since the number of 1s in the pattern equals the number of cells in the partition, it suffices to show that no cell contains two 1s. We show this by demonstrating that no distance between two 1s in a pattern equals the distance between two positions in a cell of the partition. We use two little theorems about binary arithmetic of natural numbers:

(i) the least-significant 1 in the difference of two distinct numbers occurs in a position where one of the numbers has a 1, the other a 0;

(ii) two numbers whose least-significant 1s occur in different positions, differ.

The distance between two 1s in a pattern equals modulo $n$ (which is a "higher" power of 2) the distance between two 1s in the ps-value (e.g. in our last example $7-0=(8+4)-5$). The ps-value has its 1s in positions whose binary numbers have their 1s in the $y$-positions of the generator.

From little theorem (i) we conclude that the least-significant 1 in the binary representation of the distance between two 1s in a pattern occurs in a $y$-position of the generator.

By construction, a cell contains values that in binary only differ in $x$-positions of the generator, and thus we conclude from little theorem (i) that the least-significant 1 in the binary representation of the distance between two positions in a cell occurs in an $x$-position of the generator.

Thanks to little theorem (ii), the last two conclusions imply that no distance between two 1s in a pattern equals the distance between two positions in a cell of the partition. (End of Proof.)

**Lemma 2.** Let generator $G0$ be alphabetically less than generator $G1$; then each pattern associated with $G1$ has an even number of 1s in each cell of the partition associated with $G0$.

**Proof.** This proof is a little bit more complicated because the notion of alphabetic order has to be taken into account. The proof is by mathematical induction over the length of the generators. In base and step, $P$ and $Q$ are generators of the same length $b$ and $n = 2^b$.

*Base.* The base case —see (4a)— are two generators whose left-most characters differ, i.e. $G0 = x{\cdot}\,P$ and $G1 = y{\cdot}\,Q$.

By construction, the 1s in a pattern associated with $y{\cdot}\,Q$ can be grouped in pairs of 1s that are $n$ apart. Also by construction, positions that are $n$ apart occur in the same cell of the partition associated with $x{\cdot}\,P$. From these two observations, the theorem follows for the base case.

*Step.* Here —see (4b) and (4c)— we consider two generators whose left-most characters are equal. With $P < Q$ we distinguish two cases.

(i) $\quad G0 = x{\cdot}\,P \;\wedge\; G1 = x{\cdot}\,Q\;.$

Patterns associated with $x{\cdot}\,Q$ are twice as long and have the same number of 1s as those associated with $Q$ and are obtained from the latter by moving a number of its 1s at the right-hand side over $n$ places to the left. In the transition from $P$ to $x{\cdot}\,P$, each cell is doubled in size by taking its union with the image obtained by moving it over $n$ places to the left. Consequently, a new pattern has as many 1s in a new cell as some original pattern had in some original cell.

(ii) $\quad G0 = y{\cdot}\,P \;\wedge\; G1 = y{\cdot}\,Q\;.$

Patterns associated with $y{\cdot}\,Q$ are twice as long and have twice as many 1s as those associated with $Q$ and are obtained from the latter by prefixing each pattern with a copy of itself. In the transition from $P$ to $y{\cdot}\,P$, cell size remains unchanged but their number is doubled: each original cell spawns the image obtained by moving it over $n$ places to the left. Again a new pattern has as many 1s in a new cell as some original pattern had in some original cell.

And thus Lemma 2 has been proved for generators of arbitrary length. (End of Proof.)

Consider now a sequence of moves in (3). According to Lemma 0, there is in that sequence a unique move with the alphabetically smallest generator. Now focus on the parities of the cells of the partition of that generator. According to Lemma 1, that unique move changes these parities, while according to Lemma 2, all other moves leave them unchanged.

**Acknowledgements.** I thank Rutger M. Dijkstra for drawing my attention to the problem and for generalizing four to any power of two. In matters of presentation I am endebted to the Tuesday Afternoon Clubs of Eindhoven and Austin.

# A ring of subtracters

Joop van den Eijnde

## 1 The Story

Kardemomme is a most peculiar little town. It may be situated somewhere up north, perhaps in Scandinavia. Nobody seems to know exactly where. Except Frans Kruseman Aretz... It is told that this wise man, a professor, must have spent a great deal of time with the strange, but very friendly, people of Kardemomme. Each time, returning from holiday, he fascinates his colleagues with stories of the Kardemomme society. Not to mention the professor's students, some of which are suspected of deliberately flunking their exams, eager to read more about Kardemomme in the next session.

Thus we heard about the famous harbour of Kardemomme, its extraordinary new communication network, its distinguished tower hotel, and the unusual elevator in the social security building [1]. What most of you probably don't know, is that the town features an elite club, with many enthusiastic members. The club, started long ago with a single member, is called the FKA. What it means is a well-kept secret, but an educated guess would be "Free Kardemomme Association"; nothing special, really.

Every night at ten sharp all members gather in the community building, and discuss the wonders of mathematics. But before the meeting starts, a stunning opening ritual is performed. Every club member, starting with the youngest, looks up his immediate successor in age and compares the amount of club coins they both own. The younger one then pockets the difference in both amounts, settling the excess or shortage of what he owned before with the central club fund, which is almost inexhaustible. Of course the eldest member turns to the youngest again, the latter recalling the amount he owned when he came in that night.

Naturally there is an endless queue of town inhabitants anxious to join the club. However, the club regulations are very strict. New members are allowed only when the opening ritual has lost its attraction, that is when no member has any coins left. If this happens, a number of newcomers exactly equal to the original membership will be admitted. Then all members, new as well as old, get an arbitrary amount of coins from the club fund, and the club is in business again.

Everybody used to be quite satisfied with these arrangements, but only recently several would-be members have become worried. The FKA is a big club now, and it has been ages since the last group of new members was admitted. In desperation they wonder:

will the opening ritual ever stop, and if so, when can we expect to join up? As they are not initiated in the mysterious ways of mathematics yet, they turn to their last resort. Had not Frans Kruseman told tales of the famous **Eindhoven School of Computing Science**, a group of scholars, fond of puzzles, and capable of solving any mathematical problem just by deriving the solution from the specification?

## 2 The problem

More then five years ago the *Ring of Subtracters* problem was conveyed to me by Wim Feijen. Its origin remains unclear, but for all I know it could be **Kardemomme**. In [2] a special case of it was presented as "Professor Ducci's Observation", after an Italian professor in the 1930's. In this case, one starts with a cyclic arrangement of four natural numbers. From this ring of numbers a new one is constructed by replacing each number with the absolute difference of this number and its clockwise successor. Regardless of the original numbers, repeating this process always ends with four zeros. The proof in [2] that this must be so is long, using a style we, as members of the **Eindhoven School**, have been taught to abhor, involving a great deal of case analysis.

Turning to the general problem, we note that the ring may consist of $2^n$ arbitrary natural numbers, for any natural $n$ (natural numbers include zero, naturally). This corresponds to the FKA starting out with one member only, and doubling its membership each time the opening ritual would be vacuous. The fact that every night each member compares his amount of coins with the same other member, his direct successor in age, is of course crucial in establishing a fixed cyclic arrangement.

Clearly, the maximum of all numbers in the ring is a descending function of time. This limits the number of possible states of the ring to $(M+1)^{2^n}$, $M$ being the original maximum, so the process eventually becomes cyclic. Once some state turns up for the second time, the people in the queue know where they stand. This is no consolation to them, however, because, as they see it, they may be dead and buried before that happens. And if the cycle contains no all-zeros state, they will never be members...

## 3 The solution

The people of **Kardemomme** can rest assured: the opening ritual is bound to stop, starting from any initial state. Even better, this will happen long before $(M+1)^{2^n}$ days have passed. If you like to verify this for yourself, this is the place to stop reading.

The solution I found is based on the observation that the process of subtracting successors is asymmetric with respect to odd and even numbers. Two consecutive odd numbers produce an even number, whereas two even number also produce an even number. Hence, states with odd numbers are unstable, they keep changing. On the other hand, states

with only even numbers remain even forever. Since the desirable all-zeros state is even, we may wonder if any odd state is bound to turn even. Our solution, then, consists of three steps, formulated as lemmas, all three with a very straightforward proof. They are the following.

**Lemma 1:** After $2^n$ steps all numbers in the ring are even.

**Lemma 2:** After $m * 2^n$ steps all numbers in the ring are divisible by $2^m$.

**Lemma 3:** Starting from a state with maximum number $M$, after at most $\lceil \log(M+1) \rceil * 2^n$ steps all numbers are zero.

**Proof of Lemma 1:**

Let us label the numbers of the ring clockwise with consecutive natural numbers, from 0 for an arbitrary number (say, corresponding to the youngest member) up to $2^n - 1$. Now define a boolean function $E$ by

$E \cdot i \cdot j \equiv$ "after $j$ steps the i-th number is even",

for all natural $i$ and $j$. The first argument can be taken modulo $2^n$. The state in the next step then satisfies

$$E \cdot i \cdot (j+1) \equiv E \cdot i \cdot j \equiv E \cdot (i+1) \cdot j.$$

Noting that $1 = 2^0$, the latter equation is generalized by replacing 0 with an arbitrary natural number $k$, yielding the *induction hypothesis*, or IH for short:

$$E \cdot i \cdot (j + 2^k) \equiv E \cdot i \cdot j \equiv E \cdot (i + 2^k) \cdot j,$$

for all $i$ and $j$.

The construction of the induction hypothesis takes care of the base case. For the induction step we derive

$E \cdot i \cdot (j + 2^{k+1})$

$\equiv$ {IH, substituting $j + 2^k$ for $j$}

$E \cdot i \cdot (j + 2^k) \equiv E \cdot (i + 2^k) \cdot (j + 2^k)$

$\equiv$ {2× IH, in the 2nd instance substituting $i + 2^k$ for $i$}

$$E \cdot i \cdot j \equiv E \cdot (i + 2^k) \cdot j \equiv E \cdot (i + 2^k) \cdot j \equiv E \cdot (i + 2^{k+1}) \cdot j$$

$\equiv$ ⟨calculus⟩

$$E \cdot i \cdot j \equiv E \cdot (i + 2^{k+1}) \cdot j,$$

proving that IH is indeed valid for all $k$. Note that in the derivation we made good use of the associativity of the equivalence. Taking $j = 0$ and $k = n$ IH implies

$$E \cdot i \cdot 2^n \equiv E \cdot i \cdot 0 \equiv E \cdot (i + 2^n) \cdot 0,$$

and, since number $i + 2^n$ is identical to i, the second equivalence yields true, and we are done.

□

**Proof of Lemma 2:**

Clearly, divisibility of all numbers by a common factor, say $2^m$, is invariant under a subtraction step. In fact, this common divisor can be factored out. After another $2^n$ steps, the quotient numbers are divisible by 2, and hence the numbers themselves are divisible by $2^{m+1}$.

□

**Proof of Lemma 3:**

Since no number in the ring can ever exceed the original maximum $M$, all numbers divisible by $2^m$ must be 0 as soon as $2^m \geq M + 1$. This is bound to happen after at most $S$ steps, where

$$S = (\downarrow m : 2^m \geq M+1 : m) * 2^n = (\downarrow m : m \geq \log(M+1) : m) * 2^n = \lceil \log(M+1) \rceil * 2^n$$

□

## 4 The discussion

It appears that the inhabitants of **Kardemomme** can be satisfied with the upper bound derived: it is only a modest multiple of the number of members, because the logarithm increases very slowly with increasing argument. Of course, we are not satisfied yet. The

question is: is the upper bound sharp? In other words, do there exist instances requiring that many steps?

It turns out that the upper bound for the number of steps needed to factor out a 2, in Lemma 1, is indeed sharp. A simple instance requiring $2^n$ steps is a ring with all 1's, except for a single 0. After one step we have all 0's, except for two consecutive 1's. Next the first 1 has to propagate backwards all the way along the ring before it can cancel out the second, which stays put until that happens.

The logarithm factor may not be sharp. For one thing, it depends on the maximum over the ring remaining unchanged until everything cancels out. This seems unlikely, but for now it remains an open question. However, I do know there is no upper bound independent of $M$, for $n$ fixed. This follows from a construction by Hans Zantema [3], which is too nice not to be recorded here.

**Proof of the absence of an absolute upper bound:**

Consider the case $n = 2$. If for a state $(0, a_1, a_2, a_3)$, with $0 < a_1 < a_2 < a_3$ and $c = a_3 - a_1 - a_2 > 0$, the process ends in $S$ steps, then for state $(0, b_1, b_2, b_3)$ the process will end in $S + 1$ steps, if we choose

$b_1 = c$, $b_2 = 2 * (a_1 + c)$, and $b_3 = 2 * a_3$+c.

This is due to the fact that two steps from $(0, b_1, b_2, b_3)$ take us to the same state as one step from $(0, a_1, a_2, a_3)$, apart from a factor of 2. Since we have $0 < b_1 < b_2 < b_3$ and $b_3 - b_1 - b_2 > 0$, this construction can be repeated indefinitely. For $n > 2$ the ring can be filled up with as many copies of $(0, a_1, a_2, a_3)$ as needed.

□

Note that the cases $n = 0$ and $n = 1$ are dismissed as trivially ending in 1 and 2 steps respectively.

Another remark is that if we restrict all numbers in the ring to 0's and 1's the case $n = 4$ reduces to the "Problem of the 16 lights", which is solved in [4], in a way similar to the proof of Lemma 1. I prefer the present proof because it is slightly more calculational. Maybe the presentation using lights appeals to Frans Kruseman Aretz, because of his former position at a well-known bulb factory in the Netherlands...

## 5 The epilogue

The problem of the ring of subtracters was solved by me in May 1990, but so far I never got round to writing it down. The departure of Frans Kruseman Aretz from the department of Computing Science at Eindhoven University of Technology gives me the opportunity

to finish it, and dedicate the resulting story to Frans, a colleague whom I have always liked and respected very much. I know he is fond of mathematical puzzles, as I am. Unfortunately, I never got the chance to cooperate with him on a scientific project. One of the reasons for this was that we both worked part-time for the university, Frans on Tuesdays only, and I every day but Tuesday... Sometimes I envied the members of his little group, working together in close harmony. I sure hope we all can enjoy his presence, albeit occasional, at the university for a long time to come.

## 6 The references

[1] F.E.J. Kruseman Aretz, Exams *Operating Systems in Concurrent Pascal (2K670)*, Eindhoven University of Technology, 29-11-88, 15-8-89, 27-11-90, and 13-8-91.

[2] Ross Honsberger, *Ingenuity in Mathematics* (Random House/Singer 1970), Essay Ten, 73-83.

[3] Hans Zantema, Private communication (by electronic mail), 17-7-1990.

[4] Edsger W. Dijkstra, *A problem solved by my nephew Sybrand L. Dijkstra*, personal note EWD904 (Department of Computer Sciences, University of Texas at Austin, 14-12-84).

# Short or beautiful?

Huub ten Eikelder
dedicated to prof. dr. F.E.J. Kruseman Aretz

Suppose you are walking. Then it may happen that at a point there are two routes to your destination: a long but beautiful route or a short and not so beautiful one. Sometimes the choice will be simple, sometimes it can be a small dilemma. In Computing Science the choice between short and beautiful is seldom a problem. Very often these properties go hand in hand. In this note we present an example where this property, at least in our opinion, does not hold. We shall give two proofs of a theorem. One is the proof found in the literature. This proof contains a very beautiful part. The other proof lacks such a beautiful element, but it is much shorter. Every reader of this note may have its own preference.
In Section 1 we present the theorem. The beautiful and the short proof are given in Sections 2 and 3 respectively. Finally, some reflections are given in Section 4. That Section should not be read too seriously.

## 1 The theorem

We will discuss a theorem in the field of approximation methods for combinatorial optimization problems. First we introduce what is probably the most well-known combinatorial optimization problem, the Travelling Salesman Problem. Then we explain an approximation method for solving combinatorial optimization problems. After that we state the theorem to be discussed.

**Problem 1 (Travelling Salesman Problem (TSP))** *Given is a number $n \in \mathbb{N}$, a symmetric $n \times n$ matrix $(d_{ij})$ with elements in $\mathbb{N}$ and a number $B \in \mathbb{N}$. The elements of $d$ are interpreted as distances between $n$ cities. Does there exist a tour, i.e. a closed route visiting all cities once, with length at most $B$?*
□

The set of all tours will be called $S$. Note that $S$ is not dependent on the distance matrix $d$. Only the *length* $f(t)$ of a tour $t \in S$ depends on $d$. In fact this is the *decision* version of the TSP problem. In the more frequently encountered *optimization* version the number $B$ is not given and the question is to compute a shortest tour. Clearly a solution of the

optimization version of the TSP problem yields immediately a solution of the decision version.

It is well known that the TSP is a $\mathcal{NP}$-complete problem. Hence in most cases finding a solution to a TSP problem is practically impossible. For the optimization version of the TSP various techniques have been developed to find approximate solutions, i.e. tours with a length that exceeds the length of an optimal tour only a little. One of these techniques is the use of *local search methods*. In many practical situations it provides good quality solutions, using a computing time that is bounded by a (low order) polynomial in the size ($n$) of the problem. An essential ingredient in local search is the so-called *neighbourhoud structure*, i.e. a mapping

$$\mathcal{N} : S \rightarrow \mathcal{P}(S) \ .$$

The elements of $\mathcal{N}(t)$ are called the neighbours of tour $t$. Note that the notion of neighbour is independent of the distance matrix $d$. The local search algorithm is given by

$$
\begin{array}{l}
[[\textbf{var } t, t' : S; W : \mathcal{P}(S) \\
\mid t := \textit{some initial tour}\,; W := \emptyset \\
\ \textbf{do } \mathcal{N}(t) \backslash W \neq \emptyset \rightarrow \\
\qquad\qquad t' :\in \mathcal{N}(t) \setminus W \\
\qquad\qquad ;\textbf{if } f(t') < f(t) \rightarrow t, W := t', \emptyset \\
\qquad\qquad \ [] \ f(t') \geq f(t) \rightarrow W := W \cup \{t'\} \\
\qquad\qquad \textbf{fi} \\
\ \textbf{od} \\
\ (* \ t \textit{ is a local minimum} \ *) \\
]] \ .
\end{array}
$$

Informally the algorithm describes a walk through the solution space $S$ until a tour is found that has no neighbour with smaller length. Such a tour is called a *local minimum* with respect to $\mathcal{N}$. Note that the fact that a tour is a local minimum depends on the distance matrix $d$. If all elements of $d$ are equal, all tours have the same length and hence all tours are local minima. The major drawback of local search is of course that it terminates with a local minimum and not necessarily with a *global minimum*, i.e. a shortest tour in $S$. The quality of the found local minimum depends of course on the used neighbourhoodstructure. A neighbourhoodstructure is called *exact* if, for all distance matrices, a local minimum is also a global minimum. For instance the neighbourhoodstructure $\mathcal{N}$ with $\mathcal{N}(t) = S$ for all tours $t$, is trivially exact. Using local search with this neighbourhood is nothing but searching through the whole solution space $S$.
Of course smaller exact neighbourhoodstructures would be very interesting. The following theorem shows that we cannot hope for finding small exact neighbourhoods.

**Theorem 2** *If $\mathcal{P} \neq \mathcal{NP}$, there does not exist an exact neighbourhoodstructure for the TSP with a size bounded by a polynomial in $n$.*
□

This result has first been given by Papadimitriou and Steiglitz [PS76]. A somewhat more readable version can be found in [PS82]. In this note we will give two proofs of this theorem. In Section 2 we present the proof given in [PS82]. This proof contains a beautiful part, viz. the reduction of the Hamilton cycle problem to the Restricted Hamilton cycle problem. Everybody who has some feeling for beauty, will appreciate the construction using the diamond subgraphs in this proof. After that, we shall present in Section 3 a short, straightforward proof. This alternative proof does not have the beauty of the proof mentioned above, but it is much shorter. The choice between the two proofs is left to the reader.

In the remaining part of this Section, some notions used in both proofs are presented. First we define the following decision problem.

**Problem 3 (Hamilton cycle (HC))** *Given is an undirected graph $G$. Does there exist a Hamilton cycle in $G$, i.e. a closed walk through $G$ which visits every node exactly once?*
□

It is well known that HC is a $\mathcal{NP}$-complete problem. The $\mathcal{NP}$-completeness of HC will be the starting point for both proofs. To prove that a problem $\Pi$ is $\mathcal{NP}$-complete, we have to show that i) $\Pi \in \mathcal{NP}$ and ii) some known $\mathcal{NP}$-complete problem $\Pi'$ reduces to $\Pi$. The first part of these proofs is usually elementary and will be skipped in this note. In ii) a mapping from $\Pi'$ instances to equivalent $\Pi$ instances has to be given and it must be verified that this mapping can be computed in a time bounded by a polynomial in the size of the $\Pi'$ instance. This verification will also be skipped here.

## 2 The beautiful proof

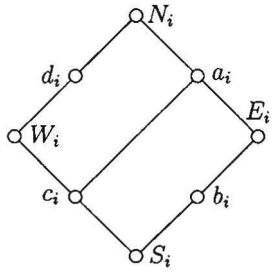
The proof given by Papadimitriou and Steiglitz [PS82] introduces two more decision problems and shows that they are also $\mathcal{NP}$-complete.

**Problem 4 (Restricted Hamilton cycle (RHC))** *Given is an undirected graph $G$ and a Hamilton path in $G$, i.e. a walk between two nodes in $G$ that visits every node exactly once. Does there exist a Hamilton cycle in $G$?*
□

**Theorem 5** *RHC is $\mathcal{NP}$-complete.*

**Proof**: The $\mathcal{NP}$-completeness of RHC is shown by reducing HC to RHC as follows. Given is an instance of HC, i.e. a graph $G = (V, E)$ with say, $V = \{1, \ldots, n\}$. The corresponding instance of RHC consistys of a graph $G'$ and a Hamilton path $p$ in $G'$. $G'$ is constucted as follows.

– For every node $i \in V$, $G'$ contains a diamond subgraph




– Besides the edges introduced above, $G'$ contains the edges

$$\{\{E_i, W_j\} \mid \{i,j\} \in E\} \cup \{\{S_i, N_{i+1}\} \mid 1 \leq i < n\} \ .$$

Then trivially $G'$ contains a Hamilton path $p$ from $N_1$ to $S_n$. In fact the Hamilton path $p$ is of no help when constructing a Hamilton cycle in $G'$. In a walk that must pass all nodes once, every diamond can only be visited once and must be traversed either in the East-West or in North-South direction. Due to the form of the diamonds and the additional edges mentioned above, this direction has to be the same for all diamonds. In the North-South direction, a Hamilton cycle is impossible, since the edge between $S_n$ and $N_1$ is missing. Moreover, a Hamilton cycle that traverses all diamonds in the East-West direction exists only if a corresponding Hamilton cycle in $G$ exists. Hence we have mapped every instance of HC to an equivalent instance of RHC. This yields the $\mathcal{NP}$-completeness of RHC.
□

**Problem 6 (Travelling Salesman Suboptimality (TSPS).)** *Given is a number $n \in \mathbb{N}$, a symmetric $n \times n$ distance matrix $(d_{ij})$ with elements in $\mathbb{N}$ and a tour $t$ of the corresponding TSP problem. Is tour $t$ suboptimal, i.e. does there exist a tour $t'$ with $f(t') < f(t)$?*
□

**Theorem 7** *TSPS is $\mathcal{NP}$-complete.*

**Proof**: We reduce RHC to TSPS. Given is an instance of RHC, i.e. a graph $G = (V, E)$ with $V = \{1, \ldots, n\}$ and a Hamilton path $p$ in $G$ between say, nodes $i$ and $j$. If $\{i,j\} \in E$, then the RHC instance is a "yes-instance" and it is mapped to an arbitrary yes-instance of TSPS. Next assume $\{i,j\} \notin E$. Then the RHC instance is mapped to a TSPS instance

with $n$ cities, distance matrix $d$ given by $d_{ij} = 1$ if $\{i,j\} \in E$ and $d_{ij} = 2$ if $\{i,j\} \notin E$, and a tour $t$ obtained as follows. Adding the edge $\{i,j\}$ to the Hamilton path $p$ results in a Hamilton cycle in $G$, which defines a unique tour $t$ in the TSPS instance.
Note that tour $t$ has length $n-1+2 = n+1$, and that all tours must have a length at least $n$. Hence trivially $t$ is suboptimal if and only if there exists a tour with length $n$, which is equivalent to the existence of a Hamilton cycle in the HC instance. Thus TSPS is $\mathcal{NP}$-complete.
□

Now we can ultimately proof theorem 2.

**Proof** (of theorem 2): Suppose there exists an exact neighbourhood $\mathcal{N}$ for the TSP problem with $n$ cities, with a size bounded by a polynomial in $n$. Then for an arbitrary tour $t$ we can, by searching through $\mathcal{N}(t)$, decide whether $t$ is a local minimum in a time bounded by a polynomial in $n$. However, since the neighbourhood $\mathcal{N}$ was assumed to be exact, this means that we can decide in polynomial time whether $t$ is a global minimum. Otherwise stated, we can decide suboptimality of $t$ in polynomial time. Assuming that $\mathcal{P} \neq \mathcal{NP}$ this contradicts the $\mathcal{NP}$-completeness of TSPS.
□

## 3 The short proof

Here we give an alternative proof of theorem 2. First we define another decision problem and prove its $\mathcal{NP}$-completeness.

**Problem 8 (Travelling Salesman Problem with two distances(TSP2))** *Given is a number $n \in \mathbb{N}$, a symmetric $n \times n$ matrix $(d_{ij})$ with elements in $\{1,2\}$ and a number $B \in \mathbb{N}$. Does there exist a tour with length at most $B$?*
□

**Theorem 9** *TSP2 is $\mathcal{NP}$-complete.*

**Proof**: We prove that TSP2 is $\mathcal{NP}$-complete by reducing HC to it. Given is an instance of HC, i.e. a graph $G = (V,E)$ with $V = \{1,\ldots,n\}$. This HC instance is mapped to a TSP2 instance with $n$ cities, distance matrix $d$ given by $d_{ij} = 1$ if $\{i,j\} \in E$ and $d_{ij} = 2$ if $\{i,j\} \notin E$, and $B = n$.
Then trivially a Hamilton cycle in the HC instance corresponds to a tour with length $B$ in the TSP2 instance. Thus TSP2 is $\mathcal{NP}$-complete.
□

Now the short proof of theorem 2.

**Proof** (of theorem 2): Suppose there exists an exact neighbourhoodstructure $\mathcal{N}$ for the TSP problem with $n$ cities, with a size bounded by some polynomial $p(n)$ in $n$. Recall that this means that, for every distance matrix $d$, a local minimum is a global minimum. In particular this holds for all distance matrices with elements 1 or 2, i.e. for TSP2 instances. Now perform local search for such an instance using neighbourhood $\mathcal{N}$. As the tourlength is contained in $\{l \in \mathbb{N} \mid n \leq l \leq 2n\}$ and decreases in every step of local search, at most $n+1$ neighbourhoods are searched through. Hence a local mimimum is obtained in a time at most $(n+1)p(n)$, which is again a polynomial in $n$. However, since $\mathcal{N}$ was exact, we found the minimum tourlength in polynomial time. Comparing this minimum tourlength with $B$ solves the TSP2 instance in polynomial time. Again assuming that $\mathcal{P} \neq \mathcal{NP}$, this contradicts the $\mathcal{NP}$-completeness of TSP2.
□

## 4 Finally

An attentive reader will have remarked that we used some small tricks in our presentation of the two proofs. In Section 1 we discussed the Hamilton cycle problem and recapitulated how a proof of $\mathcal{NP}$-completeness can be given. This material was used in both proofs. Adding this material to the Sections 2 and 3 would make both proofs longer, such that the relative difference between the length of the two proofs becomes smaller. Also adding the picture in the beautiful proof was of much help. Note that it takes about the same space as half of the short proof.

Of course one can also look at side effects of the proofs. Here the beautiful proof clearly wins. After that proof we also know that RHC and TSPS are $\mathcal{NP}$-complete, while the short proof yields only the $\mathcal{NP}$-completeness of TSP2. However, this last result is somewhat more surprising, as the author found out that it is relatively easy to convince people that TSP2 instances are in $\mathcal{P}$. You start with taking distances 0 and 1 instead of 1 and 2.

## References

[PS76] Padimitriou, C.H. and H. Steiglitz. "Some Complexity Results for the Travelling Salesman Problem", *Proc. 8th Ann. ACM Symp. on Theory of Computing*, ACM, New York, 1-9, 1976.

[PS82] Padimitriou, C.H. and H. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, New Jersey, 1982.

# Phase synchronization for a string of machines

The Eindhoven Tuesday Afternoon Club[1]

dedicated to Prof.Dr. F.E.J. Kruseman Aretz

In this note we present yet another example of how traditionally complicated artefacts like parallel programs can nowadays be developed in a highly systematic manner by means of simple formalisms such as the predicate calculus and the theory of Owicki and Gries. Operational thought is largely – if not completely – abandoned, even in arguing about progress properties.

Moreover, we refute the often heard conjecture that the theory of Owicki and Gries would – at best – be suitable for the a posteriori verification of shared-memory parallel programs, because at this occasion we will use it for the *construction* of a *distributed* algorithm.

* * *

We consider $N+1$ machines, numbered 0 through $N$. They are arranged in a linear network such that machine $i$, $1 \leq i \leq N-1$, can communicate with machines $i-1$ and $i+1$ only, machine 0 with just machine 1, and machine $N$ with just machine $N-1$.

Each machine is engaged in a computation proper given by

$$\text{Mach.}i : \quad *[\, S_i \,] ,$$

where $S_i$ is a terminating computation.

In order to express our synchronization requirement we introduce fresh variables $x_i$ and adjust the program texts as follows:

$$\begin{array}{ll} \text{Pre:} & \langle \forall j :: x_j = 0 \rangle \\ \text{Mach.}i : & *[\ \{\ \langle \forall j :: x_i \leq x_j \rangle\ \}\ S_i \\ & \quad ;\ x_i := 1 + x_i \\ & \quad ]\ . \end{array}$$

[1] A. Bijlsma, R.W. Bulterman, W.H.J. Feijen, A.J.M. van Gasteren, R.R. Hoogerwoord, C.S. Scholten, F.W. van der Sommen

(It goes without saying that no further changes of $x_i$ are allowed in what follows.)

The requirement now is to superimpose, on the computation proper, an additional algorithm so that the plugged-in assertion

$$Z_i : \quad \langle \forall j :: x_i \leq x_j \rangle$$

be a correct precondition to $S_i$, for all $i$.

* * *

Before we embark on a solution, we wish to point out how we will deal with individual progess in examples like the above.

*Irrespective* of what our ultimate solution will be, relation

$$\text{MB}: \quad \langle \forall i, j :: x_i \leq 1 + x_j \rangle$$

is a system invariant: just apply the axiom of assignment for $x_i := 1 + x_i$ and use precondition $Z_i$. Relation MB is a so-called multibound [WF195], and in combination with the structure of our machines, its impact is that only two things can happen as far as "liveness" is concerned, viz. in whatever solution we may come up with

| | |
|---|---|
| either | all machines are guaranteed to make individual progress |
| or | all machines get stuck (i.e. total deadlock). |

So, in what follows we can show individual progress by just showing the absence of total deadlock.

* * *

Now, if there were no restrictions on the communication facilities, a first and simple solution would be

$$\begin{array}{ll} \text{Pre}: & \langle \forall j :: x_j = 0 \rangle \\ \text{Mach.}i: & *[\ \underline{\text{if}}\ \langle \forall j :: x_i \leq x_j \rangle \rightarrow \text{skip}\ \underline{\text{fi}} \\ & \ ;\ \{\ Z_i : \langle \forall j :: x_i \leq x_j \rangle\ \}\ S_i \\ & \ ;\ x_i := 1 + x_i \\ & ]\ . \end{array}$$

The local correctness of $Z_i$ follows from the guard, and its global correctness by the Rule of Widening – other machines only increase $x_j$, thus making $Z_i$ more true than before –.

* * *

The problem with the above solution, however, is the guard which, due to the limited communication facilities, just cannot be evaluated by Mach.$i$. That guard is equivalent to

(0) $\quad x_i \leq \langle \downarrow j :: x_j \rangle$ ,

and inspired by the topology of the network, we rewrite it into

$$x_i \leq \langle \downarrow j : j \leq i : x_j \rangle \downarrow \langle \downarrow j : j \geq i : x_j \rangle \ ,$$

and then – introducing variables $l_i$ and $r_i$ – into

(1) $\quad x_i \leq l_i \downarrow r_i$ ,

hoping that we can maintain

(2a) $\quad l_i = \langle \downarrow j : j \leq i : x_j \rangle \quad$ and

(2b) $\quad r_i = \langle \downarrow j : j \geq i : x_j \rangle$ .

Maintaining relations (2) is, however, far too demanding because a simple increase of just one $x_j$ might necessitate an adjustment of many $l$'s and many $r$'s.

Here is the place to remember a very simple, but useful, theorem from the art of multiprogramming, viz. the theorem dubbed Strengthening the Guard. It states that strengthening a guard of an if-statement is harmless to the partial correctness, i.e. the correctness of the annotation. Hence, by this theorem, it suffices that our original guard (0) be *implied* by our proposed guard (1), and this is the case if we can maintain, instead of (2), the weaker

(3a) $\quad l_i \leq \langle \downarrow j : j \leq i : x_j \rangle \quad$ and

(3b) $\quad r_i \leq \langle \downarrow j : j \geq i : x_j \rangle$ .

And this is what we shall do.

* * *

In view of the shapes of our guards (1), progress of the multiprogram is best served when the $l$'s and $r$'s are chosen as large as possible. In particular, there is no point in maintaining (3) by decreasing $l_i$ or $r_i$. This, we will have to bear in mind.

By the shape of the network, there are only few meaningful possibilities for updating $l_i$. If Mach.$i$ is to keep track of the value for $l_i$, and Mach.$(i-1)$ for $l_{i-1}$, it is sweetly reasonable that we rewrite (3a) into

$$l_i \leq \langle \downarrow j : j \leq i-1 : x_j \rangle \downarrow x_i \ , \qquad 1 \leq i \ ,$$

which, by (3a) $(i := i-1)$ is implied by

$$l_i \leq l_{i-1} \downarrow x_i \ ;$$

and this is established by the assignment

$$l_i := l_{i-1} \downarrow x_i \ , \qquad 1 \leq i \ .$$

For Mach.0 we find that

$$l_0 := x_0$$

maintains (3a).

Now, fortunately, if we stick to these assignments to the $l$'s, the $l$'s are never decreased: $l_0$ does not decrease because $x_0$ doesn't, and $l_i$, $1 \leq i$, does not decrease because neither $x_i$ nor – by induction – $l_{i-1}$ does so.

* * *

The next problem is how to superimpose these assignments on our existing multiprogram. For reasons of simplicity, we decide to equip Mach.$i$ with a co-component

$$\text{CL}.i : \qquad *[\ l_i := l_{i-1} \downarrow x_i\ ] \ , \qquad 1 \leq i$$

running in parallel with Mach.$i$ and the rest of the system, and continuously updating $l_i$. For reasons of symmetry we also introduce a co-component for updating $r_i$. We thus arrive at the solution depicted in Figure 0.

| | |
|---|---|
| Pre: | $\langle \forall j :: x_j = 0 \wedge l_j = 0 \wedge r_j = 0 \rangle$ |
| Inv: | $\langle \forall i :: l_i \leq \langle \downarrow j : j \leq i : x_j \rangle$<br>$\wedge\, r_i \leq \langle \downarrow j : j \geq i : x_j \rangle \rangle$ |
| Mach.$i$ | $(0 \leq i \leq N)$:<br>$*$ [ if $x_i \leq l_i \downarrow r_i \rightarrow$ skip fi<br>; { $Z_i : \langle \forall j :: x_i \leq x_j \rangle$ } $S_i$<br>; $x_i := 1 + x_i$<br>] |
| CL.$i$ | $(1 \leq i \leq N)$:<br>$*$ [ $l_i := l_{i-1} \downarrow x_i$ ] |
| CL.0 | $*$ [ $l_0 := x_0$ ] |
| CR.$i$ | $(0 \leq i \leq N-1)$:<br>$*$ [ $r_i := r_{i+1} \downarrow x_i$ ] |
| CR.$N$: | $*$ [ $r_n := x_N$ ] |

Figure 0.

* * *

Our remaining obligation is to investigate whether or not this solution exhibits the danger of total deadlock. We shall argue that it does not.

To that end, assume that all Mach.$i$ are stuck in their guarded skips. As a result all $x_i$ are constant. Because the co-components CL.$i$ continue to operate, within a finite number of steps all $l_i$ will be equal to some $x$-value: $l_0$ by construction, and $l_i$, $1 \leq i$, by induction and construction. Symmetrically, all $r_i$ will eventually be equal to some $x$-value. As a consequence, a Mach.$i$ that holds the minimal $x$-value then has a stably true guard and can therefore proceed.

* * *

It very much depends on the architecture of the executing machinery whether or not our solution is acceptable. If it is acceptable, that is fine, and if it is not we may further elaborate on it.

Just for the sake of the argument and for showing how we wish to reason about further detailing our solution, we shall indicate how the ever growing integer variables $x$, $l$, and $r$ can be eliminated from the program.

First, consider Mach.$i$ with the following annotation

$$
\begin{array}{l}
*[\ \underline{\text{if}}\ x_i \leq l_i \downarrow r_i \rightarrow \text{skip}\ \underline{\text{fi}} \\
\quad ;\ S_i \\
\quad \{\ x_i \leq l_i\ \} \\
\quad ;\ x_i := 1 + x_i \\
\quad ]\ .
\end{array}
$$

Assertion $x_i \leq l_i$ is locally correct – from the guard –, and it is globally correct because $l_i$ does not decrease. As a consequence, relation

$$x_i \leq 1 + l_i$$

is an invariant of the system. From (3a), we also have the invariance of

$$l_i \leq x_i\ ,$$

and therefore we have

$$(4) \qquad 0 \leq x_i - l_i \leq 1\ .$$

Hence, the differences $x_i - l_i$ are very small.

From MB – $\langle \forall i, j :: x_i \leq 1 + x_j \rangle$ – we see that the differences between the $x$-values are very small as well. What about the differences between $l$-values?

Assignment $l_i := l_{i-1} \downarrow x_i$ in CL.$i$ establishes

$$l_i \leq l_{i-1}\ ,$$

and because $l_{i-1}$ does not decrease, this relation is maintained. So, the $l$-sequence is descending. Moreover, we observe

$$
\begin{array}{ll}
 & l_{i-1} \\
\leq & \quad \{\ (3a)\ (i := i - 1)\ \}
\end{array}
$$

$$
\begin{array}{ll}
 & x_{i-1} \\
\leq & \{\ \text{MB}\ \} \\
 & 1 + x_i \\
\leq & \{\ (4)\ \} \\
 & 2 + l_i\ ,
\end{array}
$$

and hence we have

(5) $\quad l_i \leq l_{i-1} \leq 2 + l_i\ .$

(That the difference $l_{i-1} - l_i$ can indeed assume the value 2 follows from a simulation, not given here.)

The question now is how we can use (4) and (5) to eliminate the $x$'s and the $l$'s from our program texts.

In view of (4) and the shape of Mach.$i$, we introduce new variables $d_i$ and $e_i$, coupled to the old ones by

$$d_i = x_i - l_i \quad \text{and} \quad e_i = x_i - r_i\ ,$$

in terms of which Mach.$i$ can readily be rewritten as

$$
\begin{array}{lll}
\text{Mach.}i\ : & * [\ \underline{\text{if}}\ d_i \leq 0 \wedge e_i \leq 0 \rightarrow \text{skip}\ \underline{\text{fi}} \\
 & \quad ;\ S_i \\
 & \quad ;\ d_i, e_i := 1, 1 \\
 & \qquad\quad (\text{or}\ d_i, e_i := 1 + d_i, 1 + e_i) \\
 & ]\ .
\end{array}
$$

(The two-valued $d$'s and $e$'s can be implemented by booleans or by circuitry, if so desired.)

For the co-components the situation is slightly more difficult. Because $l_i$ does not decrease, we can rewrite

$$\text{CL.}i\ : \quad * [\ l_i := l_{i-1} \downarrow x_i\ ]$$

into the equivalent

$$
\begin{array}{lll}
\text{CL.}i\ : & * [\ \underline{\text{if}}\ l_i < l_{i-1} \downarrow x_i \rightarrow l_i := l_{i-1} \downarrow x_i \\
 & \quad [\!]\ l_i \geq l_{i-1} \downarrow x_i \rightarrow \text{skip} \\
 & \quad \underline{\text{fi}} \\
 & ]\ ,
\end{array}
$$

and this one into the equivalent

$$\text{CL}.i: \quad *[\ \underline{\text{if}}\ l_i < l_{i-1} \downarrow x_i \rightarrow l_i := l_{i-1} \downarrow x_i\ \underline{\text{fi}}\ ]\ .$$

Now, let us spell out the guard:

$$\begin{array}{ll} & l_i < l_{i-1} \downarrow x_i \\ \equiv & \quad \{\ \text{def. } \downarrow\ \} \\ & l_1 < l_{i-1} \wedge l_i < x_i \\ \equiv & \quad \{\ (4)\ \} \\ & l_i < l_{i-1} \wedge 1 + l_i = x_i \qquad (*) \\ \equiv & \quad \{\ (5)\ \}\ \{\ d_i = x_i - l_i\ \} \\ & l_i \neq l_{i-1} \wedge d_i = 1\ . \end{array}$$

From the line marked $(*)$ we conclude that the guard implies $l_{i-1} \downarrow x_i = 1 + l_i$, so that the assignment $l_i := l_{i-1} \downarrow x_i$ can be simplified to $l_i := 1 + l_i$. For the sake of maintaining $d_i = x_i - l_i$, we have to expand it to $l_i, d_i := 1 + l_i, 0$ (or: $l_i, d_i := 1 + l_i, -1 + d_i$).

Furthermore, because the guard is stable – i.e. cannot be falsified by the other components of the multiprogram – there is no need to embed its evaluation and the subsequent assignment into one atomic statement: they can be uncoupled (see [FvdS94]). Thus, we get as our next approximation for CL.$i$

$$\begin{array}{ll} \text{CL}.i: & *[\ \underline{\text{if}}\ l_i \neq l_{i-1} \wedge d_i = 1 \rightarrow \text{skip}\ \underline{\text{fi}} \\ & \quad ;\ l_i, d_i := 1 + l_i, 0 \\ & ]\ . \end{array}$$

And from this we clearly see that, as far as the values of $l_i$ and $l_{i-1}$ are concerned, we are only interested in their being different or not.

In view of this latter observation and of (5), we now introduce new variables $\lambda_i$ coupled to the old ones by

$$\lambda_i = l_i \ \underline{\text{mod}}\ 3\ .$$

Then, $l_i \neq l_{i-1}$ can be rewritten as $\lambda_i \neq \lambda_{i-1}$ and $l_i := 1 + l_i$ as $\lambda_i := (1 + \lambda_i)\ \underline{\text{mod}}\ 3$.

(The only thing that matters about the 3 is that it is larger than the 2 occurring in (5). It is precisely here where we use the limited range for $l_{i-1} - l_i$.)

Summarizing, we arrive at the algorithm depicted in Figure 1.

| | | |
|---|---|---|
| Pre: | $\langle \forall j :: d_i = 0 \wedge e_i = 0 \wedge \lambda_i = 0 \wedge \rho_i = 0 \rangle$ | |
| Mach.$i$ | $(0 \leq i \leq N)$: | $*$ [ if $d_i \leq 0 \wedge e_i \leq 0 \rightarrow$ skip fi<br>; $S_i$<br>; $d_i, e_i := 1, 1$<br>] |
| Cl.$i$ | $(1 \leq i \leq N)$: | $*$ [ if $\lambda_i \neq \lambda_{i-1} \wedge d_i = 1 \rightarrow$ skip fi<br>; $\lambda_i, d_i := (1 + \lambda_i) \bmod 3, 0$<br>] |
| CL.0 | : | $*$ [ if $d_0 = 1 \rightarrow$ skip fi<br>; $\lambda_0, d_0 := (1 + \lambda_0) \bmod 3, 0$<br>] |
| CR.$i$ | $(0 \leq i \leq N - 1)$: | $*$ [ if $\rho_i \neq \rho_{i+1} \wedge e_i = 1 \rightarrow$ skip fi<br>; $\rho_i, e_i := (1 + \rho_i) \bmod 3, 0$<br>] |
| CR.$N$ | : | $*$ [ if $e_N = 1 \rightarrow$ skip fi<br>; $\rho_N, e_N := (1 + \rho_N) \bmod 3, 0$<br>] |

Figure 1.

So much for this particular elaboration. Of course, many other variations or deviations are possible, but the point is that we wanted to illustrate how computing science can nowadays handle programming problems that – not too long ago – were frighteningly complex. It is pleasing to observe that for deriving programs like the one in this note we only need very simple formalisms, while at the same time it is hard to conceive how an operational mind could ever arrive at such solutions.

* * *

We kindly and respectfully offer this design to Prof.Dr. F.E.J. Kruseman Aretz, who always was and still is a dignified professor of computing science, an intellectual of high standing, a modest yet excellent scientist, and a master in teaching and education. It is our departed J.L.A. van de Snepscheut who once said of him: "He is the best teacher I ever had!".

Acknowledgement. We thank R.W. Bulterman for proposing this algorithm for a tree-shaped network.

WF195 W.H.J. Feijen, The multibound, Technical Note, Eindhoven University of Technology, Jan '95.

FvdS94 F.W. van der Sommen, Multiprogram Derivations, Master's Thesis, Eindhoven University of Technology, Oct '94.

# De meestertruuk

Peter van Emde Boas

In de jaren rond 1970, toen een student nog gewoon wiskunde met natuurkunde kon studeren en de Informatica als vak nog niet was uitgevonden (althans niet onder die naam in het Koninkrijk der Nederlanden) bekleedde Kruseman Aretz een aantal jaren een leeropdracht als bijzonder hoogleraar aan de Universiteit van Amsterdam. De colleges die ik bij die gelegenheid van hem gevolgd heb vormden voor mij in feite de eerste kennismaking met het vak, en daarmee een van de redenen (zij het niet de doorslaggevende) dat ik in 1971 de voorziene onderzoekslijn in de zuivere wiskunde opgaf om in plaats daarvan in 1974 te promoveren over een onderwerp in de complexiteitstheorie (maar dat is een ander verhaal).

Van de door Kruseman Aretz verzorgde colleges heb ik er, voorzover ik nu kan nagaan, drie gevolgd. In de beschikbare uren (2 uur per week gedurende twee trimesters) heb ik zeer veel geleerd van de fundamenten van het vak. Ik leerde te begrijpen hoe een computer in feite werkt, over de grondbeginselen van de theorie van formele talen, wat een compiler is en hoe die kan werken, en hoe een computer tot een bruikbaar instrument wordt door en een operating systeem op te zetten, en hoe een dergelijk systeem functioneert. Dit alles was opgehangen aan de beschrijving van het destijds op het Mathematisch Centrum draaiende Algol 60 systeem voor de EL X8 machine.

Het is opvallend te beseffen hoeveel van de kennis die ik in die periode van Kruseman Aretz heb opgedaan vijftien jaar later, toen ik voor tweedejaars Informatici aan de UvA een cursus verzorgde over programmeertaal concepten (een vak dat helaas bij latere curriculumwijzigingen ter ziele is gegaan) nog steeds bruikbaar was bij het verstrekken van extra informatie om het leerboek dat bij dit vak gebruikt werd aan te vullen.

Ik wil de gelegenheid van de samenstelling van dit liber amicorum dan ook graag gebruiken om een van deze antieke stukjes kennis (waarvan ik vermoedelijk de laatste ben die dit probleem in Amsterdam aan studenten heeft getracht uit te leggen) aan de vergetelheid ontrukken.

Het betreft hier de truuk die de Algol 60 programmeur in staat stelt om binnen een Algol 60 programma een potentieel oneindige Turingmachine band te simuleren. Wij zoeken hierbij een echte oplossing en niet zoiets als de bedrog methode bestaande uit het declareren van een "voldoend groot" array - dat is geen correcte oplossing en eenieder die deze oplossing voorstelt geeft er blijk van de schoonheid van het probleem niet te hebben ingezien.

In Algol 60 worden alle variabelen en arrays gedeclareerd bij de ingang van een blok of een procedure. Afmetingen van arrays moeten bij blokingang bekend zijn, en kunnen niet worden veranderd tijdens hun levensduur. Het is wel mogelijk een blok of een procedure te verlaten en in een later stadium opnieuw binnen te gaan waarbij de afmetingen zijn veranderd, maar dan is de inhoud van het array intussen verloren gegaan. Dat is dus geen oplossing.

Omdat in Algol 60 recursieve procedures zijn toegestaan, en deze procedures in het berekeningsmodel en de implementatie ervan gebruik maken van een stapel bestaat er in feite toch een vorm van een gegevensstructuur die tijdens de berekening kan groeien: de stapel. Helaas heeft de stapel de eigenschap dat je slechts de top ervan kunt gebruiken; wil je lezen wat ergens diep onder de top ligt opgeslagen dan moet je de tussenliggende informatie weggooien. De mij later geleerde methode om een Turingmachine band te simuleren met twee stapels is helaas ook niet te gebruiken, omdat de stapel die wordt opgebouwd via aanroepen van recursieve procedures uniek is; de programmeur kan niet een deel van de stapel opbouwen terwijl hij elders dezelfde stapel afbreekt.

Een methode die wel werkt is het gebruik van een recursieve procedure waarin bij iedere aanroep een locaal array wordt gedeclareerd van een groeiende afmeting. Via parameters kan de oude inhoud worden meegegeven en worden gecopieerd in de nieuwe tape. Helaas groeit bij deze methode het feitelijke geheugengebruik als de som van alle gebruikte bandsegmenten. Laten wij de segmenten lineair groeien dan is het totale geheugengebruik kwadratisch hetgeen bepaald onwenselijk genoemd moet worden. Tegenwoordig zal iedereen die enig inzicht in het ontwerp van algoritmen onmiddelijk opmerken dat een handige programmeur in deze situatie zal besluiten bij iedere aanroep de lengte van het bandsegment te verdubbelen, waarna slechts een constante factor geheugen overhead resteert. Toch blijft het maar een onelegante oplossing vanwege het copieren van bandinhouden.

Nadat dit probleem in een eerder stadium was gebruikt als tentamenopgave (ik herinner mij dat o.a. Willem Paul de Roever deze opgave werd voorgeschoteld), kregen in een later jaar de studenten de oplossing uitgelegd op het college.

De Meestertruuk maakt gebruik van het parametermechanisme "call by name" dat de ontwerpers van Algol 60 zo elegant in de taal hebben opgenomen. Een mechanisme dat zo natuurlijk werd geacht te zijn dat de Algol 60 programmeur (in tegenstelling tot de "call by value") niet eens geacht wordt te specificeren wanneer hij dit mechanisme gebruikt (het is de default; call by value moet wel worden gespecificeerd).

Voor de goede orde citeer ik nog even de werking van call by name: De relevante passages in het Revised Report leggen dit als volgt uit:
*4.7.3.2 Name replacement (call by name)*

*Any formal parameter not quoted in the value list is replaced, throughout the procedure body, by the corresponding actual parameter, after enclosing this latter in parentheses wherever syntactically possible. Possible conflicts between identifiers inserted through this process and other identifiers will be avoided by suitable systematic changes of the formal*

*ol local identifiers involved.*
*4.7.3.3 Body replacement and execution.*

*Finally the procedure body, modified as above, is inserted in place of the procedure statement and executed. If the procedure is called from a place outside the scope of any non-local quantity of the procedure body the conflicts between the identifiers inserted through this process of body replacement and the identifiers whose declarations are valid at the place of the procedure statement or function designator will be avoided through suitable changes of the latter identifiers.*

Op het eerste gezicht helemaal niet lastig, maar het geniep zit in de kleine lettertjes waarin wordt uitgelegd hoe de alpha-conversie demon in actie komt bij name-clashes.

Het in 1983-85 door mij gebruikte leerboek vond met dit alles call by name maar een lastig mechanisme en wist er weinig pakkends over te vertellen. In het bijzonder niet wat voor mooie en gemene dingen je ermee kunt doen. Wel werd ten onrechte gesuggereerd dat call by name iets te maken heeft met de onverwachte uitkomst van een procedureaanroep als:
`swap( a[a[1]], a[a[2]] )`
in de context van de procedure definitie[1]:

```
PROCEDURE swap( x,y ); INTEGER x,y ;
BEGIN INTEGER temp;
temp := x; x:= y; y := temp
END;
```

Dezelfde onbedoelde uitkomst doet zich echter ook voor als de drie onderstaande toekenningsopdrachten rechtstreeks in de code worden uitgeschreven.
`temp:= a[a[1]]; a[a[2]]:= a[a[1]]; a[a[1]]:= temp`

Het gaat hier in feite om het probleem van de assignment aan een arrayelement die zich slechts laat begrijpen als een assignment aan een arrayvariabele als geheel. Ook dat is echter een ander verhaal.

De mooie dingen die met call by name mogelijk worden en die mij zijn bijgebleven zijn de mogelijkheid van "lazy evaluatie" (name parameters worden pas uitgerekend als ze nodig zijn), de truuk van Jensen (waar een procedure een constant ogende name parameter meekrijgt bij declaratie, waarvoor bij aanroep een zich als functie gedragende expressie wordt meegegeven als actuele parameter), en de Meestertruuk.

De Meestertruuk wordt beschreven door de onderstaande procedure, waarbij zij gebruikt wordt om een array a te vullen met een rij reals waarvan de lengte niet bekend is, maar

[1] Verschillen tussen de (lexicale) syntax van Algol 60 als hier gepresenteerd en de oorspronkelijk door mij geleerde komen voort uit mijn luiheid met betrekking tot de exploratie van de mogelijkheden van LaTeX, waarvoor excuses

bekend wordt door het inlezen van de waarde 0:

```
PROCEDURE ka( waarden, niveau, index); VALUE niveau;
REAL waarden; INTEGER niveau, index;
BEGIN
REAL invoer, huidigewaarde;
read(invoer);
IF invoer /= 0
THEN huidigewaarde := invoer;
ka( IF index = niveau THEN huidigewaarde ELSE waarden, niveau+1, index)
ELSE
# een programmafragment waarbij het array wordt gebruikt,
hierbij wordt het element a[i] gesimuleert door
de uitdrukking ( index := i; waarden ) #
END;
```

De oeraanroep van deze procedure vindt plaats in het programmafragment:

```
INTEGER index; REAL dum;
ka( dum, 1, index );
```

De werking van deze procedure berust op de mechanismen die in actie komen bij het oplossen van de door de recursie geintroduceerde name-clashes. `invoer` en `huidigewaarde` zijn locale variabelen van de procedure, zodat iedere aanroep ervan een eigen incarnatie van deze variabelen heeft. Gebruik van deze namen refereert dan ook aan de meest recente incarnatie, doch dit geldt niet voor het gebruik van deze namen in de doorgegeven name parameter, want daarbij refereert de naam aan haar betekenis bij de procedureaanroep, en die ligt buiten deze actuele incarnatie. Het is dit proces dat het mogelijk maakt via de parameter `waarden` in feite *alle* lagen van de stapel te bereiken.

In feite ontstaat na oplossen van de name-clashes op recursiediepte `k` een actuele uitdrukking voor de parameter `waarden` die zich gedraagt als:

```
IF index = k THEN hw.k
ELSE IF index = k-1 THEN hw.k-1
ELSE IF index = k-2 THEN hw.k-2
....
ELSE IF index = 1 THEN hw.1
ELSE dum
```

Hierbij zijn de incarnaties van de value parameter `niveau` vervangen door hun feitelijke waarden en staat `hw.i` voor de incarnatie op niveau `i` van de parameter `huidigewaarden`. Het moge duidelijk zijn (op basis van het inzicht dat de actuele parameter deze vorm heeft) dat inderdaad alle niveaus van de stapel nu toegankelijk zijn geworden. En dat is dus de Meestertruuk.

Ten slotte nog een vraag achteraf. Kunnen wij het patroon dat aan de Meestertruuk ten grondslag ligt ook aantreffen in de natuurlijke taal? Het is immers in de huidige tijd onaannemelijk dat je gesprekspartners nog op de hoogte zullen zijn van de duistere uithoeken van de taal Algol 60, en een gemeenheid in een onbekende taal leent zich nu eenmaal slecht als gespreksonderwerp. Ik heb uiteraard mijn hoofd over deze vraag gebogen met als resultaat de volgende oplossing.

Wij zijn er aan gewend als wij via de radio het weerbericht beluisteren dit te begrijpen als een voorspelling voor het weer van het komende etmaal. Het zou echter een fluitje van een cent zijn voor het KNMI om (zonder daar wezenlijk meer zendtijd voor te gebruiken) het weerbericht om te vormen tot een weer-rapport dat de gehele periode (vanaf de ingebruikname) bestrijkt. Hiertoe dient het weerbericht in de toekomst een formaat te krijgen zoals in de onderstaande voorspelling voor het weer van 3 September 1995:

*Als het vandaag 3 September 1995 is wisselend bewolkt met regenbuien; zie anders het weerbericht van gisteren.*

Men vulle voor "vandaag" een historische dag in, en na het invullen van voldoende veel oudere weersvoorspellingen zal de gewenste informatie boven water komen. Het oordeel over de gepastheid van dit voorbeeld laat ik uiteraard over aan de aanstaande emeritus.

# Enkele gedachten over de disciplines der electrotechniek en informatica

Loe Feijs

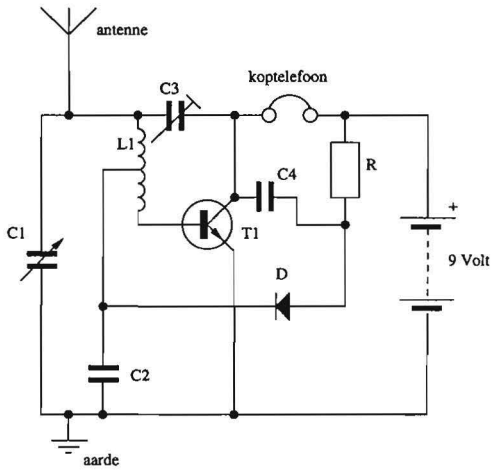
Het is een voorrecht om niet één, maar twee technisch-wetenschappelijke ontwikkelingen te mogen beleven die de aanduiding 'industriële revolutie' verdienen. De Electrotechniek en de Informatica zijn zulke ontwikkelingen, met elk een eigen discipline. Terwijl de Electrotechniek reeds eerder deze eeuw tot wasdom gekomen is, zijn de ontwikkelingen in de Informatica nog steeds stormachtig. Aan de TUE studeerde ik eerstgenoemde discipline en later kreeg ik de kans om, onder begeleiding van Prof. Frans Kruseman Aretz, een overstap te maken en betrokken te zijn bij studie en wetenschappelijk onderzoek in de Informatica. In deze bijdrage neem ik de gelegenheid om enkele voorzichtige vergelijkingen te maken tussen beide disciplines.

Laten wij eens uitgaan van de gedachte dat de centrale rol van 'schakelingen' in de Electrotechniek vergeleken kan worden met de centrale rol van 'algoritmen' in de Informatica.

Op een aantal punten klopt de vergelijking redelijk. Bijvoorbeeld is het een zeer veel voorkomende activiteit in de Electrotechniek om voor een gegeven probleemstelling een schakeling te ontwerpen, net zo als het in de Informatica vaak voorkomt dat een algoritme ontworpen moet worden. Een andere overeenkomst is dat zowel schakelingen als algoritmen beschouwd kunnen worden in abstracte vorm maar daarnaast vertaald kunnen worden in concretere vormen. Een schakeling gaat via gedetaileerde schakelschema's en via keuze van onderdelen in een of andere technologie naar een gedrukte bedrading en dan naar een realisatie die in bedrijf gesteld en beproefd kan worden. Een algoritme kan na detaillering van datarepresentaties uitgedrukt worden in een programmeertaal naar keuze en gecompileerd worden tot een instructiereeks, die op een passende processor ten uitvoer gebracht kan worden. Er is een schier onuitputtelijke verzameling van slimme schakelingen bedacht. Veel schakelingen kregen ook eigen namen (b.v. hotelschakeling, balansversterker, supperregeneratieve ontvanger) of ontvingen de naam van een uitvinder of onderzoeker (b.v. Colpitts oscillator, Heising modulator). Schakelingen kunnen op wetenschappelijke wijze geanalyseerd worden, gebruikmakend van natuurkundige wetten en wiskundige principes (o.a. wetten van Kirchhoff en Ohm, complexe getallen). Er is ook een enorme collectie van slimme algoritmen bedacht en ook veel algoritmen kregen eigen namen (b.v. bubble sort, quick sort, merge sort) of ontvingen de naam van een uitvinder of onderzoeker (b.v. Warshall's algoritme, Robinson's unificatiealgortime). Algoritmen kunnen op wetenschappelijke wijze geanalyseerd of zelfs ontworpen worden, gebruikmakend van logische regels en wiskundige principes (o.a. regels der predicatenlogica, homomorfismen).

Als we proberen verder te gaan wordt de vergelijking moeilijker. Het lukt niet om een representatief lijstje van schakelingen naast een representatief lijstje van algoritmen te leggen zodat er een zinnig een-eenduidig verband tussen beide lijstjes ontstaat. Natuurlijk niet, want het is niet zo dat de ene discipline een deelstukje of een vermomming van de andere is.

In plaats daarvan, laten we eens zo maar een schakeling bekijken en zo maar een algoritme: de reflex-ontvangerschakeling (Figuur 1) en het 'bubble sort' algoritme (Figuur 2).




Figuur 1: Reflex-ontvangerschakeling.

De schakeling is al oud en dateert vemoedelijk van rond 1925. Figuur 1 is een erg practische verschijningsvorm. Hoewel er slechts 10 onderdelen gebruikt zijn, is de schakeling niet echt eenvoudig. In Warrings boek *21 simple transistor radios you can build* worden enkele waarden gegeven: C1 $\leq$ 500 pF, C2 = 0,05 $\mu$F, C3 $\leq$ 10 pF, C4 = 1 nF, R = 1 M$\Omega$. Het hoogfrequente signaal komt vanaf de antenne in een afgestemde kring bestaande uit C1 en het bovenstuk van L1 (voor 'hoogfrequent' is C2 nagenoeg een kortsluiting). T1 is een transistor die dit signaal versterkt, en waarvan de linkeraansluiting (de basis) als ingang fungeert. Het versterkte signaal komt er via de aansluiting rechtsboven (de collector) weer uit en kan slechts via C4 naar de diode D die in staat is om een amplitude-gemoduleerd signaal gelijk te richten, zodat het laagfrequente signaal aan de bovenzijde van C2 ontstaat. Voor 'laagfrequent' is C2 een open circuit en L1 een kortsluiting en de transistor zal dus ook dit signaal versterken (onder de veronderstelling van lineariteit). Het versterkte signaal komt zo terecht in de koptelefoon. Natuurlijk is dit slechts een sterk vereenvoudigde uitleg: er valt heel wat te rekenen aan deze schakeling (maar daar gaan we nu niet op in).

Tot zover de schakeling en nu naar het algoritme.

```
program bubblesort;
    const n = 20;
    var a : array[1..n] of integer;
        i,j : integer;
    procedure ruil(var x,y : integer);
    var z : integer;
    begin {ruil}
      z := x; x := y; y := z
    end {ruil};
begin {bubblesort}
    for i := 1 to n do read(a[i]);
    for i := n downto 2 do
        for j := 2 to i do
            if a[j]<a[j-1] then ruil(a[j],a[j-1]);
    for i := 1 to n do writeln(a[i])
end.
```

Figuur 2: 'Bubble sort'-algoritme.

Het algoritme is ook al lang niet nieuw meer en dateert vermoedelijk van rond 1955. Figuur 2 is wederom een erg practische verschijningsvorm, ongeveer zoals gegeven in Cheng en Nienhuys' boek, *programmeren met Pascal (ook met TURBO)*. De regels voor invoer en uitvoer zijn niet weggelaten (net zoals de ontvanger ook inclusief antenne en koptelefoon is). Hoewel Figuur 2 slechts 16 regels telt, is het een bekend maar pittig klusje om voor een sorteeralgoritme middels invarianten de correctheid na te gaan. Een analyse van de efficiëntie vraagt serieuze wiskunde (we gaan hier nu niet op in).

Nu we een en ander zo heel concreet hebben gemaakt, kunnen we verder gaan met vergelijken. Allereerst een overeenkomst: beide voorbeelden stammen uit een relatief vroege fase van de opbloei van de betreffende discipline, en beide gaan dan ook uit van bijzondere efficiëntieoverwegingen. Bij de reflex-ontvangerschakeling is dit de overweging dat het kostbare versterkende element (de transistor) dubbel gebruikt kan worden: het signaal doorloopt tweemaal de transistor. Bij het 'bubble sort' algoritme is dit de overweging dat het sorteren in-situ gebeurt, dat wil zeggen, er worden geen hulp-arrays gebruikt, maar de ene array `a[1..n]` herbergt invoer, uitvoer en en alle optredende tussensituaties. De reflexontvanger is inmiddels in de vergetelheid geraakt: anno 1995 kijken we niet op duizend transistors meer of minder. Bij voorbeeld als we een versterker moeten maken die slechts 2,5 × moet versterken smijten we er een hele operationele versterker tegenaan die makkelijk een miljoen keer kan versterken, maar gebruiken tegenkoppeling om dit tot 2,5 te reduceren. Of we gebruiken een soort computer die dan vele malen per seconde de gedigitaliseerde monsters met 2,5 moet vermenigvuldigen. Komt er ook een tijd dat nodeloos array-gebruik ons niet meer kan schelen? Ik weet het niet zeker, maar ik denk van wel.

Nu een ander aspect. Het ligt voor de hand om het concept van terugkoppeling te zien als verwant aan het concept van iteratie (dan wel recursie). In de wiskunde van de concepten lijkt dit te kloppen: beide concepten geven aanleiding tot dekpuntvergelijkingen. Maar practisch ziet het er toch anders uit. Kijkend naar de reflexontvanger is het dubbelgebruik van T1 een soort terugkoppeling, maar er is nog een tweede terugkoppeling, die werkzaam is voor het hoogfrequente signaal: C3 voert signaal terug van T1 naar de ingangskring. De transistor keert de fase 180°, maar dat doet L1, die in het middel geaard is (althans voor 'hoogfrequent') óók, en dus hebben we hier een meekoppeling. Als de meekoppeling te groot is ontstaan problemen, maar anders is het effect ervan dat de versterking iets groter wordt en de kring iets selectiever. Van latere datum is het gebruik van terugkoppeling in de vorm van tegenkoppeling, met name wanneer subtielere doelen dan pure versterking nagestreefd worden, zoals lage vervorming en bedieningsgemak. Heel anders zijn de overwegingen bij de twee geneste `for` lussen in `bubblesort`. Ze zijn gewoon wezenlijk voor het algoritme, en het onderscheid tussen mee- en tegenkoppeling is niet van toepassing op dit algoritme. De binnenste `for` lus neemt het grootste element in `a[1..i]` mee 'omhoog' in de array (als een luchtbel). De buitenste lus voert dit proces zo vaak uit dat de hele array gesorteerd raakt.

Los van onze twee voorbeelden, nog enkele andere opmerkelijke verschillen tussen de Electrotechniek en de Informatica. Opvallend is dat elk een uitgebreid arsenaal aan transformatietechnieken hebben: Fourier-, Laplace-transformaties en dergelijke enerzijds en programma-transformatieregels (folding, loop-fusion, theorie der catamorfismen), process-algebraïsche wetten en dergelijke anderzijds. Maar er is een eigenaardig verschil: een programmatransformatie transformeert een algoritme in een ander algoritme (met behoud van gedrag), maar een Fourier-transformatie is een manier om het signaal (= het gedrag van de schakeling) te transformeren, in plaats van de schakeling zelf. Zullen er ook uitgebreide schakeling-transformatieregels komen of programma-gedragstransformaties? Ik weet het niet.

Thans enkele afsluitende opmerkingen. Ik denk dat beide disciplines zich verder zullen ontwikkelen, en ik hoop dat dat kan op een manier waarbij hun beoefenaars dat doen op basis van respect voor elkaar (dat hoop ik voor allerlei andere betrekkingen tussen mensen ook trouwens). Het valt niet te verwachten dat beide disciplines naar elkaar toegroeien in de zin dat de een in de ander op zal gaan. Maar wel zijn er vele koppelingen, met name daar waar technologie geleverd door de Electrotechniek gebruikt wordt in de Informatica, en omgekeerd (denk bijvoorbeeld aan CAD gereedschappen). En wat ook blijft is het gemeenschappelijke feit dat de bestudeerde en/of geconstrueerde systemen naast zeer abstracte, ook zeer concrete vormen kunnen aannemen; en dat dus abstractievermogen, ondersteund met wiskundige technieken, van groot belang blijft.

Graag wil ik mijn bijzondere waardering uitspreken voor Prof. Frans Kruseman Aretz en in het bijzonder ook voor zijn begeleiding bij een aantal van mijn stappen in de discipline van de Informatica.

# Grammaire à la grand-mère

Rik van Geldrop

## 1 Voor een heer van stand.

Speciaal voor jou, Frans, heb ik een dineetje van context-vrije grammatica's samengesteld. Een eenvoudige doch voedzame maaltijd, die bestaat uit correctheidsbewijzen van grammatica transformaties en constructies. De bereidingsmethode van deze bewijzen is niet het gebruikelijke "inductie naar het aantal afleidingsstappen" maar de fixed point calculus, zoals die (onder leiding van chef-kok Backhouse) tot stand is gekomen en gepubliceerd werd in "Information Processing Letters", vol.53, nr.3, 1995.

Als entrée heb ik gekozen voor de "substitutie regel". De consommé, altijd goed voor een predictive parser, is "links factorisatie". Plat de résistance is "Links recursie elimineren", een must voor top down parsing, en als dessert zijn er friandises bestaande uit "constructies op grammatica's". Elke gang van dit diner wordt in buffetvorm geserveerd. Neem maar zo vaak en zo veel als je wilt.

Voor de verteerbaarheid van het geheel: als ik in het vervolg het woord grammatica gebruik, bedoel ik een context-vrije grammatica $G = (V, T, S, P)$, met nonterminals $V$, terminals $T$, startsymbool $S$ en producties $P$. De verzamelingen $V$ en $T$ zijn disjunct en beiden eindig en niet leeg. Vat de producties van $G$ op als een stelsel (wederzijds recursieve) vergelijkingen (in de nonterminals) over een reguliere algebra. (In dit geval een productalgebra waarbij elke factor een reguliere algebra ( $\mathcal{P}.T^*$, de talen over $T^*$) is. $\mathcal{P}.T^*$ is geordend met inclusie, heeft taalconcatenatie $(\cdot)$ als associatieve vermenigvuldigingsoperator en $\{\varepsilon\}$ als eenheid van $(\cdot)$. Zoals bekend zijn $W\cdot$ en $\cdot W$ universeel disjunctief voor elke $W \in \mathcal{P}.T^*$.) Vanwege de vorm van de producties, samenstellingen van supremum- en vermenigvuldigingsoperaties, weten we via Knaster-Tarski dat het stelsel monotoon en dus oplosbaar is en een kleinste oplossing heeft. Deze kleinste oplossing legt voor iedere nonterminal een object vast in een reguliere algebra $(\mathcal{P}.T^*)$, de $S$-component hiervan is de taal voortgebracht door $G$.

De huidige fixed point calculus maakt het mogelijk componenten van kleinste oplossingen van monotone stelsels te berekenen. Voor binaire stelsels gaat dit rechtstreeks. Grotere moeten eerst tot binaire worden teruggebracht door vectoren van onbekenden te introduceren en gebruik te maken van isomorfieën op productruimten, maar dat past vast wel in jouw denkraam.

Een stelsel gedefinieerd door de producties van een grammatica is een speciaal geval van een monotoon stelsel en de aangekondigde grammatica transformaties zijn speciale gevallen van de stelsel transformaties, die verderop bewezen zullen worden. In deze bewijzen wordt geabstraheerd van de speciale vorm van "grammatica" stelsels en van de bijzondere rol die het startsymbool $S$ in een grammatica speelt; er wordt slechts aangenomen dat alle voorkomende functies en operatoren monotoon zijn.

Wat de notatie betreft: ik gebruik infix operatoren om de rechterleden van willekeurige producties aan te geven, maar, als de vorm van zo'n rechterlid een rol speelt, gebruik ik tevens prefix operatoren. Moet er verder nog vermeld worden dat grammaticaland de gewoonte heeft om de (algebraïsche) $\cdot$, $+$ en 1 door juxtapositie, $|$ en $\varepsilon$ voor te stellen?

Ter informatie aan Loes, mijn collega-kok: voor dit diner hoef je niets bijzonders in huis te hebben. Alleen formule (5) enkele malen flink opkloppen, aan de kook brengen en daarna, al roerende, goed laten afkoelen. Inderdaad, een glaasje witte wijn past er uitstekend bij!

## 2 CF-grammatica's en fixed point calculus.

Monotone functies op een volledig tralie hebben een kleinste fixed point. De belangrijkste eigenschap van een kleinste fixed point is dat het samenvalt met het kleinste prefix point (Knaster-Tarski), maar er zijn er nog veel meer. We noemen

(1) (*induction*) $\mu f \leq x \;\Leftarrow\; f.x \leq x$ for all $x$

(2) (*computation*) $f\,.\,\mu f = \mu f$

(3) (*rolling*) $\mu\langle x \mapsto f.(g.x)\rangle = f\,.\,\mu\langle x \mapsto g.(f.x)\rangle$

(4) (*diagonal*) $\mu\langle x \mapsto x \oplus x\rangle = \mu\langle x \mapsto \mu\langle y \mapsto x \oplus y\rangle\rangle$

voor iedere (monotone) binaire operator $\oplus$.

(5) (*mutual*) $\mu\langle (x,y) \mapsto (x \odot y, x \otimes y)\rangle = (\mu\langle x \mapsto x \odot p.x\rangle, \mu\langle y \mapsto q.y \otimes y\rangle)$

waarbij $p.x = \mu\langle y \mapsto x \otimes y\rangle$ en $q.y = \mu\langle x \mapsto x \odot y\rangle$ .

Als het functie domein niet alleen een volledig tralie, maar zelfs een reguliere algebra is, kunnen de eigenschappen van kleinste fixed points nog uitgebreid worden met o.a.

(6) (*snoc-cons*) $\mu\langle x \mapsto 1 + x \cdot b\rangle = \mu\langle x \mapsto 1 + b \cdot x\rangle$

(7) (*fusion*) $a \cdot \mu\langle x \mapsto 1 + x \cdot b\rangle = \mu\langle x \mapsto a + x \cdot b\rangle$

Gelijkheid van twee kleinste fixed points kan met (1) en (2) bewezen worden. Om constructieve redenen (overzetten van categorie theoretische constructies naar programma constructies) kiezen we echter voor "equational reasoning" in de nu volgende gelijkheidsbewijzen. (Dus *induction*, (1), zal hierbij niet worden gebruikt.)

## 2.1 Substitutie.

De meest elementaire grammatica transformatie is de "substitutie regel". Losjes omschreven houdt dit in dat het rechterlid van nonterminal $X$ gesubstitueerd mag worden in het rechterlid van nonterminal $Y$, zonder dat de betekenis van de grammatica hierdoor verandert. In de fixed point calculus correspondeert dit met

**Regel 8** $\mu\langle(x,y)\mapsto(x\odot y,\, x\otimes y)\rangle = \mu\langle(x,y)\mapsto(x\odot(x\otimes y),x\otimes y)\rangle$

**Bewijs** *Mutual*, (5), is voor zowel linker- als rechterlid van (8) het aangewezen gereedschap. Voor de helderheid gieten we het bewijs in de volgende vorm: pas (5) toe op beide leden van (8) en bewijs daarna de resterende verplichtingen (die dan per component geformuleerd zijn).

Ter verduidelijking van de instantiaties introduceren we operator $\oplus$, gedefinieerd door $x\oplus y = x\odot(x\otimes y)$. De benodigde hulpfuncties zullen in de hints worden gedefinieerd.

$$
\begin{array}{ll}
 & \mu\langle(x,y)\mapsto(x\oplus y,x\otimes y)\rangle \\
= & \quad \{\ \textit{mutual}\text{: def. } p'.x = \mu\langle y\mapsto x\otimes y\rangle \\
 & \qquad\qquad \text{def. } q'.y = \mu\langle x\mapsto x\oplus y\rangle\ \} \\
 & (\mu\langle x\mapsto x\oplus p'.x\rangle,\mu\langle y\mapsto q'.y\otimes y\rangle) \\
= & \quad \{\ \bullet\ \ \mu\langle x\mapsto x\oplus p'.x\rangle = \mu\langle x\mapsto x\odot p.x\rangle \\
 & \qquad \bullet\ \ \mu\langle y\mapsto q'.y\otimes y\rangle = \mu\langle y\mapsto q.y\otimes y\rangle\ \} \\
 & (\mu\langle x\mapsto x\odot p.x\rangle,\mu\langle y\mapsto q.y\otimes y\rangle) \\
= & \quad \{\ \textit{mutual}\text{: def. } p.x = \mu\langle y\mapsto x\otimes y\rangle \\
 & \qquad\qquad \text{def. } q.y = \mu\langle x\mapsto x\odot y\rangle\ \} \\
 & \mu\langle(x,y)\mapsto(x\odot y,\, x\otimes y)\rangle
\end{array}
$$

Resteert te bewijzen dat

$$\mu\langle x\mapsto x\oplus p'.x\rangle = \mu\langle x\mapsto x\odot p.x\rangle$$

$$\mu\langle y\mapsto q'.y\otimes y\rangle = \mu\langle y\mapsto q.y\otimes y\rangle$$

De eerste verplichting is simpel. Immers $p' = p$ en, uit de definitie van $\oplus$ en *computation* op $p.x$, volgt $x\oplus p.x = x\odot(x\otimes p.x) = x\odot p.x$.
De tweede verplichting is lastiger,

$$
\begin{array}{ll}
 & \mu\langle y \mapsto \mu\langle x \mapsto x \odot (x \otimes y)\rangle \otimes y\rangle \\
= & \quad \{ \ \textit{diagonal} \ \} \\
 & \mu\langle y \mapsto \mu\langle x \mapsto \mu\langle z \mapsto z \odot (x \otimes y)\rangle\rangle \otimes y\rangle \\
= & \quad \{ \ \text{def. q} \ \} \\
 & \mu\langle y \mapsto \mu\langle x \mapsto q.(x \otimes y)\rangle \otimes y\rangle \\
= & \quad \{ \ \textit{rolling} \ \} \\
 & \mu\langle y \mapsto \mu\langle x \mapsto q.x \otimes y\rangle\rangle \\
= & \quad \{ \ \textit{diagonal} \ \} \\
 & \mu\langle y \mapsto q.y \otimes y\rangle
\end{array}
$$

□

Hiermee is bewezen dat, in het rechterlid van de $x$-producties, elk voorkomen van nonterminal $y$ vervangen mag worden door het rechterlid van de $y$-producties. Het is mogelijk een substitutie regel te geven waarin slechts een of meerdere voorkomens van een nonterminal worden vervangen, maar daar gaan we nu niet op in.

### 2.2 Links factorisatie.

In grammatica's kan men nieuwe nonterminals introduceren en overtollige nonterminals elimineren. In de fixed point calculus zullen we dit uitdrukken met behulp van gelabelde projecties. (Het gebruik van gelabelde projecties maakt de volgorde van de vergelijkingen in een stelsel irrelevant, maar het legt wèl een claim op de te gebruiken dummies. Voor grammatica toepassingen is zo'n claim geen bezwaar omdat de dummies uit de nonterminals gekozen zullen worden.) Een voorbeeld van zo'n regel krijgen we door in (5), $x \otimes y := \alpha.x$ te substitueren. Voor de hulpfunctie $p$ krijgen we dan $p.x = \mu\langle y \mapsto \alpha.x\rangle = \alpha.x$.

$$(9) \quad \pi_x(\mu\langle (x,y) \mapsto (x \odot y, \alpha.x)\rangle) = \mu\langle x \mapsto x \odot \alpha.x\rangle$$

In het bijzonder, voor $x,y := (x,y),u$ ; $\alpha.(x,y) := h.(x,y)+k.(x,y)$ en $(x,y)\odot u := (f.(x,y) + g.(x,y)\cdot u \ , \ x \ominus y)$, zegt (9),

$$
\begin{array}{ll}
 & \pi_{(x,y)}(\mu\langle((x,y),u) \mapsto ((f.(x,y) + g.(x,y)\cdot u \ , \ x \ominus y) \ , \ h.(x,y)+k.(x,y))\rangle) \\
= & \quad \{ \ (9) \ \} \\
 & \mu\langle (x,y) \mapsto (f.(x,y) + g.(x,y)\cdot(h.(x,y)+k.(x,y)) \ , \ x \ominus y)\rangle \\
= & \quad \{ \ a\cdot \text{ over } + \ \} \\
 & \mu\langle (x,y) \mapsto (f.(x,y) + g.(x,y)\cdot h.(x,y) + g.(x,y)\cdot k.(x,y) \ , \ x \ominus y)\rangle
\end{array}
$$

wat, in grammatica terminologie, correspondeert met het introduceren van een nieuwe, niet recursieve, nonterminal $u$. Links factorisatie is daar een instantiatie van. Ter illustratie zullen we de "dangling-else" verwijderen. Neem $x,y,u := S,E,S'$; $f.(S,E) := a$; $g.(S,E) := i{\cdot}E{\cdot}t{\cdot}S$; $h.(S,E) := 1$; $k.(S,E) := e{\cdot}S$ en $S \ominus E := b$ dan volgt uit het bovenstaande dat de grammatica's $S \to a \mid iEtS \mid iEtSeS$, $E \to b$ en $S \to a \mid iEtSS'$, $S' \to \varepsilon \mid eS$, $E \to b$ equivalent zijn.

### 2.3 Links recursie elimineren.

De introductie van een nieuwe recursieve nonterminal komt voor bij het elimineren van links recursie. In dit verband spelen twee grammatica transformaties een rol. Een ervan zullen we nu in de fixed point calculus bewijzen. De andere gaat analoog.

**Regel 10**

$$\begin{array}{ll} & \mu\langle(x,y)\mapsto(f.(x,y)+x\cdot g.(x,y)\ ,\ x\ominus y)\rangle \\ = & \qquad \{\quad\} \\ & \pi_{(x,y)}(\mu\langle(x,u,y)\mapsto(f.(x,y)\cdot u\ ,\ 1+g.(x,y)\cdot u\ ,\ x\ominus y)\rangle) \end{array}$$

**Bewijs** Om (10) te kunnen bewijzen converteren we eerst het stelsel uit het rechterlid van ternair naar binair. Omdat, in het ternaire stelsel, $y$ onafhankelijk is van $u$, kiezen we voor de isomorfie $X\times Y\times Z \cong (X\times Y)\times Z$.Het toepassen van deze isomorfie levert niet alleen een binair stelsel, maar transformeert ook de projectie $\pi_{(x,y)}$ tot $\bar{\pi}_{(x,y)} = (\pi_x \circ \pi_{(x,u)})\times\pi_y$. Het rechterlid van (10) wordt dus herschreven tot

(11) $\bar{\pi}_{(x,y)}\ (\mu\langle((x,u),y)\mapsto((f.(x,y)\cdot u\ ,\ 1+g.(x,y)\cdot u),x\ominus y)\rangle)$

Nu kunnen we, m.b.v. *mutual*, bewijzen dat (11) en het linkerlid van (10) gelijk zijn. We kiezen dezelfde bewijsstructuur als bij (8).

Ter verduidelijking van de instantiaties introduceren we de operatoren $\odot$, $\otimes$ en $\oplus$ gedefinieerd door

$$(x,u)\odot y = (f.(x,y)\cdot u\ ,\ 1+g.(x,y)\cdot u)$$

$$(x,u)\otimes y = x\ominus y$$

$$x\oplus y = f.(x,y)+x\cdot g.(x,y)$$

De benodigde hulpfuncties worden weer in de hints gedefinieerd.

$$
\begin{array}{ll}
 & \bar{\pi}_{(x,y)}\ (\mu\langle((x,u),y)\mapsto((x,u)\odot y,(x,u)\otimes y)\rangle) \\
= & \quad \{\ \textit{mutual}:\ \text{def}\ p.(x,u) = \mu\langle y\mapsto(x,u)\otimes y\rangle \\
 & \qquad\qquad\ \ \text{def}\ q.y = \mu\langle(x,u)\mapsto(x,u)\odot y\rangle\ \ \} \\
 & \bar{\pi}_{(x,y)}\ (\mu\langle(x,u)\mapsto(x,u)\odot p.(x,u)\rangle, \mu\langle y\mapsto q.y\otimes y\rangle) \\
= & \quad \{\ \text{def}\ \bar{\pi}_{(x,y)},\ \text{projecties}\ \ \} \\
 & (\pi_x(\mu\langle(x,u)\mapsto(x,u)\odot p.(x,u)\rangle), \mu\langle y\mapsto q.y\otimes y\rangle) \\
= & \quad \{\ \bullet\ \ \pi_x(\mu\langle(x,u)\mapsto(x,u)\odot p.(x,u)\rangle) = \mu\langle x\mapsto x\oplus p'.x\rangle\ , \\
 & \qquad \bullet\ \ \mu\langle y\mapsto q.y\otimes y\rangle = \mu\langle y\mapsto q'.y\ominus y\rangle\ ,\ \ \} \\
 & (\mu\langle x\mapsto x\oplus p'.x\rangle, \mu\langle y\mapsto q'.y\ominus y\rangle) \\
= & \quad \{\ \textit{mutual}:\ \text{def}\ p'.x = \mu\langle y\mapsto x\ominus y\rangle \\
 & \qquad\qquad\ \ \text{def}\ q'.y = \mu\langle x\mapsto x\oplus y\rangle\ \ \} \\
 & \mu\langle(x,y)\mapsto(x\oplus y, x\ominus y)\rangle
\end{array}
$$

Resteert te bewijzen dat

(12) $\pi_x(\mu\langle(x,u)\mapsto(x,u)\odot p.(x,u)\rangle) = \mu\langle x\mapsto x\oplus p'.x\rangle$

(13) $\mu\langle y\mapsto q.y\otimes y\rangle = \mu\langle y\mapsto q'.y\ominus y\rangle$

Voor het bewijs van (13) is het, gezien de definitie van $\otimes$, voldoende om aan te tonen dat de eerste component van $q.y$ gelijk is aan $q'.y$. Voor $q.y$ met dummy $(x,u)$ wil dat zeggen

(14) $\pi_x(\mu\langle(x,u)\mapsto(x,u)\odot y\rangle) = \mu\langle x\mapsto x\oplus y\rangle$ voor alle $y$

Als volgt

$$
\begin{array}{ll}
 & \pi_x(\mu\langle(x,u)\mapsto(f.(x,y)\cdot u\ ,\ 1+g.(x,y)\cdot u)\rangle) \\
= & \quad \{\ \textit{mutual},\ \text{projectie}\ \ \} \\
 & \mu\langle x\mapsto f.(x,y)\cdot\mu\langle u\mapsto 1+g.(x,y)\cdot u\rangle\rangle \\
= & \quad \{\ \textit{snoc-cons}\ \ \} \\
 & \mu\langle x\mapsto f.(x,y)\cdot\mu\langle u\mapsto 1+u\cdot g.(x,y)\rangle\rangle \\
= & \quad \{\ \textit{fusion}\ \ \} \\
 & \mu\langle x\mapsto\mu\langle u\mapsto f.(x,y)+u\cdot g.(x,y)\rangle\rangle \\
= & \quad \{\ \textit{diagonal}\ \ \} \\
 & \mu\langle x\mapsto f.(x,y)+x\cdot g.(x,y)\rangle
\end{array}
$$

Voor het bewijs van (12) merken we eerst op dat $p.(x,u) = p'.x$:

$$p.(x,u) = \mu\langle y\mapsto(x,u)\otimes y\rangle = \mu\langle y\mapsto x\ominus y\rangle = p'.x$$

Dan

$$
\begin{array}{ll}
 & \pi_x(\mu\langle(x,u)\mapsto(x,u)\odot p'.x\rangle) \\
= & \quad \{ \ \textit{diagonal} \ \} \\
 & \pi_x(\mu\langle(x,u)\mapsto\mu\langle(z,v)\mapsto(z,v)\odot p'.x\rangle\rangle) \\
= & \quad \{ \ \text{def } q \text{ met dummy } (z,v) \ \} \\
 & \pi_x(\mu\langle(x,u)\mapsto q.(p'.x)\rangle) \\
= & \quad \{ \ q.(p'.x) \text{ onafhankelijk van } u, (9) \ \} \\
 & \pi_x(\mu\langle x\mapsto\pi_z(q.(p'.x))\rangle) \\
= & \quad \{ \ (14) \ \} \\
 & \mu\langle x\mapsto q'.(p'.x)\rangle \\
= & \quad \{ \ \text{def } q' \ \} \\
 & \mu\langle x\mapsto\mu\langle z\mapsto z\oplus p'.x\rangle\rangle \\
= & \quad \{ \ \textit{diagonal} \ \} \\
 & \mu\langle x\mapsto x\oplus p'.x\rangle
\end{array}
$$

□

Als nu, in een grammatica, $f.(x,y)$ de alternatieven van de $x$-producties beschrijft die niet met $x$ beginnen, dan wordt door (10) de links recursie van nonterminal $x$ geëlimineerd. Merk op dat het verwijderen van ambiguïteiten zoals $STATS \rightarrow STAT \mid STATS;STATS$ hier ook onder valt. (Neem aan dat $STAT \rightarrow b$ de $STAT$-producties voorstellen en instantieer in (10): $x,y,u := STATS, STAT, RSTATS$; $f.(STATS, STAT) := STAT$; $g.(STATS, STAT) := ;\cdot STATS$ en $STATS \ominus STAT := b$, dan volgt dat de grammatica's $STATS \rightarrow STAT \mid STATS;STATS$, $STAT \rightarrow b$ en $STATS \rightarrow STAT\ RSTATS$, $STAT \rightarrow b$, $RSTATS \rightarrow \varepsilon \mid ;\ STATS\ RSTATS$ equivalent zijn.)

### 2.4 Nieuwe grammatica's uit bestaande.

Voor constructies op grammatica's zoals concatenatie, vereniging en Kleene-star, zijn weer twee andere varianten van (5) nuttig:

(15) $\mu\langle(x,y)\mapsto(f.y, g.y)\rangle = (f\,.\,\mu g\,,\,\mu g)$

(16) $\mu\langle(x,y)\mapsto(f.x, g.y)\rangle = (\mu f\,,\mu g)$

Ter illustratie geven we de concatenatie: Als $L_1$ en $L_2$ de talen zijn gegenereerd door $(V_1,T,S_1,P_1)$ en $(V_2,T,S_2,P_2)$ respectievelijk, waarbij $V_1 \cap V_2 = \emptyset$, dan wordt de taal $L_1 \cdot L_2$ gegenereerd door de grammatica $G = ((V_1 \cup V_2 \cup \{S\}), T, S, (P_1 \cup P_2 \cup \{S \rightarrow S_1\ S_2\}))$, waarbij $S$ een nieuwe nonterminal is.

Zij $\underline{u} = h.\underline{u}$ het stelsel dat correspondeert met $P_1$. (Dus $S_1$ is een component van $\underline{u}$ en $\pi_{S_1}(\mu h) = L_1$). Analoog voor $\underline{v} = k.\underline{v}$ als stelsel voor $P_2$. Het stelsel dat correspondeert

met $P_1 \cup P_2$ is dan $(\underline{u},\underline{v}) = (h.\underline{u}, k.\underline{v})$ en heeft, volgens (16), $(\mu h, \mu k)$ als kleinste oplossing. Instantieer nu (15) met $x := S$, $y := (\underline{u},\underline{v})$, $f.(\underline{u},\underline{v}) := S_1 \cdot S_2$ en $g.(\underline{u},\underline{v}) := (h.\underline{u}, k.\underline{v})$, dan genereert $G$ de taal

$$\pi_{S_1}(\mu g) \cdot \pi_{S_2}(\mu g) = \pi_{S_1}(\mu h) \cdot \pi_{S_2}(\mu k) = L_1 \cdot L_2$$

## 3 Als je begrijpt wat ik bedoel.

Jou kennende, Frans, neem ik aan dat je deze calculationele benadering van grammatica's wel kunt waarderen. Mocht je behoefte hebben aan meer: er is nog genoeg. Bijvoorbeeld overtollige nonterminals elimineren, grammatica's transformeren naar een vorm waarin $S$ de enige nonterminal is die de lege string kan opleveren, enz. enz. Ook bij het onderwijs in automatentheorie en compilers heb ik de fixed point calculus regelmatig toegepast. Door al deze exercities ben ik er meer en meer van overtuigd geraakt dat er een kern van waarheid schuilt in de zegswijze: "de kracht van een verhaal zit hem in de wereld van de vergelijking". Als je begrijpt wat ik bedoel.

# Ieder afscheid is ook een nieuw begin ...

Dieter Hammer

Beste Frans,

Het is mij een bijzonder genoegen een kleine bijdrage te leveren aan dit vriendenboek ter gelegenheid van je officiële afscheid van de vakgroep.

Hoewel je voor mij een van de nestoren van de vakgroep bent, zal ik hier niet op je wetenschappelijke prestaties ingaan; dat hebben anderen in dit boekje zeker al veel beter opgeschreven dan ik het als relatief jong lid, ooit zou kunnen doen. Ik vind het even belangrijk om een poging te doen je menselijke kanten iets belichten.

Frans, door je stille en bescheiden aard ben je zeker niet iemand die zich meteen opdringt. Ik kende je lange tijd alleen als promotor en van verschillende vergaderingen. Wat mij daarbij meteen opviel was je systematische manier om tegen problemen aan de kijken: in de eerste plaats niet als betrokkene, maar objectief waarnemend, omvattend analyserend en naar een voor alle betrokkenen "optimale" oplossing zoekend. Je keek daarbij niet alleen naar de feiten, maar ook naar voorbeelden uit het het verleden, naar de vermoedelijke uitwerkingen van een opzet in de toekomst, zowel voor de direkt betrokkenen als ook voor de omgeving. Deze principiële instelling werd je niet altijd in dank afgenomen, maar ik denk wel dat je gelijk had.

Mede door het feit dat je als deeltijdshoogleraar slechts een dag per week aanwezig was, maar ook door de verregaande specialisering op onze universiteiten, duurde het jaren voor wij samen een klus te klaren hadden: een aanzet voor een toekomstvisie van de vakgroep. Naast je gedegen werkwijze viel mij daarbij vooral op hoe open je voor nieuwe ideeën stond. Ik was eerlijk gezet een beetje bang dat je de empirisch/heuristische methoden die ik uit de organisatieleer aan kwam dragen af zou keuren, als slecht gedefinieerd, niet formeel, en niet verenigbaar met de attitude van een wiskundig georiënteerde vakgroep. Tot mijn grote opluchting leverde dit geen enkel probleem op en wij kwamen tot een vruchtbare samenwerking waar wij, zo ik meen, allebei plezier in hadden.

Vrij vlug daarna kwam de moeilijke periode van de reorganisatie van de vakgroep. De achtergronden van het mislukken van de poging om je tot voorzitter van de strategiecommissie te maken ken ik alleen gedeeltelijk; ik zal ze maar van "politieke" aard noemen. Ik heb er wel het gevoel aan over gehouden dat je door de hele gang van zaken behoorlijk gebruuskeerd bent. Ik vind het nog steeds jammer dat wij met zijn allen niet genoeg civiele courage opgebracht hebben om er tenminste voor te zorgen dat de zaak helder en voor alle betrokkenen aanvaardbaar, en daarmee ook verwerkbaar, wordt.

Wat ik door al deze omstandigheden wel geleerd heb is om je als mens te waarderen; als iemand met een echt "wetenschappelijke" instelling die oprechtheid en het zoeken naar de "waarheid" boven persoonlijke belangen stelt. Ik hoop dan ook dat ik je ondanks je officiële afscheid van de vakgroep nog vaak van mens tot mens mag ontmoeten.

# Rots in de Eindhovense informatica branding.

Kees van Hee

Frans, jij bent een van de succesvolle pioniers van de informatica in Nederland. Je hebt de machines van Electrologica nog van zeer nabij meegemaakt en je hebt de ontwikkeling van electronisch rekenen tot het methodisch programma-ontwerp mede richting gegeven. In zekere zin heb je geluk gehad dat je voor deze spannende ontdekkingsreis hebt kunnen aanmonsteren. Dank zij je voortreffelijke wetenschappelijke bijdragen ben je gedurende de reis aan boord gebleven. De ontdekkingsreis voerde langs operating systems, imperatieve programmeertalen, formele semantiek en declaratieve talen. In deze opsomming hoort ook zeker de opkomst van de database theorie en de daarop gebaseerde database managementsystemen, maar ik heb de indruk dat deze ontdekkingen jou minder hebben geboeid. Iedere keer ging er weer een nieuwe wereld open. Het is de vraag of de dichtheid aan ontdekkingen in de toekomst ook zo groot zal zijn. Daarover zal ik aan het slot nog een bespiegeling wagen.

In Eindhoven is van het begin van de jaren 60 tot medio (de) jaren 80 een monocultuur gegroeid onder leiding van Dijkstra. Deze monocultuur duiden wij vaak aan als de Eindhovense programmeerstijl of de Eindhovense school. In een monocultuur passen geen andere gewassen en deze worden door de kwekers van de monocultuur al snel onkruid genoemd. Velen die iets anders wilden kweken zijn in die tijd gesneuveld maar jij niet. Dit komt deels doordat je zelf een belangrijke bijdrage aan deze Eindhovense school hebt gegeven en ook omdat je heilig gelooft in de Eindhovense methode van programma-afleiding. Maar je hebt je niet beperkt tot dit onderwerp en je hebt je uitvoerig beziggehouden met onderwerpen als denotationele semantiek en programmeertalen als Lisp. Kortom je hebt gewas gekweekt buiten de monocultuur om. Dit is mijn eerste argument om je een 'rots in de Eindhovense branding' te noemen.

In 1984 ging Dijkstra weg en kwam ik binnen; twee volkomen onafhankelijke gebeurtenissen. Ik kende je daarvoor nauwelijks. De informatica-vakgroep ging snel groeien dankzij de bloei van de opleiding voor informatica-ingenieurs. Groei gaat vaak gepaard met stuipjes en we hebben samen veel van deze stuipjes meegemaakt. Met de komst van De Roever, Hammer en mijzelf kwamen er nieuwe stromingen op gang en jij bleef overeind. Dit is de tweede reden om je een rots in de branding te noemen. Vaak deden we een beroep op je als nestor van de Eindhovense informatica. Meestal lukte het je een algemeen aanvaardbare formulering te vinden waarmee we een plan konden voltooien. In de jaren dat ik vakgroepvoorzitter was heb ik je inbreng zeer gewaardeerd.

Hoewel je nu van de bekende "welverdiende rust" gaat genieten, zul je, naar ik vermoed, de informatica niet vaarwel zeggen. Daarom een paar bespiegelingen over de mogelijke

ontwikkelingen van de informatica. Misschien reis je, als gepensioneerd ontdekkingsreiziger, nog een eindje mee. Werden de eerste computers gemaakt om moeilijk rekenwerk te automatiseren, gaande weg kregen zij andere functies. Geen van de mannen van het eerste uur zullen toen vermoed hebben dat computers in onze tijd voornamelijk gebruikt zouden worden voor het componeren en communiceren van teksten en het doen van spelletjes. Wie weet waar zij over 10 à 20 jaar voor gebruikt zullen worden. De hoofdactiviteit van de informaticus is toch het ontwerp van algoritmen. Een algoritme is te beschouwen als een produktieproces met selectie, iteratie, parallellisatie en synchronisatie als coördinatiemechanismen. Algoritme-ontwerp is het afleiden van het produktieproces uit de specificaties van het product. Een algoritme is dus een produktieproces dat één (type) produkt voortbrengt. Informatici ontwerpen ook andere produktieprocessen zoals operating systems, die een oneindige stroom van verschillende typen diensten kunnen verlenen. Informatici zijn dan ook bij uitstek procesontwerpers. Zij letten zowel op de correctheid van processen als op de efficiency ervan.

Een vraag die mij al jaren bezighoudt is: waarom beperken informatici zich tot processen in computers en passen zij hun expertise niet toe op andere terreinen? Een natuurlijke gebiedsuitbreiding voor informatici zijn de processen die door computersystemen ondersteund worden. Requirements engineering richt zich op het in kaart brengen (of in meer wetenschappelijke termen: het modelleren) van de processen die door computersystemen ondersteund moeten worden. Voor embedded software spelen deze processen zich af in instrumenten of machines. Voor informatiesystemen in organisaties zijn het vaak administratieve processen. We zullen de te ondersteunen processen bedrijfsprocessen noemen. Als deze processen niet deugdelijk in elkaar zitten of als ze niet goed begrepen worden, lopen we het risico dat computerondersteuning faalt. Uit een model van de bedrijfsprocessen moeten de eisen voor zo'n computersysteem afgeleid worden. Uit de eisen moeten dan de produktieprocessen in het computersysteem afgeleid worden. Steeds vaker zie je dat de informaticus een rol speelt in het (her-)ontwerp van de bedrijfsprocessen.

Naar ik hoop heb ik je nieuwsgierigheid een beetje geprikkeld en zul je je eens aan een ontdekkingsreis in dit gebied gaan wagen. Ik ben er overigens van overtuigd dat je nog vele reizen op je verlanglijstje hebt. In elk geval wens ik je veel plezier in de komende periode.

# Type theory and the seven steps towards object-based happiness

Kees Hemerik
Dedicated to Frans Kruseman Aretz

*"Simplex sigillum veri"*
Hermannus Boerhaave

*"Make the subject as simple as possible, but not simpler"*
Albert Einstein

## 1 Introduction

The theory of typed lambda calculi – *type theory* for short – provides a unifying framework for programming languages and logics [1, 6]. A major application area of the theory is the analysis and design of programming language concepts, where it has been shown that many notions and pseudo-notions of programming languages can be reduced to a remarkably small number of well-founded type-theoretical concepts (see e.g. [3]).

One area that could greatly benefit from type theory is that of object-oriented programming (OOP) languages. This area is of great importance for large scale software construction. Unfortunately its theory is under-developed, much of the literature is vague and fuzzy, and some of the major OOP languages are overly complex.

One of the better books on OOP is [4] by Bertrand Meyer, which studies object-oriented techniques and their relation to software engineering. As the basis of his study, Meyer uses the "Seven Steps towards Object-Based Happiness", a stepwise introduction to the main characteristics of OOP languages.

In this paper we show how these Seven Steps can be expressed in type theory, thus providing some Type Theory Based Enlightenment.

## 2 Type theory

Our presentation is based on the system $F^{\omega}_{\leq}$ (pronounced "F-omega-sub"), an extension of Girard's system $F^{\omega}$ with subtyping and bounded quantification. For a description

of $F^{\omega}_{\leq}$ we refer to [5]. We extend this calculus with a simple record calculus, which we only introduce by means of some examples. A record with a field `x` having value 3 and a field `y` having value `true` is denoted as `[x to 3, y to true]` and has type `Pi[x to Int, y to Bool]`. Field selection is done by applying the record to a field name, e.g. `[x to 3, y to true].x`, which reduces to 3. Fields may be deleted from records, e.g. `[x to 3, y to true] -- {y}`, which yields `[x to 3]`. Records may also be concatenated, provided their field names are disjoint. For instance, if we have

```
[x to 3, y to true] : Pi[x to Int, y to Bool]
[z to 2] : Pi[z to Int]
```

concatenation yields

```
[x to 3, y to true] ++ [z to 2] : Pi[x to Int, y to Bool] ++ Pi[z to Int]
```

which simplifies to

```
[x to 3, y to true, z to 2] : Pi[x to Int, y to Bool, z to Int]
```

For these records the usual record subtyping rules hold (see e.g. [2, 5]).

## 3 Seven steps towards object-based happiness

Of Meyer's Seven Steps, one – on automatic memory management – is void in type theory, whereas another step – on polymorphism and dynamic binding – is actually two steps. Therefore we allow ourselves a slight rearrangement and take the following Seven Steps as our guideline:

1. Object-based modular structure
2. Data abstraction
3. Subtype polymorphism
4. Classes
5. Inheritance
6. Late binding
7. Multiple inheritance

### Step 1: Object-based modular structure

Suppose that we want to make a program for manipulating two-dimensional points. In traditional programming languages we would define a record type for storing cartesian coordinates, create some points, and provide some functions for manipulating points, e.g.:

```
RPoint_Cart = Pi[x to Real, y to Real] : *
[x to 5, y to 0] : RPoint_Cart

setXY  = lam(s:RPoint_Cart) lam(newX:Real) lam(newY:Real)
           [x to newX, y to newY]
       : RPoint_Cart -> Real -> Real -> RPoint_Cart
getX   = lam(s:RPoint_Cart) s.x
       : RPoint_Cart -> Real
getY   = lam(s:RPoint_Cart) s.y
       : RPoint_Cart -> Real
double = lam(s:RPoint_Cart) [x to mul.2.(s.x), y to mul.2.(s.y)]
       : RPoint_Cart -> RPoint_Cart
```

The first principle of the object-oriented approach is, that program structure should be organized around the data, and that operations (or *methods* in OOP parlance) should be associated with the data on which they operate. This can be achieved by grouping methods in a record and combining this with a state in another record.

```
p0 =
[state   to [x to 5, y to 0]
,methods to [setXY  to lam(s:RPoint_Cart) lam(newX:Real) lam(newY:Real)
                         [x to newX, y to newY]
            ,getX   to lam(s:RPoint_Cart) s.x
            ,getY   to lam(s:RPoint_Cart) s.y
            ,double to lam(s:RPoint_Cart)
                         [x to mul.2.(s.x), y to mul.2.(s.y)]
             ]
]
: Pi[state   to RPoint_Cart
    ,methods to Pi[setXY  to RPoint_Cart -> Real -> Real -> RPoint_Cart
                  ,getX   to RPoint_Cart -> Real
                  ,getY   to RPoint_Cart -> Real
                  ,double to RPoint_Cart -> RPoint_Cart
                  ]
    ]
```

Of course the type of the `methods` field depends on the type of the `state` field. In order to make this dependency explicit, we define the *method interface* `MPoint`, which gives the types of the methods, relative to a representation type `R`.

```
MPoint =
lam(R:*) Pi[setXY  to R -> Real -> Real -> R
           ,getX   to R -> Real
           ,getY   to R -> Real
           ,double to R -> R
           ]
: * -> *
```

which enables us to rewrite the type of `p0` as

```
Pi[state   to RPoint_Cart
  ,methods to MPoint.RPoint_Cart
  ]
```

The `double` method of `p0` can be called as follows

```
p0.methods.double.(p0.state)
```

In the example above the methods `setXY`, `getX`, `getY` and `double` can be defined independently from one another. In general, a method may call other methods. For instance, we could have a different version of the `double` method, which calls the `setXY`, `getX` and `getY` methods. To cope with this kind of cross-referencing, the record of methods should be defined recursively, i.e. as fixed point of the following function

```
cfPoint_Cart =
lam(self:(MPoint.RPoint_Cart))
  [setXY  to lam(s:RPoint_Cart) lam(newX:Real) lam(newY:Real)
               [x to newX, y to newY]
  ,getX   to lam(s:RPoint_Cart) s.x
  ,getY   to lam(s:RPoint_Cart) s.y
  ,double to lam(s:RPoint_Cart)
               self.setXY.s.(mul.2(self.getX.s)).(mul.2.(self.getY.s))
  ]
: (MPoint.RPoint_Cart) -> (MPoint.RPoint_Cart)
```

Using this function and the polymorphic fixed point operator

```
rec: Pi(A:*) (A->A)->A
```

the point p0 above can be redefined more appropriately as

```
p0 =
[state   to [x to 5, y to 0]
,methods to rec.(MPoint.RPoint_Cart).cfPoint_Cart
]
: Pi[state   to RPoint_Cart
    ,methods to MPoint.RPoint_Cart
    ]
```

### Step 2: Data abstraction

The implementation of points given in Step 1 is just one possibility. Instead of cartesian coordinates one could equally well use polar coordinates, for instance. In order to hide the chosen representation and to prevent access to the state other than through the methods, *data abstraction* should be used. In type theory, this is expressed by means of a $\Sigma$-type:

```
Point = Sigma(R:*) Pi[state to R, methods to MPoint.R] : *
```

which expresses that there exists some implementation type R, but prevents access to it. In this definition the only thing specific to points is the point method interface MPoint. Abstraction over this interface yields the type constructor

```
Object = lam(M:*->*) Sigma(R:*) Pi[state to R, methods to M.R]
       : (*->*)->*
```

Using Object, the definition of Point can be rewritten as

```
Point = Object.MPoint : *
```

which clearly states that Point is the type of objects with interface MPoint. Because Point is a $\Sigma$-type, creation of new point objects not only involves grouping state and methods, as we did when creating p0 in Step 1, but also "packing" in order to hide the representation type RPoint_Cart.

```
p1 =
pack<RPoint_Cart
    ,[state   to [x to 5, y to 0]
     ,methods to
        rec.(MPoint.RPoint_Cart).cfPoint_Cart
     ]
    > to Point
```

A consequence of this encapsulation is, that a method call of an object now involves unpacking, applying the method to the hidden state, possibly followed by repacking. For instance, calling the `double` method of `p1` – which we shall abbreviate to `p1'double` – amounts to

```
unpack p as <R,r>
in pack<R
       ,[state   to r.methods.double.(r.state)
        ,methods to r.methods
        ]
       > to Point
```

Note that elements of the same object type (`Point`) may have different representation types for their internal state and different implementations of their methods. We shall return to this subject in Step 4.

### Step 3: Subtype polymorphism

Many OOP languages offer some form of *subtype polymorphism*, in the sense that wherever an object may be used, a more specific object – i.e. one with additional methods – may be used as well. This kind of polymorphism fits nicely with that of $F^{\omega}_{\leq}$. Consider e.g. an object type `ExtPoint`, which in addition to the methods of `Point` offers a method `mirror` for mirroring a point w.r.t. the origin. The method interface for `ExtPoint` is

```
MExtPoint =
lam(R:*) Pi[setXY  to R -> Real -> Real -> R
           ,getX   to R -> Real
           ,getY   to R -> Real
           ,double to R -> R
           ,mirror to R -> R
           ]
: * -> *
```

According to the subtyping rules of $F^{\omega}_{\leq}$ – in particular those for $\Pi$-types and lambda abstractions – we have that `MExtPoint` $\leq$ `MPoint`. Then – by the rules for $\Sigma$-types, $\Pi$-types, and conversion – it follows that `Object.MExtPoint` $\leq$ `Object.MPoint`, just as desired. This is not surprising, because the rules for labelled $\Pi$- and $\Sigma$-types – first formulated in [2] – where designed for this purpose.

### Step 4: Classes

As already mentioned, elements of the same object type may have different representation types for their internal state and different implementations of their methods. In order to

provide more structure, the universe of objects may be divided in *classes* of objects that have the same implementation. Objects of a class can be generated from a blueprint, which is essentially a tuple of methods corresponding to a certain representation type. To understand the mechanism, let us have another look at the way we have generated objects in Step 2.

First we have introduced a representation type `RPoint_Cart` for the internal state of an object

```
RPoint_Cart = Pi[x to Real, y to Real] : *
```

and a function `cfPoint_Cart` for constructing the method tuple of an object

```
cfPoint_Cart =
lam(self:(MPoint.RPoint_Cart))
  [setXY  to lam(s:RPoint_Cart) lam(newX:Real) lam(newY:Real)
               [x to newX, y to newY]
  ,getX   to lam(s:RPoint_Cart) s.x
  ,getY   to lam(s:RPoint_Cart) s.y
  ,double to lam(s:RPoint_Cart)
               self.setXY.s.(mul.2.(self.getX.s)).(mul.2.(self.getY.s))
  ]
: (MPoint.RPoint_Cart) -> (MPoint.RPoint_Cart)
```

The method tuple is taken to be the least fixed point of this function, which we call `clPoint_Cart` for future reference

```
clPoint_Cart =
rec.(MPoint.RPoint_Cart).cfPoint_Cart : (MPoint.RPoint_Cart)
```

Finally the point object `p1` is formed by tupling the method tuple `clPoint_Cart` with an initial state, followed by packing to hide the internal representation

```
p1 =
pack<RPoint_Cart
    ,[state   to [x to 5, y to 0]
     ,methods to clPoint_Cart
     ]
    > to Point
```

Now, if we want to create point objects with a different implementation – e.g. based on polar coordinates – we can follow exactly the same pattern. First we define a representation type for the internal state of an object

```
RPoint_Polar = Pi[rho to Real, phi to Real] : *
```

and a function cfPoint_Polar for constructing the method tuple of an object (we ignore partiality and overflow problems)

```
cfPoint_Polar =
lam(self:(MPoint.RPoint_Polar))
  [setXY  to lam(s:RPoint_Polar) lam(newX:Real) lam(newY:Real)
             [rho to sqrt.(plus.(sqr.newX).(sqr.newY))
             ,phi to arctg.(div.newY.newX)
             ]
  ,getX   to lam(s:RPoint_Polar)  mul.(s.rho).(cos.(s.phi))
  ,getY   to lam(s:RPoint_Polar)  mul.(s.rho).(sin.(s.phi))
  ,double to lam(s:RPoint_Polar)  [rho to mul.2.(s.rho), phi to s.phi]
  ]
: (MPoint.RPoint_Polar) -> (MPoint.RPoint_Polar)
```

Then we construct the method tuple as least fixed point of this function

```
clPoint_Polar =
rec.(MPoint.RPoint_Polar).cfPoint_Polar : (MPoint.RPoint_Polar)
```

and finally we form a point p2 by tupling clPoint_Polar with an initial state and packing

```
p2 =
pack<RPoint_Cart
    ,[state   to [rho to 5, phi to 0]
     ,methods to clPoint_Polar
     ]
    > to Point
```

Obviously, the creation process of p1 has much in common with that of p2. The only differences are in the representation type (RPoint_Cart vs. RPoint_Polar) and in the method implementations (cfPoint_Cart vs. cfPoint_Polar). Therefore, let us try to factor out the common part of the creations, so that we may supply the rest as parameters. To this end, we first define two simple type operators

```
Class  = lam(M:*->*) lam(R:*) M.R : (*->*)->*->*
ClassF = lam(M:*->*) lam(R:*) M.R -> M.R : (*->*)->*->*
```

which enable us to rewrite the typings as

```
cfPoint_Cart : ClassF.MPoint.RPoint_Cart
clPoint_Part : Class.MPoint.RPoint_Cart
```

and

```
cfPoint_Polar : ClassF.MPoint.RPoint_Polar
clPoint_Polar : Class.MPoint.RPoint_Polar
```

Next, we define a function `new`, which takes care of the rest of the creation process

```
new =
lam(M:*->*) lam(R:*) lam(cl:(Class.M.R)) lam(s:R)
  pack<R
      ,[state   to s
       ,methods to cl
       ]
      >
  to (Object.M)
: Pi(M:*->*) Pi(R:*) (Class.M.R) -> R -> (Object.M)
```

Now we can write the construction of `p1` and `p2` as

```
p1 = new.MPoint.RPoint_Cart.clPoint_Cart.[x to 5, y to 0]
: Object MPoint
p2 = new.MPoint.RPoint_Polar.clPoint_Polar.[rho to 5, phi to 0]
: Object MPoint
```

which neatly expresses that objects are created from a method interface, a representation type, method implementations corresponding to that representation type, and an initial value of the representation type. In fact, the entities `MPoint`, `RPoint_Cart`, and `clPoint_Cart`, together with the initial value `[x to 5, y to 0]`, completely determine the class of point objects represented in cartesian coordinates. This enables us to talk about the *class* `Point_Rect`. It is not difficult to design a syntax for class definitions – resembling those of existing object-oriented programming languages – from which definitions of the afore-mentioned entities can automatically be extracted. For lack of space we refrain from doing so. See e.g. [5] for some examples.

### Step 5: Inheritance

*Inheritance* is the phenomenon of forming new classes by extension or modification of existing classes. The subclass inherits the features of the superclass and modifies or extends them. The inheritance relation naturally leads to a hierarchy of classes.

Inheritance can apply to both internal state and methods. For ease of explanation we shall first discuss inheritance of methods, and subsequently inheritance of internal state.

Suppose, for the sake of argument, that we want to create a new class `ExtPoint_Cart` of extended cartesian points, which differs from the class `Point_Cart` in the following two aspects:

- The `getX` method returns 3 plus the `x`-coordinate of a point.
- It has a new method `mirror`, which mirrors the point w.r.t. the origin.

One way to achieve this could be to define the method function `cfExtPoint_Cart` from scratch, in the same vein as the definitions of the method functions `cfPoint_Cart` and `cfPoint_Polar` in Step 4. The key idea of inheritance is to avoid this definition from scratch by suitably modifying and extending the entities of the class `Point_Cart`, as follows:

```
MExtPoint =
lam(R:*) (MPoint R) ++ Pi[mirror to R -> R]

cfExtPoint_Cart =
lam(self:(MExtPoint.RPoint_Cart))
  clPoint_Cart
  -- {getX}
  ++ [getX to lam(s:RPoint_Cart) plus.3.(s.x)]
  ++ [mirror to lam(s:RPoint_Cart)
                  self.setXY.s.(neg.(self.getX.s)).(neg.(self.getY.s)) ]
: (MExtPoint.RPoint_Cart) -> (MExtPoint.RPoint_Cart)

clExtPoint_Cart =
rec.(MExtPoint.RPoint_Cart).cfExtPoint_Cart
: (MExtPoint.RPoint_Cart)
```

The crux of these definitions is in the body of the function `cfExtPoint_Cart`. The following should be noted:

- The method tuple `clPoint_Cart` is used, which does not depend on the parameter `self`, hence is taken over unchanged from the class `Point_Cart`. In particular, the `double` method – which was defined in terms of the `getX` method – is not affected by the redefinition of the `getX` method.
- The method `getX` is overwritten.

- The method `mirror` is added, and makes use of the methods `setXY` and `getY` inherited from the class `Point_Cart` and the redefined method `getX`.

Let us now consider inheritance of internal state. Suppose we want to extend the internal state with a field `c` of type `Color`. This can be achieved by

```
RColPoint = RPoint_Cart ++ Pi[c to Color]
```

Our intention is that the class `ColPoint` inherits all the methods of class `Point_Cart`. But this poses a problem, as the methods of class `Point_Cart` all operate on a state of type `RPoint_Cart`, whereas in class `ColPoint` they should work on a state of type `RColPoint`. In order to inherit from a class, its methods should be defined in such a way, that they work on more refined states as well. In $F^{\omega}_{\leq}$ this can be achieved by using *bounded parametric polymorphism*: each method is provided with an additional type parameter with an upper bound indicating the required fields. Any more refined type is acceptable. In order to use this technique to inherit state from class `Point_Cart`, we should redefine the function `cfPoint_Cart` of Step 4 as

```
cfPoint_Cart =
lam(self:(MPoint.RPoint_Cart))
  [setXY  to lam(R<=RPoint_Cart) lam(s:R) lam(newX:Real) lam(newY:Real)
              s -- {x,y} ++ [x to newX, y to newY]
  ,getX   to lam(R<=RPoint_Cart) lam(s:R) s.x
  ,getY   to lam(R<=RPoint_Cart) lam(s:R) s.y
  ,double to
     lam(R<=RPoint_Cart) lam(s:R)
       self.setXY.R.s.(mul.2.(self.getX.R.s)).(mul.2.(self.getY.R.s))
  ]
: (MPoint.RPoint_Cart) -> (MPoint.RPoint_Cart)
```

The definition of a method call – given by example for the call `p'double` in Step 2 – should be revised accordingly as

```
unpack p as <R,r>
in pack<R
       ,[state   to r.methods.double.R.(r.state)
        ,methods to r.methods
        ]
       > to Point
```

### Step 6: Late binding

In Step 5 we pointed out that the `double` method – which was defined in terms of the `getX` method – was not influenced by redefinition of the `getX` method. It is also possible, however, to arrange matters in such a way that the `double` method *does* make use of the redefined `getX` method. This phenomenon is known as *late binding* and is often used in object-oriented programming. In the framework developed here it can be achieved by means of a different definition of the function `cfExtPoint_Cart`, as follows.

```
cfExtPoint_Cart =
lam(self:(MExtPoint.RPoint_Cart))
  (cfPoint_Cart.self)
  -- {getX}
  ++ [getX to lam(R<=RPoint_Cart) lam(s:R) plus.3.(s.x)]
  ++ [mirror to lam(R<=RPoint_Cart) lam(s:R)
                  self.setXY.R.s.(neg.(self.getX.R.s)).(neg(self.getY.R.s)) ]
: (MExtPoint.RPoint_Cart) -> (MExtPoint.RPoint_Cart)

clExtPoint_Cart =
rec.(MExtPoint.RPoint_Cart).cfExtPoint_Cart
: (MExtPoint.RPoint_Cart)
```

When the function `cfExtPoint_Cart` is defined in this way, the redefinition of `getX` *does* affect the `double` method. This can easily be seen by unfolding the definition of `cfPoint_Cart` in the body of `cfExtPoint_Cart`, which yields

```
lam(self:(MExtPoint.RPoint_Cart))
  [setXY  to lam(R<=RPoint_Cart) lam(s:R) lam(newX:Real) lam(newY:Real)
               s -- {x,y} ++ [x to newX, y to newY]
  ,getX   to lam(R<=RPoint_Cart) lam(s:R) s.x
  ,getY   to lam(R<=RPoint_Cart) lam(s:R) s.y
  ,double to lam(R<=RPoint_Cart) lam(s:R)
               self.setXY.R.s(mul.2.(self.getX.R.s)).(mul.2.(self.getY.R.s))
  ]
  -- {getX}
  ++ [getX to lam(R<=RPoint_Cart) lam(s:R) plus.3.(s.x)]
  ++ [mirror to lam(R<=RPoint_Cart) lam(s:R)
                  self.setXY.R.s.(neg.(self.getX.R.s)).(neg(self.getY.R.s)) ]
: (MExtPoint.RPoint_Cart) -> (MExtPoint.RPoint_Cart)
```

It is surprising to see how simply the difference between early and late binding can be expressed by modifying the body of the function `cfPoint_Cart`.

### Step 7: Multiple inheritance

In many cases it is necessary or convenient to inherit from more than one superclass. This kind of inheritance is called *multiple inheritance*. Multiple inheritance may pose some problems when the field names of the super classes are not disjoint. Let us first consider the case where this problem does not occur.

We assume a – very uninteresting – class `Col` of colors, with methods for setting and getting a color. Our goal is to combine the features of the class `Col` and the class `Point_Rect` to obtain a class `ColPoint_Rect` of colored cartesian points. The class `Col` is characterized by `MCol`, `RCol`, `cfCol`, and `clCol`, defined as follows:

```
MCol =
lam(R:*) [setC to Pi(R1<=R) R1 -> Color -> R1
         ,getC to Pi(R1<=R) R1 -> Color]
: * -> *

RCol = Pi[c to Color]
: *

cfCol =
lam(self:(MCol.RCol))
  [setC to lam(R<=RCol) lam(s:R) lam(newC:Color) [c to newC]
  ,getC to lam(R<=RCol) lam(s:R) s.c
  ]
: (MCol.RCol) -> (MCol.RCol)

clCol = rec.(MCol.RCol).cfCol
: (MCol.RCol)
```

In order to form the class `ColPoint_Rect` by means of multiple inheritance, we simply define its entities `MColPoint`, `RColPoint`, and `cfColPoint` by concatenating the corresponding entities from the classes `Col` and `Point_Rect`, as follows:

```
MColPoint =
lam(R:*) (MPoint.R) ++ (MCol.R)
: * -> *

RColPoint = RPoint_Cart ++ RCol
: *

cfColPoint =
lam(self:(MColPoint.RColPoint)
```

```
  (cfPoint_Cart.self) ++ (cfCol.self)
: (MColPoint.RColPoint) -> (MColPoint.RColPoint)
```

Thereafter, we define `clColPoint` to be the least fixed point of `cfColPoint` in the usual way.

```
clColPoint = rec.(MColPoint.RColPoint).cfColPoint
: (MColPoint.RColPoint)
```

All this is very straightforward. Problems may arise when we want to concatenate records whose fields are not disjoint. Consider for instance

```
A = [..., f to F1, g to G1, ...]
B = [..., f to F2, g to G2, ...]
```

In order to form the concatenation `A ++ B`, we have to explicitly indicate which `f` and `g` fields should be left out to avoid name clashes. For instance, if we want to leave out the `f` field of `A` and the `g` field of `B`, we can write

```
( A -- {f}) ++ ( B -- {g})
```

**Acknowledgement**

This presentation was inspired by [5] and [7]. Thanks are due to Jan Zwanenburg and Fairouz Kamareddine for reading and commenting on earlier versions of this paper.

## References

[1] Henk Barendregt. *Lambda calculi with types.* In: *Handbook of Logic in Computer Science*, Volume 2. Edited by S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum. Oxford Science Publications, Oxford, 1992.

[2] Luca Cardelli. *A Semantics of Multiple Inheritance.* Information and Computation **76**, 138–164, 1988.

[3] Luca Cardelli and Peter Wegner. *On Understanding Types, Data Abstraction, and Polymorphism.* Computing Surveys **17** (4), 1985.

[4] Bertrand Meyer. *Object-oriented Software Construction.* Prentice-Hall, 1988.

[5] Benjamin C. Pierce, David N. Turner. *Simple Type-Theoretic Foundations For Object-Oriented Programming.* J. Functional Programming **1**(1), Jan. 1993.

[6] Erik Poll. *A Programming Logic Based on Type Theory.* Ph.D. Thesis. Eindhoven University of Technology, 1994.

[7] Jan Zwanenburg. *Record concatenation with intersection types.* To appear.

# Expression evaluation revisited

Rob R. Hoogerwoord

## 0 Introduction

Expressions are linear strings of symbols representing tree structures. Evaluators, which are mechanisms for computing the "values" of such trees, are usually designed to take expressions for their inputs, instead of trees. This is a matter of efficiency: executing a linear sequence of instructions may take less time than performing a tree walk, and this linear sequence may take less space than the tree it represents.

From a definition of trees as a starting point, an evaluator can be designed by first *choosing* a linear representation and next deriving the evaluator. This choice is not trivial, though, because trees can be represented linearly in many different ways, such as by prefix, infix, or postfix codes: at the outset, it is not at all clear which choice will lead to an efficient design.

In this paper we illustrate a design approach in which the linear representation just *emerges* as a by-product of the design process. This example also suggests that expression parsing could well do without the grammars-and-languages paradigm. The example is about simple expressions, which either are *primitive* terms –like constants, variables, or function applications–, or expressions *composed* from other ones by means of binary operators.

**On notation:** With $\odot$ for any binary operator, $(x\odot)$, $(\odot y)$, and $(\odot)$ denote the functions $\lambda y: x\odot y$, $\lambda x: x\odot y$, and $\lambda x, y: x\odot y$ respectively.
For any type $B$ we use $\mathcal{L}_*(B)$ for the type of finite lists with elements in $B$. We use $[]$ ("empty") for the empty list and both $\triangleright$ ("cons") and $\triangleleft$ ("snoc") as list constructors: for element $b$ and list $x$ both $b\triangleright x$ and $x\triangleleft b$ are (usually different) lists. (For example: $[b,c,d] = b\triangleright[c,d]$ and $[b,c,d] = [b,c]\triangleleft d$ .) The reason to use both $\triangleright$ and $\triangleleft$ is mainly aesthetical; for example, in an equation like $F\cdot x\cdot(\langle b\rangle\triangleright ss) = F\cdot(x\triangleleft b)\cdot ss$ the variables can now occur in the same order in both sides of the equation.
The symbol $*$ ("map") denotes the (binary) map-operator: for $f$ of type $B\rightarrow C$ the function $(f*)$ has type $\mathcal{L}_*(B)\rightarrow\mathcal{L}_*(C)$, with:

$$\begin{array}{lcl} f*[] & = & [] \\ f*(b\triangleright x) & = & f\cdot b\triangleright f*x. \end{array}$$

□

## 1 Continued function application

Here we reinvent the concept of *folded operators*, as discussed more extensively in R.S. Bird and Ph. Wadler's textbook "Introduction to functional programming" (Prentice-Hall, 1988). We use this to obtain (so-called) continued function application, to be used in Section 3.

We consider a binary operator $\oplus$ of type $B \times U \rightarrow U$, for some types $B$ and $U$. In this section dummies $b$ and $c$ have type $B$ whereas $u$ has type $U$. With $\oplus$ we can form expressions of type $U$, like:

$$u \ , \ b \oplus u \ , \text{and} \ b \oplus (c \oplus u) \ .$$

These expressions have in common that they depend on *one* value of type $U$ and *some* –zero or more– values of type $B$. The order of the $B$ values is relevant, so we can also consider them as the elements of a *list* of type $\mathcal{L}_*(B)$. We now rewrite the above expressions in terms of a single $U$ and a single $\mathcal{L}_*(B)$, by introducing a binary operator $\odot$, of type $\mathcal{L}_*(B) \times U \rightarrow U$, thus:

$$\begin{array}{rcl} u & = & [\,] \odot u \\ b \oplus u & = & [b] \odot u \\ b \oplus (c \oplus u) & = & [b, c] \odot u \ . \end{array}$$

Moreover, $b \oplus (c \oplus u) = (b \oplus u)(u := c \oplus u)$, so for consistency's sake our new operator must also satisfy:

$$[b, c] \odot u \ = \ [b] \odot (c \oplus u) \ .$$

By a simple generalization –just replace $[b]$ by a list $x$– we obtain the following recursive definition for $\odot$:

$$\begin{array}{rcl} [\,] \odot u & = & u \\ (x \triangleleft c) \odot u & = & x \odot (c \oplus u) \ . \end{array}$$

Operator $\odot$ has other interesting properties, but we shall not need these; $\odot$ is well-known: it is called *foldr*$(\oplus)$ in Bird and Wadler's text, but in calculations I prefer a (short) name like $\odot$ over a (long) expression like *foldr*$(\oplus)$. The above shows that such *folded operators* can be invented without operational interpretations, just by isolating the common pattern from a few similar expressions.

* * *

Function application is a binary operator and we can apply the above to it: with $B := (U \to U)$ function application indeed has type $B \times U \to U$. So, we now use $\odot$ for folded $\cdot$, and we obtain as definition:

$$\begin{array}{rcl} [\,] \odot u & = & u \\ (x \triangleleft g) \odot u & = & x \odot (g \cdot u) \quad , \end{array}$$

with properties like:

$$[f, g] \odot u \;=\; f \cdot (g \cdot u) \quad .$$

Hence, a list $x$ of type $\mathcal{L}_*(U \to U)$ now represents the continued composition of its (function) elements and $\odot$ amounts to continued function application: $x \odot u$ is the application to $u$ of the functions in $x$. In Section 3 we shall use this to transform linear recursion into tail recursion. (Besides, this example shows (once more) that it really helps to have an explicit symbol for function application.)

## 2 Trees and their values

In what follows, dummy $b$ has type $B$ and dummy $c$ has type $C$, for some *disjoint* types $B$ and $C$, so we have $(\forall b, c :: b \neq c)$. The datatype $T$ of trees is defined (recursively) as follows:

$$\begin{array}{lcll} \langle b \rangle & \in & T & , \text{ for all } b \\ \langle c, s, t \rangle & \in & T & , \text{ for all } c \text{ and } s, t \in T \quad . \end{array}$$

Next, we assume that, for some fixed type $M$, functions $f$ and $g$ have been given, with the following types:

$$\begin{array}{lcl} f & \in & B \to M \\ g & \in & C \to M \to M \to M \quad . \end{array}$$

Type $M$ is the type of the values of trees, as is reflected by the following definition of function $V \in T \to M$, with the intention that $V \cdot s$ be the value of tree $s$:

$$\begin{array}{lcl} V \cdot \langle b \rangle & = & f \cdot b \\ V \cdot \langle c, s, t \rangle & = & g \cdot c \cdot (V \cdot s) \cdot (V \cdot t) \quad . \end{array}$$

## 3 An evaluator

Assuming that we know how to implement $f$ and $g$, we are interested in implementations of $V$ with such a fine grain of detail that the result can be considered as code for a Von Neumann type of machine. In particular we wish to eliminate the recursion from $V$'s definition, although we may equally well say that we wish to make the implementation of this recursion explicit.

A standard technique to eliminate recursion is to transform it into tail-recursion, which can be implemented by simple iteration. As a first step, we must reduce the *two* recursive occurrences of $V$ in its definition to a *single* one, thus making the definition linearly recursive. The main design decision now is that we study $V*ss$, for $ss$ of type $\mathcal{L}_*(T)$, for two reasons. First, it is a generalization, because $[V{\cdot}s] = V*[s]$, and, second, $V{\cdot}s$ and $V{\cdot}t$ are the elements of the list $V*[s,t]$; thus, we try to combine the two recursive applications of $V$ into a single one.

As always we have $V*[\,] = [\,]$; furthermore, in compliance with the case analysis in $V$'s definition, we derive:

$$
\begin{array}{ll}
 & V*(\langle b\rangle \triangleright ss) \\
= & \quad \{\ *\ \} \\
 & V{\cdot}\langle b\rangle \triangleright V*ss \\
= & \quad \{\ V\ \} \\
 & f{\cdot}b \triangleright V*ss \quad ,
\end{array}
$$

and:

$$
\begin{array}{ll}
 & V*(\langle c,s,t\rangle \triangleright ss) \\
= & \quad \{\ *\ \} \\
 & V{\cdot}\langle c,s,t\rangle \triangleright V*ss \\
= & \quad \{\ V\ \} \\
 & g{\cdot}c{\cdot}(V{\cdot}s){\cdot}(V{\cdot}t) \triangleright V*ss \\
= & \quad \{\ \text{eureka! introduction of } \otimes \text{ (see below)}\ \} \\
 & g{\cdot}c \otimes (V{\cdot}s \triangleright V{\cdot}t \triangleright V*ss) \\
= & \quad \{\ *\ \text{(twice)}\ \} \\
 & g{\cdot}c \otimes (V*(s \triangleright t \triangleright ss)) \quad .
\end{array}
$$

Here we have introduced an operator $\otimes$ defined by:

$$h \otimes (m \triangleright n \triangleright ms) \;=\; h{\cdot}m{\cdot}n \triangleright ms \quad ,$$

for all $h \in M \to M \to M$, $m, n \in M$, and $ms \in \mathcal{L}_*(M)$. This is not at all far-fetched: the only way to stay within the pattern –expressions of the shape $V*(\cdots)$ – is to collect $V{\cdot}s$, $V{\cdot}t$, and $V{*}ss$ into $V*(s \triangleright t \triangleright ss)$. Thus we obtain as a linearly recursive definition for function $(V*)$ :

$$\begin{array}{lcl} V*[] & = & [] \\ V*(\langle b\rangle \triangleright ss) & = & f{\cdot}b \triangleright V{*}ss \\ V*(\langle s,c,t\rangle \triangleright ss) & = & g{\cdot}c \otimes (V*(s \triangleright t \triangleright ss)) \ . \end{array}$$

Next we transform the linear recursion into tail recursion. We may rewrite expression $f{\cdot}b \triangleright V{*}ss$ as $(f{\cdot}b\triangleright){\cdot}(V{*}ss)$, thus making explicit that it is an application of a function to $V{*}ss$. Also, we can view $g{\cdot}c \otimes (V*(s \triangleright t \triangleright ss))$ as an application of $(g{\cdot}c\otimes)$ to $V*(s \triangleright t \triangleright ss)$. Finally, $V{*}ss$ itself is an application of the identity function to $V{*}ss$.

As a generalization, we now consider formulae of the shape $x \odot (V{*}ss)$, with $x$ a list of functions and with $\odot$ for continued function application (as defined in Section 1). Every function in $x$ either is a $(f{\cdot}b\triangleright)$ or is a $(g{\cdot}c\otimes)$, both of type $\mathcal{L}_*(M) \to \mathcal{L}_*(M)$. Function $(f{\cdot}b\triangleright)$ only depends upon $b$, whereas $(g{\cdot}c\otimes)$ only depends upon $c$; hence, we can use the $b$ and $c$ values to represent the corresponding functions. Because the types of $b$ and $c$ are disjoint this representation is unambiguous. Therefore, we redefine $\odot$ as follows, now with $x$ of type $\mathcal{L}_*(B \cup C)$:

$$\begin{array}{lcl} [] \odot ms & = & ms \\ (x \triangleleft b) \odot ms & = & x \odot (f{\cdot}b \triangleright ms) \\ (x \triangleleft c) \odot ms & = & x \odot (g{\cdot}c \otimes ms) \ . \end{array}$$

Now we have all that is necessary to deal with the following generalization of $(V*)$; we introduce function $F$, of type $\mathcal{L}_*(B \cup C) \to \mathcal{L}_*(T) \to \mathcal{L}_*(M)$, with specification:

$$F{\cdot}x{\cdot}ss \ = \ x \odot V{*}ss \ .$$

Then $V$ can be defined in terms of $F$:

$$[V{\cdot}s] \ = \ F{\cdot}[]{\cdot}[s] \ ,$$

and, using the definitions for $(V*)$ and $\odot$, we obtain as definition for $F$:

$$\begin{array}{lcl} F{\cdot}x{\cdot}[] & = & x \odot [] \\ F{\cdot}x{\cdot}(\langle b\rangle \triangleright ss) & = & F{\cdot}(x \triangleleft b){\cdot}ss \\ F{\cdot}x{\cdot}(\langle c,s,t\rangle \triangleright ss) & = & F{\cdot}(x \triangleleft c){\cdot}(s \triangleright t \triangleright ss) \ . \end{array}$$

* * *

The *single* occurrence of $\odot$ in this definition can be factored out by a standard transformation; actually, this is the inverse of what nowadays is called *fusion*: we might call it *fission*. That is, $F$ can be viewed as the composition of $(\odot[])$ and a function $G$:

$$F{\cdot}x{\cdot}ss \;=\; G{\cdot}x{\cdot}ss \odot [] \;,$$

where a definition for $G$ is obtained from $F$'s definition by simply omitting $\odot\,[]$ .

Summarizing, we obtain the following set of definitions, in which we also have eliminated $\otimes$ by combining its definition with the definition of $\odot$:

$$\begin{array}{lcl}
[V{\cdot}s] & = & G{\cdot}[]{\cdot}[s] \odot [] \\
 & & \\
G{\cdot}x{\cdot}[] & = & x \\
G{\cdot}x{\cdot}(\langle b\rangle \triangleright ss) & = & G{\cdot}(x \triangleleft b){\cdot}ss \\
G{\cdot}x{\cdot}(\langle c,s,t\rangle \triangleright ss) & = & G{\cdot}(x \triangleleft c){\cdot}(s \triangleright t \triangleright ss) \\
 & & \\
[] \odot ms & = & ms \\
(x \triangleleft b) \odot ms & = & x \odot (f{\cdot}b \triangleright ms) \\
(x \triangleleft c) \odot (m \triangleright n \triangleright ms) & = & x \odot (g{\cdot}c{\cdot}m{\cdot}n \triangleright ms)
\end{array}$$

These definitions admit a nice operational interpretation. The value of $[V{\cdot}s]$ can be computed in two phases: first, $G{\cdot}[]{\cdot}[s]$ is computed, which yields some list $x$ (of type $\mathcal{L}_*(B \cup C)$), and, second, $x \odot []$ is evaluated. The first phase, and the resulting list $x$, are independent of $f$ and $g$: it can be considered as a purely *syntactic* transformation, that is, $x$ is just a linear representation of tree $s$. Actually, $x$ is the postfix-code representation of $s$ when, conforming with the snoc operations, we "read $x$ from right to left". (For example, $G{\cdot}[]{\cdot}[\langle c_0, \langle c_1, b_0, b_1\rangle, \langle b_2\rangle\rangle] \;=\; [c_0, c_1, b_0, b_1, b_2]$ .)

The first phase can be performed "at compile time" and is known as "code generation". The second phase then becomes "program execution" where the list computed by the first phase is the program to be executed. A value $b$ in this list can be viewed as an instruction to "push value $f{\cdot}b$ onto the stack" whereas a value $c$ can be viewed as an instruction to "apply operation $g{\cdot}c$ to the top two elements of the stack and replace them by the result".

* * *

The above is independent of the actual choice of $M$, $f$, and $g$. With $M = \mathsf{Int}$ and an appropriate choice for $f$ and $g$ we obtain an evaluator for integer expressions. But we may also set:

$$\begin{array}{lcl} M & = & T \\ f{\cdot}b & = & \langle b \rangle \\ g{\cdot}c{\cdot}s{\cdot}t & = & \langle c, s, t \rangle \ . \end{array}$$

Now $V$ becomes the identity function on $T$ and we obtain:

$$[s] \ = \ G{\cdot}[\,]{\cdot}[s] \odot [\,] \ ,$$

which means that $(\odot[\,])$ reconstructs $[s]$ from $G{\cdot}[\,]{\cdot}[s]$: now $(\odot[\,])$ is a *(bottom-up) parser* for postfix-code expressions. For this case we obtain:

$$\begin{array}{lcl} [\,] \odot ss & = & ss \\ (x \lhd b) \odot ss & = & x \odot (\langle b \rangle \rhd ss) \\ (x \lhd c) \odot (s \rhd t \rhd ss) & = & x \odot (\langle c, s, t \rangle \rhd ss) \ . \end{array}$$

Generally, $ss = G{\cdot}[\,]{\cdot}ss \odot [\,]$, so $(\odot[\,])$ is the inverse of $G{\cdot}[\,]$: parsing is the inverse operation of representing a tree by a list.

**acknowledgement:** To the Eindhoven Tuesday Afternoon Club, for their reading of and comments on an earlier version.

□

# Het vermoeden van Kruseman Aretz

Piet van der Houwen

Het vermoeden van Kruseman Aretz doet een uitspraak over het rationaal zijn van de coëfficienten van zekere polynomen die een rol spelen in de numerieke integratie van differentiaalvergelijkingen. Frans deed deze uitspraak in het najaar van 1968 naar aanleiding van een werkbespreking van de Rekenafdeling van het Mathematisch Centrum. Op dit moment is de situatie dat 'driekwart' van zijn vermoeden bevestigd is.

Frans, in mijn bijdrage aan dit Vriendenboek wil ik je vertellen hoe het nu, 27 jaar later, met dat vermoeden staat. Maar eerst nog even kort wat er aan die werkbespreking vooraf ging.

Op een namiddag in het voorjaar van 1968 kwam Van Wijngaarden mijn kamer bij het MC binnenvallen met de mededeling dat hij me even wilde spreken. Hij vertelde me dat Dekker binnenkort voor een jaar naar de Verenigde Staten zou vertrekken en dat hij een vervanger zocht. Het leek hem een goed idee dat ik die vervanger zou zijn. Het voornaamste was het draaiende houden van de 'winkel' waar rekenopdrachten voor derden uitgevoerd werden. Als ik er meer van wilde weten, moest ik maar naar Kruseman Aretz gaan, die zou me wel opvangen. Op dat moment waren Dekker en Kruseman Aretz niet veel meer dan namen voor me, waarvan ik eigenlijk alleen maar wist dat ze sous-chefs van de Rekenafdeling waren en waar ik een beetje tegen opzag, zoals je dat in die tijd deed bij mensen die hoger op de maatschappelijke ladder stonden.

De volgende dag bleek dat Dirk Dekker nog maar twee dagen in het land was, zodat van inwerken niet veel meer kwam. De eerste weken in de winkel waren dan ook vol verrassingen. Maar Frans, Van Wijngaarden had gelijk, wanneer ik problemen had met de winkel, dan kon ik altijd bij je terecht (alhoewel ik het vermoeden heb dat Van Wijngaarden je nooit over dat opvangen gesproken heeft). Het werd een gewoonte om samen te lunchen waarbij het wel en wee van de Rekenafdeling, en in het bijzonder het wel en wee van de winkel, besproken werd. Aan die lunches heb ik goede herinneringen overgehouden. Ik zie ze ook als een begin van onze latere dinertjes met z'n vieren over en weer, eerst Abcoude-Amsterdam, daarna Eersel-Bussum. Ik weet ook nog heel goed hoe gestreeld ik mij voelde toen je Aly en mij voor het eerst uitnodigde om te komen eten bij Loes en jou thuis in Abcoude.

Terug naar het MC. Na verloop van tijd kwamen tijdens onze 'zakenlunches' ook andere en leukere problemen aan bod dan die van de winkel. Eén van die problemen betrof een minimaxprobleem dat aan de orde was gesteld in die werkbespreking in het najaar van

1968. Het had te maken met polynomen van de vorm

$$P_m^p(z) := 1 + \frac{1}{1!}z + \frac{1}{2!}z^2 + \frac{1}{3!}z^3 + \ldots + \frac{1}{p!}z^p + \beta_{p+1}z^{p+1} + \ldots + \beta_m z^m.$$

Hierin zijn $p$ en $m$ gegeven gehele getallen ($m > p \geq 1$) en zijn $\beta_{p+1}$, $\beta_{p+2}$, ..., $\beta_m$ nader te bepalen reële getallen. Laat $[-i\beta, +i\beta]$ het zuiver imaginaire interval zijn waarop het polynoom $P_m^p(z)$ in modulus kleiner dan of gelijk aan 1 is. De vraag is nu om de coëfficienten $\beta_{p+1}$, $\beta_{p+2}$, ..., $\beta_m$ zó te kiezen dat $\beta$ zo groot mogelijk is. Voor lage waarden van $m$ zijn met elementaire middelen dergelijke 'optimale' polynomen, aan te geven met $I_m^p$, wel te vinden. Ter illustratie werden in die werkbespreking de volgende voorbeelden gegeven:

$$I_2^1(z) = 1 + z + z^2, \quad \beta = 1; \qquad I_3^1(z) = 1 + z + \tfrac{1}{2}z^2 + \tfrac{1}{4}z^3, \quad \beta = 2;$$
$$I_3^2(z) = 1 + z + \tfrac{1}{2}z^2 + \tfrac{1}{4}z^3, \quad \beta = 2; \quad I_4^2(z) = 1 + z + \tfrac{1}{2}z^2 + \tfrac{1}{6}z^3 + \tfrac{1}{24}z^4, \quad \beta = 2\sqrt{2}.$$

De beschikbaarheid van een expliciete voorstelling van hogere graads hyperbolische polynomen is van belang in de numerieke integratie van (hyperbolische) differentiaalvergelijkingen. In de numerieke wiskunde worden $p$, $\beta$ en $[-i\beta, +i\beta]$ respectievelijk de *orde van nauwkeurigheid*, de *imaginaire stabiliteitsgrens* en het *imaginaire stabiliteitsinterval* genoemd. De polynomen $I_m^p$ zelf worden vaak hyperbolische polynomen genoemd. Hiervan zijn die met $p = 1$ en $p = 2$ voor de rekenpraktijk het belangrijkste.

Ik kan me herinneren dat je het vinden van deze hyperbolische polynomen $I_m^p$ een prachtig wiskundig probleem vond en dat je je aan de volgende veronderstelling waagde, die we toen het *Vermoeden van Kruseman Aretz* hebben genoemd:

**Vermoeden van Kruseman Aretz.** De polynomen $I_m^1$ en $I_m^2$ hebben rationale coëfficienten. □

Ik geloof dat je het zelfs wel aandurfde om te stellen dat de coëfficienten inversen van gehele getallen zijn. Dat gaat op voor de hier boven gegeven voorbeelden, maar zou later blijken onjuist te zijn voor grotere waarden van $m$. Overigens is het *rationaal* veronderstellen van de coëfficienten ook al een riskante uitspraak! Want stel dat we het minimaxprobleem veranderen en de coëfficienten zó kiezen dat $P_m^p$ in modulus $\leq 1$ is op het *reële* interval $[-\beta, 0]$ waarin $\beta$ weer zo groot mogelijk is. Dergelijke 'optimale' polynomen (ook wel parabolische polynomen genoemd omdat ze een rol spelen bij de integratie van parabolische differentiaalvergelijkingen) hebben voor $p = 1$ inderdaad rationale coëfficienten maar niet meer voor $p \geq 2$. Het blijkt namelijk dat de coëfficienten $\beta_j$ aan niet-lineaire vergelijkingen voldoen die in het algemeen geen rationale oplossingen hebben.

Maar hoe zit het nu met die hyperbolische polynomen $I_m^p$? In al die jaren dat we elkaar toch met enige regelmaat zagen, zijn ze nooit meer ter sprake gekomen. Toen ik van de redactie van het Vriendenboek een uitnodiging kreeg om een bijdrage te leveren, was mijn

eerste reactie dat ik nu een kans had om je in één keer 'bij te praten'. In de volgende paragraaf volgt een overzicht van wat er tot dusver over $I_m^p$ bekend is en hoe het met je vermoeden uit 1968 staat. Daarnaast, ik kan het niet nalaten, wil ik ook iets over het *parabolische* minimaxprobleem vertellen (paragraaf 2). Bovendien, als ik het mij goed herinner, heb je naar aanleiding van een rapportje dat Jan Kok en ik over het parabolische minimaxprobleem hadden geschreven, bij een bezoek aan het MC (ik denk in 1971) je laten ontvallen dat je dit probleem een even grote uitdaging vond als het hyperbolische minimaxprobleem.

## 1 De jacht op de hyperbolische polynomen

Zoals gezegd, een expliciete representatie van $I_m^p$ is van belang in de numerieke wiskunde, en een aantal numerici hebben er dan ook 'jacht' op gemaakt (overigens niet alleen voor $p = 1$ en $p = 2$, maar ook voor $p > 2$). In het begin (eind zestiger en begin zeventiger jaren) heb ik daar ijverig aan meegedaan. De eerste stap was het vinden van een bovengrens voor de imaginaire stabiliteitsgrens $\beta$. Voor $p = 1$ kon vrij eenvoudig aangetoond worden dat $\beta \leq m - 1$ als $m$ oneven is en $\beta \leq m$ als $m$ even is (later in 1983 bewees Vichnevetski [15] dat $\beta \leq m - 1$ voor alle $p$ en $m$). Vervolgens lukte het om voor $p = 1$ en $m$ oneven, polynomen te vinden die een imaginaire stabiliteitsgrens $\beta = m - 1$ bezitten. Deze polynomen moesten dus wel de gezochte polynomen $I_m^1$ zijn. Het een en ander werd vastgelegd in een rapport van de Rekenafdeling (MR 105, mei 1969), waarin de volgende representatie gegeven wordt:

$$I_m^1(z) = T_k\left(1 + \frac{z^2}{2k^2}\right) + \frac{z}{k}\left(1 + \frac{z^2}{4k^2}\right) U_{k-1}\left(1 + \frac{z^2}{2k^2}\right), \quad m = 2k+1, \; k \geq 1.$$

Hierin stellen $T_k$ en $U_k$ de Chebyshev polynomen van eerste en tweede soort voor. Nu we de 'oneven helft' van de $I_m^1$ expliciet in handen hebben, kan het vermoeden voor deze helft eenvoudig geverifieerd worden, door op te merken dat Chebyshev polynomen rationale coëfficienten hebben, zodat de oneven $I_m^1$ ook wel rationale coëfficienten moeten hebben (voor het gemak wordt een polynoom 'even' of 'oneven' genoemd als zijn graad even of oneven is). Uit het voorgaande volgt dat een 'kwart' van het vermoeden bevestigd is. Maar wat blijkt verder, de coëfficient van $z^2$ van $I_m^1$ is voor alle oneven $m$ gelijk aan $1/2$, zodat kennelijk $I_m^1 = I_m^2$ voor alle oneven $m$, en derhalve

$$I_m^2(z) = T_k\left(1 + \frac{z^2}{2k^2}\right) + \frac{z}{k}\left(1 + \frac{z^2}{4k^2}\right) U_{k-1}\left(1 + \frac{z^2}{2k^2}\right), \quad m = 2k+1, \; k \geq 1.$$

Hiermee is dus je vermoeden al voor de 'helft' bevestigd (ik denk dat je dit resultaat nog net meegemaakt hebt voor je in 1969 naar Eindhoven vertrok).

Bovenstaande expliciete representaties voor $I_m^1$ hebben de eigenschap dat ze aan een recurrente betrekking voldoen. Dit is belangrijk in praktische toepassingen en houdt verband met de efficientie van de uit deze polynomen af te leiden integratiemethode. Verder is het in de rekenpraktijk niet noodzakelijk om de polynomen $I_m^1$ en $I_m^2$ voor *even* waarden van $m$ beschikbaar te hebben (alleen de grootte van de graad $m$ speelt een rol in de efficientie van het rekenschema en niet het even of oneven zijn van $m$). Dus van praktisch oogpunt uit gezien was de situatie bevredigend. Daarom heb ik de hyperbolische polynomen vaarwel gezegd en heb me in 1970 op de parabolische polynomen gestort (in paragraaf 2 meer daarover). In 1977/78 bleek dat hyperbolische polynomen van bijzondere interesse waren in aerodynamische berekeningen en toen begon het ontbreken van de even $I_m^1$ en $I_m^2$ toch te knagen. Dit was een reden om de constructie van deze polynomen als prijsvraag in het Nieuw Archief voor Wiskunde (Deel XXVI, No. 2, 1978) te plaatsen. Dat had succes. In 1985 toonden Sonneveld en van Leer [14] aan dat bovenstaand polynoom $I_m^1$ ook voorgesteld kan worden in de vorm

$$I_m^1(z) = i^{m-1}T_{m-1}\left(\frac{z}{i(m-1)}\right) + \frac{1}{2}i^m T_{m-1}\left(\frac{z}{i(m-1)}\right) - T_{m-1}\left(\frac{z}{i(m-1)}\right), \quad m \geq 2.$$

Het aardige is nu dat deze laatste representatie direct laat zien dat ook voor *even* waarden van $m$ een polynoom gedefinieerd wordt en dat voor alle $m$ de coëfficienten hiervan reëel zijn. Verder blijkt dat voor even $m$ de coëfficient van $z$ gelijk 1 is en dat voor alle $m$ de imaginaire stabiliteitsgrens $\beta = m - 1$. Uit het al eerder genoemde resultaat van Vichnevetski van 1983 volgt dan direct dat voor alle $m$ deze representatie de hyperbolische polynomen $I_m^1$ definieert. Overigens wisten Sonneveld en van Leer niet dat nog weer een andere representatie voor $I_m^1$ al in 1984 door Kinnmark en Gray [8] gevonden was, namelijk

$$I_m^1(z) = (-i)^m \left[ iT_{m-1}\left(\frac{iz}{m-1}\right) - \left(1 + \frac{z^2}{(m-1)^2}\right) U_{m-2}\left(\frac{iz}{m-1}\right)\right], \quad m \geq 2.$$

Kinnmark en Gray [9] hebben ook nog naar de polynomen $I_m^3$ en $I_m^4$ gezocht en benaderingen geconstrueerd die een imaginaire stabiliteitsgrens hebben welke verbazend dicht bij het optimum $\beta = m - 1$ van Vichnevetski ligt. Zo hebben de oneven Kinnmark-Gray-benaderingen voor $I_m^3$ en de even Kinnmark-Gray-benaderingen voor $I_m^4$ beide een $\beta = \sqrt{(m-1)^2 - 1}$.

Nu weer terug naar het vermoeden van Kruseman Aretz. Uiteraard zijn ook de coëfficienten van de even $I_m^1$ voor alle $m$ rationaal, zodat nu 'driekwart' van het vermoeden is bevestigd. Om uitsluitsel te krijgen over het laatste kwart van het vermoeden moeten we de even $I_m^2$ vinden. Tot dusver zijn deze nog niet gevonden en het is ook niet bekend of er wel een expliciete voorstelling bestaat. Vermoedelijk hebben Kinnmark en Gray zich niet gerealiseerd dat dit 'gat' er nog is, want in geen van hun artikelen (ook niet in hun artikelen van 1986 en '87) wordt er iets gezegd over de even $I_m^2$ van *even* graad $m$. Een 'kwart' van het vermoeden van Kruseman Aretz blijft dus als open probleem staan.

Ik wil ook nog even terugkomen op wat bovenstaande uitgeschreven voorbeelden voor $I_m^1$ en $I_m^2$ aanvankelijk suggereerden, namelijk dat de coëfficienten misschien wel inversen van gehele getallen zouden zijn. Zoals gezegd, dat is niet juist voor grotere waarden van $m$, maar wél geldt dat de polynomen $Q_m(z) := I_m^1((m-1)z)$ gehele coëfficienten hebben. Dit kan eenvoudig uit de Sonneveld- van Leer- of Kinnmark-Gray-representaties afgeleid worden.

## 2 De jacht op de parabolische polynomen

Parabolische polynomen zijn polynomen van de vorm $P_m^p$ waarin de coëfficienten $\beta_j$ zó gekozen zijn dat $P_m^p$ in modulus $\leq 1$ is op het reële interval $[-\beta, 0]$ voor zo groot mogelijk $\beta$. We zullen ze aangeven met $R_m^p$. Voor $p = 1$ zijn de parabolische polynomen al heel lang bekend en worden gegeven door de 'verschoven' Chebyshev polynomen

$$R_m^1(z) := T_m\left(1 + \frac{z}{m^2}\right), \quad \beta = 2m^2.$$

Omdat $T_m$ rationale coëfficienten heeft, moet $R_m^1$ die ook hebben, zodat je vermoeden ook voor $R_m^1$ geldt. Deze polynomen zijn keer op keer 'herontdekt'. Zover ik weet worden ze het eerst aangegeven in het proefschrift van Yuan' Chzao-Din [16] in verband met de numerieke integratie van lineaire parabolische beginwaardeproblemen. In 1959 komt Franklin met een artikel in Journal for Mathematical Physics [3], en in 1960 komen Guillou en Lago in de Proceedings [4] van de eerste AFCAL conferentie met het dezelfde resultaat. Deze auteurs waren niet op de hoogte van elkanders werk. Zelfs nog onlangs in 1985 kwam Burrage [2] er mee aanzetten. Ook hij was niet op de hoogte van het werk van 25 jaar terug! De reden hiervoor is dat het werk aan parabolische (en hyperbolische) polynomen wel te vinden is in specialistische researchmonografieën, maar niet in de algemenere handboeken voor de numerieke wiskunde.

Voor $p \geq 2$ zijn slechts benaderingen van $R_m^p$ bekend. In het proefschrift van Metzger [12] uit 1967 worden benaderingen voor $p \leq 4$ en $m \leq 5$ gegeven en in een NASA rapport van Lomax [11] uit 1968 wordt een algemene methode aangegeven om de coëfficienten numeriek te berekenen. Deze methode is gebaseerd op het vermoeden dat de optimale polynomen voldoen aan de zogenaamde 'equal ripple' eigenschap. Deze eigenschap houdt in dat het aantal lokale extrema (afwisselend de waarden $+1$ en $-1$ aannemend) gelijk aan $m - p$ is (in 1972 werd deze eigenschap door Riha [13] bewezen). Op grond van deze eigenschap kunnen (niet-lineaire) vergelijkingen voor de coëfficienten $\beta_j$ opgesteld worden. De conditie van deze vergelijkingen van Lomax is echter zo slecht dat het met de in 1968 beschikbare rekenprecisie onmogelijk was de coëfficienten voldoende nauwkeurig te berekenen. Lomax nam toen zijn toevlucht tot kleinste kwadraten benaderingen die echter polynomen leveren die vrij ver verwijderd zijn van de optimale. Ook hier weer kenden Metzger, Lomax en Riha elkanders werk niet!

In 1970 raakten we bij het MC ook geïnteresseerd in de polynomen $R_m^p$. Ook wij kenden toen het werk van Metzger en Lomax nog niet. Het lukte om een iteratiemethode te construeren die de coëfficienten met hoge precisie benaderde. Voor $p \leq 4$ en $m \leq 10 + p$ zijn deze coëfficienten en de bijbehorende reële stabiliteitsgrens $\beta$ gepubliceerd in [5,6]). Op grond van deze numerieke benaderingen konden we concluderen dat de stabiliteitsgrens $\beta$ met een tweede macht van $m$ toeneemt (de imaginaire stabiliteitsgrens is slechts lineair in $m$!). Om precies te zijn, er geldt

$$\beta \approx \gamma_p m^2 \text{ voor } m \to \infty, \ \gamma_2 = 0.814, \ \gamma_3 = 0.489, \ \gamma_4 = 0.341.$$

Toch was de situatie niet bevredigend, want evenals bij de integratie van hyperbolische differentiaalvergelijkingen is het voor een efficiente integratiemethode zeer wenselijk over een expliciete representatie van $R_m^p$ te beschikken, die voor alle $m$ geldt en die aan een recurrente betrekking voldoet. Er is dan ook gezocht naar expliciete representaties van de vorm $P_m^p$ met een 'relatief' grote reële stabiliteitsgrens $\beta$. Een voorbeeld van zo'n benadering voor $p = 2$ zijn de Bakker-polynomen [1] die aan een drietermsrecursie voldoen, maar 'slechts' een stabiliteitsgrens $2(m^2 - 1)/3 \approx 0.67m^2$ bezitten.

Ik heb al deze details over de polynomen $R_m^p$ gegeven om meer indruk te kunnen maken met het volgende resultaat dat we bij toeval hebben gevonden [7]:

**Stelling.** *Er bestaat een expliciete voorstelling voor polynomen van de vorm $P_m^2$ die aan een drietermsrecursie voldoet en reële stabiliteitsgrens $\beta = 2[tan(\pi/2m)]^{-2} \approx 0.810m^2$ heeft.* □

Dit is een verrassend goede benadering (99.5 % van de optimale grens!) voor de optimale grens $\beta = 0.814m^2$. Eerst dachten we dat de bijbehorende polynomen, gegeven door

$$P_m^2(z) = \frac{2}{2-z} - \frac{z}{2-z} T_m\left(\cos(\pi/m) + z\frac{1-cos(\pi/m)}{2}\right),$$

misschien toch optimaal waren, maar dat is niet het geval, want ze hebben niet de 'equal ripple' eigenschap.

Het vinden van polynomen met grote stabiliteitsgrenzen, of algemener, grote stabiliteitsgebieden in het complexe vlak, blijft de numerici intrigeren en is nog steeds actueel (op dit moment is bijvoorbeeld de groep van Lebedev in Moskou zeer actief bezig met dit soort problemen [10]).

Frans, mijn verhaal is uit. Zodra de even $I_m^2$ gevonden worden zal ik het je laten weten!

## Referenties

[1] Bakker, M. (1971): Analytische aspecten van een minimax probleem, Report TN 62, Mathematisch Centrum, Amsterdam.

[2] Burrage, K. (1985): Order and stability properties of explicit multivalue methods, Appl. Numer. Math. 1, 363-379.

[3] Franklin, J.N. (1959): Numerical stability in digital and analogue computation for diffusion problems, J. Math. Phys. 37, 305-315.

[4] Guillou, A. & Lago, B. (1960): Stabiliteitsgebieden van één-staps and meer-staps formules voor differentiaal vergelijkingen (Frans), 1e Congres de l'Association Francaise de Calcul, AFCAL, Grenoble, Sept. 1960, 43-56.

[5] Houwen, P.J. van der (1972): Explicit Runge-Kutta methods with increased stability boundaries, Numer. Math. 20, 149-164.

[6] Houwen, P.J. van der (1977): Construction of integration formulas for initial-value problems, North-Holland, Amsterdam.

[7] Houwen, P.J. van der & Sommeijer, B.P. (1982): A special class of multistep Runge-Kutta methods with extended real stability interval, IMA J. Numer. Anal. 2, 183-209.

[8] Kinnmark, I.P.E. & Gray, W.G. (1984): One-step integration methods with maximum stability regions, Math. Comput. Simulation 16, 87-92.

[9] Kinnmark, I.P.E. & Gray, W.G. (1984): One-step integration methods of third-fourth order accuracy with large hyperbolic stability limits, Math. Comput. Simulation 16, 181-184.

[10] Lebedev, V.L. (1994): How to solve stiff systems of equations by explicit difference schemes, In: Numerical Methods and Applications (ed. G.I. Marchuk), CRC Press, Boca Raton, Ann Arbor, London-Tokyo, 45-80.

[11] Lomax, H. (1968): On the construction of highly stable, explicit numerical methods for integrating coupled ODEs with parasitic eigenvalues, NASA Technical Note NASAIN D/4547.

[12] Metzger, C.L. (1967): Runge-Kutta-methoden waarvan het aantal tussenpunten groter is dan hun orde (Frans), Proefschrift (Troisieme cycle), Université de Grenoble.

[13] Riha, W. (1972): Optimal stability polynomials, Computing 9, 37-43.

[14] Sonneveld, P & Leer, B. van (1985): A minimax problem along the imaginary axis, Nieuw Archief voor Wiskunde (4) 3, 19-22.

[15] Vichnevetsky, R. (1983): New stability theorems concerning one-step numerical methods for ordinary differential equations, Math. Comput. Simulation 25, 199-205.

[16] Yuan' Chzao-Din (1958): Enkele differentieschemas voor de oplossing van het eerste randwaardeprobleem voor lineaire differentiaalvergelijkingen met parti'le afgeleiden (Russisch), Proefschrift, Moskow State University.

# Abstractie en de weg terug

Hans Jonkers

Zoals uit verhitte discussies in verschillende vaktijdschriften en op verschillende conferenties blijkt is het nut van het gebruik van formele technieken in de software-ontwikkeling nog steeds veelomstreden. Een groot deel van deze discussie wordt vertroebeld door het feit dat verschillende interpretaties van het begrip 'formele technieken' gebruikt worden. De een identificeert formele technieken bijvoorbeeld met het gebruik van wiskundige technieken in zijn algemeenheid, de ander met het gebruik van formele specificatietalen en weer een ander met het uitvoeren van formele of gemechaniseerde correctheidsbewijzen.

De basis van iedere formele techniek wordt gevormd door wiskunde. Een formele techniek houdt zich bezig met wiskundige objecten van een bepaald soort en levert mechanismen aan om deze objecten op wiskundige wijze te manipuleren en over deze objecten te redeneren. Dit kan al dan niet ondersteund worden door formele talen en gereedschappen die manipulaties in het wiskundige domein herleiden tot manipulaties op het symbolische vlak.

Het enige excuus om formele technieken niet gewoon gelijk te stellen met wiskunde is dat de wiskundige objecten waarmee deze technieken zich bezighouden iets uitstaande hebben met de problemen die wij hebben bij het ontwikkelen van systemen waarin software een belangrijk onderdeel vormt. De objecten vormen *abstracties* of *modellen* van essentiële aspecten van het te ontwikkelen systeem, en met name van die aspecten die gerelateerd zijn aan de software in dat systeem. De claim is dat via het gebruik van deze abstracties wij deze systemen beter en uiteindelijk ook efficiënter kunnen ontwikkelen.

Dat het gebruik van abstracties in de vorm van modellen zijn nut heeft bij het ontwikkelen van systemen zullen weinigen ontkennen. In de meeste ingenieursdisciplines is het gebruik van dit soort modellen de gewoonste zaak van de wereld. Een civiel ingenieur die een brug moet ontwerpen zal het wel uit zijn hoofd laten om niet eerst een wiskundig model van de brug te maken en daarop uitgebreide sterkteberekeningen uit te voeren. Waarom dan, kan men zich afvragen, is het gebruik van wiskundige modellen en de daarmee samenhangende technieken voor de gemiddelde software-ontwikkelaar niet vanzelfsprekend?

Ten einde een (gedeeltelijk) antwoord te geven op bovenstaande vraag is het nuttig te bezien welke weg een ontwikkelaar dient af te leggen bij het inzetten van formele technieken. Hierbij kan allereerst opgemerkt worden dat systeemontwikkeling in essentie een informeel probleem is. De vraag is hoe te komen van een verzameling informele systeemeisen naar een operationeel systeem dat aan die eisen voldoet. Dit operationele systeem is weliswaar

concreter dan de eisen waaraan het dient te voldoen, maar nog steeds informeel in de zin dat het geen wiskundig object is. Wiskundige objecten gaan bijvoorbeeld nooit kapot.

Bij het ontwerpen van systemen met gebruikmaking van formele technieken heeft men dus noodzakelijkerwijs met twee overgangen te maken: die van informeel naar formeel (de 'heenweg') en die van formeel naar informeel (de 'terugweg'). De heenweg correspondeert met het proces van abstractie, waarin een deelprobleem in de ontwikkeling van het systeem tot idealiter zijn essentie wordt gereduceerd en dit probleem wordt gegoten in een model. Na het volgen van de heenweg bevindt men zich in het wiskundige domein, waarin van allerlei wiskundige technieken van analytische zowel als synthetische aard gebruikt gemaakt kan worden om bepaalde oplossingen voor het probleem te onderzoeken of te construeren. De terugweg bestaat uit de interpretatie van deze resultaten in termen van het operationele systeem, hetgeen in de regel weer leidt tot terugkoppeling op het gekozen model.

Het gemak waarmee ontwikkelaars zich langs deze twee wegen kunnen bewegen is min of meer bepalend voor de effectiviteit van het gebruik van formele technieken. Met andere woorden, ontwikkelaars dienen in staat te zijn simultaan actief te zijn op beide niveaus: het *realisatie-niveau*, waar zij zich in het applicatiedomein bewegen, en het *model-niveau*, waar zij zich in het wiskundige domein bewegen. In het ideale geval schakelt een ontwikkelaar ongemerkt tussen beide niveaus en wordt het systematisch gebruik van wiskundige abstracties een tweede natuur. In de praktijk wringt hier evenwel vaak de schoen.

Veel formele technieken lanceren de argeloze ontwikkelaar vanuit zijn applicatiegebied in een voor hem geheel vreemde wereld. De objecten in deze wereld, zoals initiële algebra's, categorieën, etc. zijn vaak dermate abstract dat hij moeite heeft in te zien wat deze te maken hebben met zijn probleem. Bovendien is er een taalprobleem: de bewoners van deze vreemde wereld drukken zich uit in een soort hiëroglyfen die bekend staan als *formele specificaties*. Een conversatie op niveau in deze wereld wordt gevoerd als een symbolenballet, waarvan de diepere betekenis de ontwikkelaar veelal zal ontgaan.

Het bovenstaande is natuurlijk gechargeerd, maar wat het gevoelen van de gemiddelde industriële ontwikkelaar met betrekking tot formele technieken betreft zal het er niet ver naast zitten. Vaak wordt bij het presenteren van formele technieken nog wel aangegeven hoe zij gezien moeten worden als een abstractie van een stuk praktische problematiek, maar daarna wordt het gebruik ervan gepresenteerd als een formeel spel dat zich in een gesloten wiskundig domein afspeelt. In het extreme geval voert men de 'volledige' ontwikkeling van het systeem binnen het wiskundige domein uit. De weg terug naar het realisatiedomein vindt dan hooguit eenmalig aan het einde van deze ontwikkeling plaats. De gemiddelde ontwikkelaar zal dan evenwel al lang het spoor bijster zijn.

Het is gemakkelijk de oorzaak van het probleem bij de huidige ontwikkelaars te zoeken, die van onvoldoende niveau zouden zijn om formele technieken effectief te gebruiken. Door eenvoudig een nieuwe generatie ontwikkelaars met voldoende abstractievermogen en wiskundige capaciteiten te kweken zou het probleem zich vanzelf oplossen. Nog afgezien van de vraag of deze analyse juist is, heeft de industrie niet de tijd op een dergelijke nieuwe

generatie te wachten en lijkt het beter de hand eerst eens in eigen boezem te steken. De essentiële vraag hierbij is hoe de kloof met de intuïtie van de ontwikkelaar te overbruggen zodat niet alleen de heenweg naar, maar met name de terugweg van het formele domein een route wordt die moeiteloos gevonden wordt.

Een eerste aanzet van een oplossing wordt gevormd door bij het introduceren van formele technieken de nadruk op het gebruik van formalisme te verleggen naar het gebruik van wiskundig modellen. Het zijn deze modellen, d.w.z. de semantische objecten, waarmee de ontwikkelaar vertrouwd dient te zijn wil hij een formele techniek effectief kunnen gebruiken. Het presenteren van de formele techniek vanuit het formalisme kan veelal averechts werken, doordat het formalisme en het symbolisch manipuleren binnen het formalisme als essentie van de techniek gezien worden. De formele taal die gebruikt wordt is alleen een hulpmiddel en bovendien zijn er vele verschillende formalismen denkbaar om dezelfde semantische objecten te beschrijven. Wat dat betreft is het attribuut 'formeel' in 'formele technieken' enigszins misleidend.

De situatie is vergelijkbaar met de manier waarop traditionele wiskunde, zoals bijvoorbeeld klassieke meetkunde, wordt geïntroduceerd. Meetkunde speelt een elementaire rol in veel ingenieursdisciplines. Om te kunnen werken met meetkundige objecten (lijnen, driehoeken, cilinders, etc.) als abstracties van de geometrische aspecten van een te ontwikkelen systeem (zoals een bouwwerk) is het voor een ingenieur allereerst nodig dat hij met deze objecten vertrouwd is en daar op de gangbare wiskundige manier over kan rederen. Het presenteren van formele talen om deze objecten te beschrijven is zeker niet het eerste waaraan men denkt, alhoewel dergelijke talen bij het beschrijven van complexe meetkundige objecten en het ontwikkelen van ondersteunend gereedschap wel van belang zijn.

De theoretische informatica heeft gezorgd voor een overvloed aan verschillende wiskundige modellen die dienen als semantische basis voor een al even grote overvloed aan formalismen. Deze modellen zijn veelal gekozen om de ontwikkeling van wiskundige theorie te optimaliseren. Om dergelijke modellen werkelijk aantrekkelijk te maken als middel om formele technieken in de praktijk te introduceren dient aan hen een dosis 'ingenieurs-intuïtie' toegevoegd te worden, die het de ontwikkelaar vergemakkelijkt dergelijke abstracties in zijn eigen belevingswereld te herkennen en ze ook na manipulatie weer moeiteloos terug te vertalen naar die belevingswereld.

Een voorbeeld is een model voor het dynamisch gedrag van systemen dat gebaseerd is op het semantische model van transitiesystemen, maar aangekleed met voor de ontwikkelaar vertrouwde begrippen zoals processen, events, interrupts, signalen, etc. Een ander voorbeeld is een model voor software-architecturen waarin aan concepten zoals componenten, lagen, 'use'-relaties, lokaliteit, etc. een eenduidige betekenis gegeven worden door ze te baseren op grafentheorie. Essentieel hierbij is dat enerzijds geen compromis gesloten wordt wat betreft het onderliggende wiskundige karakter van de modellen, maar dat anderzijds er een sterke connectie is met concepten zoals die in praktijk van alledag bij software-ontwikkeling een rol spelen.

Er zijn duidelijke indicaties dat deze benadering waarbij het gebruik van modellen in plaats van formalismen vooropstaat het introduceren van formele technieken vergemakkelijkt. Geschikte modellen blijken in de praktijk veel eerder geaccepteerd te worden dan formalismen, waarschijnlijk doordat deze modellen de ontwikkelaar helpen zijn probleem te structureren op een manier die hem niet meteen in een taalkundige dwangbuis perst. Zij vormen daarmee een opstap naar het gebruik van formele beschrijvingstechnieken en geavanceerder gebruik van formele technieken. De roep om dat soort technieken zal vanzelf komen naarmate het gebruik van deze modellen meer ingeburgerd raakt.

Formele technieken zullen uiteindelijk hun ingang vinden in het proces van softwareontwikkeling en daarin hun vruchten afwerpen. Over de vraag hoe dit op afzienbare termijn te bewerkstelligen kan men van mening verschillen. De essentie van dit betoog is dat we er met het ontwikkelen van formele technieken als een soort spel in een gesloten wiskundig domein alleen niet komen. Het is van beslissend belang een ingenieursbenadering te ontwikkelen zodanig dat de ontwikkelaar de weg terug van de abstracties in het formele domein naar zijn eigen praktijk in den blinde weet te vinden.

Beste Frans, ik heb het voorrecht gehad om bij mijn eerste schreden op het informaticapad jou als mijn leermeester te hebben. In jouw rol als docent, afstudeerhoogleraar en later ook als mijn promotor heb jij er altijd blijk van gegeven groot belang te hechten aan het gebruik van abstractie bij het ontwikkelen van software. Hierbij was 'de weg terug' voor jou immer iets vanzelfsprekends. Voor het vele dat ik van je geleerd heb wil ik je hier mijn dankbaarheid uitspreken.

# Mergeable priority queues and heaps

Anne Kaldewaij

**Abstract**

Apart from the introduction, which is about Frans Kruseman Aretz, we present a number of priority queue implementations, all in terms of heaps.

## 1 Introduction

What did Frans like most with respect to his work at Eindhoven University? Certainly, he was not fond of meetings, proposal writing, acquiring funds, and the like. He loved his profession: being a professional computing scientist. Frans studied many different topics and he lectured about most of these (such as denotational semantics, LISP, parsing, concurrent PASCAL, to mention a few). Of all these, he preferred teaching about programming (i.e. the art of designing) above all. I attended a number of his lectures when I started my job at Eindhoven University. I not only learned a lot of the specific subjects but also from the enthousiasm with which he conveyed his ideas, again most dominant in lectures about program design.

As second promotor, Frans carefully studied my thesis, read all (sometimes boring) derivations, and found almost all errors it contained. His advices were to the point and he did not seem to be bothered by all the time it took to read the unreadable parts.

Frans is too modest and too much a gentleman to be a fighter. He plays the game in his own style. In this way, he influenced the development of the new computing science curriculum (which took place in 1987 and 1988) more than he may have realized.

This technical note is devoted to him. Of course, it is about programming.

## 2 Mergeable priority queues and heaps

In computing science (usually) a *heap* is a data structure that implements a priority queue. A *priority queue* is an abstract data type with finite bags as objects and with operations **empty** (initialize as empty bag), **ins** (add an element to a bag), **min** (report the minimum of a bag), **isempty** (report whether a bag is empty), and **delmin** (remove an instance of the minimum from a bag). A *mergeable priority queue* has as additional operation the **merge**

(add two bags). This description and the names of these operations are as one finds them in literature.

For the sake of convenience, we will consider bags over int, the set of integers.

An implementation of a priority queue is a data type $P$, together with an *abstraction function*. The abstraction function is denoted by $[\,]$, i.e. for $x \in P$, $[\![x]\!]$ is the finite bag corresponding to $x$. Furthermore, a set of functions definitions is given, in this case:

| | |
|---|---|
| empty : $P$ | $[\![\text{empty}]\!] = \emptyset$ |
| ins : int $\times P \to P$ | $[\![\text{ins}.a.x]\!] = [\![x]\!] + \{a\}$ |
| min : $P \setminus \{\text{empty}\} \to Z$ | $\text{min}.x =$ minimum of $[\![x]\!]$ |
| delmin : $P \setminus \{\text{empty}\} \to P$ | $[\![\text{delmin}.x]\!] = [\![x]\!] - \{\text{min}.x\}$ |
| isempty : $P \to$ bool | $\text{isempty}.x \;=\; [\![x]\!] = \emptyset$ |
| merge : $P \times P \to P$ | $[\![\text{merge}.x.y]\!] = [\![x]\!] + [\![y]\!]$ |

In this note, we will use node-oriented binary trees (over int) for $P$. Let us denote this set of trees by $T$. It is defined as the smallest set satisfying

$$\langle\rangle \in T$$
$$l \in T \wedge r \in T \wedge a \in \text{int} \;\Rightarrow\; \langle l, a, r\rangle \in T$$

We choose for $P$ the subset of $T$ that satisfies the *heap property* defined by

$$\begin{aligned} &\text{heap}.\langle\rangle &&\equiv \quad \text{true} \\ &\text{heap}.\langle l, a, r\rangle &&\equiv \quad \text{heap}.l \wedge \text{heap}.r \wedge a = \text{minimum}.\langle l, a, r\rangle \end{aligned}$$

where $\text{minimum}.\langle\rangle = \infty$ and $\text{minimum}.\langle l, a, r\rangle = \text{minimum}.l \downarrow a \downarrow \text{minimum}.r$

($a \downarrow b$ denotes the minimum of $a$ and $b$)

The abstraction function is (as expected) given by

$$\begin{aligned} [\![\langle\rangle]\!] &= \emptyset \\ [\![\langle l, a, r\rangle]\!] &= [\![l]\!] + \{a\} + [\![r]\!] \end{aligned}$$

Having made this choice, all that remains to be done is deriving suitable function definitions for empty, add, min, delmin, isempty, and merge. Suitable in the sense that (apart from satisfying their specifications) they should be efficient. Efficiency will be expressed in terms of numbers of unfoldings of functions.

## 3 Solutions

From the choice of $P$ (binary trees that satisfy the heap property), it is easily verified that the following definitions are correct:

$$\begin{aligned} \text{empty} &= \langle\rangle \\ \text{isempty}.x &= x = \langle\rangle \\ \text{min}.\langle l, a, r\rangle &= a \end{aligned}$$

From

$$\begin{aligned} & [\![\text{delmin}\langle l, a, r\rangle]\!] \\ = & \quad \{ \text{specification of delmin} \} \\ & [\![\langle l, a, r\rangle]\!] - \{\text{min}.\langle l, a, r\rangle\} \\ = & \quad \{ \text{definition of min, see above} \} \\ & [\![\langle l, a, r\rangle]\!] - \{a\} \\ = & \quad \{ \text{definition of } [\![\,]\!] \} \\ & [\![l]\!] + \{a\} + [\![r]\!] - \{a\} \\ = & \quad \{ \text{calculus} \} \\ & [\![l]\!] + [\![r]\!] \\ = & \quad \{ \text{specification of merge} \} \\ & [\![\text{merge}.l.r]\!] \end{aligned}$$

we infer that delmin can be expressed in terms of merge. Similarly, addition of a single element can be expressed as the merge with a one-element heap.
Writing the, still to be defined, merge of two heaps as $\bowtie$, we have as general solution:

$$\begin{aligned} \text{empty} &= \langle\rangle \\ \text{isempty}.x &= (x = \langle\rangle) \\ \text{min}.\langle l, a, r\rangle &= a \\ \text{merge}.x.y &= x \bowtie y \\ \text{delmin}.\langle l, a, r\rangle &= l \bowtie r \\ \text{ins}.a.x &= \langle\langle\rangle, a, \langle\rangle\rangle \bowtie x \end{aligned}$$

The only problem left is a program for $\bowtie$.

Note that the first three programs have constant time complexity.

Since sorting a bag can be accomplished by adding all its elements to a priority queue followed by successively removing the minimum from that queue, the complexity of $\bowtie$ cannot be better than logarithmic in the size of its arguments. Hence, in the next section we try to find logarithmic solutions for $\bowtie$.

### 3.1 Programs for $\bowtie$

The specification of $\bowtie$ is given by

$$[\![x \bowtie y]\!] = [\![x]\!] + [\![y]\!]$$

$$\text{heap}.x \wedge \text{heap}.y \;\Rightarrow\; \text{heap}.(x \bowtie y)$$

Substitution of $\langle\,\rangle$ for $x$ or $y$ yields as solution:

$$\begin{array}{lllll} x & \bowtie & \langle\,\rangle & = & x \\ \langle\,\rangle & \bowtie & y & = & y \end{array}$$

Now, let $x$ and $y$ be non-empty heaps. To be a heap, the root of $x \bowtie y$ is the smallest of the roots of $x$ and $y$. Let us assume that $x$ has the smallest root. Writing $x = \langle l, a, r\rangle$, we then have:

$$\langle l, a, r\rangle \bowtie y = \langle u, a, v\rangle$$

where $u$ and $v$ should be such that

$$u \text{ and } v \text{ are heaps}$$

$$[\![u]\!] + [\![v]\!] = [\![l]\!] + [\![r]\!] + [\![y]\!]$$

These requirements will be satisfied by induction. Possible straightforward solutions are:

$$\begin{array}{lll} \langle l, a, r\rangle \bowtie y & = & \langle l \bowtie r, a, y\rangle \\ & & \langle l \bowtie y, a, r\rangle \\ & & \langle l, a, r \bowtie y\rangle \\ & & \langle r \bowtie y, a, l\rangle \\ & & \langle r, a, l \bowtie y\rangle \end{array}$$

The only reason to choose between these is efficiency. The first proposal is immediately dropped, since after one unfolding $y$ does not play a rôle any more: all depends on the structure of one of the arguments only.

Let us consider the second proposal. The merge of $x$ and $y$ gives rise to the merge of the left subtree of $x$ and $y$. The only (reasonable) thing one can claim about the left subtree

of $x$ is that its height is at least one smaller than the height of $x$. Hence, this is a solution that has a time complexity bounded by the sum of the heights of $x$ and $y$. For heights to be logarithmic in the size, some kind of tree balancing is required. However, how this choice for $\bowtie$ should be adapted to keep the resulting merge height-balanced again is not clear at all.

Another technique is to make a combination of possible solutions. We may consider a solution of the shape

$$\begin{array}{lcll} \langle l,a,r\rangle \bowtie y & = & \langle l \bowtie y,a,r\rangle, & A \\ & & \langle l,a,r \bowtie y\rangle, & B \end{array}$$

where $A$ and $B$ are boolean expressions such that $A \vee B$ holds.

One could try solutions such as

$$\begin{array}{lcll} \langle l,a,r\rangle \bowtie y & = & \langle l \bowtie y,a,r\rangle, & \text{size}.l \leq \text{size}.r \\ & & \langle l,a,r \bowtie y\rangle, & \text{size}.r \leq \text{size}.l \end{array}$$

or

$$\begin{array}{lcll} \langle l,a,r\rangle \bowtie y & = & \langle l \bowtie y,a,r\rangle, & \text{height}.l \leq \text{height}.r \\ & & \langle l,a,r \bowtie y\rangle, & \text{height}.r \leq \text{height}.l \end{array}$$

This way of trying to find good solutions does not help very much. Both proposals turn out to be inadequate.
Functions height and size have the property that their values decrease in each unfolding of the merge as proposed above. However, neither of these is logarithmic in the size of its argument.

A function that has this last property is the *shortest path length*, denoted sp, which is defined by

$$\begin{array}{lcl} \text{sp}.\langle\,\rangle & = & 0 \\ \text{sp}.\langle l,a,r\rangle & = & 1 + \text{sp}.l \downarrow \text{sp}.r \end{array}$$

Indeed, we have $\text{sp}.x \leq \log \#x$, where $\#x$ is defined by

$$\begin{array}{lcl} \#\langle\,\rangle & = & 1 \\ \#\langle l,a,r\rangle & = & \#l + \#r \end{array}$$

i.e., $\#x$ is one more than the size of $x$ (this definition is chosen to avoid logarithms with argument 0, and to obtain an easier recurrence relation). We prove the relation between sp and $\#$ using induction.

For the base, we have $2^{\mathrm{sp}.\langle\rangle} = \#\langle\rangle$, and for the step, we derive

$$
\begin{array}{ll}
 & 2^{\mathrm{sp}.\langle l,a,r\rangle} \\
= & \quad \{\text{definition of sp}\} \\
 & 2^{1+\mathrm{sp}.l \downarrow \mathrm{sp}.r} \\
= & \quad \{\text{calculus}\} \\
 & 2 * (2^{\mathrm{sp}.l \downarrow \mathrm{sp}.r}) \\
= & \quad \{\text{monotonicity of exponentiation}\} \\
 & 2 * (2^{\mathrm{sp}.l} \downarrow 2^{\mathrm{sp}.r}) \\
\leq & \quad \{\text{induction}\} \\
 & 2 * (\#l \downarrow \#r) \\
\leq & \quad \{\text{calculus}\} \\
 & \#l + \#r \\
= & \quad \{\text{definition of } \#\} \\
 & \#\langle l,a,r\rangle
\end{array}
$$

If we program the merge (for the case that the root of $x$ is at most the root of $y$) as

$$
\begin{array}{lll}
\langle l,a,r\rangle \bowtie y & = & \langle l \bowtie y,a,r\rangle, \quad \mathrm{sp}.l \leq \mathrm{sp}.r \\
 & & \langle l,a,r \bowtie y\rangle, \quad \mathrm{sp}.r \leq \mathrm{sp}.l
\end{array}
$$

then the sum of the sp-values of the arguments of $\bowtie$ decreases by at least one in each unfolding, hence, we have obtained a logarithmic solution.

As an extra attribute, we have to add an sp-value to each node of each heap, and we have to maintain these values when merging heaps. Since sp is a *decomposable function* (cf. [3]) this poses no problems. A complete program text has as case analysis:

- empty versus non-empty
- which heap contains the minimum
- which subtree has the smallest sp-value

In the next section, we consider the *leftist heap*, invented by Crane ([1]), and described in [5], in which the last case is eliminated.

### 3.2 Leftist heaps

To avoid case analysis in program texts, we may restricts our heaps to the class of heaps known as *leftist heaps* (cf. [1]). For a leftist heap, the shortest paths are alway in the right subtrees. Formally, this class may be defined by

$$\begin{aligned} \text{leftist.}\langle\rangle &= \text{true} \\ \text{leftist.}\langle l, a, r\rangle &= \text{leftist.}l \wedge \text{leftist.}r \wedge \text{sp.}r \leq \text{sp.}l \end{aligned}$$

Being leftist is a very modest balance criterium. Using as auxiliary function balance, given by:

$$\begin{aligned} \text{balance.}\langle l, a, r\rangle &= \langle l, a, r\rangle, \quad \text{sp.}r \leq \text{sp.}l \\ &\quad\ \langle r, a, l\rangle, \quad \text{sp.}l \leq sp.r \end{aligned}$$

we obtain as solution to the merge:

$$\begin{aligned} x \bowtie \langle\rangle &= x \\ \langle\rangle \bowtie y &= y \\ \langle l, a, r\rangle \bowtie \langle u, b, v\rangle &= \text{balance.}\langle l, a, r \bowtie \langle u, b, v\rangle\rangle, \quad a \leq b \\ &\quad\ \text{balance.}\langle u, b, v \bowtie \langle l, a, r\rangle\rangle, \quad b \leq a \end{aligned}$$

Adding elements to an initially empty heap in this way leads to a tree that leans over to the left. Operationally speaking the $\bowtie$ can be viewed as a merge of the rightmost paths of the trees involved, whereas the more general solution of the the previous section can be viewed as a merge along the shortest paths of the trees involved. In the case of a leftist heap, the righmost path is a shortest path to a leaf.

### 3.3 Skew heaps

The fact that the additional attribute sp is needed to obtain a worst case logarithmic solution seems unavoidable. If we are satisfied with an *amortized* logarithmic solution (for instance, when the priority queue is used in a sequential program, such as sorting), then no attribute is needed. One can choose as merge one of the other alternatives:

$$\begin{aligned} \langle l, a, r\rangle \bowtie \langle u, b, v\rangle &= \langle r \bowtie \langle u, b, v\rangle, a, l\rangle, \quad a \leq b \\ &\quad\ \langle v \bowtie \langle l, a, r\rangle, b, u\rangle, \quad b \leq a \end{aligned}$$

This gives rise to the *skew heaps*, invented by Sleator and Tarjan (operationally described in [6]). As a potential function they use the number of nodes in a tree for which the right subtree contains more elements than the left subtree. In [4] a tighter bound has been derived, using a more complicated potential function. The treatment thereof, however, would make this paper too long and much less attractive.

## 4 Concluding remarks

We have tried to show how different implementations of mergeable priority queues can be obtained in a systematic way. The functional description is much clearer than the operational (imperative) description one finds in literature.

One may wonder why the beloved leaf trees have not been used. The point is that in leaf tree implementations all values reside in leaves, and, hence, operation delmin will inevitably give rise to solutions that have a time complexity in which the height appears. For some games, the node-oriented tree is not too bad.

It is, however, possible to design a leaf tree based solution for which the merge is logarithmic in the size of the tree. The treatment thereof will eventually appear at another opportunity.

## 5 Acknowledgements

The observation that the leftist heap is just an instance of shortest path merging is due to Victor J. Dielissen (cf. [2]), who investigated a large number of existing priority queue implementations. He invented the first (new) solution and called it *weaving trees*, since the inorder traversal of the merge is the weave of the inorder traversals of the arguments. A systematic analysis of possible alternatives for merging heaps was done together with Berry Schoenmakers. This lead to the publication of [4].
My contribution consists mostly of polishing, and leaving out the ugly stuff that emerges when studying data structures.

## References

[1] Crane C.A.,
Linear lists and priority queues as balanced binary trees
PhD Thesis, Stanford University, 1972.

[2] Dielissen V.J., private communication.

[3] Kaldewaij A. and Dielissen V.J.,
Decomposable functions and leaf trees: a systematic approach,
in Programming, concepts, methods and calculi: proceedings of the IFIP TC2/WG2.1/WG2.2/WG2.3 Working Conference on Programming Concepts, Methods and calculi (PROCOMET'94), San Miniato, Italy, 6-10 June 1994 / edited by Ernst-Rüdiger Olderog, Elsevier, Amsterdam, 1994, pp 3-17.

[4] Kaldewaij A. and Schoenmakers L.A.M., the derivation of a tighter bound for top-down skew heaps, Information Processing Letters 37 (1991), 265–271.

[5] Knuth D.E.,
The art of computer programmaing 3: Sorting and Searching
Addison Wesley, 1973.

[6] Sleator D.D. and Tarjan R.E., Self-adjusting heaps, SIAM J. Comput. 15 (1986), 52–69.

# The problem of the beer glasses in the Polish barrel

Joep Kessels

## 1 Introduction

In 1979 a problem reached the NatLab under the name of "Beer glasses in the Polish barrel". Allegedly the problem originated from a Polish logician, which partly explains the name. The problem is to devise a winning strategy for a game against a clever demon, which resides in a barrel together with four beer glasses. The glasses are placed on a disk at the four points of the compass in either an upright or an upside-down position. The barrel is closed and its only information-providing mechanism is a bell which rings as soon as all four glasses are upright. The goal of the game is to make the bell ring. The player makes a move by indicating a subset of the compass points, subsequently the demon rotates the disk over a multiple of $90^0$ after which the glasses at the indicated locations will be turned over.

In the next section we will do some mathematics resulting in a function mapping the set of beer glass configurations onto natural numbers. For each configuration the function's value can be seen as a measure of its regularity, the configuration with all four glasses upright being the most regular one. As may be expected from such a function, its value does not depend on the orientation of the configuration and therefore remains invariant under any move of the demon. Using this function we will first devise a winning strategy for the case when the barrel is open and the player can see the beer glasses. Subsequently we will solve the much harder original problem described above.

## 2 First some mathematics

The solution we present holds not only for four beer glasses, but for any number of glasses being a power of two. In the analysis given below we will assume $N$ to be a power of two. The locations of the beer glasses are numbered from 0 to $N-1$ with glass $N-1$ next to glass 0. A configuration of glasses is represented by the pair $(N, A)$, where $N$ indicates the number of beer glasses and $A$ [1] is a polynomial representing the beer glass orientations. The degree of $A$ is less than $N$ and each coefficient $A_n$ is a binary value indicating the orientation of the beer glass at location $n$: 0 means upright and 1 means upside down.

[1] We use a polynomial name like $A$ as a shorthand for the expression $\sum_{n=0} A_n * x^n$.

Turning a beer glass over then corresponds to adding 1 in $GF(2)$ to the corresponding coefficient. A move of the player can therefore also be represented by a polynomial with a degree less than $N$, which has to be added in $GF(2^N)$ to the configuration polynomial.

We define the *parity* of a polynomial $A$, denoted by $par(A)$, as the sum in $GF(2)$ of its coefficients.

$$par(A) \triangleq \sum_{n=0} A_n.$$

**Lemma 1**

$$par(A) = 0 \;\Leftrightarrow\; (x+1)|A,$$

where $P|Q$ stands for $P$ divides $Q$.

Proof: From the definition of *par* follows $par(A) = A(1)$, which implies that $par(A) = 0$ means that 1 is a root of $A$. In the formula given above we have used the fact that the plus and minus operation are the same in $GF(2)$.

A rotation over one location corresponds to the function $rot_N$ defined below, which multiplies its argument polynomial by $x$ modulo $(x^N+1)$. The modulo reduction comes from the fact that the glasses are placed in a circular arrangement with $x^N = x^0$, which can be rewritten as $x^N + 1 = 0$.

$$rot_N(A) \triangleq (x * A) \textbf{ mod } (x^N + 1).$$

The following lemma is a special case of what is called the freshman's dream [2]. It shows that in $GF(2)$ exponentiation distributes over addition provided the exponent is a power of two.

**Lemma 2** For polynomials $A$ and $B$

$$(A+B)^N = A^N + B^N.$$

Proof: The proof is by induction on the powers of two. The lemma clearly holds for $N = 1$ and from the fact that $(A+B)^2 = A^2 + B^2$ in $GF(2)$ it follows that if the lemma holds for $N$, it also holds for $2 * N$.

We now define the function *reg*, which maps beer glass configurations onto natural numbers.

$$reg(N, A) \triangleq (\textbf{max } n :\ 0 \le n \le N :\ (x+1)^n | A).$$

If polynomial $A$ is zero, $reg(N, A)$ is $N$, otherwise it is the multiplicity of root 1. The definition of $reg(N, 0)$ as $N$ agrees with the fact that $(x+1)^N$ is equivalent to 0.

The lemma given below follows directly from the definition of *reg*.

[2] Pointed out by Ad Peeters

**Lemma 3** For configuration $(N, A)$ with nonzero polynomial $A$ and $k \triangleq reg(N, A)$,

$$(k < N) \wedge par(A/(x+1)^k) = 1.$$

The value $reg(N, A)$ can be seen as a measure of the regularity of configuration $(N, A)$ and therefore we will call it the *regularity degree* of the configuration.

The table given below shows the regularity degree for all configurations with four beer glasses. A regularity degree of 4 means that all glasses are upright and degree 3 means that all glasses are upside down. The two configurations with alternating beer glass orientations have degree 2. The four configurations with two neighbouring glasses upright and the other two upside-down have degree 1. The eight configurations in which one beer glass orientation differs from the others have the minimum degree 0.

| $A \in$ | $reg(4, A)$ |
|---|---|
| $\{0\}$ | 4 |
| $\{1+x+x^2+x^3\}$ | 3 |
| $\{1+x^2, x+x^3\}$ | 2 |
| $\{1+x, x+x^2, x^2+x^3, x+x^3\}$ | 1 |
| $\{1, x, x^2, x^3, 1+x+x^2, x+x^2+x^3, 1+x^2+x^3, 1+x+x^3\}$ | 0 |

Note that for each regularity degree the corresponding set of configurations is closed under rotation. Moreover if the degree is less than 3, the set is also closed under inversion of all glasses.

Below we prove that the regularity degree of a configuration remains invariant under rotation.

**Theorem 1**

$$reg(N, rot_N(A)) = reg(N, A).$$

Proof: The theorem trivially holds for $A = 0$. We will prove the theorem for $A \neq 0$. We define $k \triangleq reg(N, A)$ and $P \triangleq A/(x+1)^k$. Lemma 3 implies $par(P) = 1$.

$$\begin{aligned} & reg(N, rot_N(A)) \\ = & \quad \{ \text{ definition } rot_N \ \} \\ & reg(N, (x * A) \textbf{ mod } (x^N + 1)) \end{aligned}$$

$$
\begin{array}{ll}
= & \{\ (x * A)\ \mathbf{div}\ x^N = A_{N-1}\ \} \\
& reg(N, x * A + A_{N-1} * (x^N + 1)) \\
= & \{\ \text{lemma 2}\ \} \\
& reg(N, x * A + A_{N-1} * (x+1)^N) \\
= & \{\ A = (x+1)^k * P \text{ and } k < N\ \} \\
& reg(N, (x+1)^k * (x * P + A_{N-1} * (x+1)^{N-k})) \\
= & \{\ (x * P + A_{N-1} * (x+1)^{N-k}) \text{ not divisible by } (x+1), \\
& \quad \text{since } par(x * P) = 1 \ \wedge\ par(A_{N-1} * (x+1)^{N-k}) = 0\ \} \\
& k.
\end{array}
$$

From this theorem it follows that the demon cannot modify the regularity degree of the beer glass configuration. The player, however, can change the regularity degree. We will now present two theorems, which are going to be used in the next section to devise a winning strategy.

**Theorem 2**

$$reg(N, A + B) \geq min(reg(N, A), reg(N, B)).$$

Proof: If $k = min(reg(N, A), reg(N, B))$, both $A$ and $B$ are divisible by $(x+1)^k$, which is therefore also a divisor of $A + B$.

The theorems 1 and 2 show that for any $k$ the set of polynomials with a degree of at least $k$ is closed under rotation (moves of the demon) and addition (moves of the player).

**Theorem 3** For two nonzero polynomials $A$ and $B$

$$(reg(N, A) = k \ \wedge\ reg(N, B) = k) \ \Rightarrow\ reg(N, A + B) > k.$$

Proof: We define $P \triangleq A/(x+1)^k$ and $Q \triangleq B/(x+1)^k$ and from the fact that $A$ and $B$ are nonzero it follows that $par(P) = 1$ and $par(Q) = 1$.

$$
\begin{array}{ll}
& reg(N, A + B) \\
= & \{\ \text{definitions } P \text{ and } Q\ \} \\
& reg(N, (x+1)^k * (P + Q)) \\
> & \{\ (P+Q) \text{ divisible by } (x+1), \text{ since } par(P+Q) = 0\ \} \\
& k.
\end{array}
$$

## 3 Playing the game

After the analysis given in the previous section it is easy to play the game when the glasses in the barrel can be seen. If the player gives the existing beer glass configuration as his next move, both the configuration and the move have the same regularity degree. From theorem 3 it then follows that the move will increase the configuration's regularity degree, which of course can only happen a finite number of times before the bell rings.

If the barrel is closed, however, devising a winning strategy is not such a trivial matter. The closure property of theorem 2 suggests to make only moves that have a regularity degree that is not smaller than that of the configuration's one. Each move must either increase the configuration's regularity (when both degrees are equal) or it must leave the regularity invariant and instead increase the player's knowledge about the configuration.

Let us try to solve the problem through recursion. We are going to define recursive procedure $strat(n)$, which generates a sequence of moves that makes the bell ring when the regularity degree of the configuration is at least $n$ and otherwise leaves the regularity degree invariant. From the definition it follows that the empty sequence suffices for $strat(N)$. For $n < N$ we can now construct $strat(n)$ provided we have $strat(n+1)$. If we first apply $strat(n+1)$ without getting the bell to ring, we can be sure that the configuration's regularity has invariantly remained at most $n$. We can then make a move with a degree $n$, for instance $(x+1)^n$. If the configuration's degree is $n$, the move will increase it to at least $n+1$ and subsequent application of $strat(n+1)$ will therefore make the bell ring. If, however, the degree is less than $n$, all moves will have left the degree invariant.

$$
\begin{aligned}
strat(n) \triangleq\ & \textbf{if } n = N \rightarrow \textbf{skip} \\
& \| \; n < N \rightarrow strat(n+1);\ move((x+1)^n);\ strat(n+1) \\
& \textbf{fi}.
\end{aligned}
$$

From the definition it follows that procedure call $strat(0)$ generates the winning strategy.

As an example we give the winning strategy for four beer glasses. The move polynomials are represented by integers on the basis of the fact that if we interpret such a polynomial $A$ as an integer polynomial, the value $A(2)$ uniquely represents $A$.

$$
\underbrace{\overbrace{15,5,15}^{strat(2)},3,\overbrace{15,5,15}^{strat(2)}}_{strat(1)},1,\underbrace{\overbrace{15,5,15}^{strat(2)},3,\overbrace{15,5,15}^{strat(2)}}_{strat(1)}
$$

## 4 Some concluding remarks

I remember discussing the problem in 1979 with Han Boekhorst and Pierre Jansen in the pub "'t Rozenknopje" and as usual Han came up with a generalization of the problem.

He invented beer glasses that have a prime number of stable positions. The corresponding number of beer glass locations must then be a power of that prime number. When Han and I left the sector Bosma, we both received an electronic version of the Polish barrel built by Gert Slavenburg.

My interest in the problem was rewoken when I heard that in 1992, at a workshop in Oxford, Jan van de Snepscheut posed the problem again but now in terms of coins at the corners of a table. With Tom Verhoeff's help I was able to get some indications of the origin of the problem. Martin Gardner discusses a similar problem in his Mathematical Games column in two subsequent issues of the Scientific American [1, 2]. He states that the problem was passed to him by somebody who believes it comes from the then still existing USSR. The problem is formulated in terms of drinking glasses on a square table and it differs from the problem described above in that the glasses are placed at the bottom of deep wells at the corners of the table, where they cannot be seen. In each move you first spin the table after which there is no way of distinguishing the corners. The player then reaches with each hand into a different well and may choose to adjust the orientation of the glasses. This time the bell rings already when all four glasses have the same orientation. Several generalizations of this problem are presented in [3].

Starting from the winning strategy for the Polish barrel problem with four glasses given in the previous section makes solving the problem described above a piece of cake. The weaker condition of getting the bell to ring allows us to remove all moves 15 from the Polish barrel strategy. The remaining sequence 5,3,5,1,5,3,5 contains only moves in which at most two glasses are turned over. This sequence is therefore a winning strategy for the problem posed by Martin Gardner and, what is more, it can even be played by somebody who cannot distinguish an upright glass from an upside-down one.

For a player who can make that distinction, however, shorter sequences exist. Note that such a player can make moves that do not correspond to addition and the closure property of section 2 therefore no longer holds. My son Roel pointed out that for instance the first three moves, which either make the bell ring or leave the configuration's regularity degree at one, can be replaced by two moves, each of which ends with two glasses upright: one move takes two opposite glasses and the other two neighbouring ones. The result of both moves is that at most one of the glasses is upside down. The next two moves, namely 1 followed by 5, which either make the bell ring or leave the configuration's regularity degree at least at three, can then be replaced by one move in which the player inspects the orientation of two opposite glasses. If the upside-down glass is detected, the bell is made to ring by turning that glass over, otherwise one of the two inspected glasses is turned over.

I must admit that the regularity-degree function comes like a rabbit out of the hat. When writing this paper I tried to introduce it by reasoning but did not succeed. Perhaps I did not try hard enough, because the rabbit-out-of-the-hat character agrees with the pleasant surprise it gave me when I stumbled upon it. I have always been convinced that engineering problems must exist in which the regularity-degree function can be used,

but until now I have not encountered any. Below we give four more properties of the regularity-degree function. We assume the degree of $A$ to be less than $N$.

- If not all beer glasses have the same orientation, the regularity degree remains invariant under inversion of all beer glass orientations. In a formula:

$$reg(N, A) < N - 1 \;\Rightarrow\; reg(N, A) = reg(N, A + (x + 1)^{N-1}).$$

  Since $(x+1)^{N-1} = 1 + x + x^2 + ...x^{N-1}$, adding $(x+1)^{N-1}$ to $A$ amounts to bitwise inversion of the $N$ least significant bits of $A$.

- For configurations of which one half is zero and the other is not zero we have

$$A \neq 0 \;\Rightarrow\; reg(2 * N, A) = reg(N, A).$$

- For configurations that consist of two equal parts we have

$$reg(2 * N, A + x^N * A) = reg(N, A) + N.$$

- For configurations of which one half is the inverse of the other we have

$$reg(2 * N, A + x^N * A + (x + 1)^{N-1}) = N - 1.$$

**Acknowledgements**

I want to thank Frans Kruseman Aretz for the many times he reviewed my papers and gave valuable suggestions for improvements. When writing this paper I struggled with the notation and I am sure his advice would have been of great help.

## References

[1] Martin Gardner, *Scientific American*, Feb. 1979, p. 16.

[2] Martin Gardner, *Scientific American*, March 1979, pp. 19-20.

[3] William T. Laaser and Lyle Ramshaw, *Probing the Rotating Table* In *The Mathematical Gardner* Ed David A. Klarner; Wadsworth International, Belmont CA 1981, pp. 285-307.

# From line numbers to origins

Paul Klint

*Dedicated to Frans Kruseman Aretz.*

**Abstract**

Many aspects of computer science were first shown to me by Frans Kruseman Aretz in the period 1969–1971.

The first computer architecture I ever saw was the OEBRA (for Onbestaanbare Elektronische Binaire Rekenautomaat): a very simple machine originally designed by A. van der Sluis. Frans explained it in full detail by giving all hardware diagrams as well as an interpreter for OEBRA assembly language programs (written in Algol60). Later on he showed how a real PDP8 (with a main memory of 4K 12 bit words) could assemble programs by exclusively using paper tape as storage medium (for representing the assembler itself, the source program, the intermediate output of the assembler, and, finally, the binary version of the source program).

On another occasion we visited the machine room of the EL-X8 of the Mathematisch Centrum then located at the Tweede Boerhaevestraat 48 in Amsterdam. Acting like a professional piano player, Frans used the switch register for entering a program that would set the *complete* memory (28K 27 bit words) of the machine to zero's. Before pushing the "run" button, however, he asked each student how long it would take to execute this program. Probably due to selective amnesia, I do not remember what my guess was, but the very short flash of the lights on the console registers while executing the program impressed me deeply. This X8 was a really fast machine with its 2.5$\mu$sec cycle time (= 0.4 MHz)!

The first operating system I saw was the Milli operating system for the X8 (written by Frans Kruseman Aretz in ELAN, the assembly language of the X8). The first compiler I saw was the Algol60 compiler for the X8 (written, again, by Frans Kruseman Aretz in Algol60). His standard approach was to give a brief global overview of the issues and techniques involved and then start a meticulous explanation at the source code level of each system he was discussing.

I remember various small, but very instructive, programs he treated during his courses: for instance, a Lisp interpreter in Algol60 (highlighting variable binding and recursive evaluation), and an interpreter for Turing machines (using Algol60's call-by-name mechanism to represent the infinite tape of the Turing machine). A puzzle he gave me once was to determine the number of logical functions of three variables that can be built using at most one inverter gate.

Frans supervised my Master's thesis [Kli73] concerning a semi-automatic proof of the termination of the X8 Algol60 compiler as well as my PhD thesis [Kli82] on

the design of string manipulation languages. Clearly, I owe very much to him and I am glad to contribute the following paper on the occasion of his retirement from the University of Eindhoven.

This contribution is devoted to another theme of common interest (which can, for instance, be verified by inspecting the bibliographical notes in [WG84], p323): maintaining references to a program's source text during its execution. [Kru71] deals with a technique for generating the minimal number of instructions for correctly maintaining line numbers at run time. [Kli79] deals with a technique for concisely encoding line numbers in abstract machine instructions.

I give a new formalization of *origin tracking* [vDKT93], a technique for maintaining references between the normal form and the initial term of a term rewriting process.

## 1 Introduction

Term rewriting systems are frequently used to execute algebraic specifications: the equations in the specification are interpreted as rewrite rules and a given initial term is reduced according to these rules; if no further reductions are possible we have obtained an irreducible term (the *normal form*) constituting the answer of the computation.

A typical function in a specification (such as an evaluator, type checker or translator) operates on the abstract syntax tree of a program (which is part of the initial term). During the term rewriting process, pieces of the program such as identifiers, expressions, or statements, recur in intermediate terms.

In [vDKT93] we have introduced the notion of "origin tracking": by establishing reverse links from the normal form to the initial term of the term rewriting process we obtain information that is important for interactive tools like error reporters (associate an error message with a part of the source program) and animators (visualize the statement currently being executed).

The existing formalization of origin tracking proceeds in two stages. First, relations are defined for elementary reduction steps $T_i \rightarrow T_{i+1}$. Next, these relations are extended to complete reduction sequences $T_0 \rightarrow T_1 \rightarrow ... \rightarrow T_n$. In particular, relations are established between subterms of an intermediate term $T_i$, and subterms of the initial term $T_0$. The process of incrementally computing origins is called origin tracking. In Appendix B of [vD94] an ASF+SDF specification is given following along these lines. Here, I want to simplify the two stage approach by concentrating on the information that has to be propagated for elementary reduction steps only. In this way, we get a direct formalization of origin tracking itself. The basic intuition of this new approach is to attach unique labels to the initial term and to define how these labels are propagated during rewriting.

First, a straightforward formalization of term rewriting is given in Section 2 and then we extend it in Section 3 with origin tracking.

# 2 Term rewriting

## 2.1 Terms

First, we define the basic syntactic structure of terms. Observe that functions are always unary, but they may operate on lists. We assume (but dot not show) a module `Layout` containing definitions for comments and white space.

### Module BasicTerms

**imports** Layout(2.1)

**exports**

**sorts** FUN VAR TERM

**lexical syntax**

```
[a-z][A-Za-z0-9\-]* → FUN
[A-Z][A-Za-z0-9\-]* → VAR
```

**context-free syntax**

```
VAR                 → TERM
nil                 → TERM
TERM ";" TERM       → TERM  {right}
FUN "(" TERM ")"    → TERM
"(" TERM ")"        → TERM  {bracket}
```

**variables**

```
Var [0-9']*  → VAR
Fun [0-9']*  → FUN
T [0-9']*    → TERM
```

**equations**

Ensure that lists are always right-associative.

$$(T_1; T_2); T_3 = T_1; T_2; T_3 \qquad \text{[list-1]}$$

Next, we introduce functions for classification, selection (i.e., decomposition) and replacement of terms. We assume (but do not show) a definition of the module `Booleans`.

## Module Terms

**imports** BasicTerms$^{(2.1)}$ Booleans$^{(2.1)}$

**exports**

**context-free syntax**

```
is-fun(TERM)                  → BOOL
is-non-empty-list(TERM)       → BOOL
is-nil(TERM)                  → BOOL

fun(TERM)                     → FUN
arg(TERM)                     → TERM
head(TERM)                    → TERM
tail(TERM)                    → TERM

repl-arg(TERM, TERM)          → TERM
repl-list(TERM, TERM, TERM)   → TERM
```

**equations**

Classification functions on terms.

```
is-fun(Fun(T)) = true                                   [is-fun-1]
is-fun(T)      = false  otherwise                       [is-fun-2]

is-non-empty-list(T1; T2) = true                        [is-non-empty-list-1]
is-non-empty-list(T)      = false  otherwise            [is-non-empty-list-2]

is-nil(nil) = true                                      [is-nil-1]
is-nil(T)   = false  otherwise                          [is-nil-2]
```

Selection functions on terms.

```
fun(Fun(T)) = Fun                                       [fun-1]
fun(T)      = error  otherwise                          [fun-2]

arg(Fun(T)) = T                                         [arg-1]
arg(T)      = nil  otherwise                            [arg-2]

head(T1; T2) = T1                                       [head-1]
head(T)      = nil  otherwise                           [head-1]
```

$$\text{tail}(T_1;\ T_2) = T_2 \qquad \text{[tail-1]}$$
$$\text{tail}(T) = \text{nil} \quad \textbf{otherwise} \qquad \text{[tail-2]}$$

Replacement functions on terms. Given a function application (or a list) and a new argument (or two new list elements) construct a new function application (or list). In this manner replacement operations are made explicit and can be extended later on (see Section 3.2).

$$\text{repl-arg}(\mathit{Fun}(T_1),\ T_2) = \mathit{Fun}(T_2) \qquad \text{[repl-arg-1]}$$
$$\text{repl-arg}(T_1,\ T_2) = T_1 \quad \textbf{otherwise} \qquad \text{[repl-arg-2]}$$

$$\text{repl-list}(T_1;\ T_2,\ T_1',\ T_2') = T_1';\ T_2' \qquad \text{[repl-list-1]}$$
$$\text{repl-list}(T,\ T_1',\ T_2') = T_1';\ T_2' \quad \textbf{otherwise} \qquad \text{[repl-list-2]}$$

The definition of a replacement function for complete terms (`repl-term`) will be given in Section 2.4.

## 2.2 Substitutions

Substitutions define a mapping from variables to terms and are represented as lists of the form $[Var_1 \rightarrow Term_1, ..., Var_n \rightarrow Term_n]$. Three operations are defined for substitutions: composition of two substitutions ($\sigma_1 \circ \sigma_2$), application of a substitution $\sigma$ to a term $T$ ($T^\sigma$), and checking that a variable $Var$ is in the domain of a substitution $\sigma$ ($Var \in \sigma$).

### Module Substitutions

**imports** Terms(2.1) Booleans(2.1)

**exports**

**sorts** SUBSTITUTION ONE-SUBS

**context-free syntax**

```
VAR ↦ TERM                    → ONE-SUBS
"[" {ONE-SUBS ","}* "]"       → SUBSTITUTION
SUBSTITUTION ∘ SUBSTITUTION   → SUBSTITUTION

TERM "^" SUBSTITUTION         → TERM

VAR ∈ SUBSTITUTION            → BOOL
SUBSTITUTION "(" VAR ")"      → TERM
```

**exports**

**variables**

$Subs\ [0\text{-}9']{*}\text{"*"} \rightarrow \{\text{ONE-SUBS ","}\}{*}$
$\sigma\ [0\text{-}9']{*} \rightarrow \text{SUBSTITUTION}$

**equations**

Composition of substitutions.

$$[Subs_1^*] \circ [Subs_2^*] = [Subs_1^*, Subs_2^*] \quad \text{[c-1]}$$

Variable occurs in substitution.

$$Var \in [Subs_1^*, Var \mapsto T, Subs_2^*] = \text{true} \quad \text{[in-1]}$$
$$Var \in \sigma = \text{false} \quad \textbf{otherwise} \quad \text{[in-2]}$$

Retrieve variable associated with a variable in a given substitution.

$$[Var \mapsto T, Subs^*](Var) = T \quad \text{[sv-1]}$$

$$Var_1 \neq Var_2 \Rightarrow [Var_1 \mapsto T, Subs^*](Var_2) = [Subs^*](Var_2) \quad \text{[sv-2]}$$

$$[](Var) = Var \quad \text{[sv-3]}$$

Apply substitution to a term.

$$Var \in \sigma = \text{true} \Rightarrow \quad Var^{\sigma} = \sigma(Var) \quad \text{[as-1]}$$
$$Var \in \sigma = \text{false} \Rightarrow \quad Var^{\sigma} = Var \quad \text{[as-2]}$$
$$\text{is-fun}(T) = \text{true} \Rightarrow \quad T^{\sigma} = \text{repl-arg}(T, \text{arg}(T)^{\sigma}) \quad \text{[as-3]}$$
$$\text{is-non-empty-list}(T) = \text{true} \Rightarrow \quad T^{\sigma} = \text{repl-list}(T, \text{head}(T)^{\sigma}, \text{tail}(T)^{\sigma}) \quad \text{[as-4]}$$
$$\text{is-nil}(T) = \text{true} \Rightarrow \quad T^{\sigma} = T \quad \text{[as-5]}$$

## 2.3 Matching

Matching two terms $T_1$ and $T_2$ yields zero or more substitutions $\{\sigma_1, \sigma_2, ...\}$ such that $T_1^{\sigma_i} \equiv T_2$ $(i = 1, 2, ...)$ holds. If a match yields $\{\}$ (the empty set of substitutions), terms $T_1$ and $T_2$ do not match.

For the current paper it would suffice to introduce a unitary matching function that either succeeds (yielding a single substitution) or fails. The definition given here is more general and can also handle (non-unitary) list matching.

## Module Match

**imports** Substitutions$^{(2.2)}$

**exports**

**sorts** MATCH

**context-free syntax**

| | | |
|---|---|---|
| SUBSTITUTION | → MATCH | |
| MATCH & MATCH | → MATCH | |
| "(" MATCH ")" | → MATCH | {**bracket**} |
| "{" {MATCH ","}* "}" | → MATCH | |
| match(TERM, TERM) | → MATCH | |

**variables**

| | | |
|---|---|---|
| $\mu$ [0-9']* | → MATCH | |
| $\mu$ [0-9']*"*" | → {MATCH ","}* | |
| $\mu$ [0-9']*"+" | → {MATCH ","}+ | |

**equations**

Nested lists of matches may be flattened.

$$\{\mu_1^*, \{\mu_2^*\}, \mu_3^*\} = \{\mu_1^*, \mu_2^*, \mu_3^*\} \quad \text{[lm-1]}$$

Determine the "and" of two matches. The equations for matches consisting of a single substitution are:

$$\frac{Var \in \sigma = \text{true}, \quad \sigma(Var) = T}{[Var \mapsto T, Subs^*] \,\&\, \sigma = [Subs^*] \,\&\, \sigma} \quad \text{[mand1]}$$

$$\frac{Var \in \sigma = \text{true}, \quad \sigma(Var) \neq T}{[Var \mapsto T, Subs^*] \,\&\, \sigma = \{\}} \quad \text{[mand2]}$$

$$\frac{Var \in \sigma_1 = \text{false}, \quad [Subs^*] \,\&\, \sigma_1 = \sigma_2}{[Var \mapsto T, Subs^*] \,\&\, \sigma_1 = [Var \mapsto T] \circ \sigma_2} \quad \text{[mand3]}$$

$$[] \,\&\, \sigma = \sigma \quad \text{[mand4]}$$

$$\sigma \,\&\, [] = \sigma \quad \text{[mand5]}$$

The equations for matches consisting of a list of substitutions are:

$$\mu \,\&\, \{\} = \{\} \quad \text{[mand-6]}$$
$$\{\} \,\&\, \mu = \{\} \quad \text{[mand-7]}$$
$$\{\mu_1\} \,\&\, \mu_2 = \{\mu_1 \,\&\, \mu_2\} \quad \text{[mand-8]}$$
$$\mu_1 \,\&\, \{\mu_2\} = \{\mu_1 \,\&\, \mu_2\} \quad \text{[mand-9]}$$
$$\{\mu_1, \mu_2^*\} \,\&\, \mu_3 = \{\mu_1 \,\&\, \mu_3, \{\mu_2^*\} \,\&\, \mu_3\} \quad \text{[mand-10]}$$
$$\mu_1 \,\&\, \{\mu_2, \mu_3^*\} = \{\mu_1 \,\&\, \mu_2, \mu_1 \,\&\, \{\mu_3^*\}\} \quad \text{[mand-11]}$$

Match terms $T_1$ and $T_2$.

$$\mathsf{match}(\mathit{Var}, T) = \{[\mathit{Var} \mapsto T]\} \quad \text{[match-1]}$$

$$\frac{\mathsf{is\text{-}fun}(T_1) = \mathsf{true},\ \mathsf{is\text{-}fun}(T_2) = \mathsf{true},\ \mathsf{fun}(T_1) = \mathsf{fun}(T_2)}{\mathsf{match}(T_1, T_2) = \mathsf{match}(\mathsf{arg}(T_1), \mathsf{arg}(T_2))} \quad \text{[match-2]}$$

$$\frac{\mathsf{is\text{-}nil}(T_1) = \mathsf{true},\ \mathsf{is\text{-}nil}(T_2) = \mathsf{true}}{\mathsf{match}(T_1, T_2) = \{[]\}} \quad \text{[match-3]}$$

$$\frac{\mathsf{is\text{-}non\text{-}empty\text{-}list}(T_1) = \mathsf{true},\ \mathsf{is\text{-}non\text{-}empty\text{-}list}(T_2) = \mathsf{true}}{\mathsf{match}(T_1, T_2) = \mathsf{match}(\mathsf{head}(T_1), \mathsf{head}(T_2)) \,\&\, \mathsf{match}(\mathsf{tail}(T_1), \mathsf{tail}(T_2))} \quad \text{[match-4]}$$

$$\mathsf{match}(T_1, T_2) = \{\} \quad \textbf{otherwise} \quad \text{[match-5]}$$

### 2.4 Rewrite rules

Rewrite rules (in their simplest form) look like $T_1 \rightarrow T_2$. We introduce the decomposition functions `lhs` and `rhs` to retrieve the sides of a rule and we define term replacement.

## Module Rules

**imports** Terms(2.1) Substitutions(2.2)

**exports**

**sorts** RULE TRS

**context-free syntax**

| | |
|---|---|
| TERM "→" TERM | → RULE |
| lhs(RULE) | → TERM |
| rhs(RULE) | → TERM |

```
"{" {RULE ","}* "}"                    → TRS
repl-term(TERM, SUBSTITUTION, RULE) → TERM
```

**variables**

```
Rule              → RULE
Rule [0-9']*"*"   → {RULE ","}*
R [0-9']*"*"      → {RULE ","}*
TRS [0-9']*       → TRS
```

**equations**

Selector functions for left hand side and right hand side of a rule.

$$\mathsf{lhs}(T_1 \rightarrow T_2) = T_1 \quad \text{[lhs-1]}$$

$$\mathsf{rhs}(T_1 \rightarrow T_2) = T_2 \quad \text{[rhs-1]}$$

Replace a term by an instantiated right-hand side of a rule. This function will be used to replace redexes during rewriting. It is parametrized with a complete rule as opposed to, for instance, only the right hand side of a rule, in order to provide all information that may be relevant for more advanced replacement operations in which the syntactic structure of the complete rule may determine the propagation of certain additional information during replacement. This becomes relevant when extending term replacement for more sophisticated forms of origin tracking.

$$\mathsf{repl\text{-}term}(T, \sigma, \mathit{Rule}) = \mathsf{rhs}(\mathit{Rule})^{\sigma} \quad \textbf{otherwise} \quad \text{[repl-term-1]}$$

## 2.5 Term rewriting systems

Given a term $T$ and a term rewriting system $TRS$, term rewriting amounts to constructing the sequence of terms $T \equiv T_0 \rightarrow T_1 \rightarrow ... \rightarrow T_n$ such that for each step $T_i \rightarrow T_{i+1}$ the following holds:

- in $T_i$ occurs a subterm $T$,
- $TRS$ contains a rule $Rule$,
- $match(T, lhs(Rule)) = \{\sigma_1, ..., \sigma_m\}$ $(m \geq 0)$,
- $T' = rhs(Rule)^{\sigma_j}$ (for some $\sigma_j$, $0 \leq j \leq m$), and
- $T_{i+1} = T_i[T := T']$ (i.e., the occurrence of $T$ in $T_i$ is replaced by $T'$).

In addition, $T_n$ should be a normal form. In the following specification we use the artifact of a "step": either the constant `nostep` if no step is possible or a pair of the form $[Rule, \mu]$, where *Rule* is a rewrite rule and $\mu$ is a match.

Note that the following definitions can be generalized in several directions to take into account:

- the ordering that is used when searching for applicable rules (e.g., random, textual, or specificity of left hand sides);
- reduction strategy (e.g., innermost, outermost, mixed);
- more advanced features in rules (e.g., list variables, conditions).

Here, we will use textual ordering, innermost reduction, and unconditional rules.

## Module TRS

**imports** Match(2.3) Rules(2.4)

**exports**

**sorts** STEP

**context-free syntax**

| | |
|---|---|
| "[" RULE "," MATCH "]" | → STEP |
| nostep | → STEP |
| normalize(TERM, TRS) | → TERM |
| normalize1(TERM, TERM, TRS) | → TERM |
| normalize-args(TERM, TRS) | → TERM |
| find-rule(TERM, TRS, TRS) | → STEP |
| apply-rule(TERM, RULE, MATCH, TRS) | → STEP |

**equations**

Normalize a term to normal form. If it is a non-empty list, normalize the list elements.

$$\frac{\text{is-non-empty-list}(T) = \text{true}}{\text{normalize}(T, \text{TRS}) = \text{repl-list}(T, \text{normalize}(\text{head}(T), \text{TRS}), \text{normalize}(\text{tail}(T), \text{TRS}))} \quad [n1]$$

Else, if there is a matching rule, apply it and continue normalization. Observe that matters related to the rewriting strategy (e.g., normalization of function arguments) are delegated to `find-rule`.

$$\frac{\begin{array}{c}\text{is-non-empty-list}(T) = \text{false},\\ \text{find-rule}(T, \text{TRS}, \text{TRS}) = [Rule, \{\mu_1^*, \sigma, \mu_2^*\}]\end{array}}{\text{normalize}(T, \text{TRS}) = \text{normalize}(\text{repl-term}(T, \sigma, Rule), \text{TRS})} \quad [n2]$$

Otherwise, first normalize the term's arguments and retry normalization.

$$\text{normalize}(T, \text{TRS}) = \text{normalize1}(T, \text{normalize-args}(T, \text{TRS}), \text{TRS}) \quad \textbf{otherwise} \qquad \text{[n3]}$$

Normalization halts if the previous term is equal to the current one; otherwise normalization is continued.

$$\text{normalize1}(T, T, \text{TRS}) = T \qquad \text{[n1-1]}$$

$$\text{normalize1}(T, T', \text{TRS}) = \text{normalize}(T', \text{TRS}) \quad \textbf{otherwise} \qquad \text{[n1-2]}$$

Normalize the arguments of a term.

$$\frac{\text{is-fun}(T) = \text{true}}{\text{normalize-args}(T, \text{TRS}) = \text{repl-arg}(T, \text{normalize}(\text{arg}(T), \text{TRS}))} \qquad \text{[na-1]}$$

$$\frac{\text{is-non-empty-list}(T) = \text{true}}{\text{normalize-args}(T, \text{TRS}) = \text{repl-list}(T, \text{normalize}(\text{head}(T), \text{TRS}), \text{normalize}(\text{tail}(T), \text{TRS}))} \qquad \text{[na-2]}$$

$$\text{normalize-args}(T, \text{TRS}) = T \quad \textbf{otherwise} \qquad \text{[na-3]}$$

Find a rule that can be applied to a given term. The first rule that can be applied is used.

$$\frac{\text{apply-rule}(T, \mathit{Rule}, \{\text{match}(\text{lhs}(\mathit{Rule}), T)\}, \text{TRS}) = [\mathit{Rule}, \mu]}{\text{find-rule}(T, \text{TRS}, \{\mathit{Rule}, \mathit{Rule}^*\}) = [\mathit{Rule}, \mu]} \qquad \text{[fr-1]}$$

$$\frac{\text{apply-rule}(T, \mathit{Rule}, \{\text{match}(\text{lhs}(\mathit{Rule}), T)\}, \text{TRS}) = \text{nostep}}{\text{find-rule}(T, \text{TRS}, \{\mathit{Rule}, \mathit{Rule}^*\}) = \text{find-rule}(T, \text{TRS}, \{\mathit{Rule}^*\})} \qquad \text{[fr-2]}$$

$$\text{find-rule}(T, \text{TRS}, \{\}) = \text{nostep} \qquad \text{[fr-3]}$$

Apply a rule to a term. Given a term $T$ and a rule *Rule*, can *Rule* be used to reduce $T$? Here, we will use a fixed, innermost strategy, but these definitions can easily be extended to cover other strategies as well.

$$\frac{T' = \text{normalize-args}(T, \text{TRS}),\ \text{match}(\text{lhs}(\mathit{Rule}), T') = \{\mu_1^+\}}{\text{apply-rule}(T, \mathit{Rule}, \{\mu^+\}, \text{TRS}) = [\mathit{Rule}, \{\mu_1^+\}]} \qquad \text{[ar-1]}$$

$$\text{apply-rule}(T, \mathit{Rule}, \mu, \text{TRS}) = \text{nostep} \quad \textbf{otherwise} \qquad \text{[ar-3]}$$

### 2.6 An example: list reversal

Consider the reversal of a list of two elements, defined as follows (taken from [vD94]):

```
normalize(
 rev(cons(one(nil) ;cons(two(nil);cons(two(nil);null(nil))))),
 { rev(null(nil))         -> null(nil),
   rev(cons(E;L))         -> append(rev(L);cons(E;null(nil))),
   append(null(nil);L)    -> L,
   append(cons(E;L1);L2) -> cons(E;append(L1;L2))
 }
)
```

Note that our term syntax does not support constants. As a result, all constants have to be written as functions applied to an empty argument list: we have to write `one(nil)`, `two(nil)`, and `null(nil)` instead of `one`, `two` and `null`.

The above initial term will yield the normal form:

```
cons(two(nil) ; cons(two(nil) ; cons(one(nil) ; null(nil))))
```

## 3 Origin tracking

### 3.1 Background

As already explained in the introduction, origin tracking amounts to establishing reverse links between the normal form and the initial term of a term rewriting process. In [vDKT93], all rewriting steps $T_0 \rightarrow T_1 \rightarrow ... \rightarrow T_n$ are taken into account and *backward* relations are established between the normal form $T_n$ and the initial term $T_0$. Four relations are distinguished: *Common Variables*, *Redex-Contractum*, *Context*, and *Common Subterms*.

Here, we introduce a labeling of the initial term and define how these are propagated in the forward direction during rewriting. Labels appearing in the normal form correspond to the desired origin information. The presentation proceeds in three steps. First, we introduce the notion of *origin sets*: sets of labels of the form $\{L_1, L_2, ...\}$. Next, we extend the syntactic form of terms in order to permit the use of origin sets as labels for (sub)terms. Typically, a term `f(a;g(b))` may be labeled as follows: `{lab1}: f(a;g({lab2,lab3}: b))`. Finally, we extend the definitions of the access functions for terms in order to properly preserve origin sets.

It turns out that in this new approach the relations *Common Variables*, and *Context* become implicit since they are taken care of by the labeling. For reasons of simplicity (and space), we will not introduce the *Common Subterms* relation. Using the terminology of [vD94], we therefore restrict the presentation to *primary origins* only.

### 3.2 Specification of origin tracking

## Module Origins

**imports** $\text{TRS}^{(2.5)}$

**exports**

**sorts** ORG-SET

**context-free syntax**

| | |
|---|---|
| "{" {FUN ","}* "}" | → ORG-SET |
| ORG-SET "∪" ORG-SET | → ORG-SET |
| ORG-SET ":" TERM | → TERM |
| org-set(TERM) | → ORG-SET |
| propagate(TERM, ORG-SET) | → TERM |
| collect(TERM) | → ORG-SET |
| collect1(TERM) | → ORG-SET |

**priorities**

TERM ";"TERM → TERM < ORG-SET ":"TERM → TERM

**hiddens**

**variables**

| | |
|---|---|
| $O$ [0-9']* | → ORG-SET |
| $O$ [0-9']*"*" | → {FUN ","}* |

**equations**

Define origin sets as sets of function symbols.

$$\{O_1^*\} \cup \{O_2^*\} = \{O_1^*, O_2^*\} \quad \text{[un-1]}$$

$$\{O_1^*, \mathit{Fun}, O_2^*, \mathit{Fun}, O_3^*\} = \{O_1^*, \mathit{Fun}, O_2^*, O_3^*\} \quad \text{[un-2]}$$

Define the cases that an empty origin set, respectively a multiple origin sets, is associated with a term.

$$\{\} : T = T \quad \text{[empty-org]}$$

$$O_1 : O_2 : T = O_1 \cup O_2 : T \quad \text{[dup-org]}$$

Extend the classification functions on terms.

$$\text{is-fun}(O : T) = \text{is-fun}(T) \quad \text{[is-fun-3]}$$

$$\text{is-non-empty-list}(O : T) = \text{is-non-empty-list}(T) \quad \text{[is-list-3]}$$

$$\text{is-nil}(O : T) = \text{is-nil}(T) \quad \text{[is-nil-3]}$$

Extend the selection functions on terms.

$$\mathsf{fun}(O : T) = \mathsf{fun}(T) \quad \text{[fun-3]}$$

$$\mathsf{arg}(O : T) = \mathsf{arg}(T) \quad \text{[arg-3]}$$

$$\mathsf{head}(O : T) = \mathsf{head}(T) \quad \text{[head-3]}$$

$$\mathsf{tail}(O : T) = \mathsf{tail}(T) \quad \text{[tail-3]}$$

Extend the replacement functions on terms.

$$\mathsf{repl\text{-}arg}(O : T_1, T_2) = O : \mathsf{repl\text{-}arg}(T_1, T_2) \quad \text{[repl-arg-2]}$$

$$\mathsf{repl\text{-}list}(O : T_1, T_2, T_3) = O : \mathsf{repl\text{-}list}(T_1, T_2, T_3) \quad \text{[repl-list-2]}$$

Replace a term $T$ by an instantiated right-hand side of a rule. First, collect the origin sets associated with $T$ and then propagate them to the instantiated right-hand side.

$$\mathsf{repl\text{-}term}(T, \sigma, \mathit{Rule}) = \mathsf{propagate}(\mathsf{rhs}(\mathit{Rule})^{\sigma}, \mathsf{collect}(T)) \quad \text{[repl-term-2]}$$

Retrieve the origin set directly associated with a term. If no origin set is associated with it, return the empty set.

$$\mathsf{org\text{-}set}(O : T) = O \quad \text{[org-set-1]}$$

$$\mathsf{org\text{-}set}(T) = \{\} \quad \textbf{otherwise} \quad \text{[org-set-2]}$$

Determine the origin set of a redex. If an origin set is attached at the outermost level, return it.

$$\mathsf{collect}(O : T) = O \quad \text{[coll-1]}$$

$$\mathsf{collect}(T) = \mathsf{collect1}(T) \quad \textbf{otherwise} \quad \text{[coll-2]}$$

Otherwise, return the origin information attached to embedded function arguments or list elements.

$$\frac{\mathsf{is\text{-}fun}(T) = \mathsf{true}}{\mathsf{collect1}(T) = \mathsf{org\text{-}set}(\mathsf{arg}(T))} \quad \text{[coll1-1]}$$

$$\frac{\mathsf{is\text{-}non\text{-}empty\text{-}list}(T) = \mathsf{true}}{\mathsf{collect1}(T) = \mathsf{org\text{-}set}(\mathsf{head}(T)) \cup \mathsf{org\text{-}set}(\mathsf{tail}(T))} \quad \text{[coll1-2]}$$

$$\mathsf{collect1}(T) = \{\} \quad \textbf{otherwise} \quad \text{[coll1-3]}$$

Propagate a given origin set to the arguments or list elements in a term.

$$\frac{\mathsf{is\text{-}fun}(T) = \mathsf{true}}{\mathsf{propagate}(T, O) = O : \mathsf{org\text{-}set}(T) : \mathsf{fun}(T)(\mathsf{propagate}(\mathsf{arg}(T), O))} \quad \text{[prop-1]}$$

$$\frac{\mathsf{is\text{-}non\text{-}empty\text{-}list}(T) = \mathsf{true}}{\mathsf{propagate}(T, O) = \mathsf{org\text{-}set}(T) : (\mathsf{propagate}(\mathsf{head}(T), O); \mathsf{propagate}(\mathsf{tail}(T), O))} \quad \text{[prop-2]}$$

$$\mathsf{propagate}(T, O) = T \quad \textbf{otherwise} \quad \text{[prop-3]}$$

### 3.3 Example: list reversal with origin sets

Applying the same rewrite rules as in Section 2.6 to a labeled term

```
normalize(
  rev(cons({a}:one(nil) ;cons({b}:two(nil);cons({c}:two(nil);null(nil))))),
 { rev(null(nil))         -> null(nil),
   rev(cons(E;L))         -> append(rev(L);cons(E;null(nil))),
   append(null(nil);L)    -> L,
   append(cons(E;L1);L2) -> cons(E;append(L1;L2))
 }
)
```

will yield an appropriately labeled normal form:

```
cons({c}:two(nil) ; cons({b}:two(nil) ; cons({a}:one(nil) ; null(nil))))
```

By removing all origin sets we obtain the same normal form as yielded by ordinary rewriting. Also observe that with ordinary rewriting the two occurrences of the constant `two(nil)` could not be distinguished. Using origin tracking, the different origins of these two constants are now explicitly indicated in the normal form.

## 4 Discussion

This paper presents origin tracking as a straightforward extension of ordinary term rewriting. Clearly, many issues have not been discussed here (e.g., rewriting strategies, conditional rules). However, the approach has several merits:

- The definition of origin tracking is much simpler than the one given in earlier papers.
- It provides a starting point for studying different origin propagation rules.
- It gives guidance to an implementation of origin tracking.

## Acknowledgements

Appendix B of [vD94] formed the starting point for this exercise. Arie van Deursen commented on a draft of this paper.

## References

[Kli73] P. Klint. Enumerability and termination. Technical report, University of Amsterdam, 1973.

[Kli79] P. Klint. Line numbers made cheap. *Communications of the ACM*, 22:557–559, 1979.

[Kli82] P. Klint. *From Spring to Summer – Design, Definition, and Implementation of Programming Languages for String Manipulation and Pattern Matching.* PhD thesis, Technical University Eindhoven, 1982.

[Kru71] F.E.J. Kruseman Aretz. On the bookkeeping of source-text line numbers during the execution phase of ALGOL 60 programs. In *MC-25 Informatica Symposium*, volume 37 of *Mathematical Centre Tracts*, pages 6.1–6.12, 1971.

[vD94] A. van Deursen. *Executable Language Definitions – Case Studies and Origin Tracking Techniques.* PhD thesis, University of Amsterdam, Programming Research Group, 1994.

[vDKT93] A. van Deursen, P. Klint, and F. Tip. Origin tracking. *Journal of Symbolic Computation*, 15:523–545, 1993.

[WG84] W.M. Waite and G. Goos. *Compiler Construction.* Springer, 1984.

# Mijmeringen over de leesmap

Jan Korst, Wim Verhaegh en Emile Aarts

Wie ooit het voorrecht heeft gehad bij Loes en Frans Kruseman Aretz thuis te zijn ontvangen, bijvoorbeeld om te genieten van hun voortreffelijke kookkunst, zal het niet ontgaan zijn dat in huize Kruseman Aretz het gebruik van de woonkamer verder gaat dan in een doorsnee Nederlandse huiskamer het geval is. Zo staat de recentelijk aangeschafte televisie in de opening van de open haard en worden de muren gebruikt om kranten en tijdschriften langs op te stapelen. Dat laatste is niet verwonderlijk want Frans wil natuurlijk ieder nummer van enige krant of tijdschrift waarop hij is geabonneerd lezen. Dat lukt niet altijd binnen de gestelde tijd, waardoor bewaren tot een later tijdstip geboden is om dan te bezien of hij er toch nog aan toe komt. Dit levert echter een probleem op. Niet alleen dat de stapels veel te hoog worden; er is natuurlijk ook altijd weer de vraag of Frans er nog een abonnement bij zou kunnen nemen zonder zijn leescapaciteit te overschrijden [Kruseman Aretz, 1995].

Wij zijn zo vrij geweest om eens over dit probleem na te denken en graag doen we bij deze gelegenheid verslag over enkele resultaten die wij gevonden hebben.

**Enkele opmerkingen vooraf**

Het soort problemen waarover we het hebben laat zich modelleren als een periodiek scheduling-probleem [Korst, 1992; Verhaegh 1995]. Hiertoe nemen we het volgende aan. Kranten en tijdschriften komen in de vorm van *nummers* van *periodieken*. Frans besteedt per dag $c$ tijdseenheden aan het lezen van periodieken. Dat doet hij zes dagen per week, want 's zondags bakt hij brood en luistert hij naar muziek. De tijd die buiten de zes periodes ligt laten we buiten beschouwing. Dus voor ons bestaat een *leesweek* uit zes *leesdagen* ter lengte $c$. Verder nemen we aan dat de nummers van een periodiek strikt periodiek arriveren, d.w.z. met constante tussenpozen. De lengte van deze tussenposen noemen we de *periode* van een periodiek en de tijd nodig om een nummer te lezen noemen we de *leesduur*. We gaan er verder van uit dat Frans slechts één nummer tegelijkertijd leest. We spreken van *opstapeling* als het gemiddeld aantal nummers dat per tijdseenheid arriveert groter is dan het gemiddeld aantal nummers dat Frans per tijdseenheid leest. Tot slot spreken we over lezen *met onderbreking* als Frans een nummer tijdens het lezen even weglegt om eerst een nummer van een ander periodiek te lezen. Als Frans een nummer uitleest zodra hij er eenmaal aan begonnen is spreken we over lezen *zonder onderbreking*.

Dit leidt tot de volgende formulering.

**Probleem 1** (P1)

Gegeven: een verzameling $T$ van periodieken en voor ieder periodiek $t$ een periode $p(t) \in$

$\mathbb{N}$ en een leesduur $e(t) \in \mathbb{N}$. Gevraagd: kan Frans al deze periodieken lezen zonder opstapeling?

**Stelling 1** [Coffman, 1976]
Een instantie van P1 is een ja-instantie dan en slechts dan als

$$\sum_{t \in T} e(t)/p(t) \leq 1.$$

Stelling 1 is waar zowel voor het geval dat Frans leest met onderbreking als zonder onderbreking. Omdat de conditie van Stelling 1 polynomiaal berekenbaar is, noemen we het probeem P-eenvoudig. Meer formeel: het probleem behoort tot de complexiteitsklasse $\mathcal{P}$.

**Aanname 1** (completering)
Frans heeft een nummer van een periodiek gelezen voordat het volgende nummer arriveert.

**Probleem 2** (P2)
Probleem P1 met onderbreking en completering.

**Stelling 2** [Liu & Layland, 1973]
Een instantie van P2 is een ja-instantie dan en slechts dan als

$$\sum_{t \in T} e(t)/p(t) \leq 1,$$

en dus is $P2$ P-eenvoudig.

**Aanname 2** (priorering)
Elk periodiek heeft een vaste, vooraf zelf te bepalen, prioriteit. Indien een nummer arriveert van een periodiek dat een hogere prioriteit heeft dan het periodiek waarvan Frans op dat moment een nummer leest, dan onderbreekt Frans het lezen van dat nummer en leest hij eerst het zojuist gearriveerde nummer. Het zal duidelijk zijn dat priorering onderbreking impliceert.

**Probleem 3** (P3)
Probleem P1 met completering en priorering.

**Stelling 3** [Liu & Layland, 1973]
Een instantie van P3 is een ja-instantie als

$$\sum_{t \in T} e(t)/p(t) \leq n(2^{1/n} - 1),$$

met $n = |T|$ en dus is $P3$ P-eenvoudig.

Voor $n = 1$ reduceert de conditie van Stelling 3 tot de conditie van Stelling 2. Voor $n \to \infty$ geldt dat $n(2^{1/n} - 1) \to \ln 2 \approx 0,693$. Dus als het aantal periodieken in de leesmap toeneemt neemt de effectieve leescapaciteit af. Dat komt doordat de vaste volgorde de beschikbare ruimte beperkt.

**Aanname 3** (gretigheid)
Zodra er een nummer arriveert start Frans met lezen en hij stopt niet met lezen zolang er nog ongelezen nummers liggen die in de tussenliggende tijd zijn aangekomen.

Bij P1, P2 en P3 maakt het niet uit of de aankomsttijden van de nummers van een periodiek van tevoren bekend zijn of niet. Gretigheid in combinatie met de voorwaarde dat er zonder onderbreking gelezen moet worden introduceert echter wel een verschil tussen deze twee situaties, zoals volgt uit de volgende stelling.

**Probleem 4** (P4)
Probleem P1 zonder onderbreking, met completering, gretigheid en onbekende aankomsttijden.

**Stelling 4** [Jeffay, Stanat & Martel, 1991]
Neem aan dat de periodes zijn geordend, d.w.z. $p(1) \leq p(2) \leq ... \leq p(n)$ voor een gegeven verzameling periodieken $T = \{1, \ldots, n\}$, hetgeen kan worden aangenomen zonder verlies van algemeenheid. Een instantie van P4 is een ja-instantie dan en slechts dan als

1. $\sum_{j=1}^{n} e(j)/p(j) \leq 1$ en
2. voor alle $i, 0 < i < n$, en voor alle $L, p(1) < L < p(i)$,
   $L \geq e(i) + \sum_{j=1}^{i-1} \lfloor L - 1/p(j) \rfloor c(j)$,

en dus is $P4$ P-eenvoudig.

**Probleem 5** (P5)
Probleem P1 met completering en gretigheid, zonder onderbreking en bekende aankomsttijden.

**Stelling 5** [Jeffay, Stanat & Martel, 1991]
P5 is NP-lastig in de sterke zin.

Uit Stelling 5 volgt dat problemen lastig kunnen worden als je te veel weet.

**Aanname 4** (striktheid)
Frans leest de nummers van een periodiek strikt periodiek, d.w.z. Frans leest ieder periodiek $t$ met een periode $p(t)$. Zodoende kan hij een beknopt leesrooster maken waarbij elk periodiek op vaste tijdstippen in de leesweek wordt gelezen.

**Probleem 6** (P6)
Probleem P1 zonder onderbreking, met completering, striktheid en bekende aankomsttijden.

**Stelling 6** [Korst, 1992]
P6 is NP-volledig in de sterke zin.

Uit Stelling 6 volgt dat te veel regelmaat maar lastig kan zijn. Dit is echter niet het geval als de lezer gretig is, hetgeen volgt uit het volgende resultaat.

**Probleem 7** (P7)
Probleem P1 zonder onderbreking, met completering, gretigheid, striktheid en bekende aankomsttijden.

**Stelling 7** [Korst, 1992]
Een instantie van P7 is een ja-instantie dan en slechts dan als

$$(a(t) - a(s)) \bmod \text{ggd}(p(t), p(s)) \geq e(s),$$

voor ieder tweetal periodieken $t, s \in T$, $t \neq s$, met aankomsttijden $a(t)$ en $a(s)$, en dus is P7 P-eenvoudig.

Het zal duidelijk zijn dat nog lang niet alle leesmapproblemen aan de orde zijn geweest. Zo hebben het nog niet gehad over de problemen die ontstaan als Frans toch ook op zondag wil lezen of het geval waarbij een periodiek slechts een bepaalde tijd per jaar uitkomt, b.v. maandelijks maar niet in juli en augustus. In deze gevallen worden de problemen meer-dimensionaal periodiek en veel van de antwoorden op gelijksoortige vragen als we bovenstaand besproken hebben staan in het proefschrift van Wim Verhaegh [1995].

Frans, voor nu hebben we genoeg gemijmerd; we wensen je nog lang een goede gezondheid en veel plezierige leesuren toe.

## Bibliografie

COFFMAN, E.G. JR. [1976], *Computer and Job-Shop Scheduling Theory*, Wiley, New York.

JEFFAY, K., D.F. STANAT, AND C.U. MARTEL [1991], On non-preemptive scheduling of periodic and sporadic tasks, *Proceedings of the 1991 IEEE Real-Time Systems Symposium*, pp. 129–139.

KORST, J.H.M. [1992], *Periodic Multiprocessor Scheduling*, Ph.D. Thesis, Eindhoven University of Technology.

KRUSEMAN ARETZ, F.E.J. [1995], Personal communication (tijdens een wandeling rond de vijver).

LIU, C.L. AND J.W. LAYLAND [1973], Scheduling algorithms for multiprogramming in a hard-real-time environment, *Journal of the Association for Computing Machinery* **20**, 46–61.

VERHAEGH, W.F.J. [1995], *Multidimensional Periodic Scheduling*, Ph.D. Thesis, Eindhoven University of Technology.

# Statistical selection: a way of thinking !

Paul van der Laan

'An approximate answer to the right problem is worth a good deal more than an exact answer to an approximate problem.'
*John Tukey*

'Although this may seem a paradox, all exact science is dominated by the idea of approximation.'
*Bertrand Russell*

'If you think that statistical selection is of no importance, think again!'

## Summary

Statistical selection of the best population is discussed in general terms and the principles of statistical selection procedures are presented. Advantages and disadvantages of Subset Selection, one of the main approaches, are indicated. The selection of an almost best population is considered and compared with the selection of the best one from an application point of view.

## 1 Introduction

In practice we are often confronted with the problem of selection. For instance in the field of testing varieties, testing drugs and choosing the optimal production process statistical selection is often an essential feature.
For all kinds of selection problems a quantitative methodology of selection is needed. The problem of statistical selection concerning several populations (varieties, treatments, drugs, processes, etc.) has long been a concern for statisticians. Let us consider the problem of selecting the best population from a number $k$ (integer $k \geq 2$) of populations. The best population is defined as the population with the largest expectation. If there are more than one contenders for the best because there are ties, it is assumed that one of these is appropriately tagged.
Furtheron, we assume that the experiment has been designed as a completely randomized design with $n$ observations for each population. Statistical selection procedures can

possibly help us to improve our selection process. A short description of the basic approaches will be given in Section 2. There are many goals to consider and we shall do that in Section 3. The advantages and disadvantages of a special approach of statistical selection, namely Subset Selection, will be mentioned in Section 4.

It is important that selection of the best population can be guaranteed in some sense or another. The principle of a statistical selection procedure is that the probability of an error, an incorrect selection, is under control. It is possible to compare, in a certain sense, the situation with that of statistical testing of a hypothesis H. The rejection of H, while in reality H is true, is indicated by 'an error of the first kind'. The characteristic feauture of a statistical test is that the probability of an error of the first kind is under control, say smaller than or equal to 0.05. A second aspect is the probability that H will be rejected if in reality H is false. The probability that H will be rejected, if it is false, is called the power (of the test considered). A requirement may be that in a certain situation the power must be larger than 0.90, say. It is possible to compare the probability of a correct selection using a statistical selection procedure, with the power of a statistical test. One can say that this probability is as important as the power of a test. To be sure, or almost sure, that we don't miss the best population, the probability of correct selection of the best population has to be taken into account. Of course, from a practical point of view, one may be satisfied with an almost best population instead of the best population. But also in that situation we want to have a confidence requirement that the best or an almost best population will be selected. We return to this point in Section 5. Finally, we conclude with some comments in Section 6.

## 2 Statistical selection

The main approaches in handling with selection of the best population are the Subset Selection approach (for details we refer to *Gupta* and *Panchapakesan*; 1979) and the Indifference Zone approach (for details we refer to *Gibbons*, *Olkin* and *Sobel*; 1977, *Gupta* and *Panchapakesan*, 1979). These two basic approaches will be outlined in a few words. Assume $k$ ($k \geq 2$) independent Normal random variables $X_1, \ldots, X_k$ are given. These variables are associated with the $k$ populations indicated by $V_1, \ldots, V_k$ and may be sample means. The assumed Normal distributions have common known variance $\sigma^2$ and unknown means $\mu_1, \ldots, \mu_k$. The assumption of a common known variance has been made for simplicity. The goal is to select the population with mean $\mu_{[k]}$, where $\mu_{[1]} \leq \ldots \leq \mu_{[k]}$ denote the ordered values of $\mu_1, \ldots, \mu_k$. Let *CS* denote correct selection.
The Subset Selection procedure selects a subset, nonempty and as small as possible, with the probability requirement that the probability of a *CS* is at least $P^*$, with $1/k < P^* < 1$. A *CS* means in this context that the best population is an element of the selected subset. The Subset Selection rule is defined as follows:

Select population $V_i$ in the subset if and only if

$$X_i \geq \max_{1 \leq j \leq k} X_j - d\sigma/\sqrt{n}$$

$(i = 1, \ldots, k)$, where $X_i$ is the sample mean of the $i$-th sample. The selection constant $d$ $(d > 0)$ must be determined in such a way that $P(CS) \geq P^*$, with $1/k < P^* < 1$, for all possible parameter configurations. Tables with values of the selection constant $d$ can be found in, for instance, *Gibbons*, *Olkin* and *Sobel* (1977).

The second approach is the so-called Indifference Zone approach. The goal is to indicate or select the best population. The procedure is to select that population that resulted in the largest sample mean. The probability requirement is that the probability of a *CS* is at least $P^*$, with $1/k < P^* < 1$, whenever the best population is at least $\delta^*$ $(> 0)$ away from the second best. In this context *CS* means that the best population with mean $\mu_{[k]}$ produces the largest sample mean and consequently it is also indicated as the best population. The minimal probability $P^*$ can only be guaranteed if the common sample size $n$ is large enough. From the probability requirement the next quantity

$$\tau = \delta^* \sqrt{n}/\sigma$$

can be computed and can be found in, for instance, *Gibbons*, *Olkin* and *Sobel* (1977). Notice that the following holds

$$n = (\tau\sigma/\delta^*)^2 \, .$$

With this chosen (minimal) value of $n$ it can be guaranteed with minimal probability $P^*$ that the selected population is less than $\delta^*$ away from the best one.

## 3 Selection goals

In this section we shall describe a number of goals which may be relevant from a practical point of view. We mention the following goals:

i. Selecting the best population.

ii. Selecting the $t$ best populations, where integer $t$ is larger than or equal to 2. We can do this with or without ordering of the populations. In the first case we indicate a population as the best one, another population as the second best, etc. In the second case we produce a collection of $t$ populations without ranking them.

iii. Selecting a subset that contains only good populations.

iv. Elimination of inferior (or bad) populations.

v. Ordering (ranking) all populations from worst to best.

vi. Selecting a collection of populations, that will contain at least the best population.

vii. Selecting a collection of populations which will contain at least the $t$ ($t \geq 2$) best populations.

viii. Selecting a fixed number of populations that includes the $t$ best populations.

ix. Selecting a random number of populations such that all populations better than a standard population are included in the selected subset.

x. Selecting a subset whose size is smaller than or equal to $m$ ($1 \leq m < k$) and which will include at least one good population.

xi. Selecting a subset that excludes all the non-$t$-best populations, where a population is strictly non-$t$-best if the distance (in mean) for this population to the $t$-th best population is larger than or equal to a certain amount.

In the literature different generalizations and modifications have been proposed. We refer to *Gupta* and *Panchapakesan* (1979) for references.

## 4 Subset Selection

Subset Selection is in a certain sense a flexible form of selection, because the number of replications has not to be determined in advance. After the experiment has been carried out, the selection can be prosecuted. The influence of the number of replications can be conducted from the (expected) size of the subset. A relatively large subset means, apart form random fluctuations, that the number of replications is small or the means of the populations are close together, or both. If a correct selection $CS$ is defined as the event that the best population is in the subset then the probability on $CS$ can be compared with the power of a test. Both characteristics indicate the probability on a correct decision while the populations may be (or are) different. Whereas Subset Selection can be used as a screening procedure, the Indifference Zone approach produces, in a certain sense, a more precise result. For the last method indicates the best population, at least we hope so. A condition is that a minimal number of observations have been done. Moreover, when the variance is unknown then the Subset Selection can be carried out using a different value for the selection constant $d$, but the Indifference Zone method requires a procedure which consists of two stepts. The first step is necessary to estimate the unknown variance in order to make it possible to derive a value for the required common sample size for the samples of the second step.

## 5 An almost best population

The requirement to select the best population may be a strong one if the best population is not far away from the other populations. Let us assume that the best population and the second best are near each other. More accurately: the expectations of these two populations are close together, say on a distance less than $\epsilon$ , where $\epsilon > 0$, but relatively small. In such a situation it is often not of practical interest whether one selects the best population or the next best. In this situation the next best population can be indicated as an almost best population. The same idea can be extracted from the Indifference Zone for which the distance between the best population and the next best is smaller than $\delta^*$ $(> 0)$. Not every difference in expectation is important. In many real world problems it is of interest to see whether the best or an almost best population can be selected. This idea leads to the consideration that one may generalize the selection goal to a selection of an almost best population. A consequence of this generalization is that the least favourable configuration becomes more difficult. Not an essential disadvantage using computers. The goal is to select a small but nonempty subset such that the selected subset will contain the best population or an almost best one with a high probability. In general, the generalization to selecting an almost best population will result in subsets of smaller expected size. The concept of $\epsilon$-best started, in fact, with the notion of delta-correct ranking introduced by *Fabian* (1962) and *Lehmann* (1963). For the location problem, *Van der Laan* (1992) studied the use of an $\epsilon$-best population in Subset Selection with as goal the selection of a non-empty subset which includes at least one $\epsilon$-best population with a minimal guaranteed probability $P^*$ $(1/k < P^* < 1)$.

## 6 Some concluding remarks

During decades of years we are used to apply statistical tests, like analysis of variance tests, to problems that are real selection problems. In a number of applications we want, ultimately, to find the best population, where best is defined in a less or more complicated manner. We think it is important to investigate the possibilities to use statistical selection procedures for certain problems in the field of application. The first thing we need is to formulate adequately the problem. If a problem is a selection problem, then formulation as a selection problem has to be considered. Following this statement an exact formulation as a selection problem is worthwile and then an analysis (exact or approximate) is required. Not for all designs of experiments this problem has been solved. For a number of designs a simulation is feasible in order to find an accurate estimate of the selection constant required for the prosecution of the selection procedure.
The change from analysis of variance type techniques to selection type techniques is presumably not simple. It is a change of 'a way of thinking'. We notice that sofar a lot of theoretical problems, not all problems, have been solved. We refer to *Dudewicz* (1980), *Gibbons*, *Olkin* and *Sobel* (1977, 1979), *Gupta* (1977) and *Van der Laan* (1987) for an introduction to statistical selection.

## References

*Dudewicz, E.* (1980).
Ranking (ordering) and selection: An overview of how to select the best. Technometrics 22, 113-119.

*Fabian, V.* (1962).
On multiple decision methods for ranking population means. Ann. Math. Statist. 33, 248-254.

*Gibbons, J.D., I. Olkin* and *M. Sobel* (1977).
Selecting and Ordering Populations: A New Statistical Methodology. Wiley, New York.

*Gibbons, J.D., I. Olkin* and *M. Sobel* (1979).
An introduction to ranking and selection. The American Statistician 33, 185-195.

*Gupta, S.S.* (1977).
Selection and ranking procedures: a brief introduction. Commun. Statist. Theor. Meth. A6, 993-1001.

*Gupta, S.S.* and *S. Panchapakesan* (1979).
Multiple Decision Procedures. Wiley, New York.

*Lehmann, E.L.* (1963).
A class of selection procedures based on rank. Math. Ann. 150, 268-275.

*Van der Laan, P.* (1987).
Some remarks on ranking and selection of treatments. Cultivar Testing Bulletin 12, 203-218.

*Van der Laan, P.* (1992).
Subset selection of an almost best treatment. Biometrical Journal 34, 647-656.

# OOTIsche cijfers

Marloes van Lierop

**Abstract**

Presentatie van cijfermatige gegevens over de OOTI's, de AIO's van de ontwerpersopleiding Technische Informatica.

## 1 Inleiding.

De Ontwerpersopleiding Technische Informatica is van start gegaan op 1 september 1988. Van meet af aan heeft prof. Kruseman Aretz deel uitgemaakt van de Opleidingsgroep, en is daardoor mede verantwoordelijk voor de vorm en inhoud van deze suksesvolle opleiding.

Gezien zijn meermalen getoonde interesse in kwalitatieve en kwantitatieve gegevens over de OOTI's, presenteer ik hier cijfers over instroom en uitstroom, de gevolgde vooropleiding, de eindprojecten, en de werkkring van de gediplomeerden, samen met enkele voorzichtige conclusies. In de laatste paragraaf waag ik mij aan wat voorspellingen.

De cijfers zijn gebaseerd op de stand van zaken eind augustus 1995.

## 2 In- en uitstroom.

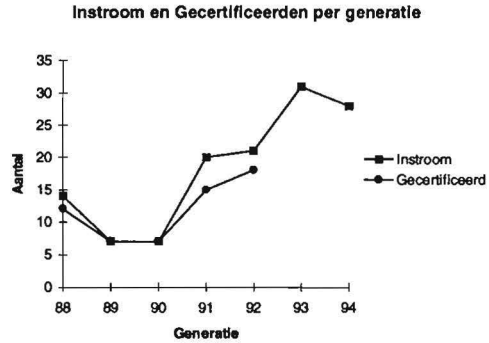
In de beginjaren van de opleiding waren er grofweg drie mogelijke tijdstippen (plus of min 1 maand) in een studiejaar waarop men met de opleiding kon beginnen: 1 september, 1 december, en 1 maart. Wegens roostertechnische problemen is in de loop van de tijd 1 maart komen te vervallen. Dit verklaart mijn definitie van een generatie: met generatie N wordt bedoeld de groep OOTI's die de opleiding begonnen zijn tussen 1 augustus in jaar N en 1 augustus in jaar N+1.

In tabel 1 staat per generatie aangegeven de omvang van de instroom, de uitval, en de gediplomeerde uitstroom. Onder uitval rekenen wij die OOTI's die, al dan niet voortijdig, de opleiding verlaten hebben zonder diploma. Daar de opleiding 2 jaar duurt, zijn van generatie 93 en 94 de opgenomen uitvalcijfers te beschouwen als voorlopige cijfers; zij geven de stand van zaken weer van eind augustus 1995.

| Generatie | 88 | 89 | 90 | 91 | 92 | 93 | 94 | Totaal |
|---|---|---|---|---|---|---|---|---|
| Instroom | 14 | 7 | 7 | 20 | 21 | 31 | 28 | **128** |
| Uitval | 2 | 0 | 0 | 5 | 3 | 1 | 2 | **13** |
| Gecertificeerd | 12 | 7 | 7 | 15 | 18 | | | **59** |

Tabel 1: Omvang in- en uitstroom, per generatie en totaal

In figuur 1 zijn de in- en uitstroomgegevens grafisch weergegeven. Hierin valt vooral op de sterke stijging van de instroom in 1991 en 1993.




Figuur 1

De totale uitval (13 van de 128) bedraagt iets meer dan 10%, en mag laag genoemd worden. Wel moet opgemerkt worden dat er vooraf selectie plaats vindt doordat men moet solliciteren voor een plaats in de opleiding. De redenen voor vertrek zijn zeer divers: naast alsnog gebleken ongeschiktheid, hetzij qua niveau (3 maal), hetzij qua motivatie (3 gevallen), is het aanbod van een aantrekkelijke baan (5 maal) ook een reden om te stoppen. Gelukkig is dat tot nu toe enkel bij net gestarte OOTI's voorgekomen. De gemiddelde verblijfsduur van de hele groep van 13 uitvallers bedraagt ruim 5 maanden. Slechts twee personen hebben de twee jaar volgemaakt om alsnog zonder diploma te vertrekken.

In paragraaf 3 wordt van de uitvallers de vooropleiding vergeleken met die van de gediplomeerden.

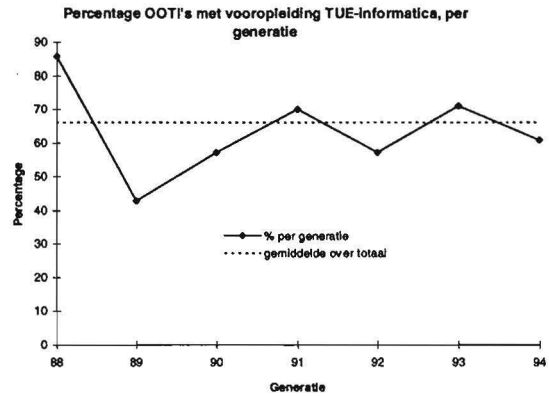
## 3 Vooropleiding.

De informatica-ingenieurs van de TUE hebben altijd een substantieel deel (gemiddeld 2/3!) uitgemaakt van de instroom. Het resterende part is onderverdeeld in informatici van andere universiteiten en niet-informatici. Deze twee groepen zijn over het totaal van de 7 gemeten generaties ongeveer even groot. De cijfers zijn te vinden in tabel 2.

| Generatie | 88 | 89 | 90 | 91 | 92 | 93 | 94 | Totaal # | Totaal % |
|---|---|---|---|---|---|---|---|---|---|
| Informatica, TUE | 12 | 3 | 4 | 14 | 12 | 22 | 17 | **84** | **66** |
| Informatica,elders | 2 | 3 | 2 | 5 | 4 | 3 | 4 | **23** | **18** |
| Anders | 0 | 1 | 1 | 1 | 5 | 6 | 7 | **21** | **16** |
| **Totaal** | **14** | **7** | **7** | **20** | **21** | **31** | **28** | **128** | **100** |

Tabel 2: Vooropleiding OOTI's, per generatie en totaal

Voor de TUE-informatici zijn de fluctuaties over de generaties weergegeven in Figuur 2. (Een dergelijke grafiek is voor de beide andere categorieen niet zinvol omdat het daar over te kleine aantallen gaat.)




Figuur 2

De vooropleiding van degenen die geen informatica hebben gestudeerd varieert van harde technische opleidingen (electrotechniek, werktuigbouwkunde) via meer aan informatica verwante opleidingen (b.v. informatietechniek) tot verrassende opleidingen als biologie. Alle bij OOTI's voorkomende vooropleidingen staan opgesomd in tabel 3. Hieruit blijkt duidelijk dat Informatietechniek, hoewel een laatkomer, veruit de meeste OOTI's levert, met Electrotechniek als goede tweede.

| Generatie | 88 | 89 | 90 | 91 | 92 | 93 | 94 | Totaal |
|---|---|---|---|---|---|---|---|---|
| Bestuurlijke informatica | | 1 | | | | | | 1 |
| Natuurkunde | | | 1 | | 1 | | | 2 |
| Medische informatica | | | | 1 | | | | 1 |
| Wiskunde | | | | | 1 | | | 1 |
| Informatietechniek | | | | | 2 | 4 | 2 | 8 |
| Electrotechniek | | | | | 1 | | 3 | 4 |
| Biologie | | | | | | 1 | | 1 |
| Werktuigbouwkunde | | | | | | 1 | | 1 |
| Transport en communicatie | | | | | | | 1 | 1 |
| Moleculaire wetenschappen | | | | | | | 1 | 1 |
| **Totaal** | **0** | **1** | **1** | **1** | **5** | **6** | **7** | **21** |

Tabel 3: Vooropleiding anders dan informatica, per generatie en totaal

Van de totale instroom (128) komen 96 OOTI's (75%) van de TUE. Van de overigen kan in tabel 4 gevonden worden waar zij zoal hun vooropleiding genoten.

| Generatie | 88 | 89 | 90 | 91 | 92 | 93 | 94 | Totaal |
|---|---|---|---|---|---|---|---|---|
| KUN | 1 | | 1 | 2 | 1 | 1 | 1 | 7 |
| RUU | 1 | | | | | 1 | | 2 |
| TU Twente | | 1 | | 1 | | 1 | 1 | 4 |
| EUR | | 1 | | | | | | 1 |
| RUG | | 2 | | 1 | 3 | | 1 | 7 |
| UvA | | | 1 | | | | | 1 |
| RUL | | | | 1 | | | | 1 |
| TUD | | | | 1 | | 1 | | 2 |
| LUW | | | | | | | 1 | 1 |
| Buitenlandse opl. | | | | | 3 | | 3 | 6 |
| **Totaal per generatie** | **2** | **4** | **2** | **6** | **7** | **4** | **7** | **32** |

Tabel 4: Universiteit van herkomst (anders dan TUE), per generatie en totaal

Een voorzichtige verklaring voor onze populariteit bij Nijmeegse studenten zou de kleine geografische afstand tussen TUE en KUN kunnen zijn. Voor Groningen geldt dat de informatica-opleiding daar veel overeenkomsten heeft met die aan de TUE; de aansluiting met de ontwerpersopleiding is daarom prima.

Ook in tabel 4 is te lezen dat de eerste buitenlanders hun intrede hebben gedaan in 1992. Toen zijn ook voor het eerst de OOTI-colleges in het Engels gegeven. Dit heeft weinig problemen gegeven, integendeel zelfs want door veel OOTI's wordt het als een pluspunt gezien dat Engels min of meer de voertaal is.

Om te onderzoeken of de vooropleiding een factor van betekenis is bij de slaagkans voor de ontwerpersopleiding, heb ik per categorie (informatica TUE, informatica elders, overig) gekeken naar de totale instroom en de uitval. De cijfers zijn weergegeven in tabel 5.

| Vooropleiding | Instroom per categorie | Uitval per categorie | Percentage uitval per categorie |
|---|---|---|---|
| Informatica, TUE | 84 | 8 | 9.5 |
| Informatica, elders | 23 | 3 | 13.0 |
| overig | 21 | 2 | 9.5 |

Tabel 5: Instroom en uitval naar vooropleiding

Uit de laatste kolom van tabel 5 kan geconcludeerd worden dat OOTI's met een informatica-opleiding buiten de TUE een iets grotere kans lopen het diploma niet te halen dan de beide andere categorieen. Maar de afwijkingen ten opzichte van het uitvalpercentage over de gehele instroom (10%) zijn zo gering dat men ook kan stellen dat vooropleiding geen goede indicator is voor slaagkans.

## 4 Eindprojecten.

Het laatste onderdeel van de opleiding betreft het uitvoeren van een ontwerpproject. Zo'n project duurt negen maanden, en kan plaats vinden binnen de TUE of bij een bedrijf. De voorkeur van de meeste OOTI's gaat uit naar een bedrijf omdat men daar meer praktijkervaring op kan doen dan binnen de universiteit. Echter, in de beginjaren werden de meeste projecten uitgevoerd bij de vakgroep Informatica zelf. Dit was gebaseerd op de gedachte dat de grote werkbelasting voor het opzetten van de opleiding enigszins gecompenseerd moest worden door de inzet van OOTI's op eigen projecten.

In de loop der jaren werd duidelijk dat het via het IAM toegewezen budget voor de opleiding, gebaseerd op het aantal OOTI's in opleiding, ruim voldoende bleek om de onderwijsinspanning te bekostigen. Daardoor kon aan de behoefte van de OOTI's om externe eindprojecten te doen tegemoet gekomen worden. Bovendien bleken de extern uitgevoerde projecten dermate suksesvol dat gestart werd met het vragen van vergoedingen voor OOTI's op externe eindprojecten. Op die manier kan een aanzienlijk derde geldstroombedrag binnen gehaald worden.

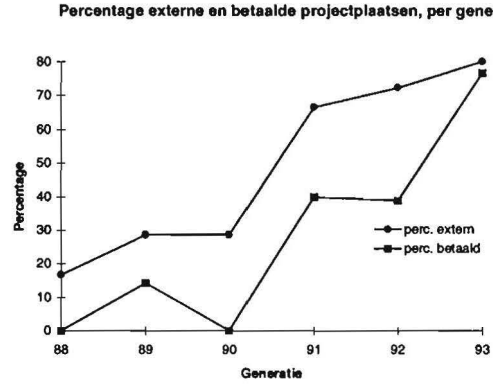
In tabel 6 staat per generatie weergegeven waar het eindproject werd uitgevoerd.

| Locatie | 88 | 89 | 90 | 91 | 92 | 93 |
|---|---|---|---|---|---|---|
| TUE, vakgroep informatica | 10 | 3 | 2 | 3 | 5 | 2 |
| TUE, elders | 0 | 2 | 3 | 2 | 0 | 4 |
| Extern | 2 | 2 | 2 | 10 | 13 | 24 |
| **Totaal** | **12** | **7** | **7** | **15** | **18** | **30** |

Tabel 6: Locatie eindprojecten, per generatie gecertificeerde OOTI's

De groei van het percentage externe projectplaatsen over de jaren is ook goed te zien in Figuur 3. Daar is eveneens in opgenomen de grafiek van het percentage betaalde

eindprojectplaatsen. Opgemerkt moet worden dat niet alle betaalde projecten extern uitgevoerd worden: er is een enkel project uit de categorie "TUE, elders" waarvoor betaald wordt. Dat betekent dat in de toekomst de onderste lijn de bovenste kan gaan snijden!




Figuur 3

## 5 Werkkring.

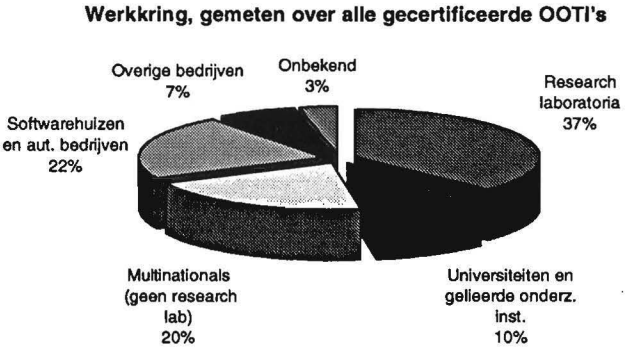
Dat de opleiding in een behoefte van het bedrijfsleven voorziet moge blijken uit het feit dat gediplomeerde OOTI's zonder veel moeite een baan vinden, ook in tijden dat er niet veel vraag is naar informatici. Een neutraal meetpunt hiervoor is de hoeveelheid wachtgeld die de universiteit moet betalen aan de afgestudeerden van een opleiding. Van alle ontwerpersopleidingen aan de TUE blijkt Technische Informatica het minst te kosten aan wachtgeld, terwijl het de op een na grootste opleiding is.

De eerste generaties gediplomeerde OOTI's kwamen vooral in de grote research laboratoria terecht. Bij de generaties 91 en 92 (die dus in 1993 resp. '94 op de markt kwamen) veranderde dat en kwam het Midden- en Kleinbedrijf meer in beeld. In tabel 7 staat een grove classificatie naar werkkring per generatie.

| Werkgever | 88 | 89 | 90 | 91 | 92 | Totaal |
|---|---|---|---|---|---|---|
| Research laboratoria | 6 | 5 | 4 | 4 | 3 | 22 |
| Universiteiten en gelieerde onderz.inst. | 2 | | | 1 | 3 | 6 |
| Multinationals (geen research lab) | 2 | | | 5 | 5 | 12 |
| Softwarehuizen en aut. bedrijven | 2 | 1 | 3 | 5 | 2 | 13 |
| Overige bedrijven | | 1 | | | 3 | 4 |
| Onbekend | | | | | 2 | 2 |
| **Totaal** | 12 | 7 | 7 | 15 | 18 | 59 |

Tabel 7: Classificatie werkgever, per generatie en totaal

De cijfers uit de laatste kolom van tabel 7, betreffende de totale groep van gediplomeerde OOTI's, staan als percentages grafisch weergegeven in figuur 4.




Figuur 4

Opvallend is toch het grote deel dat in een onderzoeksomgeving terecht komt. Niettemin zijn er maar heel weinig OOTI's die in hun functie hoofdzakelijk bezig zijn met onderzoek.

Om een idee te geven om wat voor bedrijven het gaat, noem ik de vier grootste afnemers van de 57 gediplomeerde OOTI's waarvan de werkgever bekend is: Philips (28%), KPN Research (19%), NLR en FEL/TNO (ieder 9%).

## 6 Voorspellingen.

### Instroom

Ten aanzien van de instroom zijn er enkele omstandigheden die een verlagende werking zullen hebben: de dalende uitstroom van de eerste fase, de toenemende vraag naar informatici waardoor voor hen de noodzaak van profilering met een vervolgopleiding afneemt, en het opheffen van de militaire dienstplicht waardoor de voorheen bestaande mogelijkheid om 2 jaar uitstel te krijgen geen reden meer is om de opleiding te gaan doen.
Echter, de sterke markt voor informatici kan voor afgestudeerden uit andere richtingen juist een reden zijn om een informatica-vervolgopleiding in overweging te nemen.

De verlenging van de studieduur voor technische studies naar vijf jaar zal op de lange duur wellicht de status van de ontwerpersopleidingen wijzigen, en wie weet zelfs hun bestaansrecht.
Op korte termijn valt een positief effect te verwachten: voor afgestudeerden van niet-technische universiteiten is het volgen van een ontwerpersopleiding een manier om zich te profileren ten opzichte van de ingenieurs met hun vijfjarig programma.

Kortom, als OOTI erin slaagt de aandacht op zich te vestigen van informatica-doctorandi en afgestudeerden uit andere disciplines, dan kan de terugval in instroom beperkt blijven. De percentages uit tabel 4 betreffende de vooropleiding zullen dan uiteraard grondig wijzigen: het TUE-informatica aandeel zal verminderen ten gunste van beide andere categorieen. En die ontwikkeling kan alleen maar toegejuicht worden.

### Eindprojecten

De groei in externe en betaalde eindprojecten zoals die zichtbaar is in figuur 3 zal nog wel even doorzetten. Het streven is om het percentage betaalde projecten op minstens 90% te krijgen, en dat lijkt mij makkelijk haalbaar. De hoogte van de door het bedrijf te betalen vergoeding wordt door het marktmechanisme bepaald, en zal voorlopig eerder stijgen dan dalen. Nadrukkelijk moet gezegd worden dat bij toewijzing van een project allereerst naar geschiktheid gekeken wordt, en dat het financiele aspect hieraan ondergeschikt is.

### Werkkring

Van de eerstvolgende generatie die op de markt komt (generatie 93) is nu al bekend dat een aanzienlijk deel bij Philips komt te werken, vooral bij het Nat.lab. en TASS. Philips zal zijn koppositie als OOTI-afnemer dus zeker versterken. Nu de markt voor informatici weer zo goed is, verwacht ik dat de grote bedrijven weer meer OOTI's gaan trekken ten koste van het MKB.

### Kruseman Aretz

Van de in totaal 13 bijeenkomsten van de Opleidingsgroep en 9 bijeenkomsten van de Externe Adviescommissie heeft prof. Kruseman Aretz er slechts 2 resp. 1 gemist. Ik voorspel dat zijn aanwezigheidspercentages (nu resp. 85 en 89%) nog zullen stijgen tot minstens 90%. En aangezien zijn bijdrage in beide groeperingen zeer gewaardeerd wordt, hoop ik van harte dat deze voorspelling ook uitkomt.

Mijn laatste voorspelling tenslotte is dat ik ongetwijfeld nog wel eens "u" zal zeggen tegen Frans!

# Tien jaar informatica colloquium

Erik Jan Marinissen

Het Nat.Lab. Informatica Colloquium bestaat tien jaar. Ter gelegenheid van dit feit wordt vrijdag 17 februari een feestelijk lustrum-symposium georganiseerd in het Evoluon onder de titel *'Multimedia - Snelweg of Doolhof?'* Aanleiding voor een gesprek met prof.dr. Emile Aarts en prof.dr. Frans Kruseman Aretz, tien jaar geleden de twee grondleggers van het colloquium en zich beiden tot op heden nog sterk betrokken voelend bij het reilen en zeilen van hun boreling.

## Historie

De geschiedenis van de oprichting van het huidige Informatica Colloquium vertoont overeenkomsten met de weg die de informatica zélf heeft moeten afleggen om als wetenschap erkend te worden. Waar eerder informatica synoniem was voor programmeren, een activiteit die wetenschappers uit verschillende disciplines 'er wel even bij deden', had de informatica zich in het begin van de jaren tachtig een zelfstandige wetenschappelijke positie verworven. Kruseman: "Voor de oprichting van het Informatica Colloquium is er een tijdje een Software Colloquium geweest op het lab. Het had een heel ander karakter dan ons huidige colloquium. Makers van software systeempjes kwamen het gemaakte toelichten. Die voordrachten waren veelal weinig methodisch van aard en dus slechts voor een beperkt publiek interessant." Gebrek aan belangstelling en sprekers heeft er dan ook voor gezorgd dat dit colloquium geen lang leven beschoren was.

## Voedingsbodem

Begin jaren tachtig waren in Nederland akademische informatica-opleidingen nog maar net van start gegaan en de meeste mensen die op dat moment in de informatica werkzaam waren, hadden dan ook zelf een andere opleiding genoten. Loek Nijman, de kersverse groepsleider van de groep Computerarchitekturen, was één van de eersten die onderkenden dat er behoefte was aan versterking van de disciplinaire kennis van informatica op het Nat.Lab. Hij probeerde dat onder andere te bereiken door het opstarten van enkele boekenleesclubs. Emile Aarts, in die tijd net binnengekomen op het lab, heeft het Software Colloquium zelf niet meegemaakt, maar miste wel de aanwezigheid van een methodisch georiënteerd colloquium op zijn vakgebied. Aarts: "Met mijn komst op het Nat.Lab. heb ik een overstap gemaakt van de natuurkunde naar de informatica. Ik had een enorme behoefte aan disciplinaire kennis in mijn nieuwe vak. Een colloquium kan een goede manier zijn om zulke kennis op te doen. In de waaier van al bestaande Nat.Lab. colloquia

*Dit artikel is op 3 februari 1995 in nagenoeg identieke vorm verschenen in het *'Nat.Lab. Journaal'*, het twee-wekelijks verschijnend bedrijfsblad van het Philips Natuurkundig Laboratorium.

miste ik dan ook een Informatica Colloquium." Aarts stak zijn mening niet onder stoelen of banken en werd op een gegeven moment door een kollega in kontakt gebracht met Kruseman Aretz, op dat moment de enige (deeltijd) hoogleraar informatica op het lab en in het bezit van dezelfde behoefte aan disciplinaire kennisoverdracht. Uit dit eerste kontakt kwamen twee dingen voort: het Informatica Colloquium en een goede vriendschap. Beide duren tot op de dag van vandaag voort.

**De start**

Zo ging in het najaar van 1984 het Informatica Colloquium van start, met adjunkt-direkteur Henk Bosma als beschermheer. Aarts en Kruseman hadden zichzelf als organiserende commissie twee doelen gesteld: (1) het colloquium moest disciplinair van karakter zijn, gericht op bijscholing van het in de informatica werkzame deel van de Nat.Lab. bevolking (Aarts: "permanente edukatie") en (2) men wilde de beste informatici van Nederland als sprekers. Terugkijkend op de afgelopen tien jaar vinden beide heren dat de gestelde doelen bereikt zijn. Aarts: "Natuurlijk heb je er wel eens een zwak verhaal tussen zitten, maar over het algemeen heeft het colloquium goede technisch-inhoudelijke voordrachten, waarin de methode van probleemoplossen belangrijker is dan het probleem zelf."

**Kopstukken**

Ook voor de rij sprekers die aan het publiek is voorbijgetrokken in de loop der jaren hoeft men zich niet te schamen. Naast internationale grootheden als Dijkstra, Hoare en Hopfield, heeft iedereen die enige rol van betekenis speelt in informatica-Nederland wel eens een voordracht gehouden. "En voor degenen die niet werden uitgenodigd, hadden we meestal zo onze redenen", voegt Aarts daar breed grijnzend aan toe. Het colloquium groeide al snel uit tot een landelijk erkend, openbaar toegankelijk fenomeen. Het programma werd gepubliceerd in diverse vakbladen en voor elke bijeenkomst gingen konvokaties de deur uit naar alle Nederlandse informaticafakulteiten. Het werd als een eer beschouwd om als spreker gevraagd te worden. Kruseman: "Er is wel eens een hoogleraar uit den lande - neen, een naam noem ik niet - op me af gekomen, die zich enigszins verbolgen afvroeg waarom hij nog nooit was uitgenodigd." Rond de invoering van de badge-draagplicht op het Nat.Lab. besloot de direktie dat het colloquium alleen toegankelijk moest zijn voor Philipspersoneel; de vrij rondlopende externe colloquiumbezoekers vormden een te groot *security* risiko. Ondanks (of misschien juist wel dankzij) het feit dat er slechts weinig externe bezoekers waren, beschouwen Kruseman en Aarts dit nog steeds als een onnodige inperking van de akademische vrijheid van 'hun' colloquium.

**Lunch**

Bij het opzetten van het Informatica Colloquium werd de tweewekelijkse frekwentie van andere, al bestaande Nat.Lab. colloquia overgenomen. Kruseman: "Dat geldt ook voor de lunch met de spreker voorafgaand aan het colloquium. Dat hebben we overgenomen van het Vaste-Stof Colloquium, in die tijd georganiseerd door Martin Schuurmans." In de beginjaren was er tijdens de lunches altijd een adjunkt-direkteur aanwezig. Aarts: "Daarmee gaf de direktie aan dat zij zich betrokken voelde bij het colloquium en tevens was het een uiting van gastvrijheid ten aanzien van de spreker. Het gaf de lunch cachet." De

*Betrokkenen bij het Nat.Lab. Informatica Colloquium bijeen tijdens de lunch in gebouw WB. V.l.n.r. René Leermakers, Erik Jan Marinissen, Hans Dolfing, Lex Augusteijn, Frans Kruseman Aretz, Martin Rem, Marly Roncken en Emile Aarts.*

lunch van organiserende commissie en spreker is blijven bestaan; de direktieaanwezigheid daarbij overigens niet. Ook overgenomen van het Vaste-Stof Colloquium is de (bescheiden) onkostenvergoeding voor sprekers. Kruseman: "Voor ons stond voorop dat sprekers graag moesten willen spreken op dit colloquium, los van een financiële vergoeding. Die ene spreker die een honorarium van fl. 2000,- plus een retourvlucht Amsterdam - Eindhoven vroeg, hebben we dan ook afgewezen. Dat was in strijd met onze filosofie."

**Centurion**

Na een aantal jaren, toen het Informatica Colloquium inmiddels stevig op de rails stond en zich een volwaardige plaats tussen de andere Nat.Lab. colloquia had verworven, hebben Aarts en Kruseman de organisatie overgedragen aan een nieuwe commissie. Inmiddels hebben in totaal vier commissies, telkens bestaande uit drie personen uit verschillende sektoren, in zittingsperiodes van twee jaar de organisatie van het Informatica Colloquium voortgezet. Ten tijde van de Centurion reorganisaties maakte het colloquium een moeilijke tijd door. De 'nieuwe zakelijkheidswind' die door het Nat.Lab. trok deed de bezoekersaan-

tallen geen goed. Adjunkt-direkteuren en groepsleiders, maar ook gewone medewerkers hadden blijkbaar geen tijd of behoefte meer om aan 'permanente edukatie' te doen. Inmiddels is de situatie weer verbeterd. De huidige beschermheer Remi Bourgonjon heeft een begeleidingscommissie in het leven geroepen, waarin naast hemzelf ook Emile Aarts en Martin Rem zitting hebben. Deze commissie moet de continuiteit van het colloquium garanderen en tevens de zittende colloquiumcommissie ondersteunen met adviezen over de samenstelling van het programma. De huidige colloquiumcommissie, bestaande uit Hans Dolfing, Erik Jan Marinissen en Pieter Struik, heeft weer een interessant programma voor 1995 kunnen samenstellen en het vertrouwen is groot dat de ingezette stijgende lijn in de bezoekersaantallen voortgezet kan worden.

**Lustrum**

Kruseman en Aarts zijn natuurlijk trouwe bezoekers van het colloquium gebleven. Ook hun wekelijkse gezamelijke lunch in de kantine van gebouw WB is blijven bestaan, zelfs nu Kruseman gepensioneerd is. Het was dan ook tijdens één van die lunches dat het idee naar boven kwam dat het tien-jarig bestaan van het Informatica Colloquium niet ongemerkt voorbij mocht gaan. Het heeft wat voorbereidingstijd gevergd (zodanig dat het colloquium inmiddels eigenlijk al tien-en-een-half jaar bestaat), maar nu is er dan ook een lustrum-symposium met een interessant programma en, in de beste traditie van het colloquium zelf, klinkende namen op de sprekerslijst. Met als thema *multimedia* moet dit symposium zorgen voor bezinning op de plaats van de informatica.

**'Multimedia - Snelweg of Doolhof?'**

Ter gelegenheid van het tien-jarige bestaan van het Informatica Colloquium wordt op vrijdag 17 februari 1995 een lustrum-symposium gehouden. Sprekers:

- F.P. Carruba, lid Raad van Bestuur Philips Electronics NV
- R.H. Bourgonjon, lid direktie Philips Natuurkundig Laboratorium
- S.D. Swierstra, hoogleraar informatica Rijks Universiteit Utrecht
- E.J. Wintzen, president BSO

Het lustrum-symposium vindt plaats in de Philips Hall van het Evoluon, van 14.00 tot 17.00 uur (ontvangst vanaf 13.30 uur).
Hierbij worden belangstellenden uit de Nat.Lab. bevolking van harte uitgenodigd.

*** Lustrum-symposium**

# Een eigenschap van gebroken kwadratische functies

W. van der Meiden

1. Een *gebroken kwadratische functie* (afgekort, ook in het meervoud, *gkf*) is een functie van de gedaante

$$\varphi(z) = T(z)/N(z)$$

waarin $T$ en $N$ polynomen zijn met complexe coëfficiënten, van graad 2 of 1, met een complexe variabele $z$.
De gkf is niet een gebroken lineaire functie (glf); als

$$T(z) = az^2 + 2bz + c, \quad N(z) = a'z^2 + 2b'z + c' ,$$

dan zijn $a$ en $a'$ niet beide gelijk aan 0.
Bovendien mogen $T$ en $N$ niet een factor gemeen hebben. Opdat $T$ en $N$ een factor gemeen hebben, is nodig en voldoende dat hun resultante

$$\Delta = \Delta(T,N) = \begin{vmatrix} a & 2b & c & 0 \\ 0 & a & 2b & c \\ a' & 2b' & c' & 0 \\ 0 & a' & 2b' & c' \end{vmatrix} = (ac' - a'c)^2 - 4(ab' - a'b)(bc' - b'c)$$

gelijk is aan 0 ([Sch] §282; [W] §34).
Opdat $\varphi$ niet een glf is, is nodig en voldoende dat $\Delta \neq 0$. In het navolgende wordt voortdurend ondersteld dat $\Delta \neq 0$.

2. Een bekend voorbeeld van een gkf is de functie van Joukowski ([C], p. 237; [H], p. 294-99; [Sp], p. 224,229; ook [B])

$$j(z) = \tfrac{1}{2}(z + z^{-1}) ,$$

genoemd naar Nikolai Egorovich Zhukowskii (1847-1921; [M], p. 390).
Een eigenschap van Joukowski's functie is dat

$$w = j(z) \Leftrightarrow \frac{w-1}{w+1} = \left(\frac{z-1}{z+1}\right)^2$$

en als men de glf $\psi$ invoert, $\psi(z) = (z-1)/(z+1)$, dan staat er

$$j(z) = \psi^{-1}(\psi(z)^2) \ .$$

Met behulp van deze betrekking kan men gemakkelijk de volgende stelling bewijzen, die bij Nehari als een vraagstuk voorkomt: Er bestaat een 1-parameterstelsel van cirkels in $\boldsymbol{C}$ die door $j$ tweevoudig worden afgebeeld op cirkels ([N], p. 272).
In deze notitie wordt die eigenschap verscherpt en afgeleid voor willekeurige gkf. Daarbij wordt een beroep gedaan op bekende eigenschappen van glf, zoals:
Cirkels worden afgebeeld op cirkels, cirkelbundels op cirkelbundels, en geconjugeerde bundels op geconjugeerde bundels ([Schw]). Deze eigenschappen zijn, zoals Carathéodory heeft bewezen, ook karakteristiek.
Doch sommige afbeeldingen, die niet glf zijn, beelden wel sommige cirkels af op cirkels:
HULPSTELLING: De functie $w = q(z) = z^2$ beeldt cirkels met middelpunt 0, en geen andere cirkels, af op cirkels (met middelpunt 0); en zij beeldt lijnen door 0, en geen andere, af in lijnen (door 0).
(Opmerking: nulcirkels zijn niet meegerekend; lijnen zijn cirkels door $\infty$.)
*Bewijs*: Beschouw een cirkel in het $w$-vlak, met vergelijking

$$(u-m)^2 + (v-n)^2 = r^2 \ , \quad r > 0 \ ;$$

dan heeft het origineel in het $z$-vlak als vergelijking

$$(x^2+y^2)^2 - 2mx^2 + 2my^2 - 4nxy + m^2 + n^2 - r^2 = 0 \ .$$

Dit stelt een bicirculaire vierdegraadskromme voor, die alleen een cirkel kan bevatten door ontaarding. De vergelijking van een ontaarding moet zijn

$$(x^2+y^2+ax+by+c)(x^2+y^2+dx+ey+f) = 0 \ ,$$

omdat $x^2+y^2$ irreducibel is.
Door gelijkstelling van coëfficiënten en met enig rekenwerk vindt men dat de enig moge lijke ontaarding is

$$(x^2+y^2+r)(x^2+y^2-r) = 0 \ ,$$

dat is, in $\mathbb{R}^2$, de vergelijking van de cirkel $x^2+y^2 = r$ met beeld $u^2+v^2 = r^2$ in het $w$-vlak.
Beschouw in het $w$-vlak een lijn met vergelijking

$$mu + nv = r \ , \quad m, n \text{ niet beide } 0 \ ;$$

het origineel in het $z$-vlak is een hyperbool met vergelijking

$$m(x^2-y^2) + 2nxy = r \ ,$$

dan en slechts dan in rechten ontaard als het middelpunt er op ligt: $r = 0$. De twee rechten zijn dan $y = x\tan\alpha$ en $y = x\tan(\alpha + \frac{\pi}{2})$ waarin $\tan 2\alpha = -m/n$. □

3. Zij $\varphi(z) = T(z)/N(z)$ een gkf met $T(z) = az^2 + 2bz + c$, $N(z) = a'z^2 + 2b'z + c'$. De windingspunten van $\varphi$ worden gevonden uit de vergelijking $\varphi'(z) = 0$. Nu is

$$\varphi'(z) = 2\tau(z)/N(z)^2 ,$$

met

$$\tau(z) = \tfrac{1}{2}(T'(z)N(z) - T(z)N'(z)) = (ab' - a'b)z^2 + (ac' - a'c)z + (bc' - b'c) .$$

Als $ab' - a'b \neq 0$ dan is $\tau$ een kwadratische functie met discriminant

$$\Delta = (ac' - a'c)^2 - 4(ab' - a'b)(bc' - b'c) ,$$

zodat $\tau$ dan twee verschillende eindige nulpunten $\alpha$ en $\beta$ heeft. Het kan zijn dat $\tau$ en $N$ een nulpunt gemeen hebben; dan hebben de polynomen $\tau$ en $N$ een factor gemeen, en dan hebben, daar $T$ en $N$ relatief priem zijn, $N'$ en $N$ een factor gemeen; dus $N$ is een kwadraat, $N(z) = a'(z - \beta)^2$.

STELLING 1: Voor een gkf $\varphi(z) = T(z)/N(z)$, met $ab' - a'b \neq 0$ en $N$ niet een kwadraat, geldt

$$\frac{\varphi(z) - \varphi(\alpha)}{\varphi(z) - \varphi(\beta)} = -\frac{N(\beta)}{N(\alpha)}\left(\frac{z - \alpha}{z - \beta}\right)^2 ,$$

waarin $\alpha$ en $\beta$ de windingspunten van $\varphi$ zijn.

*Bewijs*: De vergelijking $\varphi'(z) = 0$ is nu equivalent met $\tau(z) = 0$ en van de tweede graad. Er zijn dus twee verschillende wortels $\alpha$ en $\beta$ (met $\varphi(\alpha) \neq \infty \neq \varphi(\beta)$). Door berekening vindt men

$$\varphi(z) - \varphi(\alpha) = N(\alpha)^{-1}N(z)^{-1}(z - \alpha)\{(z - \alpha)\tau'(\alpha) + 2\tau(\ alpha)\}$$

en daar $\tau'(\alpha) = -\tau'(\beta)$ volgt hieruit de betrekking. □

De stelling geldt ook als de teller een kwadraat is, $T(z) = a(z - \alpha)^2$. Dan is $a \neq 0$ en

$$\tau(z) = a(z - \alpha)\{(a'\alpha + b')z + (b'\alpha + c')\} = a(z - \alpha)\{\tfrac{1}{2}(z - \alpha)N'(\alpha) + N(\alpha)\} ,$$

$$\frac{\varphi(z) - \varphi(\alpha)}{\varphi(z) - \varphi(\beta)} = \frac{\varphi(z)}{\varphi(z) - \varphi(\beta)} = -\frac{N(\beta)}{N(\alpha)}\left(\frac{z - \alpha}{z - \beta}\right)^2 .$$

Stel nu dat de noemer een kwadraat is, en de teller niet:

$$\varphi(z) = \frac{T(z)}{a'(z-\beta)^2}\ , \quad a' \neq 0\ ,$$

dan voldoet de omgekeerde

$$\psi(z) = \frac{1}{\varphi(z)} = \frac{a'(z-\beta)^2}{T(z)}$$

aan de voorwaarden van stelling 1, zodat, daar $\psi(\beta) = 0$,

$$\frac{\psi(z)-\psi(\beta)}{\psi(z)-\psi(\alpha)} = \frac{\psi(z)}{\psi(z)-\psi(\alpha)} = -\frac{T(\alpha)}{T(\beta)}\left(\frac{z-\beta}{z-\alpha}\right)^2\ ,$$

$$\varphi(z) - \varphi(\alpha) = \frac{T(\beta)}{N(\alpha)}\left(\frac{z-\alpha}{z-\beta}\right)^2\ .$$

Indien bovendien de teller een kwadraat is, dan is

$$\varphi(z) = \frac{a}{a'}\left(\frac{z-\alpha}{z-\beta}\right)^2\ .$$

4. Ingeval $ab' - a'b = 0$ kan niet gelden $a' = 0$; immers, als $a' = 0$ dan is $a \neq 0$ en dan volgt uit $ab' - a'b = 0$ dat $b' = 0$, in strijd met de definitie van gkf. Dus $N(z)$ heeft graad 2, en de vergelijking $\varphi'(z) = 0$ heeft een wortel $\alpha = \infty$. Evenzo is $a \neq 0$ en graad $T(z) = 2$.
Als $ab' - a'b = 0$ kan niet gelden $ac' - a'c = 0$; immers zou dan $\Delta = 0$ zijn. Dus $ac' - a'c \neq 0$.
Als nu $bc' - b'c = 0$ dan voldoet $(a', b', c')$ aan een stelsel homogene lineaire vergelijkingen met matrix $\begin{pmatrix} -b & a & 0 \\ 0 & -c & b \end{pmatrix}$; als de rang hiervan 2 is, dan is $(a', b', c') = \mu(a, b, c)$ en $\Delta = 0$; dus is de rang 1 en geldt in dit geval $b = b' = 0$.
Als $ab' - a'b = 0$, dan is

$$\tau(z) = (ac' - a'c)z + (bc' - b'c)\ ,$$

en dan zijn de windingspunten van $\varphi$

$$\alpha = \infty \text{ en } \beta = -(bc' - b'c)/(ac' - a'c) = -b/a \neq \infty\ ;$$

als $b = 0$ dan is $\beta = 0$; als $N$ een kwadraat is, dan is $N(z) = a'(z-\beta)^2$.

5. Als $ab' - a'b = 0$, $N(z)$ is geen kwadraat en $\alpha = \infty$, dan is $\varphi(\alpha) \neq \infty$,

$$\varphi(z) - \varphi(\infty) = \frac{T(z)}{N(z)} - \frac{a}{a'} = -\frac{ac' - a'c}{a'N(z)} ,$$

$$\varphi(z) - \varphi(\beta) = \frac{(ac' - a'c)(z-\beta)^2}{N(\beta)N(z)}$$

als tevoren, en

$$\frac{\varphi(z) - \varphi(\infty)}{\varphi(z) - \varphi(\beta)} = -\frac{N(\beta)}{a'}\left(\frac{1}{z-\beta}\right)^2 .$$

Als evenwel bovendien $N(z) = a'(z-\beta)^2$, dan is

$$\varphi(z) - \varphi(\infty) = -\frac{ac' - a'c}{a'^2}\left(\frac{1}{z-\beta}\right)^2 .$$

6. We vatten de resultaten in een schema samen waaruit blijkt:

STELLING 2: Bij iedere gkf $\varphi : \boldsymbol{C} \to \boldsymbol{C}$ bestaan glf $\rho, \sigma : \boldsymbol{C} \to \boldsymbol{C}$ en een getal $\nu \in \boldsymbol{C}$ zó dat

$$(\sigma \circ \varphi)(z) = \nu(\rho(z)^2), \text{ of } \varphi(z) = \sigma^{-1}(\nu(\rho(z)^2) .$$

$\varphi$ heeft twee verschillende windingspunten, $\alpha$ en $\beta$.
Als $ab' - a'b \neq 0$, dan is $\rho(z) = (z-\alpha)/(z-\beta)$; als $ab' - a'b = 0$, dan is $\rho(z) = 1/(z-\beta)$.
Als $N(z) \neq a'(z-\beta)^2$, dan is $\sigma(w) = (w - \varphi(\alpha))/(w - \varphi(\beta))$; als $N(z) = a'(z-\beta)^2$ dan is $\sigma(w) = w - \varphi(\alpha)$.

| | $ab' - a'b \neq 0$<br>$\alpha \neq \infty \neq \beta$ | $ab' - a'b = 0$<br>$a \neq 0,\ a' \neq 0,\ ac' - a'c \neq 0$<br>$\alpha = \infty,\ \beta = -b/a$ |
|---|---|---|
| $N(z) \neq a'(z-\beta)^2$ | $\dfrac{\varphi(z) - \varphi(\alpha)}{\varphi(z) - \varphi(\beta)} = -\dfrac{N(\beta)}{N(\alpha)}\left(\dfrac{z-\alpha}{z-\beta}\right)^2$<br>① $\quad \nu = -\dfrac{N(\beta)}{N(\alpha)}$ | $\dfrac{\varphi(z) - \varphi(\infty)}{\varphi(z) - \varphi(\beta)} = -\dfrac{N(\beta)}{a'}\left(\dfrac{1}{z-\beta}\right)^2$<br>② $\quad \nu = -\dfrac{N(\beta)}{a'}$ |
| $N(z) = a'(z-\beta)^2$ | $\varphi(z) - \varphi(\alpha) = \dfrac{T(\beta)}{N(\alpha)}\left(\dfrac{z-\alpha}{z-\beta}\right)^2$<br>③ $\quad \nu = \dfrac{T(\beta)}{N(\alpha)}$ | $\varphi(z) - \varphi(\infty) = -\dfrac{ac' - a'c}{a'^2}\left(\dfrac{1}{z-\beta}\right)^2$<br>④ $\quad \nu = -\dfrac{ac' - a'c}{a'^2}$ |

Voorbeelden:

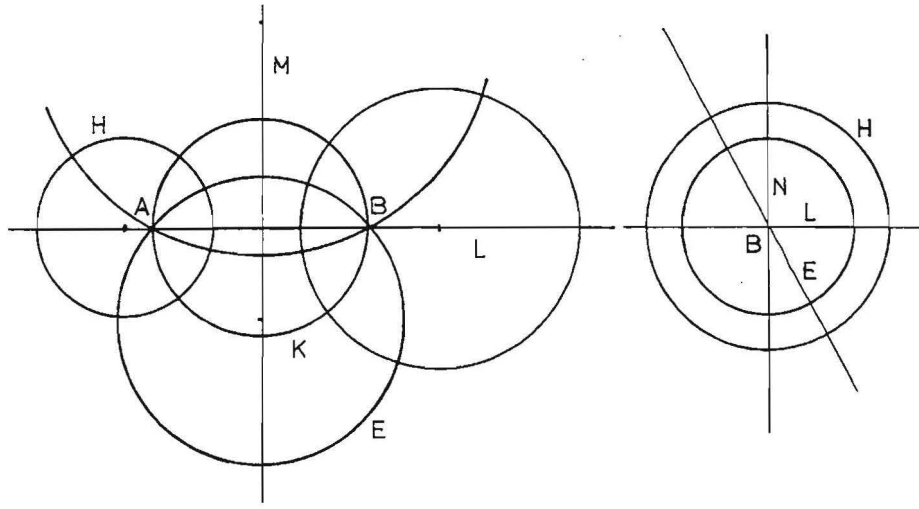
1. $\varphi(z) = (z-2)/(z^2-1)$; $\alpha, \beta = 1 \pm \frac{1}{2}\sqrt{3}$; $\dfrac{w - \frac{1}{2}\beta}{w - \frac{1}{2}\alpha} = \dfrac{\beta}{\alpha}\left(\dfrac{z-\alpha}{z-\beta}\right)^2$.

2. $\varphi(z) = z^2/(z^2+1)$; $\alpha = \infty$, $\beta = 0$; $\dfrac{w-1}{w} = -\left(\dfrac{1}{z}\right)^2$.

3. $\varphi(z) = (z-i)/(z-1)^2$; $\alpha = -1+2i$, $\beta = 1$; $w + \frac{1}{8}(1+i) = \dfrac{-1+i}{8i}\left(\dfrac{z+1-2i}{z-1}\right)^2$.

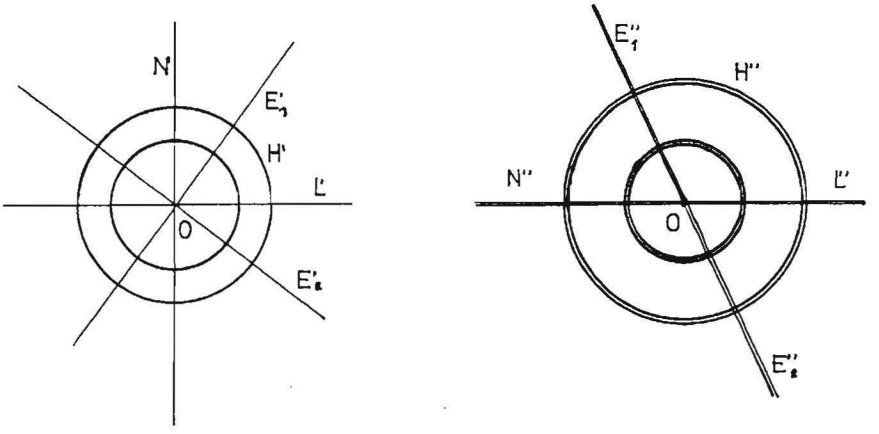
4. $\varphi(z) = (z^2+1)/z^2$, $\alpha = \infty$, $\beta = 0$; $w - 1 = \left(\dfrac{1}{z}\right)^2$.

7. Bij twee punten $\alpha, \beta \in \boldsymbol{C}$, $\alpha \neq \beta$, behoren twee cirkelbundels, namelijk de elliptische bundel $\mathcal{E}$ van cirkels door $\alpha$ en $\beta$, en de hyperbolische bundel $\mathcal{H}$ waartoe $\alpha$ en $\beta$, opgevat als puntcirkels $A$ en $B$, behoren. Tot $\mathcal{E}$ behoren de cirkel $K$ met middelpunt $\frac{1}{2}(\alpha+\beta)$ en de lijn $L$ door $\alpha$ en $\beta$; tot $\mathcal{H}$ behoort de middelloodlijn $M$ van het lijnstuk $AB$.




De cirkels in $\mathcal{E}$ staan loodrecht op de cirkels in $\mathcal{H}$.
Als $\alpha = \infty$ dan bestaat $\mathcal{E}$ uit rechten door $\beta$, $\mathcal{H}$ uit concentri sche cirkels met middelpunt $\beta$.
Als $\alpha \neq \infty \neq \beta$ dan worden bij de glf $\rho_1(z) = (z-\alpha)/(z-\beta)$ de bij $\alpha$ en $\beta$ beho-

rende bundels $\mathcal{E}$ en $\mathcal{H}$ afgebeeld op de bundels $\mathcal{E}'$ en $\mathcal{H}''$ die horen bij $0 = \rho_1(\alpha)$ en $\infty = \rho_1(\beta)$.




Als $\alpha = \infty$, dan worden de bij $\alpha$ en $\beta$ behorende bundels $\mathcal{E}$ en $\mathcal{H}$ door de glf $\rho_2(z) = (z - \beta)^{-1}$ afgebeeld op de bundels $\mathcal{E}'$ en $\mathcal{H}'$ die behoren bij $0 = \rho_2(\alpha)$ en $\infty = \rho_2(\beta)$.
Door te kwadrateren gaat de bundel $\mathcal{H}'$ als verzameling bijectief over in $\mathcal{H}'$; ieder exemplaar $H' \in \mathcal{H}'$ wordt tweevoudig afgebeeld op een exemplaar $H'' \in \mathcal{H}'$. De lijnen van de bundel $\mathcal{E}'$ worden tweevoudig afgebeeld op halfrechten door 0, die een stelsel $\mathcal{E}''$ vormen. De toevoeging van lijnen aan halfrechten, $\mathcal{E}' \to \mathcal{E}''$, is bijectief.
Door met een numerieke factor $\nu \neq 0$ te vermenigvuldigen worden de stelsels $\mathcal{H}''$ en $\mathcal{E}''$ weer bijectief op zichzelf afgebeeld.
In het geval dat $N(z) = a'(z - \beta)^2$ worden de stelsels bij $\varphi(\alpha)$ opgeteld om tenslotte over te gaan in $\mathcal{E}''' = \varphi(a) + \mathcal{E}'' = \varphi(\mathcal{E})$ en $\mathcal{H}''' = \varphi(a) + \mathcal{H}'' = \varphi(\mathcal{H})$.
In het andere geval zijn $\mathcal{E}''$ en $\mathcal{H}''$ beelden van stelsels $\mathcal{E}'''$ en $\mathcal{H}'''$ bij de glf $\sigma : \boldsymbol{C} \to \boldsymbol{C}$, $\sigma(w) = (w - \varphi(\alpha))/(w - \varphi(\beta))$. Hierbij is $\sigma^{-1}(0) = \varphi(\alpha)$, $\sigma^{-1}(\infty) = \varphi(\beta)$; $\mathcal{H}''' = \sigma^{-1}(\mathcal{H}'')$ is de cirkelbundel waarvan $\varphi(A)$ en $\varphi(B)$ exemplaren zijn; $\mathcal{E}''' = \sigma^{-1}(\mathcal{E}'')$ is het stelsel van cirkelbogen door $\varphi(A)$ en $\varphi(B)$; hierbij worden een halfrechte $E''$ en zijn verlengde $E''_\perp$ in dezelfde cirkel $E'''_0$ afgebeeld, ieder corresponderend met één der bogen van $\varphi(\alpha)$ naar $\varphi(\beta)$.
Zo komen wij tot de volgende

STELLING 3: Iedere gkf $\varphi : \boldsymbol{C} \to \boldsymbol{C}$ heeft twee windingspunten $\alpha$ en $\beta$. De cirkels uit de hyperbolische bundel $\mathcal{H}$ waarvan $\alpha$ en $\beta$ de basispunten zijn, worden tweevoudig afgebeeld op cirkels van de hyperbolische bundel $\varphi(\mathcal{H})$ met basispunten $\varphi(\alpha)$ en $\varphi(\beta)$; van de cirkels uit de elliptische bundel $\mathcal{E}$, die door $\alpha$ en $\beta$ gaan, zijn de beelden cirkelbo gen van $\varphi(a)$ naar $\varphi(b)$; hierbij worden orthogonale cirkels $E, E_\perp \in \mathcal{E}$ tweevoudig afgebeeld op complementaire bogen van één cirkel $E'''_0$. □

Voorbeeld: Bij Joukowski's functie $j(z) = \frac{1}{2}(z + z^{-1})$ worden de cirkel $|z| = 1$ en de $x$-as, beide exemplaren van de bundel $\mathcal{E}$ door 1 en $-1$, tweevoudig afgebeeld op achtereenvolgens het interval $[-1, 1]$ en $\mathbb{R}\backslash(-1, 1)$, die samen een exem plaar van de bundel door $j(1)$ en $j(-1)$ vormen.

Naschrift: Met dank aan J. Boersma voor enkele nuttige en critische opmerkingen.

**Literatuur**

[B] J. Boersma, Syllabus bij het college voortgezette functietheorie, lentetrimester 1995.

[C] R.V. Churchill et al, Complex variables and applications (3rd ed.). New York, etc.: McGraw–Hill, 1974.

[H] P. Henrici, Applied and computational complex analysis I. New York, etc.: Wiley, (1974) 1988.

[M] K.O. May, Bibliography and research manual of the history of mathematics. Toronto: University of Toronto Press, 1973.

[N] Z. Nehari, Conformal mapping. New York, etc.: McGraw–Hill, 1952 (Dover ed. 1975).

[Sch] F. Schuh, Nieuw leerboek der hogere algebra I. Zutphen: W.J. Thieme & Cie, 1943.

[Schw] H. Schwerdtfeger, Geometry of complex numbers. London: Constable, 1962 (Dover ed. 1979).

[Sp] M.R. Spiegel, Theory and problems of complex variables (SI (metric) editi on). New York, etc.: McGraw–Hill, 1981.

[W] B.L. van der Waerden, Algebra I (7. Aufl.). Berlin, etc.: Springer–Verlag, 1966.

# Bridging a gap with an old theorem

Wim Nuij

## Introduction

In geometric modeling, like in many other places, we go to a good deal of trouble to make live easy. To render a surface $f(x) = C$ it would be nice if we could just specify the function $f$ and the constant $C$, and leave the rest to the computing machinery. In this computation we have to deal with some minor problems, which we will solve for the simplest case you can imagine: $f$ is a quadratic function of the three-dimensional variable $x$. Of course it is not feasable to render each point of the quadric, we content ourselves with choosing a net of points on the surface, and rendering the resulting meshes in a standard way.

Now here a problem arises. The surface could be a two sheet hyperboloid, so we have to make shure that an edge connecting two points actually can be seen as an approximation of a curve on the surface, and is not bridging the gap between the two sheets. We will try to solve this problem as simply as possible, we will use computation of the value of $f$ in a given point, computation of its derivative (gradient), and of course Geometry. Please note: In the calculations below we will not let the exceptional cases disrupt the course of the reasoning.

## The planar case

As a first step we solve the problem for a plane figure. This is a good start because the restriction of a quadratic function to a plane is still a quadratic function. In more geometric terms: the intersection of a quadric and a plane is a conic section. So we are given two points on a conic section and have to determine if they are lying on the same branch (the question is relevant only for hyperbolas.)

By choosing appropriate coordinates we can transform the equation for the conic section into

$$(\mathcal{A}x, x) = \gamma$$

where $\mathcal{A}$ is a symmetric mapping and $\gamma$ can be taken positive. We will only use the existence of this representation in our reasoning, we will not perform the transformation. Denoting the given points on the conic by $a$ and $b$, we have

$$(\mathcal{A}a, a) = \gamma$$

and

$$(\mathcal{A}b, b) = \gamma$$

The tangents in $a$ and $b$ have the equations

$$(\mathcal{A}a, x) = \gamma$$

and

$$(\mathcal{A}b, x) = \gamma$$

respectively. The intersection of these tangents will be denoted by $s$. Some readers may recognize the situation: the point $s$ is the pole of $ab$ with respect to the conic, and $ab$ is the polar of $s$. This pole $s$ also satisfies $(\mathcal{A}a, x) = (\mathcal{A}b, x)$, a line that passes through the center 0 of the conic. We will prove that this line also passes through $m$, the midpoint of $ab$. This is the old theorem we alluded to in the title. Starting with

$$(\mathcal{A}a, a) = (\mathcal{A}b, b),$$

using that $\mathcal{A}$ is symmetric

$$(\mathcal{A}a, a) + (\mathcal{A}a, b) = (\mathcal{A}b, a) + (\mathcal{A}b, b),$$

and dividing by 2 we arrive at

$$(\mathcal{A}a, m) = (\mathcal{A}b, m),$$

which proves the assertion.

The above result allows us to put $m = \lambda s$, and we calculate

$$(\mathcal{A}a, m) = (\mathcal{A}a, \lambda s) = \lambda\gamma,$$

$$(\mathcal{A}b, m) = (\mathcal{A}b, \lambda s) = \lambda\gamma,$$

so adding and dividing by 2 results in

$$(\mathcal{A}m, m) = \lambda\gamma,$$

whereas

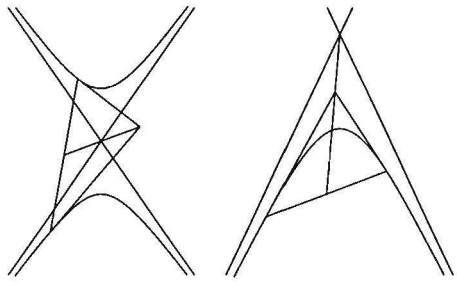
$$(\mathcal{A}s, s) = \gamma/\lambda.$$

Now if $\lambda > 0$ we see that the values $(\mathcal{A}m, m)$ and $(\mathcal{A}s, s)$ lie on opposite sides of $\gamma$ – the line $sm$ intersects the conic, and all lines through $s$ and a point of the linesegment $ab$ intersect the conic. We conclude that there is a curve from $a$ to $b$ on the conic.

On the other hand, if $\lambda < 0$ the quadratic function $(\mathcal{A}x, x)$ is atmost zero on the whole line through $s$ and $m$, so this line does not intersect the conic, and separates the two parts of the conic containing $a$ and $b$.

Now we get rid of the special coordinates we used in the proof. By assumption we can compute the gradient of $f$ in $a$ and $b$, so we know the tangents and finding the intersection $s$ is easy. Computing $m$ is even easier. We conclude:

If $f(a)$ and $f(b)$ are between the values $f(m)$ and $f(s)$ then $a$ and $b$ are on the same component of the conic, and if the values $f(m)$ and $f(s)$ are on the same side of $f(a)$ and $f(b)$ then $a$ and $b$ are on different parts.



## The space case

Now it is time to return to the original problem. In 3–space things are a little bit more complicated, one plane could intersect a hyperboloid along an ellips while another plane shows a hyperbola as intersection. Due to the general notation we need not change our formulae, only now the tangents are planes and their intersection is a line. The theorem now states: "The midpoint of a chord of the quadric lies in the plane through the center of the quadric and the intersection of the two planes tangent to the quadric at the endpoints of the chord".

Let the quadric be given by $f(x) = \gamma$, let $a$ and $b$ be points on the quadric, let $m$ be the midpoint of the linesegment $ab$ and suppose $f(m) > \gamma$. The intersection of the tangent planes in $a$ and $b$ is called $s$. On the line $s$ the function $f$ is quadratic, so we need only the function values in three points on $s$ to determine it completely. If there exists a point on $s$ where $f$ is less than $\gamma$, then we can apply the theorem to the conic section that results from intersecting the quadric with the plane through that point and $ab$, and conclude that there is a curve on the quadric connecting $a$ and $b$. If on the other hand $f$ is larger than $\gamma$ on the whole line $s$, then each line through $m$ and a point of $s$ separates two parts of an hyperbola, so the plane through $m$ and $s$ separates the two sheets of an two–sheet hyperboloid, and $a$ and $b$ are on different sheets.

## Concluding remarks

The problem as stated is a non–problem: using some higher mathematics we can find coordinates such that the matrix of $\mathcal{A}$ is diagonal and the question is easily answered. In the application I had in mind the quadratic function is an approximation of a function in a neighborhood where the linear approximation fails due to the gradient having a zero.

The calculations above are rather simple but I wonder if I hit upon this solution hadn't I learned projective geometry some forty years ago.

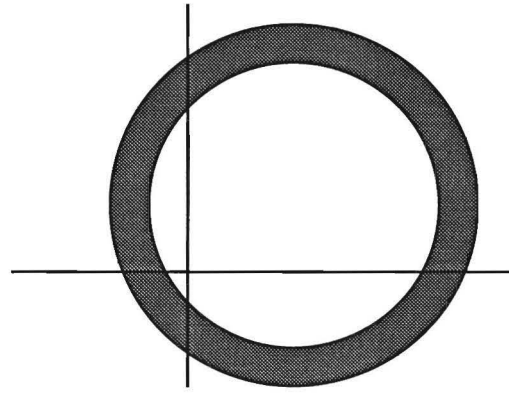
# Alfred Tarski, David Hilbert en spatial databases

Jan Paredaens

Frans, op onze talloze dinsdag-lunches op de TUE heb ik onder andere je belangstelling ervaren om practische problemen te onderbouwen met een mooie, eenvoudige maar toch formele theorie. Graag maak ik van deze gelegenheid gebruik om een stukje te schrijven in deze zin. Ik doe namelijk op dit ogenblik onderzoek, samen met een aantal medewerkers in Antwerpen en Eindhoven, in spatial databases. Er wordt recent vrij veel onderzoek in die richting gedaan, maar weinig aandacht wordt besteed aan het begrijpen van de diepere structuren die zich achter de mooie plaatjes, onder de baanbrekende visuele user interfaces of tussen de super geïntegreerde implementaties verschuilen. Wel hierover deze korte bijdrage, die kadert in onderzoek dat ik samen met Bart Kuijpers in de Universiteit van Antwerpen verricht.

Spatial databases zijn databases waarin twee soorten informatie behandeld worden: de thematische informatie (zeg maar de gewone informatie waarmee elke database werkt, zoals bedienden met hun adres, salaris en telefoonnummer) en de ruimtelijke informatie (zoals een waterkaart van Friesland, een netwerk van de Belgische spoorwegen of een recente foto van Laura Pausini). Alles draait om de voorstelling van deze ruimtelijke informatie. In se bestaat deze ruimtelijke informatie bijna altijd uit een oneindig (zelfs niet aftelbaar) aantal punten, zeg maar van het vlak voor de eenvoud. En zoals altijd en overal in ons vakgebied, wensen we deze informatie op een eindige manier voor te stellen. Er zijn mij maar twee algemene technieken bekend om dit te doen:

- we voeren een assenstelsel in en gebruiken analytische vergelijkingen gemengd met een soort logische formules. In de volgende figuur vind je een klein en (te) eenvoudig voorbeeld. Deze techniek is minder interessant omdat de formules zeer vlug te ingewikkeld worden, wil men een voldoende benadering krijgen van de werkelijkheid.



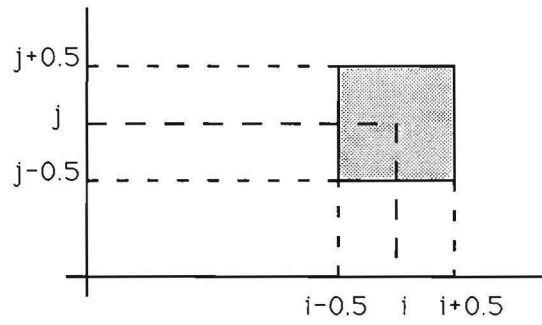
$$\{(r,s)|(81 < (r-6)^2 + (s-5)^2) \wedge ((r-6)^2 + (s-5)^2 < 121)\}$$

- we nemen een eindig aantal punten, al het ware een soort representanten, en slagen deze in de database op. Het gebruik van rasters, het digitaliseren van foto's, het concept van pixels liggen allemaal in de sfeer van deze aanpak.

Het is deze tweede techniek die we nader gaan bekijken. In feite verdelen we de oneindige verzameling punten in een eindig aantal klassen. Elke van deze klassen wordt dan voorgesteld door één punt. Ik zal misschien eerst even een aantal voorbeelden geven.

• [Rastervoorbeeld] een zeer bekend voorbeeld is het gebruik van gehele coördinaten. Stel dat we een begrensde figuur $D$ in het vlak hebben. We zullen $D$ voorstellen door een eindige verzameling van punten met gehele coordinaten $\rho(D)$ op de volgende manier:

$\rho(D) = \{(i,j)|$ er bestaat een punt $(r,s)$ in $D$ dat voldoende dichtbij $(i,j)$ ligt, d.w.z. $i - 0.5 \leq r < i + 0.5$ en $j - 0.5 \leq s < j + 0.5\}$. Voor elke gehele waarde $i$ en gehele waarde $j$ vormt de vierkant met hoekpunten $(i-0.5, j-0.5)$, $(i-0.5, j+0.5)$, $(i+0.5, j-0.5)$, $(i+0.5, j+0.5)$ dus een klasse. Als in dat vierkant een punt van $D$ voorkomt dan komt het punt $(i,j)$ in $\rho(D)$ voor.

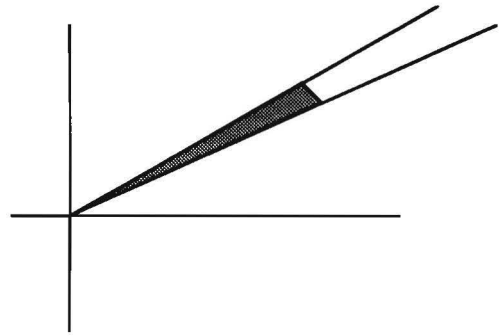


• [Driehoekenvoorbeeld] in plaats van van vierkanten kunnen we driehoeken of zeshoeken als klassen gebruiken.

• indien we enkel geïnteresseerd zijn in de punten van het eerste kwadrant, maken we van de drie andere kwadranten samen één grote klasse. Als de figuur $D$ een punt heeft in die andere kwadranten dan komt de representant van deze grote klasse in $\rho(D)$ voor.

• indien we enkel geïnteresseerd zijn in de $x$-coordinaten, in de projectie op de $x$-as dus, dan beschouwen we de verticale smalle banden $\{(r,s)|i-0.5 \leq r < i+0.5\}$ als klassen.

• men kan ook als klassen de gebieden nemen van het vlak die men vanuit de oorsprong onder een bepaalde hoek-interval ziet.



• [Afstandsvoorbeeld] stel dat we enkel geïnteresseerd zijn in de afstand van elk punt tot de oorsprong dan beschouwen we het oneindig aantal concentrische cirkels rond de oorsprong als klassen.

Je begrijpt, Frans, dat dit telkens neerkomt op het definiëren van een equivalentie relatie.

In het Rastervoorbeeld zijn twee punten $(r_1, s_1)$ en $(r_2, s_2)$ equivalent als $\forall i \in Z(r_1 < i \Leftrightarrow r_2 < i) \wedge (s_1 < i \Leftrightarrow s_2 < i)$. In het driehoekenvoorbeeld zijn twee punten $(r_1, s_1)$ en $(r_2, s_2)$ equivalent als

$$\begin{array}{l} \forall i \in Z((r_1 < i \Leftrightarrow r_2 < i) \wedge \\ \quad (s_1 < \sqrt{3}(r_1 - i) \Leftrightarrow (s_2 < \sqrt{3}(r_2 - i) \wedge \\ \quad (s_1 < \sqrt{3}(i - r_1) \Leftrightarrow (s_2 < \sqrt{3}(i - r_2)) \end{array}$$

Dus onze voorstelling wordt volledig gekarakteriseerd door een equivalentierelatie $\approx$ op de punten van het vlak. We noemen nu twee delen van het vlak, $D_1$ en $D_2$ equivalent, $D_1 \approx D_2$ als ze op dezelfde manier voorgesteld worden, m.a.w. als

$$(\forall p \in D_1 \; \exists q \in D_2 \; p \approx q) \wedge (\forall q \in D_2 \; \exists p \in D_1 \; p \approx q)$$

Merk op Frans, dat deze equivalentierelatie gesloten is voor de unie, maar niet voor de intersectie noch voor het complement.

Nu stelt zich natuurlijk de vraag, welke equivalentierelatie kiezen we, of anders gezegd aan welke voorwaarde(n) moet de gekozen equivalentierelatie voldoen. Hiervoor moet ik je eerst zeggen wat een vraag of een query is. Een query $Q$ is niets anders dan een functie van de verzameling van alle delen van het vlak naar een beeldverzameling. Stel nu dat $Q$ de verzameling is van alle queries die zullen gesteld worden aan mijn database. Het spreekt voor zichzelf dat voor elke query $Q$ van $Q$ moet gelden dat als we $Q$ loslaten op twee equivalente delen we hetzelfde antwoord krijgen, of nog

$$\forall Q \in Q \forall D_1 \forall D_2 (D_1 \approx D_2 \Rightarrow Q(D_1) = Q(D_2)) \quad (1)$$

Als (1) voldaan is voor $Q$ en de equivalentierelatie $\approx$ dan noemen we $\approx$ $Q$-compatibel.

Als we even terugkijken naar ons Rastervoorbeeld dan zullen de volgende queries inderdaad hetzelfde antwoord geven op twee equivalente delen:

- zijn er punten in het eerste kwadrant?
- zijn er punten waarvan de $x$-coordinaat tussen 1 en 2 ligt ?
- ligt het gegeven deel volledig binnen het vierkant met hoekpunten $(0,0)$ en $(10,10)$ ?
- spiegel het gegeven deel volgens de $y$-as.

Bij de volgende queries gaat het echter mis:

- ligt er een punt op een afstand kleiner dan 1 van $(0,0)$ ?
- ligt het gegeven deel volledig binnen het vierkant met hoekpunten $(0.5,0.5)$ en $(10,10)$ ?
- translateer het gegeven deel naar oost, volgens de $x$-as over een afstand 1.3.

Tot hiertoe lijkt dit allemaal vrij eenvoudig. De hamvraag is hier echter of de uitdrukking (1) in het algemeen beslisbaar is voor een gegeven equivalentierelatie $\approx$ en een gegeven verzameling $Q$ van queries. Uiteraard is het antwoord negatief. Eerst en vooral moeten we ons beperken tot berekenbare queries, beslisbare equivalentierelaties en beslisbare delen van het vlak. In dat geval is de uitdrukking (1) van tweede orde en vermoeden we sterk dat ze onbeslisbaar is.

Een zeer interessant geval doet zich voor indien de equivalentierelatie semi-algebraisch is (d.i. kan uitgedrukt worden door logische formules waarin vergelijkingen voorkomen met polynomen met reële coëfficiënten) en indien de queries $Q$ ook semi-algebraïsch zijn. Als $Q$ daarenboven nog de eigenschap heeft dat $Q(D) = U_{p \in D} Q(p)$, dan noemen we $Q$ een puntquery. Merk op Frans, dat de queries hierboven allemaal semi-algebraïsche puntqueries zijn.

"ligt het gegeven deel volledig binnen het vierkant met hoekpunten $(0,0)$ en $(10,10)$ ?" wordt uitgedrukt door :

$$\forall r \in I\!R\, \forall s \in I\!R\, ((r,s) \in D \Rightarrow r > 0 \wedge r < 10 \wedge s > 0 \wedge s < 10)$$

en "is het gegeven deel een cirkel ?" is geen puntquery maar is wel semi-algebraïsch :

$$\exists r \in I\!R\, \exists r_1 \in I\!R\, \exists s_1 \in I\!R\, \forall r_2 \in I\!R\, \forall s_2 \in I\!R \\ ((r_2,s_2) \in D \Leftrightarrow (r_2 - r_1)^2 + (s_2 - s_1)^2 = r)$$

Onze rasterequivalentie beperkt tot het vierkant $((0,0),(10,10))$ en de afstandsequivalentierelatie zijn ook beide semi-algebraïsch. Zij worden respectievelijk uitgedrukt door :

$$\forall i \in \{0,1,...,10\}(r_1 < i \Leftrightarrow r_2 < i) \wedge (s_1 < i \Leftrightarrow s_2 < i) r_1^2 + s_1^2 = r_2^2 + s_2^2$$

Subtiel maar zeer belangrijk is dat de onbeperkte rasterequivalentie niet semi-algebraïsch is daar we een variabelen moeten laten lopen over alle **gehele** getallen.

Steunend op een zeer belangrijke stelling van Tarski waarin hij stelt dat men alle gebonden variabelen in een semi-algebraïsche uitdrukking kan elimineren kunnen wij het volgende bewijzen :

Het is beslisbaar of een semi-algebraïsche equivalentie relatie $Q$-compatibel is, als $Q$ enkel semi-algebraïsche puntqueries bevat.

Het geval waar $Q$ ook semi-algebraïsche niet-puntqueries bevat hebben we nog niet volledig opgelost. We trachten het te herleiden tot het tiende Hilbert probleem. Hier gebruiken we echter enkel polynomen met twee variabelen en voor zover wij kunnen nagaan is het probleem of het beslisbaar is om na te gaan of een polynoom met twee variabelen een gehele oplossing heeft, nog steeds open. Maar ... we zoeken verder.

Ziezo Frans, onze oude wiskundigen en logici spelen toch nog altijd een belangrijke rol, zelfs in de domeinen waarvan zij het bestaan nog niet kenden.

# Discrete computer graphics

Frans J. Peters

## 1 Introduction

Computer Graphics is concerned with rendering images from scene description, where the scenes may be composed of many objects. The scene may be two-dimensional, such as a page of text, three-dimensional, such as the model of a building, or multi-dimensional, such as scientific data generated by a simulation package. Usually, the objects in the scene are described by a set of parameters, such as the coordinates of points in 3D space, plus a set of rules describing how the surfaces that bound the objects are constructed from these parameters. A simple example in 2D would be the definition of a polygon, consisting of the vertices as parameters, plus the rule that these vertices are to be connected in a certain order by straight line segments.

As most graphics output devices nowadays are raster based, it seems quite natural to use integers only. As computing with real numbers presents all kinds of difficulties, algorithms have been developed that deal with the models in integer arithmetic only. The most well known example is Bresenham's algorithm for rendering line segments [2]. However, there is a price to pay: using integers only requires that all kinds of familiar notions (like intersection of two straight lines and the familiar Jordan curve theorem) are to be redefined. A discrete geometry needs to be developed.

Besides cheap hardware, the restriction to integer arithmetic has another important advantage: robustness. Integer arithmetic allows for exact operations; there is no need for elaborate numerical analysis on the robustness of the algorithms, nor there is a need for techniques like interval arithmetic [10].

In the following, some of the results that have been obtained in the area of discrete graphics [8, 11, 12] will be presented.

## 2 Line functions

Let us start with an apparently simple concept: line segments. What are proper definitions for line segments in a discrete geometry? Such definitions can be given in a practical - i.e. algorithmic - way. Given two end points compute the pixels that provide the approximation of the line segment that connects those end points. This is called digitisation of line segments. Most widely used is Bresenham's algorithm [2].

However, different ways of digitisation can be thought of. It is therefore possible to specify desired properties of such digitisations. The following definitions can be found in [12]. Let us define $f$ to be a line function (short for line segment digitisation function) iff $f$ maps line segments with integer end points $b$ and $e$ onto sets of pixels; that is to say $f(b, e)$ is a set of pixels.

Desirable properties can be:

**closeness** A line function $f$ is called close iff for all $b$, $e$, for each pixel $p \in f(b, e)$ there is a point $r$ on the (original) line segment such that the distance between $p$ and $r$ is less than one (for a suitably chosen distance function), and conversely, for each point $r$ there is a pixel $p$ on a distance less than one.

**minimality** A line function $f$ is minimal iff for all $b$, $e$ with $b = (bx, by)$ and $e = (ex, ey)$ the number of pixels in $f(b, e)$ equals $1 + \max(|\ bx - ex\ |, |\ by - ey\ |)$.

**translation invariance** A line function $f$ is called translation invariant iff for all $b$, $e$, $r$:

$$f(r + b, r + e) = r \oplus f(b, e),$$

where $r \oplus f(b, e) = \{r + v \mid v \in f(b, e)\}$

**convexity** A line function $f$ is called convex iff for all $b$, $e$ the following holds:

$$\text{for all } r, s \in f(b, e) : f(r, s) \subset f(b, e).$$

Convexity is a very desirable property for line functions in interactive applications where parts of lines on the screen have to be erased [12, 5].

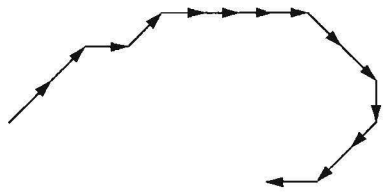
van Lierop [12] presents numerous line functions and investigates the properties they have. Maes [7] presents a very surprising result: closeness and convexity are mutually exclusive properties. Hence, if $f$ is a convex function, it is not close. Interesting to know then is its deviation, i.e. what the distance can be between a pixel $p$ from the set $f(b, e)$ and the point on the line segment between $b$ and $e$ that is most close to $p$. All known convex functions have a deviation that is at least logarithmic in $m$, where the image space consists of $m \times m$ pixels.

van Lierop [12] shows that the function resulting from Bresenham's algorithm is close, minimal and translation invariant. Note that Bresenham's algorithm allows efficient implementation in integer arithmetic requiring shifts and additions only.

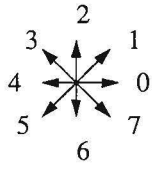
## 3 General discrete curves

So far we have dealt with (discretisations) of straight lines only. van de Wetering [11] presents general discrete curves. Chain coding [6] has been introduced for the representation of such curves. A chain consists of a consecutive number of codes, where a code

represents a relative displacement vector (see figure 1). Hence a starting point together with a chain represents a (discrete) curve.



(a) a discrete curve




(b) encoding of the vectors

[ 11010000776554]

(c) the corresponding chain

Figure 1: A discrete curve and its chain coding

For modeling purposes however this is not a convenient representation. A more convenient representation is based upon (a restricted number of) control points. For this, so called w-curves are introduced. If a w-curve is given by control point $p_0$ up to $p_n$, then that curve consists of the chain codes associated with the Bresenham's digitisation of the line segments $[p_i, p_{i-1}]$ for $0 < i \leq n$; the precise way in which these chain codes are mixed (yielding the chain code for the w-curve) is determined by so called distribution functions. By choosing different distribution functions, one obtains different curves (comparable to

the effect of the choice of different blending functions in continuous geometric modeling). The usefulness of w-curves is readily indicated by the fact that they can be used to represent discretisations of not only circles and ellipses, but also second and third order Bezier curves.

Algorithms for rendering these w-curves use only simple integer arithmetic; such algorithms have been developed for filling closed curves as well [11]. In the next section another approach (based upon the point containment paradigm) is presented.

## 4 Point containment

Usually, regions in 2D are defined by their outlines. An outline is a closed (not necessarily simple) curve. The notion of winding number is used to define which points belong to the inside of the region specified by such a closed curve. The interior of the region consists of all those points that have a non-zero winding number; another definition is: the interior consists of all points with a odd winding number.

There are two essentially different ways to define the notion of winding number: the analytical version employs a complex contour integral (see e.g. [1]), the geometric version counts the number of directed intersections with a ray to infinity (see e.g. [4]). E.J. Pol developed a theory powerful enough to show not only the equivalence of these versions, but also to prove a discrete version of the famous Jordan's theorem [8]. As it would require too much space in this short paper, we will neither prove, nor present here that Discrete Jordan Theorem, but instead restrict ourselves to an important application.

The Discrete Jordan Theorem is used to prove the correctness of algorithms for filling outlines as developed by M. Corthout. Next we will sketch such an algorithm. It is our purpose to convey the simplicity and elegance of the algorithm.

Let the region to be filled be given by a list $L$ of boundary segments, where each segment is a discrete Bezier curve of degree at most $n$. Curves of degree three are defined by four control points.

The filling algorithm is based upon the point containment paradigm [3, 8]: for each pixel it is determined whether it belongs to the region to be filled. Hence, for each pixel its winding number with respect to $L$ is computed. The winding number $W$ of query point $P$ (i.e. a pixel given by its integer coordinates) is computed by summing the contributions of all the segments of $L$ to $W$. Those contributions are calculated as follows.

Assume the segment $L$ to be given by the four control points $L_0$, $L_1$, $L_2$, and $L_3$. The contribution of $L$ to $W$ consists of the number of (directed) intersections of the ray from $P$ to minus infinity parallel to the $X$-axis.

If $P$ is inside a shaded region as indicated in figure 2, the contribution to $W$ is zero. If $P$ is inside the other shaded region in figure 2, the contribution to $W$ is:

$$s((L_3 - P)^y) - s((L0 - P)^y) \tag{1}$$

where the superscript $y$ denotes the $y$-coordinate and $s$ is an integer sign function defined by:

$$s(x) = 1 \textit{ iff } x < 0, s(x) = 0 \textit{ iff } x \geq 0$$

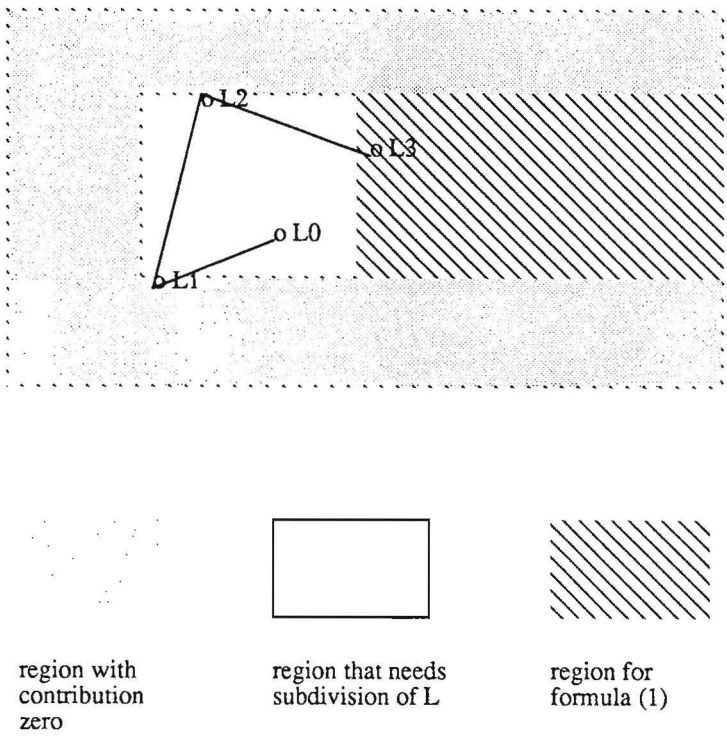



Figure 2: Distinct regions for filling algorithm

If $P$ is inside the non-shaded region in figure 2, a divide and conquer technique is applied. In the base case (i.e. the segment can not be subdivided because it is too small), the contribution to the winding number $W$ is easily calculated; otherwise, the segment is subdivided into two halves according to de Casteljau, and by recursive calls the contributions of each of the two halves are calculated.

Figure 3 shows the pseudo code for the algorithm; note that to simplify the algorithm a normalizing transformation is applied: a translation is applied such that query point $P$

coincides with the origin.

```
WindingNumber (L,P) /* computes winding number of P wrt. L */
Point P;
ListOfLists L;
{  sum = 0;
   for (i=0; i<Length(L); i++)
      sum += Count(L[i]-P); /* normalising */
   return(sum);
}

Count(L)   /* determines contribution Bézier curve L to winding number */
List L;
{  n = Length(L);

   minx = minimum X-coordinates of L;
   maxx = maximum X-coordinates of L;
   miny = minimum Y-coordinates of L;
   maxy = maximum Y-coordinates of L;

   if (minx > || miny >= 0 || maxy < 0) return(0);     /* see Figure 2 */

   if (maxx <=0 ) return( s(L[n].y) - s(L[0].y) );     /* see Figure 2 */

   if (maxx-minx <= 1 && maxy-miny <=1)                /* base case */
  {    if (L[0].x==0 && L[0].y==0 && L[n].y<0) return(1);
       if (L[n].x==0 && L[n].y==0 && L[0].y<0) return(-1);
       return(0);
  }

   left  = LeftHalf(L);        /* divide by applying de Casteljau */
   right = RightHalf(L);

   return (Count(left) + Count(right));
}
```

Figure 3: Pseudocode for filling

The most remarkable property of the algorithm is the absence of special cases (compare this to the commonly employed algorithms for point-in-polygon tests). Many more properties of this algorithm are shown in [8], such as:

- the algorithm is tiling
- the algorithm is easily generalized to rational curves (this generalization involves only a single multiplication per coordinate per control point per query point)
- at most $\lceil {}^2\log m \rceil + n$ subdivisions are required before the stopping criterion is reached (assuming the image space consists of $m \times m$ pixels)
- the algorithm requires a maximum stack depth of $n - 1$. (Remember n to be the maximum degree of the Bezier curves allowed.)

In summary, the algorithm has a proven robustness, i.e. it can handle arbitrarily complex outlines while its memory usage remains very limited: only a few local variables and a small stack memory is required. The size of the stack and the number of locals is independent of the outline complexity.

The algorithm (with various extensions) has been implemented in silicon. The resulting chip supports the rasterisation of the basic graphics primitives of the PostScript page description language. The chip design contains about 220 K transistors.

The major disadvantage of point containment algorithms is their quadratic time complexity as a function of resolution. In [8] it is shown however that this objection can be met by applying the so called coherence test for large regions. For these coherence tests again the basic point containment test is used.

Recently, it has been shown that the concepts and the theory underlying the Discrete Jordan Theorem (and hence the theorem itself) can be generalized not only to the 3-dimensional case, but even to arbitrary dimensions [9].

## References

[1] Ahlfors, L.V., Complex Analysis, McGraw-Hill, New York, 1953.

[2] Bresenham, J.E. Algorithm for Computer Control of a Digital Plotter, IBM Systems Journal, 4 (1), pp. 25 - 30, 1965

[3] [3] Corthout, M. and H. Jonkers, A new point containment algorithm for B-regions in the discrete plane. Theoretical Foundations of Computer Graphics and CAD, Springer Verlag, pp. 279 - 306, 1988

[4] Courant, R. and H. Robbins, What is Mathematics? Oxford University Press, 1941.

[5] Franklin, W.R. Problems with Raster Graphics Algorithms, in: Data Structures for Raster Graphics, (Proceedings of a Workshop held at Steensel, The Netherlands, June 24-28, 1985); ed. L.R.A. Kess ner,F.J. Peters and M.L.P. van Lierop. Berlin: Springer Verlag, 1986

[6] Freeman, H. Computer Processing of Line-Drawing Images, Computing Surveys, 6, pp. 57 - 97, 1974

[7] Maes, M. Digitization of Straight Line Segments, Closeness and Convexity, Computer Vision, Graphics and Image Processing, 52, pp. 297 - 305, 1990.

[8] Pol, E.J.D. and M.E.A. Corthout, Point Containment and the PHAROS Chip, Ph.D. Thesis, Leiden University, 1992

[9] Pol, E.J.D., M.E.A. Corthout and F.J. Peters, The Discrete Jordan Theorem for self-intersectring closed boundaries in arbitrary dimensions. Unpublished manuscript. 1994.

[10] Snyder, J.M. Interval Analysis for Computer Graphics, Computer Graphics, 26 (2), pp. 121 - 130, 1992

[11] van de Wetering, H. Chain Coding in Computer Graphics, Ph.D. Thesis, Eindhoven University of Technology, 1991

[12] van Lierop, M.L.P. Digitisation Functions in Computer Graphics, Ph.D. Thesis, Eindhoven University of Technology, 1987

# The carré problem

Martin Rem

A carré is a sequence of even length whose first half equals its second half. Examples of carrés are '123123' and 'dodo'. The problem we want to solve is the construction of a system that determines for each segment of length $2N$ of its input stream (for given $N$) whether it is a carré. The system reads an infinite stream of input values and produces an equally infinite stream of boolean output values. Calling these two streams $A(i: i \geq 0)$ and $B(i: i \geq 0)$, respectively, the problem can be specified by input/output relation, for $i \geq 0$,

$$B(i) = (\forall j: 0 \leq j < N: A(i+j) = A(i+N+j))$$

which states that boolean $B(i)$ expresses the equality of segments $A[i..i+N)$ and $A[i+N..i+2N)$.

We show that the carré problem can be solved with an $N$-place shift register and one additional process. An $N$-place shift register is a system with one input stream, say $A$, and one output stream, say $B$. Its input/output relation is $B(i) = A(i-N)$ for $i \geq N$. The first $N-1$ output values are thus left unspecified: $B(N) = A(0)$, $B(N+1) = A(1)$, etc. Its communication behavior is $(A\,,\,B)^*$; in words: it repeatedly performs concurrent communication actions, one at each of its two ports. The design of shift registers is an art in itself (see, e.g. [1]). In this note we address the design of the additional process only.

The specification of the carré problem given above requires the first $B$-output, i.e. $B(0)$, to indicate whether segment $A[0..2N)$ is a carré. Thus, $2N$ input values have to be examined before the first output value can be generated. Since such specifications complicate the code (in particular the initialization), it has become customary to leave the first so-many output values unspecified, just as we did for the shift register. In this case we leave the first $2N-1$ output values unspecified. This changes the specification into

$$B(i) = (\forall j: 0 \leq j < N: A(i-N-j) = A(i-j))$$

for $i \geq 2N-1$. The universal quantification now expresses the equality of segments $A(i-2N..i-N]$ and $A(i-N..i]$, which, by virtue of $i \geq 2N-1$ (i.e. $i-2N \geq -1$), lie completely within the input stream.

Rather than immediately solving the problem for boolean outputs, we first generalize the problem into one with natural outputs. Thereto we design a process with natural output port $b$ along which an output stream $b(i\colon i \geq 0)$ is generated that is given by

$$b(i) = (\uparrow k\colon 0 \leq k \leq i{-}N{+}1 \;\wedge\; (\forall j\colon 0 \leq j < k\colon A(i{-}N{-}j) = A(i{-}j))\colon k)$$

for $i \geq N{-}1$ ($\uparrow$ denotes the maximum quantifier). Because $B(i) = (b(i) \geq N)$ for $i \geq 2N{-}1$, this is indeed a generalization of the boolean problem. Since $i{-}N{+}1 \geq 0$, the range of $k$ is nonempty: it includes $k = 0$. The universal quantification above expresses that

$$A(i{-}N{-}k..i{-}N] = A(i{-}k..i]$$

Since $i{-}N{-}k \geq -1$, both segments in this equality are part of the input stream.

We investigate how $b(i)$ may be computed for $i \geq N{-}1$. Obviously $b(N{-}1) = 0$. Next consider $b(i)$ for $i > N{-}1$. We distinguish two cases: $A(i{-}N) \neq A(i)$ and $A(i{-}N) = A(i)$. In the former case we have $b(i) = 0$. For the latter case we derive

$$\begin{array}{ll}
 & b(i) \\
= & \quad \{\text{i/o relation for port } b\} \\
 & (\uparrow k\colon 0 \leq k \leq i{-}N{+}1 \;\wedge\; (\forall j\colon 0 \leq j < k\colon A(i{-}N{-}j) = A(i{-}j))\colon k) \\
= & \quad \{A(i{-}N) = A(i)\} \\
 & (\uparrow k\colon 0 \leq k \leq i{-}N{+}1 \;\wedge\; (\forall j\colon 1 \leq j < k\colon A(i{-}N{-}j) = A(i{-}j))\colon k) \\
= & \quad \{\text{dummy change: } j := j{+}1,\ k := k{+}1\} \\
 & (\uparrow k\colon -1 \leq k \leq i{-}N \;\wedge\; (\forall j\colon 0 \leq j < k\colon A(i{-}N{-}j{-}1) = A(i{-}j){-}1)\colon k{+}1) \\
= & \quad \{\text{since } i > N{-}1,\ k = 0 \text{ is a solution (and a tiny bit of calculus)}\} \\
 & (\uparrow k\colon 0 \leq k \leq i{-}N \;\wedge\; (\forall j\colon 0 \leq j < k\colon A(i{-}N{-}j{-}1) = A(i{-}j){-}1)\colon k) + 1 \\
= & \quad \{\text{i/o relation for port } b\} \\
 & b(i{-}1) + 1
\end{array}$$

We, consequently, have for $i > N{-}1$

$$\begin{array}{l}
b(i) = \textbf{if } A(i{-}N) \neq A(i) \rightarrow 0 \\
\qquad [\!] \; A(i{-}N) = A(i) \rightarrow b(i{-}1) + 1 \\
\qquad \textbf{fi}
\end{array}$$

The formula above shows how the $b$ outputs may be computed from the $A$ inputs (and from the preceding $b$ output).

In order to produce the outputs at port $b$ the process is equipped with two integer input ports, $a$ and $c$, along which input stream $A$ is received:

$$a(i) = A(i) \qquad i \geq 0$$
$$c(i) = A(i{-}N) \qquad i \geq N$$

Substitution in the formula for $b$ yields, for $i > N{-}1$

$$\begin{array}{l} b(i) = \textbf{if } c(i) \neq a(i) \rightarrow 0 \\ \quad [\!] \; c(i) = a(i) \rightarrow b(i{-}1) + 1 \\ \quad \textbf{fi} \end{array}$$

As communication behavior of the process we adopt

$$(a\,,\, c\,;b)^*$$

This behavior expresses an infinite repetition of concurrent communications at ports $a$ and $c$ followed by a communication at port $b$. It is a behavior that requires minimal buffering: the input values are received just before they are used to compute the output values.

We introduce two variables, $av$ and $cv$, to receive the input values at ports $a$ and $c$. Value $b(i)$ is recorded in variable $bv$; more precisely, we maintain $i \geq N \;\Rightarrow\; bv = b(i{-}1)$ as an invariant:

$$\begin{array}{l} (\textbf{in } a, c\text{: type, } \textbf{out } b\text{: nat}) \cdot \\ [\![ av, cv\text{: type, } bv, i\text{: nat; } i := 0 \\ \;;(a?av\,,\, c?cv\ \{av = a(i) \wedge cv = c(i)\} \\ \;\;;\textbf{if } i < N{-}1 \rightarrow \text{skip} \\ \;\;\;[\!] \; i = N{-}1 \rightarrow bv := 0\ \{bv = b(i)\} \\ \;\;\;[\!] \; i > N{-}1 \rightarrow \textbf{if } cv \neq av \rightarrow bv := 0 \\ \qquad\qquad\qquad\qquad [\!] \; cv = av \rightarrow bv := bv{+}1 \\ \qquad\qquad\qquad\qquad \textbf{fi}\ \{bv = b(i)\} \\ \;\;\textbf{fi} \\ \;\;;b!bv\,,\, i := i{+}1\ \{i \geq N \;\Rightarrow\; bv = b(i{-}1)\} \\ \;)^* \\ ]\!] \end{array}$$

where 'type' denotes the type of the input stream.

Variable $i$ may be eliminated at the expense of introducing an $N$-counter:

$$\begin{array}{l} (\textbf{in } a, c\text{: type, } \textbf{out } b\text{: nat}) \cdot \\ [\![ av, cv\text{: type, } bv\text{: nat; } bv := 0 \end{array}$$

$$
\begin{array}{l}
;(a?av\,,\ c?cv\ ;b!bv)^N \\
;(a?av\,,\ c?cv \\
\ \ ;\textbf{if}\ cv \neq av \rightarrow bv := 0 \\
\ \ \ \ [\!]\ cv = av \rightarrow bv := bv{+}1 \\
\ \ \textbf{fi} \\
\ \ ;b!bv \\
\ \ )^* \\
[\!]
\end{array}
$$

The solution above may be changed into one for the original problem (with boolean output port $b$) by replacing output statement $b!bv$ by $b!(bv \geq N)$. Moreover, variable $bv$ can be bounded to $0 \leq bv \leq N$:

$$
\begin{array}{l}
(\textbf{in}\ a, c\colon \text{type},\ \textbf{out}\ b\colon \text{bool})\ \cdot \\
[\!]av, cv\colon \text{type},\ bv\colon \text{nat};\ bv := 0 \\
\ \ ;(a?av\,,\ c?cv\ ;b!bv)^N \\
\ \ ;(a?av\,,\ c?cv \\
\ \ \ \ ;\textbf{if}\ cv \neq av \rightarrow bv := 0 \\
\ \ \ \ \ \ [\!]\ cv = av \rightarrow bv := (bv{+}1)\downarrow N \\
\ \ \ \ \textbf{fi} \\
\ \ \ \ ;b!(bv = N) \\
\ \ \ \ )^* \\
[\!]
\end{array}
$$

The process we have designed has two input ports, $a$ and $c$, along which it concurrently receives two values of the input stream: values $A(i)$ and $A(i{-}N)$. Given that there is only a single input stream $A$, this may be accomplished by employing an $N$-place shift register. The input streams to the process then have to be tapped from the front end and the back end of the shift register.

Actually, the program can be simplified even further. Since the output values are specified for $i \geq 2N{-}1$ only, the $N$-fold repetition

$$(a?av\,,\ c?cv\ ;b!bv)^N$$

may be dropped: either $c(i) \neq a(i)$ for some $i$, $0 \leq i < 2N{-}1$, in which case $bv$ is appropriately set to 0, or equality holds for all such $i$, in which case $bv$ becomes $N$, as required.

The process we have designed is surprising in its simplicity. The main reason for its discussion, however, is that it is a rather simple example in which the use of input/output

relations, a technique stemming from parallel programming [2], is effectively combined with the use of invariants.

It is a genuine pleasure to include this problem in the liber amicorum for Frans Kruseman Aretz. His lectures that I attended in the late sixties as a mathematics student in Amsterdam were decisive in arousing my interest in computing. The clearly structured and precise way in which Frans Kruseman Aretz is used to deliver his lectures has never failed to fascinate me.

[1 ] Kees van Berkel. *Handshake Circuits: An Asynchronous Architecture for VLSI Programming.* Cambridge International Series on Parallel Computation 5, Cambridge University Press, 1993.

[2 ] Martin Rem. *Lecture Notes in Parallel Computing.* Technical report, Eindhoven University of Technology, 1994.

# Over universiteit en bedrijfsleven of drie keer het Lange Voorhout

R.P. van de Riet

## 1 Drie redenen

Er waren drie redenen om dit verhaal onder deze titel te schrijven:

1. Een gesprek met een Universitair Docent van de Technische Universiteit Eindhoven (TUE) over een zekere druk die hij ondervond van zijn omgeving om meer contacten met het bedrijfsleven te leggen;
2. een diner in Hotel Des Indes dat gegeven werd om afscheid te nemen van een tweetal leden van het Gebiedsbestuur Exacte Wetenschappen van NWO (GBE) en
3. de vraag om een bijdrage te leveren aan een boekje dat aangeboden zou worden aan Frans Kruseman Aretz, bij zijn afscheid van de TUE, brachten me op het idee "Lange Voorhout" en "Bedrijfsleven".

Drie keer in mijn (ook al weer lange) leven ben ik op het Lange Voorhout geweest, drie keer had het bezoek iets te maken met mijn werk en contacten met het bedrijfsleven. Een keer was Frans er direct bij betrokken. De twee andere keren had hij er slechts indirect mee te maken.

## 2 Waarom contacten met het bedrijfsleven?

Wanneer ik over deze vraag met studenten praat vertel ik ze over mijn beginjaren op het Mathematisch Centrum (MC) (nu Centrum voor Wiskunde en Informatica (CWI)) toen ik mijn wiskundige kennis moest aanwenden om problemen die in de praktijk bestonden op te lossen.

Het eerste probleem was dat van de watertoren die een afwijkende vorm moest krijgen. De dikte van de stalen wand moest worden berekend zó dat het staal voldoende sterk was om de spanning veroorzaakt door het gewicht van het water en dat van de wand zelf op te vangen. De bijbehorende (gewone) differentiaalvergelijking, ik meen van de vierde graad,

moest worden opgelost. Ik heb daarvoor een computerprogramma geschreven. Voor alle zekerheid liet ik de zaak twee keer laten uitrekenen, een keer met differentiemethoden en een tweede keer met polynomen. De ingenieur van de Gemeente Waterleidingen Amsterdam was tevreden. De toren werd daarop gebouwd en de dikte van het staal bleek voldoende om de toren tot op de dag van vandaag te laten staan.

Hij staat aan de Amstelveense weg en ik kijk er nog steeds met enige voldoening naar wanneer ik er langs kom. Overigens, dat hij er nog staat komt natuurlijk ook omdat de ingenieur de wand 2 millimeter dikker maakte dan ik had berekend: "dat was voor de roestvorming" zei hij.

Later bleek dat ik voor de berekening geen computer nodig had gehad; ik werd daar op gewezen door een curator van het MC die NB mijn rapportje over de zaak gelezen had. De curator was niemand minder dan prof. W.R.Koiter, destijds een expert op het gebied van mechanica. Hij had het met de rekenliniaal nagerekend. Gelukkig klopte mijn berekening met de zijne.

Later heb ik voor hetzelfde bedrijf een programma gemaakt dat waterslag berekende in een configuratie van buizen waarmee water van de rivier de Lek naar de Waterleidingduinen vervoerd werd. Die berekeningen leidden tot aanzienlijke besparingen. Hetzelfde programma is door ingenieursbureau Comprimo overgenomen en gebruikt om berekeningen aan oliepijpleidingen uit te voeren.

Een tamelijk intensieve samenwerking kwam tot stand met een onderzoeker van het Research Laboratorium van Unilever: Dr. M. van den Tempel. Met hem heb ik golven bestudeerd op het oppervlak van water die gedempt werden door zeep (dit was dertig jaar voor Omo Power!).

Dit onderzoek leidde tot een paar publicaties en indirect tot mijn proefschrift. Het ging er om fenomenen te bestuderen die zich op een bewegend oppervlak afspelen. Dan praat je over partiële differentiaalvergelijkingen in meerdere dimensies van hogere orde, niet lineair. Kortom moeilijk. Voor mij een inspiratiebron om een fomulemanipulatiesysteem voor de computer te maken dat geschreven was in ALGOL60 en mij hielp met de complexe analyses. Een soort Mathematica avant la lettre.

Dit systeem kon ik later gebruiken voor een moeilijke berekening voor het Nationaal Laboratorium voor Luchtvaart, nu Nationaal Laboratorium voor Luchtvaart en Ruimtevaart. Hier betrof het een vraag over de vorm van een schokgolf veroorzaakt door een rond stomp voorwerp, zoals de neus van een vliegtuig. In dit geval werkte mijn systeem zo dat er eerst een wiskundige analyse werd uitgevoerd betreffende de differentiaalvergelijkingen en dat daarna een programma (weer in ALGOL60) werd gegenereerd dat zelf de gevraagde berekening uitvoerde. In zekere zin een compiler, met als invoer de differentiaalvergelijkingen en als uitvoer een programma.Het is wel duidelijk waarom ik dit allemaal vertel: zonder de inspiratie van het (toenmalig) bedrijfsleven was mijn wetenschappelijke loopbaan anders verlopen. Een zeer sterke aanbeveling dus om positief te spreken over de invloed van het bedrijfsleven op wetenschappelijk onderzoek.

## 3 De eerste keer Lange Voorhout

In die tijd, het waren de eind zestiger jaren, had de Rekenafdeling van het MC ook andere contacten met het bedrijfsleven. Ik was toen Frans opgevolgd als sous-chef van die afdeling. Prof. van Wijngaarden was de chef, maar hij was ook directeur van het MC én leider van het ALGOL68 project. Hij liet dus veel aan zijn sous-chefs over.

De firma Philips had Electrologica over genomen en was dus verantwoordelijk voor de computer die bij het MC stond de EL-X8. In een gemeenschappelijk project tussen Philips' Natuurkundig Laboratorium, de nieuwe werkgever van Frans, de Rijksuniversiteit Utrecht, met Sietse van der Meulen en het MC, waar ik de verantwoording droeg, werd voor de X8 een nieuw bedrijfssysteem gemaakt. Aanvankelijk was het doel een time-sharing systeem; helaas is dat om allerlei redenen de mist in gegaan. Wellicht dat een van die redenen was dat Frans inmiddels (op zijn eentje) een bedrijfssysteem gemaakt had (het Millisysteem) dat meerdere (batch) stromen kende, en hoewel geen time-sharing faciliteiten leverde, in de praktijk, waar de enige programmeertaal ALGOL60 was, zeer goed voldeed.

De samenwerking betrof meer dan het maken van een nieuw bedrijfssysteem: er werd ook een nieuwe terminal ontworpen, die Olivetti mocht leveren. Met zijn drieën (Frans, Sietse en ik) gingen we naar Den Haag, de plaats van het hoofdkantoor van Olivetti. Daar bespraken we met de heer E.J.Bosman, onze tegenspeler bij Olivetti, zeer wetenschappelijke onderwerpen, zoals de onderlinge verhouding van de "." en de "," van het ";" symbool.

Van nog grotere importantie was de vorm en plaats van het "lage tientje". Dit symbool was door ons heilig verklaard. Olivetti zou koste wat het kost ons terminals leveren met dit symbool. Kennelijk was dit moeilijk want het heeft maanden geduurd voor we tevreden waren. Waarom dit symbool voor ons van zo'n groot gewicht was is tegenwoordig moeilijk meer uit te leggen. Het had iets te maken met beroepseer of geloofsijver. Wij waren erg verknocht aan ALGOL60 (N.B. in een tijd dat hard gewerkt werd aan ALGOL68). In ALGOL60 gebruik je een "laag tientje"dat er aldus uitziet: $_{10}$. Om een getal te noteren met een exponent; tegenwoordig gebruikt men hiervoor, in navolging van Pascal (de opvolger van ALGOL60) de letter "e".

Toen ik onlangs de heer Bosman tegen kwam, hebben we er nog smakelijk om gelachen. Wetenschappers en het bedrijfsleven: het heeft Olivetti een aardige duit gekost en weinig opgeleverd. Of wij er wetenschappelijk iets mee gewonnen hebben?

Mijn eerste reden om aan het Lange Voorhout te denken is Frans inmiddels bekend: hij weet dat Olivetti indertijd aldaar resideerde.

## 4 De tweede keer Lange Voorhout

Het was gedurende de begin zeventiger jaren dat een paar informatici een club oprichtten om hun expertise aan te bieden ten behoeve van mensen in ontwikkelingslanden. Idealisten

waren het die dachten dat contacten zo maar gelegd konden worden.

Ik was de voorzitter en Vlietstra de secretaris. Vlietstra werkte bij Philips in Eindhoven, een collega van Frans en representant van het bedrijfsleven dus. We hebben samen heel wat beleefd. Zo hadden we het plan opgevat om met name contacten in Indonesië te leggen. Er was een fraaie folder gemaakt die aangaf op welke gebieden we bereid en in staat waren de helpende hand te bieden. Het maken van de folder had veel voeten in de aarde. We hadden natuurlijk geen geld dus het drukken was een probleem. Vlietstra kon dit bij Philips laten doen (het was de tijd voor Centurion!). De bedoeling was dat die folder zou worden verspreid door de ambassades.

Pas later kwamen we er achter dat ambassades zich daar niet voor lenen. Wij zijn dus blijven zitten met stapels folders. Om ons zo goed mogelijk te oriënteren hebben Vlietstra en ik ook een bezoek gebracht bij Prins Claus, die toen bivakkeerde in het Paleis Noordeinde aan het Lange Voorhout. Inderdaad de tweede keer dat ik daar was.

Omdat we al om acht uur 's ochtends besteld waren bij de Prins brachten we de nacht in een hotel door (niet Des Indes, dat was te duur). Z.K.H. Prins Claus luisterde geïnteresseerd naar wat we te vertellen hadden, en gaf ons het dringende advies onze zo goed bedoelde acties te stoppen, we zouden anders zeker gefrustreerd raken. Feitelijk was dat het soort advies wat we niet verwacht hadden, maar wat toch erg reëel was. Niet lang daarna is de club ook gestopt.

Eigenlijk is de club als een nachtkaarsje uitgegaan: op onze vergaderingen kwamen op een goed moment alleen nog Vlietstra en ikzelf.

Een en ander klinkt allemaal wat negatief, maar er zijn ook positieve wapenfeiten te melden.

Ik noem een brandbrief die we van iemand uit Nigeria kregen die een door IBM gemaakte offerte wilde toetsen; wij konden de expertise van het (toenmalige) Rijks Kantoormachine Centrale (KMC) inschakelen. Ook heb ik rond nieuwjaarsdag een bestelling gedaan van een twintigtal Informatica-boeken bij de VU Boekhandel ten behoeve van een collega in Islamabad die wel geld zei te hebben, maar geen boekhandel in de buurt had. Een garantie voor de aankoop van de boeken kreeg ik vrij gemakkelijk na een telefoontje naar het ministerie van Buitenlandse Zaken. Ook kreeg ik een prima contact met de heer Suwarso, een wetenschappelijk onderlegd zakenman in Indonesië. Hij was oprichter en directeur van Pandata, de grootste software- en infomatica- onderneming in dat land.

Door dat contact is er een cursus door onze PTT georganiseerd voor een twintigtal Indonesische technici. Op uitnodiging van Suwarso ben ik later in Jakarta geweest en heb daar lezingen gegeven over vijfde generatie computers en kennisbanken. Prachtig (voor mij maar ook voor hen?).

Ook hier contacten met het bedrijfsleven: Philips, Pandata, anderen. Goed, profijtelijk? Wie zal het zeggen.

## 5 De derde keer Lange Voorhout

Ik kom bij mijn derde keer Lange Voorhout. Ook nu weer contacten met het bedrijfsleven. De aanloop is nu een beetje langer.

We beginnen met het ontstaan van Stichting Informatica-onderzoek (SION) ergens begin 1980. Daarbij speelde Frans ook een belangrijke rol: er was sprake van een brief getiteld: "voor wie het aangaat". Ik zal het daar nu niet over hebben; wel over de contacten met het bedrijfsleven die door SION ontstonden.

Op een van de eerste bijeenkomsten van de Informatica Sociëteit, die bij het MC gehouden werd, en waar mensen van het informatica-bedrijfsleven, van de ministeries van Onderwijs en Economische Zaken en vanuit de Universiteiten bijeen kwamen, werd gediscussieerd over de relatie tussen industrie en universiteiten op informaticagebied.

Ik herinner me nog goed hoe aan de ene kant de oprichter van Volmac, van Oosterom, de stelling verdedigde dat als iemand in zijn zaak een goed idee zou publiceren dat dat een reden voor ontslag was, terwijl ik naar voren bracht dat een wetenschapper in zijn carrière juist veel moet publiceren om omhoog te komen. (en dat is nog steeds het geval).

Na afloop van de vergadering bracht van Oosterom me met zijn fraaie Mercedes naar huis waar hij me vertelde dat hij de enige was in zijn bedrijf die onderzoek deed. Personeelsleden hadden daarvoor geen tijd; zij behoorden te werken en geld te verdienen. Hij was ook de enige die tijdschriften op informaticagebied las. Inmiddels was mij duidelijk geworden dat wat de een onder onderzoek verstaat dit voor een ander geheel iets anders kan zijn.

Een andere relatie met het bedrijfsleven was de serieuze poging de jaarlijkse SION conferentie te laten samenvallen met het door het NGI georganiseerde congres, waar vooral het bedrijfsleven vertegenwoordigd was. Inderdaad is dit een aantal jaren zo gegaan. Toen we deze gezamenlijke conferentie in het Amsterdamse Concertgebouw hielden mocht ik een openingstoespraak houden. Sindsdien kan ik spreken over "mijn optreden op het podium van het Concertgebouw". Helaas (wellicht is "gelukkig" hier beter) was het orgel van Maarschalkerweerd toen in zo'n deplorabele staat dat ik er niet op kon spelen. Aan de samenwerking met het bedrijfsleven door deze gezamenlijke conferentie kwam een eind. De aanleiding was een financieel probleem, mede veroorzaakt door weinig belangstelling vanuit het bedrijfsleven. In een bestuursvergadering van SION werd over de samenwerking gesproken. Het was een collega van Frans, Dr. Henk Bosma, van het Natuurkundig Laboratorium van Philips, die de doorslaggevende reden aangaf: de wereld van het bedrijfsleven was toch erg ver verwijderd van die van de wetenschappers.

Na zo'n acht jaar voorzitter te zijn geweest van SION, al die tijd in oprichting, werd ik gevraagd zitting te nemen in het prestigieuze Gebiedsbestuur Exacte Wetenschappen (GBE) van NWO. Naast zeer gerenommeerde scheikundigen, natuurkundigen, een wiskundige, een sterrekundige en een ruimteonderzoekdeskundige mocht ook een informaticus mee doen met het bepalen van wetenschapsbeleid. Op de een of andere manier was Informatica toch een beetje volwassen geworden en mocht meespelen op de eerste rij van NWO

waar het gaat om het verdelen van de financiële middelen.

Uiteraard lag het onderwerp: "contacten met het bedrijfsleven" vaak op tafel. Bij NWO is de laatste jaren het idee gegroeid dat NWO door de politiek alleen beoordeeld wordt op onderzoek dat een maatschappelijk draagvlak heeft. Extra subsidies van Economische Zaken worden dan in het vooruitzicht gesteld. Het Gebiedsbestuur Technische Wetenschappen (STW) speelt hier zeer goed op in. De onlangs uitgegeven publicatie van NWO: "Kennis verrijkt" laat ook zien dat het NWO ernst is als het gaat om maatschappelijk relevant onderzoek. Ook het programma Prioriteit wordt door NWO als maatschappelijk bewuste vlag gebruikt.

Het GBE deelt dit standpunt van het Algemeen Bestuur van NWO; men vindt echter dat er toch altijd een grote plaats moet worden ingeruimd voor onderzoek waarvan het nut (voorlopig) volstrekt onduidelijk is. Er zijn vele voorbeelden in de geschiedenis aan te wijzen waar dat nut pas veel later duidelijk werd. Ik geef er hier twee minder bekende voorbeelden van:

Een eerste voorbeeld is het onderzoek dat Charles Babbage honderdvijftig jaar geleden uitvoerde betreffende de constructie van de Analytical Engine. Een constructie waarvan anderhalve eeuw geleden alleen de instrumentmakers profiteerden, vanwege de grote precisie die deze constructie vereiste. Nu weten we dat de Analytical Engine een echte voorloper van de computer was.

Als tweede voorbeeld het volgende: in de wiskunde is het gebied van de getallentheorie altijd als esoterisch beschouwd. Wie is er nu geïnteresseerd in de wetenschap dat het aantal priemgetallen kleiner dan n evenredig is met de logaritme van n. De bekende getaltheoreticus E. Landau verwijst in zijn boek "Elementare Zahlentheorie" naar iemand anders (Gordan) die had gezegd: "Die Zahlentheorie ist nützlich, weil man nämlich mit ihr promovieren kann", en voegde hij er aan toe: "Ich habe mit einer Antwort auf diese Frage 1899 promoviert". Momenteel worden diezelfde priemgetallen echter als zeer nuttig ervaren en er zijn vele industrietjes die goed geld aan priemgetallen verdienen door hun nuttig gebruik in de cryptografie. De aangehaalde stelling zegt dat de kans dat een getal priem is snel kleiner wordt naarmate het groter wordt. Belangrijk voor een inbreker die via raden probeert in te breken in een op RSA gebaseerd cryptosysteem.

Tegen de tendens om teveel de knieën te buigen voor maatschappelijk nut werd door de GBE ook vaak geprotesteerd. Het is voor Nederland en de toekomst van zijn wetenschap buitengewoon belangrijk dat er ook ruimte is voor het vrije onderzoek waarbij het minder belangrijk is of het bedrijfsleven er snel van kan profiteren. Deze stelling heeft het GBE verdedigd bij NWO's Algemeen Bestuur. Ook schroomde het GBE niet sommige van zijn Stichtingen er op te wijzen dat hun zucht om opdrachten van het bedrijfsleven binnen te halen ook te ver kan gaan. In dat GBE zat ook iemand van het bedrijfsleven, in dit geval niet van Philips maar van Shell, die in deze discussies vele stenen bijdroeg en bepaald niet achter liep als het ging om het signaleren van genoemde gevaren. Zes jaar mocht ik in dit illustere gezelschap verkeren, daarna moest ik aftreden. Ons, de man van Shell en ikzelf, werd toen een afscheidsdiner aangeboden in Hotel Des Indes aan het Lange Voorhout,

inderdaad de derde keer.

## 6 Lessen!

Zijn er lessen te leren uit dit verhaal? Wat is mijn advies aan de eerder genoemde UD van de TUE?

Het is verleidelijk om extreme standpunten in te nemen, bijvoorbeeld

A: onderzoek, zonder dat het bedrijfsleven er achter staat, is niet de moeite waard en zou niet uitgevoerd moeten worden. De belasting-betaler heeft er recht op dat geld voor onderzoek nuttig besteed wordt en niet aan onderzoekers uitgegeven wordt die het verschil niet kennen tussen werk en hobby.

Of het andere extremum:

B: Het bedrijfsleven is alleen geïnteresseerd in korte termijn problemen. In onderzoek dat echt wetenschappelijk en publicabel is is het bedrijfsleven dus niet geïnteresseerd, het is derhalve zonde van de tijd en inspanning om contacten proberen te leggen.

In het voorgaande heb ik diverse voorbeelden gegeven die pleiten voor een van beide standpunten. Mijn eerste ervaringen op het MC pleiten sterk voor standpunt A. Dankzij het onderzoek ten behoeve van Unilever ben ik immers gepromoveerd.

Mijn ervaringen als voorzitter van SION wijzen daarentegen meer in de richting standpunt B. Voor B pleiten ook diverse ervaringen met afstuderende studenten die graag bij het bedrijfsleven een stage willen volgen, waarvan wij eisen dat die wetenschappelijk interessant moet zijn. Al te vaak blijkt dan dat men binnen het bedrijf alleen een student als goedkope programmeerkracht ziet. Wanneer door ons geëist wordt dat ook wetenschappelijke literatuur bestudeerd moet worden is men niet thuis (niet op de hoogte). Deze ervaring heb ik niet alleen gehad met kleine bedrijfjes, maar ook met grote (laten we geen namen noemen).

De balans slaat voor mij echter duidelijk in de richting van A door en wel om twee redenen.

1. In de eerste plaats een principiële: Beoefening van het vak informatica houdt in mijn visie in dat je voortdurend na moet denken over de toepassingen. Ik beweer steeds dat informatici drie zaken moeten kunnen: iets maken, dat kan zijn een ontwerp van een informatiesysteem, of een programma, of een stuk hardware gekoppeld met een programma, ze moeten na kunnen denken over dat wat ze gemaakt hebben, en, in de derde plaats, ze moeten datgene wat gemaakt is kunnen toepassen. Bij dit laatste is het natuurlijk erg verstandig om via het bedrijfsleven te zien waar toepassingen nuttig zijn. In het bijzonder waar de informatica via Cyberspace nu zo'n grote impact gaat krijgen op het "gewone" leven is die band met de toepassingen erg belangrijk.

2. In de tweede plaats omdat ik ook veel positieve ervaring met bedrijven heb. Daarvan nog twee voorbeelden. Met het GAK (Gemeenschappelijk Administratie Kantoor) bestaat nu al weer vele jaren een zeer goede relatie doordat de heer Chang, die vanuit het GAK de contacten met de Universiteiten regelt op buitengewone wijze weet in te spelen op onze (wetenschappelijke) interesses. Feitelijk blijken dat vaak ook zijn interesses te zijn. Zo ontstaan afstudeerprojecten waar follow-up voor ons in zit en voor het GAK. Diverse projecten blijken zeer nuttig te zijn voor het GAK; bovendien worden veel studenten als nieuwe medewerkers gerecruteerd, terwijl anderzijds papers en soms proefschriften het gevolg kunnen zijn van die stages.
Een tweede voorbeeld is een contact met het multi-nationale accountantsfirma Deloitte & Touche. Ongeveer vijf jaar geleden kwamen ze bij mij om advies. Het betrof een systeem waarmee de computer de accountant helpt om zijn contrôlewerk te plannen, en er over te rapporteren. Er zou een kennisbank moeten worden opgezet met kennis over het werk van accountants, kennis over de bedrijfstakken en kennis over de specifieke klanten. Een project dat met twee partners in Engeland en de Verenigde Staten zou worden uitgevoerd. De vraag was of ik kon helpen met het opzetten van een (wiskundig) model, zo dat vragen over het systeem in een wetenschappelijk kader konden worden gesteld.
Gelukkig was er iemand die echt aan het model kon en wilde werken, Philip Elsas, een employé van het bedrijf die bij ons als informaticus was afgestudeerd en nu het plan had opgevat te gaan promoveren. Een ideale situatie waar ik meteen enthousiast over werd. Inmiddels is het proefschrift vrijwel klaar, zijn een viertal studenten afgestudeerd bij dit bedrijf op onderwerpen die nauw samen hingen met het project, zijn twee van hen aangenomen als employé en is van het te maken systeem inmiddels al een tweede release wereldwijd verspreid en gebruikt. Mogelijkheden voor verdere samenwerking worden momenteel onderzocht.

## 7 Conclusie

In de voorbeelden die ik gegeven heb is wel een patroon te herkennen: met bedrijven die naar mij toe kwamen kon ik een samenwerking opzetten die resulteerde in succes hetgeen dus pleit voor standpunt A; met bedrijven waar ikzelf op af ging liep de samenwerking meestal op een teleurstelling uit: standpunt B.

Daarom zou ik het volgende advies aan de UD bij de TUE willen geven: zorg er als wetenschapper voor dat bedrijven in jouw werk geïnteresseerd zijn zodat ze naar jou toekomen; de kans is groot dat daar contacten uit groeien die voor beide partijen profijtelijk zijn.

## 8 Slot

Zo kom ik aan het eind van dit aan Frans opgedragen verhaal, waarin ik drie keer het Lange Voorhout aandeed, toch nog tot een boodschap. Omdat jij Frans sinds je tijd bij het MC steeds met één been in de Universiteit en met de vier andere in het bedrijfsleven hebt gezeten, denk ik dat de problematiek je zal aanspreken. Dat je er inmiddels niet meer over hoeft na te denken is een geluk voor jou. Bij je afscheid van die Universiteit wens ik jou en Loes nog vele plezierige jaren toe.

# Frans, bedankt!

Huub Schols

Vele jaren heb ik met jou kontakt mogen hebben aan de Technische Universiteit Eindhoven, aanvankelijk als student daarna als promovendus en collega. Onze interaktie heb ik altijd erg op prijs gesteld ; ik ben ervan overtuigd dat ik de afgelopen jaren veel van je heb geleerd. Ik wil bij deze gelegenheid één aspect benadrukken.

Je hebt een duidelijke visie met betrekking tot het structureren van manuscripten. Aan deze visie heb ik mijn inzichten mogen toetsen, zowel bij het maken van mijn afstudeerverslag als bij het maken van mijn proefschrift. Met name in die gevallen waarin de inhoud van het manuscript geaccepteerd was en bovendien de struktuur zelf gebalanceerd was, wist jij mij te confronteren met de wisselwerking tussen deze twee aspecten. Je hebt de intuïtie om de vinger te leggen op een detail en om vervolgens via de struktuur van het manuscript op een andere plaats een onjuiste of ambigue formulering te vinden. Ik heb genoten en veel geleerd van de sessies die wij samen al puzzelend hebben doorgebracht. Met name mijn proefschrift is hierdoor uiteindelijk verschenen in een vorm waarover ik zelf tevreden was.

Frans, ik ben blij jou als één van mijn leermeesters aan de Technische Universiteit Eindhoven gehad te hebben.

***Paradise regained?***

*De wereld is volmaakt in orde:*
*geen dijk waaraan je kunt gezet,*
*geen laan om uitgestuurd te worden,*
*geen rem die je een stunt belet.*

*Geen Maarten om de pijp te geven,*
*pas nu begint het rijke leven!*

# One formula in two, grammar-free, contexts

Dedicated to Frans Kruseman Aretz

Fred Steutel

**Abstract.** Spitzer's well-known identity in random walk theory has exactly the same structure as the canonical representation of infinitely divisible Laplace transforms and probability generating functions. This 'coincidence' implies that the waiting time of the $N+1$-st customer in an $G|G|1$-queue is infinitely divisible, if $N$ has a geometric distribution on $\mathbb{Z}_+$, and is independent of the queueing process.

## 0. Introduction and summary

It is well known that random sums of the form

$$S_N = X_1 + ... + X_N$$

are infinitely divisible (inf div) if $X_1, X_2, ...$ are i.i.d. and $N$ is geometrically distributed on $\{0, 1, ...\}$ and independent of $(X_n)_1^\infty$.
In this note it will appear that from Spitzer's identity it follows that

$$W_{N+1} = W_1 + W_2 - W_1 + ... + W_{N+1} - W_N$$

is inf div, where $W_n$ denotes the waiting time of the $n$-th customer in a $G|G|1$-queue, and $N$ is independent of $(W_n)$; here, however the $W_{n+1} - W_n$ are *dependent*, and do *not* have the same distribution. In Section 1, Spitzer's identity is given with some necessary context, Section 2 contains basic facts on inf div distributions, and in Section 3 the fact that both ingredients have the same structure is exposed and used. Section 4 gives some

additional remarks.
In what follows $F$, with or without suffix, will denote a distribution function, and $\widehat{F}$ its Laplace-Stieltjes transform (LSt).

## 1. Spitzer's identity

In the well-known $G|G|1$-queueing system customers arrrive at times $0, A_1, A_1+A_2, \ldots$ and are served during periods $B_1, B_2, \ldots$; all $A$'s and $B$'s are independent. We write $S_0 = 0$,

$$S_n = \sum_{k=1}^{n} (B_k - A_k), \ n = 1, 2, \ldots ,$$

and $S_k^+ = \max(0, S_k)$, $k = 1, 2, \ldots$. If it is assumed that the first customer finds the server free, then the waiting time $W_n$ of the $n$-th customer is given by $W_1 = 0$, and

$$W_{n+1} \stackrel{d}{=} \max(S_0, S_1, \ldots, S_n), \ n = 1, 2, \ldots . \tag{1.1}$$

Now, Spitzer's identity (Loève (1977)) reads, for $|z| < 1$ and $Re\ s \geq 0$,

$$\sum_{n=0}^{\infty} Ee^{-sW_{n+1}} z^n = \exp\left\{ \sum_{k=1}^{\infty} \frac{1}{k} Ee^{-sS_k^+} z^k \right\} . \tag{1.2}$$

## 2. Infinite divisibility

A random variable $X$ is called inf div if for every $n \in \mathbb{N}$ one has

$$X \stackrel{d}{=} X_{1,n} + \ldots + X_{n,n} ,$$

where the $X_{j,n}$ are iid. We only need the following results (Feller (1971), Steutel (1970)).

**Lemma 1.** A nonnegative random variable is inf div if and only if it has a LSt of the form

$$\widehat{F}(s) = Ee^{-sX} = \exp\left\{ \int_0^{\infty} \frac{e^{-sx} - 1}{x} dK(x) \right\} , \tag{2.1}$$

where $K$ is a nondecreasing function, which, necessarily, has the property $\int_1^{\infty} x^{-1} dK(x) < \infty$.

**Lemma 2.** A nonnegative, integer-valued random variable $M$ with $P(M = n) = p_n$, and $p_0 > 0$ is inf div if and only if its probability generating function has the form,

$$P(z) := \sum_{n=0}^{\infty} p_n z^n = \exp\left\{\sum_{n=0}^{\infty} \frac{r_n}{n+1}(z^{n+1}-1)\right\}, \tag{2.2}$$

with $r_n \geq 0$, $n = 0, 1, 2, ...$, and, necessarily, $\sum_0^{\infty} r_n/(n+1) < \infty$.

## 3. $W_{N+1}$ is infinitely divisible

We rewrite (1.2) as follows. Put $z = p \in (0,1)$ and multiply by $(1-p)$; this yields (use $-\sum_1^{\infty} p^k/k = \log(1-p)$),

$$\sum_{n=0}^{\infty}(1-p)p^n E e^{-sW_{n+1}} = \exp\left\{\sum_{k=1}^{\infty}\frac{p^k}{k}(Ee^{-sS_k^+}-1)\right\}. \tag{3.1}$$

The left-hand side of the equation above is equal to $E\exp(-sW_{N+1})$, where $N$ is independent of $(W_n)_1^{\infty}$, and

$$P(N=n) = (1-p)p^n \quad (n = 0, 1, ..., n). \tag{3.2}$$

The right-hand side can be rewritten as (cf. (2.1))

$$\exp\left\{\int_0^{\infty}\frac{e^{-sx}-1}{x}dK(x)\right\},$$

with $K$ given by

$$K(x) = \sum_{k=1}^{\infty}\frac{p^k}{k}\int_0^x y\, dF_{S_k^+}(y) \tag{3.3}$$

Combining the results above we obtain the main result of this note.

**Theorem.** If $W_n$ is the waiting time of the $n$-th customer in a $G|G|1$-queue, started empty, and $N$ is a rv independent of $(W_n)_1^{\infty}$ satisfying (3.2), then the rv $W_{N+1}$ is infinitely divisible.

It should be pointed out that for fixed $n$ the $W_n$ are in general not inf div, since they are bounded if the $B$'s are bounded; if $W_n \xrightarrow{d} W$ as $n \to \infty$, then $W$ is inf div (see end of Section 4).

## 4. Further remarks

Spitzer's identity (1.2) can also be related to Lemma 2. Apart from a multiplicative constant in the right-hand side, (1.2) is exactly of the form (2.2) with

$$p_n = p_n(s) = Ee^{-sW_{n+1}};\ r_n = Ee^{-S^+_{n+1}}\ .$$

So we see that, for every $s$, the sequence $(Ee^{-sW_{n+1}})_0^\infty$ is an infinitely divisible sequence (not necessarily summing to 1).
Another result follows if we take logarithms in (1.2) and (2.2), and differentiate. Equation (2.2) then yields

$$np_n = \sum_{k=0}^{n-1} p_k r_{n-k-1}\quad (n = 1, 2, ...)\ .$$

Similarly, (1.2) leads to

$$nEe^{-sW_{n+1}} = \sum_{k=0}^{n-1} Ee^{-sW_{k+1}} \cdot Ee^{-sS^+_{n-k}}\ ,$$

or in terms of distribution functions

$$F_{W_{n+1}}(w) = \frac{1}{n}\sum_{k=1}^{n-1}(F_{W_{k+1}} * F_{S^+_{n-k}}\quad (n = 1, 2, ...)\ , \tag{4.1}$$

which is not easily verified directly for $n > 1$. In fact, one can derive (4.1) from (1.1) and obtain a proof of Spitzer's identiy. This was done by Heinrich (1985), who proves (4.1) by induction; in Kotz et al. (1985), where I found the reference to Heinrich, the identity is misquoted: the factor $1/k$ in the right-hand side of (1.2) is missing. Finally, letting $p \uparrow 1$ in (3.1) we obtain the LSt $\widehat{F}_W$ of the limiting waiting time (assume $ES_1 < 0$):

$$\widehat{F}_W(s) = \exp\left\{\int_0^\infty (e^{-sx} - 1)d\sum_1^\infty \frac{1}{k}F_{S_k^+}(x)\right\}\ ,$$

which shows that $W$ is inf div, as is well known.

## References

Feller, W. (1971), An introduction to probability theory and its applications, Vol. 2, 2-nd ed., Wiley, New York, etc.

Heinrich, L. (1985), An elementary proof of Spitzer's identity, Statistics **16**, 249–252.

Kotz, S., Johnson, N.L. and Read, C.B. (1985), Encyclopedia of Statistical Sciences, Vol. 8, Wiley, New York, etc.

Loève, M. (1977), Probability theory 1, Springer-Verlag, New York, etc.

Steutel, F.W. (1970), Preservation of infinite divisibility under mixing, M.C. Tracts nr. 33, Mathematical Centre, Amsterdam.

# A termination and time complexity argument

Jan Tijmen Udding

**Abstract**

Termination of an algorithm is usually obvious. In a few cases, however, it is a challenge to find a correct termination argument. In this note we give an example of such a problem and provide not only an argument for termination but also for the time complexity of the algorithm.

## 1 Introduction

In EWD1197 [1] Edsger W. Dijkstra gives a termination argument for a certain game on a cyclic arrangement of integer variables. The reasoning, though elegant and sufficient to prove termination, does not give any insight in the time complexity of the game, i.e. the number of moves that the game takes. In this little note we define a variant function that is bounded below and that decreases by 1 in each step, thus giving the time complexity of the game. It turns out that the time complexity is independent of the moves chosen.

From EWD1197 we quote:

> Consider a cyclic arrangement of a set of integer variables. Any pair $x$, $y$ of these variables such that $x$ is the clockwise neighbour of $y$ enables a "move" if $x < y$ : the move then consists of adding 1 to $x$ and subtracting 1 from $y$, i.e. in guarded command notation
>
> $$x < y \rightarrow x, y := x + 1, y - 1 \tag{1}$$
>
> A game consists of an initialization of the variables, followed by a sequence of moves until no move is possible.

The paper then continues to argue that the game eventually ends, provided that the average of the variables is an integer. It does not give any insight in the number of steps that such a game would require. That is the problem that we set out to solve in this note.

## 2 The game revisited

We begin by scaling the problem with respect to the average of the given variables. We subtract the average from each of the variables. It should be clear that this does not change the essentials of the game. Phrased differently, we assume that the (integer) variables are such that their sum is 0. This will be the only place where we use the assumption that the average is an integer. Next we cut the ring at an arbitrary position, that is, from now on we assume each variable to have an index between 0 and $N-1$, $N$ being the number of variables, and we shall refer to the values of these variables as $x_i$ for $0 \leq i < N$. Also, we assume indices to be reduced modulo $N$ so as to range between 0 and $N-1$. We choose the clockwise neighbor of $i$ to be $i-1$.

Now a move consists of choosing an index $i$ such that $x_{i-1} < x_i$ and performing the assignment

$$x_{i-1}, x_i := x_{i-1} + 1, x_i - 1. \tag{2}$$

It should be clear that we have truthfully phrased the same game in a setting that we feel is more amenable to the manipulations to follow.

A variant function is an expression made up of the variables of the game, the $x_i$'s in this case, that is bounded below, and that decreases by at least 1 in every step. The existence of such a function does not only prove termination of the game, but also gives insight in the time complexity of the game, especially if it decreases by exactly 1 in each step.

As was argued in EWD1197, the sum of the variables is constant during the game, and hence useless as a variant function. In each move one variable's distance to 0, the average, becomes smaller, whereas the other's distance to 0 might increase. Since in the end all variables are 0, this would give rise to a variant function if the decrease in size could be given a larger weight than the increase. However, a quick inspection yields that if the two numbers involved in a move differ by 1, then there is no way to make this work, since effectively the two values are swapped in this case.

Another way to break the symmetry is to assign each variable $x_i$ its own weight $i$. Then it is clear that the weighted sum $(\Sigma i :: i * x_i)$ decreases by 1 in each move, provided that for a move an index larger than 0 is chosen. If index 0 is chosen then the weighted sum increases by $N-1$ and we still do not have a function that decreases in each step. Yet, had we taken a weighted sum starting at a different point, then that weighted sum would have decreased by 1. That suggests the following variant function.

$$vf = (\mathbf{MAX}\ j :: (\Sigma i :: i * x_{j+i}))$$

The domain of quantifiers such as **MAX** and $\Sigma$ is supposed to be 0 through $N-1$, unless stated differently. In the sequel we show that this function decreases by 1 in each step.

## 3 The analysis

We scale the problem again, now with respect to some weighted sum. Let us define

$$M = (\Sigma i :: i * x_i),$$

A small calculation reveals that the $N$ possible weighted sums can be written as

$$W_j = (\Sigma i :: i * x_{j+i}) = M + N * (\Sigma i : 0 \leq i < j : x_i), \tag{3}$$

for $0 \leq j < N$.

Hence, in order to maximize $W_j$, as in the definition of *vf*, we have to maximize $S_j$, for $0 \leq j < N$, where

$$S_j = (\Sigma i : 0 \leq i < j : x_i).$$

An index for which $S$, or equivalently $W$, assumes its maximum value is referred to as a maximum index. For any maximum index $k$ we observe

$$x_{k-1} \geq 0 \wedge x_k \leq 0. \tag{4}$$

Notice that this also holds in the boundary cases $N = 1$ or $k = 0$, due to the fact that the average and thus $S_N$ equals 0. Let for the rest of this note $k$ be a maximum index at some point in the game.

Now let $j$ be the index of the next move, i.e. $x_{j-1} < x_j$; then the move will increase $x_{j-1}$ by 1 and decrease $x_j$ by 1, according to (2). By (4), it follows that $j \neq k$. Therefore, using that $W_k$ can also be written as the weighted sum $(\Sigma i :: i * x_{k+i})$, according to (3), it follows that $W_k$ decreases by 1. The only question is whether $W$ (or equivalently $S$) still assumes its maximum for $k$, in which case we can conclude that also *vf*, being the maximum of all weighted sums, decreases by 1. We consider the situation right before the move, and discuss the case $j = 0$ separately.

In case $j = 0$, $x_0$ decreases by 1 and $X_{N-1}$ increases by 1. Hence, all $S_i$'s will decrease by 1 except $S_0$, which remains 0. Now we have to show that $S$ still assumes its maximum in $k$ after the move, which means that $S_k$ should be at least 1 before the move. This being a move, i.e. $x_{N-1} < x_0$, and the total sum being 0, we conclude that $N - 1$ is an index where $S$ is at least 1.

In case $j \neq 0$, $S_j$ increases by 1, while for all indices $i$, $i \neq j$, $S_i$ remains the same. Since $j$ cannot be an index for which $S$ assumes its maximum value, on account of (4) and the definition of a move (2), we conclude again that $k$ remains an index for which $S$ assumes its maximum value.

Thus, $vf$ decreases by 1 in each move. Clearly, $vf \geq 0$. Moreover, it should be clear that a move is possible whenever $vf > 0$. Therefore, the number of moves in the game equals

$$(\mathbf{MAX}\ j :: (\Sigma i :: i * x_{j+i}))$$

Not scaling the problem with respect to the average, we have to subtract the sum of all variables from this maximum. Apparently, the order in which the moves are made have no effect on the length of the game.

## 4 Postscriptum

The case analysis $j = 0$ vs. $j \neq 0$ can be avoided by making sure that a move cannot be performed on the pair $x_{N-1}$, $x_0$. If $k$ is a maximum index then it follows from the reasoning above that never a move on the pair $x_{k-1}$, $x_k$ is possible. Therefore, a convenient place to cut the ring is at a maximum index. Unfortunately, I was unable to introduce the concept of a maximum index before cutting the ring.

## References

[1] E.W. Dijkstra. A termination argument. EWD1197, January 1995.

# The lost group chart and related problems

Tom Verhoeff

Dedicated to Prof. Dr. F. E. J. Kruseman Aretz

## 1 Introduction

Last year, Marga and I bought ourselves a new home. This solved our problem of room shortage but created others ...

In the basement of the house is a fuse box, where the main electricity supply is distributed over ten separately fused groups. Each fuse protects a number of the—more than fifty—power outlets in and around the house. For reasons that need not concern us here, the mapping from outlet to fuse group is rather haphazard. Unfortunately, the previous owner had lost the group chart that lists for every outlet the fuse group to which it belongs. Thus, I encountered the problem of reconstructing the lost group chart.

The fuse box in the basement also has ten switches, one for each group. All outlets in a group are connected to the mains supply through the corresponding switch. A voltage tester can be used to determine whether an outlet is 'alive' or not. By suitably toggling switches and testing outlets it should be possible to reconstruct the group chart. Because I like brain teasers better than manual labor, I imposed the additional constraint that the number of switch toggles and outlet tests be minimized.

## 2 Specification

The reconstruction problem can be formulated as a programming exercise in the following way (also see Figure 1). $M$ outlets labeled 1 through $M$ (at $A$) are connected by $M$ wires to $N$ switches labeled 1 through $N$ (at $B$). Each outlet is connected to exactly one of the switches. Each switch can be connected to zero[1] or more wires.

The reconstruction procedure has to determine how the outlets are connected to the switches using the following two kinds of operations. Each switch can be *toggled* from a

[1] This takes into account the possibility of unused fuses.

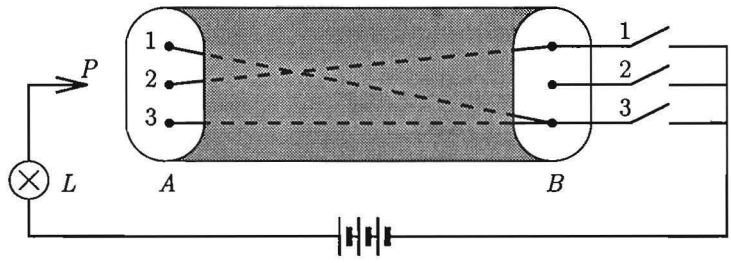



Figure 1: Three outlets connected to three switches ($M = N = 3$)

conducting state to a non-conducting state and vice versa. Initially all switches are, say, non-conducting. An outlet can be *tested* with probe $P$: lamp $L$ will light up if and only if the tested outlet is connected to a conducting switch.

A more formal specification of the programming exercise is given below in a dialect of the programming language Pascal.

```
const
  M: integer = ...; { number of outlets, 1 ≤ M }
  N: integer = ...; { number of switches, 1 ≤ N }

type
  outlet = 1..M;
  switch = 1..N;
  chart = array [outlet] of switch;

const
  f: chart = ...; { the group chart to be reconstructed }

var
  C: set of switch; { the set of conducting switches }

function test(i: outlet): boolean; { return: f[i] ∈ C }

procedure toggle(j: switch); { pre: C = C' ; post: C = C' ÷ [j] }

procedure reconstruct(var g: chart); { pre: C = [] ; post: g = f }
```

Function call $test(i)$ returns whether the lamp lights up when testing outlet $i$. Procedure call $toggle(j)$ changes the state of switch $j$ (operator $\div$ stands for the symmetric set difference). Procedure *reconstruct* is to be programmed using only *test* to inspect $f$ and *toggle* to modify $C$, while minimizing the number of calls of *test* and *toggle*.

## 3 Straightforward Solution

A straightforward design for procedure *reconstruct* determines for each outlet the switch to which that outlet is connected, based on the property

$$C = [j] \wedge \mathit{test}(i) \;\Rightarrow\; f[i] = j$$

Here is the code (the heading of *reconstruct* is not be repeated):

```
var i: outlet; j: switch;
begin
for i := 1 to M do { inv: C = [] ∧ (∀a : 1 ≤ a < i : g[a] = f[a]) }
  for j := 1 to N do begin
    toggle(j) ; { C = [j] }
    if test(i) then g[i] := j ;
    toggle(j)
    end { for j }
end; { reconstruct }
```

This procedure does $2MN$ toggles and $MN$ tests. The average-case performance can be improved by breaking off the inner loop (over $j$) as soon as $g[i]$ has been determined. This would halve the expected number of toggles and tests.

Another improvement is obtained by reordering the two loops and taking the calls to *toggle* outside the inner loop, that is, by determining for each switch all outlets that are connected to that switch:

```
var i: outlet; j: switch;
begin
for j := 1 to N do begin
  { inv: C = [] ∧ (∀i : f[i] < j : g[i] = f[i]) }
  toggle(j) ;
  for i := 1 to M do { inv: C = [j] }
    if test(i) then g[i] := j ;
  toggle(j)
  end { for j }
end; { reconstruct }
```

The procedure now does $2N$ toggles and $MN$ tests. The average-case performance can be improved by not testing outlets that have already been determined. This also removes the need to reset the switches. Furthermore, when $N-1$ switches have been covered the remaining outlets are known to be connected to switch $N$. The following design incorporates these three improvements. It is based on the property

$$C = [1..j] \wedge f[i] \geq j \wedge \mathit{test}(i) \;\Rightarrow\; f[i] = j$$

Below is the—surprisingly compact—code. The number of toggles is now $N-1$, the bare minimum under worst-case conditions. The expected number of tests is $\frac{1}{2}M(N-1)$.

```
var i: outlet; j: switch;
begin
for i := 1 to M do g[i] := N ;
for j := 1 to N−1 do begin
  { inv: C = [1..j−1] ∧ (∀ i :: g[i] = if f[i] < j then f[i] else N) }
  toggle(j) ;
  for i := 1 to M do { inv: C = [1..j] }
    if g[i] = N then { f[i] ≥ j }
      if test(i) then g[i] := j
  end { for j }
end; { reconstruct }
```

## 4 Sophisticated Solution

The total number of possible charts is $N^M$. Because each test provides at most one bit of information, the worst-case minimum number of tests to reconstruct the chart is $\lceil M \log_2 N \rceil$. This analysis can be refined as follows.

The knowledge gathered about $f$ during reconstruction can be captured by stating for each outlet to what *set* of switches it is possibly connected. Let me denote the candidate set for outlet $i$ by $F(i)$. Initially, $F(i) = [1..N]$ for all $i$. If testing outlet $i$ yields *true*, its set of candidate switches is reduced to $F(i) \cap C$. If the test yields *false*, the candidate set reduces to $F(i) - C$. Testing continues until all $F(i)$ are singletons. The best one can hope to accomplish by one test is halving the outlet's candidate set. Therefore, the worst-case minimum number of tests to reduce one candidate set to a singleton is $\lceil \log_2 N \rceil$. This lower bound can be attained by a binary search:

```
procedure BinarySearch(i: outlet); { post: g[i] = f[i] }
var L, R, j, k: switch; p: boolean;
begin
L := 1 ; R := N ; p := false ; { L ≤ f[i] ≤ R, switches [L..R] are p-conducting }
while L ≠ R do begin
  k := (L + R − 1) div 2 ; { L ≤ k ≤ R, #[L..k] ≤ #[k+1..R] }
  for j := L to k do toggle(j) ;
  { switches [L..k] are ¬p-conducting, [k+1..R] are p-conducting }
  if test(i) = p then { f[i] ∈ [k+1..R] } L := k+1
  else { f[i] ∈ [L..k] } begin R := k ; p := not p end
  end { while } ;
g[i] := L
end; { BinarySearch }
```

Note that to reduce the number of *toggle* calls the shorter half of the interval is toggled. Reconstructing the whole chart requires at least $M\lceil \log_2 N \rceil$ tests (under worst-case conditions) and this can indeed be accomplished by combining all $M$ binary searches. For each outlet the *interval* of candidate switches is maintained in an array $h$:

```
var i: switch;
    h: array [outlet] of record hL, hR: switch end;
    { h[i].hL ≤ f[i] ≤ h[i].hR }
begin
for i := 1 to M do begin h[i].hL := 1 ; h[i].hR := N end ;
CBS(1, N, false, M) ;
for i := 1 to M do g[i] := h[i].hL
end; { reconstruct }
```

Recursive procedure *CBS* halves candidate intervals until they are singletons, minimizing the total number of calls to *toggle* and *test*:

```
procedure CBS(L, R: switch; p: boolean; t: integer);
  { pre: L ≤ R, switches [L..R] are p-conducting and service t outlets }
  var i: outlet; j, k: switch; u: integer;
  begin
  if (L ≠ R) and (t > 0) then begin
    k := (L + R − 1) div 2 ;
    for j := L to k do toggle(j) ;
    { switches [L..k] are ¬p-conducting, [k+1..R] are p-conducting }
    u := 0 ;  { switches [L..k] service u outlets in [1..i−1] }
    for i := 1 to M do with h[i] do
      if (hL = L) and (hR = R) then
        if test(i) = p then hL := k+1 ;
        else begin hR := k ; u := u+1 end
    CBS(L, k, not p, u) ; CBS(k+1, R, p, t−u)
    end { if }
  end; { CBS }
```

The following table summarizes the intervals that occur when doing a combined binary search involving ten switches:

| phases | switches → | | | | | | | | | | #toggles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ↓ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | ↓ |
| 1 | • | • | • | • | • | ○ | ○ | ○ | ○ | ○ | 5 |
| 2 | ○ | ○ | • | • | • | • | • | ○ | ○ | ○ | 4 |
| 3 | • | ○ | ○ | • | • | ○ | • | • | ○ | ○ | 4 |
| 4 | | | | ○ | • | | | | • | ○ | 2 |
| #toggles → | 3 | 2 | 2 | 2 | 1 | 2 | 1 | 1 | 1 | 0 | 15 |

For instance, in phase 1, switches [1..5] are made conducting (marked •) and switches [6..10] are kept non-conducting (marked ∘). In total, at most 15 toggles are done. In general the number of toggles is approximately $\frac{1}{2}N \log_2 N$, since in each phase at most half the switches are toggled.

Parameter $t$ and variable $u$ were introduced to suppress superfluous toggles. To understand their effect consider a chart $f$ for ten switches (see table above) where no outlets are connected to switches [1..2]. This will already be detected in phase 2, when invoking $CBS(1, 2, false, 0)$. Without parameter $t$, switch 1 would still be toggled. For the worst-case situation with ten switches, *CBS* either saves a toggle (when no outlet is connected to switches [1..2]) or a test (when some outlet is connected to switches [1..2]) compared to omitting parameter $t$.

## 5 Practical Solution

Let me consider a slightly more general problem, where chart $f$ is a *partial* function from outlets to switches. That is, outlets need not be connected at all (possibly due to broken wires, which is not unrealistic in older houses). The specification is modified as follows:

```
type
  switch' = 0..N;
  chart = array [outlet] of switch';
  { for f: chart, f[i] = 0 means that outlet i is not connected }
```

The earlier solutions can easily be adapted. The following solution is of interest because it is efficient and can be carried out with a minimum of administrative overhead. It is based on writing the switch numbers in binary notation. Each bit is used to determine the switch state in the corresponding test phase. Each test yields one bit of the switch number of the tested outlet. Here is the code:

```
var i: outlet; j: switch; k: integer;
begin
for i := 1 to M do g[i] := 0 ;
k := 1 ; while k ≤ N do k := 2 * k ;
while k ≠ 1 do begin
  k := k div 2 ;
  for j := 1 to N do
    if odd(j div k) ≠ (j in C) then toggle(j) ;
  for i := 1 to M do
    if test(i) then g[i] := g[i] + k
  end { while }
end; { reconstruct }
```

Note that $odd(j \mathbf{div} k)$ yields the bit of weight $k$ in $j$. The number of tests is $M\lceil \log_2(N+1) \rceil$. The number of toggles is slightly more than the binary search method of the preceding section. For practical reasons I have used the binary notation method at home.

## 6 Conclusion

The reconstruction problem (with $M = N \leq 90$) appeared as a programming task at the 7th International Olympiad in Informatics[2] which was held at Eindhoven University of Technology in 1995. There were more than two hundred participants from secondary schools of some fifty countries all over the world. To my surprise ten competitors came up with the intended optimal solution (doing a combined binary search and suppressing superfluous toggles).

The problem has some intriguing variants. I already mentioned the extension from total to partial charts $f$. But what if, for instance, the initial state of the switches is unknown and $C$ may not be inspected? This corresponds to a randomized fuse box where the visible state of the switches does not reveal their conductivity.

What if each wire is also connected to a unique switch on side $A$, allowing multiple wires to be tested? In that case the minimum number of tests (under worst-case conditions) is conjectured to be $\lceil M \log_2 N \rceil$ instead $M\lceil \log_2 N \rceil$. For instance, for $M = N = 3$ there are 27 possible charts. With switches on side $B$ only, at least $3\lceil \log_2 3 \rceil = 6$ tests are required. With additional switches on side $A$, it can be done in $\lceil 3 \log_2 3 \rceil = 5$ tests (try it).

What if chart $f$ is known to be a permutation? This situation arises when recycling cables with indistinguishable wires. Can the reconstruction be done in $\lceil \log_2 N! \rceil$ tests? I have the feeling that this variant is as hard as minimum-comparison sorting. Would additional switches on side $A$ help?

When considered carefully, the reconstruction problem produces little programs that read like lovely poems. I would like to thank Frans, whom I have always enjoyed as an excellent educator, for helping me develop a taste for such poetry.

[2]For more information about the IOI see <URL:http://www.win.tue.nl/win/ioi/>.

# A Boyer-Moore (or Watson-Watson) type algorithm for regular tree pattern matching

Bruce W. Watson

This chapter is dedicated, with thanks, to Prof.Dr. Frans Kruseman Aretz — my tutor and first promotor during my Ph.D research.

In this chapter, I outline a new algorithm for regular tree pattern matching. The existence of this algorithm was first mentioned in the statements accompanying my dissertation, [2]. In order to avoid repeating the material in my dissertation, it is assumed that the reader is familiar with Chapters 4 and 5 of the dissertation.

## 1 Introduction

The most popular exact pattern matching algorithms[1] (for strings or trees) can be classified into one of two families: the Knuth-Morris-Pratt (KMP) or Boyer-Moore (BM) families.

Interest in Boyer-Moore type algorithms is driven by the fact that they are frequently much faster (in practice) than their KMP counterparts. For a discussion of this phenomenon, see [2]. Since a KMP type algorithm for regular tree pattern matching was presented in [1], the missing piece has been a BM type algorithm for tree pattern matching.

Instead of providing a set of formal definitions, we introduce most of the concepts using examples.

## 2 The problem

For the regular tree pattern matching problem, we consider node labeled trees (the labels are taken from a fixed alphabet). All nodes with a particular label have a fixed arity (number of children). We will write all of our trees in a linear (prefix) form, instead of drawing pictures; for example:

$$+(a, *(b, a))$$

[1] As opposed to *approximate* pattern matching algorithms.

Each of the nodes has an associated depth, with the depth of the root being 0.

A tree grammar is a finite set of productions, with nonterminals (written as uppercase letters, as opposed to regular node labels which are written in lowercase letters or as mathematical operators) on the left and tree templates on the right. Nonterminals are permitted to appear at the leaves of the tree templates. For example, each of the following lines is a production:

$$\begin{array}{lcl} A & \longrightarrow & +(B,B) \\ B & \longrightarrow & a \\ B & \longrightarrow & *(C,a) \\ C & \longrightarrow & b \\ C & \longrightarrow & *(b,B) \end{array}$$

The productions match at the input tree nodes in the intuitive way. In our sample input tree $(+(a,*(b,a)))$, we have the following matched patterns:

- The left $a$ is matched by $B \longrightarrow a$.
- The right $a$ is matched by $B \longrightarrow a$.
- The $b$ node is matched by $C \longrightarrow b$.
- The $*$ node is matched by $B \longrightarrow *(C,a)$ and by $C \longrightarrow *(b,B)$.
- The $+$ node is matched by $A \longrightarrow +(B,B)$.

In the next section, we will subdivide the pattern matching problem into a smaller problem which can be solved more readily.

## 3 Subproblems

One way of reducing the pattern matching problem to a simpler one, is to encode the trees as strings. We do this using so-called *path strings*. In this scheme, the tree is represented as a set of strings (there is one string for each leaf in the tree). Each string consists of alternating node labels and child numbers. For example, our input tree $+(a,*(b,a))$ is represented by $+1a$, $+2*1b$, and $+2*2a$.

In a similar manner, we encode each of the right sides of the productions as a set of path strings. The only difference is that we omit the nonterminals. For example, production $A \longrightarrow +(B,B)$ is represented by the two path strings $+1B$ and $+2B$, from which we drop the nonterminals to get $+1$ and $+2$. The example set of right sides is encoded as $+1$, $+2$, $a$, $*1$, $*2a$, $b$, $*1b$, and $*2$. These path strings can then be mapped back to their corresponding production right sides. These *pattern path strings* will be used in a reduced problem. Note that the pattern path strings will always begin with a node label.

Given this encoding, we will only concern ourselves with finding matches of the pattern path strings in the set of strings representing the input tree. The matching tree productions can then be easily reconstructed — this is not considered further here. In our example set of input path strings, we have the following pattern path string matches:

- $+1$ and $+2$ match at the root.
- $a$ matches at the left and the right $a$ nodes.
- $*1$, $*2$, $*1b$, and $*2a$ match at the $*$ node.
- $b$ matches at the $b$ node.

From this information, we can then piece together the tree matches. Note that, in effect, we are making use of multiple keyword pattern matching with the pattern path strings as the keywords. To solve this problem, we could use the Commentz-Walter algorithm (among others) [2, Section 4.4].

## 4 Solving the reduced problem

We begin by presenting a brute-force (naïve) algorithm, solving our simplest subproblem. In presenting the algorithm, we will assume (as in the string pattern matching algorithms presented in my dissertation) the following:

- We use a forward trie $\tau$ (constructed from the pattern path strings) for the actual pattern matching. The symbol $\perp$ is used to indicate when the trie takes an undefined value.
- We assume that the start state for the trie is named $q_0$.
- There is a special procedure $RM$ (for 'register matches') which is used to register matches at nodes in the tree. Precisely how it registers the matches is not relevant.

The mainline of the algorithm is:

$lev := (\mathbf{MAX}\ n : n \in nodes : n.level);$
**do** $lev \geq 0 \longrightarrow$
    **for** $n : n \in nodes \wedge n.level = lev \longrightarrow$
        $AM(q_0, n)$
    **rof**;
    $lev := lev - 1$
**od**

This algorithm simply traverses the tree from the bottom up, using procedure $AM$ (for 'attempt match') to check for matches and $RM$ to register the matches. The procedure $AM$ is given as:

```
proc AM(q, n) is
        RM(q, n);
        if τ(q, n.label) ≠ ⊥ then
                q := τ(q, n.label);
                RM(q, n);
                for i ∈ [1, n.arity] ⟶
                        if τ(q, i) ≠ ⊥ then
                                AM(τ(q, i), n.child(i))
                        fi
                rof
        fi
corp
```

This procedure uses the trie and traverses the input tree (starting at node $n$) top-down to find matches. Note that it is recursive.

As with the other BM type algorithms, we wish to make shifts of more than one level (in the tree) in the mainline program. The shift will be computed in a manner similar to that in the Commentz-Walter family of algorithms — since we are using multiple keyword pattern matching.

Since procedure $AM$ tries a number of possible paths rooted at a node $n$, there will be a number of potential contributing shifts. In order to make a *safe* shift, we will have to use the smallest of these contributing shifts.

We will use a novel method of implementing the actual shift: if a shift of distance $k$ is required after a match attempt at node $n$, we will store $n.level - k$ in a location $permit[n]$ ($permit$ is an array which is indexed by $n$). If a match attempt is initiated at some node $n'$ (above n), then the match attempt will not continue (down in the tree) past node $n$ unless $n' \leq permit[n]$. This can be done safely since all matches begun lower than $permit[n]$ cannot possibly lead to a match below $n$.

To implement this, we use the following mainline[2]:

```
for n : n ∈ nodes ∧ n.isleaf ⟶
        permit[n] := n.level
rof;
lev := (MAX n : n ∈ nodes : n.level);
```

[2]To abort a match attempt when it is futile, we also pass a third argument to $AM$ — the level at which the match attempt was started. The procedure now returns the integer shift (in terms of levels).

```
do lev ≥ 0 ⟶
        for n : n ∈ nodes ∧ n.level = lev ⟶
                permit[n] := n.level − AM(q0, n, lev)
        rof;
        lev := lev − 1
od
```

Correspondingly, we make use of the shift function *shift* which gives the shift distance in levels in the tree[3].

```
proc AM(q, n, beginlev) returns sh is
        RM(q, n);
        if τ(q, n.label) = ⊥ ∨ beginlev > permit[n] then
                sh := shift(q)
        else
                q := τ(q, n.label);
                RM(q, n);
                if n.arity = 0 then
                        sh := shift(q)
                else
                        for i ∈ [1, n.arity] ⟶
                                if τ(q, i) = ⊥ then
                                        sh := sh min shift(q)
                                else
                                        sh := sh min AM(τ(q, i), n.child(i), beginlev)
                                fi
                        rof
                fi
        fi
corp
```

## 5 Conclusions

I have outlined a Watson-Watson type algorithm for regular tree pattern matching, thereby backing-up the statement accompanying my dissertation [2]. Unfortunately, this algorithm has not yet been implemented and so nothing is known about its running time performance in practice. It does, however, appear that the algorithm will not be as efficient as the KMP type algorithm (solving the same problem) given by Aho and Ganapathi in [1].

[3]The distance in levels is the ceiling of half of the distance given by the Commentz-Walter shift functions, since the shift distance given for the pattern path strings will be in terms of the path strings and a path string may be up to twice as long as the level of the leaf at which it ends.

Like the other classes of BM type algorithms (for single keyword, multiple keyword, or regular expression string pattern matching), there are likely to be other (perhaps more efficient) variants of this algorithm. For example, it may be possible to devise such an algorithm which operates in a top-down manner, instead of in a bottom-up manner; alternatively, it may be possible to reduce the primary problem in a different way to what we have done here. These alternatives are left as an exercise for the reader.

**Acknowledgements:** I would like to thank Richard Watson (co-developer of the Watson-Watson regular expression pattern matching algorithm for strings) and Nanette Saes for their assistance in preparing this note.

## References

[1] AHO, A.V. and M. GANAPATHI. Efficient tree pattern matching: an aid to code generation, in: *Proceedings of the 12th ACM Symposium on Principles of Programming Languages,* New Orleans, p. 334–340, 1985.

[2] WATSON, B.W. *Taxonomies and Toolkits of Regular Language Algorithms,* Ph.D dissertation, Faculty of Computing Science, Eindhoven University of Technology, The Netherlands, 1995, ISBN 90-386-0396-7.

# Art or science ?
# The example of decision making

Jaap Wessels[1]

## 0 Introduction

For quite a number of activities, there is a recurring discussion about the question: is execution of this activity essentially artistic or essentially scientific? *Artistic* then stands for creative, intuitive, based on experience rather than knowledge. *Scientific* in this context stands for systematic, rational, consistent, objective. Sometimes the term *Art* is replaced by *Craftmanship*, but that does not change the discussion essentially. In French the terms are even closely related. I prefer the term *Art*, because it pushes the discussion more to the extreme.
We have these discussions frequently about activities like *computer programming, architectural design, product design, applying mathematics to industrial problems, management.* For all these cases and many others, it is my opinion that the answer should not be Art *or* Science, but Art *as well as* Science. In the present paper I will try to explain this answer for the case of *Decision Making* and also draw a few conclusions from the answer for the Science of Decision Making.

In Section 1, it is explained what is meant by Decision Making in this paper. Section 2 will explain why Decision Making needs the artistic as well as the scientific approach. As support for the argumentation, Section 3 explains the four main aspects of decision making and Section 4 presents methods of decision support which are based on one of these aspects. Finally, Section 5 sketches some of the attempts to develop methods which integrate some aspects.

The paper is dedicated to my colleague Frans Kruseman Aretz at the occasion of his retirement as a Professor of Computing Science at Eindhoven University of Technology. I don't think that Frans was much involved in discussions about *Decision Making*, but I remember quite well the similar discussions about *Computing Programming* in which he was also involved. Usually you learn a lot from colleagues without being able to identify

[1]The paper was written during a stay at the International Institute for Applied Systems Analysis at Laxenburg, Austria. The discussions on this topic with colleagues at IIASA are kindly acknowledged. The paper reflects some results of a mental exercise, which was undertaken in order to develop a strategic view regarding research on methodology of decision analysis.

what you learned. In this case, I have two specific points which I owe clearly to Frans. In my second year as a student a the University of Amsterdam, in the academic year 1957/58, we had to do standard experiments in a students laboratory for physics. As a graduate student, Frans was working as an assistant in this laboratory and we had to discuss our reports with him. At that time, I was astonished how quickly he was able to make our lack of understanding visible and how straightforwardly he discovered our ways of working towards the desired results. When I was in his position a few years later, I could use his tricks, but never with the same skill. Later, in Eindhoven, I have used many times the small paper on Fast Fourier Transforms that Frans published together with Zonneveld in the Liber Amicorum at the occasion of Bouwkamp's retirement from Philips Research. It hits the central concept right on the head, without drowning in all those technical intricacies.
These two points may not be on the main track of his career, but they seem rather typical to me.

## 1 Something about decision making

Of course, we decide the whole day long about what to eat, to go by car or by bike, to take the first or the second crossing. About this type of selecting actions, one can discuss a lot and, indeed, it is a topic for scientific research. However, it is not the topic that will be considered in this paper. The present paper is about a more formal type of decisions made in more formal settings. Therefore, one might call it *institutional decision making.* This is a considerable restriction, but nevertheless it is still a broad area. It ranges from operational decisions within firms to decisions about strategic alliances by governments. So, on one hand we consider decisions about the sequence of handling orders in a production environment and on the other hand we encounter decisions like joining the European Union or NATO. In between, we find decisions like chosing a distribution structure by a food chain, selecting new product-market combinations, selecting between several infrastructural investment plans by a government, negotiating a treaty for acid rain abatement between several governments in a region.
All such decisions are taken in a more or less formal setting which should guarantee that different aspects are taken into account. For the less-far-reaching decisions, this implies that a planner may take the decisions, as long as they remain within a given range, but that he should be able to defend his choice. For the farther reaching decisions, this leads to rather complex decision processes in which different groups may take part.

Since the Second World War, it has become more and more popular to design methods for analysing decision situations and supporting decision processes. This development is representing the scientific approach to decision making. Although such methods have been extremely successful in many cases, there are still many other situations where progress of the scientific approach is not very apparent. The most frequently recurring terms in the success stories are *Information Processing* and *Mathematical Models.*

## 2 Why art as well as science?

For executing the activity of institutional decision making, one may roughly distinguish the *artistic approach* and the *scientific approach*, where *artistic* stands for creative, intuitive, based on experience rather than knowledge, integrating different aspects, subjective. *Scientific*, on the other hand, stands for systematic, rational, consistent, objective, based on knowledge rather than on experience.
Both approaches have their adherers who can justify their choice by showing their success stories and by hinting at the failures of the other approach. On one side, it is argued that the scientific approach will never be able to integrate all aspects of real decision problems in one holistic view. On the other side, the argument is used that the scientific approach guarantees an objective view of advantages and disadvantage of possible alternative decisions. Both arguments seem to be right.
From an academic point of view it would be nicest if decision making could rely completely on the scientific approach. So, let us consider why this is not realistic. I see two main arguments for this opinion:

1. It is an illusion to believe that all aspects of decision making are really open for a scientific approach.
   Here one may think of the very complicated aspect of interpersonal relationships which may have an important influence on decision making. Actually already in very down-to-earth production planning processes we find features which seem to be difficult to grasp: for production planning we need to understand the demand processes. This is often extremely difficult, not to say impossible. In quite a number of cases our understanding of such processes, even in a statistical way, will always be limited.

2. Most scientific approaches are essentially based on only one aspect of the decision problem with only poor incorporation of other aspects.
   Indeed, in most cases, the natural possibilities of incorporating other aspects are weak.

Of course, one may argue that both arguments say something about the current state of science and one may expect progress in both directions. This is definitely true, however, progress does not imply that the arguments don't hold any more, but only that there is a gradual shift towards the scientific approach. In the next sections, the second argument will be worked out in a little bit more detail, in order to explain that it indeed refers to an essential difficulty. For the moment, way may conclude that we have to accept that the scientific approach has its limitations. As a consequence, it would be wise to develop our decision support methodology in such a way that it allows for a constructive interplay between scientific and artistic activities within decision processes. This consequence is more substantial then one would think at first sight. Particularly, for mathematicians and computing scientists involved in supporting decision making. Take for example the European negotiations about the abatement of acid rain. In these negotiations more than

25 countries are involved. There is really something to negotiate here, since most countries export a substantial part of their emissions responsible for acid rain. Reciprocally most countries suffer from acid rain which is caused to a substantial degree by emissions in other countries. Consequently, counter measures against emissions in one country are beneficial for other countries, but benefits are not proportional to the damage these countries cause with their own emissions in our measure-taking country. This feature makes the negotiations extremely complex. Accepting that the scientific approach cannot solve this decision problem completely, implies acceptance of the essential role played by the craftmanship or the artistry of the diplomats involved. And the latter acceptance, in its turn, implies that the scientific approach is no longer aiming at providing a full solution (which might be amended a little bit by the diplomats), but aims at providing the diplomats with information and insights in such a way that it lifts their craftmanship to a higher level.

## 3 The main aspects of decision problems

Below, four aspects will be described, which are essential to all institutional decision problems. In order to clarify these aspects they will be illustrated by a decision problem regarding the purification of a highly polluted river basin in Slowakia. In this case it has to be decided what decisions to take with respect to purification investments and emission constraints along the river Nitra and its contributaries. There are several towns involved, which all dump their household and industrial liquids without purification in the river basin. The area possesses quite a bit of heavy industry, largely based on outdated technology. In other parts of the area, agriculture is an important user of river water, but also a polluter. The Nitra river runs into the Danube.

The first aspect to be mentioned is:

a. *Information*: Huge amounts of data are relevant for the decision making in our illustrative example. Here one may think of rainfall data, demographic data, data regarding water use and pollution by industrial, agricultural and household activities, biological information, geographic information about the water bed of the river basin.

Another basic aspect consists of:

b. *The underlying physical or economic processes*: In our example this regards water flows, dispersion processes of pollutants, chemical and biological reactions in the river basin.

The next aspect regards the decision making already more directly. However, it relies heavily on the aspects mentioned before:

c. *The relation between decisions and consequences*: For our example we may mention the following types of consequences which depend highly on the decisions: the investment costs, the water quality at different places in the river basin with respect to different pollutants, economic and social consequences on national, regional and local level, the relationship with other Danube countries (particularly downstream).

And, finally we have to deal with:

d. *The decision making process*: In our case the decision making process is very complex because many gremia are involved such as local, regional and national authorities, industries, agricultural pressure groups, population interest groups, neighbouring countries, international sponsoring agencies. Another feature which complicates the decision making process is the governmental instability.

## 4 Scientific approaches essentially based on one aspect

As said before, practically all attempts to develop a scientific approach towards decision making essentially base themselves on one aspect and only try halfheartedly to incorporate one or more of the other aspects. Of course, if in a particular problem one aspect is dominant, this procedure leads to a satisfactory solution. However, most frequently, more aspects are essential, like in the Nitra case.
In the present section, we will mention for each of the aspects one or more approaches which are predominantly based on that aspect.

a. *Information*:
Management Information Systems,
Geographic Information Systems.
Such approaches are strong in providing insight in the current situation – eventually also in the past – but relatively weak in showing consequences of possible decisions, let alone in suggesting good decisions.

b. *The underlying physical or economic processes*:
Scenario analysis based on a model of the underlying processes.
Such scenario analyses may exploit different types of analytic tools such as discrete event simulation, queueing theory, differential equations.
For technical reasons the models are often less detailed than the information would permit. Moreover, it may be cumbersome to find a good scenario by trial and error.

c. *The relation between decisions and consequences*:
Mathematical Programming,
Multi-Criteria Decision Analysis,
Game-Theoretic Modelling and Analysis.

In such approaches, decisions are modelled as variables, restrictive relations between decision variables represent constraints and goals are represented by functions of the decision variables. In the Nitra case one would have decision variable $x_i$ representing investment decision $i$ with a value 1 if the investment is made and with 0 otherwise. Such approaches are often based on highly aggregated views.

Neural Nets.

This approach exploits the fact that in many decision problems it is afterwards very clear which decision should have been taken. This property can then be exploited to construct training pairs.

d. *The decision making process*:

Decision Trees,
Group Decision Support,
Negotiation Facilitating,
Multi-Attribute Utility Theory,
Expert Systems,
Neural Nets,
AHP (in many variants).

Some of these approaches even concentrate on a subaspect, other aim at mimicking the decision behaviour of an expert.

## 5 Attempts to base approaches on at least two aspects in an integrated way

The previous section lists an overwhelming majority of generally accepted approaches, each concentrating on one aspect of the decision making process. Nevertheless, there are attempts to integrate some aspects. From a practical point of view, one may state that such attempts are either still in an exploratory state, or they are only aiming at restricted groups of applications.
From a research point of view, such attempts are at the same time risky and exciting. Let us mention some of these attempts.

(i) The most successful attempts integrate the underlying processes and the consequences of decisions. Such attempts can be found in optimal process control and have proved to be effective in restricted sets of applications, mostly of an operational nature. For the mathematical sciences these attempts have proved to be very stimulating. The same aspects are integrated by attempts to exploit simulation experiments in such a way that the search for good scenarios is effectively supported. In this direction the mathematics involved are quite exciting, the practical results are incidental but promising.
Scenario analysis is also enhanced with policy search exploiting rule-based methods.

(ii) Integration of the information aspect with the underlying processes is attempted in approaches based on specification methods. The most widely used specification method is Petri Nets. For particular types of application, results are promising.

(iii) A final example regards the integration of the decision making process and the relation between decisions and consequences. Practically, this comes down to an integration of hard and soft information, where the hard information is provided by model studies which usually regard only specific features of the problem, whereas the soft information regards the notions about consequences which cannot easily be modelled. For example in the Nitra case, it is quite well possible to model the relation between decisions and fysical consequences, however, the modelling of social consequences is much more questionable.

We may conclude that development of methods integrating different aspects is scientifically exciting and will bring some nice results in the next few years, but we have to accept that fully integrated and generally applicable approaches will not be available in the foreseeable future and probably never.

# A wheel! (but relationally)

Jaap van der Woude

Few reinventions match the popularity of the wheel, the transformation of primitive recursion to tailrecursion comes close. It may be some kind of relief that the wheel to be presented shortly is just a depthless view on the descent-ascent sphere (globus augustinianum) symbolising the neverending role of relations, mathematically as well as between the scientific interests of you and us. Indeed, the intermediate, instigating and collaborating force behind this wheel is Rik van Geldrop, the interface between the two groups that are bound to miss your presence.

We will sketch a special case of the recursive-descent-to-recursive-ascent transformation in a relational tayloring; the general treatment lacked the space efficiency needed for this booklet.

Let $(D, \leq)$ be a wellfounded poset with minimal element set $M$. Given the functions $f_0 : M \rightarrow C$, $\otimes : D \times C \rightarrow C$ and contracting endofunction $g : D \rightarrow D$ (i.e. $g.d < d$), the equation in $f$

$$
\begin{array}{lll}
f.m & = & \textsf{if } m \in M \ \rightarrow \ f_0.m \\
 & & \Box \ m \notin M \ \rightarrow \ m \otimes f.(g.m) \\
 & & \textsf{fi}
\end{array} \tag{1}
$$

defines a (unique) function from $D$ to $C$.
The stack of $\otimes$-invocations may necessitate another view on this function. As we shall see, a suitable relational translation enables a smooth transition to a tailrecursive form via the wellknown equality

$$R^* \ = \ \mu_X(I \sqcup X \circ R) \ = \ \mu_X(I \sqcup R \circ X) \ = \ {}^*R \ . \tag{2}$$

The equation (1) is translated to a relational equation as follows:
The graph of a function $f$ is represented by the so-called rightcondition $F$

$$zF(p,q) \ \equiv \ f.p = q \quad \text{for every } z \ . \tag{3}$$

In particular, $f_0$ is represented by $F_0$ and (1) by

$$F \ = \ F_0 \sqcup F \circ R \ . \tag{4}$$

where the relation $R$ captures $g$ and $\otimes$ by

$$(x,y)R(m,q) \ \equiv \ x = g.m \ \wedge \ m \otimes y = q \ \wedge \ m \notin M \ . \tag{5}$$

Indeed, for the second disjunct of (4):

$$
\begin{array}{ll}
 & zF(m,q) \\
\equiv & \quad \{ \quad F;\ m \notin M;\ f \text{ solves (1)} \quad \} \\
 & q \ = \ m \otimes f.(g.m) \\
\equiv & \quad \{ \quad \text{intro } \exists\ x,y \quad \} \\
 & f.x{=}y \ \wedge \ x{=}g.m \ \wedge \ q{=}m{\otimes}y \\
\equiv & \quad \{ \quad F;\ R \quad \} \\
 & zF(x,y) \wedge (x,y)R(m,q) \\
\equiv & \quad \{ \quad \text{"}\circ\text{"} \ ; \text{ elim } \exists\ x,y \quad \} \\
 & z \ F{\circ}R \ (m,q) \ .
\end{array}
$$

The rightcondition representation can be performed in general, and there is (again in general) a ono to one correspondence between the solutions of (1) and (4).
Since $(D, \leq)$ is wellfounded and $g$ is contracting, $R$ admits only finite left-decreasing chains (is left-wellfounded), so equation (4) has a unique solution:

(6) $\qquad F_0 \circ R^* \quad$ is the unique solution of (4) .

In calculating $f.n$ for some individual $n{\in}D$ we may restrict our attention to $F \circ (\{n\}{\times}C)$, where the monotype $\{n\}{\times}C$ denotes the partial skip, i.e., a part of the diagonal in $(D{\times}C){\times}(D{\times}C)$. For this calculation the Copernican turn

$$
\begin{array}{ll}
 & F \circ (\{n\}{\times}C) \\
= & \quad \{ \quad (6) \quad \} \\
 & F_0 \circ R^* \circ (\{n\}{\times}C) \\
= & \quad \{ \quad R^* \ = \ {}^*R \quad \} \\
 & F_0 \circ {}^*R \circ (\{n\}{\times}C) \\
= & \quad \{ \quad Q \text{ below} \quad \} \\
 & F_0 \circ Q \ ,
\end{array}
$$

$$
\text{where } \ Q \ = \ \mu_X(\{n\}{\times}C \sqcup R{\circ}X) \ ,
$$

results, eventually, in a tailrecursive format. Indeed if we fall for the temptation to write

(7) $\qquad y{=}q.n.a.b \ \equiv \ (a,b)Q(n,y) \ ,$

we arrive at

$$
\begin{array}{ll}
 & y \;=\; f.n \\
\equiv & \quad \{ \quad F \text{ ; intro } \forall z \quad \} \\
 & zF(n,y) \\
\equiv & \quad \{ \quad \text{above} \quad \} \\
 & \exists(u,v\colon zF_0(u,v)\colon (u,v)Q(n,y)) \\
\equiv & \quad \{ \quad F_0\text{; elim } \forall z \quad \} \\
 & \exists(u::\ (u, f_0.u)Q(n,y)) \\
\equiv & \quad \{ \quad q \quad \} \\
 & \exists(u::\ y = q.n.u.(f_0.u))\ ,
\end{array}
$$

while

$$
\begin{array}{ll}
 & y \;=\; q.n.a.b \\
\equiv & \quad \{ \quad Q\text{; assume } a{=}n \quad \} \\
 & y \;=\; b\ ,
\end{array}
$$

and

$$
\begin{array}{ll}
 & y \;=\; q.n.a.b \\
\equiv & \quad \{ \quad Q\text{; assume } a{\neq}n \quad \} \\
 & \exists(u,v\colon (a,b)R(u,v)\colon (u,v)Q(n,y)) \\
\equiv & \quad \{ \quad R\ ;\ q \quad \} \\
 & \exists(u\colon\ a = g.u\colon\ y = q.n.u.(u{\otimes}b))\ .
\end{array}
$$

So $f.n$ can be calculated by applying the tailrecursive algorithm $q.n$ to a result of $f_0$. However, the question arises whether (7) is allowed: is $q.n$ a function and, if so, how about the choices of "$u$"? In both cases they seem nondeterminate, if at all possible.

A more precise investigation of $Q$, in particular of the leftdomain of $Q$, shows that the above reasoning is not fraudulent: for partial skip $A$

$$
\begin{array}{ll}
 & Q \;=\; A \circ Q \\
\Leftarrow & \quad \{ \quad Q \sqsupseteq A \circ Q \text{ ; least fixpoint induction} \quad \} \\
 & (\{n\}{\times}C) \sqcup R{\circ}A{\circ}Q \;\sqsubseteq\; A \circ Q \\
\Leftarrow & \quad \{ \quad \text{unfold } Q \text{ on the RHS ; } \circ\, Q \text{ is monotonic} \quad \}
\end{array}
$$

$$
\begin{array}{ll}
 & \{n\}\times C \sqsubseteq A \ \wedge \ R \circ A \sqsubseteq A \circ R \\
\Leftarrow & \quad \{ \quad R \text{ ; assume the shape } A{=}K\times C \quad \} \\
 & n{\in}K \wedge \forall(k{:}\ k{\in}K{:}\ g.k \in K) \\
\Leftarrow & \quad \{ \quad \} \\
 & K \ = \ \{g^i.n \mid i{\in}\mathbb{N}\} \ .
\end{array}
$$

This calculation yields a more accurate knowledge of the leftdomain of $Q$, namely $K\times C$. By wellfoundedness of $(D,\leq)$ and contractivity of $g$ the set $K$ "is" a finite decreasing chain and the equality $Q = (K\times C) \circ Q$ allows *determinate* choices for the "$u$"'s above. Indeed, the least element of $K$ is in $M$ and every $K$-member has a unique predecessor in $K$. Functionality of $q.n$ can be shown via functionality of $f$ and $F_0 \circ Q \ = \ F \circ (\{n\}\times C)$.

Denoting the least element of $K$ by $k$ and the predecessor function on $K$ by $\sigma$, i.e. for $a{\in}K$,

(8) $$\sigma.a = u \ \equiv \ u{\in}K \ \wedge \ a = g.u \ .$$

we arrive at the following tailrecursive schema for the calculation of $f.n$:

(9)
$$
\begin{array}{rcl}
f.n & = & q.n.k.(f_0.k) \\
q.n.a.b & = & \text{if } a = n \ \rightarrow \ b \\
 & & \Box \ a \neq n \ \rightarrow \ q.n.(\sigma.a).(\sigma.a \otimes b) \\
 & & \text{fi} \ .
\end{array}
$$

Which is only satisfying if $k$ and $\sigma.a$ are readily computable from $a$ and $g$ (and $n$). If not, there is a way out by precomputing $K$ as a list and tupling the $K$-shift with $q.n$, thus resurrecting the d... stack.

**Example 10** . Let $fusc : \mathbb{N}\rightarrow\mathbb{N}$ be defined by

$$
\begin{array}{rcll}
fusc.n & = & n & \text{if } n \leq 1 \\
fusc.(2*n) & = & fusc.n & \text{if } n \geq 1 \\
fusc.(2*n+1) & = & fusc.(n+1) + fusc.n & \text{if } n \geq 1 \ .
\end{array}
$$

By the standard tupling technique *fusc* may be reformulated as the projection after the primitive recursive function $f : \mathbb{N} \rightarrow \mathbb{N}\times\mathbb{N}$ given by

$$
\begin{array}{rcl}
f.n & = & \text{if } n = 0 \ \rightarrow \ (0,1) \\
 & & \Box \ n \neq 0 \ \rightarrow \ n \otimes f.(n\text{div}2) \\
 & & \text{fi} \ ,
\end{array}
$$

$$\text{where } \ n{\otimes}(x,y) \ = \ (x + y * (n\text{mod}2) \ , \ x * (n{+}1 \bmod 2) + y)$$

The transformation to a tailrecursive calculation of $f.n$, as in (9), results in

$$
\begin{array}{rcl}
f.n & = & q.n.0.(0,1) \\
q.n.a.(x,y) & = & \mathsf{if}\ a = n \ \rightarrow\ (x,y) \\
 & & \Box\ a \neq n \ \rightarrow\ q.n.(\sigma.a).(\sigma.a \otimes (x,y)) \\
 & & \mathsf{fi}\ ,
\end{array}
\tag{11}
$$

with the expected nuisance of the not readily computable $\sigma$, caused by the noninjectivity of div2. Chickening out by precomputing $K$ in increasing order of $\sigma$-powers applied to 0, and splitting $K$ in a head and tail part while removing the superfluous $n$, leads to

$$
\begin{array}{rcl}
f.n & = & \psi.(\mathsf{hd}.\kappa).(\mathsf{tl}.\kappa).(0,1) \\
\psi.a.\alpha.(x,y) & = & \mathsf{if}\ a = n \ \rightarrow\ (x,y) \\
 & & \Box\ a \neq n \ \rightarrow\ \psi.(\mathsf{hd}.\alpha).(\mathsf{tl}.\alpha).(\mathsf{hd}.\alpha \otimes (x,y)) \\
 & & \mathsf{fi} \\
 & & \\
\kappa & = & \lambda.n.[] \\
\lambda.p.\alpha & = & \mathsf{if}\ p = 0 \ \rightarrow\ p : \alpha \\
 & & \Box\ p \neq 0 \ \rightarrow\ \lambda.(p\ \mathsf{div}2).(p : \alpha) \\
 & & \mathsf{fi}\ .
\end{array}
\tag{12}
$$

In order to obtain the above tailrecursive algorithm for $\kappa$ we did *not* use the presented primitive- to tailrecursive transformation. The (re)occurrence of the function div2 would give the same problem for $\kappa$ as for $q$, yielding a nonterminating series of transformations. Generalisation by abstraction is a simple and standard method for programming $\kappa$.

By now you have every right to be disappointed: $\kappa$ is a heavy burden on the space complexity, it is a stack of $\mathsf{log}.n$ integers! Very little invention is needed to see that the list $\kappa$ corresponds to the increasing list of prefixes of the binary expansion of $n$, so a stack of $\mathsf{log}.n$ booleans should suffice:

$$
\begin{array}{rcl}
f.n & = & \phi.0.(\mathsf{bin}.n).(0,1) \\
\phi.a.\alpha.(x,y) & = & \mathsf{if}\ a = n \ \rightarrow\ (x,y) \\
 & & \Box\ a \neq n \ \rightarrow\ \phi.(2 * a + \mathsf{hd}.\alpha).(\mathsf{tl}.\alpha).(\mathsf{hd}.\alpha \odot (x,y) \\
 & & \mathsf{fi}\ .
\end{array}
\tag{13}
$$

$$\text{where}\ \ \varepsilon\odot(x,y) \ = \ (x + y * \varepsilon\ ,\ x * (1-\varepsilon) + y)\ .$$

□

The final representation transformation may seem ad hoc, but it is an instance of a general transformation applicable to (9). The algebraic setting of (9) consists of

$$(\ K,\ \leq,\ {=}n : K \rightarrow \mathbb{B},\ \sigma : K \rightarrow K,\ \otimes : K \times C \rightarrow C\ )$$

A homomorphism $\varphi : K \rightarrow E$ to a similar algebra

$$( E, \preceq,\ \approx n : E \rightarrow \mathbb{B}, \tau : E \rightarrow E, \oplus : E \times C \rightarrow C ) ,$$

such that $\varphi$ is monotonic, $(=n) = (\approx n) \circ \varphi$, $\varphi \circ \sigma = \tau \circ \varphi$ and $\otimes = \oplus \circ (\varphi \times Id)$ can be shown to satisfy

(14) $$q.n.a.b = p.n.(\varphi.a).b ,$$

where

$$\begin{aligned} p.n.x.b = \ & \mathsf{if}\ x \approx n \rightarrow b \\ & \Box\ x \not\approx n \rightarrow p.n.(\tau.x).(\tau.x \oplus b) \\ & \mathsf{fi} . \end{aligned}$$

In the example we may choose

$$\begin{aligned} E &= \mathbb{N} \times \mathbb{B}^* \\ \varphi.a &= (a, \mathsf{drop}.(\mathsf{bin}.a).(\mathsf{bin}.n)) \\ \tau.(a, \alpha) &= (2 * a + \mathsf{hd}.\alpha, \mathsf{tl}.\alpha) \\ (a, \alpha) \oplus (x, y)) &= \mathsf{hd}.\alpha \odot (x, y) , \end{aligned}$$

noting that $\mathsf{hd}.\alpha = \sigma.a \bmod 2$ .

Some rather straightforward verification is left for a rainy sunday afternoon, as is the derivation of an elegant tailrecursive algorithm for *fusc* via generalisation by abstraction. In fact the presented example is inferior to what the innocent functional programmer will concoct or, for that matter, any sophomore with some ability in "Programmeren 4".

Why use a wheel, where skates are more appropriate?
Do you know an icerink in Zandoerle?
There might be a blacksmith, however, ready to experiment with relational things gradually shaping up to something round.

# Sublinear pattern matching

Gerard Zwaan

## 1 Introduction

The material presented in this paper was influenced by Frans Kruseman Aretz through his constructive criticisms, his questions, his being dissatisfied with earlier versions of some formulae, and his spotting of even the smallest inconsistencies and errors. The presented material is the result of joint work with B.W. Watson. A more extensive coverage can be found in [WZ95].

This paper deals with pattern matching in two respects. On the one hand it contains a classification (taxonomy) and derivation of a number of well known algorithms solving the keyword pattern matching problem, on the other hand it is the result of recognizing patterns in these algorithms. The latter process is not described here, but is reflected— although in reverse order— by the systematic derivation of the algorithms from a common starting point and their classification. The approach to algorithm classification used was inspired by the method described by Jonkers [Jon83].

The keyword (or string) pattern matching problem can informally be described as the problem of finding all occurrences of keywords (strings) from a given set as substrings in a given (input) string (e.g. a text, a DNA sequence). Taking the number of symbol comparisons as a measure of matching time we focus on a family of multiple keyword pattern matching algorithms having a matching time that may be sublinear in the length of the input string. This implies that not all symbols of the input string have to be inspected. An example of a sublinear matching time is displayed in figure 1 on page 336. Trying to find occurrences of pattern `baab` one aligns it with substrings of the input string and starts comparing symbols from right to left. Subsequently—whether a match is found or not—the pattern shifted to the right over a distance depending on the information gained in the last matching attempt. In this way only 6 symbol comparisons are needed whereas the input string consists of 11 symbols. The sublinearity is obtained by comparing symbols from right to left and shifting the pattern from left to right. All algorithms in this paper exhibit this behaviour. They only differ in the distance the pattern is shifted after a matching attempt. Given a pattern of length $m$ and an input string of length $n$ the matching time is at least $n/m$ (matching $a^m$ in $b^n$), at most $nm$ (matching $a^m$

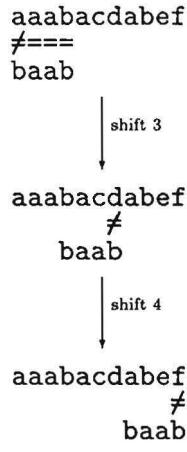



Figure 1: Example of a sublinear matching time

in $a^n$), and on the average between between $n/m$ and $n$. Ideally, one would like to shift the pattern to its next occurrence in the input string, but then calculation of the shift distance is equivalent to the pattern matching problem itself. Hence, we aim at so called *safe shift distances* that are approximations from below of the ideal shift distance and that are more easily and efficiently calculated. (Notice that shift distance 1 is most easily calculated but is not quite satisfactory with respect to matching time.) These approximations are derived in a systematic way and serve to classify the corresponding algorithms. Among the algorithms derived are the multiple keyword generalization of the single keyword Boyer-Moore algorithm [BM77], the Commentz-Walter algorithm [CW79a, CW79b], a common ancestor of the two preceding algorithms, and an algorithm by Fan and Su [FS93, FS94] (only after deriving this algorithm we found its description by Fan and Su). Both the Boyer-Moore and Commentz-Walter algorithms provided the inspiration for our derivations and our classifying principle.

In section 2 we give a formal definition of the pattern matching problem. From a trivial solution to this problem we derive an algorithm that is the starting point for the derivation of the algorithms in section 3. In section 3 we start start by allowing shift distances larger than one by introducing the notion of the maximal safe shift distance. Allowing larger shift distances accounts for the possibly sublinear matching time of all of the algorithms to be derived in this section. Each algorithm is characterized by a safe shift distance being an approximation from below of the maximal safe shift distance. The safe shift distances are derived in a systematic way in descending order of magnitude. Section 4 contains the conclusions. Appendix A contains definitions and properties used throughout this paper. Precomputation of all of the functions introduced in section 3 is not discussed in this paper. The interested reader is referred to [WZ95].

## 2 The problem and some naive solutions

The keyword pattern matching problem is to find all occurrences of keywords from a set as substrings in an input string. Formally, given an alphabet $V$ (a non-empty finite set of symbols), an input string $S \in V^*$, and a finite non-empty pattern set $P \subseteq V^*$, establish

$$R: \; O = \big( \cup \; l, v, r : lvr = S : \{l\} \times (\{v\} \cap P) \times \{r\} \big) .$$

Notice that in the above quantification $v$ is a substring of $S$ being matched against any keyword from set $P$, and that matches are represented by triples consisting of the part of $S$ before the match, the matching substring, and the part of $S$ after the match. The redundancy in this representation can be eliminated but we will not do so.

A trivial (but unrealistic) solution to the problem is[1]

**Algorithm 2.1**

$$O := \big( \cup \; l, v, r : lvr = S : \{l\} \times (\{v\} \cap P) \times \{r\} \big) \quad \{ \; R \; \}$$

There are two basic directions in which to proceed while developing naive algorithms to solve this problem. Informally, a substring of $S$ can be considered a "suffix of a prefix of $S$" or a "prefix of a suffix of $S$". Only the first possibility is considered here, since the second possibility only leads to algorithms that are the mirror images of algorithms obtained by following the first possibility (basically, it amounts to reversing all strings in the problem). Moreover, this is the way that the Boyer-Moore, Commentz-Walter, and Fan and Su algorithms treat substrings of input string $S$. The notion "suffixes of prefixes of $S$" is expressed by the following equality

$$\begin{aligned} &\big( \cup \; l, v, r : lvr = S : \{l\} \times (\{v\} \cap P) \times \{r\} \big) = \\ &\quad \big( \cup \; u, r : ur = S : \big( \cup \; l, v : lv = u : \{l\} \times (\{v\} \cap P) \times \{r\} \big) \big) \end{aligned}$$

where $u$ constitutes a prefix of $S$ and $v$ a suffix of prefix $u$. The Boyer-Moore and Commentz-Walter algorithms shift the patterns from left to right along $S$ and compare the symbols of the patterns with the aligned symbols of $S$ from right to left. In order for an algorithm to have such a behaviour it should examine prefixes of $S$ in order of increasing length and for every prefix it should examine its suffixes in order of increasing length. Hence, we obtain the following algorithm with two nested repetitions[2]:

[1] Throughout this paper we will adopt the convention that, unless stated otherwise, program variables and bound variables with names from the beginning of the Latin alphabet (i.e. $a, b, c$) will range over $V$, while variables with names from the end of the Latin alphabet (i.e. $l, q, r, u, v, w$) will range over $V^*$.

[2] the operators $\rceil$, $\lfloor$, $\lceil$, and $\rfloor$ are defined in definition A.1; relation $\leq_p$ is defined in definition A.3

**Algorithm 2.2**

$u, r := \varepsilon, S;\ O := \{\varepsilon\} \times (\{\varepsilon\} \cap P) \times \{S\};$
{ invariant: $O = \big(\cup\, x, y, z : xyz = S \wedge xy \leq_p u : \{x\} \times (\{y\} \cap P) \times \{z\}\big)$
$\wedge\ ur = S$ }
**do** $r \neq \varepsilon \rightarrow$
$u, r := u(r\restriction 1), r\downarrow 1;\ l, v := u, \varepsilon;\ O := O \cup \{u\} \times (\{\varepsilon\} \cap P) \times \{r\};$
{ invariant: $lv = u$ }
**do** $l \neq \varepsilon \rightarrow$
$l, v := l\downarrow 1, (l\restriction 1)v;$
$O := O \cup \{l\} \times (\{v\} \cap P) \times \{r\}$
**od**
**od**{ $R$ }

Notice that statement $u, r := u(r\restriction 1), r\downarrow 1$ reflects the shifting of all patterns to the right over a distance of 1. In the next section we will see how one can improve the shift distance. Beforehand, however, we bring the behaviour of the algorithm more in line with the behaviour sketched in the example in section 1. This change is brought about by reducing the worst case matching time of the algorithm which is $\mathcal{O}(|S|^2)$, assuming that intersection with $P$ is a $\mathcal{O}(1)$ operation.

Consider the set of suffixes of keywords **suff**$(P)$ (function **suff** is defined in definition A.2). A string $w$ is an element of **suff**$(P)$ if and only if it can be extended on the left to a pattern in $P$, i.e. $\big(\exists w' : w' \in V^* : w'w \in P\big)$. It follows that if $w \notin$ **suff**$(P)$ any extension of $w$ on the left is not an element of **suff**$(P)$ either. Consequently, the inner repetition in algorithm 2.2 can terminate as soon as $(l\restriction 1)v \notin$ **suff**$(P)$ holds, since then all suffixes of $u$ that are equal to or longer than $(l\restriction 1)v$ are not in **suff**$(P)$ and hence not in $P$. The inner repetition guard is therefore strengthened to

$$l \neq \varepsilon \textbf{ cand } (l\restriction 1)v \in \textbf{suff}(P).$$

Observe that $v \in$ **suff**$(P)$ is an now invariant of the inner repetition. This invariant is initially established by the assignment $v := \varepsilon$ since $P \neq \varnothing$ and thus $\varepsilon \in$ **suff**$(P)$. The adapted algorithm—not given here—has $\mathcal{O}(|S| \cdot \big(\mathbf{MAX}p : p \in P : |p|\big))$ matching time. It will serve as a starting point for the derivation of all algorithms in the following section.

## 3 Sublinear pattern matching algorithms

In this section we derive sublinear pattern matching algorithms starting with algorithm 2.2 —with guard $l \neq \varepsilon$ replaced by $l \neq \varepsilon$ **cand** $(l\restriction 1)v \in$ **suff**$(P)$—by exploring the possibility

of safely (without missing matches) making shifts of more than one symbol, i.e. replacing assignment $u, r := u(r\uparrow 1), r\downarrow 1$ by assignment $u, r := u(r\uparrow k), r\downarrow k$ for some $k$ satisfying

$$1 \leq k \leq \big(\mathbf{MIN}\, n : 1 \leq n \wedge \mathbf{suff}(u(r\uparrow n)) \cap P \neq \varnothing : n\big).$$

The upperbound is the distance to the next match, the maximal safe shift distance. A number $k$ satisfying this condition is called a *safe shift distance.* Since computing the upperbound on $k$ is essentially the same as the problem we are trying to solve we aim at easier to compute approximations from below of the upperbound. These are derived by systematically weakening the predicate $\mathbf{suff}(u(r\uparrow n)) \cap P \neq \varnothing$ in the range of the upperbound, weakening resulting predicates, applying rules for minimum and maximum over a disjunctive or conjunctive domain, and enlarging domains. Considerations that play a role in these derivations are, for instance, whether or not to look ahead at symbols of the unscanned part of the input string, whether to use or not to use information on the last scanned symbol (usually a non-matching symbol), and the extent to which this information is coupled with the information on the recognized suffix. Thus, several algorithms are obtained amongst which the generalized version of the Boyer-Moore algorithm [BM77], the Commentz-Walter algorithm [CW79a, CW79b], and the algorithm by Fan and Su [FS93, FS94]. We derive ever smaller shift functions since the smaller the shift function the less precomputation time and storage space for the functions constituting the shift function is usually needed. For instance, the algorithm by Fan and Su [FS93, FS94] is faster than the Commentz-Walter algorithm [CW79a, CW79b], but it needs a two dimensional table to store one of the functions constituting its shift function whereas the Commentz-Walter algorithm only needs one dimensional tables.

In the derivations we use part of the postcondition of the inner repetition in the adapted version of algorithm 2.2 ($u = lv \wedge v \in \mathbf{suff}(P)$). Adding $l, v := \varepsilon, \varepsilon$ to the initial assignments turns $u = lv \wedge v \in \mathbf{suff}(P)$ into an invariant of the outer repetition. Due to the dependence of the upperbound on $l$, $v$, and $r$ we will aim at shift functions $k$ that depend on $l$, $v$, and $r$ and write $k(l,v,r)$. Hence, we arrive at the following algorithm scheme for all algorithms to be derived in this section:

**Algorithm 3.1**

---

$u, r := \varepsilon, S;\ O := \{\varepsilon\} \times (\{\varepsilon\} \cap P) \times \{S\};\ l, v := \varepsilon, \varepsilon;$
$\{$ invariant: $O = \big(\cup\, x, y, z : xyz = S \wedge xy \leq_p u : \{x\} \times (\{y\} \cap P) \times \{z\}\big)$
$\wedge\ u = lv \wedge v \in \mathbf{suff}(P) \wedge (l = \varepsilon\ \mathbf{cor}\ (l\uparrow 1)v \notin \mathbf{suff}(P))\ \}$
**do** $r \neq \varepsilon \rightarrow u, r := u(r\uparrow k(l,v,r)), r\downarrow k(l,v,r);$
$l, v := u, \varepsilon;\ O := O \cup \{u\} \times (\{\varepsilon\} \cap P) \times \{r\};$
**do** $l \neq \varepsilon$ **cand** $(l\uparrow 1)v \in \mathbf{suff}(P) \rightarrow l, v := l\downarrow 1, (l\uparrow 1)v;$
$O := O \cup \{l\} \times (\{v\} \cap P) \times \{r\}$
**od**
**od**$\{\ R\ \}$

---

Particular algorithms are obtained by substituting their shift functions for $k(l,v,r)$. Such a substitution may not always yield an algorithm that exactly corresponds with its original description in the literature; sometimes an additional transformation of the resulting algorithm is needed (for instance, a phase shift of the repetition; see [WZ92] for a phase shifted version of the algorithm scheme). Conjunct $l = \varepsilon$ **cor** $(l{\uparrow}1)v \notin \mathbf{suff}(P)$ is added to the invariant in order to stress that provided $l$ is nonempty symbol $l{\uparrow}1$ is non-matching.

Precomputation of the functions introduced in the subsequent subsections is not discussed in this paper. A uniform treatment of the precomputation can be found in [WZ95].

### 3.1 No lookahead at the unscanned part of the input string

It can be shown that

$$\begin{aligned}&\big(\mathbf{MIN}\, n : 1 \le n \wedge \mathbf{suff}\big(u(r{\uparrow}n)\big) \cap P \neq \varnothing : n\big)\\ &\quad \ge \big(\mathbf{MIN}\, n : 1 \le n \wedge \mathbf{suff}(uV^n) \cap P \neq \varnothing : n\big)\,.\end{aligned}$$

This inequality yields an approximation from below of the upperbound on $k$ that does not depend on $r$. In terms of algorithms this means that we refrain from looking ahead at the symbols of $r$, the yet unscanned part of the input string. Since this is in accordance with most of the algorithms we are aiming at, it will be the starting point of most of our further derivations (one symbol lookahead at the unscanned part of the input string is discussed in subsection 3.8). In view of the form of this approximation we will aim at shift functions $k$ being dependent only on $u$, i.e. on $l$ and $v$ (remember $u = lv \wedge v \in \mathbf{suff}(P)$). We will write $k(l,v)$ instead of $k(l,v,r)$.

### 3.2 Restriction to one symbol lookahead

In all derivations in this subsection and the following subsections we assume

$$u = lv \wedge v \in \mathbf{suff}(P).$$

Restriction to one symbol lookahead ($l{\uparrow}1$, the last symbol of $u$ scanned in the inner loop) leads to the algorithm by Fan and Su [FS93, FS94]. It is obtained by weakening the predicate in the approximation of the upperbound from subsection 3.1 in the following way:

$$\begin{aligned}&\mathbf{suff}(uV^n) \cap P \neq \varnothing\\ =\quad &\{\, u = lv \,\}\end{aligned}$$

$$
\begin{array}{ll}
 & \mathbf{suff}(lvV^n) \cap P \neq \varnothing \\
\Rightarrow & \quad \{\, l = (l\lfloor 1)(l\lceil 1),\ l\lfloor 1 \in V^*,\ \text{monotonicity of } \mathbf{suff} \text{ and } \cap \,\} \\
 & \mathbf{suff}\big(V^*(l\lceil 1)vV^n\big) \cap P \neq \varnothing \\
= & \quad \{\, \text{property A.4} \,\} \\
 & V^*(l\lceil 1)vV^n \cap V^*P \neq \varnothing \\
= & \quad \{\, l = \varepsilon\text{: property A.5.i; } l \neq \varepsilon\text{: property A.5.ii} \,\} \\
 & V^*(l\lceil 1)vV^n \cap P \neq \varnothing \vee vV^n \cap V^*P \neq \varnothing
\end{array}
$$

Notice that we have obtained a weaker predicate solely by discarding any information on $l\lfloor 1$. The only information on $l$ that is still taken into account is $l\lceil 1$ being either empty or consisting of one symbol. In the latter case we say to have one symbol lookahead. Observe that the symbol is the last symbol of $u$ scanned in the inner loop and that it is a non-matching symbol. After substituting the weaker predicate we obtain shift distance $k_{opt}(l,v)$ where $k_{opt} \in V^* \times \mathbf{suff}(P) \to \mathbb{N}$ is defined for $x \in V^*$ and $y \in \mathbf{suff}(P)$ by

$$k_{opt}(x,y) = \big(\mathbf{MIN}\, n : n \geq 1 \wedge \big(V^*(x\lceil 1)yV^n \cap P \neq \varnothing \vee yV^n \cap V^*P \neq \varnothing\big) : n\big).$$

Function $k_{opt}$ can be expressed as follows

$$k_{opt}(x,y) = \begin{cases} d_{opt}(x\lceil 1, y)\ \mathbf{min}\ d_{sp}(y) & x \neq \varepsilon \\ d_i(y)\ \mathbf{min}\ d_{sp}(y) & x = \varepsilon \end{cases}$$

where $d_{opt} \in V \times \mathbf{suff}(P) \to \mathbb{N}$ is defined by

$$d_{opt}(a,y) = \big(\mathbf{MIN}\, n : n \geq 1 \wedge V^*ayV^n \cap P \neq \varnothing : n\big) \qquad (a \in V, y \in \mathbf{suff}(P)),$$

$d_{sp} \in \mathbf{suff}(P) \to \mathbb{N}$ is defined by

$$d_{sp}(y) = \big(\mathbf{MIN}\, n : n \geq 1 \wedge yV^n \cap V^*P \neq \varnothing : n\big) \qquad (y \in \mathbf{suff}(P)),$$

(function $d_2$ in [CW79a, CW79b]), and $d_i \in \mathbf{suff}(P) \to \mathbb{N}$ is defined by

$$d_i(y) = \big(\mathbf{MIN}\, n : n \geq 1 \wedge V^*yV^n \cap P \neq \varnothing : n\big) \qquad (y \in \mathbf{suff}(P)),$$

(function $d_1$ in [CW79a, CW79b]). Functions $d_{opt}$ and $d_i$ account for occurrences of $ay$ and $y$, respectively, within some keyword (i.e. as infix of some keyword), whereas function $d_{sp}$ accounts for occurrences of suffixes of $y$ as proper prefixes of some keyword.

We arrived at this algorithm not knowing it had already been described by Fan and Su in [FS93, FS94]. From their informal description it undoubtedly follows that they describe the same algorithm though their formal treatment of the algorithm and, especially, the precomputation is rather involved. Finally, notice that to store function $d_{opt}$ one needs a two dimensional table, whereas functions $d_i$ and $d_{sp}$ only need one dimensional tables. In the following subsections we derive shift functions smaller than $k_{opt}$ that are expressed solely in functions needing one dimensional tables for storage.

### 3.3 Lookahead symbol is mismatching

We derive an approximation from below of $d_{opt}$ that yields an algorithm that is the common ancestor of the multiple keyword generalization of the Boyer-Moore algorithm [BM77] and the Commentz-Walter algorithm [CW79a, CW79b]. Essentially, the resulting shift function is not based on the identity of the lookahead symbol $l\lceil 1$ but only uses the fact that the lookahead symbol is mismatching, as is done in the Boyer-Moore shift function. In this way one might say that the recognized suffix and the (mismatching) lookahead symbol have to some extent been decoupled.

We start by weakening the predicate from $d_{opt}$. Assume $l \neq \varepsilon$ and $(l\lceil 1)v \notin \mathbf{suff}(P)$. We derive

$$
\begin{array}{ll}
 & V^*(l\lceil 1)vV^n \cap P \neq \varnothing \\
= & \quad \{\, v \in V^{|v|},\ \text{monotonicity of } \cap \,\} \\
 & V^*(l\lceil 1)V^{|v|+n} \cap P \neq \varnothing \wedge V^*(l\lceil 1)vV^n \cap P \neq \varnothing \\
\Rightarrow & \quad \{\, (l\lceil 1)v \notin \mathbf{suff}(P),\ \text{so } l\lceil 1 \in \{\, a \mid a \in V \wedge av \notin \mathbf{suff}(P) \,\},\ \text{definition } MS \,\} \\
 & V^*(l\lceil 1)V^{|v|+n} \cap P \neq \varnothing \wedge V^*\big(V \setminus MS(v)\big)vV^n \cap P \neq \varnothing
\end{array}
$$

where $MS \in \mathbf{suff}(P) \to \mathcal{P}(V)$ is defined by

$$MS(y) = \{\, a \mid a \in V \wedge ay \in \mathbf{suff}(P) \,\} \qquad (y \in \mathbf{suff}(P)).$$

The first conjunct will lead to a shift component based on the identity of the lookahead symbol that is identical to a component of the Commentz-Walter shift function. The second conjunct will lead to a shift component—based on the recognized suffix and the fact that the lookahead symbol is mismatching—that is identical to a component of the Boyer-Moore shift function. Replacing the range predicate of $d_{opt}$ by the last predicate of the preceding derivation we proceed

$$
\begin{array}{ll}
 & \big(\mathbf{MIN}\, n : n \geq 1 \wedge V^*(l\lceil 1)V^{|v|+n} \cap P \neq \varnothing \wedge V^*\big(V \setminus MS(v)\big)vV^n \cap P \neq \varnothing : n\big) \\
\geq & \quad \{\, \mathbf{MIN} \text{ with conjunctive range} \,\}
\end{array}
$$

$$
\begin{aligned}
&\big(\mathbf{MIN}\, n : n \geq 1 \wedge V^*(l\!\uparrow\!1)V^{|v|+n} \cap P \neq \varnothing : n\big) \\
&\mathbf{max}\, \big(\mathbf{MIN}\, n : n \geq 1 \wedge V^*\big(V \setminus MS(v)\big)vV^n \cap P \neq \varnothing : n\big) \\
= \quad & \{\text{change of bound variable: } m = |v| + n\text{; definition } d_{vi} \text{ (after derivation)}\} \\
&\big(\mathbf{MIN}\, m : m \geq |v| + 1 \wedge V^*(l\!\uparrow\!1)V^m \cap P \neq \varnothing : m - |v|\big)\, \mathbf{max}\, d_{vi}(v) \\
\geq \quad & \{\text{enlarging domain}\} \\
&\big(\mathbf{MIN}\, m : m \geq 1 \wedge V^*(l\!\uparrow\!1)V^m \cap P \neq \varnothing : m - |v|\big)\, \mathbf{max}\, d_{vi}(v) \\
= \quad & \{l \neq \varepsilon\text{, definition of } char_{cw} \text{ (after derivation)}\} \\
&char_{cw}(l\!\uparrow\!1, |v|)\, \mathbf{max}\, d_{vi}(v)
\end{aligned}
$$

where $d_{vi} \in \mathbf{suff}(P) \to \mathbb{N}$ is defined by

$$d_{vi}(y) = \big(\mathbf{MIN}\, n : n \geq 1 \wedge V^*\big(V \setminus MS(y)\big)yV^n \cap P \neq \varnothing : n\big) \qquad (y \in \mathbf{suff}(P))$$

and $char_{cw} \in V \times \mathbb{N} \to \mathbb{N}$ is defined by

$$char_{cw}(a, z) = \big(\mathbf{MIN}\, n : n \geq 1 \wedge V^*aV^n \cap P \neq \varnothing : n - z\big) \qquad (a \in V, z \in \mathbb{N}).$$

This results in shift distance $k_{bmcw}(l,v)$ where $k_{bmcw} \in V^* \times \mathbf{suff}(P) \to \mathbb{N}$ is defined by

$$k_{bmcw}(x, y) = \begin{cases} \big(char_{cw}(x\!\uparrow\!1, |y|)\, \mathbf{max}\, d_{vi}(y)\big)\, \mathbf{min}\, d_{sp}(y) & (x \in V^+, y \in \mathbf{suff}(P)) \\ d_i(y)\, \mathbf{min}\, d_{sp}(y) & (x = \varepsilon, y \in \mathbf{suff}(P)). \end{cases}$$

Notice that we have

$$k_{opt}(x, y) \geq k_{bmcw}(x, y) \qquad (x \in V^*, y \in \mathbf{suff}(P)).$$

We chose the name $k_{bmcw}$ to reflect that essential ideas from both the Boyer-Moore and Commentz-Walter algorithms are used. In the next two subsections these algorithms are derived from the algorithm obtained in this subsection.

### 3.4 The multiple keyword Boyer-Moore algorithm

We proceed by deriving the multiple keyword generalization of the Boyer-Moore algorithm [BM77] from the algorithm in subsection 3.3. It only differs from the algorithm there in the way the lookahead symbol is taken into account. Assuming $l \neq \varepsilon$ we have—since $V^*(l\!\uparrow\!1)V^n \cap P \neq \varnothing \Rightarrow V^*(l\!\uparrow\!1)V^n \cap V^*P \neq \varnothing$ and $P \neq \varnothing$—

$$char_{cw}(l\!\uparrow\!1, |v|) \geq char_{bm}(l\!\uparrow\!1) - |v|$$

where $char_{bm} \in V \rightarrow \mathbb{N}$ is defined by

$$char_{bm}(a) = \big(\mathbf{MIN}\, n : n \geq 1 \wedge V^* a V^n \cap V^* P \neq \varnothing : n\big) \qquad (a \in V).$$

It results in shift distance $k_{bm}(l,v)$ where $k_{bm} \in V^* \times \mathbf{suff}(P) \rightarrow \mathbb{N}$ is defined by

$$k_{bm}(x,y) = \begin{cases} \big((char_{bm}(x\uparrow 1) - |y|)\,\mathbf{max}\, d_{vi}(y)\big)\,\mathbf{min}\, d_{sp}(y) & (x \in V^+, y \in \mathbf{suff}(P)) \\ d_i(y)\,\mathbf{min}\, d_{sp}(y) & (x = \varepsilon, y \in \mathbf{suff}(P)). \end{cases}$$

We obtain the multiple keyword generalization of the Boyer-Moore algorithm [BM77]. The standard Boyer-Moore algorithm can be obtained by restricting $P$ to one keyword. Notice that we have

$$k_{bmcw}(x,y) \geq k_{bm}(x,y) \qquad (x \in V^*, y \in \mathbf{suff}(P)).$$

Inequality can only occur if the lookahead symbol does not occur in any keyword except as the last symbol.

The formula for the Boyer-Moore shift function given here differs from the one given in [BM77]. It can be shown that

$$k_{bm}(x,y) = (char_{bm}(x\uparrow 1) - |y|)\,\mathbf{max}\big(d_{vi}(y)\,\mathbf{min}\, d_{sp}(y)\big).$$

This formula coincides with the one in [BM77].

### 3.5 The Commentz-Walter algorithm

Instead of approximating $char_{cw}$ in $k_{bmcw}$ from below by $char_{bm}$ we now approximate $d_{vi}$ in $k_{bmcw}$ from below by $d_i$. This results in the Commentz-Walter algorithm [CW79a, CW79b]. We derive

$$\begin{aligned}
& d_{vi}(v) \\
= \quad & \{\text{definition } d_{vi}\} \\
& \big(\mathbf{MIN}\, n : n \geq 1 \wedge V^*\big(V \setminus MS(v)\big) v V^n \cap P \neq \varnothing : n\big) \\
\geq \quad & \{V^*\big(V \setminus MS(v)\big) v V^n \cap P \neq \varnothing \Rightarrow V^* v V^n \cap P \neq \varnothing\} \\
& \big(\mathbf{MIN}\, n : n \geq 1 \wedge V^* v V^n \cap P \neq \varnothing : n\big) \\
= \quad & \{\text{definition } d_i\} \\
& d_i(v).
\end{aligned}$$

Notice that the weakening is done by no longer requiring that $v$ is preceded by a non-matching symbol (i.e. a symbol from $V \setminus MS(v)$). This results in shift distance $k_{cw}(l,v)$ where $k_{cw} \in V^* \times \mathbf{suff}(P) \rightarrow \mathbb{N}$ is defined by

$$k_{cw}(x,y) = \begin{cases} \big(char_{cw}(x\lceil 1, |y|) \max d_i(y)\big) \min d_{sp}(y) & (x \in V^+, y \in \mathbf{suff}(P)) \\ d_i(y) \min d_{sp}(y) & (x = \varepsilon, y \in \mathbf{suff}(P)) \end{cases}$$

being the shift distance of the Commentz-Walter algorithm [CW79a, CW79b]. Notice that we have

$$k_{bmcw}(x,y) \geq k_{cw}(x,y) \qquad (x \in V^*, y \in \mathbf{suff}(P)).$$

Such a comparison can not be made between $k_{bm}$ and $k_{cw}$ as the following example shows.

**Example 3.1** Let $V \in \{a,b,c,d\}$, $P = \{cababa\}$, and $x \in V^*$. Shift functions $k_{bm}$ and $k_{cw}$ are incomparable since

$$\begin{array}{rcrcl} k_{opt}(xd,a) & = & +\infty \min 6 & = & 6 \\ k_{bmcw}(xd,a) & = & (+\infty \max 4) \min 6 & = & 6 \\ k_{bm}(xd,a) & = & \big((6-1) \max 4\big) \min 6 & = & 5 \\ k_{cw}(xd,a) & = & (+\infty \max 2) \min 6 & = & 6 \end{array}$$

and

$$\begin{array}{rcrcl} k_{opt}(xa,a) & = & +\infty \min 6 & = & 6 \\ k_{bmcw}(xa,a) & = & \big((2-1) \max 4\big) \min 6 & = & 4 \\ k_{bm}(xa,a) & = & \big((2-1) \max 4\big) \min 6 & = & 4 \\ k_{cw}(xa,a) & = & \big((2-1) \max 2\big) \min 6 & = & 2. \end{array}$$

It also follows that in some cases $k_{bmcw}$ is smaller than $k_{opt}$ and that in some cases $k_{bm}$ and $k_{cw}$ are smaller than $k_{bmcw}$. □

It is not possible that both $k_{bmcw}(x,y) > k_{bm}(x,y)$ and $k_{bmcw}(x,y) > k_{cw}(x,y)$ hold for some $x \in V^+$ and $y \in \mathbf{suff}(P)$ since the first inequality implies $char_{cw}(x\lceil 1, |y|) = +\infty$ and this in its turn implies $k_{bmcw}(x,y) = d_{sp}(y) = k_{cw}(x,y)$.

### 3.6 Complete decoupling of recognized suffix and lookahead symbol

The derivations in the previous subsections effect an ever stronger decoupling of the recognized suffix $v$ and the lookahead symbol $l\lceil 1$ in the subsequent shift functions. By approximating $d_{vi}$ in $k_{bm}$ from below by $d_i$ or $char_{cw}$ in $k_{cw}$ by $char_{bm}$ (or both in $k_{bmcw}$)

we obtain a complete decoupling. It results in shift distance $k_{dsl}(l,v)$ where function $k_{dsl} \in V^* \times \mathbf{suff}(P) \to \mathbb{N}$ is defined by

$$k_{dsl}(x,y) = \begin{cases} \big((char_{bm}(x\lceil 1) - |y|) \,\mathbf{max}\, d_i(y)\big) \,\mathbf{min}\, d_{sp}(y) & (x \in V^+, y \in \mathbf{suff}(P)) \\ d_i(y) \,\mathbf{min}\, d_{sp}(y) & (x = \varepsilon, y \in \mathbf{suff}(P)). \end{cases}$$

The algorithm obtained is a common descendant of both the Boyer-Moore and Commentz-Walter algorithms.

### 3.7 Discarding the lookahead symbol

The predicate in the range of $k_{opt}$ is weakened by substituting its first disjunct by the weaker predicate $V^*vV^n \cap P \neq \varnothing$. This is due to $V^*(l\lceil 1) \subseteq V^*$ and the monotonicity of $\cap$. This weakening step is referred to as discarding the lookahead symbol $l\lceil 1$. The shift distance corresponding to this weakening is $k_{nla}(v)$ where $k_{nla} \in \mathbf{suff}(P) \to \mathbb{N}$ is defined by

$$k_{nla}(y) = d_i(y) \,\mathbf{min}\, d_{sp}(y) \qquad (y \in \mathbf{suff}(P)).$$

Notice that this shift function can also be viewed as an approximation from below of $k_{dsl}$.

### 3.8 One symbol lookahead at the unscanned part of the input string

We now consider looking ahead at the first symbol of the unscanned part $r$ of the input string. The first symbol of $r$ will be taken into account independently of the other available information. In this way we obtain stronger variants of all of the shift functions derived thus far. Assuming $r \neq \varepsilon$ we derive

$$\begin{aligned}
& \big(\mathbf{MIN}\, n : 1 \leq n \wedge \mathbf{suff}\big(u(r\lceil n)\big) \cap P \neq \varnothing : n\big) \\
= \quad & \{\text{domain split, } 1 \leq n \leq |r|\text{: } r\lceil n = (r\lceil 1)\big((r\lfloor 1)\rceil(n-1)\big),\ |r| < n\text{: } r\lceil n = r\,\} \\
& \big(\mathbf{MIN}\, n : 1 \leq n \leq |r| \wedge \mathbf{suff}\big(u(r\lceil 1)\big((r\lfloor 1)\rceil(n-1)\big)\big) \cap P \neq \varnothing : n\big) \\
& \mathbf{min}\, \big(\mathbf{MIN}\, n : |r| < n \wedge \mathbf{suff}(ur) \cap P \neq \varnothing : n\big) \\
\geq \quad & \{1 \leq n \leq |r|\text{: } (r\lfloor 1)\rceil(n-1) \in V^{n-1},\ \text{monotonicity of } \mathbf{suff} \text{ and } \mathit{cap}\,\} \\
& \big(\mathbf{MIN}\, n : 1 \leq n \leq |r| \wedge \mathbf{suff}\big(u(r\lceil 1)V^{n-1}\big) \cap P \neq \varnothing : n\big) \\
& \mathbf{min}\, \big(\mathbf{MIN}\, n : |r| < n \wedge \mathbf{suff}(ur) \cap P \neq \varnothing : n\big) \\
= \quad & \{r \neq \varepsilon,\ r \in (r\lceil 1)V^{|r|-1},\ \mathbf{suff}(ur) \cap P \neq \varnothing \Rightarrow \mathbf{suff}\big(u(r\lceil 1)V^{|r|-1}\big) \cap P \neq \varnothing\,\} \\
& \big(\mathbf{MIN}\, n : 1 \leq n \leq |r| \wedge \mathbf{suff}\big(u(r\lceil 1)V^{n-1}\big) \cap P \neq \varnothing : n\big)
\end{aligned}$$

$$
\begin{aligned}
&\geq \quad \{\, u \in V^*,\ \text{monotonicity of } \mathbf{suff} \text{ and } \cap \,\} \\
&\quad \big(\mathbf{MIN}\, n : 1 \leq n \leq |r| \wedge \mathbf{suff}\big(V^*(r\!\restriction\! 1)V^{n-1}\big) \cap P \neq \varnothing : n\big) \\
&\geq \quad \{\, \text{enlarging domain, changing bound variable: } m = n-1 \,\} \\
&\quad \big(\mathbf{MIN}\, m : 0 \leq m \wedge \mathbf{suff}\big(V^*(r\!\restriction\! 1)V^{m}\big) \cap P \neq \varnothing : m+1\big) \\
&= \quad \{\, \text{property A.4, } P \neq \varnothing,\ V^*(r\!\restriction\! 1)V^{|p|} \cap V^*p \neq \varnothing\ (p \in P),\ \text{nonempty domain} \,\} \\
&\quad \big(\mathbf{MIN}\, m : 0 \leq m \wedge V^*(r\!\restriction\! 1)V^{m} \cap V^*P \neq \varnothing : m\big) + 1 \\
&= \quad \{\, \text{definition } \mathit{char}_{la} \text{ (after derivation)} \,\} \\
&\quad \mathit{char}_{la}(r\!\restriction\! 1) + 1
\end{aligned}
$$

where $\mathit{char}_{la} \in V \rightarrow \mathbb{N}$ is defined by

$$\mathit{char}_{la}(a) = \big(\mathbf{MIN}\, n : 0 \leq n \wedge V^*aV^n \cap V^*P \neq \varnothing : n\big) \quad (a \in V).$$

Let $M(u,r)$ denote the first expression in the preceding derivation (the maximal safe shift distance), and let $N(u)$ denote its approximation from below introduced in subsection 3.1. We then have

$$
\begin{aligned}
&\quad M(u,r) \\
&= \quad \{\, \text{property } \mathbf{max} \,\} \\
&\quad M(u,r)\ \mathbf{max}\ M(u,r) \\
&\geq \quad \{\, \text{inequality in subsection 3.1, preceding derivation} \,\} \\
&\quad N(u)\ \mathbf{max}\big(\mathit{char}_{la}(r\!\restriction\! 1) + 1\big)
\end{aligned}
$$

Since all shift functions derived in the previous subsections are approximations from below of $N(u)$ the above derivation shows they all may be extended with $\mathbf{max}(\mathit{char}_{la}(r\!\restriction\! 1)+1)$ to form a class of stronger shift functions of signature $k(l,v,r)$. The first derivation in this subsection shows that it is also possible to couple the information on $r\!\restriction\! 1$ with the information on $l$ and $v$ ($u = lv$). We will not pursue that direction any further in this paper.

## 4 Conclusions

In this paper we systematically derived from a common starting point a number of sublinear keyword pattern matching algorithms, among which are the multiple keyword generalization of the single keyword Boyer-Moore algorithm [BM77], the Commentz-Walter

algorithm [CW79a, CW79b], and the algorithm presented by Fan and Su [FS93, FS94]. The algorithms are presented in a common framework permitting an easier comprehension of and a better comparison between the algorithms. Our approach was inspired by the method described by Jonkers [Jon83].

Introduction of the notion of safe shift distances proved to be essential for the derivation of the various algorithms. All algorithms are characterized by a—systematically derived and more or less easy to compute—approximation from below of the maximal safe shift distance, computation of the latter being equivalent to the keyword pattern matching problem itself. The systematic derivation provided a means to compare the algorithms and their matching speeds, and to get a better understanding of the algorithms and their interrelations. Perhaps this better understanding will help further the use of the algorithms from this family. Our derivations show the Commentz-Walter algorithm not to be the multiple keyword generalization of the Boyer-Moore algorithm (as was the original intention of Commentz-Walter) and that such a generalization can indeed be obtained. Of the algorithms presented the algorithm by Fan and Su [FS93, FS94] is the fastest (at the expense of additional precomputation time and additional storage requirements), followed by the common ancestor of the Boyer-Moore and Commentz-Walter algorithms, and then by both the multiple keyword generalization of the Boyer-Moore algorithm [BM77] and the Commentz-Walter algorithm [CW79a, CW79b]. The latter two are incomparable in matching speed. It is clear that we have not derived and presented all possible algorithms in this family of algorithms. Our derivation method, however, clearly indicates how yet other members of this family of algorithms may be derived.

## A Definitions and properties

This section provides a series of definitions and properties which are used throughout this paper. In the following let $V$ be an alphabet.

For a string $w \in V^*$ $w^R$ denotes the reversal of $w$. For any language $L \subseteq V^*$ we define $L^R = \{w^R \mid w \in L\}$ (the reversal of language $L$).

**Definition A.1** *The infix operators* $\uparrow, \downarrow, \lceil, \lfloor : V^* \times \mathbb{N} \to V^*$ *are defined by*

$$\begin{array}{rcll} v\uparrow 0 & = & \varepsilon & (v \in V^*) \\ \varepsilon\uparrow(k+1) & = & \varepsilon & (k \geq 0) \\ (aw)\uparrow(k+1) & = & a(w\uparrow k) & (k \geq 0, a \in V, w \in V^*) \\ v\downarrow 0 & = & v & (v \in V^*) \\ \varepsilon\downarrow(k+1) & = & \varepsilon & (k \geq 0) \\ (aw)\downarrow(k+1) & = & w\downarrow k & (k \geq 0, a \in W, w \in V^*) \end{array}$$

*Define* $\lceil$ *as* $v\lceil k = (v^R\uparrow k)^R$ *and* $\lfloor$ *as* $v\lfloor k = (v^R\downarrow k)^R$. *The operators* $\uparrow, \downarrow, \lceil,$ *and* $\lfloor$ *are called "left take," "left drop," "right take," and "right drop" respectively.* □

**Definition A.2** *Functions* **pref** : $\mathcal{P}(V^*) \to \mathcal{P}(V^*)$ *and* **suff** : $\mathcal{P}(V^*) \to \mathcal{P}(V^*)$ *are defined by*

$$\mathbf{pref}(L) = \{w \mid w \in V^* \wedge (\exists x : x \in V^* : wx \in L)\}$$

*and*

$$\mathbf{suff}(L) = \left(\mathbf{pref}(L^R)\right)^R = \{w \mid w \in V^* \wedge (\exists x : x \in V^* : xw \in L)\}.$$

□

For $w \in V^*$ we will write **pref**$(w)$ (**suff**$(w)$) instead of **pref**$(\{w\})$ (**suff**$(\{w\})$).

**Definition A.3** *The relations* $\leq_p$ *and* $\leq_s$ *over* $V^* \times V^*$ *are defined by* $u \leq_p v \equiv u \in \mathbf{pref}(v)$ *and* $u \leq_s v \equiv u \in \mathbf{suff}(v)$. □

**Property A.4** *Let* $A, B \subseteq V^*$. *Then* $\mathbf{pref}(A) \cap B \neq \varnothing \equiv A \cap BV^* \neq \varnothing$ *and* $\mathbf{suff}(A) \cap B \neq \varnothing \equiv A \cap V^*B \neq \varnothing$. □

**Property A.5** *Let* $A, B \subseteq V^*$ *and* $a \in V$. *Then*

*i.* $V^*A \cap V^*B \neq \varnothing \equiv V^*A \cap B \neq \varnothing \vee A \cap V^*B \neq \varnothing$

*ii.* $V^*aA \cap V^*B \neq \varnothing \equiv V^*aA \cap B \neq \varnothing \vee A \cap V^*B \neq \varnothing$

*iii.* $V^*A \cap V^*B \neq \varnothing \equiv V^*A \cap B \neq \varnothing \vee A \cap V^+B \neq \varnothing$

□

## References

[BM77] Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, October 1977.

[CW79a] Beate Commentz-Walter. A string matching algorithm fast on the average. In H.A. Maurer, editor, *Proceedings 6th International Colloquium on Automata, Languages and Programming*, volume 71 of *Lecture Notes in Computer Science*, pages 118–132. Springer, July 1979.

[CW79b] Beate Commentz-Walter. A string matching algorithm fast on the average. Technical Report TR 79.09.007, IBM-Germany, Scientific Center Heidelberg, September 1979.

[FS93] Jang-Jong Fan and Key-Yih Su. An efficient algorithm for matching multiple patterns. *IEEE Transactions on Knowledge and Data Engineering*, 5(2):339–351, April 1993.

[FS94] Jang-Jong Fan and Key-Yih Su. An efficient algorithm for matching multiple patterns. In Jun-ichi Aoe, editor, *Computer Algorithms: String Pattern Matching Strategies*. IEEE Computer Society Press, 1994.

[Jon83] H.B.M. Jonkers. *Abstraction, specification and implementation techniques, with an application to garbage collection.* Number 166 in Mathematical Centre Tracts. Mathematisch Centrum, Amsterdam, 1983.

[WZ92] Bruce W. Watson and Gerard Zwaan. A taxonomy of keyword pattern matching algorithms. Computing Science Notes 92/27, Eindhoven University of Technology, Eindhoven, the Netherlands, December 1992. The report is available at URL `ftp://ftp.win.tue.nl/pub/techreports/pi/pattm/taxonomy-/1st.edition/pattm.ps.gz`.

[WZ95] Bruce W. Watson and Gerard Zwaan. A taxonomy of sublinear multiple keyword pattern matching algorithms. Computing Science Reports 95/13, Eindhoven University of Technology, Eindhoven, the Netherlands, April 1995. The report is available at URL `ftp://ftp.win.tue.nl/pub/techreports/zwaan-/sublin.300dpi.ps.gz`.