Nieuwsbrief van de Nederlandse Vereniging voor Theoretische Informatica

Mieke Bruné, Jan Willem Klop, Jan Rutten (redactie)[*]

Inhoudsopgave

[*]CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands. Email: mieke@cwi.nl.

1 Van de Redactie

Beste NVTI-leden,
Graag bieden we u hiermee de NVTI Nieuwsbrief van 1998 aan. Naast de nuttige adreslijst (in bijgewerkte versie) vindt u ook als vast element de statuten, met o.a. de regels voor het kiezen van het volgende NVTI bestuur, een kwestie die nu aan de orde is zoals u in een recent email bericht hebt kunnen lezen. Ook het programma van de Theoriedag 1998 vindt u in dit nummer. Voorts vindt u bijdragen van de onderzoekscholen IPA, OZL en SIKS. Het doet ons genoegen dat ook dit jaar enkele van onze collega's bereid zijn gevonden een korte bijdrage te schrijven over een thema dat naar wij hopen en verwachten relevant en interessant is voor een groot deel van onze gemeenschap. Ook voor de volgende aflevering van de Nieuwsbrief zal de redactie zoeken naar auteurs van dergelijke bijdragen; we houden ons aanbevolen voor suggesties voor onderwerpen en auteurs. Ook dit jaar is de Theoriedag van 13 maart, alsmede dit Nieuwsbrief- nummer, weer tot stand gekomen dank zij externe steun en sponsoring. Hiervoor danken we SION, de onderzoekschool IPA, en Elsevier's Publishing Company.

De redactie,
Mieke Bruné (mieke@cwi.nl)
Jan Willem Klop (jwk@cwi.nl)
Jan Rutten (janr@cwi.nl)

2 Samenstelling Bestuur

Prof.dr. J.C.M. Baeten (TUE)
Prof.dr. J.W. Klop (VUA/CWI) secretaris
Prof.dr. J.N. Kok (RUL)
Prof.dr. G. Rozenberg (RUL) voorzitter
Dr. J.J.M.M. Rutten (CWI)
Dr. J. Torenvliet (UvA)

3 Van de voorzitter

Geacht NVTI-lid,

De Nederlandse Vereniging voor Theoretische Informatica (NVTI) is bedoeld om de belangen te behartigen van alle geïnteresseerden in de theoretische informatica en in het bijzonder om te dienen als een communicatieforum voor deze gemeenschap. De NVTI-nieuwsbrief, samen met de verenigingsdagen en de web site (http://www.cwi.nl/NVTI/), speelt hierbij een belangrijke rol. De nieuwsbrief bevat veel informatie die, naar onze mening, van belang is voor onze gemeenschap.

Uw commentaar op de nieuwsbrief stellen wij zeer op prijs. Voor een goed functioneren van de NVTI is het noodzakelijk dat er creatieve ideeën komen van de zijde van de leden. Met nadruk nodigen wij iedereen uit om mee te denken en ideeën en voorstellen te sturen aan een van de leden van het NVTI-bestuur.

Zoals u waarschijnlijk al weet, vindt op 13 maart de Theoriedag plaats. Voor deze dag is een interessant programma opgesteld (zoals in deze nieuwsbrief te lezen is). Ook wordt op deze Theoriedag het nieuwe NVTI-bestuur gekozen.

Ik hoop velen van u te ontmoeten op 13 maart.

G. Rozenberg, voorzitter NVTI

4 Theoriedag 1998

Vrijdag 13 maart 1998, Jaarbeurs Congrescentrum Utrecht

Het is ons een genoegen u uit te nodigen tot het bijwonen van de Theoriedag 98 van de NVTI, de Nederlandse Vereniging voor Theoretische Informatica, die zich ten doel stelt de theoretische informatica te bevorderen en haar beoefening en toepassingen aan te moedigen. De Theoriedag 98 zal gehouden worden op vrijdag 13 maart 1998 in het Jaarbeurs Congrescentrum te Utrecht, en is een voortzetting van de reeks jaarlijkse bijeenkomsten van de NVTI die drie jaar geleden met de oprichtingsbijeenkomst begon. Evenals vorige jaren hebben wij een aantal prominente sprekers uit binnen- en buitenland bereid gevonden deze dag gestalte te geven door voordrachten over recente en belangrijke stromingen in de theoretische informatica (zie hieronder). Naast deze wetenschappelijke inhoud heeft de dag ook een informatief gedeelte, in de vorm van een algemene vergadering waarin de meest relevante informatie over de NVTI gegeven zal worden, alsmede presentaties van de onderzoekscholen OZL, IPA en SIKS. Verder zal er door Elsevier een korte beschrijving worden gegeven van de stand van zaken op het gebied van 'electronic publishing'.

Lidmaatschap NVTI
Alle leden van de voormalige WTI (Werkgemeenschap Theoretische Informatica) zijn automatisch lid van de NVTI geworden. Aan het lidmaatschap zijn geen kosten verbonden; u krijgt de aankondigingen van de NVTI per email of anderszins toegestuurd. Was u geen lid van de WTI en wilt u lid van de NVTI worden: u kunt zich aanmelden bij het contactadres beneden (M. Brune', CWI), met vermelding van de relevante gegevens, naam, voorletters, affiliatie indien van toepassing, correspondentieadres, email, telefoonnummer.

Programma

09.30-10.00:	Ontvangst met koffie
10.00-10.10:	Opening
10.10-11.00:	Lezing Prof.dr. M. Nielsen, BRICS
11.00-11.30:	Koffie
11.30-12.20:	Lezing Prof.dr. J.A. Bergstra, UvA/RUU
12.20-12.50:	Presentatie Elsevier
12.50-14.10:	Lunch
14.10-15.00:	Lezing Prof.dr. A. Apostolico, Purdue University/Padova
15.00-15.20:	Thee
15.20-16.10:	Lezing Prof.dr. E.H.L. Aarts, TUE/Philips
16.10-16.40:	Presentatie Onderzoeksscholen (OZL, IPA, SICS)
16.40-17.10:	Algemene ledenvergadering NVTI

Lunchdeelname
Het is mogelijk aan een georganiseerde lunch in het Jaarbeurs Congrescentrum deel te nemen; hiervoor is aanmelding verplicht. Dit kan per email of telefonisch bij Mieke Bruné (mieke@cwi.nl, 020-592 4249), tot een week tevoren (6 maart). De kosten kunnen ter plaatse voldaan worden; deze bedragen (ongeveer) f 25,-. Wij wijzen erop dat in de onmiddellijke nabijheid van de vergaderzaal ook uitstekende lunchfaciliteiten gevonden kunnen worden, voor wie niet aan de georganiseerde lunch wenst deel te nemen.

Abstracts van voordrachten

On Notions of bisimulation and associated logics

Prof.dr. M. Nielsen BRICS, Computer Science Department, University of Aarhus

Recently, there have been several attempts of abstract approaches to notions of behavioural equivalences for models of computation. The approach based on open maps introduced by Joyal, Nielsen and Winskel represents one of these, which has mainly been applied to models for concurrency. One virtue of this (and other approaches as well) is that one obtains, for general reasons, a number of desirable properties, e.g. the existence of characteristic games and logics. This talk will present some of the logics which have come out of the use of open maps. In particular, various versions of modal logics with past modalities for concurrent systems will be presented, with emphasis on model checking and their use as specification logics. Also, the use of the approach to timed systems will be illustrated, if time permits.

Process Algebra and Many-valued Logics
by Jan Bergstra, Universiteit van Amsterdam/Universiteit Utrecht

This talk is about joint work with Alban Ponse (University of Amsterdam). A central notion in the theory of programming notations is the conditional construct which expresses that the progress of a computation depends on the validity of some proposition. The proposition, also called condition, may have parameters, may depend on a state or be contextsensitive in some other way. What interests us is the effect of the use of nonclassical truth values in the conditional construct. We will distinguish deterministic and nondeterministic truthvalues. The deterministic ones are: T(true), F(false), M(meaningless) and D(divergent), the nondeterministic ones are C(active choice) and N(passive choice). The deterministic values can be traced back to the work of Bochvar and Kleene around 1935 and MacCarty around 1950. We use a home-made integration of their 3-valued logics into a 4-valued one. This integration (Bergstra, Bethke, Rodenburg, 1996) left room for siginificant design decisions regarding both syntax and semantics.

This fourvalued logic can be naturally integrated into ACP style process algebra thus leading to a formulation of process algebra that deals with errors that emerge from data handling. Again some unexpected choices have to be made. The truth value C can be added to T and F thus leading to symmetric 3-valued logic. This logic can be extended with D, M or both leading to two 4-valued logics and a 5-valued logic. We use the phrase symmetry because McCarty's connectives are symmetric on C. Integrating this 5-valued logic with ACP leads to the conclusion that the alternative composition of ACP (CCS) is a special case of the conditional construct with C taken as the value of the condition. Then we introduce a second nondeterministic truth value N. It leads to extensions of all of the mentioned logics. These extensions range from a 3-valued to a 16-valued one. Again these can be integrated with ACP style process algebra. This academic exercise finally leads to an interesting perspective on the proper treatment of Dijkstra's cooperating sequential processes in the context of process algebra.

Local Search for WHIZZKIDS
Prof.dr. E.H.L. Aarts, Technische Universiteit Eindhoven/
Philips Research Laboratories

Local search methods constitute a successful and interesting class of generally applicable algorithms that can be used to handle hard combinatorial optimization problems. Application of a local search method presupposes the specification of a solution set, a cost function, and a neighborhood structure. Execution of the method proceeds through iteration among neighboring solutions and comparison of the difference in cost of subsequent solutions with a threshold. The nature of the thresholds can be attributed to the various types of local search methods that are known, such as iterative improvement, variable-depth search, simulated annealing, and tabu search. By extending the iteration process from point based to population based neighborhoods, the concept of local search may also include genetic algorithms.

In the lecture we briefly review some theoretical concepts of local search including some important complexity and convergence results. To demonstrate some of the issues that are relevant for application, we discuss the use of local search algorithms for the solution of the combinatorial problem issued by the WHIZZKIDS'97 contest.

5 Mededelingen van de onderzoekscholen

Hieronder volgen korte beschrijvingen van de onderzoekscholen:

- Instituut voor Programmatuurkunde en Algoritmiek;
- Landelijke Onderzoekschool Logica;
- School voor Informatie- en KennisSystemen;

5.1 Instituut voor Programmatuurkunde en Algoritmiek (IPA), door: Jos Baeten

1997 was een succesvol jaar voor de onderzoekschool IPA (Instituut voor Programmatuurkunde en Algoritmiek). We verworven erkenning door de KNAW, en het onderzoek in IPA werd over het algemeen zeer gunstig beoordeeld in de onderzoeksvisitatie van de VSNU.

In de onderzoekschool participeren nu ook onderzoekers van de Rijksuniversiteit Groningen (als lid) en de Universiteit van Amsterdam (geassocieerd), zodat het aantal universiteiten in IPA is toegenomen tot 8 (behalve de genoemde de Vrije Universiteit Amsterdam, de Technische Universiteit Eindhoven, de Rijksuniversiteit Leiden, de Katholieke Universiteit Nijmegen, de Universiteit Twente en de Universiteit Utrecht), en daarnaast zijn er participanten van het CWI te Amsterdam en Philips Research in Eindhoven.

Hieronder geven we een opsomming van een aantal bijeenkomsten georganiseerd door IPA of gesponsord door IPA.

Agenda

Third Dutch Proof Tools Day

6 maart 1998, De Uithof, Universiteit Utrecht.
ProTaGoNist is een Nederlandse werkgroep over bewijsgereedschappen. Een aantal groepen in IPA die belangstelling hebben voor stellingbewijzers en stellingcheckers participeren in deze werkgroep. Ook toepassingen van deze technologie voor het bewijzen van programmacorrectheid worden onderzocht. De "Third Dutch Proof Tools Day" wordt georganiseerd door ProTaGoNist met hulp van IPA, en bevat een aantal voordrachten over verschillende bewijsgereedschappen en een discussie.
URL: http://www.cs.ruu.nl/people/tanja/CFP.html.

IPA Lentedagen

over **Parallellisme** 27-28 april 1998, De Koningshof, Veldhoven.
Aan de orde komen parallelle berekeningen en praktische toepassingen van parallel rekenen. Er zijn sprekers uit de academische wereld en uit het bedrijfsleven.

IPA Feedbacks

over **Software Architectuur** 5 juni 1998, De Reehorst, Ede.
Deze dag is speciaal bedoeld voor interactie tussen mensen uit de wetenschap en het bedrijfsleven. Nadere informatie volgt.

User Interfaces for Theorem Provers

13-14 juli, Technische Universiteit Eindhoven.
Een internationale workshop over onderzoek naar de analyse en het ontwerp van user interfaces voor stellingbewijzers. In het bijzonder wordt wederzijdse bevruchting nagestreefd tussen het gebied van de mens-machine interactie (HCI) and bewijsgereedschappen.
URL: http://www.win.tue.nl/cs/ipa/uitp/.

EEF Summerschool

in **Cryptography & Data Security** 20-24 juli, Aarhus University (Denmark).
Een aantal leidende personen op dit gebied, uit de academische wereld en uit de industrie, geven overzichtslezingen en gespecialiseerde voordrachten over vele aspecten van de moderne cryptologie. Deze zomerschool wordt georganiseerd door het European Educational Forum (EEF), een gezamenlijk initiatief van de onderzoekscholen BRICS uit Denemarken, IPA en TUCS uit Finland.
URL: http://www.brics.dk/Activities/98/CryptDatSecSchool/.

EEF Summerschool

on **Foundations of Computer Science: Semantics** 10-21 augustus, Finland.
De volgende aflevering in de serie EEF zomerscholen over de grondslagen van de informatica, gebaseerd op de serie Handbooks of Logic in Computer Science van Oxford University Press. Nadere informatie volgt.

Evenementen in 1997

In 1997 hadden de IPA lentedagen als thema testen, en de herfstdagen gingen over Natural Computation. De EEF Summerschool on Foundations of Computer Science werd gehouden in Mierlo, en had als thema Computationele en Syntactische Methoden, en de Synergos Summerschool was in Finland, over Natural Computation.

De IPA cursussen voor beginnende promovendi zijn van start gegaan: in 1997 was de eerste cursus, over Formele Methoden. De cursus Software Technology was in februari 1998, en later in 1998 volgt nog de cursus Algoritmen en Complexiteit.

Addressen

Bezoekadres
Technische Universiteit Eindhoven
Hoofdgebouw HG 7.17
Den Dolech 2
5612 AZ Eindhoven

Postadres
IPA, Fac. Wiskunde & Informatica
Technische Universiteit Eindhoven
Postbus 513
5600 MB Eindhoven

tel. (040)-2474124 (IPA Secretariaat)
fax (040)-2463992
e-mail ipa@win.tue.nl
URL http://www.win.tue.nl/cs/ipa/

5.2 The Dutch Research School in Logic (OzsL), door: Jan van Eijck

Current State of the School

The Dutch Research School in Logic (OzsL) was granted a second 5 year lease of life by KNAW in 1997. In this second school period constituency of the school has changed somewhat. Current partipating institutes are the University of Amsterdam (ILLC), the Free University of Amsterdam, the University of Utrecht (Uil-OTS), the University of Groningen (BCN) and Tilburg University. The school maintains close working cooperations with neighbouring schools in the areas of mathematics, computer science and linguistics, while within the school logic functions as a meeting ground between these three areas.

Chairman of the school board is Gerard Renardel, while Jan van Eijck has succeeded Anne Troelstra as scientific director of the school. In December 1998 Erik-Jan van der Linden, manager of the school, has resigned from his position to start his own IT consultancy firm. The new manager of the school is Peter Blok (`pblok@wins.uva.nl`). The email address of the school is `ozsl.wins.uva.nl`, while the School WWW page can be found at:

`www.wins.uva.nl/research/ozsl/lin`.

Events

The curriculum of the Dutch Research School in Logic consists of two school weeks, to be held each Spring and Autumn, plus additional master courses of longer duration, mostly by eminent visitors from abroad, and the yearly European Summer Schools in Logic, Language and Information (ESSLLI).

This Spring School Week will be held at CWI, March 2–5. The School Week offers courses on Finite Model Theory (prof Dawar), on Proof Complexity (prof Buss), on Computational Linguistics (Bouma and Van Noord), and on Automated Reasoning (de Nivelle and o Nuellain). Further information about the program and the organizational details is available through the School WWW page.

This Spring, master classes will be given by professor Wim Blok (algebraic logic), professor Grigori Mints (proof theory), and professor Hainal Andreka, professor Istvan Nemeti and associates (the logic of relativity theory). Further information can be obtained through the School office (contact Marco de Vries `marco@wins.uva.nl`) and via the School WWW page.

The ESSLLI Summer School of 1998 will be held in Saarbruecken, August 17–28. Details about the program can be found via WWW at `coli.uni-sb.de/~esslli/`.

Subscriptions to the school bulletin, LIN (Logic in the Netherlands) are available through Marco de Vries (`marco@wins.uva.nl`). LIN has both a paper and an electronic incarnation (see `www.wins.uva.nl/research/ozsl/lin` for the latter); it gives general background on school issues, plus information about new initiatives.

5.3 School voor Informatie- en KennisSystemen SIKS, door: K. Versmissen

Voor SIKS, de *School voor Informatie- en KennisSystemen*, stond 1997 in het teken van de voorbereiding op het indienen van een aanvraag voor erkenning door de KNAW.

Een belangrijke eerste stap in dit proces werd eind januari gezet, toen de gezamenlijke onderzoeksleiders van de school twee dagen lang bijeen kwamen op zich te bezinnen op recente ontwikkelingen. Als gevolg hiervan is de kijk van SIKS op het vakgebied grondig bijgesteld en geactualiseerd.

Hoewel de onderzoekers in SIKS vaak heel verschillende achtergronden hebben, houden zij zich grotendeels met dezelfde vragen en problemen bezig:

- Hoe dient de werkelijkheid gemodelleerd te worden opdat de resulterende beschrijving door een IKS[1] gerealiseerd kan worden?
- Wat is er nodig om een bepaald toepassingsgebied adequaat door een IKS te ondersteunen?
- Hoe kan informatie effectief worden uitgewisseld in omgevingen met zowel IKS als personen? Meer in het bijzonder, hoe kan interactie tussen personen en IKS worden gerealiseerd en verbeterd?

De op de bovenstaande vragen gebaseerde beschrijving van het onderzoek in SIKS kenmerkt zich door haar multidimensionaliteit. Er zijn namelijk drie min of meer onafhankelijke manieren waarop het onderzoek kan worden geclassificeerd:

Onderzoeksthema's

- modelleer- en redeneerconcepten, metatheorie
- modelleren van domeinen en IKS, representatie en redeneren

[1] De afkorting IKS staat voor 'Informatie- en Kennissysteem' dan wel het meervoud van deze term.

- realisatie en performance van representatie- en redeneersystemen

Systeemkarakteristieken

- multimedia en hypermedia
- coöperatieve en gedistribueerde systemen, cyberspace
- informatiemanagementsystemen

Toepassingsgebieden

- technisch en wetenschappelijk
- organisatorisch en administratief
- professioneel (b.v. medisch, juridisch)
- overig (b.v. ontspanning)

In aanvulling op deze algemene beschrijving van het onderzoeksgebied heeft SIKS ook een vijftal onderzoeksfocussen voor de komende jaren vastgesteld:

- kenniswetenschap
- coöperatieve systemen
- requirements engineering en formele specificatie van IKS
- multimedia
- architectuur van IKS

Een ander belangrijk onderdeel van de voorbereiding op het aanvragen van KNAW-erkenning was de verdere uitbouw van het onderwijsaanbod van SIKS. Dit resulteerde in het in werking treden, op 1 september 1997, van een aantal maatregelen die te maken hebben met de opleiding en begeleiding van promovendi. De ruggegraat van het onderwijs in SIKS wordt nu gevormd door een tweejaarlijks roulerend pakket van acht basiscursussen van een halve week ieder, die alle promovendi geacht worden te volgen. Hierop voortbouwend kunnen gespecialiseerde cursussen worden gevolgd, waarvan SIKS er zelf een aantal aanbiedt. Het onderwijsprogramma van de SIKS-promovendus wordt gecomplementeerd met een aantal diverse, zelf te kiezen activiteiten, zoals deelname aan congressen of SIKS-themadagen.

Naast een dozijn promoties vonden er in 1997 de volgende SIKS-activiteiten plaats:

21–23 april	Promovendicursus 'Redeneervormen voor AI', Zeist (georganiseerd door SIKS in samenwerking met de Onderzoekschool Logica)
12 mei	Themadag 'Coöperatieve Systemen', Amsterdam
2 oktober	Doctoraalcolloquium, Utrecht
12–13 november	NAIC'97, Antwerpen (georganiseerd door de NVKI, in samenwerking met SIKS)
26 november	Themadag 'Enterprise Modeling', Delft
1–3 december	Basiscursus 'Kennismodelleren' Apeldoorn
4–5 december	Specialisatiecursus 'Modelleren', Apeldoorn

Voor 1998 zijn de volgende activiteiten gepland:

??	Databasedag
april/mei	Themadag 'Natuurlijke taal en IKS'
mei	Doctoraalcolloquium
25–28 mei	Basiscursus 'Databases' en 'Multimedia'
juni	Master class
september	Basiscursus 'Interactieve Systemen'
september	Specialisatiecursus 'World Wide Web'
October	Themadag
November	Doctoraalcolloquium
November	NAIC'98 (georganiseerd door de NVKI, in samenwerking met SIKS)
December	Basiscursussen 'Combinatorieke Methoden' en 'Electronic Commerce'

Daarnaast zal naar verwachting in juni bekend worden gemaakt of de verwachte KNAW-erkenning er inderdaad komt. Blijkt dat inderdaad het geval, dan zal dit vanzelfsprekend op gepaste wijze gevierd worden.

Voor actuele informatie over SIKS kunt u terecht op de WWW-pagina's van de school, te vinden onder de URL http://www.siks.nl/. Verder kunt u zich met vragen en opmerkingen richten aan: Koen Versmissen, coördinator SIKS, Postbus 80.089, 3508 TB UTRECHT. Tel: 030-2534083. E-mail: koen@siks.nl.

6 Wetenschappelijke bijdragen

6.1 What Do You Mean, Coordination? by Farhad Arbab

CWI, Amsterdam

Abstract

Coordination models and languages represent a new approach to design and development of concurrent systems. The interest in coordination has intensified in the last few years, as evidenced by the increasing number of conferences, tracks, and papers devoted to this topic, and by the recent upsurge of research activity in the theoretical computer science community in this field. The field is relatively new, and while many coordination models and languages form a tight cluster of very similar variants, some others are drastically different and they appear to have nothing in common with each other. All this makes it difficult for the uninitiated to discern the underlying similarities of various approaches to coordination. This paper is an "easy reader" introduction to coordination models and languages, their common aims and purpose, their relevance, and their place in the computing arena. The work on coordination at CWI is presented here as a specific example.

1. Introduction

The size, speed, capacity, and price of computers have all dramatically changed in the last half century. Still more dramatic are the subtle changes of the society's perception of what computers are and what they can, should, and are expected to do. Clearly, this change of perception would not have been possible without the technological advances that reduced the size and price of computers, while increasing their speed and capacity. Nevertheless, the social impact of this change of perception and its feedback influence on the advancement of computer technology itself, are too significant to be regarded as mere by-products of those technological advances.

The term "computer" today has a very different meaning than it did in the early part of this century. Even after such novelties as mechanical and electrical calculators had become commonplace, book-keeping, in its various forms, was a time consuming and labor intensive endeavor for businesses and government agencies alike. Analogous to "typist pools" that lingered on until much later, enterprises such as accountant firms and insurance companies employed armies of people to process, record, and extract the large volumes of essentially numerical data that were relevant for their business. Since "computer" was the term that designated these people, the machine that could clearly magnify their effectiveness and held the promise of replacing them altogether, became known as the "electronic computer."

In spite of the fact that from the beginning, symbol manipulation was as much an inherent ability of electronic computers as arithmetic and juggling of numbers, the perception that computers are really tools for performing fast numerical computations was prevalent. Problems that did not involve a respectable amount of number crunching were either rejected outright as non-problems, or were considered as problems not worthy of attempts to apply computers and computing to. Subscribers to such views were not all naive outsiders, many an insider considered such areas as business and management, databases, and graphics, to be not only on the fringes of computer applications, but also on the fringes of legitimacy. As late as 1970, J. E. Thornton, vice president of Advanced Design Laboratory of Control Data Corporation, who was personally responsible for most of the detailed design of the landmark CDC 6600 computer system, wrote[38]:

> There is, of course, a class of problems which is essentially *noncomputational* but which requires a massive and sophisticated storage system. Such uses as inventory control, production control, and the general category of information retrieval would qualify. *Frankly, these do not need a computer.* There are, however, legitimate justifications for a large computer system as a "partner" with the computational usage. [*emphasis* added.]

Of course, by that time many people were not only convinced that legitimate ***computational*** applications need not involve heavy number crunching, but were already actively working to bring about the changes that turned fringe activities such as databases and graphics into the core of computing, both as "science" as well as in the domain of applications. Nevertheless, Thornton's statement at the time represented the views of a non-negligible minority that has only gradually diminished since. While the numerical applications of computing have steadily grown in number, size, and significance, its non-numerical applications have simply grown even faster and vaster.

2. Computing

The formal notions of computing and computability were introduced by Church, in terms of λ-calculus, and Turing, in terms of Turing Machines. Both Church and Turing were inspired by Hilbert's challenge to define a solid foundation for (mechanical) methods of finding mathematical truth. Hilbert's program consisted of finding a set of axioms as the unassailable foundation of mathematics, such that only mathematical truths could be derived from them by the application of any (truth preserving) mechanical operation, and that all mathematical truths could be derived that way. But, what exactly is a ***mechanical operation***? This was what Church, Turing, and others were to define. Turing himself also intended for his abstract machine to formalize the workings of the human mind. Ironically, his own reasoning on the famous halting problem can be used to show that Turing Machines cannot find all mathematical truths, let alone model the workings of the human mind. Gödel's incompleteness theorem, which brought the premature end of Hilbert's program for mathematics, clearly shows the limits of formal systems and mechanical truth derivation methods. Interestingly, Gödel's incompleteness theorem and Turing's halting problem turned out to be equivalent in their essence: they both show that there are (even mathematical) truths that cannot be derived mechanically, and in both cases, the crucial step in the proof is a variation of the diagonalization first used by Cantor to show that the infinity of real numbers between any two numbers is greater than the infinity of natural numbers.

It is far from obvious why Turing's simple abstract machine, or Church's λ-calculus, is a reasonable formalization of what we intuitively mean by ***any mechanical operation***. However, all extensions of the Turing Machine that have been considered, are shown to be mathematically equivalent to, and no more powerful than, the basic Turing Machine. Turing and Church showed the equivalence of Turing Machines and λ-calculus. This, plus the fact that other formalizations (e.g., Post's) have all turned out to be equivalent, has increased the credibility of the conjecture that a Turing Machine can actually be made to perform any mechanical operation whatsoever. Indeed, it has become reasonable to mathematically *define* a mechanical operation as any operation that can be performed by a Turing Machine, and to accept the view known as Church's thesis: that the notion of Turing Machines (or λ-calculus, or other equivalents) mathematically defines the concept of an algorithm (or an effective, or recursive, or mechanical procedure).

3. Interaction

Church's thesis can simply be considered as a mathematical definition of what computing is in a strictly technical sense. Real computers, on the other hand, do much more than mere computing in this restrictive sense. Among other things, they are sources of heat and noise, and have always been revered (and despised) as (dis)tasteful architectural artifacts, pieces of furniture, or decoration mantle-pieces. More interestingly, computers also *interact*: they can act as facilitators, mediators, and coordinators that enable the collaboration of other agents. These other agents may in turn be other computers (or computer programs), sensors and actuators that involve their real world environment, or human beings. The role of a computer as an agent that performs computing, in the strict technical sense of the word, should not be confused with its role as a mediator agent that, e.g., empowers its human users to collaborate with one another. The fact that the computer, in this case, may perform some computation in order to enable the collaboration of other agents, is ancillary to the fact that it needs to interact with these agents to enable their collaboration. To emphasize this distinction, Wegner proposes the concept of an Interaction Machine[39, 40].

A Turing Machine operates as a closed system: it receives its input tape, starts computing, and (hopefully) halts, at which point its output tape contains the result of its computation. In every step of a computation, the symbol written by a Turing Machine on its tape depends only on its internal state and the current symbol it reads from the tape. An Interaction Machine is an extension of a Turing Machine that can interact with its environment with new input and output primitive actions. Unlike other extensions of the Turing Machine (such as more tapes, more controls, etc.) this one actually changes the essence of the behavior of the machine. This extension makes Interaction Machines *open systems.*

Consider an Interaction Machine I operating in an environment described as a dynamical system E. The symbol that I writes on its tape at a given step, not only depends on its internal state and the current symbol it reads from the tape, but can also depend the input it obtains directly from E. Because the behavior of E cannot be described by a computable function, I cannot be replaced by a Turing Machine. The best approximation of I by a Turing Machine, T, would require an encoding of the actual input that I obtains from E, which can be known only *after* the input operation. The computation that T performs, in this case, is the same as that of I, but I does more than T because it interacts with its environment E. What T does, in a sense, is analogous to predicting yesterday's weather: it is interesting that it can be done (assuming that it can be done), but it doesn't quite pass muster! To emphasize the distinction, we can imagine that the interaction of I with E is not limited to just this one direct input: suppose I also does a direct output to E, followed by another direct input from E. Now, because the value of the second input from E to I depends on the earlier interaction of E and I, no input tape can encode this "computation" for any Turing Machine.

It is the ability of computers (as Interaction Machines) to interact with the real world, rather than their ability (as mere Turing Machines) to carry on ever-more-sophisticated computations, that is having the most dramatic impact on our societies. In the traditional models of human-computer interaction, users prepare and consume the information needed and produced by their applications, or select from the alternatives allowed by a rigid structure of computation. In contrast to these models, the emerging models of human-computer interaction remove the barriers between users and their applications. The role of a user is no longer limited to that of an observer or an operator: increasingly, users become active components of their running applications, where they examine, alter, and steer on-going computations. This form of cooperation between humans and computers, and between humans via computers, is a vital necessity in many contemporary applications, where realistic results can be achieved only if human intuition and common-sense is combined with raw, formal reasoning and computation. The applications of computer facilitated collaborative work are among the increasingly important areas of activity in the foreseeable future. They can be regarded as natural extensions of systems where several users simultaneously examine, alter, interact, and steer on-going computations. Interaction Machines are suitable conceptual models for describing such applications.

Interaction Machines have unpredictable input from their external environment, and can directly affect their environment, unpredictably, due to such input. Because of this property, Interaction Machines may seem too open for formal studies: the unpredictable way that the environment can affect their behavior can make their behavior underspecified, or even ill-defined. But, this view is misleading. Interaction Machines are both useful and interesting for formal studies.

On the one hand, the open-ness of Interaction Machines and their consequent underspecified behavior is a valuable true-to-life property. Real systems are composed of components that interact with one another, where each is an open system in isolation. Typically, the behavior of each of these components is ill-defined, except within the confines of a set of constraints on its interactions with its environment. When a number of such open systems come together as components to comprise a larger system, the topology of their interactions forms a context that constrains their mutual interactions and yields well-defined behavior.

On the other hand, the concept of Interaction Machines suggests a clear dichotomy for the formal study of their behavior, both as components in a larger system, as well as in isolation. Just like a Turing Machine, the behavior of an Interaction Machine can be studied as a computation (in the sense of Church's thesis) between each pair of its successive interactions. More interestingly, one can abstract away from all such computations, regarding them as internal details of individual components, and embark on a formal study of the constraints, contexts, and conditions on the interactions among the components in a system (as well as between the system and its environment) that ensure and preserve well-behavedness. And this material is the thread that weaves the fabric of coordination.

4. Concurrency

Interaction and concurrency are closely related concepts. Concurrency means that computations in a system overlap in time. The computations in a concurrent system may actually run in parallel (i.e., use more than one physical processor at a time) or be interleaved with one another on a single processor. The parallel computations in a system may or may not be geographically distributed. What distinguishes an interactive system from other concurrent systems is the fact that an interactive system has unpredictable inputs from an external environment that it does not control.

The study and the application of concurrency in computer science has a long history. The study of deadlocks, the dining philosophers, and the definition of semaphores and monitors were all well established by the early seventies. Theoretical work on concurrency, e.g., CSP[22, 23], CCS[28], process algebra[12], and π-calculus[29, 30], has helped to show the difficulty of dealing with concurrency, especially when the number of concurrent activities becomes large. Most of these models are more effective for describing closed systems. A number of programming languages have used some of these theoretical models as their bases, e.g., Occam[24] uses CSP and LOTOS[13] uses CCS. However, it is illuminating to note that the original context for the interest in concurrency was somewhat different than the demands of the applications of today in two respects:

- In the early days of computing, hardware resources were prohibitively expensive and had to be shared among several programs that had nothing to do with each other, except for the fact that they were unlucky enough to have to compete with each other for a share of the same resources. This was the *concurrency of competition*. Today, it is quite feasible to allocate tens, hundreds, and thousands of processors to the same task (if only we could do it right). This is the *concurrency of cooperation*. The distinction is that whereas it is sufficient to keep independent competing entities from trampling on each other over shared resources, cooperating entities also depend on the (partial) results they produce for each other. Proper passing and sharing of these results require more complex protocols, which become even more complex as the number of cooperating entities and the degree of their cooperation increase.

- The falling costs of processor and communication hardware only recently dropped below the threshold where having very large numbers of "active entities" in an application makes sense. Massively parallel systems with thousands of processors are a reality today. Current trends in processor hardware and operating system kernel support for threads[2] make it possible to efficiently have in the order of hundreds of active entities running in a process on each processor. Thus, it is not unrealistic to think that a single application can be composed of hundreds of thousands of active entities. Compared to classical uses of concurrency, this is a jump of several orders of magnitude in numbers, and in our view, represents (the need for) a qualitative change.

The practical use of massively parallel systems is often limited to applications that fall into one of a few simple concurrency structures. In fact, although MIMD[3] systems have been available for some time, they are hardly ever used as MIMD systems in an application. The basic problem in using the MIMD paradigm in large applications is coordination: how to ensure proper communication among the hundreds and thousands of different pieces of code that comprise the active

[2]Threads are preemptively-scheduled light-weight processes that run within one operating-system level process and share the same address space.

[3]Multiple Instruction, Multiple Data

entities in a single application. A restriction of the MIMD model, called SPMD[4], introduces a *barrier* mechanism as the only coordination construct. This model simplifies the problem of concurrency control by allowing several processors, all executing the same program, but on different data, proceed at their own pace up to a common barrier, where they then synchronize.

There are applications that do not fit in the uniformity offered by the SPMD model and require more flexible coordination. Examples include computations involving large dynamic trees, symbolic computation on parallel machines, and dynamic pipelines. Taking full advantage of the potential offered by massively parallel systems in these and other applications requires massive, non-replicated, concurrency.

The primary concern in the design of a concurrent application must be its model of cooperation: how the various active entities comprising the application are to cooperate with each other. Eventually, a set of communication primitives must be used to realize whatever model of cooperation application designers opt for, and the concerns for performance may indirectly affect their design. Nevertheless, it is important to realize that the conceptual gap between the system supported communication primitives and a concurrent application must often be filled with a non-trivial model of cooperation.

The models of cooperation used in concurrent applications of today are essentially a set of ad hoc templates that have been found to be useful in practice. There is no paradigm wherein we can systematically talk about cooperation of active entities, and wherein we can compose cooperation scenarios such as (and as alternatives to) models like client-server, workers pool, etc., out of a set of primitives and structuring constructs. Consequently, programmers must directly deal with the lower-level communication primitives that comprise the realization of the cooperation model of a concurrent application.

5. Coordination

Coordination languages, models, and systems constitute a new field of study in programming and software systems, with the goal of finding solutions to the problem of managing the interaction among concurrent programs. Coordination can be defined as the study of the dynamic topologies of interactions among Interaction Machines, and the construction of protocols to realize such topologies that ensure well-behavedness. Analogous to the way in which topology abstracts away the metric details of geometry and focuses on the invariant properties of (seemingly very different) shapes, coordination abstracts away the details of computation in Interaction Machines, and focuses on the invariant properties of (seemingly very different) programs. As such, coordination focuses on *program patterns* that specifically deal with interaction.

Coordination is relevant in design, development, debugging, maintenance, and reuse of all concurrent systems. Coordination models and languages are meant to close the conceptual gap between the cooperation model of an application and the lower-level communication model used in its implementation. The inability to deal with the cooperation model of a concurrent application in an explicit form contributes to the difficulty of developing working concurrent applications that contain large numbers of active entities with non-trivial cooperation protocols. In spite of the fact that the implementation of a complex protocol is often the most difficult and error prone part of an application development effort, the end result is typically not recognized as a "commodity" in its own right, because the protocol is only implicit in the behavior of the rest of the concurrent software. This makes maintenance and modification of the cooperation protocols of concurrent applications much more difficult than necessary, and their reuse next to impossible.

A number of software platforms and libraries are presently popular for easing the development of concurrent applications. Such systems, e.g., PVM, MPI, CORBA, etc., are sometimes called *middleware*. Coordination languages can be thought of as the linguistic counterpart of these platforms which offer middleware support for software composition. One of the best known coordination languages is Linda[16, 27], which is based on the notion of a shared tuple space. The tuple space of Linda is a centrally managed space which contains all pieces of information that

[4]Single Program, Multiple Data

processes want to communicate. Linda processes can be written in any language augmented with Linda primitives. There are only four primitives provided by Linda, each of which associatively operates on (e.g., reads or writes) a single tuple in the tuple space.

Besides the "generative tuple space" of Linda, a number of other interesting models have been proposed and used to support coordination languages and systems. Examples include various forms of "parallel multiset rewriting" or "chemical reactions" as in Gamma [10], models with explicit support for coordinators as in MANIFOLD[7], and "software bus" as in ToolBus[11]. A significant number of these models and languages are based on a few common notions, such as pattern-based, associative communication [1], to complement the name-oriented, data-based communication of traditional languages for parallel programming.

Coordination languages have been applied to the parallelization of computation intensive sequential programs in the fields of simulation of Fluid Dynamics systems, matching of DNA strings, molecular synthesis, parallel and distributed simulation, monitoring of medical data, computer graphics, analysis of financial data integrated into decision support systems, and game playing (chess). See [2, 17, 21] for some concrete examples.

6. Classification

Coordination models and languages can be classified as either *data-oriented* or *control-oriented.* For instance, Linda uses a data-oriented coordination model, whereas MANIFOLD is a control-oriented coordination language. The activity in a data-oriented application tends to center around a substantial shared body of data; the application is essentially concerned with what happens to *the data.* Examples include database and transaction systems such as banking and airline reservation applications. On the other hand, the activity in a control-oriented application tends to center around processing or flow of control and, often, the very notion of *the data*, as such, simply does not exist; such an application is essentially described as a collection of activities that genuinely consume their input data, and subsequently produce, remember, and transform "new data" that they generate by themselves. Examples include applications that involve work-flow in organizations, and multi-phase applications where the content, format, and/or modality of information substantially changes from one phase to the next.

Coordination models and languages can also be classified as either *endogenous* or *exogenous.* For instance, Linda is based on an endogenous model, whereas MANIFOLD is an exogenous coordination language. Endogenous models and languages provide primitives that must be incorporated *within* a computation for its coordination. In applications that use such models, primitives that affect the coordination of each module are inside the module itself. In contrast, exogenous models and languages provide primitives that support coordination of entities from *without.* In applications that use exogenous models primitives that affect the coordination of each module are outside the module itself.

Endogenous models are sometimes more natural for a given application. However, they generally lead to intermixing of coordination primitives with computation code, which entangles the semantics of computation with coordination protocols. This intermixing tends to scatter communication/coordination primitives throughout the source code, making the cooperation model and the coordination protocol of an application nebulous and implicit: generally, there is no piece of source code identifiable as the cooperation model or the coordination protocol of an application, that can be designed, developed, debugged, maintained, and reused, in isolation from the rest of the application code. On the other hand, exogenous models encourage development of coordination modules separately and independently of the computation modules they are supposed to coordinate. Consequently, the result of the substantial effort invested in the design and development of the coordination component of an application can manifest itself as tangible "pure coordinator modules" which are easier to understand, and can also be reused in other applications.

7. Coordination at CWI

Experimental research on coordination has been going on at CWI since 1990. This work has produced IWIM[4, 3], a novel model for control-oriented coordination; MANIFOLD[5, 3, 9, 7], a pure coordination language based on the IWIM model; preliminary studies on the formal semantics of MANIFOLD[34, 33]; and Visifold[15], a visual programming environment for MANIFOLD. The implementation of the second version of MANIFOLD is now complete and it is being used in a number of applications. Theoretical work on the formal semantics of MANIFOLD and the mathematical models underlying IWIM and MANIFOLD are presently on-going.

Theoretical and experimental work on coordination were unified in 1997 under the *Theme SEN3: Coordination Languages* within the *Software Engineering Cluster* at CWI. This Theme aims at cross-fertilization of formal and applied research on coordination. The activity in SEN3 is on (1) development of formal methods, notably (operational) semantic models as a unifying basis for the development of debugging and visualization tools for coordination languages; (2) enhancements to and experiments with the MANIFOLD language and its visual programming and debugging environment; and (3) using MANIFOLD to work on real applications of coordination programming in areas such as numerical computing, distributed constraint satisfaction, and shallow water modeling.

8. Manifold

MANIFOLD is a coordination language for managing complex, dynamically changing interconnections among sets of independent, concurrent, cooperating processes[5]. The processes that comprise an application are either computation or coordinator processes. Computation processes can be written in any conventional programming language. Coordinator processes are clearly distinguished from the others in that they are written in the MANIFOLD language. The purpose of a coordinator process is to establish and manage the communications among other (computation or coordinator) processes. MANIFOLD is a control-oriented, exogenous coordination language based on the IWIM model.

IWIM stands for *Idealized Worker Idealized Manager* and is a generic, abstract model of communication that supports the separation of responsibilities and encourages a weak dependence of workers (processes) on their environment. Two major concepts in IWIM are separation of concerns and anonymous communication. Separation of concerns means that computation concerns are isolated from the communication and cooperation concerns into, respectively, worker and manager (or coordinator) modules. Anonymous communication means that the parties (i.e., modules or processes) engaged in communication with each other need not know each other. IWIM-sanctioned communication is either through broadcast of events, or through point-to-point channel connections that, generally, are established between two communicating processes by a third party coordinator process.

MANIFOLD is a strongly-typed, block-structured, event driven language, meant for writing coordinator program modules. As modules written in MANIFOLD represent the idealized managers of the IWIM model, strictly speaking, there is no need for the constructs and the entities that are common in conventional programming languages; thus, semantically, there is no need for integers, floats, strings, arithmetic expressions, sequential composition, conditional statements, loops, etc.[5] The only entities that MANIFOLD recognizes are processes, ports, events, and streams (which are asynchronous channels), and the only control structure that exists in MANIFOLD is an event-driven state transition mechanism. Programming in MANIFOLD is a game of dynamically creating (coordinator and/or worker) process instances and dynamically (re)connecting the ports of some of these processes via streams, in reaction to observed event occurrences. The fact that computation and coordinator processes are absolutely indistinguishable from the point of view of other processes,

[5]For convenience, however, some of these constructs, syntactically, do exist in the MANIFOLD language. Currently, only the front-end of the MANIFOLD language compiler knows about such "syntactic sugar" and translates them into processes, state transitions, etc., so that as far as the run-time system (or even the code generator of the MANIFOLD compiler) is concerned, these familiar constructs "do not exist" in MANIFOLD.

means that coordinator processes can, recursively, manage the communication of other coordinator processes, just as if they were computation processes. This means that any coordinator can also be used as a higher-level or meta-coordinator, to build a sophisticated hierarchy of coordination protocols. Such higher-level coordinators are not possible in most other coordination languages and models.

MANIFOLD encourages a discipline for the design of concurrent software that results in two separate sets of modules: pure coordination, and pure computation. This separation disentangles the semantics of computation modules from the semantics of the coordination protocols. The coordination modules construct and maintain a dynamic data-flow graph where each node is a process. These modules do no computation, but only make the prescribed changes to the connections among various processes in the application, which changes only the topology of the graph. The computation modules, on the other hand, cannot possibly change the topology of this graph, making both sets of modules easier to verify and more reusable. The concept of reusable pure coordination modules in MANIFOLD is demonstrated, e.g., by using (the object code of) the same MANIFOLD coordinator program that was developed for a parallel/distributed bucket sort algorithm, to perform function evaluation and numerical optimization using domain decomposition[6, 18].

The MANIFOLD system runs on multiple platforms and consists of a compiler, a run-time system library, a number of utility programs, and libraries of builtin and predefined processes of general interest. Presently, it runs on IBM RS6000 AIX, IBM SP1/2, Solaris, Linux, and SGI IRIX. A MANIFOLD application consists of a (potentially very large) number of processes running on a network of heterogeneous hosts, some of which may be parallel systems. Processes in the same application may be written in different programming languages and some of them may not know anything about MANIFOLD, nor the fact that they are cooperating with other processes through MANIFOLD in a concurrent application. A number of these processes may run as independent operating-system-level processes, and some will run together as light-weight processes (preemptively scheduled threads) inside an operating-system-level process. None of this detail is relevant at the level of the MANIFOLD source code, and the programmer need not know anything about the eventual configuration of his or her application in order to write a MANIFOLD program.

MANIFOLD has been successfully used to implement parallel and distributed versions of a semi-coarsened multi-grid Euler solver algorithm in MAS2 at CWI. This represents a real-life heavy-duty Computational Fluid Dynamics application where MANIFOLD enabled restructuring of existing sequential Fortran code using pure coordination protocols that allow it to run on parallel and distributed platforms. The results of this work are very favorable: no modification to the computational Fortran 77 code, simple, small, reusable MANIFOLD coordination modules, and linear speed-up of total execution time with respect to the number of processors (e.g., from almost 9 to over 2 hours)[19, 20].

Other applications of MANIFOLD include its use in modeling cooperative Information Systems[31, 32], coordination of Loosely-Coupled Genetic Algorithms on parallel and distributed platforms[36, 37], coordination of multiple solvers in a concurrent constraint programming system[8], and a distributed propositional theorem checker in the *Theme SEN2: Specification and Analysis of Embedded Systems* at CWI.

9. Examples

An interesting example of an exogenous coordination protocol is a MANIFOLD program described in [6] that receives two process type definitions (A, and M) and recursively creates instances of itself (processes x_i), A (processes a_i), and M (processes m_i), and (re)connects the streams routing the flow of information among them. This abstract coordination protocol can be compiled separately, and linked with the object code of other processes to build an application. It is immaterial what exactly processes A and M do, and process instances x_i, a_i, and m_i, that are created at run time, can run on various hosts on a heterogeneous platform. The depth of the recursion (i.e., the exact number of x_i's) depends on the number of input units, and the number of units each instance of a_i

decides for itself to consume. What matters is the pattern of communication among, and creation of, these process instances.

Entirely different applications can use (the object code of) this same **MANIFOLD** program as their coordination protocol[6]. For example, we have used this protocol to perform parallel/distributed sorting, by supplying:

- as A, a sorter, each instance of which takes in $n > 0$ input units, sorts them, and produces the result as its output; and
- as M, a merger, each instance of which produces as its output the merged sequence of the two sequences of units it receives as its input.

We have also used this protocol to perform parallel/distributed numerical optimization, e.g., of a complex function by supplying:

- as A, an evaluator, each instance of which takes in an input unit describing a (sub)domain of a function, and produces its best estimate of the optimum value of the function in that (sub)domain; and
- as M, a selector, each instance of which produces as its output the best optimum value it receives as its input.

10. Formal Models and Semantics

The formal (operational) semantics of **MANIFOLD** is defined in terms of a two-level transition system[14]. The first level consists of a (large) number of transition systems, each of which defines the semantics of a single process, independently of the rest. The second level consists of a single transition system that defines the interactions among the first-level transition systems. The details of the internal activity of the first-level transition systems (e.g., their computations) are irrelevant for, and therefore unobservable by, the second-level transition system. This two-level approach to formal semantics reflects the dichotomy of computation vs. coordination that is inherent in **MANIFOLD**, and represents a novel application of transition system in formal semantics. More generally, this approach is useful for the definition of the formal semantics of other coordination languages as well. The key concept here is that the second level abstracts away the (computational) semantics of the first level processes, and is concerned only with their (mutually engaging) externally observable behavior.

Related to transition systems are the notions of *bisimulation* and *bisimilarity* which reflect the intuitive concept of the equivalence (or similarity) of the externally observable behavior of concurrent systems. Bisimulation is used to define the formal notion of *coinduction* as the counterpart for the familiar principle of induction. Coinduction, bisimulation, and *coalgebras*[25, 35] comprise a mathematical machinery analogous to the more familiar notions of induction, congruence, and algebras, that is suitable for the study of concurrency and coordination.

Traditionally, *initial algebras* have been used as mathematical models for finite data types, such as finite lists. Their counterparts, *final coalgebras*, are used as mathematical models for infinite data types and for the semantics of object-oriented programming languages. More generally, coalgebras can serve as models for dynamical and transition systems. Coalgebraic models seem very appealing candidates for a mathematically sound foundation for the semantics of **MANIFOLD**. **MANIFOLD**'s strict separation of computation from communication, plus the fact that it is based on an exogenous model of coordination, leads to a clear dichotomy of internal vs. externally observable behavior of each process. This, in turn, corresponds directly with the inherent "strict information hiding" property of coalgebras. On the other hand, coalgebraic models for the semantics of **MANIFOLD** raise interesting challenges in the field of coalgebras: to reflect the compositionality of **MANIFOLD**, a suitable theory of composition of coalgebras is necessary.

An alternative approach to a mathematical foundation for the semantics of **MANIFOLD** can be sought in other category-theoretical models. The essence of the semantics of a coordinator process in MANIFOLD can be described as transitions between states, each of which defines a different topology of information-carrying streams among various sets of processes. What is defined in each such state is reminiscent of an (asynchronous) electronic circuit. Category theoretical models have been used to describe simple circuit diagrams[26]. Extensions of such models to account for the dynamic topological reconfiguration of MANIFOLD is a non-trivial challenge which, nevertheless, points to an interesting model of computation.

References

[1] ANDREOLI, J.-M., CIANCARINI, P., AND PARESCHI, R. Interaction Abstract Machines. In *Trends in Object-Based Concurrent Computing*. MIT Press, 1993, pp. 257–280.

[2] ANDREOLI, J.-M., HANKIN, C., AND LE MÉTAYER, D., Eds. *Coordination Programming: Mechanisms, Models and Semantics*. Imperial College Press, 1996.

[3] ARBAB, F. Coordination of massively concurrent activities. Tech. Rep. CS–R9565, Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands, November 1995. Available on-line http://www.cwi.nl/ftp/CWIreports/IS/CS-R9565.ps.Z.

[4] ARBAB, F. The IWIM model for coordination of concurrent activities. In *Coordination Languages and Models* (April 1996), P. Ciancarini and C. Hankin, Eds., vol. 1061 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 34–56.

[5] ARBAB, F. Manifold version 2: Language reference manual. Tech. rep., Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands, 1996. Available on-line http://www.cwi.nl/ftp/manifold/refman.ps.Z.

[6] ARBAB, F., BLOM, C. L., BURGER, F. J., AND EVERAARS, C. T. H. Reusable coordinator modules for massively concurrent applications. In *Proceedings of Euro-Par '96* (August 1996), L. Bouge, P. Fraigniaud, A. Mignotte, and Y. Robert, Eds., vol. 1123 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 664–677.

[7] ARBAB, F., HERMAN, I., AND SPILLING, P. An overview of Manifold and its implementation. *Concurrency: Practice and Experience 5*, 1 (February 1993), 23–70.

[8] ARBAB, F., AND MONFROY, E. Using coordination for cooperative constraint solving. In *Proceedings of the 1998 ACM Symposium on Applied Computing; Special Track on Coordination Models, Languages and Applications* (Atlanta, Georgia, February-March 1998), ACM.

[9] ARBAB, F., AND RUTTEN, E. P. B. M. Manifold: a programming model for massive parallelism. In *Proceedings of the IEEE Working Conference on Massively Parallel Programming Models* (Berlin, September 1993).

[10] BANÂTRE, J.-P., AND LE MÉTAYER, D. Programming by multiset transformations. *Communications of the ACM 36*, 1 (January 1993), 98–111.

[11] BERGSTRA, J., AND KLINT, P. The ToolBus Coordination Architecture. In *Proc. 1st Int. Conf. on Coordination Models and Languages* (Cesena, Italy, April 1996), P. Ciancarini and C. Hankin, Eds., vol. 1061 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 75–88.

[12] BERGSTRA, J. A., AND KLOP, J. W. Process algebra for synchronous communication. *Information and Control 60* (1984), 109–137.

[13] BOLOGNESI, T., AND BRINKSMA, E. Introduction to the ISO specification language LOTOS. *Computer Networds and ISDN Systems 14* (1986), 25–59.

[14] BONSANGUE, M. M., ARBAB, F., DE BAKKER, J. W., RUTTEN, J. J. M. M., AND SCUTELLÁ, A. A transition system semantics for a control-driven coordination language. Tech. Rep. to appear, Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands, 1998.

[15] Bouvry, P., and Arbab, F. Visifold: A visual environment for a coordination language. In *Coordination Languages and Models* (April 1996), P. Ciancarini and C. Hankin, Eds., vol. 1061 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 403–406.

[16] Carriero, N., and Gelernter, D. LINDA in context. *Communications of the ACM 32* (1989), 444–458.

[17] Ciancarini, P., and Hankin, C., Eds. *1st Int. Conf. on Coordination Languages and Models*, vol. 1061 of *Lecture Notes in Computer Science*. Springer-Verlag, April 1996.

[18] Everaars, C. T. H., and Arbab, F. Coordination of distributed/parallel multiple-grid domain decomposition. In *Proceedings of Irregular '96* (August 1996), A. Ferreira, J. Rolim, Y. Saad, and T. Yang, Eds., vol. 1117 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 131–144.

[19] Everaars, C. T. H., Arbab, F., and Burger, F. J. Restructuring sequential Fortran code into a parallel/distributed application. In *Proceedings of the International Conference on Software Maintenance '96* (November 1996), IEEE, pp. 13–22.

[20] Everaars, C. T. H., and Koren, B. Using coordination to parallelize sparse-grid methods for 3D CFD problems. *Parallel Computing* (1998). to appear in the special issue on Coordination.

[21] Garlan, D., and Le Métayer, D., Eds. *2nd Int. Conf. on Coordination Languages and Models*, vol. 11282 of *Lecture Notes in Computer Science*. Springer-Verlag, September 1997.

[22] Hoare, C. Communicating Sequential Processes. *Communications of the ACM 21* (August 1978).

[23] Hoare, C. *Communicating Sequential Processes*. Prentice Hall International Series in Computer Science. Prentice-Hall, 1985.

[24] INMOS Ltd. *OCCAM 2, Reference Manual*. Series in Computer Science. Prentice-Hall, 1988.

[25] Jacobs, B., and Rutten, J. A tutorial on (co)algebras and (co)induction. *Bulletin of EATCS 62* (1997), 222–259.
Available on-line http://www.cs.kun.nl/ bart/PAPERS/JR.ps.Z.

[26] Katis, P., Sabadini, N., and Walters, R. The bicategory of circuits. Tech. Rep. 94-22, University of Sydney, 1994. To appear in: Journal of Pure and Applied Algebra.

[27] Leler, W. LINDA meets UNIX. *IEEE Computer 23* (February 1990), 43–54.

[28] Milner, R. *Communication and Concurrency*. Prentice Hall International Series in Computer Science. Prentice Hall, 1989.

[29] Milner, R. The polyadic π-calculus: A tutorial. Tech. Rep. Res. Report LFCS-91-180, Lab. for Foundations of Computer Science, Edinburgh University, 1991.

[30] Milner, R. Elements of interaction. *Communications of the ACM 36*, 1 (January 1993), 78–89.

[31] Papadopoulos, G., and Arbab, F. Control-based coordination of human and other activities in cooperative information systems. In *Proceedings of the Second International Conference on Coordination Languages and Models* (September 1997), Lecture Notes in Computer Science, Springer-Verlag, pp. 422–425.

[32] Papadopoulos, G. A., and Arbab, F. Modelling activities in information systems using the coordination language Manifold. In *Proceedings of the 1998 ACM Symposium on Applied Computing; Special Track on Coordination Models, Languages and Applications* (Atlanta, Georgia, February-March 1998), ACM.

[33] Rutten, E. P. B. M. Minifold: a kernel for the coordination language Manifold. Tech. Rep. CS-R9252, Centrum voor Wiskunde en Informatica, Amsterdam, November 1992.

[34] Rutten, E. P. B. M., Arbab, F., and Herman, I. Formal specification of Manifold: a preliminary study. Tech. Rep. CS-R9215, Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands, 1992.

[35] RUTTEN, J. J. M. M. Universal coalgebra: A theory of systems. Tech. Rep. CS-R9652, Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands, 1996. Available on-line http://www.cwi.nl/ftp/CWIreports/AP/CS-R9652.ps.Z.

[36] SEREDYNSKI, F., BOUVRY, P., AND ARBAB, F. Distributed evolutionary optimization in Manifold: the rosenbrock's function case study. In *FEA'97 - First International Workshop on Frontiers in Evolutionary Algorithms (part of the third Joint Conference on Information Sciences)* (Mar. 1997). Duke University (USA).

[37] SEREDYNSKI, F., BOUVRY, P., AND ARBAB, F. Parallel and distributed evolutionary computation with Manifold. In *Proceedings of PaCT-97* (September 1997), V. Malyshkin, Ed., vol. 1277 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 94–108.

[38] THORNTON, J. E. *Design of a Computer: The Control Data 6600.* Scott, Foresman and Company, 1970.

[39] WEGNER, P. Interaction as a basis for empirical computer science. *ACM Computing Surveys 27*, 1 (Mar. 1995), 45–48.

[40] WEGNER, P. Interactive foundations of computing. *Theoretical Computer Science 192*, 2 (20 Feb. 1998), 315–351.

6.2 Using Otter in Coq, by Marc Bezem and Dimitri Hendriks

Utrecht University, Department of Philosophy, Utrecht

1. Introduction

The proof generation capabilities of proof construction systems such as Coq, based on type theory, could still be improved. Resolution based theorem provers such as Otter are more powerful in this respect, but have the drawback that they work with normal forms of formulae, so-called clausal forms. Clauses are prenex-Skolem-conjunctive normal forms which need not be exactly logically equivalent to the original formulae. This makes resolution proofs hard to read and understand, and makes the navigation of the theorem prover through the search space very difficult. Type theory, with its highly expressive language, is much better in this respect, but the proof generation capabilities suffer from the small granularity of the inference steps and the corresponding astronomic size of the search space. Typically, one hyperresolution step requires a few dozens of inference steps in type theory. The idea of this project is to combine the best of both worlds.

The ideal procedure would be as follows. Identify a non-trivial step in a Coq session that amounts to a first order tautology. Export this tautology to Otter, and delegate its proof to the Otter inference engine with all its clever handles such as strategies, weights, the hot-list, and so on. Convert the resolution proof to type theoretic format and import the result back in Coq.

Most of the necessary metatheory is already known. The prenex and conjunctive normal form transformations can be axiomatized by classical logic. Skolemization can be axiomatized by the Axiom of Choice. Higher order logic is particularly suited for this axiomatization: we get logical equivalence modulo classical logic plus the Axiom of Choice, instead of awkward invariants as equiconsistency or equisatisfiability in the first order case. The automation of this part of the project has been carried out and will be described in the sequel. Converting resolution proofs to lambda terms is a parsing and code generation problem of manageable difficulty. This is planned as the next step in the project.

Of course application has to be limited to mathematics that is compatible with classical logic plus the Axiom of Choice. This is the price to be paid for the automated theorem proving procedure that we propose: it may invoke unnecessary applications of classical logic and/or the Axiom of Choice. In particular it is possible that the automated proofs of intuitionistic tautologies are not optimal in the sense that they use classical logic.

2. Example: the Drinker's Principle

As running example we use a well-known classical tautology called the Drinker's Principle: in every group of people is somebody who, if (s)he is drunk, then everybody is drunk.

Otter refutes almost instantaneously:

```
-(exists x (drunk(x) -> (all y (drunk(y)))))
```

after first clausifying this into:

```
0 [] drunk(x).
0 [] -drunk($f1(x)).
```

where $f1 is a Skolem function, by the following refutation, with $F for false:

```
1 [] drunk(x).
2 [] -drunk($f1(x)).
3 [binary,2.1,1.1] $F.
```

The example illustrates well that the clausal form is quite different from the original formulation of the problem. We leave it as an exercise to the reader to make the relation between the original formulation of the problem and the clausal form precise.

In Coq we have the tactic program session below, to be entered by the user. Here x:X expresses the typing relation, to be interpreted as 'x belongs to X' when X:Set ('X is a set'), and as 'x is a proof of X' when X:Prop ('X is a proposition'). Furthermore, (x:X) denotes universal quantification, [x:X] lambda abstraction, and (M N) well typed application. Coq uses -> for implication between propositions as well as for function spaces between sets, ~ for negation and Ex for existential quantification.

```
Lemma Drinker's_Principle:
((p:Prop)~~p->p)->(D:Set)(d:D)(drunk:D->Prop)
 (Ex [x:D](drunk x)->(y:D)(drunk y)).
Proof.
Intros classic D d drunk.
Apply classic. Red. Intro.
Apply H. Exists d.
Intros.
Apply classic. Red. Intro.
Apply H. Exists y.
Intro. Absurd (drunk y).
Assumption. Assumption.
Qed.
```

This tactic program generates the following lambda term:

```
Drinker's_Principle =
[classic:(p:Prop)~~p->p][D:Set][d:D][drunk:D->Prop]
 (classic (Ex [x:D](drunk x)->(y:D)(drunk y))
  [H:~(Ex [x:D](drunk x)->(y:D)(drunk y))]
   (H (ex_intro D [x:D](drunk x)->(y:D)(drunk y) d
    [_:(drunk d)][y:D]
     (classic (drunk y)
      [H1:~(drunk y)]
       (H (ex_intro D [x:D](drunk x)->(y0:D)(drunk y0) y
        [H2:(drunk y)](False_ind (y0:D)(drunk y0) (H1 H2)))))))
     : ((p:Prop)~~p->p)->(D:Set)(d:D)(drunk:D->Prop)
        (Ex [x:D](drunk x)->(y:D)(drunk y))
```

The example illustrates the differences in style. Otter is real automated theorem proving. The Otter proof is a refutation of the clausal form of the negation of the Drinker's Principle. As it stands, the proof is incomplete with respect to the original statement. In fact, due to the Skolem function $f1, the signature of the language has been extended and a different statement has been proved. On the other hand, Coq is interactive, the proof is detailed and fully explicit as lambda term. It clearly shows the use of classical logic, the polymorphism in D and drunk:D->Prop. Moreover, the assumption that the domain is not empty is made explicit by (d:D) in the formulation of the lemma.

3. Clausification

The universe Prop of all propositions includes higher order propositions, in fact full impredicative type theory, and is as such too large for our purposes. Moreover, Coq supplies only limited computational power on Prop. Therefore we found it necessary to define form, an inductive set in which first order propositions can be represented formally. As every inductive set, form is equipped with higher order primitive recursion as powerful computational device.

First of all we need a domain of discourse D and define the inductive set form depending on D.

```
Parameter D: Set.
```

```
Inductive form : Set :=
  f_atom: D->form
| f_not:  form->form
| f_and:  form->form->form
| f_or:   form->form->form
| f_impl: form->form->form
| f_ex:   (D->form)->form
| f_all:  (D->form)->form.
```

Second, we define HM, the canonical mapping[6] from form to Prop, by primitive recursion. The distinctive recursive cases are made explicit as comments (* ... *). In each case the lambda term after the comment is to be substituted for the corresponding constructor. As a consequence, (HM (f_atom d)) reduces to (drunk d), (HM (f_not f0)) to ~(HM f0), and so on. We declare an unary predicate symbol drunk to form atoms.

```
Parameter drunk: D->Prop.

Fixpoint HM [f:form] : Prop :=
<Prop>Case f of
    (*f_atom d*) [d:D](drunk d)
    (*f_not f0*) [f0:form]~(HM f0)
 (*f_and f0 f1*) [f0,f1:form](HM f0)/\(HM f1)
  (*f_or f0 f1*) [f0,f1:form](HM f0)\/(HM f1)
(*f_impl f0 f1*) [f0,f1:form](HM f0)->(HM f1)
     (*f_ex df*) [df:D->form](Ex [x:D](HM (df x)))
    (*f_all df*) [df:D->form](x:D)(HM (df x))
end.
```

See Figure 1 for the schema of the clausification. A given first-order formula ϕ in Prop will be translated to its corresponding formal counter-part f_ϕ by a syntax-based translation TR outside Coq (as yet: by hand). Since objects in form are homomorphic to certain objects in Prop, we have defined above the canonical homomorphism HM from form to Prop. HM and TR should be such that ϕ and HM(TR(ϕ)) are syntactically identical, whenever TR applies. Also, TR(HM f_ϕ) has to be identical to f_ϕ, for every f_ϕ. When applied to any given f_ϕ, the function CLAUS computes the clausal form of ϕ as an object in Prop. Preservation of logical truth modulo the principle of Excluded Middle (EM), the Axiom of Choice (AC) and the condition that D is non-empty, is ensured by the following theorem:

```
Theorem CLAUSeq: EM->AC->D->(f_phi:form)(HM f_phi)<->(CLAUS f_phi).
```

The complete clausification program and the proof of the above theorem can be found on the following Internet address:

http://www.phil.ruu.nl/~bezem/Coq/OinC.v

4. Back to the Drinker's Principle

We phrase the formal counter-part of the discussed Drinker's Principle as follows:

```
Definition f_DP :=
          (f_ex [x:D](f_impl (f_atom x)(f_all [y:D](f_atom y)))).
```

[6]HM can also be viewed as a truth predicate. The classical complications entailed by such a predicate (diagonalization, paradoxes, and worse) are properly avoided by Coq. For example, we cannot take form for D in the inductive definition of form above, as D occurs negatively in the argument types of the constructors f_ex and f_all. Furthermore, after abstraction from D as parameter of the inductive definition, we get form:Set->Set and we cannot apply form to itself for simple typing reasons.

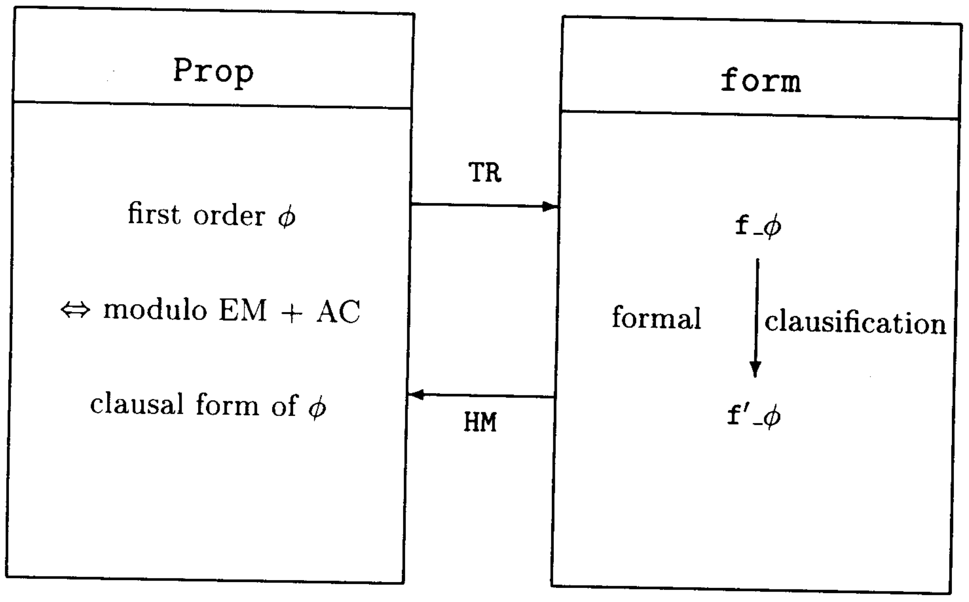

Figure 1: Schema of the clausification process; the proof of the equivalence in the left box can be generated uniformly in f_ϕ.

Indeed, applying HM yields the desired result:

```
Compute (HM f_DP).
     = (Ex [x:D](drunk x)->(y:D)(drunk y)) : Prop
```

To illustrate CLAUS we compute the clausal form of the negation of the Drinker's Principle:

```
Compute (CLAUS (f_not f_DP)).
     = (Ex [f:SKF]
        (x:D)(drunk x)/\(y:D)~(drunk (f O (cons y nil)))) : Prop
```

Here SKF is the type of the Skolem functions.

Now we prove the Drinker's Principle again, this time by clausification and resolution. The clausification is done automatically and thereafter the clauses are resolved by hand.

```
Lemma DP : EM->AC->D->(HM f_DP).
```

First we introduce the hypotheses into the context. Then *reductio ad absurdum* is applied, i.e. (HM f_DP) is negated and False is to be deduced.

```
  em : EM
  ac : AC
  d : D
  H : ~(HM f_DP)
  ============================
   False
```

We want to have the clausal form of H in the context. So we use theorem CLAUSeq from left to right, called CLAUSimpl:

```
Elim ((CLAUSimpl em ac d (f_not f_DP)) H).
Simpl.
Intros f H0.
```

```
Elim HO.
Intros clause1 clause2.

  em : EM
  ac : AC
  d : D
  H : ~(HM f_DP)
  f : SKF
  HO : ((x:D)(drunk x))/\((y:D)~(drunk (f 0 (cons y nil))))
  clause1 : (x:D)(drunk x)
  clause2 : (y:D)~(drunk (f 0 (cons y nil)))
  ============================
   False
```

Now we can compare claus1 and claus2 with the ones Otter came up with:

```
1 [] drunk(x).
2 [] -drunk($f1(x)).
```

They are essentially the same. The Skolem function `f` in `clause2` may look more complicated, but the term `(f 0 (cons y nil))` is, after appropriate renaming, the same as `$f1(x)`.

Until here the proof procedure has been automated. Due to the simplicity of the example, the Coq-proof can now be completed easily:

```
Apply (clause2 d).
Apply clause1.
Qed.
```

The result is a lambda term as proof which is about two times as big as the handcrafted lambda term from Section 6.2.

What remains to be automated is the general case of uniformly converting resolution proofs to lambda terms.

We conclude with a list of references for further reading on the subject.

References

[1] C. Cornes et al. *The Coq Proof Assistent Reference Manual.* INRIA, 1996.

[2] J.-Y. Girard, Y. Lafont and P. Taylor. *Proofs and Types.* Cambridge Tracts in Theoretical Computer Science 7. Cambridge University Press, 1989.

[3] D.W. Loveland. *Automated theorem proving: a logical basis.* Fundamental studies in computer science. North-Holland, Amsterdam, 1978.

[4] W. McCune. *Otter 3.0 Reference Manual and Guide.* Tech. Report ANL-94/6, Argonne National Laboratory, Argonne, IL, 1994.

6.3 Theory of Genetic Algorithms (extended abstract), by Thomas Bäck, Jeannette M. de Graaf, Joost N. Kok and Walter A. Kosters

Leiden University, Department of Computer Science [7]

1. Introduction

In this paper we introduce some methods for examining the fundamental properties of genetic algorithms ([Hol75, Jon75, Gol89, Mit96]), probabilistic search and optimization algorithms that work on populations of bit strings, i.e., strings of a fixed length consisting of zeroes and ones. Genetic algorithms are a subfield of evolutionary computation. For a good overview of evolutionary computation one may consult the Handbook of Evolutionary Computation [BFM97]. In the Handbook also many applications of genetic algorithms can be found.
We do not give a complete overview of the theory of genetic algorithms; in future papers we intend to cover more related topics. We will present results and methods that give an impression of the kind of theory that is present in evolutionary computation today.
We hope that this paper provides a starting point for researchers in theoretical computer science who are interested in the theory of evolutionary computation. This paper is an extended abstract of a paper that appeared in the Bulletin of the European Association for Theoretical Computer Science, Number 63, October 1997, pp. 161–192.

2. Evolutionary Computation

Genetic algorithms form a subclass of evolutionary computation, as is the case for evolution strategies (e.g., [Rec73, Sch77, Rec94, Sch95, Bäc96]), evolutionary programming (e.g., [FOW66, Fog95]), classifier systems (e.g., [HHNT86]) and genetic programming (e.g., [Koz92a, Kin94]). Evolutionary computation is a problem solving method, inspired by the biological principle of evolution. The algorithms based on this evolution principle are generally called evolutionary algorithms. They are mostly used to solve search and optimization problems. For more general overviews of the field the reader is referred to [Fog95, Bäc96, Mic96, BFM97]; see also Section 6.3.

2.1 Evolutionary Algorithms

Evolutionary algorithms work on populations of individuals, each individual representing a point in search space. An individual may occur more than once in a population, i.e., populations are multisets of individuals. There is a fitness function f (also referred to as objective function) which measures the quality of the individuals in the population. An evolutionary algorithm tries to improve (to a better fitness) a population of individuals by creating new search points, the so-called offspring population. Selection of individuals plays an important role in this process: offspring is created by applying genetic operators—mostly recombination and mutation—to selected multisets of individuals and often only selected offspring is used for the new population. Selection is a two-stage process: first it has to be decided who can compete (in analogy with the natural process of mating selection) and after applying the genetic operators a selection mechanism (analogous with the natural process of environmental selection) decides who will survive. The general form of an evolutionary algorithm is:

```
$t := 0$;
initialize(P(0));
evaluate(P(0));
**while** not done **do**
    $P'(t)$ := mating_selection($P(t)$);
    $P'(t)$ := recombination($P'(t)$);
    $P'(t)$ := mutation($P'(t)$);
```

[7]Thomas Bäck is also affiliated with the Center for Applied Systems Analysis, Informatik-Centrum Dortmund, Joseph-von-Fraunhofer-Str. 20, 44227 Dortmund, Germany

```
        evaluate(P'($t$));
        P($t+1$) := environmental_selection(P($t$), P'($t$));
        $t := t + 1$;
    od
```

In the algorithm, t denotes a generation counter, and evaluate(P) implies an evaluation of the objective function values for all members of population P. The algorithm terminates when for instance the fitness of the current population P(t) at time t does not improve anymore, or after a fixed number of iterations.

2.2 Selection

The selection process is usually based on the fitness of the individuals. There are three parameters that play an important role: μ is the population size, λ is the number of offspring and κ is the maximal lifetime (or age) of individuals in the population. They are usually fixed during the evolution process.
Next we describe four popular selection mechanisms. First we have examples of mating selection, whereas the last one is the most commonly used environmental selection. An overview of selection methods commonly used in evolutionary algorithms may be found in [DG91, Bäc94b, BT96].

1. Proportional selection ([Hol75]): The probability for an individual to be selected is its fitness divided by the sum of the fitness values of the individuals occurring in the population. This is also called roulette-wheel selection ([Gol89]). For every individual one allocates some "space" on the roulette wheel proportional to the fitness. To select an individual the roulette-wheel is turned and an individual is selected. To select m individuals this process is repeated m times.

2. Tournament selection: Selection of an individual is done by means of a tournament. Uniformly random a number q of individuals is chosen from the population and the one with the highest fitness among those is selected. To select m individuals this process is repeated m times. The parameter of this selection mechanism is the size q of the tournament.

3. Linear ranking ([Bak85]): Individuals in the population are sorted according to their fitness values. Each individual is assigned a rank in the range from one to the number of elements in the population μ. Let $\mathcal{P}(i)$ denote the probability that individual i (in rank) is chosen. The lowest rank corresponds to the lowest fitness. Linear ranking has a parameter α for the probability that the individual with rank one is chosen:

$$\mathcal{P}(1) = \alpha \, .$$

   Now each next element in rank gets an offset of β added to the probability of the individual one lower in rank:

$$\mathcal{P}(i+1) = \mathcal{P}(i) + \beta \, .$$

   Individuals are chosen relative to their rank instead of their fitness value. The offset β should be chosen such that

$$\sum_i P(i) = 1 \, .$$

4. Deterministic selection ([Sch77, Sch81, BS93, BHS97]): The best μ individuals are selected.

A more detailed analysis of different selection schemes can be found in the work of Bäck [Bäc94a], Deb and Goldberg [DG91] and Blickle and Thiele [BT96].

2.3 Recombination and Mutation

Generally a recombination operator repeatedly takes two individuals (the parents) and produces two or more new individuals (the offspring), which are a combination of their parents. The most popular example is crossover, an operator that exchanges a (usually small) number of random substrings between the parents. For a detailed overview of recombination operators in genetic algorithms, the reader is referred to the Handbook of Evolutionary Computation [BFM97].
The mutation operator toggles bits with a certain small probability. Mutation guarantees that the evolutionary algorithm can reach every point in the search space, and even ensures convergence under suitable conditions.

3. Genetic Algorithms

In genetic algorithms individuals are encoded as bit strings, i.e., strings of a fixed length n consisting of zeroes and ones. A population is a multiset of bit strings of fixed length. A genetic algorithm applies genetic operators to selected individuals in order to improve the multiset, or rather the fitness of its individuals. A standard mating selection mechanism is proportional selection, and standard genetic operators are uniform crossover [Sys89], one-point crossover [Hol75] and mutation [Hol75]. After applying the genetic operators, giving new offspring, the offspring replaces the parent population.
The genetic algorithm has the following form:

```
$t := 0$;
initialize(P(0));
evaluate(P(0));
while not done do
    $P'(t)$ := proportional_selection($P(t)$);
    $P'(t)$ := crossover($P'(t)$);
    $P'(t)$ := mutation($P'(t)$);
    evaluate($P'(t)$);
    $P(t+1) := P'(t)$;                (*)
    $t := t+1$;
od
```

Many generalizations of the classical genetic algorithm as defined above are used nowadays, and there is not always a clear distinction between these generalized genetic algorithms and other evolutionary algorithms. It is for instance quite natural to keep the best individual in the population in order to ensure an improving course of evolution. This method is often called elitism. When assumptions like elitism must hold one should consider generalized genetic algorithms with a slight adaptation of (*), for example by always keeping the best individual from $P(t)$ in the new population $P'(t)$.
In Section 6.3 we described the method of proportional selection. Next we explain how a population changes under the genetic operators uniform crossover, one-point crossover and mutation. Uniform crossover ([Sys89]) takes two bit strings (parents) and with probability p_c (the crossover rate) produces an offspring by taking randomly, with equal probability, as the i-th element one of the two i-th elements from the parents. One-point crossover ([Hol75]) takes two parents and with crossover rate p_c selects a random crossover point $\ell \in \{1, \ldots, n-1\}$ and gives as result a bit string consisting of the ℓ first bits from one parent and the $n-\ell$ last bits from the other parent. In both crossovers, if the operation is not chosen (i.e., with probability $1-p_c$), then the result is (randomly, with equal probability) one of the parents. A genetic algorithm using one-point crossover is called a simple genetic algorithm. Mutation changes each element of a bit string to the opposite value with probability p_m (the mutation rate). This mutation rate is usually a very small value assuring mutation to happen rarely [Hol75, Gol89].

4. The Schema Theorem

The first attempt to theoretically analyze the performance of genetic algorithms led to the well-known Schema Theorem by Holland ([Hol75], [Gol89]). It explains the nature of a genetic algorithm in terms of schemata, and the effect of crossover and mutation on them. We assume here that the genetic algorithm uses one-point crossover, so we consider simple genetic algorithms. Furthermore the population size is finite (and fixed).

A schema describes a subset of strings (individuals) with similarities at certain positions. To be more precise: a schema is a string of length n containing characters from the alphabet $\{0, 1, *\}$, and it represents all strings that match it on every position other than those with a $*$. For example, the schema $*101*0$ represents the four strings 010100, 010110, 110100 and 110110. The order of a schema H, denoted by $o(H)$, is the number of fixed positions, i.e., the number of 0's and 1's present in the schema. For example, $o(*10*1*) = 3$. The defining length of a schema H, denoted by $\delta(H)$, is the distance between the first and the last fixed position in H. For example, $\delta(*10**0) = 6 - 2 = 4$. The fitness of a schema in a population is defined as the average fitness of all strings in the population that match the schema.

In order to determine the combined effect of the genetic operators proportional selection, one-point crossover and mutation on schemata contained within a population of strings, let $m(H, t)$ be the expected number of strings in the population at time t matching the schema H. Then it is not difficult to show that ([Hol75, Gol89]):

$$m(H, t+1) \geq m(H, t) \cdot \frac{f(H)}{\overline{f}} \cdot (1 - p_c \cdot \frac{\delta(H)}{n-1}) \cdot (1 - p_m)^{o(H)} .$$

Here f is the fitness function, $f(H)$ denotes the average fitness of the schema H and $\overline{f}$ is the average fitness of the population. From this equation Holland concluded ([Gol89, Mic96]):

Schema Theorem: Short, low-order, above-average schemata receive an exponentially increasing number of occurrences in subsequent populations.

Of course the exponential increase of schema-occurrences can only be true for a short period, since the population size is finite. Furthermore, from Equation (6.3) at a certain time t we can not conclude in general an increase of $m(H, t)$ in subsequent generations, since the average fitness $f(H)$ of the schema and $\overline{f}$ of the population change when the population changes.

The short, low-order, above-average schemata, which seem to play an important role in the performance of the genetic algorithm, were named building blocks. The genetic algorithm discovers these building blocks and then repeatedly recombines them in order to produce strings of higher fitness, or in the words of Goldberg [Gol89]:

Building Block Hypothesis: A genetic algorithm seeks near optimal performance through the juxtaposition of building blocks.

There is a lot of discussion on the value of this hypothesis and whether or not it really explains the true nature of a genetic algorithm. Although some research has been done in order to support or even prove this hypothesis, only empirical results are available (see for a recent empirical result e.g., [WLR97]). Several authors have offered their remarks on Holland's hypothesis. See for instance Grefenstette and Baker [GB89] and Bäck [Bäc96], where some remarks on the multi-armed bandit analogy used by Holland are offered. Note that the hypothesis implies that the coding of the problem at hand is very important and should stimulate the formation of building blocks. (See [Rad94, RS95] for a discussion of more general representations and the corresponding generalizations of schema theory.)

A further possible—and sometimes criticized—explanation of the reason why genetic algorithms perform well was given by Holland when he showed that in a population of μ individuals there are at least μ^3 schemata being processed usefully, that is with an exponentially increasing amount of occurrences. Holland called this property implicit parallelism. (See Holland [Hol75], and later Bertoni and Dorigo [BD93].)

Note that the results in this section are more or less independent of the fitness function f. More knowledge about f should give more information and better results. Furthermore one might question the practical usefulness of the results stated, especially because the Schema Theorem focusses on the destructive rather than on the constructive properties of genetic operators. In recent years more analytic results have been given to explain the performance of genetic algorithms and (more general) of evolutionary algorithms, without the use of schemata.

5. Probabilities

Following Liepins and Vose [LV91], we consider infinite populations, i.e., we view a population as a probability distribution and we study the way in which such a distribution changes under the genetic operators. By repeated application of the genetic operators to the distributions, it is possible to trace the distribution from generation to generation, thus simulating genetic algorithms. There is a relationship between the deterministic path of the distributions and models of genetic algorithms with finite population size motivating the tracing of distributions (Nix and Vose [NV92], Vose [Vos92], Whitley [Whi92]). Using special functions (e.g., Walsh products [FK95]) and Fourier analysis ([KKF97]) it is possible to give a detailed analysis of this evolving distribution. Instead of experimenting with just a few populations this theory enables us to follow the full evolution of the distribution.

We use distributions $P(x)$ to express the probability that a string x occurs at a certain time. A distribution associates to each bit string x a probability $P(x)$ such that $\sum_x P(x) = 1$. We are interested in the connection between the distribution just before $(P(x))$ and immediately after $(P'(x))$ a genetic operation. Of course, it is also possible to keep part of the original distribution, using $(1-q)P(x) + qP'(x)$, where q denotes the probability of the genetic operation involved.

We identify bit strings consisting of n bits with subsets of a universe U with n elements in the appropriate way: an element of U is in the "set" x if and only if the corresponding bit equals 1. Using the usual binary representation it is also possible to view bit strings as integers.

First we consider how the distribution changes under the application of the operators. From the definitions of the operators we obtain the following formulae corresponding to the Vose and Liepins model [LV91].

For proportional selection, the probability of a bit string is weighted proportional to its fitness. For example, if the fitness of a bit string is twice as good as the average, proportional selection results in doubling the probability. Selection uses a fitness function f, and satisfies

$$P'(x) = \frac{f(x)}{E[f]} P(x) \ ,$$

where $E[f]$ denotes the expected value of f with respect to the probability distribution P.

For mutation we have

$$P'(x) = \sum_{y} p^{||\mathrm{xor}(x,y)||} (1-p)^{n-||\mathrm{xor}(x,y)||} P(y) \ ,$$

where $p \in [0,1]$ is the mutation rate, $\mathrm{xor}(l,m) = (l \setminus m) \cup (m \setminus l)$ is the symmetric difference of the sets l and m; here $l \setminus m$ consists of those elements in l that are not in m and $||s||$ denotes the number of elements of the set s. The summation runs over all bit strings (sets) y; in order to generate x all bits in $\mathrm{xor}(x,y)$ should flip, the other bits remaining the same.

Uniform crossover can be expressed as follows. If both parents have a 1 in a certain bit position, the child receives a 1 too (then we are part of the intersection of the two sets); the same holds for a 0. If the two bits from the parents differ, 0 and 1 are equally likely for the child (now we are part of the symmetric difference). Hence we get

$$P'(x) = \sum_{\substack{y,z \\ y \cap z \subseteq x \subseteq y \cup z}} (1/2)^{||\mathrm{xor}(y,z)||} P(y) P(z) \ .$$

For Walsh products $R_i(x) = (-1)^{||i \setminus x||}$, one can show that in this case the expected values of these functions after and before the crossover are related by

$$E'[R_i] = (1/2)^{||i||} \sum_{j \subseteq i} E[R_j] E[R_{i \setminus j}] \, .$$

In [FK95] and [KKF97] this and similar results are basic for a deeper analysis of the genetic algorithm.
For one-point crossover we take the sum over the crossover points of the probabilities for the corresponding prefix and the corresponding postfix:

$$P'(x) = \frac{1}{n-1} \sum_{\ell=1}^{n-1} \left(\sum_{\substack{y \,:\, prefix(y,\ell)= \\ prefix(x,\ell)}} P(y) \cdot \sum_{\substack{z \,:\, postfix(z,n-\ell)= \\ postfix(x,n-\ell)}} P(z) \right) ,$$

where $prefix(x,\ell)$ means the prefix of length ℓ of bit string x and $postfix(x,\ell)$ denotes the postfix of length ℓ of bit string x.
If we consider the simple genetic algorithm (proportional selection followed by one-point crossover and mutation), then we can combine the definitions into a transition matrix. This matrix has a number of interesting symmetries and it is possible to numerically trace the probabilities (see Liepins and Vose [LV91] and Whitley [Whi92]).

6. Convergence

Suppose we have a genetic algorithm, operating on a finite set of populations. For the sake of convenience, the fitness of a population is defined as the fitness of its best individual. Now we have (cf. [EAH91]):

Convergence theorem: Suppose that the genetic algorithm is elitist, meaning that in every step at least the current best individual survives. Suppose furthermore that for every population P there is a nonzero probability $\mathcal{P}(P)$ that in the next generation the fitness of the population is really better. Then the fitness of the population at time t converges to the optimal value, for $t \to \infty$.

For most genetic algorithms the sequence of populations is a Markov chain. The relation between the theory of Markov chains and evolutionary algorithms is extensively studied in [Rud96], giving deeper insight in the convergence rate.

7. Further Reading

We do not claim that the overview given in this paper is close to complete. Only some of the methods used in evolutionary computation are treated. If one wants to know more about the theory and applications of genetic algorithms or other types of evolutionary algorithms one is referred to the references already mentioned in the text and the suggestions for further reading below.
As good starting points for further studies on the different types of evolutionary algorithms we recommend [Bäc96, BFM97, Dav91, Fog95, Gol89, Kin94, Koz92b, Mic96, Mit96, Sch95]. Several moderated mailing lists (e.g., `GA-List-Request@aic.nrl.navy.mil` and `EP-List-Request@magenta.me.fau.edu` and newsgroups (such as `comp.ai.genetic`) allow for keeping track of current events and discussions in the field. Useful WWW-sites are:

`http://alife.santafe.edu/~joke/encore/www/`,
`http://www.aic.nrl.navy.mil/galist`,
`http://www.dcs.napier.ac.uk/evonet/evonet.htm` and
`http://www.scs.carleton.ca/csgs/resources/gaal.html`.

There are several relevant conferences in the field of evolutionary computation:

- Parallel Problem Solving from Nature, PPSN [VERS96],
- International Conference on Genetic Algorithms, ICGA [Bäc97],
- IEEE International Conference on Evolutionary Computation, ICEC [IEE97],
- Annual Conference on Evolutionary Programming, EP [ARME97],
- Annual Conference on Genetic Programming, GP [KDD+97].

References

[ARME97] P. J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart, editors. *Proceedings of the Sixth International Conference on Evolutionary Programming, EP97*, volume 1213 of *Lecture Notes in Computer Science*. Springer, Berlin, 1997.

[Bäc94a] Th. Bäck. Evolutionary algorithms: Comparison of approaches. In R. C. Paton, editor, *Computing with Biological Metaphors*, chapter 14, pages 227–243. Chapman & Hall, London, 1994.

[Bäc94b] Th. Bäck. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 57–62. IEEE Press, Piscataway, NJ, 1994.

[Bäc96] Th. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[Bäc97] Th. Bäck, editor. *Genetic Algorithms: Proceedings of the 7th International Conference*. Morgan Kaufmann Publishers, San Francisco, CA, 1997.

[Bak85] J. E. Baker. Adaptive selection methods for genetic algorithms. In J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pages 101–111. Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.

[BD93] A. Bertoni and M. Dorigo. Implicit parallellism in genetic algorithms. *Artificial Intelligence*, 61(2):307–314, 1993.

[BFM97] Th. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Oxford University Press, New York, and Institute of Physics Publishing, Bristol, 1997.

[BHS97] Th. Bäck, U. Hammel, and H.-P. Schwefel. Evolutionary computation: History and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17, 1997.

[BS93] Th. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[BT96] T. Blickle and L. Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4):361–394, 1996.

[Dav91] L. Davis, editor. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[DG91] K. Deb and D. E. Goldberg. Analyzing deception in trap functions. IlliGAL Report 91009, University of Illinois at Urbana-Champaign, IL, December 1991.

[EAH91] A. E. Eiben, E. H. L. Aarts, and K. M. Van Hee. Global convergence of genetic algorithms: An infinite Markov chain analysis. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature — Proceedings 1st Workshop PPSN I*, volume 496 of *Lecture Notes in Computer Science*, pages 4-12. Springer, Berlin, 1991.

[FK95] P. Floréen and J.N. Kok. Tracing the behavior of genetic algorithms using expected values of bit and Walsh products. In L.J. Eshelman, editor, *Proceedings of the sixth international conference on genetic algorithms*, pages 201–208. Morgan Kaufmann Publishers, Inc., 1995.

[Fog95] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ, 1995.

[FOW66] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, New York, 1966.

[GB89] J. J. Grefenstette and J. E. Baker. How genetic algorithms work: A critical look at implicit parallelism. In Schaffer [Sch89], pages 20–27.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, Reading, MA, 1989.

[HHNT86] J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard. *Induction: Processes of Inference, Learning, and Discovery.* The MIT Press, Cambridge, MA, 1986.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, MI, 1975.

[IEE97] IEEE. *Proceedings of the Fourth IEEE Conference on Evolutionary Computation, Indianapolis, IN.* IEEE Press, Piscataway, NJ, 1997.

[Jon75] K. A. De Jong. *An analysis of the behaviour of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975. Diss. Abstr. Int. 36(10), 5140B, University Microfilms No. 76–9381.

[KDD$^+$97] J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors. *Genetic Programming 1997. Proceedings of the Second Annual Conference.* Morgan Kaufmann, San Francisco, CA, 1997.

[Kin94] K. E. Kinnear, editor. *Advances in Genetic Programming.* The MIT Press, Cambridge, MA, 1994.

[KKF97] W.A. Kosters, J.N. Kok, and P. Floréen. Fourier analysis of genetic algorithms. Preprint, Leiden University, 1997. To appear in Theoretical Computer Science.

[Koz92a] J. R. Koza. *Genetic Programming.* MIT Press, 1992.

[Koz92b] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Complex Adaptive Systems. The MIT Press, Cambridge, MA, 1992.

[LV91] G. E. Liepins and M. D. Vose. Polynomials, basis sets, and deceptiveness in genetic algorithms. *Complex Systems*, 5:45–61, 1991.

[Mic96] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, Berlin, 1996.

[Mit96] M. Mitchell. *An Introduction to Genetic Algorithms.* The MIT Press, Cambridge, MA, 1996.

[NV92] A. Nix and M. D. Vose. Modeling genetic algorithms with markov chains. *Annals of Mathematics and Artificial Intelligence*, 5:79–88, 1992.

[Rad94] N. J. Radcliffe. The algebra of genetic algorithms. *Annals of Maths and Artificial Intelligence*, 10:339–384, 1994.

[Rec73] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann–Holzboog, Stuttgart, 1973.

[Rec94] I. Rechenberg. *Evolutionsstrategie '94*, volume 1 of *Werkstatt Bionik und Evolutionstechnik.* Frommann–Holzboog, Stuttgart, 1994.

[RS95] N. J. Radcliffe and P. D. Surry. Fitness variance of formae and performance prediction. In M. D. Vose L. D. Whitley, editor, *Foundations of Genetic Algorithms 3*, pages 51–72. Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[Rud96] G. Rudolph. *Convergence Properties of Evolutionary Algorithms.* PhD Thesis, University of Dortmund, 1996.

[Sch77] H.-P. Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*, volume 26 of *Interdisciplinary Systems Research.* Birkhäuser, Basel, 1977.

[Sch81] H.-P. Schwefel. *Numerical Optimization of Computer Models.* Wiley, Chichester, 1981.

[Sch89] J. D. Schaffer, editor. *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[Sch95] H.-P. Schwefel. *Evolution and Optimum Seeking.* Sixth-Generation Computer Technology Series. Wiley, New York, 1995.

[Sys89] G. Syswerda. Uniform crossover in genetic algorithms. In Schaffer [Sch89], pages 2–9.

[VERS96] H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors. *Parallel Problem Solving from Nature IV. Proceedings of the International Conference on Evolutionary Computation*, volume 1141 of *Lecture Notes in Computer Science*. Springer, Berlin, 1996.

[Vos92] M. D. Vose. Modeling simple genetic algorithms. In Whitley [Whi93], pages 63–73.

[Whi92] L. D. Whitley. An executable model of a simple genetic algorithm. In *Foundations of Genetic Algorithms 2* [Whi93], pages 45–62.

[Whi93] L. D. Whitley, editor. *Foundations of Genetic Algorithms 2*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[WLR97] A.S. Wu, R.K. Lindsay, and R.L. Riolo. Empirical observations on the roles of crossover and mutation. In Bäck [Bäc97], pages 362–369.

Pages Deleted

The following pages contain(ed) the memberlist of the NVTI. These pages have been deleted to protect the privacy of our members.

8 Statuten

Artikel 1.

1. De vereniging draagt de naam: "Nederlandse Vereniging voor Theoretische Informatica".
2. Zij heeft haar zetel te Amsterdam.
3. De vereniging is aangegaan voor onbepaalde tijd.
4. De vereniging stelt zich ten doel de theoretische informatica te bevorderen haar beoefening en haar toepassingen aan te moedigen.

Artikel 2.

De vereniging kent gewone leden en ereleden. Ereleden worden benoemd door het bestuur.

Artikel 3.

De vereniging kan niet worden ontbonden dan met toestemming van tenminste drievierde van het aantal gewone leden.

Artikel 4.

Het verenigingsjaar is het kalenderjaar.

Artikel 5.

De vereniging tracht het doel omschreven in artikel 1 te bereiken door

a. het houden van wetenschappelijke vergaderingen en het organiseren van symposia en congressen;

b. het uitgeven van een of meer tijdschriften, waaronder een nieuwsbrief of vergelijkbaar informatiemedium;

c. en verder door alle zodanige wettige middelen als in enige algemene vergadering goedgevonden zal worden.

Artikel 6.

1. Het bestuur schrijft de in artikel 5.a bedoelde bijeenkomsten uit en stelt het programma van elk van deze bijeenkomsten samen.
2. De redacties der tijdschriften als bedoeld in artikel 5.b worden door het bestuur benoemd.

Artikel 7.

Iedere natuurlijke persoon kan lid van de vereniging worden. Instellingen hebben geen stemrecht.

Artikel 8.

Indien enig lid niet langer als zodanig wenst te worden beschouwd, dient hij de ledenadministratie van de vereniging daarvan kennis te geven.

Artikel 9.

Ieder lid ontvangt een exemplaar der statuten, opgenomen in de nieuwsbrief van de vereniging. Een exemplaar van de statuten kan ook opgevraagd worden bij de secretaris. Ieder lid ontvangt de tijdschriften als bedoeld in artikel 5.b.

Artikel 10.

Het bestuur bestaat uit tenminste zes personen die direct door de jaarvergadering worden gekozen, voor een periode van drie jaar. Het bestuur heeft het recht het precieze aantal bestuursleden te bepalen. Bij de samenstelling van het bestuur dient rekening gehouden te worden met de wenselijkheid dat vertegenwoordigers van de verschillende werkgebieden van de theoretische informatica in Nederland in het bestuur worden opgenomen. Het bestuur kiest uit zijn midden de voorzitter, secretaris en penningmeester.

Artikel 11.

Eens per drie jaar vindt een verkiezing plaats van het bestuur door de jaarvergadering. De door de jaarvergadering gekozen bestuursleden hebben een zittingsduur van maximaal twee maal drie jaar. Na deze periode zijn zij niet terstond herkiesbaar, met uitzondering van secretaris en penningmeester. De voorzitter wordt gekozen voor de tijd van drie jaar en is na afloop van zijn ambtstermijn niet onmiddellijk als zodanig herkiesbaar. In zijn functie als bestuurslid blijft het in de vorige alinea bepaalde van kracht.

Artikel 12.

Het bestuur stelt de kandidaten voor voor eventuele vacatures. Kandidaten kunnen ook voorgesteld worden door gewone leden, minstens een maand voor de jaarvergadering via de secretaris. Dit dient schriftelijk te gebeuren op voordracht van tenminste vijftien leden. In het geval dat het aantal kandidaten gelijk is aan het aantal vacatures worden de gestelde kandidaten door de jaarvergadering in het bestuur gekozen geacht. Indien het aantal kandidaten groter is dan het aantal vacatures wordt op de jaarvergadering door schriftelijke stemming beslist. Ieder aanwezig lid brengt een stem uit op evenveel kandidaten als er vacatures zijn. Van de zo ontstane rangschikking worden de kandidaten met de meeste punten verkozen, tot het aantal vacatures. Hierbij geldt voor de jaarvergadering een quorum van dertig. In het geval dat het aantal aanwezige leden op de jaarvergadering onder het quorum ligt, kiest het zittende bestuur de nieuwe leden. Bij gelijk aantal stemmen geeft de stem van de voorzitter (of indien niet aanwezig, van de secretaris) de doorslag.

Artikel 13.

Het bestuur bepaalt elk jaar het precieze aantal bestuursleden, mits in overeenstemming met artikel 10. In het geval van aftreden of uitbreiding wordt de zo ontstane vacature aangekondigd via mailing of nieuwsbrief, minstens twee maanden voor de eerstvolgende jaarvergadering. Kandidaten voor de ontstane vacatures worden voorgesteld door bestuur en gewone leden zoals bepaald in artikel 12. Bij aftreden van bestuursleden in eerste of tweede jaar van de driejarige cyclus worden de vacatures vervuld op de eerstvolgende jaarvergadering. Bij aftreden in het derde jaar vindt vervulling van de vacatures plaats tegelijk met de algemene driejaarlijkse bestuursverkiezing. Voorts kan het bestuur beslissen om vervanging van een aftredend bestuurslid te laten vervullen tot de eerstvolgende jaarvergadering. Bij uitbreiding van het bestuur in het eerste of tweede jaar van de cyclus worden de vacatures vervuld op de eerstvolgende jaarvergadering. Bij uitbreiding in het derde jaar vindt vervulling van de vacatures plaats tegelijk met de driejaarlijkse bestuursverkiezing. Bij inkrimping stelt het bestuur vast welke leden van het bestuur zullen aftreden.

Artikel 14.

De voorzitter, de secretaris en de penningmeester vormen samen het dagelijks bestuur. De voorzitter leidt alle vergaderingen. Bij afwezigheid wordt hij vervangen door de secretaris en indien ook deze afwezig is door het in jaren oudste aanwezig lid van het bestuur. De secretaris is belast met het houden der notulen van alle huishoudelijke vergaderingen en met het voeren der correspondentie.

Artikel 15.

Het bestuur vergadert zo vaak als de voorzitter dit nodig acht of dit door drie zijner leden wordt gewenst.

Artikel 16.

Minstens eenmaal per jaar wordt door het bestuur een algemene vergadering bijeengeroepen; één van deze vergaderingen wordt expliciet aangeduid met de naam van jaarvergadering; deze vindt plaats op een door het bestuur te bepalen dag en plaats.

Artikel 17.

De jaarvergadering zal steeds gekoppeld zijn aan een wetenschappelijk symposium. De op het algemene gedeelte vaan de jaarvergadering te behandelen onderwerpen zijn

a. Verslag door de secretaris;

b. Rekening en verantwoording van de penningmeester;

c. Verslagen van de redacties der door de vereniging uitgegeven tijdschriften;

d. Eventuele verkiezing van bestuursleden;

e. Wat verder ter tafel komt. Het bestuur is verplicht een bepaald punt op de agenda van een algemene vergadering te plaatsen indien uiterlijk vier weken van te voren tenminste vijftien gewone leden schriftelijk de wens daartoe aan het bestuur te kennen geven.

Artikel 18.

Deze statuten kunnen slechts worden gewijzigd, nadat op een algemene vergadering een commissie voor statutenwijziging is benoemd. Deze commissie doet binnen zes maanden haar voorstellen via het bestuur aan de leden toekomen. Gedurende drie maanden daarna kunnen amendementen schriftelijk worden ingediend bij het bestuur, dat deze ter kennis van de gewone leden brengt, waarna een algemene vergadering de voorstellen en de ingediende amendementen behandelt. Ter vergadering kunnen nieuwe amendementen in behandeling worden genomen, die betrekking hebben op de voorstellen van de commissie of de schriftelijk ingediende amendementen. Eerst wordt over elk der amendementen afzonderlijk gestemd; een amendement kan worden aangenomen met gewone meerderheid van stemmen. Het al dan niet geamendeerde voorstel wordt daarna in zijn geheel in stemming gebracht, tenzij de vergadering met gewone meerderheid van stemmen besluit tot afzonderlijke stemming over bepaalde artikelen, waarna de resterende artikelen in hun geheel in stemming gebracht worden. In beide gevallen kunnen de voorgestelde wijzigingen slechts worden aangenomen met een meerderheid van tweederde van het aantal uitgebrachte stemmen. Aangenomen statutenwijzigingen treden onmiddellijk in werking.

Artikel 19.

Op een vergadering worden besluiten genomen bij gewone meerderheid van stemmen, tenzij deze statuten anders bepalen. Elk aanwezig gewoon lid heeft daarbij het recht een stem uit te brengen. Stemming over zaken geschiedt mondeling of schriftelijk, die over personen met gesloten briefjes. Uitsluitend bij schriftelijke stemmingen worden blanco stemmen gerekend geldig te zijn uitgebracht.

Artikel 20.

a. De jaarvergadering geeft bij huishoudelijk reglement nadere regels omtrent alle onderwerpen, waarvan de regeling door de statuten wordt vereist, of de jaarvergadering gewenst voorkomt.

b. Het huishoudelijk reglement zal geen bepalingen mogen bevatten die afwijken van of die in strijd zijn met de bepalingen van de wet of van de statuten, tenzij de afwijking door de wet of de statuten wordt toegestaan.

Artikel 21.

In gevallen waarin deze statuten niet voorzien, beslist het bestuur.