Nieuwsbrief van de Nederlandse Vereniging voor Theoretische Informatica

Jaco van de Pol Femke van Raamsdonk Marielle Stoelinga

Inhoudsopgave

Van de voorzitter

Geachte NVTI leden,

Hét evenement van de NVTI staat weer voor de deur: de landelijke NVTI theoriedag. Zoals elk jaar is het ook nu weer gelukt om vier internationaal bekende sprekers te vinden uit binnen- en buitenland. Traditiegetrouw spreken zij over zowel de algoritmische, als de logische kant van de theoretische munt. De NVTI heeft hier een hele traditie hoog te houden: zie `http://www.nvti.nl/speakerslist.html` voor een de geweldige sprekerslijst sinds 1995. Ook nu hebben we hooggespannen verwachtingen van Monika Henzinger, Georgies Gonthier, Jan Friso Groote, en Monique Laurent.

Het leeuw(inn)endeel van de organisatie van deze dag was in goede handen bij Femke van Raamsdonk, waarvoor dank. Haar bemoeienis was het gevolg van een wijzing van de samenstelling en de rolverdeling binnen het bestuur van de NVTI. Op de ledenvergadering van 20 maart 2009 werden nieuwe bestuursleden gekozen: Karen Aardal, Herman Geuvers, Wim Hesselink en Han La Poutré. Hiermee zijn nu ook Delft en Nijmegen vertegenwoordigd. Afscheid namen we van Gerard Renardel de Lavalette. Gerard, bedankt!

Deze bestuurswisseling had ook gevolgen voor de rolverdeling: Femke van Raamsdonk is de nieuwe secretaris geworden, Han La Poutré de nieuwe penningmeester, terwijl ondergetekende voorzitter werd. Veel dank bij dezen aan Joost Kok, die de vereniging vijf jaar op voortreffelijke en stabiele wijze geleid heeft.

De andere activiteit van de NVTI is de samenstelling en verspreiding van de nieuwsbrief. Met dank aan Leen Torenvliet, Jos Baeten en Jan Willem Klop zijn alle oude nieuwsbrieven nu ook electronisch beschikbaar via `www.nvti.nl`. Het bestuur heeft besloten Dr. Mariëlle Stoelinga (UT) als redacteur van de nieuwsbrief aan te stellen; het resultaat hebt u in handen! Hiermee is Joost-Pieter Katoen afgelost als (hoofd)redacteur van de NVTI nieuwsbrief. Bedankt Joost-Pieter, en ook dank aan Susanne van Dam die ook dit jaar weer gezorgd heeft voor de vermenigvulding en verspreiding door het CWI. Hierbij wil ik ook de andere sponsors van NVTI vermelden: Elsevier, NWO, CWI, IPA en SIKS: Hartelijk dank voor de trouwe sponsoring.

Rest mij alle leden een heel geslaagde en onderhoudende Theoriedag 2010 toe te wensen. Vergeet vooral niet uw (jonge!) collega's mee te nemen.

Jaco van de Pol
Voorzitter NVTI

Huidige samenstelling van het bestuur

Prof. Dr. Karen Aardal (TUD)
Prof. Dr. Jos Baeten (TU/e)
Prof. Dr. Mark de Berg (TU/e)
Prof. Dr. Harry Buhrman (CWI en UvA)
Prof. Dr. Herman Geuvers (RU)
Prof. Dr. Wim Hesselink (RUG)
Prof. Dr. Ir. Joost-Pieter Katoen (RWTH en UT)
Prof. Dr. Jan Willem Klop (VU en RU)
Prof. Dr. Joost Kok (UL)
Prof. Dr. John-Jules Meyer (UU)
Prof. Dr. Jaco van de Pol (U Twente), voorzitter
Prof. Dr. Ir. Han La Poutré (CWI, TU/e), penningmeester
Dr. Femke van Raamsdonk (VU), secretaris
Dr. Leen Torenvliet (UvA), webmaster

Van de redactie

Beste leden van de NVTI,

u leest nu de 14de editie van de NVTI nieuwsbrief. U vindt hierin het programma van de NVTI dag, mededelingen van de onderzoeksscholen IPA en SIKS en 4, naar mijn mening zeer interessante wetenschappelijke bijdragen:

Taolue Chen en Wan Fokkink schrijven over hun spannende zoektocht naar eindige axiomatiseerbaarheid van BCCSP, een basis process algebra. Zij beantwoorden diverse problemen die al voor lange tijd open stonden, en maken daarmee het plaatje voor de axiomatiseerbaarheid compleet.

Helle Hansen beschrijft een zeer leesbare beschrijving van een belangrijke bijdrage uit haar proefschrift: ze geeft een methode om een Mealy machine te synthetiseren uit een algebraische specificatie van een bit-stream functie. Deze techniek maakt het mogelijk om hardwarecircuits af te leiden uit algebraische specificaties.

Tim Willemse geeft een overzicht van verificatie door middel van parametric boolean equation systems, een methode die steeds belangrijker wordt, onder andere omdat deze model checking en equivalentie checking binnen een framework verenigt.

Cees Witteveen beschrijft hoe je constraint systemen op zo'n manier kunt decomponeren dat oplossingen voor het globale systeem verkregen kunnen worden uit oplossingen voor de deelsystemen. Constraint systemen kennen een veelheid van toepassingen (oa model checking, databases, sensor netwerken) en Witteveen gaat in op een aantal fundamentele eigenschappen van decompositie.

Tenslotte wil ik Susanne van Dam en Axel Belinfante bedanken voor hun uitstekende ondersteuning.

Namens de redactie,
Marielle Stoelinga

Nederlandse Vereniging voor Theoretische Informatica

We are happy to invite you for the Theory Day 2010 of the NVTI. The Dutch Asssociation for Theoretical Computer Science (NVTI) supports the study of theoretical computer science and its applications.

NVTI Theory Day 2010
Friday March 12, 2010, 9:30-16:40
Hoog Brabant, Utrecht (close to Central Station)

We have an interesting program with excellent speakers from The Netherlands and abroad, covering important streams in theoretical computer science. Below you will find the abstracts. Speakers:

Jan Friso Groote (TU/e)
Monika Henzinger (University of Vienna, Austria)
Georges Gonthier (Microsoft Research)
Monique Laurent (CWI, Tilburg University)

It is possible to participate in the organized lunch, for which registration is required. Please register with Ms Caroline Waij (cpwaij@few.vu.nl or 020-5983563) no later than one week before the meeting (March 5, 2010). The costs of 15 Euro can be paid at the location. We just mention that in the direct vicinity of the meeting room there are plenty of nice lunch facilities as well.

The NVTI theory day 2010 is sponsored (financially or in kind) by NWO (Netherlands Organisation for Scientific Research), Elseviers Science, CWI (Dutch Center of Mathematics and Computer Science) and the Dutch research schools IPA (Institute for Programming Research and Algorithmics) and SIKS (Dutch research school for Information and Knowledge Systems).

Please find the full program and abstracts of the lectures below.

Kind regards,
Femke van Raamsdonk,
NVTI secretary.

Program of the NVTI Day on Friday March 12, 2010

9.30-10.00:	Arrival with Coffee	
10.00-10.10:	Opening	
10.10-11.00:	Speaker:	Jan Friso Groote (TU/e)
	Title:	Parameterised Boolean Equation Systems
11.00-11.30:	Coffee/Tea	
11.30-12.20:	Speaker:	Monika Henzinger (University of Vienna, Austria)
	Title:	Algorithmic mechanism design or how web search engines make money
12.20-12.40:	Speaker:	Yvette Tuin (NWO)
12.40-14.10:	Lunch (see above for registration)	
14.10-15.00:	Speaker:	Georges Gonthier (Microsoft Research)
	Title:	Beyond the four-colour theorem: software engineering for mathematics
15.00-15.20:	Coffee/Tea	
15.20-16.10:	Speaker:	Monique Laurent (CWI, Tilburg University)
	Title:	Optimization over polynomials with sums of squares and semidefinite programming
16.10-16.40:	Business meeting NVTI	

Abstracts of the talks of NVTI Day on Friday March 12, 2010

10.10-11.00
Speaker: Jan Friso Groote (TU/e)
Title: Parameterised Boolean Equation Systems
Abstract:
We use formulas of the modal mu-calculus with time and data to express properties that behavioural models should have. In order to verify the validity of such formulas, we translate model and formula to a so-called Parameterised Boolean Equation System (PBES). The formula holds iff the solution of the PBES is true. The PBES representation is so concise that translating complex models and complex properties is straightforward and yields relatively concise PBESs (typically smaller than megabytes). Unfortunately, solving large PBESs is not so straightforward.

In this talk we present PBESs and provide a number of techniques that are known to solve them. Gauss elimination, invariants, approximation are examples. Particularly intriguing is the use of patterns which are very helpful in certain instances. It is an open question whether this pattern technique can be lifted to solve any PBES. But if so, it would clearly relate the expressiveness of PBESs to first order logic.

11.30-12.20
Speaker: Monika Henzinger (University of Vienna, Austria)
Title: Algorithmic mechanism design
or how web search engines make money
Abstract:
This talk presents a new problem in algorithmic mechanism design, namely how to assign bidders with potentially very different utility functions to items. We will describe the current state of the art and present a solution for a restricted set of utility functions. We will also explain why the problem is very relevant to internet advertisers and to web search engines.

14.10-15.00
Speaker: Georges Gonthier (Microsoft Research)
Title: Beyond the four-colour theorem:
software engineering for mathematics
Abstract:
While the use of proof assistants has been picking up in computer science, they have yet to become popular in traditional mathematics. Perhaps this is because their main function, checking proofs down to their finest details, is at odds with mathematical practice, which ignores or defers details in order to apply and combine abstractions in creative and elegant ways. This mismatch parallels that between software requirements and implementation.

In this talk we will explore how software engineering techniques like component-based design can be transposed to formal logic and help bridge the gap between rigor and abstraction, and show how these techniques were instrumental in carrying out a fully formal proof of the famous four-colour theorem.

15.20-16.10
Speaker: Monique Laurent (CWI, Tilburg University)
Title: Optimization over polynomials
with sums of squares and semidefinite programming
Abstract:
Polynomial optimization deals with the problem of minimizing a multivariate polynomial over a feasible region defined by polynomial inequalities. While polynomial time solvable when all polynomials are linear (via linear programming), the problem becomes hard in general as soon as it involves non-linear polynomials. Just adding the simple quadratic constraints $x_i^2 = x_{ii}$ on the variables, already makes the problem NP-hard as this models e.g. hard combinatorial problems like Max-Cut or Max-Clique in graphs.

A natural approach is to consider easier to solve, convex relaxations. The basic idea, which goes back to work of Hilbert, is to relax non-negative polynomials (which are hard to recognize) by sums of squares of polynomials, a notion which can be tested efficiently using semidefinite programming algorithms. In this way hierarchies of efficient convex relaxations can be build. We will present their main properties. In particular, convergence properties (that rely on real algebraic geometry representation results for positive polynomials), stopping criteria and extraction of global minimizers (that rely on results from the dual moment theory and commutative algebra), error bounds in some special instances (e.g. optimization over the simplex or the hypercube), application to computing the real solutions to polynomial equations.

www.win.tue.nl/ipa/

Institute for Programming research and Algorithmics

The research school IPA (Institute for Programming Research and Algorithmics) educates researchers in the field of programming research and algorithmics. This field encompasses the study and development of formalisms, methods and techniques to design, analyse, and construct software systems and components. IPA has three main research areas: Algorithmics & Complexity, Formal Methods, and Software Technology & Engineering. Researchers from nine universities (University of Nijmegen, Leiden University, Technische Universiteit Eindhoven, University of Twente, Utrecht University, University of Groningen, Vrije Universiteit Amsterdam, University of Amsterdam, and Delft University), the CWI and Philips Research (Eindhoven) participate in IPA.

In 1997, IPA was formally accredited by the Royal Dutch Academy of Sciences (KNAW). This accreditation was extended in 2002 and 2007. In setting its agenda for 2007 - 2012, IPA chose five focus areas, where we expect important developments in the near future and want to stimulate collaboration. In the focus area:

Beyond Turing we want to explore novel paradigms of computation that incorporate concepts that are no longer adequately modeled by the classical Turing machine such as nonuniformity of memory, adaptivity and mobility.

Algorithms & models for life sciences we wish to apply algorithmic theory and formal models to contribute to the understanding of biological processes, entities and phenomena.

Hybrid systems we want to continue to contribute to the confluence of systems and control theory and computer science in integrated methods for modelling, simulation, analysis, and design of such systems.

Model-driven software engineering we want to study various fundamental aspects of the model-driven approach to software engineering.

Software analysis we want to make progress in the extraction of facts from source code and their analysis, to obtain instruments for measuring the various quality attributes of software.

For descriptions of these areas see `www.win.tue.nl/ipa/about.html`.

Activities in 2009

IPA has two multi-day events per year which focus on current topics, the Lentedagen and the Herfstdagen. In 2009, the Lentedagen were on Algorithms for Data Analysis and Visualization and the Herfstdagen were dedicated to Quantitative Methods for Embedded Systems.

IPA organises Courses on each of its major research fields, Algorithms and Complexity, Formal Methods and Software Technology & Engineering. These courses intend to give an overview of the research of IPA in these fields, and are organized at regular intervals on a cyclic schedule. In 2009, the course on Software Technology & Engineering was held. Additionally, a Course on Principles of Model Checking organized by Joost-Pieter Katoen was offered to the IPA PhD students.

IPA-ASCI Lentedagen on Algorithms for Data Analysis and Visualization *April 15 - 17, Conferentiehotel Guldenberg, Helvoirt*
The Lentedagen are an annual multi-day event, dedicated to a specific theme of current interest to the research community of IPA and ASCI. This year's Lentedagen were dedicated to the subject Algorithms for Data Analysis and Visualization.

In numerous applications one wants to analyze large amounts of data, and discover structure in the data. Sometimes the analysis can be done analytically. In other applications it is useful to visualize the data in a suitable way, in order to help the user discover the underlying structure. In these Spring Days we focussed on the algorithmic aspects related to data analysis and visualization. Algorithmics is one of the core research areas within IPA, while visualization is an important area within ASCI. The Spring Days reflect this, by bringing together researchers from the two research schools and thus creating a varied and exciting program.

More information about the program is available through the archive on the IPA-website: `www.win.tue.nl/ipa/archive/springdays2009/`.

IPA Herfstdagen on Quantitative Methods for Embedded Systems *November 23 - 27, Hotel Astoria, Noordwijk aan Zee*
For the period 2007-2012, IPA has chosen five focus areas where it expects important developments in the field in the near future. Each year the Herfstdagen are dedicated to one of these areas. This year the focus was on the hybrid systems area, and in particular on the quantitative analysis of embedded systems using deterministic, nondeterministic and probabilistic methods.

For actual industrial products, functional correctness of software is usually not enough. Not alone do timing issues become very important when software is embedded in, and interacting with, a physical environment, but in order to stay in business, the performance of the system as a whole has to be higher than that of the competition as well. Finally, in cases where functional correctness cannot be guaranteed at all, the probability of failure has to be minimized. This calls for design and analysis methods that allow quantitative statements about a design. The Falldays this year focussed on deterministic, nondeterministic, probabilistic methods, and their application when trying to get a grip on the quantitative aspects of embedded systems.

The program for the event was composed by Pieter Cuijpers (TU/e), Mariëlle Stoelinga (UT), Jozef Hooman (RU), and Michel Reniers (IPA, TU/e). More information about the program is available through the archive on the IPA-website: `www.win.tue.nl/ipa/archive/falldays2009/`.

IPA Course on Software Technology & Engineering *June 2 - 5, TU/e, Eindhoven*
The Course Software Technology & Engineering, which was hosted by IPA at the Technische Universiteit Eindhoven focussed on the following main areas of research in software technology and engineering:

- Tree-oriented Programming (Jeroen Fokker)
- Engineering grammars using SDF (Mark van den Brand)
- Software Composition (Christian Krause)
- Software Deployment (Eelco Dolstra)
- Workflow Systems (Rinus Plasmeijer)
- Model Transformation & ATL (Ivan Kurtev)
- Query/Views/Transformation (Ivan Kurtev)

In the course lectures were combined with hands-on tool training. The program of the Course on Software Technology & Engeneering was composed by Eelco Dolstra (TUD), Jurriaan Hage (UU), Andres Löh (UU), Michel Reniers (IPA, TU/e), and Eelco Visser (TUD). More information about the program and contents is available through the archive on the IPA-website: `www.win.tue.nl/ipa/archive/stecourse2009/`.

Course on Principles of Model Checking *September 29 - October 2 and October 8+9, UT, Enschede*

A prominent verification technique that has emerged in the last thirty years is model checking, that systematically checks whether a model of a given system satisfies a property such as deadlock freedom, invariants, or request-response. This automated technique for verification and debugging has developed into a mature and widely-used industrial approach with many applications in software and hardware.

This course provided an introduction to the theory of model checking and its theoretical complexity. It introduced transition systems, safety, liveness and fairness properties, as well as omega-regular automata. It then covered the temporal logics LTL, CTL and CTL*, compared them, and treated their model-checking algorithms. Techniques to combat the state-space explosion problem are at the heart of the success of model checking. The course provided an overview of an important class of such techniques, namely abstraction. Finally, model checking of timed automata and of probabilistic automata was considered. The course consisted of lectures and active involvement in exercise classes.

In 2008, the ACM awarded the prestigious Turing Award - the Nobel Prize in Computer Science - to the pioneers of Model Checking: Ed Clarke, Allen Emerson, and Joseph Sifakis. Why? Because model checking has evolved in the last twenty-five years into a widely used verification and debugging technique for both software and hardware.

It is used (and further developed) by companies and institutes such as IBM, Intel, NASA, Cadence, Microsoft, and Siemens, to mention a few, and has culminated in a series of mostly freely downloadable software tools that allow the automated verification of, for instance, C#-programs or combinational hardware circuits.

Subtle errors, for instance due to multi-threading, that remain undiscovered using simulation or peer reviewing can potentially be revealed using model checking. Model checking is thus an effective technique to expose potential design errors and improve software and hardware reliability.

But how does it work, that is, what are its underlying principles? That was exactly the focus of this 6 day course! It is shown that model checking is based on well-known paradigms from automata theory, graph algorithms, logic, and data structures. Its complexity is analyzed using standard techniques from complexity theory.

The course was taught by Joost-Pieter Katoen. Joost-Pieter Katoen is a professor at the RWTH Aachen University (since 2004) and is associated to the University of Twente. He received his PhD in 1996 on a dissertation on true concurrency semantics. His research interests are concurrency theory, model checking, timed and probabilistic systems, and semantics. He co-authored more than 120 journal and conference papers, and recently a comprehensive book (with Christel Baier) on Principles of Model Checking (MIT Press) which provides the basis for this course.

For more information on the contents of the course please visit the webpage of the course: `http://www-i2.informatik.rwth-aachen.de/i2/principles/`.

IPA Ph.D. Defenses in 2009

M.H.G. Verhoef (RU, January 21)
Modeling and Validating Distributed Embedded Real-Time Control Systems
Promotor: prof.dr. F.W. Vaandrager. Co-promotor: dr. J.J.M. Hooman
IPA Dissertation Series 2009-01

M. de Mol (RU, March 4)
Reasoning about Functional Programs: Sparkle, a proof assistant for Clean
Promotor: prof.dr.ir. M.J. Plasmeijer. Co-promotor: dr. M.C.J.D. van Eekelen
IPA Dissertation Series 2009-02

M. Lormans (TUD, January 12)
Managing Requirements Evolution
Promotor: prof.dr. A. van Deursen
IPA Dissertation Series 2009-03

M.P.W.J. van Osch (TU/e, February 10)
Automated Model-based Testing of Hybrid Systems
Promotores: prof.dr. J.C.M. Baeten, prof.dr.ir. J.E. Rooda. Co-promotor: dr. S.P. Luttik
IPA Dissertation Series 2009-04

H. Sozer (UT, January 29)
Architecting Fault-Tolerant Software Systems
Promotor: prof.dr.ir. M. Akşit. Co-promotor: dr.ir. B. Tekinerdoğan
IPA Dissertation Series 2009-05

M.J. van Weerdenburg (TU/e, April 1)
Efficient Rewriting Techniques
Promotores: prof.dr.ir. J.F. Groote, prof.dr. M.G.J. van den Brand. Co-promotor: dr.ir. M.A. Reniers
IPA Dissertation Series 2009-06

H.H. Hansen (VUA, May 14)
Coalgebraic Modelling: Applications in Automata Theory and Modal Logic
Promotor: prof.dr. J.J.M.M. Rutten. Co-promotores: dr. Y. Venema, dr. C.A. Kupke
IPA Dissertation Series 2009-07

A. Mesbah (TUD, June 19)
Analysis and Testing of Ajax-based Single-page Web Applications
Promotor: prof.dr. A. van Deursen
IPA Dissertation Series 2009-08

A.L. Rodriguez Yakushev (UU, May 20)
Towards Getting Generic Programming Ready for Prime Time
Promotores: prof.dr. J.Th. Jeuring, prof.dr. S.D. Swierstra
IPA Dissertation Series 2009-9

K.R. Olmos Joffré (UU, May 27)
Strategies for Context Sensitive Program Transformation
Promotor: prof.dr. S.D. Swierstra. Co-promotor: dr. E. Visser
IPA Dissertation Series 2009-10

J.A.G.M. van den Berg (RU, July 2)
Reasoning about Java programs in PVS using JML
Promotor: prof.dr. B.P.F. Jacobs. Co-promotor: dr.ir. E. Poll
IPA Dissertation Series 2009-11

M.G. Khatib (UT, June 11)
MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems
Promotor: prof.dr. P.H. Hartel
IPA Dissertation Series 2009-12

S.G.M. Cornelissen (TUD, June 23)
Evaluating Dynamic Analysis Techniques for Program Comprehension
Promotor: prof.dr. A. van Deursen. Co-promotor: dr. A.E. Zaidman
IPA Dissertation Series 2009-13

D. Bolzoni (UT, June 25)
Revisiting Anomaly-based Network Intrusion Detection Systems
Promotores: prof.dr. P.H. Hartel, prof.dr. S. Etalle
IPA Dissertation Series 2009-14

H.L. Jonker (TU/e, August 25)
Security Matters: Privacy in Voting and Fairness in Digital Exchange
Promotores: prof.dr. S. Mauw, prof.dr. J.C.M. Baeten. Co-promotor: dr. J. Pan
IPA Dissertation Series 2009-15

M.R. Czenko (UT, June 26)
TuLiP - Reshaping Trust Management

Promotores: prof.dr. P.H. Hartel, prof.dr. S. Etalle
IPA Dissertation Series 2009-16

T. Chen (VUA, September 21)
Clocks, Dice and Processes
Promotores: prof.dr. W.J. Fokkink, prof.dr. J.C. van de Pol
2009-17

C. Kaliszyk (RU, September 3)
Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web
Promotor: prof.dr. J.H. Geuvers. Co-promotor: dr. F. Wiedijk
IPA Dissertation Series 2009-18

R.S.S. O'Connor (RU, October 5)
Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory
Promotor: prof.dr. J.H. Geuvers. Co-promotor: dr. B. Spitters
IPA Dissertation Series 2009-19

B. Ploeger (TU/e, August 27)
Improved Verification Methods for Concurrent Systems
Promotores: prof.dr.ir. J.F. Groote, prof.dr.ir. J.J. van Wijk. Co-promotor: dr.ir. T.A.C. Willemse
IPA Dissertation Series 2009-20

T. Han (UT, September 25)
Diagnosis, Synthesis and Analysis of Probabilistic Models
Promotor: prof.dr.ir. J.-P. Katoen
IPA Dissertation Series 2009-21

R. Li (UL, October 6)
Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis
Promotor: prof.dr. T.H.W. Bäck. Co-promotor: dr. M.T.M. Emmerich
IPA Dissertation Series 2009-22

J.H.P. Kwisthout (UU, October 29)
The Computational Complexity of Probabilistic Networks
Promotores: prof.dr. J. van Leeuwen, prof.dr.ir. L.C. van der Gaag
IPA Dissertation Series 2009-23

T.K. Cocx (UL, December 2)
Algorithmic Tools for Data-Oriented Law Enforcement
Promotor: prof.dr. J.N. Kok. Co-promotor: dr. W.A. Koster
IPA Dissertation Series 2009-24

A.I. Baars (UU, December 9)
Embedded Compilers
Promotor: prof.dr. S.D. Swierstra
IPA Dissertation Series 2009-25

M.A.C. Dekker (UT, December 2)
Flexible Access Control for Dynamic Collaborative Environments
Promotores: prof.dr. P.H. Hartel, prof.dr. S. Etalle
IPA Dissertation Series 2009-26

J.F.J. Laros (UL, December 21)
Metrics and Visualisation for Crime Analysis and Genomics
Promotor: prof.dr. J.N. Kok. Co-promotor: dr. W.A. Koster
IPA Dissertation Series 2009-27

Activities in 2010

IPA is planning several activities for 2010, including the Lentedagen (April 21-23) which will be dedicated to *Multicore computing*, the Course on Algorithms & Complexity (TU/e, Eindhoven), and the Herfstdagen (November). More information on these events will appear on the IPA-website as dates and locations for these events are confirmed.

Addresses

Visiting address
Technische Universiteit Eindhoven
Main Building HG 7.22
Den Dolech 2
5612 AZ Eindhoven
The Netherlands

Postal address
IPA, Fac. of Math. and Comp. Sci.
Technische Universiteit Eindhoven
P.O. Box 513
5600 MB Eindhoven
The Netherlands

tel. (+31)-40-2474124 (IPA Secretariat)
fax (+31)-40-2475361
e-mail `ipa@tue.nl`
url `www.win.tue.nl/ipa/`

School for Information and Knowledge Systems (SIKS) in 2009

Richard Starmans (UU)

Introduction
SIKS is the Dutch Research School for Information and Knowledge Systems. It was founded in 1996 by researchers in the field of Artificial Intelligence, Databases & Information Systems and Software Engineering. Its main concern is research and education in the field of information and computing sciences, more particular in the area of information and knowledge systems.The School currently concentrates on seven focus areas in the IKS field: Agent Technology, Computational Intelligence, Knowledge Representation and Reasoning, Web-based Information Systems, Enterprise Information Systems, Human Computer Interaction, and Data Management, Storage and Retrieval.

SIKS is an interuniversity research school that comprises 12 research groups from 10 universities and CWI. Currently, over 450 researchers are active, including 200 Ph.D.-students. The Vrije Universiteit in Amsterdam is SIKS' administrative university, and as off January 1 2006 Prof.dr. R.J. Wieringa (UT) was appointed scientific director. The office of SIKS is located at Utrecht University. SIKS received its first accreditation by KNAW in 1998 and its first re-accreditation in 2003. In June 2009 SIKS was re-accredited for another period of 6 years.

Activities
We here list the main activities (co-)organized or (co-)financed by SIKS. We distinguish basic courses, advanced courses and other activities (including master classes, workshops, one-day seminars, conferences, summer schools, doctoral consortia and research colloquia)

Basic courses:

“Learning and Reasoning”, May 25-26, 2009, Landgoed Huize Bergen, Vught
Course directors: Dr. A. Ten Teije (VU), Dr. P. Groot (RUN)

“Information Retrieval”, May 27-28, 2009, Landgoed Huize Bergen, Vught
Course director: Prof. dr. ir. Th. Van der Weide (RUN)

“Research methods and methodology for IKS", 25-27 November, 2009, Conference Center Zonheuvel, Doorn
Course directors: Dr. H. Weigand (UvT), Prof.dr. R.J. Wieringa (UT), Prof.dr. H. Akkermans (VUA), Prof.dr. J.-J.Ch. Meyer (UU), Dr. R.J.C.M. Starmans (UU).

“Agent Technology”, December 07-08, 2009, Landgoed Huize Bergen, Vught
Course directors: Prof. dr. C.M. Jonker (TUD), Prof. dr. J.-J.Ch. Meyer (UU), Prof. dr. C. Witteveen (TUD).

“System and Architecture Modeling”, 09-10 December 2009, Landgoed Huize Bergen, Vught
Course directors: Dr. P. van Eck (UT), Prof.dr. W.-J. van den Heuvel (UvT), Dr. M. Jeusveld (UvT)

Advanced courses:

"Human Technology Interaction", January 26-30, 2009, Best Western Hotel, Almen
Course director: Prof. dr. C.M. Jonker (TUD)

"Organisational Principles of IKS", February 16-17 2009, Woudschoten, Zeist.
Course director: Dr. V. Dignum (UU)

"Probabilistic Methods for Entity Resolution and Entity Ranking", April 20-21, 2009, Woudschoten, Zeist
Course directors: Dr. ir. Djoerd Hiemstra (UT), Dr. ir. Maurice van Keulen (UT)

"Summer course on Datamining", 24-27 August 2009, Maastricht
Course director: Dr. E. Smirnov (UM)

"The Semantic Web", September 24-25, 2009, Mitland Hotel, Utrecht
Course directors: Dr. P. Cimiano (TUD), Dr. S. Wang (VU)

AI for Games", October 05-06, 2009, Eindhoven
Course directors: Dr. P. Spronck (OU/UvT)

Process Mining and Data Mining, October 26-27, De Zwarte Doos, Eindhoven
Course directors: Prof.dr. W. van der Aalst (TUE), Prof.dr. P. de Bra (TUE), Dr. T. Calders (TUE), Dr. B. van Dongen (TUE), Dr. M. Pechenizkiy (TUE), Dr. T. Weijters (TUE)

Other activities:

- Conference: Benelearn 09, May 18-19, 2009, Tilburg
- Conference: BNAIC 09, October 29-30, Eindhoven
- Conference: Dutch-Belgian Database Day 2009 (DBDBD), November 30, 2009, Delft
- Conference: DIR 2009, February 02-03, 2009, Enschede
- Conference: "SIKS-Conference on Enterprise Information Systems" (EIS), October 23, 2009 Ravenstein
- Conference: "Second International Conference on Human-Robot Personal Relationships", 11-12 June, Tilburg
- Conference: "International Conference on Advanced Information Systems Engineering (CAiSE 2009)", June 08-12 2009, Amsterdam
- Conference: Business Process Modeling (BPM 2009), September 07-10, Ulm, Germany
- Masterclass: "Evolutionary Agent-Based Policy Analysis ", April 16, 2009, Amsterdam
- NVTI Theory Day 2009, March 20, 2009, Utrecht
- SIKS-PhD Career Day, February 18, Utrecht
- SIKS-day 2009, November 16, 2008, Utrecht
- SIKS-PhD Annual Cultural Event, September 11, 2009, Hilvarenbeek
- Seminar: "Second SIKS/Twente Seminar on Searching and Ranking in Enterprises", 24 June, 2009, Enschede
- Seminar: UU-SIKS Seminars (2 times in 2009)
- Seminar: 2*20 years of communication, organization and IS, December 18,2009, Utrecht
- SIKS-Agent Colloquia (10 times in 2009), Utrecht/Delft/Amsterdam
- SIKS-MICC Colloquia (6 times in 2009), Maastricht
- SIKS-TICC Colloquia (8 times in 2009), Tilburg
- Summerschool EASSS 2009, August 31 - September 04, 2009, Torino (Italy)
- Summerschool SSAIE 2009, June 16 - 19, 2009, Crete (Greece)
- Symposium: Innovating IT; Surfing the Fourth Wave, March 05, 2009, Tilburg
- Symposium: "In Search of Privacy", June, 25, 2009, Nijmegen
- Symposium: Method Engineering in Software Product Management,Sept 9, 2009, Utrecht

- Workshop: Rich Cognitive Models for Policy Design and Simulation, January 12-16, 2009, Leiden
- Workshop: International School on Collective Intelligence and Evolution DECOI 2009 February 23-27, 2009, Leiden
- Workshop: International Workshop "Engineering Societies in the Agents' World" (ESAW 2009), November 18-20, 2009, Utrecht
- Workshop: International Workshop on Value Modeling and Business Ontologies, February 9-10, 2009, Stockholm (Sweden)
- Workshop: International Workshop on Value Modeling and Business Ontologies, December 21-22, 2009, Amsterdam

Ph.D.-defenses in 2009

In 2009 46 researchers successfully defended their Ph.D.-thesis and published their work in the SIKS-dissertation Series.

2009-01
Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models
Promotor: Prof.dr. T.M. Heskes (RUN)
Promotie: 19 January 2009

2009-02
Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques
Promotor: Prof.dr. G. Schreiber (VU)
Promotie: 19 January 2009n

2009-03
Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT
Promotor: Prof.dr. H.J. Van den Herik (UvT)
Promotie: 21 January 2009

2009-04
Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering
Promotores: Prof. dr. E. Proper (RUN), Prof. dr. ir. G.- J. de Vreede, University of Nebraska at Omaha, USA
Copromotor: Dr. P. van Bommel (RUN)
Promotie: 03 March 2009

2009-05
Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality
Promotor: Prof. dr. E. Proper (RUN)
Promotie: 24 April 2009

2009-06
Muhammad Subianto (UU)
Understanding Classification
Promotor: Prof.dr.A.P.J.M. Siebes (UU)
Promotie: 14 January 2009

2009-07
Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion
Promotor: Prof. dr. ir. A. Nijholt (UT)
Copromotor: Dr. M. Poel (UT)
Promotie: 02 April 2009

2009-08
Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments
Promotores: Prof.dr. J. van den Bergh (VU), Prof.dr. A.E. Eiben (VU)
Promotie: 16 April 2009

2009-09
Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems
Promotor: Prof. dr. ir. Th. van der Weide (RUN)
Promotie: 21 April 2009

2009-10
Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications
Promotores: Prof. dr. B.J. Wielinga (UvA), Prof. dr. A.Th. Schreiber (VU)
Promotie: 12 Juni 2009

2009-11
Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web
Promotor: Prof. dr. T. M. van Engers (UVA)
Copromotores: Prof. dr. J. A. P. J. Breuker (UVA), Dr. R. G. F. Winkels (UVA)
Promotie: 25 Juni 2009

2009-12
Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)
Operating Guidelines for Services
Promotores:Prof.dr. Kees van Hee (TUE), prof.dr. Wolfgang Reisig (Humboldt-Universitaet zu Berlin)
Copromotor;prof.dr. Karsten Wolf (Universitaet Rostock)
Promotie: 21 April 2009

2009-13
Steven de Jong (UM)
Fairness in Multi-Agent Systems
Promotor: Prof. dr. H.J. van den Herik (UvT), Prof.dr. E.O. Postma (UvT)
Copromotor: Dr. K. Tuyls (TUE)
Promotie: 04 June 2009

2009-14
Maksym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)
Promotores:Prof.dr. J. Top (VU)
Promotie: 18 june 2009

2009-15
Rinke Hoekstra (UVA)
Ontology Representation - Design Patterns and Ontologies that Make Sense
Promotor: Prof.dr. J.A.P.J. Breuker (UVA)
Copromotores: Prof.dr. T.M. van Engers (UVA), Dr. R.G.F. Winkels (UVA)
Promotie: 18 September 2009

2009-16
Fritz Reul (UvT)
New Architectures in Computer Chess
Promotor: Prof.dr. H.J. van den Herik (UvT)
Copromotor: Dr. J.W.H.M. Uiterwijk (UM)
Promotie: 17 june 2009

2009-17
Laurens van der Maaten (UvT)
Feature Extraction from Visual Data
Promotores: Prof.dr. E.O.Postma (UvT), Prof.dr. H.J. van den Herik (UvT)
Copromotor: Dr. A.G. Lange (RACM)
Promotie: 23 June 2009

2009-18
Fabian Groffen (CWI)
Armada, An Evolving Database System
Promotor: Prof. dr. M.L. Kersten (CWI/UvA)
Copromotor: Dr. S. Manegold (CWI)
Promotie: 10 june 2009

2009-19
Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets
Promotor: Prof.dr. H. La Poutre (CWI/TUE)
Promotie: 02 July 2009

2009-20
Bob van der Vecht (UU)
Adjustable Autonomy: Controling Influences on Decision Making
Promotor: Prof.dr. J.-J. Ch. Meyer (UU)
Copromotor: Dr. F.Dignum (UU)
Promotie: 06 July 2009

2009-21
Stijn Vanderlooy(UM)
Ranking and Reliable Classification
Promotor: Prof.dr. H.J. van den Herik (UvT), Prof.dr. Th.A. de Roos (UM), Prof.dr.rer.nat. E. Hüllermeier, Philipps-University of Marburg, Germany
Promotie: 01 July 2009

2009-22
Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence
Promotor: Prof.dr. P.M.G. Apers (UT)
Copromotor: Dr. D. Hiemstra (UT)
Promotie: 24 June 2009

2009-23
Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment
Promotor: Prof.dr. A.E. Eiben (VU)
Copromotor: Dr. W. Kowalczyk (VU)
Promotie: 08 October 2009

2009-24
Annerieke Heuvelink (VU)
Cognitive Models for Training Simulations
Promotor: Prof. dr. J. Treur (VU)
Copromotor: Dr. K. van den Bosch (TNO), Dr. M. C. A. Klein (VU)
Promotie: 11 September 2009

2009-25
Alex van Ballegooij (CWI)
"RAM: Array Database Management through Relational Mapping"
Promotor: Prof. dr. M.L. Kersten (CWI/UvA))
Copromotor: Prof. dr. A.P. de Vries (TUD)
Promotie: 17 September 2009

2009-26
Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services
Promotores: Prof. Dr. J.-J. Ch. Meyer (UU), Prof. Dr. E. Sonenberg (University of Melbourne)
Copromotor: Dr. F. Dignum (UU)
Promotie: 05 October 2009

2009-27
Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web
Promotores: Prof. dr. E.J.R. Koper (OU), Prof.dr. M. Specht (OU)
Promotie: 18 September 2009

2009-28
Sander Evers (UT)
Sensor Data Management with Probabilistic Models
Promotores:Prof.dr.ir. P.M.G. Apers (UT)
Copromotor:Prof.dr. L. Feng, Tsinghua University (China).
Promotie: 25 September 2009

2009-29
Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications
Promotor: Prof. dr. ir. R. J. Wieringa (UT)
Co-promotor: Prof. dr. M. Reichert (University of Ulm)
Assistent promotor: Dr. ir. M. W. A. Steen (Novay) Promotie: 22 Oktober 2009

2009-30
Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage
Promotor: Prof. dr. M.L. Kersten (CWI/UvA)
Copromotor: Dr. P.A. Boncz (CWI)
Promotie: 11 September 2009

2009-31
Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text
Promotor: Prof. dr. P.W. Adriaans (UVA)
Promotie: 10 September 2009

2009-32
Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors
Promotor: Prof. dr. J.C. van Vliet (VU)
Copromotor: Dr. P. Lago (VU)
Promotie: 05 October 2009

2009-33
Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?
Promotor:Prof. dr. F.M.G. de Jong (UT), Prof. dr. ir. D.A. van Leeuwen (RU)
Promotie: 27 August 2009

2009-34
Inge van de Weerd (UU)
Advancing in Software Product Management: An Incremental Method Engineering Approach
Promotor: Prof.dr. S. Brinkkemper (UU)
Copromotor: Dr. ir. J. Versendaal (UU)
Promotie: 09 September 2009

2009-35
Wouter Koelewijn (UL)
Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling
Promotores: Prof. dr. H.J. van den Herik (UvT/UL), Prof. mr. A.H.J. Schmidt (UL)
Co-promotor: dr. L. Mommers (UL)
Promotie: 04 November 2009

2009-36
Marco Kalz (OU)
Placement Support for Learners in Learning Networks
Promotor: Prof.dr. E.J.R. Koper (OU)
Copromotor: Dr. J.M. van Bruggen (OU)
Promotie: 16 October 2009

2009-37
Hendrik Drachsler (OU)
Navigation Support for Learners in Informal Learning Networks
Promotores: Prof.dr. E.J.R. Koper (OU)
Co-promotor: Dr. H.G.K. Hummel (OU)
Promotie: 16 October 2009

2009-38
Riina Vuorikari (OU)
Tags and self-organisation: a metadata ecology for learning resources in a multilingual context
Promotor: Prof.dr. E.J.R. Koper (OU)
Promotie: 13 November 2009

2009-39
Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)
Service Substitution -- A Behavioral Approach Based on Petri Nets
Promotores: Prof.dr. K. van Hee (TUE), Prof.dr. W. Reisig (Humboldt-Universitaet zu Berlin)
Co-promotor: Prof.dr. Karsten Wolf (Universitaet Rostock)
Promotie: 1 December 2009

2009-40
Stephan Raaijmakers (UvT)
Multinomial Language Learning: Investigations into the Geometry of Language
Promotores: Prof. dr. W. Daelemans (UvT), Prof.dr. A.P.J. van den Bosch (UvT)
Promotie: 1 December 2009

2009-41
Igor Berezhnyy (UvT)
Digital Analysis of Paintings
Promotores: Prof. dr. E.O. Postma (UvT), Prof.dr. H.J. van den Herik (UvT)
Promotie: 7 December 2009

2009-42
Toine Bogers (UvT)
Recommender Systems for Social Bookmarking
Promotor: Prof.dr. A.P.J. van den Bosch (UvT)
Promotie: 8 December 2009

2009-43
Virginia Nunes Leal Franqueira (UT)
Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients
Promotor: Prof. dr. R.J. Wieringa (UT)
Co-promotor: Dr. P. van Eck (UT)
Promotie:13 November 2009

2009-44
Roberto Santana Tapia (UT)
Assessing Business-IT Alignment in Networked Organizations
Promotor: Prof. dr. R.J. Wieringa (UT)
Promotie: 4 December 2009

2009-45
Jilles Vreeken (UU)
Making Pattern Mining Useful
Promotor: Prof. dr.A.P.J.M. Siebes (UU)
Promotie:15 December 2009

2009-46
Loredana Afanasiev (UvA)
Querying XML: Benchmarks and Recursion
Promotor: Prof.dr. M. de Rijke (UvA)
Co-promotor: Dr. M.J. Marx (UvA)
Promotie: 18 December 2009

The Saga of Finite Equational Bases over BCCSP

Taolue Chen[1] and Wan Fokkink[2]

[1] DACS & FMT, University of Twente, The Netherlands
[2] Department of Theoretical Computer Science, Vrije Universiteit Amsterdam, The Netherlands

Abstract. Van Glabbeek (1990) presented the "linear time – branching time spectrum", a lattice of behavioral semantics over labeled transition systems ordered by inclusion. He studied these semantics in the setting of the basic process algebra BCCSP, and tried to give finite complete axiomatizations for them. Obtaining such axiomatizations in concurrency theory often turns out to be difficult, which had raised a host of open questions that were the subject of intensive research in recent years. All of these questions are settled over BCCSP now, either positively by giving a finite complete axiomatization, or negatively by proving that such an axiomatization does not exist. This essay reports on these results.

1 Introduction

Labeled transition systems (LTSs) constitute a fundamental model of concurrent computation which is widely used in light of its flexibility and applicability. They model processes by explicitly describing their states and transitions from state to state, together with the actions that produce them. Several notions of behavioral equivalences have been proposed, with the aim to identify those states of LTSs that afford the same observations. The lack of consensus on what constitutes an appropriate notion of observable behavior for reactive systems has led to a large number of proposals for behavioral equivalences for concurrent processes.

Van Glabbeek [22,23] presented the linear time – branching time spectrum of behavioral *preorders* and *equivalences* for finitely branching, concrete, sequential processes, which is a lattice of known behavioral preorders and equivalences over LTSs, ordered by inclusion. The semantics in the spectrum are based on *simulation* notions or on *decorated traces*. **Fig**. 1 depicts the linear time – branching time spectrum[3], where an arrow from one semantics to another means that the source of the arrow is finer than the target.

To give further insight into the identifications made by the respective behavioral equivalences in his spectrum, van Glabbeek [22,23] studied them in the setting of the process algebra BCCSP, which contains only the basic process algebraic operators from CCS and CSP, but is sufficiently powerful to express all finite synchronization trees. In particular, he associated with every behavioral equivalence in his spectrum a sound equational *axiomatization*, i.e., a collection of equations of behaviorally equivalent BCCSP terms. Most of the axiomatizations were also shown to be "complete" in the sense that whenever two *closed* BCCSP terms (i.e., terms containing no occurrence of variables) are behaviorally equivalent, then the axiomatization admits a derivation in equational logic of the corresponding equation.

In general, axiomatizations arise from the desire of isolating the features that are common to a collection of algebraic structures, namely, their semantics models. One requires that a set of axioms is *sound* (i.e., if two behaviors can be equated, then they are semantically related), and one desires that it is *complete* (i.e., if two behaviors are semantically related, then they can be equated). Having defined a semantic model for a process algebra in terms of LTSs, it is natural to

[3] Note that the *completed simulation* and *impossible futures* semantics were missing in the original spectrum [22,23], but are included here.

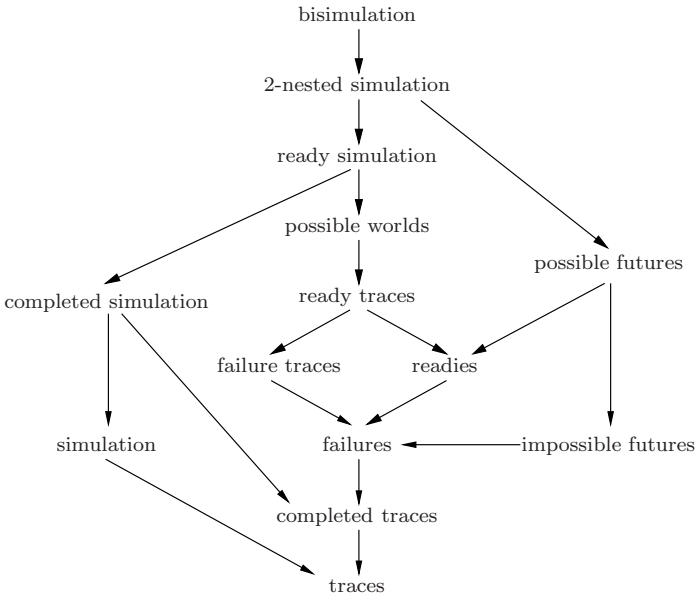


Fig. 1. The linear time – branching time spectrum

study its *(in)equational theory*, that is, the collection of (in)equations that are valid in the given model. The key questions here are:

- Are there reasonably informative *sound* and *complete* axiomatizations for the chosen semantic model?
- Does the algebra of LTSs modulo the chosen notion of behavioral semantics afford a *finite* (in)equational axiomatization?

A sound and complete axiomatization of a behavioral congruence (resp. precongruence) yields a purely syntactic characterization, independent of LTSs and of the actual details of the definition of the chosen behavioral equivalence (resp. preorder), of the semantics of the process algebra. This bridge between syntax and semantics plays an important role in both the theory and the practice of process algebras. From the theoretical perspective, complete axiomatizations of behavioral preorders or equivalences capture the essence of different notions of semantics for processes in terms of a basic collection of identities, and this often allows one to compare semantics which might have been defined in very different styles and frameworks. From the point of view of practice, these proof systems can be used to perform system verifications in a purely syntactic way using general purpose theorem provers or proof checkers, and form the basis of purpose-built axiomatic verification tools like, e.g. PAM [15]. Hence a positive answer to the first basic question raised above is therefore not just theoretically pleasing, but has potential practical applications.

In literature, different forms of completeness are often considered. For BCCSP, an axiomatization is said to be *complete* if any two behaviorally equivalent BCCSP terms (not just the closed ones) can be equated; completeness for *closed* terms only we shall henceforth refer to as *ground-completeness*. The notion of completeness of an axiomatization also relates the *proof-theoretic* notion of derivability using the rules of equational logic with the *model-theoretic* one of "validity in a model". From a proof-theoretic perspective, a useful property of a ground-complete axiomatization is that whenever all closed instances of an equation can be derived from it, then the

equation itself can also be derived from it; this property is generally referred to as *ω-completeness*. For theorem proving applications, it is particularly convenient if an axiomatization is ω-complete, because it means that proofs by (structural) induction can be avoided in favor of purely equational reasoning; see [14]. In [12] it was argued that ω-completeness is desirable for the partial evaluation of programs. It turns out that completeness and ω-completeness are closely related properties of an axiomatization. Indeed, in the setting of BCCSP, it is not hard to show that any complete axiomatization is also ω-complete. Conversely any ω-complete axiomatization is, by definition, complete.

In universal algebra, a complete axiomatization is referred to as a *basis* for the equational theory of the algebra it axiomatizes. The existence of a finite basis for an equational theory is a classic topic of study in universal algebra (see, e.g., [18]), dating back to Lyndon [16]. Murskiĭ [21] proved that "almost all" *finite* algebras (namely all quasi-primal ones) are finitely based, while in [20] he presented an example of a three-element algebra that has no finite basis. Henkin [13] showed that the algebra of naturals with addition and multiplication is finitely based, while Gurevič [11] showed that after adding exponentiation the algebra is no longer finitely based. McKenzie [17] settled Tarski's Finite Basis Problem in the negative, by showing that the general question whether a finite algebra is finitely based is undecidable.

Process algebra, as a branch of universal algebra and equational logic, naturally features the study of results pertaining to the existence or non-existence of finite bases for algebras modulo given semantics. Many of the existing axiomatizations of behavioral semantics over expressive process algebras studied in concurrency theory are powerful enough to prove all of the valid equalities between *closed* terms; they are ground-complete, but *not* ω-complete. In fact, obtaining ω-complete axiomatizations in concurrency theory often turns out to be a difficult question, even in the setting of simple languages like BCCSP. This has raised a host of open questions that have been the subject of intensive investigation by process algebraists in recent years. Fortunately, theses questions are finally settled for all the semantics in the linear time – branching time spectrum in the setting of BCCSP, either positively by giving a finite sound and ground-complete axiomatization that is ω-complete, or negatively by proving that such a finite basis for the equational theory does not exist. In this essay, we report on these positive and negative results. We hope that this will contribute to their dissemination in our research community and stimulate further investigations.

This essay is organized as follows: Section 2 gives some preliminaries, in particular, the linear time – branching time spectrum and process algebra BCCSP; Section 3 reports on results on axiomatizations of behavioral semantics in the specturm over the language BCCSP; Section 4 gives a summary.

2 Preliminaries

2.1 The linear time – branching time spectrum

A *labeled transition system* consists of a set of states S, with typical element s, and a transition relation $\rightarrow \subseteq S \times L \times S$, where L is a set of labels ranged over by a. We write $s \xrightarrow{a} s'$ if the triple (s, a, s') is an element of $\rightarrow$. The set $\mathcal{I}(s)$ consists of those labels a for which there exists s' such that $s \xrightarrow{a} s'$. Let $a_1 \cdots a_k$ be a sequence of labels; we write $s \xrightarrow{a_1 \cdots a_k} s'$ if there are states $s_0, \ldots, s_k$ such that $s = s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_k} s_k = s'$.

First we define six semantics based on decorated versions of execution traces.

Definition 1 (Decorated Traces). *Assume a labeled transition system.*

- *A sequence $a_1 \cdots a_k$, with $k \geq 0$, is a* trace *of a state s if there is a state s' such that $s \xrightarrow{a_1 \cdots a_k} s'$. It is a* completed trace *of s if moreover $\mathcal{I}(s') = \emptyset$.*

– *A pair* $(a_1 \cdots a_k, B)$*, with* $k \geq 0$ *and* $B \subseteq A$*, is a* ready pair *of a state* s_0 *if there is a sequence of transitions* $s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_k} s_k$ *with* $\mathcal{I}(s_k) = B$*. It is a* failure pair *of* s_0 *if there is such a sequence with* $\mathcal{I}(s_k) \cap B = \emptyset$*.*
– *A sequence* $B_0 a_1 B_1 \ldots a_k B_k$*, with* $k \geq 0$ *and* $B_0, \ldots, B_k \subseteq A$*, is a* ready trace *of a state* s_0 *if there is a sequence of transitions* $s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_k} s_k$ *with* $\mathcal{I}(s_i) = B_i$ *for* $i = 0, \ldots, k$*. It is a* failure trace *of* s_0 *if there is such a sequence with* $\mathcal{I}(s_i) \cap B_i = \emptyset$ *for* $i = 0, \ldots, k$*.*

We write $s \precsim_\square s'$ *with* $\square \in \{\mathrm{T}, \mathrm{CT}, \mathrm{R}, \mathrm{F}, \mathrm{RT}, \mathrm{FT}\}$ *if the traces, completed traces, ready pairs, failure pairs, ready traces, or failure traces, respectively, of* s *are included in those of* s'*.*

Next we define five semantics based on simulation.

Definition 2 (Simulations). *Assume an A-labeled transition system.*

– *A binary relation* $\mathcal{R}$ *on states is a* simulation *if* $s_0 \mathcal{R} s_1$ *and* $s_0 \xrightarrow{a} s_0'$ *imply* $s_1 \xrightarrow{a} s_1'$ *for some state* s_1' *with* $s_0' \mathcal{R} s_1'$*.*
– *A simulation* $\mathcal{R}$ *is a* completed simulation *if* $s_0 \mathcal{R} s_1$ *and* $\mathcal{I}(s_0) = \emptyset$ *imply* $\mathcal{I}(s_1) = \emptyset$*.*
– *A simulation* $\mathcal{R}$ *is a* ready simulation *if* $s_0 \mathcal{R} s_1$ *and* $a \notin \mathcal{I}(s_0)$ *imply* $a \notin \mathcal{I}(s_1)$*.*
– *A simulation* $\mathcal{R}$ *is a* 2-nested simulation *if* $\mathcal{R}^{-1}$ *is included in a simulation.*
– *A* bisimulation *is a symmetric simulation.*

We write $s \precsim_\square s'$ *with* $\square \in \{\mathrm{S}, \mathrm{CS}, \mathrm{RS}, \mathrm{2N}\}$ *if there exists a simulation, completed simulation, ready simulation or 2-nested simulation* $\mathcal{R}$*, respectively, with* $s \mathcal{R} s'$*. We write* $s \leftrightarroweq s'$ *if there exists a bisimulation* $\mathcal{R}$ *with* $s \mathcal{R} s'$*.*

Finally, we define three semantics based on *(im)possible futures* and on *possible worlds.*

Definition 3 ((Im)Possible futures/worlds). *Assume an A-labeled transition system.*

– *A pair* $(a_1 \cdots a_k, X)$*, with* $n \geq 0$ *and* $X \subseteq A^*$ *is a* possible future *of a state* s_0 *if there is a sequence of transitions* $s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_k} s_k$ *where* X *is the set of traces of* s_k*.*
– *A pair* $(a_1 \cdots a_k, X)$*, with* $k \geq 0$ *and* $X \subseteq A^*$*, is an* impossible future *of a state* s_0 *if there is a sequence of transitions* $s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_k} s_k$ *for some state* s_k *with* $\mathcal{T}(s_k) \cap X = \emptyset$*.*
– *A state* s *is* deterministic *if for each* $a \in \mathcal{I}(s)$ *there is exactly one state* s_0 *such that* $s \xrightarrow{a} s_0$*, and moreover* s_0 *is deterministic. A state* s *is a* possible world *of a state* s_0 *if* s *is deterministic and* $s \mathcal{R} s_0$ *for some ready simulation* $\mathcal{R}$*.*

We write $s \precsim_\square s'$ *with* $\square \in \{\mathrm{PF}, \mathrm{IF}, \mathrm{PW}\}$ *if the possible futures, impossible futures or the possible worlds, respectively, of* s *are included in those of* s'*.*

In general, we write $s \simeq_\square s'$ if both $s \precsim_\square s'$ and $s' \precsim_\square s$ for $\square \in \{\mathrm{T}, \mathrm{CT}, \mathrm{R}, \mathrm{F}, \mathrm{RT}, \mathrm{FT}, \mathrm{S}, \mathrm{CS}, \mathrm{RS}, \mathrm{2N}, \mathrm{PF}, \mathrm{IF}, \mathrm{PW}\}$.

2.2 BCCSP

BCCSP is a basic process algebra for expressing finite process behavior. Its signature consists of the constant $\mathbf{0}$, the binary operator $\_+\_$, and unary prefix operators $a\_$, where a ranges over a nonempty set A of actions, called the *alphabet*, with typical elements a, b, c. Intuitively, closed BCCSP terms, denoted p, q, r, represent finite process behaviors, where $\mathbf{0}$ does not exhibit any behavior, $p + q$ offers a choice between the behaviors of p and q, and ap executes action a to transform into p. This intuition is captured by the transition rules below, in which a ranges over A. They give rise to A-labeled transitions between BCCSP terms.

$$\frac{}{ax \xrightarrow{a} x} \qquad \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}$$

We also assume a countably infinite set V of variables; x, y, z denote elements of V, and X, Y, Z denote finite subsets of V. Open BCCSP terms, which may contain variables from V, are denoted t, u, v, w. A term t is called a *prefix* if $t = at'$ for some $a \in A$ and for some term t'.

The preorders $\precsim$ in the linear time – branching time spectrum are all *precongruences* with respect to BCCSP, meaning that $p_1 \precsim q_1$ and $p_2 \precsim q_2$ imply $p_1 + p_2 \precsim q_1 + q_2$ and $ap_1 \precsim aq_1$ for $a \in A$. Likewise, the equivalences in the spectrum are all *congruences* with respect to BCCSP.

A (closed) substitution, denoted ρ, σ, τ, maps variables in V to (closed) BCCSP terms. For open BCCSP terms t and u, and a preorder $\precsim$ (or equivalence $\simeq$) on closed BCCSP terms, we define $t \precsim u$ (or $t \simeq u$) if $\rho(t) \precsim \rho(u)$ (resp. $\rho(t) \simeq \rho(u)$) for all closed substitutions ρ.

An *equational axiomatization* is a collection of equations $t \approx u$, and an *inequational axiomatization* is a collection of inequations $t \preccurlyeq u$. The (in)equations in an axiomatization E are referred to as *axioms*. If E is an equational axiomatization, we write $E \vdash t \approx u$ if the equation $t \approx u$ is derivable from the axioms in E using the rules of equational logic (reflexivity, symmetry, transitivity, substitution, and closure under BCCSP contexts):

$$\frac{}{t \approx t} \quad \frac{t \approx u}{u \approx t} \quad \frac{t \approx u \;\; u \approx v}{t \approx v} \quad \frac{t \approx u}{\rho(t) \approx \rho(u)} \quad \frac{t \approx u}{at \approx au} \quad \frac{t_1 \approx u_1 \;\; t_2 \approx u_2}{t_1 + t_2 \approx u_1 + u_2}$$

For the derivation of an inequation $t \preccurlyeq u$ from an inequational axiomatization E of inequations, denoted $E \vdash t \preccurlyeq u$, the second rule, for symmetry, is omitted.

An axiomatization E is *sound* modulo $\precsim$ (or $\simeq$) if for any open BCCSP terms t, u, from $E \vdash t \preccurlyeq u$ (or $E \vdash t \approx u$) it follows that $\rho(t) \precsim \rho(u)$ (or $\rho(t) \simeq \rho(u)$) for all closed substitutions ρ. E is *ground-complete* modulo $\precsim$ (or $\simeq$) if $p \precsim q$ (or $p \simeq q$) implies $E \vdash p \preccurlyeq q$ (or $E \vdash p \approx q$), for all closed BCCSP terms p and q; it is *complete* modulo $\precsim$ (or $\simeq$) IF $p \precsim q$ (or $p \simeq q$) implies $E \vdash p \preccurlyeq q$ (or $E \vdash p \approx q$) for *all* BCCSP terms p and q. Finally, E is *ω-complete* if for any open BCCSP terms t and u with $E \vdash \rho(t) \preccurlyeq \rho(u)$ (or $E \vdash \rho(t) \approx \rho(u)$) for all closed substitutions ρ, we have $E \vdash t \preccurlyeq u$ (or $E \vdash t \approx u$). A preorder $\precsim$ or an equivalence $\simeq$ is said to be *finitely based* if there exists a finite axiomatization E that is sound and complete modulo $\precsim$ or $\simeq$.

Let $\{t_1, \ldots, t_n\}$ be a finite set of terms; we use summation $\sum\{t_1, \ldots, t_n\}$ to denote $t_1 + \cdots + t_n$, adopting the convention that the summation of the empty set denotes $\mathbf{0}$. Furthermore, we write $a^n t$ to denote the term obtained from t by prefixing it n times with a, i.e., $a^0 t = t$ and $a^{n+1} t = a(a^n t)$. When writing terms, we adopt as binding convention that $\_ + \_$ and summation bind weaker than $a\_$. With abuse of notation, we often let a finite set X denote the term $\sum_{x \in X} x$.

3 Positive and Negative Results for BCCSP

In this section we will survey positive and negative results on the existence of a finite basis for the equational (resp. inequational) theories of BCCSP modulo the equivalences (resp. preorders) in the spectrum above. The axiomatizations that we will present for the different semantics in the spectrum were mostly taken from [23]. Note that in case of an infinite alphabet, occurrences of action names in axioms are interpreted as variables, as otherwise most of the axiomatizations mentioned in this introduction would be infinite.

As the readers might see, except for bisimulation, the semantics considered in this essay have a natural formulation as a *preorder* relation, while the corresponding *equivalence* is defined as the kernel of the preorder. Recently, Aceto, Fokkink and Ingolfsdottir [1] gave an algorithm that, given a sound and ground-complete axiomatization for BCCSP modulo a preorder no finer than ready simulation, produces a sound and ground-complete axiomatization for BCCSP modulo the corresponding equivalence. Moreover, if the original axiomatization for the preorder is ω-complete, then so is the resulting axiomatization for the equivalence. So for the positive result regarding a

semantics, the stronger result is obtained by considering the preorder. (The result for the equivalence can be read as a corollary.) On the other hand, the negative results become more general if they are proved for the equivalence relations. We note that, as we will see soon, the condition "no finer than ready simulation" of the algorithm is essential. This also suggests that, for 2-nested simulation, possible futures and impossible futures semantics which fail this condition, the results for preorders and equivalences must be stated separately.

3.1 Bisimulation

The core axioms in the following table are sound and ground-complete for BCCSP modulo bisimulation. Moller [19] proved using normal forms that this axiomatization is ω-complete; Groote provided an alternative proof of this result in [10] using his inverted substitutions technique.

A1	$x + y$	$\approx y + x$
A2	$(x + y) + z$	$\approx x + (y + z)$
A3	$x + x$	$\approx x$
A6	$x + \mathbf{0}$	$\approx x$

Table 1. The axioms for bisimulation.

3.2 2-Nested Simulation

Chen and Fokkink [5] proved that BCCSP modulo any semantics no coarser than impossible futures equivalence and no finer than 2-nested simulation equivalence does not possess a finite sound and ground-complete axiomatization. (Note that possible futures equivalence is within this semantics range.) This improves a result due to Aceto, Fokkink, van Glabbeek and Ingolfsdottir [2] which covers 2-nested simulation and possible future equivalences only. The cornerstone for this negative result is the following infinite family of equations: for $m \geq 0$,

$$aa^{2m}\mathbf{0} + a(a^m\mathbf{0} + a^{2m}\mathbf{0}) \;\approx\; a(a^m\mathbf{0} + a^{2m}\mathbf{0}) \;.$$

These equations are sound modulo 2-nested simulation equivalence. However, any finite axiomatization for BCCSP which is sound modulo impossible futures equivalence cannot derive all of them.

For 2-nested simulation preorder, Aceto, Fokkink, van Glabbeek and Ingolfsdottir [2] proved that it lacks a finite, sound, ground-complete axiomatization as well. The infinite family of inequations that they used to prove this negative result is, for $m \geq 0$,

$$a^{2m} \preccurlyeq a^{2m} + a^m \;.$$

These inequations are sound modulo 2-nested simulation preorder. However, any finite axiomatization for BCCSP which is sound modulo 2-nested simulation preorder cannot derive all of them.

3.3 Possible Futures

Aceto, Fokkink, van Glabbeek and Ingolfsdottir [2] proved that BCCSP modulo possible futures preorder does not possess a finite, sound, ground-complete axiomatization. The infinite family of inequations that they used to prove this negative result is, for $m \geq 0$,

$$a(a^m + a^{2m}) + aa^{3m} \preccurlyeq a(a^m + a^{3m}) + aa^{2m} \;.$$

These equations are sound modulo possible futures preorder. However, any finite axiomatization for BCCSP which is sound modulo possible futures preorder cannot derive all of them.

As for possible futures equivalence, the aforementioned negative result (cf. the previous section) implies that BCCSP modulo possible futures equivalence does not possess a finite, sound, ground-complete axiomatization.

3.4 Impossible Futures

Chen and Fokkink [5] provided a sound and ground-complete axiomatization for BCCSP modulo impossible futures *preorder*. This is obtained by extending the four core axioms with two extra axioms:

$$a(x+y) \preccurlyeq ax + ay \ ,$$
$$a(x+y) + ax + a(y+z) \approx ax + a(y+z) \ .$$

This result is quite surprising since the aforementioned negative result (cf. the section on 2-nested simulation) implies that BCCSP modulo impossible futures *equivalence* does *not* have a finite, sound, ground-complete axiomatization.

When A is infinite, Groote's technique of inverted substitutions can be applied to show that the aforementioned ground-complete axiomatization is ω-complete. When A is finite, Chen and Fokkink [5] proved that BCCSP modulo impossible futures preorder does not possess a finite sound and ω-complete axiomatization. The infinite family of inequations that they used to prove this negative result is defined in the following way: in case of $|A| = 1$, for $m \geq 0$,

$$a^m x \ \preccurlyeq \ a^m x + x \ ,$$

while in case of $2 \leq |A| < \infty$,

$$a(a^m x) + a(a^m x + x) + \sum_{b \in A} a(a^m x + a^m b\mathbf{0}) \preccurlyeq a(a^m x + x) + \sum_{b \in A} a(a^m x + a^m b\mathbf{0}) \ .$$

To the best of our knowledge, impossible futures semantics is the first (and up to now, the only) example that affords a finite, ground-complete axiomatization for BCCSP modulo the *preorder*, while missing a finite, ground-complete axiomatization for BCCSP modulo the *equivalence*. This fact suggests that, for instance, if one wants to show $p \simeq_{\mathrm{IF}} q$ in general, one has to resort to deriving $p \precsim_{\mathrm{IF}} q$ and $q \precsim_{\mathrm{IF}} p$ separately, instead of proving it directly.

It is worth pointing out that this result does not contradict the algorithm [1] mentioned at the beginning of this section, since that algorithm only applies to semantics that are at least as coarse as *ready simulation semantics*. Since impossible futures semantics is incomparable to ready simulation semantics, it falls outside the scope of [1]. Interestingly, this result yields that no such algorithm exists for certain semantics incomparable with (or finer than) ready simulation.

3.5 Ready Simulation

Van Glabbeek gave a finite axiomatization that is sound and ground-complete for BCCSP modulo ready simulation preorder. It consists of four core axioms and the following axiom:

$$ax \preccurlyeq ax + ay \ ,$$

where a ranges over A. For ready simulation equivalence, he only presented a conditional axiom: $\mathcal{I}(x) = \mathcal{I}(y) \Rightarrow a(x+y) \approx a(x+y) + ay$. Blom, Fokkink and Nain [3] showed that a sound and

ground-complete finite equational axiomatization for BCCSP modulo ready simulation equivalence does exist. It can be obtained by extending the four core axioms with

$$a(bx + by + z) \approx a(bx + by + z) + a(bx + z) \ ,$$

where a, b range over A.

When A is infinite, Groote's technique of inverted substitutions can be applied to show that these axiomatizations are ω-complete. When A is finite, Chen, Fokkink, and Nain [7] proved that BCCSP modulo ready simulation equivalence does not possess a finite sound and ω-complete axiomatization. (Hence neither does ready simulation preorder.) The infinite family of equations that they used to prove this negative result is, for $n > 0$,

$$a^n x + a^n \mathbf{0} + \sum_{b \in A} a^n(x + b\mathbf{0}) \approx a^n \mathbf{0} + \sum_{b \in A} a^n(x + b\mathbf{0}) \ .$$

These equations are sound modulo ready simulation equivalence. However, they cannot be derived from any finite axiomatization for BCCSP sound which is sound modulo ready simulation equivalence. When $|A| = 1$, ready simulation equivalence (resp. preorder) coincides with completed trace equivalence (resp. preorder), and we will see that in this case a finite basis does exist.

3.6 Completed Simulation

Van Glabbeek gave a finite axiomatization that is sound and ground-complete for BCCSP modulo completed simulation preorder. It consists of four core axioms together with

$$ax \preccurlyeq ax + y \ ,$$

where a range over A. It follows that

$$a(bx + y + z) \approx a(bx + y + z) + a(bx + z)$$

suffices to obtain a finite, sound and ground-complete axiomatization for the completed simulation equivalence.

When $|A| > 1$, Chen, Fokkink and Nain [7] proved that BCCSP modulo completed simulation equivalence does not possess a finite sound and ω-complete axiomatization. (Hence neither does ready simulation preorder.) The infinite family of equations that they used to prove this negative result is, for $n \geq 0$,

$$a^n x + a^n \mathbf{0} + a^n(x + y) \approx a^n \mathbf{0} + a^n(x + y) \ .$$

These equations are sound modulo completed simulation equivalence. However, they cannot be derived from any finite axiomatization for BCCSP which is sound modulo completed simulation equivalence. When $|A| = 1$, completed simulation equivalence (resp. preorder) coincides with completed trace equivalence (resp. preorder), and we will see that in this case a finite basis does exist.

It is worth mentioning that completed simulation is the only semantics in the linear time – branching time spectrum that in case of an *infinite* alphabet has a finite sound and ground-complete axiomatization for BCCSP, but no finite ω-complete axiomatization.

3.7 Simulation

A sound and ground-complete axiomatization for BCCSP modulo simulation preorder is obtained by extending the four core axioms with

$$x \preccurlyeq x + y \ .$$

It follows that the following axiom

$$a(x+y) \approx a(x+y) + ay \ ,$$

suffices to obtain a sound and ground-complete axiomatization for BCCSP modulo simulation equivalence. When A is infinite, Groote's technique of inverted substitutions can be applied to show that these axiomatizations are ω-complete. When $1 < |A| < \infty$, Chen and Fokkink [4] proved that BCCSP modulo simulation equivalence does not possess a finite sound and ω-complete axiomatization. (Hence neither does simulation preorder.) The infinite family of equations that they used to prove this negative result is, for $n \geq 0$,

$$a(x+\Psi_n) + \sum_{\theta \in A^n} a(x+\Psi_n^\theta) + a\Phi_n \approx \sum_{\theta \in A^n} a(x+\Psi_n^\theta) + a\Phi_n \ .$$

Here the Φ_n are defined inductively as follows:

$$\begin{cases} \Phi_0 & = \mathbf{0} \\ \Phi_{n+1} & = \sum_{b \in A} b\Phi_n \end{cases}$$

Moreover, the Ψ_n and Ψ_n^θ are defined by:

$$\Psi_n = \textstyle\sum_{b_1 \cdots b_n \in A^n} b_1 \cdots b_n \mathbf{0}$$

$$\Psi_n^\theta = \textstyle\sum_{b_1 \cdots b_n \in A^n \setminus \{\theta\}} b_1 \cdots b_n \mathbf{0} \qquad \text{for } \theta \in A^n \ .$$

These equations are sound modulo simulation equivalence. However, any finite axiomatization for BCCSP which is sound modulo simulation equivalence cannot derive all of them. When $|A| = 1$, simulation equivalence (resp. preorder) coincides with trace equivalence (resp. preorder), and we will see that in this case a finite basis does exist.

3.8 Possible Worlds

A sound and ground-complete axiomatization for BCCSP modulo possible worlds preorder is obtained by extending the four core axioms with

$$ax \preccurlyeq ax + ay \ ,$$
$$a(bx + by + z) \preccurlyeq a(bx + z) + a(by + z) \ .$$

It follows that the following axiom

$$a(bx + by + z) \approx a(bx + z) + a(by + z)$$

suffices to obtain a sound, ground-complete axiomatization for BCCSP modulo possible worlds equivalence.

When A is infinite, Groote's technique of inverted substitutions can be applied to show that these axiomatizations are ω-complete. Fokkink and Nain [8] showed that when $1 < |A| < \infty$, BCCSP modulo any semantics no coarser than ready pair equivalence and no finer than possible worlds equivalence does not possess a finite basis. (Note that ready traces equivalence is within this semantic range.) Their proof of this negative result, which uses cover equations and applies

the compactness theorem to the equational theory for terms of depth 1, is based on the following infinite family of equations:

$$a(\sum_{i=1}^{|A|-1} x_i) + \sum_{j=1}^{|A|-1} a(\sum_{i=1}^{j-1} x_i + \sum_{i=j+1}^{n} x_i) + \sum_{j=|A|}^{n} a(\sum_{i=1}^{|A|-1} x_i + x_j + y_j) \approx$$
$$a(\sum_{i=1}^{|A|-1} x_i) + \sum_{j=1}^{|A|-1} a(\sum_{i=1}^{j-1} x_i + \sum_{i=j+1}^{n} x_i) + \sum_{j=|A|}^{n} a(\sum_{i=1}^{|A|-1} x_i + x_j + y_j) + a(\sum_{i=1}^{n} x_i).$$

These equations are sound modulo possible worlds equivalence for $n \geq |A|$. However, any finite axiomatization that is sound for BCCSP modulo ready pairs equivalence cannot derive them all. When $|A| = 1$, possible worlds equivalence (resp. preorder) coincides with completed trace equivalence (resp. preorder), and we will see that in this case a finite basis does exist.

3.9 Ready Traces

Van Glabbeek presented a conditional axiom for ready trace equivalence: $\mathcal{I}(x) = \mathcal{I}(y) \Rightarrow ax + ay \approx a(x + y)$. Blom, Fokkink and Nain [3] showed that when A is finite, a sound and ground-complete finite equational axiomatization for BCCSP modulo ready traces exists. For the ready traces preorder, it can be obtained by extending the four core axioms with

$$ax \preccurlyeq ax + ay \ ,$$
$$a(\sum_{i=1}^{|A|}(b_i x_i + b_i y_i) + z) \preccurlyeq a(\sum_{i=1}^{|A|} b_i x_i + z) + a(\sum_{i=1}^{|A|} b_i y_i + z) \ .$$

It follows that the following axiom

$$a(\sum_{i=1}^{|A|}(b_i x_i + b_i y_i) + z) \approx a(\sum_{i=1}^{|A|} b_i x_i + z) + a(\sum_{i=1}^{|A|} b_i y_i + z)$$

suffices to obtain a finite, sound and ground-complete axiomatization for BCCSP modulo ready traces equivalence.

When A is infinite, Blom, Fokkink and Nain showed using the compactness theorem that a finite sound and ground-complete axiomatization does not exist. Their proof is based on the following equations, for $n > 0$:

$$a(\sum_{i=1}^{n}(b_i c\mathbf{0} + b_i d\mathbf{0})) \approx a(\sum_{i=1}^{n} b_i c\mathbf{0}) + a(\sum_{i=1}^{n} b_i d\mathbf{0}) \ .$$

When $1 < |A| < \infty$, the aforementioned negative result from [8] (cf. the section on possible worlds) implies that BCCSP modulo ready traces does not possess a finite basis. When $|A| = 1$, ready trace equivalence (resp. preorder) coincides with completed trace equivalence (resp. preorder), and we will see that in this case a finite ω-complete axiomatization does exist.

3.10 Failure Traces

Van Glabbeek presented a conditional axiom for failure traces (the same one as for ready traces). Blom, Fokkink and Nain [3] showed using normal forms that a sound and ground-complete finite

equational axiomatization for BCCSP modulo failure traces equivalence exists. For failure traces preorder, it is obtained by extending the four core axioms with

$$ax \preccurlyeq ax + ay \ ,$$
$$a(x + y) \preccurlyeq ax + ay \ .$$

It follows that the following axioms

$$a(bx + by + z) \approx a(bx + by + z) + a(by + z) \ ,$$
$$ax + ay \approx ax + ay + a(x + y) \ .$$

suffice to obtain a finite, sound and ground-complete axiomatization for BCCSP modulo failure traces equivalence.

When A is infinite, Groote's technique of inverted substitutions can be applied to show that these axiomatizations are ω-complete. When $1 < |A| < \infty$, Chen, Fokkink and Luttik [6] showed that BCCSP modulo failure traces equivalence does not possess a finite sound and ω-complete axiomatization. (Hence neither does failure traces preorder.) The infinite family of equations that they used to prove this negative result is, for $n \geq 0$,

$$a^{n+1}x \ + \ a(a^n x \ + \ x) \ + \ a \sum_{b \in A \setminus \{a\}} a^n(b\mathbf{0} \ + \ x) \quad \approx \quad a(a^n x \ + \ x) \ + \ a \sum_{b \in A \setminus \{a\}} a^n(b\mathbf{0} \ + \ x) \ .$$

These equations are sound modulo failure traces equivalence. However, any finite axiomatization for BCCSP which is sound modulo failure traces equivalence cannot derive all of them. When $|A| = 1$, failure trace equivalence (resp. preorder) coincides with completed trace equivalence (resp. preorder), and we will see that in this case a finite basis does exist.

3.11 Ready Pairs

A sound and ground-complete axiomatization for BCCSP modulo ready pairs preorder is obtained by extending the four core axioms with

$$ax \preccurlyeq ax + ay \ ,$$
$$a(bx + by + z) \preccurlyeq a(bx + z) + a(by + w) \ .$$

It follows that the following axiom

$$a(bx + z) + a(by + w) \approx a(bx + by + z) + a(by + w)$$

suffices to obtain a finite sound and ground-complete axiomatization for BCCSP modulo ready pairs equivalence.

When A is infinite, Groote's technique of inverted substitutions can be applied to show that these axiomatizations are ω-complete. When $1 < |A| < \infty$, the aforementioned negative result from [8] (cf. the section on possible worlds) implies that BCCSP modulo ready pairs equivalence does not possess a finite basis. (Hence neither does ready pairs preorder.) When $|A| = 1$, ready pairs equivalence (resp. preorder) coincides with completed trace equivalence (resp. preorder), and we will see that in this case a finite basis does exist.

3.12 Failure Pairs

A sound and ground-complete axiomatization for BCCSP modulo failure pairs preorder is obtained by extending the four core axioms with

$$a(x+y) \preccurlyeq ax + a(y+z) \ .$$

It follows that the following two axioms

$$a(bx+by+z) \approx a(bx+by+z) + a(bx+z) \ ,$$
$$ax + a(y+z) \approx ax + a(y+z) + a(x+y)$$

suffice to obtain a finite, sound and ground-complete axiomatization for BCCSP modulo failure pairs equivalence. Fokkink and Nain [9] proved using cover equations that when A is infinite, these axiomatizations are ω-complete. They also proved that when A is finite, one extra axiom is needed to obtain an ω-complete axiomatization. For failure pairs preorder, it is formulated as

$$\sum_{i=1}^{|A|} a_i x_i \preccurlyeq \sum_{i=1}^{|A|} a_i x_i + y \ ,$$

while for failure pairs equivalence, it is formulated as

$$a(\sum_{i=1}^{|A|} b_i x_i + y + z) \approx a(\sum_{i=1}^{|A|} b_i x_i + y + z) + a(\sum_{i=1}^{|A|} b_i x_i + y) \ .$$

3.13 Completed Traces

A sound and ground-complete axiomatization for BCCSP modulo completed traces preorder is obtained by extending the four core axioms with

$$ax \preccurlyeq ax + y \ ,$$
$$a(bw+cx+y+z) \preccurlyeq a(bw+y) + a(cx+z) \ .$$

It follows that the following axiom

$$a(bw+y) + a(cx+z) \approx a(bw+cx+y+z)$$

suffices to obtain a sound and ground-complete axiomatization for BCCSP modulo completed traces equivalence.

Groote [10] proved using normal forms that in order to obtain an ω-complete axiomatization, one extra axiom is needed. For completed traces preorder, it is formulated as

$$a(x+y) \preccurlyeq ax + a(y+z) \ ,$$

while for complete traces equivalence, it is formulated as

$$ax + a(y+z) \approx ax + a(y+z) + a(x+y) \ .$$

3.14 Traces

A sound and ground-complete axiomatization for BCCSP modulo traces preorder is obtained by extending the four core axioms with

$$x \preccurlyeq x + y \ ,$$
$$a(x + y) \preccurlyeq ax + ay \ .$$

It follows that the following axiom

$$ax + ay \approx a(x + y)$$

suffices to obtain a sound and ground-complete axiomatization for BCCSP modulo traces equivalence.

Groote [10] proved using normal forms that these axiomatizations are ω-complete when $|A| > 1$. When $|A| = 1$, it is not hard to see that one extra axiom suffices to make the axiomatization ω-complete. For traces preorder, it is formulated as

$$x \preccurlyeq ax \ ,$$

while for traces equivalence, it is formulated as

$$ax + x \approx ax \ .$$

4 Conclusion

The questions whether a semantics in the linear time – branching time spectrum is finitely based over BCCSP have been settled completely. We summarize the results here:

– For most of the semantics in the linear time – branching time spectrum, corresponding preorders and equivalences share the same axiomatzability properties. The only exception is the impossible futures semantics. **Tab**. 2 presents an overview regarding this semantics, where + means that the axiomatization exists, − means that there is no such axiomatization.

	ground-comp.	ω-comp.	
	$1 \leq \lvert A\rvert \leq \infty$	$\lvert A\rvert = \infty$	$1 \leq \lvert A\rvert < \infty$
preorder	+	+	−
equivalence	−	−	−

Table 2. Axiomatizability of impossible futures for BCCSP

– BCCSP has a finite sound and ground-complete axiomatization for most of the semantics in the linear time – branching time spectrum. Only for 2-nested simulation and possible futures, and for ready traces in case of an infinite alphabet, such an axiomatization does not exist. **Tab**. 3 presents an overview, where we distinguish between an infinite alphabet and a finite alphabet.
– Regarding ω-completeness, matters are more mixed, especially when $1 < |A| < \infty$. **Tab**. 4 presents an overview, where we distinguish among an infinite alphabet, a finite alphabet with more than one element, and a singleton alphabet.

	$\|A\| < \infty$	$\|A\| = \infty$
bisimulation	+	+
2-nested simulation	−	−
possible futures	−	−
ready simulation	+	+
completed simulation	+	+
simulation	+	+
possible worlds	+	+
ready traces	+	−
failure traces	+	+
readies	+	+
failures	+	+
completed traces	+	+
traces	+	+

Table 3. The existence of ground-complete axiomatizations for BCCSP

	$\|A\| = 1$	$1 < \|A\| < \infty$	$\|A\| = \infty$
bisimulation	+	+	+
2-nested simulation	−	−	−
possible futures	−	−	−
ready simulation	+	−	+
completed simulation	+	−	−
simulation	+	−	+
possible worlds	+	−	+
ready traces	+	−	−
failure traces	+	−	+
readies	+	−	+
failures	+	+	+
completed traces	+	+	+
traces	+	+	+

Table 4. The existence of ω-complete axiomatizations for BCCSP

Acknowledgement

The work reported in this essay is mostly fruits of joint work with Luca Aceto, Rob van Glabbeek, Anna Ingolfsdottir, Bas Luttik and Sumit Nain. We would like to thank them for collaborations.

References

1. L. Aceto, W. Fokkink, and A. Ingólfsdóttir. Ready to preorder: Get your BCCSP axiomatization for free! In T. Mossakowski, U. Montanari, and M. Haveraaen, editors, *CALCO*, volume 4624 of *Lecture Notes in Computer Science*, pages 65–79. Springer, 2007.
2. L. Aceto, W. Fokkink, R. van Glabbeek, and A. Ingolfsdottir. Nested semantics over finite trees are equationally hard. 191(2):203–232, June 2004.
3. S. Blom, W. Fokkink, and S. Nain. On the axiomatizability of ready traces, ready simulation, and failure traces. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 109–118. Springer, 2003.
4. T. Chen and W. Fokkink. On finite alphabets and infinite bases III: Simulation. In C. Baier and H. Hermanns, editors, *CONCUR*, volume 4137 of *Lecture Notes in Computer Science*, pages 421–434. Springer, 2006.

5. T. Chen and W. Fokkink. On the axiomatizability of impossible futures: Preorder versus equivalence. In *LICS*, pages 156–165. IEEE Computer Society, 2008.
6. T. Chen, W. Fokkink, B. Luttik, and S. Nain. On finite alphabets and infinite bases. *Inf. Comput.*, 206(5):492–519, 2008.
7. T. Chen, W. Fokkink, and S. Nain. On finite alphabets and infinite bases II: Completed and ready simulation. In L. Aceto and A. Ingólfsdóttir, editors, *FoSSaCS*, volume 3921 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2006.
8. W. Fokkink and S. Nain. On finite alphabets and infinite bases: From ready pairs to possible worlds. In I. Walukiewicz, editor, *FoSSaCS*, volume 2987 of *Lecture Notes in Computer Science*, pages 182–194. Springer, 2004.
9. W. Fokkink and S. Nain. A finite basis for failure semantics. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 755–765. Springer, 2005.
10. J. F. Groote. A new strategy for proving ω-completeness applied to process algebra. In J. C. M. Baeten and J. W. Klop, editors, *CONCUR*, volume 458 of *Lecture Notes in Computer Science*, pages 314–331. Springer, 1990.
11. R. Gurevic. Equational theory of positive numbers with exponentiation is not finitely axiomatizable. *Ann. Pure Appl. Logic*, 49(1):1–30, 1990.
12. J. Heering. Partial evaluation and ω-completeness of algebraic specifications. *Theor. Comput. Sci.*, 43:149–167, 1986.
13. L. Henkin. The logic of equality. *American Mathematical Monthly*, 84(8):597–612, 1977.
14. A. Lazrek, P. Lescanne, and J.-J. Thiel. Tools for proving inductive equalities, relative completeness, and ω-completeness. *Inf. Comput.*, 84(1):47–70, 1990.
15. H. Lin. PAM: A process algebra manipulator. *Formal Methods in System Design*, 7(3):243–259, 1995.
16. R. Lyndon. Identities in two-valued calculi. *Transactions of the American Mathematical Society*, 71:457–465, 1951.
17. R. McKenzie. Tarski's finite basis problem is undecidable. *Journal of Algebra and Computation*, 6(1):49–104, 1996.
18. R. McKenzie, G. McNulty, and W. Taylor. *Algebras, Varieties, Lattices*. Wadsworth & Brooks/Cole, 1987.
19. F. Moller. *Axioms for Concurrency*. PhD thesis, Department of Computer Science, University of Edinburgh, July 1989. Report CST-59-89. Also published as ECS-LFCS-89-84.
20. V. Murskiĭ. The existence in the three-valued logic of a closed class with a finite basis having no finite complete system of identities. *Doklady Akademii Nauk SSSR*, 163:815–818, 1965. In Russian.
21. V. Murskiĭ. The existence of a finite basis of identities, and other properties of "almost all" finite algebras. *Problemy Kibernetiki*, 30:43–56, 1975. In Russian.
22. R. J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In J. C. M. Baeten and J. W. Klop, editors, *CONCUR*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990.
23. R. J. van Glabbeek. The linear time-branching time spectrum I. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. Elsevier Science, 2001.

Mealy Synthesis of Arithmetic Bitstream Functions

Helle Hvid Hansen
Formal Methods Group, Technische Universiteit Eindhoven, and
Centrum Wiskunde & Informatica, Amsterdam

A (binary) Mealy machine is a deterministic automaton which in each step reads an input bit, produces an output bit and moves to a next state. The induced mapping of input streams to output streams is a causal bitstream function, which we call the bitstream function realised by the Mealy machine. In this note, we describe a synthesis method which given an algebraic specification of a bitstream function f, constructs a minimal Mealy machine which realises f. In the design of digital hardware, Mealy machines specify the behaviour of sequential circuits, and there exist algorithms which construct from a (finite) Mealy machine, a sequential circuit which exhibits the specified behaviour. Combining these algorithms with our synthesis algorithm we thus obtain a complete construction from algebraic specification to sequential circuit.

The inputs to our synthesis algorithm are called function expressions and they define bitstream functions in the algebra of 2-adic numbers. Here we use that the formal power series representation of a 2-adic integer can be seen as the bitstream of its coefficients. We describe the 2-adic algebra below, but for now such a function expression can be thought of as a specification of a function on the rational numbers. The interesting property of 2-adic arithmetic is that it allows us to calculate with rational numbers in an easy manner similar to how one computes with integers.

The synthesis algorithm described here can be seen as an analogue of Brzozowski's construction in [3] of a finite deterministic automaton from a regular expression. In particular, we show that the set of function expressions can be given the structure of a Mealy machine by giving an inductive definition of derivative and output of function expressions. This Mealy machine of expressions is defined in such a way that the algebraic semantics coincides with the behavioural semantics of Mealy machines. Hence, given a function expression `E` which specifies a function f, the submachine generated by `E` realises f. In general, this submachine is not minimal, but we can ensure minimality by reducing expressions to normal form. In addition, the language of 2-adic arithmetic allows the specification of functions that are not realised by any finite Mealy machine. But we identify a subclass of so-called rational function expressions for which a finite realisation exists. Hence given a rational function expression `E`, we can effectively construct a minimal Mealy machine which realises the bitstream function specified by `E`. This is our main result.

The theory underlying the synthesis method is essentially coalgebraic (cf. [10]), but we have deliberately chosen for a presentation which does not require any familiarity with coalgebra. The fundamental ideas behind the synthesis method are due to Jan Rutten (cf. [13]). These ideas were developed into a proper algorithm in [7] and implemented by the author and David Costa (cf. [6]). Further results

on complexity and size of realisations were included in the author's PhD thesis [5, Ch. 3]. This note contains a short, but improved presentation of the basic results in the abovementioned work. In particular, the presentation of the Mealy machine of expressions, the algebraic semantics of function expressions, and the proof that algebraic semantics coincides with behavioural semantics for function expressions are new with respect to [5, 7, 13].

The rest of this paper is structured as follows. In Section 1 we give the basic definitions regarding Mealy machines, and in Section 2 we describe the 2-adic algebra of bitstreams and define the function expressions which serve as our specification language. In Section 3 we show how function expressions can be turned into a Mealy machine, and Section 4 describes the synthesis method for rational functions. Finally, we discuss related and future work in Section 5.

1 Mealy machines

We denote by $2 = \{0, 1\}$ the set of bits, and by $2^\omega = \{\sigma \colon \mathbb{N} \to 2\}$ the set of bitstreams. A (binary) Mealy machine $\langle S, m\rangle$ consists of a (possibly infinite) set S of states, and a function $m \colon S \to (2 \times S)^2$, called the Mealy structure. For every state $s \in S$ and every input bit $a \in 2$, m returns a pair $m(s)(a) = \langle b, t\rangle$ where $b \in 2$ is called the output and $t \in S$ is the next state. We will write

$$s \xrightarrow{a|b} t \quad \text{iff} \quad m(s)(a) = \langle b, t\rangle.$$

A *Mealy morphism* from $\langle S, m\rangle$ to $\langle S', m'\rangle$ is a function $h \colon S \to S'$ that preserves transitions: if $m(s)(a) = \langle b, t\rangle$ then $m'(h(s))(a) = \langle b, h(t)\rangle$; in other words,

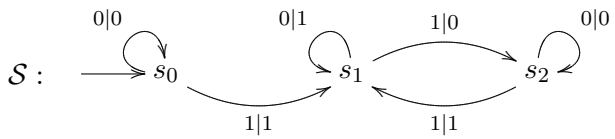
$$s \xrightarrow{a|b} t \quad \Rightarrow \quad h(s) \xrightarrow{a|b} h(t)$$

We define the (input-output) *behaviour* of a state s_0 in $\langle S, m\rangle$ as the bitstream function $beh(s_0) \colon 2^\omega \to 2^\omega$ which maps a bitstream $(a_0, a_1, a_2, \dots)$ to $(b_0, b_1, b_2, \dots)$ given by the unique sequence of transitions

$$s_0 \xrightarrow{a_0|b_0} s_1 \xrightarrow{a_1|b_1} \cdots \xrightarrow{a_k|b_k} s_{k+1} \xrightarrow{a_{k+1}|b_{k+1}} \cdots$$

We say that a state s in $\langle S, m\rangle$ *realises* a bitstream function f if $beh(s) = f$.

Example 1.1 The figure below shows an example of a Mealy machine which starting in state s_0 counts the number of 1's in the input modulo 2. Formally, on input stream $\alpha \in 2^\omega$, the k-th element of the output stream is $(beh(s_0)(\alpha))(k) = \sum_{i=0}^{k} \alpha(i) \mod 2$.



For arbitrary $\alpha \in 2^\omega$ and $k \in \mathbb{N}$, $(beh(s_0)(\alpha))(k)$ is clearly determined by $\alpha(0), \dots, \alpha(k)$. In other words, $beh(s_0)$ is a causal bitstream function.

A bitstream function $f\colon 2^\omega \to 2^\omega$ is *causal*, if for all $\alpha, \beta \in 2^\omega$ and all $k \in \mathbb{N}$: if $\alpha(i) = \beta(i)$ for all $i \leq k$, then $(f(\alpha))(k) = (f(\beta))(k)$. It is straightforward to prove that for any Mealy machine $\langle S, m\rangle$ and any $s \in S$, $beh(s)\colon 2^\omega \to 2^\omega$ is causal. Let Γ denote the set of all causal bitstream functions.

We now show that Γ carries a Mealy structure. We need the following notation. The head and tail maps on bitstreams are denoted by $hd\colon 2^\omega \to 2$ and $tl\colon 2^\omega \to 2^\omega$, respectively, that is, $hd(\alpha) = \alpha(0)$ and $tl(\alpha) = (\alpha(1), \alpha(2), \alpha(3), \ldots)$. For $\alpha \in 2^\omega$, a bit $a \in 2$ and a bitstream function f, we denote by $a:\alpha$ the bitstream $(a, \alpha(0), \alpha(1), \alpha(2), \ldots)$, and we write $f(a:-)$ for the bitstream function that maps α to $f(a:\alpha)$. Now let $f \in \Gamma$ and $a \in 2$. We define

$$\begin{array}{lcll} f[a] & := & hd \circ f(a:-) & \in \quad 2^\omega \to 2 \\ f_a & := & tl \circ f(a:-) & \in \quad 2^\omega \to 2^\omega \end{array} \tag{1}$$

Since f is causal, it follows that also f_a is causal, (hence $f \in \Gamma$) and that $f[a]$ is constant, hence $f[a]$ can be considered an element of 2. We call $f[a]$ the initial output of f on input a, and f_a is the stream function derivative of f on input a. We define the Mealy structure $\gamma\colon \Gamma \to (2 \times \Gamma)^2$ by $\gamma(f)(a) = \langle f[a], f_a\rangle$, and let $\mathbf{\Gamma} = \langle \Gamma, \gamma\rangle$.

Theorem 1.2 *(cf. [13]) For any Mealy machine $\mathcal{M} = \langle S, m\rangle$, the behaviour map $beh_{\mathcal{M}}\colon S \to \Gamma$ is the unique Mealy morphism from $\mathcal{M}$ to $\mathbf{\Gamma}$. In other words, $\mathbf{\Gamma}$ is a final Mealy machine.*

For a Mealy machine $\mathcal{M}$, we say that $beh_{\mathcal{M}}$ assigns to its states their *behavioural semantics*, and $\mathcal{M}$ is *minimal* if $beh_{\mathcal{M}}$ is injective. For example, the Mealy machine in Example 1.1 is not minimal since $beh(s_0) = beh(s_2)$. Given a state s in $\mathcal{M} = \langle S, m\rangle$, the *submachine* $\langle\!\langle s\rangle\!\rangle = \langle S', m'\rangle$ *generated by s in $\mathcal{M}$* consists of the smallest subset $S' \subseteq S$ such that $s \in S'$, m' is the restriction of m to S' and the inclusion map $S' \to S$ is a Mealy morphism. Another way of stating this requirement is to say that $\langle\!\langle s\rangle\!\rangle$ is the least subset containing s which is transition closed (cf. [10]). We call $\langle\!\langle s\rangle\!\rangle$ a *realisation* of f, if $beh_{\mathcal{M}}(s) = f$, i.e., s realises f in $\mathcal{M}$. From the finality of $\mathbf{\Gamma}$, we get:

Corollary 1.3 *For all $f \in \Gamma$, $\langle\!\langle f\rangle\!\rangle$ is a minimal Mealy realisation of f.*

In our synthesis method, given an expression that specifies a function $f \in \Gamma$, we construct a representation of $\langle\!\langle f\rangle\!\rangle$ through a symbolic least fixed point computation.

2 The 2-adic bitstream algebra

We will specify bitstream functions in the algebra of 2-adic integers (cf. [4]). A 2-adic integer is usually written as a (formal) power series of the form $\sum_{i=0}^{\infty} a_i 2^i$ where $a_i \in 2$ for all $i \in \{0, 1, 2, \ldots\}$. We identify such a power series with the bitstream $(a_0, a_1, a_2, \ldots)$. The 2-adic integers form an integral domain[1] which extends that of the rational numbers with odd denominator $Q_{odd} = \{p/q \mid p, q \in \mathbb{Z}, q \text{ odd }\}$.[2] The

[1] An integral domain is a commutative ring A with no zero-divisors, i.e., for all $a, b \in A$, if $a \cdot b = 0$ then $a = 0$ or $b = 0$.

[2] Rationals with even denominator require formal power series representations $\sum_{i=k}^{\infty} a_i 2^i$ where $k < 0$.

(strict) inclusion of Q_{odd} into the 2-adic integers is obtained by taking infinitary base 2 expansions (see e.g. [8]). For a positive integer n, this is just the binary representation of n (least significant bit on the left) padded with a tail of zeros. Here are a few examples:

$$\begin{array}{lcl} \mathrm{Bin}(2) & = & (0,1,0,0,0,\ldots), \\ \mathrm{Bin}(5) & = & (1,0,1,0,0,\ldots), \\ \mathrm{Bin}(-1) & = & (1,1,1,1,\ldots), \\ \mathrm{Bin}(-5) & = & (1,1,0,1,1,1,\ldots), \text{ and} \\ \mathrm{Bin}(1/5) & = & (1,0,1,1,0,0,1,1,0,0,1,1,0,0,\ldots). \end{array}$$

In general, if q is a negative integer, then $\mathrm{Bin}(q)$ ends in a tail of 1's, and for all $q \in Q_{odd}$, $\mathrm{Bin}(q)$ is eventually periodic.

The 2-adic bitstream algebra $\mathcal{A}_{2adic} = \langle 2^\omega, +, -\times, /, [0], [1] \rangle$ is obtained from the bitstream representation of 2-adic arithmetic. In particular, $[0] = \mathrm{Bin}(0)$ and $[1] = \mathrm{Bin}(1)$. An attractive property of 2-adic arithmetic is that one can calculate with bitstreams (including elements of Q_{odd}) in a manner similar to how one calculates with the usual integers. For example, 2-adic sum is an infinitary version of binary addition where carry bits can be propagated indefinitely. This is how one sees that $\mathrm{Bin}(-1) + \mathrm{Bin}(1) = (1,1,1,\ldots) + (1,0,0,\ldots) = [0]$. The 2-adic operations satisfy the so-called *stream differential equations* in Figure 1. Stream differential equations are a general way of defining streams as the unique solution to a set of equations. We refer to [11] for a comprehensive overview. In analogy with traditional differential calculus, the head and tail of streams in stream diferential equations are usually referred to as the initial value and derivative, respectively, and denoted by $\alpha(0)$ and α'. Moreover, $\wedge$ and $\oplus$ denote the bit operations of Boolean AND and XOR (or addition modulo 2), respectively.

We briefly explain the equations in Figure 1. The equation for sum shows that a carry term must be added in case the two initial values were both 1. The equation for minus is obtained from the requirement that $(-\alpha) + \alpha = [0]$ for all $\alpha \in 2^\omega$. By taking initial value and derivative on both sides and using the equation for $+$, we find that $(-\alpha)(0) \oplus \alpha(0) = 0$, hence $(-\alpha)(0) = \alpha(0)$, and that $(-\alpha)' + \alpha' + [\alpha(0)] = [0]$, hence $(-\alpha)' = -(\alpha' + [\alpha(0)])$. The equation for the product states that for all $\alpha, \beta \in 2^\omega$, $\alpha \times \beta$ can be calculated using the base 2 version of shift-add-multiplication known from the multiplication in decimal notation. Finally, the multiplicative inverse of $\alpha \in 2^\omega$ is defined only if $\alpha(0) = 1$, since $\alpha(0)$ must be invertible in the underlying ring. If $1/\alpha$ is defined then it satisfies $(1/\alpha) \times \alpha = [1]$. The equation for $1/\alpha$ can be derived in a similar way as for $-\alpha$.

derivative:	initial value:
$(\alpha + \beta)' = (\alpha' + \beta') + [\alpha(0) \wedge \beta(0)]$	$(\alpha + \beta)(0) = \alpha(0) \oplus \beta(0)$
$(-\alpha)' = -(\alpha' + [\alpha(0)])$	$(-\alpha)(0) = \alpha(0)$
$(\alpha \times \beta)' = (\alpha' \times \beta) + ([\alpha(0)] \times \beta')$	$(\alpha \times \beta)(0) = \alpha(0) \wedge \beta(0)$
$(1/\alpha)' = -(\alpha' \times (1/\alpha))$	$(1/\alpha)(0) = 1$ (condition: $\alpha(0) = 1$)

Figure 1: The 2-adic operations

When calculating with the 2-adic operations it is convenient to also have a

symbol for the bitstream Bin(2). We define $X := \mathrm{Bin}(2) = (0,1,0,0,0,\ldots)$. Some of the identities that hold in $\mathcal{A}_{2adic}$ are: For all $\alpha = (a_0, a_1, a_2, a_3, \ldots) \in 2^\omega$,

$$\begin{aligned} X \times \alpha &= 0{:}\alpha = (0, a_0, a_1, a_2, \ldots) \\ [1] + (X \times \alpha) &= 1{:}\alpha = (1, a_0, a_1, a_2, \ldots) \\ \alpha &= [\alpha(0)] + (X \times \alpha') \\ \alpha + \alpha &= X \times \alpha \end{aligned}$$

3 Mealy machine of expressions

We will specify bitstream functions in the language of $\mathcal{A}_{2adic}$ over a single variable $\mathtt{s}$. Formally, the set of *function expressions* $\mathtt{FExpr}$ is generated by the following grammar:

$$\mathtt{E}, \mathtt{F} \quad ::= \quad \mathtt{s} \mid \mathtt{0} \mid \mathtt{1} \mid \mathtt{X} \mid -\mathtt{E} \mid \mathtt{E} + \mathtt{F} \mid \mathtt{E} \times \mathtt{F} \mid \mathtt{1}/(\mathtt{1} + (\mathtt{X} \times \mathtt{E}))$$

We use the symbols $-, +, \times, /$ to denote the operations on bitstreams as well as the corresponding syntax constructors. The typing should always be clear from the context. We will use standard notational conventions: we write $\mathtt{E}^n$ for the n-fold product of $\mathtt{E}$ with itself, and $\mathtt{F}/(\mathtt{1} + (\mathtt{X} \times \mathtt{E}))$ or $\frac{\mathtt{F}}{\mathtt{1}+(\mathtt{X}\times\mathtt{E})}$ instead of $\mathtt{F} \times (\mathtt{1}/(\mathtt{1} + (\mathtt{X} \times \mathtt{E})))$. The algebraic semantics of function expressions is given by the expected interpretation in $\mathcal{A}_{2adic}$. The reason we only allow division by terms of the form $\mathtt{1} + (\mathtt{X} \times \mathtt{E})$ is to ensure that the inverse operation is always defined when evaluating function expressions in $\mathcal{A}_{2adic}$.

Definition 3.1 We define the *algebraic 2-adic semantics* $[\![\mathtt{E}]\!] \colon 2^\omega \to 2^\omega$ of a function expression $\mathtt{E} \in \mathtt{FExpr}$ by the following inductive clauses. Let σ be a bitstream,

$$\begin{aligned} [\![\mathtt{s}]\!](\sigma) &= \sigma & [\![-\mathtt{E}]\!](\sigma) &= -([\![\mathtt{E}]\!](\sigma)) \\ [\![\mathtt{0}]\!](\sigma) &= [0] & [\![\mathtt{E} + \mathtt{F}]\!](\sigma) &= [\![\mathtt{E}]\!](\sigma) + [\![\mathtt{F}]\!](\sigma) \\ [\![\mathtt{1}]\!](\sigma) &= [1] & [\![\mathtt{E} \times \mathtt{F}]\!](\sigma) &= [\![\mathtt{E}]\!](\sigma) \times [\![\mathtt{F}]\!](\sigma) \\ [\![\mathtt{X}]\!](\sigma) &= X & [\![\mathtt{1}/(\mathtt{1} + (\mathtt{X} \times \mathtt{E})]\!](\sigma) &= 1/(1 + (X \times [\![\mathtt{E}]\!](\sigma)) \end{aligned}$$

We say that a function expression $\mathtt{E}$ *specifies* the bitstream function $[\![\mathtt{E}]\!]$, and two function expressions $\mathtt{E}$ and $\mathtt{F}$ are *equivalent* (notation: $\mathtt{E} \equiv \mathtt{F}$) if $[\![\mathtt{E}]\!] = [\![\mathtt{F}]\!]$.

One easily shows (by structural induction on function expressions) that for all $\mathtt{E} \in \mathtt{FExpr}$, $[\![\mathtt{E}]\!]$ is a causal bitstream function. In other words, $[\![-]\!]$ is a map from $\mathtt{FExpr}$ to Γ. We will now define a Mealy structure on $\mathtt{FExpr}$ such that the algebraic semantics $[\![-]\!]$ is a Mealy morphism. In order to define a map $\xi \colon \mathtt{FExpr} \to (2 \times \mathtt{FExpr})^2$, first recall how we defined $\gamma(f)(a)$ in terms of $f(a:-)$, hd and tl (cf. equation (1)) for $f \in \Gamma$ and $a \in 2$. We will "mimick" γ in the syntax.

First, we need a syntactic version of the map $f(a:-)$ for $f \in \Gamma$ and $a \in 2$. More precisely, given $\mathtt{E} \in \mathtt{FExpr}$ and $a \in 2$, we want to find a function expression which specifies $[\![E]\!](a:-)$. Note that we have for all $\sigma \in 2^\omega$: $[\![\mathtt{X} \times \mathtt{s}]\!](\sigma) = 0 : \sigma$ and $[\![\mathtt{1} + (\mathtt{X} \times \mathtt{s})]\!](\sigma) = 1 : \sigma$. For $\mathtt{E} \in \mathtt{FExpr}$ and $a \in 2$, let $0 : \mathtt{s} := \mathtt{X} \times \mathtt{s}$ and $1 : \mathtt{s} := \mathtt{1} + (\mathtt{X} \times \mathtt{s})$, and define $\mathtt{E}(a : \mathtt{s})$ to be the function expression obtained by substituting $a : \mathtt{s}$ for $\mathtt{s}$ in $\mathtt{E}$. The following lemma can then be proved by induction on $\mathtt{E}$.

initial output (syntax)					
$\mathtt{s}[a]$	$=$	a	$(-\mathtt{E})[a]$	$=$	$\mathtt{E}[a]$
$\mathtt{0}[a]$	$=$	0	$(\mathtt{E}+\mathtt{F})[a]$	$=$	$\mathtt{E}[a] \oplus \mathtt{F}[a]$
$\mathtt{1}[a]$	$=$	1	$(\mathtt{E}\times\mathtt{F})[a]$	$=$	$\mathtt{E}[a] \wedge \mathtt{F}[a]$
$\mathtt{X}[a]$	$=$	0	$(\mathtt{1}/(\mathtt{1}+(\mathtt{X}\times\mathtt{E})))[a]$	$=$	1
stream function derivative (syntax)					
$\mathtt{s}_a$	$=$	$\mathtt{s}$	$(-\mathtt{E})_a$	$=$	$-(\mathtt{E}_a + \iota(\mathtt{E}[a]))$
$\mathtt{0}_a$	$=$	$\mathtt{0}$	$(\mathtt{E}+\mathtt{F})_a$	$=$	$(\mathtt{E}_a + \mathtt{F}_a) + \iota(\mathtt{E}[a] \wedge \mathtt{F}[a])$
$\mathtt{1}_a$	$=$	$\mathtt{0}$	$(\mathtt{E}\times\mathtt{F})_a$	$=$	$(\mathtt{E}_a \times \mathtt{F}(a:\mathtt{s})) + (\iota(\mathtt{E}[a]) \times \mathtt{F}_a)$
$\mathtt{X}_a$	$=$	$\mathtt{1}$	$(\mathtt{1}/(\mathtt{1}+(\mathtt{X}\times\mathtt{E})))_a$	$=$	$-(\mathtt{E}(a:\mathtt{s}))/(\mathtt{1}+(\mathtt{X}\times\mathtt{E}(a:\mathtt{s})))$

Figure 2: Mealy structure on function expressions

Lemma 3.2 *For all* $\mathtt{E} \in \mathtt{FExpr}$*, all* $a \in 2$ *and all* $\sigma \in 2^\omega$*:* $[\![\mathtt{E}(a:\mathtt{s})]\!](\sigma) = [\![\mathtt{E}]\!](a:\sigma)$.

Second, we need syntactic versions of the head and tail maps. That is, we want to define a pair of maps $\langle o, d\rangle \colon \mathtt{FExpr} \to (2 \times \mathtt{FExpr})$. To this end, observe that the stream differential equations in Figure 1 can be read as an inductive definition over function expressions. We only need to add the obvious clauses for the atomic expressions $[0], [1]$ and $\mathtt{X}$. For example, $o(\mathtt{X}) = \mathtt{X}(0) = 0$ and $d(\mathtt{X}) = \mathtt{X}' = [1]$. Since $\mathtt{s}$ is a variable, it is not possible to give a stream differential equation for $\mathtt{s}$. But this is no problem, since we only need to define o and d on expressions of the form $\mathtt{E}(a:\mathtt{s})$ in which $\mathtt{s}$ always occurs as a subterm of $\mathtt{X} \times \mathtt{s}$. Recall that for all $\alpha \in 2^\omega$, $tl(X \times \alpha) = \alpha$. We therefore define for all $\mathtt{F} \in \mathtt{FExpr}$: $o(\mathtt{X}\times\mathtt{s}) = 0$ and $d(\mathtt{X}\times\mathtt{s}) = \mathtt{s}$, and for product expressions $\mathtt{E} \times \mathtt{F}$ where $\mathtt{F} \neq \mathtt{s}$, we take $o(\mathtt{E} \times \mathtt{F}) = o(\mathtt{E}) \wedge o(\mathtt{F})$ and $d(\mathtt{E} \times \mathtt{F}) = (d(\mathtt{E}) \times \mathtt{F}) + (\iota(o(\mathtt{E})) \times d(\mathtt{F}))$, where $\iota \colon 2 \to \mathtt{FExpr}$ is defined by $\iota(0) = \mathtt{0}$ and $\iota(1) = \mathtt{1}$.

For $\mathtt{E} \in \mathtt{FExpr}$ and $a \in 2$ we will use the suggestive notation: $\mathtt{E}[a] := o(\mathtt{E}(a:\mathtt{s}))$ and $\mathtt{E}_a := d(\mathtt{E}(a:\mathtt{s}))$; and we define $\xi(\mathtt{E})(a) = \langle \mathtt{E}[a], \mathtt{E}_a\rangle$. The definition of ξ is detailed in Figure 2. We refer to $\langle \mathtt{FExpr}, \xi\rangle$ as the Mealy machine of expressions.

Proposition 3.3 *The map* $[\![-]\!] \colon \langle \mathtt{FExpr}, \xi\rangle \to \langle \Gamma, \gamma\rangle$ *is a Mealy morphism.*

Proof. We must show that for all $\mathtt{E} \in \mathtt{FExpr}$ and all $a \in 2$:

$$[\![\mathtt{E}]\!][a] = \mathtt{E}[a] \quad \text{and} \quad [\![\mathtt{E}_a]\!] = [\![\mathtt{E}]\!]_a.$$

The proof is by induction on the structure of $\mathtt{E}$. We only show the case for $\mathtt{s}$ and product. In the identities below, IH refers to the induction hypothesis, and L.3.2 refers to Lemma 3.2. Note also that for any $b \in 2$ and $\sigma \in 2^\omega$: $[\![\iota(b)]\!](\sigma) = [b]$. Let $a \in 2$ and $\sigma \in 2^\omega$.

$$\begin{array}{lllllll} [\![\mathtt{s}]\!][a] & = & ([\![\mathtt{s}]\!](a:\sigma))(0) & = & (a:\sigma)(0) = a & = & \mathtt{s}[a] \\ [\![\mathtt{s}]\!]_a(\sigma) & = & ([\![\mathtt{s}]\!](a:\sigma))' & = & (a:\sigma)' = \sigma & = & [\![\mathtt{s}_a]\!](\sigma) \end{array}$$

$$
\begin{array}{lcl}
[\![\mathtt{E}\times\mathtt{F}]\!][a] & = & [\![\mathtt{E}]\!][a]\wedge[\![\mathtt{F}]\!][a] \stackrel{\text{IH}}{=} \mathtt{E}[a]\wedge\mathtt{F}[a] = (\mathtt{E}\times\mathtt{F})[a].\\
[\![\mathtt{E}\times\mathtt{F}]\!]_a(\sigma) & = & ([\![\mathtt{E}\times\mathtt{F}]\!](a{:}\sigma))' = ([\![\mathtt{E}]\!](a{:}\sigma)\times[\![\mathtt{F}]\!](a{:}\sigma))'\\
& = & (([\![\mathtt{E}]\!](a{:}\sigma))'\times[\![\mathtt{F}]\!](a{:}\sigma)) + ([([\![\mathtt{E}]\!](a{:}\sigma))(0)]\times([\![\mathtt{F}]\!](a{:}\sigma))')\\
& = & ([\![\mathtt{E}]\!]_a(\sigma)\times[\![\mathtt{F}]\!](a{:}\sigma)) + ([[\![\mathtt{E}]\!][a]]\times[\![\mathtt{F}]\!]_a(\sigma))\\
& \stackrel{\text{IH}}{=} & ([\![\mathtt{E}_a]\!](\sigma)\times[\![\mathtt{F}]\!](a{:}\sigma)) + ([\mathtt{E}[a]]\times[\![\mathtt{F}_a]\!](\sigma))\\
& \stackrel{\text{L.3.2}}{=} & [\![(\mathtt{E}_a(\sigma)\times\mathtt{F}(a{:}\mathtt{s})) + (\iota(\mathtt{E}[a])\times\mathtt{F}_a)]\!](\sigma)\\
& = & [\![(\mathtt{E}\times\mathtt{F})_a]\!](\sigma).
\end{array}
$$

QED

Since $\langle\Gamma,\gamma\rangle$ is a final Mealy machine, the map $[\![-]\!]$ coincides with the unique Mealy morphism $beh\colon\langle\mathtt{FExpr},\xi\rangle\to\langle\Gamma,\gamma\rangle$. In other words, for any function expression $\mathtt{E}$, the algebraic semantics $[\![\mathtt{E}]\!]$ equals the behavioural semantics $beh(\mathtt{E})$.

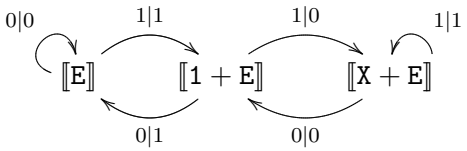
4 Synthesis

A consequence of Proposition 3.3 is that the generated submachine $\langle\!\langle\mathtt{E}\rangle\!\rangle$ is a realisation of $[\![\mathtt{E}]\!]$. Conceptually, we can construct $\langle\!\langle\mathtt{E}\rangle\!\rangle$ by computing the transition closure of $\{\mathtt{E}\}$ in $\langle\mathtt{FExpr},\xi\rangle$. However, in general, $\langle\!\langle\mathtt{E}\rangle\!\rangle$ is neither finite nor minimal. In order to obtain a minimal realisation of $[\![\mathtt{E}]\!]$ we compute $\langle\!\langle\mathtt{E}\rangle\!\rangle$ modulo equivalence. In practice, this is achieved by reducing function expressions to normal form. We briefly describe these normal forms. We call a function expression *integral* if it does not contain the inverse operation. The *polynomial normal form* $pnf(\mathtt{E})$ of an integral expression $\mathtt{E}$ is an analogue of the distributed normal form of polynomials (in the variable $\mathtt{s}$). For example, $(\mathtt{1}+\mathtt{X})\times(\mathtt{1}+\mathtt{s})+\mathtt{1}\equiv\mathtt{1}+\mathtt{s}+\mathtt{X}+(\mathtt{X}\times\mathtt{s})+\mathtt{1}\equiv\mathtt{X}^2+(\mathtt{1}+\mathtt{X})\times\mathtt{s}$. Note that in the last step we used that $\mathtt{1}+\mathtt{1}+\mathtt{X}\equiv\mathtt{X}^2$. We therefore have that $pnf((\mathtt{1}+\mathtt{X})\times(\mathtt{1}+\mathtt{s})+\mathtt{1}) = \mathtt{X}^2+(\mathtt{1}+\mathtt{X})\times\mathtt{s}$. Given arbitrary function expressions $\mathtt{E}$ and $\mathtt{F}$, we can decide whether $\mathtt{E}\equiv\mathtt{F}$ by first rewriting (using the identities of integral domains) $\mathtt{E}$ and $\mathtt{F}$ into fractions $\mathtt{P}/\mathtt{Q}$ and $\mathtt{R}/\mathtt{S}$, respectively, where $\mathtt{P},\mathtt{Q},\mathtt{R},\mathtt{S}$ are integral, and then checking whether $pnf(\mathtt{P}\times\mathtt{S}) = pnf(\mathtt{R}\times\mathtt{Q})$. These normal forms are treated in detail in [5].

Example 4.1 We illustrate by computing (a representation of) $\langle\!\langle[\![\mathtt{E}]\!]\rangle\!\rangle$ for the function expression $\mathtt{E}=(\mathtt{1}+\mathtt{X})\times\mathtt{s}$. For the transition on input 1 we find that:

$$
\begin{array}{lcl}
((\mathtt{1}+\mathtt{X})\times\mathtt{s})[1] & = & (\mathtt{1}[1]\oplus\mathtt{X}[1])\wedge\mathtt{s}[1] = (1\oplus 0)\wedge 1 = 1.\\
((\mathtt{1}+\mathtt{X})\times\mathtt{s})_1 & = & ((\mathtt{1}+\mathtt{X})_1\times\mathtt{s}(1{:}\mathtt{s})) + (\iota((\mathtt{1}+\mathtt{X})[1])\times\mathtt{s}_1)\\
& = & (((\mathtt{1}_1+\mathtt{X}_1)+\iota(\mathtt{1}[1]\wedge\mathtt{X}[1]))\times(\mathtt{1}+(\mathtt{X}\times\mathtt{s}))) + (\iota(\mathtt{1}[1]\oplus\mathtt{X}[1])\times\mathtt{s}_1)\\
& = & ((\mathtt{0}+\mathtt{1})+(\iota(1\wedge 0)\times(\mathtt{1}+(\mathtt{X}\times\mathtt{s})))) + (\iota(1\oplus 0)\times\mathtt{s})\\
& = & (((\mathtt{0}+\mathtt{1})+\mathtt{0})\times(\mathtt{1}+(\mathtt{X}\times\mathtt{s})) + (\mathtt{1}\times\mathtt{s})\\
& \equiv & \mathtt{1}+((\mathtt{1}+\mathtt{X})\times\mathtt{s}) = \mathtt{1}+\mathtt{E}
\end{array}
$$

Continuing in this way we find the following minimal realisation of $[\![\mathtt{E}]\!]$:



The normal forms essentially allow us to compute submachines in the final Mealy machine. However, for arbitrary $\mathtt{E} \in \mathtt{FExpr}$, the least fixed point construction of $\langle\!\langle [\![\mathtt{E}]\!] \rangle\!\rangle$ is not guaranteed to terminate as it is easy to specify functions that have no finite realisation. For example, if we take $\mathtt{E} = \mathtt{s} \times \mathtt{s}$ and we compute the derivatives $\mathtt{E}_0, \mathtt{E}_{00}, \mathtt{E}_{000}, \ldots$ we get (modulo equivalence) the sequence $\mathtt{X} \times \mathtt{E},\ \mathtt{X}^2 \times \mathtt{E},\ \mathtt{X}^3 \times \mathtt{E}, \ldots$ which are all pairwise inequivalent. A similar argument shows that $\mathtt{E} = \mathtt{1}/(\mathtt{1} + (\mathtt{X} \times \mathtt{s}))$ has infinitely many stream function derivatives.

We now define a class of expressions that specify functions with finite realisations. A *rational function expression* is a function expression of the form

$$\mathtt{E} = \frac{\mathtt{F} + (\mathtt{G} \times \mathtt{s})}{1 + (\mathtt{X} \times \mathtt{H})}$$

where $\mathtt{F}, \mathtt{G}, \mathtt{H} \in \mathtt{FExpr}$ are closed (i.e., do not contain $\mathtt{s}$), integral function expressions. In other words, $\mathtt{F}, \mathtt{G}$ and $\mathtt{H}$ specify constant bitstream functions whose value is of the form $\mathrm{Bin}(x)$ for some integer $x \in \mathbb{Z}$. An example of a rational function expression is $\mathtt{E} = ((-\mathtt{X}) + (\mathtt{1} + \mathtt{X}) \times \mathtt{s})/(\mathtt{1} + (\mathtt{X} \times (\mathtt{X} + \mathtt{X})))$ which specifies $f(\sigma) = (\mathrm{Bin}(-2) + (\mathrm{Bin}(1+2) \times \sigma))/\mathrm{Bin}(1+2\times(2+2)) = (\mathrm{Bin}(-2) + (\mathrm{Bin}(3) \times \sigma))/\mathrm{Bin}(9)$. A function $f \in \Gamma$ is called *rational* if there is a rational function expression $\mathtt{E}$ such that $[\![\mathtt{E}]\!] = f$. Hence the function $f(\sigma) = \mathrm{Bin}(3) \times \sigma$ specified in Example 4.1 is also rational by taking $\mathtt{F} = \mathtt{H} = \mathtt{0}$ and $\mathtt{G} = \mathtt{1} + \mathtt{X}$. The crucial properties of rational functions are listed in the following lemma.

Lemma 4.2 *For all rational functions $f \in \Gamma$, we have:*

1. *all stream function derivatives g of f are rational,*
2. *$\langle\!\langle f \rangle\!\rangle$ is finite.*

Proof. The first item follows from the observation that for all rational function expressions $\mathtt{E} = (\mathtt{F} + (\mathtt{G} \times \mathtt{s}))/(1 + (\mathtt{X} \times \mathtt{H}))$ and $a \in 2$, the stream function derivative $[\![\mathtt{E}]\!]_a$ is specified by an expression of the form $(\mathtt{D} + (\mathtt{G} \times \mathtt{s}))/(1 + (\mathtt{X} \times \mathtt{H}))$ where $\mathtt{D}$ is a closed, integral expression. This can be verified by writing out the definition of $\mathtt{E}_a$ and rewriting using the integral domain identities. This argument can be inductively extended to show that all $g \in \langle\!\langle [\![\mathtt{E}]\!] \rangle\!\rangle$ are specified by an expression of the form $(\mathtt{D}_g + (\mathtt{G} \times \mathtt{s}))/(1 + (\mathtt{X} \times \mathtt{H}))$ where $\mathtt{D}_g$ is a closed, integral expression.

The second item is proved by showing that the set $D = \{[\![\mathtt{D}_g]\!] \mid g \in \langle\!\langle [\![\mathtt{E}]\!] \rangle\!\rangle\}$ is finite. Recall that for an integer $x \in \mathbb{Z}$, $\mathrm{Bin}(x) = (x_0, x_1, x_2, \ldots)$ is an eventually constant bitstream, i.e., there is a least $n \in \mathbb{N}$ such that for all $k \geq n$, $x_k = x_n$. We call this n the *degree* of $\mathrm{Bin}(x)$ and write it as $\deg(\mathrm{Bin}(x))$. Hence the semantics $[\![\mathtt{C}]\!]$ of a closed, integral function expression $\mathtt{C}$ is an eventually constant bitstream. The main technical argument consists in showing that there is an $N \in \mathbb{N}$ (which depends on the degree of $[\![F]\!], [\![G]\!]$ and $[\![H]\!]$) such that for all $g \in \langle\!\langle [\![\mathtt{E}]\!] \rangle\!\rangle$, $\deg([\![\mathtt{D}_g]\!]) \leq N$. Since there are only finitely many bitstreams of degree at most N, the set D is finite. We refer to [5, Ch. 3] for details. QED

The above lemma ensures that synthesis from rational function expressions terminates. The last theorem summarises our results concerning rational functions.

Theorem 4.3 *For any rational function expression* $\mathtt{E}$*, the Mealy machine* $\langle\langle [\![\mathtt{E}]\!] \rangle\rangle$ *is a finite, minimal realisation of* $[\![\mathtt{E}]\!]$*, and we can effectively construct a representation of* $\langle\langle [\![\mathtt{E}]\!] \rangle\rangle$ *by computing* $\langle\langle \mathtt{E} \rangle\rangle$ *modulo equivalence.*

5 Conclusion

We have shown how to construct a finite Mealy machine realisation of rational 2-adic functions. The same principles can be used to perform Mealy synthesis of functions specified in mod-2 arithmetic (cf. [5, Ch. 3]). We expect that the method can also be extended to include bitstream functions specified in the Boolean bitstream algebra, respectively Kleene bitstream algebra, described in [12] alongside the 2-adic and mod-2 bitstream algebras. In [12], various "mixed laws" are proved, that is, identities that involve operators from several of the abovementioned bitstream algebras. But although a Mealy machine of "mixed expressions" can be defined (using the defining stream differential equations of the various operators), it is not clear how to find mixed laws that provide a complete equational axiomatisation of equivalence of mixed expressions.

We mentioned already the similarity between our work and Brzozowski's construction [3]. A notable difference is that there is no analogue of the classic Kleene theorem in our setting. More precisely, there are finite Mealy machines that realise a bitstream function which cannot be specified by any rational function expression (see [5, Ch. 3]). Given the numeric nature of rational function expressions and their level of abstraction, we do not find this is so surprising. However, the Kleene theorem does have a coalgebraic generalisation, as shown in [2], where a notion of regular expression for polynomial coalgebras is defined. Polynomial coalgebras can be thought of as generalised labelled transition systems (which, in particular, include Mealy machines), and their associated regular expressions are similar to process algebraic expressions. One direction of this generalised Kleene theorem is obtained by defining coalgebraic structure on the set of expressions such that from an expression e, a finite coalgebra can be constructed as a generated subcoalgebra (modulo a suitable congruence).

As the above suggests, the idea of generating behaviour from syntax is possible at a very general level. Such interplay between algebra (syntax) and coalgebra (behaviour) can often be captured by so-called bialgebras for a distributive law (cf. [1]). Again, the case of regular expressions and deterministic automata form an example of this much more abstract setup (cf. [9]). Another example is found in the area of process algebra where the format of structural operational rules corresponds to a distributive law (cf. [14, 1]). Such a bialgebraic picture also exists for function expressions and Mealy machines, although in a slightly less direct way compared with regular expressions and deterministic automata. A paper reporting on this result is currently under preparation.

References

[1] F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats.* PhD thesis, Vrije Universiteit Amsterdam, 2004.

[2] M.M. Bonsangue, J.J.M.M. Rutten, and A.M. Silva. A Kleene theorem for polynomial coalgebras. In *Proceedings of FoSSaCS 2009*, volume 5504 of *Lecture Notes in Computer Science*, pages 122–136, 2009.

[3] J.A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, 1964.

[4] F.Q. Gouvêa. *p-adic Numbers: An Introduction*. Springer, 1993.

[5] H.H. Hansen. *Coalgebraic Modelling: applications in automata theory and modal logic*. PhD thesis, Vrije Universiteit Amsterdam, 2009.

[6] H.H. Hansen and D. Costa. DIFFCAL. Tool webpage (source code, documentation, executable) currently available at: `http://homepages.cwi.nl/~costa/projects/diffcal`, 2005. Implementation of Mealy synthesis algorithm described in [7].

[7] H.H. Hansen, D. Costa, and J.J.M.M. Rutten. Synthesis of Mealy machines using derivatives. In *Proceedings of CMCS 2006*, volume 164(1) of *Electronic Notes in Theoretical Computer Science*, pages 27–45. Elsevier Science Publishers, 2006.

[8] E.C.R. Hehner and R.N. Horspool. A new representation of the rational numbers for fast easy arithmetic. *SIAM Journal on Computing*, 8:124–134, 1979.

[9] B. Jacobs. A bialgebraic review of deterministic automata, regular expressions and languages. In *Algebra, Meaning and Computation: Essays dedicated to Joseph A. Goguen on the Occasion of his 65th Birthday*, volume 4060 of *Lecture Notes in Computer Science*, pages 375–404. Springer, 2006.

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

[11] J.J.M.M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata and power series. *Theoretical Computer Science*, 308(1):1–53, 2003.

[12] J.J.M.M. Rutten. Algebra, bitstreams, and circuits. In *Proceedings of AAA68*, volume 16 of *Contributions to General Algebra*, pages 231–250. Verlag Johannes Heyn, Klagenfurt, 2005.

[13] J.J.M.M. Rutten. Algebraic specification and coalgebraic synthesis of Mealy machines. In *Proceedings of FACS 2005*, volume 160 of *Electronic Notes in Theoretical Computer Science*, pages 305–319, 2006.

[14] D. Turi and G.D. Plotkin. Towards a mathemathical operational semantics. In *Proceedings of LICS 1997*, pages 280–291. IEEE Computer Society, 1997.

Verification using Parameterised Boolean Equation Systems

Tim A.C. Willemse

Eindhoven University of Technology – The Netherlands
T.A.C.Willemse@TUE.nl

Abstract. In this note, we illustrate how parameterised Boolean equation systems can be employed for verifying system correctness. Several small examples are used to give a flavour of the more common techniques available for conducting typical verifications in this setting. We finish with a short discussion of the research challenges that lie ahead.

1 Introduction

Model checking and equivalence checking have become established techniques for analysing system behaviours. In the latter setting, the correctness of an implementation is established by proving that it is behaviourally equivalent to another, trusted model (the *specification*), using an appropriate notion of equivalence. All artefacts are within the same formalisms. This starkly contrasts model checking, in which the specification is assumed to be some requirement or property, formalised in some temporal logic, and the validity of the specification is checked against a specific implementation.

For finite state systems, model checking and equivalence checking are decidable for a wide variety of temporal logics and behavioural equivalences. The system behaviours are often generated from concise descriptions that represent the system behaviours. When represented explicitly, the latter tend to become extremely large, a phenomenon known as the *state space explosion problem*. As a result, and in spite of the decidability of the problem, practical verification may break down nevertheless. Ideally, automated verification is therefore done at the system description level, as it might offer the means to mitigate the state space explosion problem.

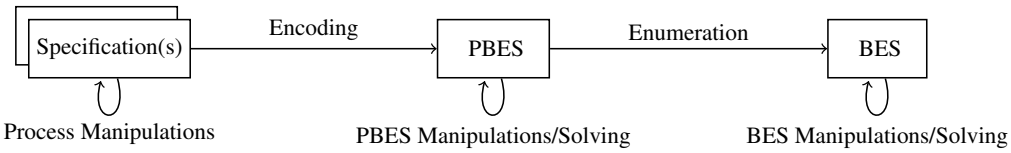


Fig. 1. Verification methodology using Parameterised Boolean Equation Systems.

In this note, we focus on a methodology for the verification of finite- and infinite-state systems that is capable of taking advantage of such system descriptions. A particular strong point of the methodology is its versatility:

1. it supports *both* model checking and equivalence checking;
2. it covers the full spectrum of automation: it offers techniques that can be applied manually, mechanically/assisted to fully automatically;
3. it allows for manipulations of all artefacts at each stage in the verification methodology.

The methodology combines *Linear Process Equations* (LPEs) [5] for describing system behaviours, *first-order modal μ-calculus* [11, 4] for specifying system requirements and *parameterised Boolean equation systems* (PBESs) [6, 11], with the special subclass of *Boolean equation systems* [10], for encoding and solving the various verification problems. The latter encodings and transformations are fully automated, see *e.g.* [4, 7, 2]. A schematic overview of the approach is shown in Figure 1.

The richness of our problem setting immediately implies a loss of decidability of many of our techniques. In practice, however, many practical problems do not fit in the confines of decidable theories, but can be solved nevertheless. As a matter of fact, the manipulations of the LPEs and the PBESs offer an additional array of advantages, even for problems that are known to be decidable. A number of well-understood manipulations for LPEs have become readily accessible, allowing one to manipulate and minimise the state space of a system prior to actual verification. PBES technology has followed a similar trend, in which a number of manipulations have been studied for simplifying a given PBES, up-to the point of solving these. These include include manual and mechanic techniques such, *e.g.*, *symbolic approximation* [7] and *pattern matching* [6], and fully automated techniques, such as static analysis techniques [13]. *Enumeration* [3], *i.e.*, the process of moving from a PBES to a BES, is inspired by explicit-state verification methodologies. It is particularly useful for automated verification: solving BESs is decidable (see *e.g.* [10]) and can be done efficiently in many practical applications, for instance by translating them to parity games and using state-of-the-art algorithms for solving these [15, 16, 18]. Also, recently developed transformations can reduce the size of a BES considerably, which can speed up the process of finding the solution [8, 14].

In this note, we focus on small examples that give a flavour of the complexity of the type of verification problems that can be dealt with in our methodology, and some of the techniques that exist for dealing with these. We finish with, what we believe, are promising research directions for further improving our methodology.

2 Preliminaries: LPEs and the Modal μ-Calculus

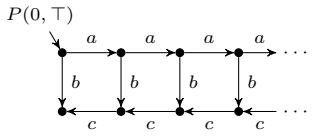
We assume the reader has some familiarity with process theory; in particular, given the introductory flavour of this note, we largely focus on syntax and do not dwell on semantics. Our setting is *action-based*, *i.e.*, events are the primary objects of concern, causing state changes. Process behaviour is described by means of a high-level syntax, *viz.*, a *Linear Process Equation*. This is an equation of the following form:

$$P(d{:}D) = \sum\{ \sum_{e_i:E_i} c_i(d,e_i) \rightarrow a_i(f_i(d,e_i)) \cdot P(g_i(d,e_i)) \mid i \in I\}$$

Here, I is a finite set of indices, $a_i : D_{a_i}$, for $i \in I$ ranges over some finite set of sorted action names $\mathcal{A}ct$, and D and E_i can be complex sorts; $c_i : D \times E_i \rightarrow B$ is a Boolean function, $f_i : D \times E_i \rightarrow D_{a_i}$ is an action parameter function that returns an argument of the sort of action a_i and $g_i : D \times E_i \rightarrow D$ is a state manipulation function. We sometimes omit the sorts of the actions if these are deemed unimportant. Intuitively, d represents the current state, which can non-deterministically evolve to a new state $g_i(d, e_i)$ (for non-deterministically chosen values for e_i), provided that action a_i is enabled (*i.e.*, condition $c_i(d, e_i)$ should evaluate to true). Given some value d_0 of sort D, process $P(d_0)$ induces a labelled transition system, see *e.g.* [5], as illustrated by the example below.

Example 1. Consider an infinite-state process that can perform an arbitrary number of a actions, then performs a b action and then performs as many c actions as a actions that were performed. Assume sorts B and N, representing the Booleans (with elements $\top$ (true) and $\bot$ (false)) and the natural numbers, respectively. A partial visualisation of this process, and the LPE are given below:

$$\begin{aligned} & P(n{:}N, d{:}B) \\ = {} & d \longrightarrow a \cdot P(n+1, d) \\ + {} & d \longrightarrow b \cdot P(n, \neg d) \\ + {} & \neg d \wedge n > 0 \longrightarrow c \cdot P(n-1, d) \end{aligned}$$



Model checking allows one to answer whether a process description satisfies some requirement phrased in a language of temporal or modal logic. One of the most basic modal logics is *Hennessy-Milner* logic, which adds the operators $\langle a \rangle \phi$ and $[a]\phi$ to propositional logic, with the intended meaning that a state satisfies $\langle a \rangle \phi$, if it has an a-successor that satisfies ϕ, and a state satisfies $[a]\phi$ if all its direct a-successors (if any) satisfy ϕ. Adding least and greatest fixed points to this language leads to one of the more complex and

powerful modal logics, *viz.*, the *propositional modal μ-calculus*, see *e.g.* [9]. In this note, we consider a first-order extension of this language, defined by the following grammar:

$$\alpha ::= b \mid a(e) \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \forall d{:}D.\ \alpha \mid \exists d{:}D.\ \alpha$$
$$\phi ::= b \mid X(e) \mid \phi \wedge \phi \mid \phi \vee \phi \mid \forall d{:}D.\ \phi \mid \exists d{:}D.\ \phi$$
$$\mid [\alpha]\phi \mid \langle\alpha\rangle\phi \mid \mu X(d{:}D := e).\ \phi \mid \nu X(d{:}D := e).\ \phi$$

Here, b is a Boolean expression, $a(e)$ is an action with parameter e and $X(e)$ is a recursion variable with parameter e; such a variable X represents a total function from some domain D to a set of states in a given process. The formulae generated by α are *action formulae*: logic is used to describe "interesting" sets of actions of a process. For instance, the action formula $\top$ indicates the entire set of actions, whereas $\exists n{:}N.\ a(n)$ describes the set of a-actions with arbitrary natural number parameter. Ignoring the first-order constructs, the state formulae ϕ are more or less in line with the state formulae from the modal μ-calculus, except for the parameterisation of the fixed point formulae. Typically, a formula $\mu X(d{:}D := e).\ \phi$ specifies a fixed point for each value of the parameter d (which is initialised to e). Thus, with some leniency for notation, $\mu X(c{:}B := \top).\ \phi$ could be understood as the value X_1 in the simultaneous fixed point expression $\mu(X_0, X_1).\ (\phi[d := \bot], \phi[d := \top])[X(\bot) := X_0, X(\top) := X_1]$. Formally, a (first-order) modal μ-calculus formula identifies a set of states in a given labelled transition system, see *e.g.* [11, 4, 6].

Example 2. Referring to process P of the previous example, one might wish to check whether it is possible to perform an infinite number of a actions. Computing the set of states that satisfy the following formula answers this question:

$$\nu X.\langle a\rangle X \tag{1}$$

Similarly, verifying whether along every a path, always a b action is attainable, requires checking:

$$\nu V.\ ([a]V \wedge \mu W.\ (\langle a\rangle W \vee \langle b\rangle\top)) \tag{2}$$

Illustrating the use of parameterisation of fixed points: verifying that the number of c actions following the b action is exactly the same as the number of a actions that preceded the b action requires checking:

$$\nu Y(i{:}N := 0).[a]Y(i+1) \wedge [b](\mu Z(j{:}N := 0).\ (([c]Z(j+1) \wedge \langle c\rangle\top) \vee (j = i \wedge [c]\bot))) \tag{3}$$

The above formula relies on the data parameters i and j to count the number of a and c actions.

3 Equation Systems

Verifying whether a μ-calculus formula holds for a given process is possible in case the process has a finite number of states, although this is not sufficient: depending on the data occurring in the formula, the problem might or might not be computable. For instance, the value for i in formula (3) can grow unbounded, even when interpreted on a finite state process.

Parameterised Boolean Equation Systems (or *equation systems* for short) are finite sequences of fixed point equations, where each equation is of the form $\big(\mu X(d{:}D) = \phi\big)$ or $\big(\nu X(d{:}D) = \phi\big)$. The left-hand side of each equation consists of a fixed point symbol, where μ indicates a least and ν a greatest fixed point, and a *sorted predicate variable* X representing formulae from some domain D to the Booleans. The right-hand side of each equation is a *predicate formula*, given by the following grammar:

$$\phi ::= b \mid X(e) \mid \phi \wedge \phi \mid \phi \vee \phi \mid \forall d{:}D.\ \phi \mid \exists d{:}D.\ \phi$$

The encodings of [4, 6] allow one to automatically construct an equation system given an LPE and a μ-calculus formula, ensuring that the equation system codes in which states of the LPE the μ-calculus formula is true. Observe that one needs to solve the equation system first, meaning that one needs to eliminate all occurrences of predicate variable instances $X(e)$ from the right-hand sides of the equations.

Example 3. The equation system encoding whether process $P(0, \top)$ satisfies formula (1) is as follows:

$$\big(\nu X(n{:}N, d{:}B) = d \wedge X(n+1, d)\big) \tag{4}$$

The validity of property (2) is encoded by the following equation system:

$$\begin{array}{l}\big(\nu V(n{:}N, d{:}B) = (d \implies V(n+1, d)) \wedge W(n, d)\big)\\ \big(\mu W(n{:}N, d{:}B) = d \vee (d \wedge W(n+1, d))\big)\end{array} \tag{5}$$

Finally, encoding the validity of property (3) for process $P(0, \top)$ leads to the following equation system:

$$\begin{array}{l}\big(\nu Y(n{:}N, d{:}B, i{:}N) = (d \implies Y(n+1, d, i+1)) \wedge (d \implies Z(n, \neg d, i, 0))\big)\\ \big(\mu Z(n{:}N, d{:}B, i{:}N, j{:}N) = (i = j \wedge (d \vee n = 0)) \vee (Z(n-1, d, i, j+1) \wedge \neg d \wedge n > 0)\big)\end{array} \tag{6}$$

A special fragment of equation systems is the subset of *Boolean equation systems*: equation systems in which the right-hand sides of the equations are void of first-order constructs and data, and the left-hand sides consist of non-parameterised proposition variables. This fragment is of particular interest since computing the solution to Boolean equation systems is decidable. The process of transforming an equation system into a Boolean equation system is akin to a state space exploration, see [3].

Example 4. Consider the equation system of (4). Assume one is interested in the value for $X(0, \top)$. For each $X(i, b)$, we introduce a fresh proposition variable $X_{i,b}$. Then one can generate the following (infinite) sequence of equations:

$$(\nu X_{0,\top} = \top \wedge X_{1,\top})\ (\nu X_{1,\top} = \top \wedge X_{2,\top})\ (\nu X_{2,\top} = \top \wedge X_{3,\top}) \cdots$$

In many practical cases, such an exploration terminates, after which the Boolean equation system can be solved, leading to a partial solution for the original equation system. Although in the above example, the exploration does not terminate, one can still transform the equation system into one for which termination is guaranteed. Some of these techniques are discussed in the next section.

A word on semantics. Predicate formulae ϕ are interpreted in the context of environments η for predicate variables and ε for data variables, notation $[\![\phi]\!]\eta\varepsilon$. An equation $(\sigma X(d{:}D) = \phi)$ can be interpreted as a σ-fixed point over the set of functions with semantic domain $\mathbb{D}$ and co-domain $\mathbb{B}$. We write $\phi_{\langle d\rangle}$, for the (syntactic) functional $(\lambda d{:}D.\ \phi)$. The interpretation of $\phi_{\langle d\rangle}$, denoted $[\![\phi_{\langle d\rangle}]\!]\eta\varepsilon$, is given by the functional $(\lambda v{\in}\mathbb{D}.\ [\![\phi]\!]\eta\varepsilon[v/d])$. The set of (total) functions $f{:}\mathbb{D} \to \mathbb{B}$, denoted by $\mathbb{B}^{\mathbb{D}}$, equipped with the point-wise ordering $\sqsubseteq$ is a complete lattice. Assuming that the domain of the predicate variable X is of sort D, the functional $[\![\phi_{\langle d\rangle}]\!]\eta\varepsilon$ induces a monotone predicate formula transformer $\lambda g{\in}\mathbb{B}^{\mathbb{D}}.\ ([\![\phi_{\langle d\rangle}]\!]\eta[g/X]\varepsilon)$. The existence of extremal fixed points of such transformers is guaranteed by Tarski's celebrated fixed point Theorem [17].

Definition 1. *The* solution of an equation system *in the context of a predicate environment η and a data environment ε is inductively defined as follows, for any $\mathcal{E}$:*

$$\begin{array}{lll}[\![\epsilon]\!]\eta\varepsilon & =_{def} & \eta\\ [\![(\sigma X(d{:}D) = \phi)\mathcal{E}]\!]\eta\varepsilon & =_{def} & [\![\mathcal{E}]\!](\eta[\sigma f \in \mathbb{B}^{\mathbb{D}}.\ [\![\phi_{\langle d\rangle}]\!]([\![\mathcal{E}]\!]\eta[f/X]\varepsilon)\varepsilon/X])\varepsilon.\end{array}$$

The definition of a solution is rather complex; this is largely due to its use of the tree-like recursion. Nevertheless, the solution of an equation system has the following characteristics:

1. It respects the equivalences of each equation semantically;
2. It prioritises the fixed point signs of equations that come first over the signs of equations that follow.

As a result, the solution is sensitive to the order of equations in an equation system: the solution to $(\mu X = Y)(\nu Y = X)$, which is $\bot$ for X and Y differs from that of $(\nu Y = X)(\mu X = Y)$, which assigns $\top$ to X and Y.

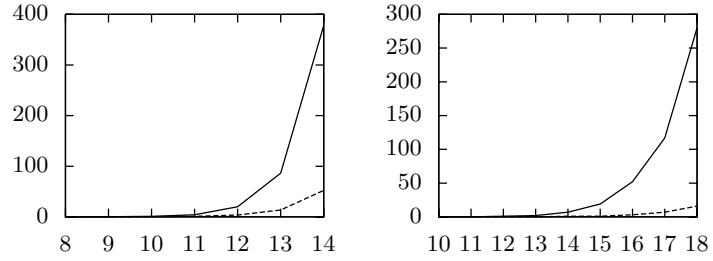


Fig. 2. The effect of constant parameter elimination on the instantiation time required to obtain a BES from an equation system encoding the deadlock-freedom property for n dining philosophers (left) and for n Milner schedulers (right). Time is measured in minutes (y-axis). Solid line: time for exploring the original equation system. Dotted line: time for exploring the equation system after constant elimination.

4 Manipulating Equation Systems

Given any equation $(\sigma X(d{:}D) = \phi)$, one can substitute ϕ with an arbitrary formula ψ that is logically equivalent, *i.e.*, $[\![\phi]\!]\eta\varepsilon = [\![\psi]\!]\eta\varepsilon$ for all environments. Simplifying the predicate formulae prior to exploring the equation system can significantly speed up the generation of the underlying Boolean equation system. Using *invariants* — a set of formulae (one per equation) that characterise the relation between left-hand side data parameters, identifying a closed subset of Boolean equations underlying the equation system — one can strengthen the right-hand sides of an equation system, thereby increasing the likelihood of being able to simplify the strengthened formulae. This, however, defers the problem of rewriting to the problem of detecting invariants, which, in general, is quite hard. Nevertheless, specific classes of invariants can be detected efficiently. Among these, we find invariants of the form $\bigwedge_{i\in I} d_i = e_i$, where each d_i is a data parameter occurring at the left-hand side of an equation, and e_i is an actual concrete value for this parameter. Such invariants allow one to actually substitute the value e_i for each occurrence of variable d_i in the right-hand side of a given equation. Likewise, one can use invariants of the form $\bigwedge_{i\in I, j\in J} d_i = d_j$, to replace the parameters d_j by d_i in the right-hand side of a given equation.

Example 5. Consider equation (4). Suppose we are interested in the solution to $X(0, \top)$. We find that $d = \top$ is in fact an invariant for the equation; hence, equation (4) can be rewritten to:

$$\big(\nu X(n{:}N) = X(n+1)\big) \tag{7}$$

Likewise, when we are interested in the solution to $V(0, \top)$ in equation system (5), we find that $d = \top$ is an invariant for both equations, allowing equation system (5) to be rewritten to:

$$\big(\nu V(n{:}N) = V(n+1) \wedge W(n)\big)\ \big(\mu W(n{:}N) = \top\big) \tag{8}$$

Finally, in case we are interested in the solution to $Y(0, \top, 0)$, defined by equation system (6), we find that $d = \top$ is an invariant for Y and $d = \bot$ is an invariant for Z. This reduces (6) to:

$$\begin{array}{l} \big(\nu Y(n{:}N, i{:}N) \quad\ = Y(n+1, i+1) \wedge Z(n, i, 0)\big) \\ \big(\mu Z(n{:}N, i{:}N, j{:}N) = (i = j \wedge n = 0) \vee (Z(n-1, i, j+1) \wedge n > 0)\big) \end{array} \tag{9}$$

Figure 2 illustrates the effect that such a constant detection can have on the time it requires to explore an equation system in an on-the-fly manner: the exploration times in these examples reduce dramatically. It should be noted that invariants do not reduce the size of a Boolean equation system; they merely help in speeding up rewriting. In contrast, some data parameters of an equation system do not manifest themselves in the solution to the equation system, while they can still be a major source for the blow-up in size of the Boolean equation system. For instance, in equation (7), one can show that the solution to $X(n)$ is $\top$, regardless of the value for n; in a sense, n is redundant.

A semantic analysis of redundancy is not feasible for most complex equation systems: it would require one to first solve the equation system. As a next best solution, one can approximate the set of redundant

parameters by means of an analysis of the predicate formulae occurring in an equation system. By constructing a graph of all the (syntactic) influences one parameter exerts on another parameter's values, one can efficiently deduce a set of parameters that do not influence the solution of an equation system at all. These can be removed immediately; the entire analysis is easily automated, see [13].

Example 6. Consider equation (7). A syntactic analysis of redundancy reveals that parameter n cannot influence the solution to X. As a result, equation (7) can be rewritten to:

$$\big(\nu X = X\big) \tag{10}$$

This equation is a Boolean equation, which is easily seen to have solution $X = \top$, see [10], so, following [13], we know that $X(n) = \top$ for all n. Parameter n is equally redundant in equation system (8), allowing this equation system to be rewritten to:

$$\big(\nu V = V \wedge W\big)\ \big(\mu W = \top\big) \tag{11}$$

Again, the resulting equation system belongs to the decidable fragment of equation systems; computing its solution leads to the answer $V = \top$ and $W = \top$. Note that none of the parameters in equation system (9) can be identified as redundant using this technique. In fact, the solution to this equation system cannot (yet) be computed automatically. An inspection by hand reveals that $n = i$ is an invariant for Y, whereas $j = i - n$ is an invariant for Z. Simplifying equation system (9) using these invariants leads to:

$$\big(\nu Y(n{:}N) = Y(n+1) \wedge Z(n,n)\big)\ \big(\mu Z(n{:}N, i{:}N) = (n > 0 \implies Z(n-1,i))\big) \tag{12}$$

Parameter i can subsequently be removed, as it turns out to be redundant:

$$\big(\nu Y(n{:}N) = Y(n+1) \wedge Z(n)\big)\ \big(\mu Z(n{:}N) = (n > 0 \implies Z(n-1))\big) \tag{13}$$

The resulting equation system immediately renders to one of the patterns identified in [7], introducing an existential quantifier and removing the recursion:

$$\big(\nu Y(n{:}N) = Y(n+1) \wedge Z(n)\big)\ \big(\mu Z(n{:}N) = \exists j{:}N.\ 0 = n - j\big) \tag{14}$$

The equation for Z can be replaced by $\top$ automatically, using a one-point rule for existential quantification. This allows one to further remove the parameter n in both equations, leading to

$$\big(\nu Y = Y \wedge Z\big)\ \big(\mu Z = \top\big) \tag{15}$$

Solving the above Boolean equation system leads to the answer $Y = Z = \top$. From this, we can deduce that process $P(0, \top)$ indeed satisfies property (3). The above computations show that one can verify complex formulae on infinite processes using a variety of tools and techniques, switching between automated and manual techniques at will.

An example of the effect that an analysis of redundancy can have in practice is shown in Table 1, taken from [13]. It shows the size of the generated Boolean equation systems before and after removing redundant parameters; the equation systems encode a model checking problem on typical communication protocols.
We finish this section with a more complex example, taken from [7]. It nicely illustrates the power of the first-order extensions of the modal μ-calculus.

Example 7. Consider a process that merges two streams of natural numbers, producing a locally ascending output sequence. A formal description of our system is given by the linear process below. The initial state of the system is described by process $\mathit{Merge}(0,0,0)$. In the LPE, the parameter σ is a state parameter; for instance, $\sigma = 1$ indicates that the process has to read from the first stream, whereas $\sigma = 2$ indicates that the merger must read stream two. The values, read from the streams, are stored in parameters i_1 and i_2.

$$\begin{aligned}
&\mathit{Merge}(\sigma, i_1, i_2 : N)\\
=\ &\sum m : N.\ \sigma = 0 \to r_1(m) \cdot \mathit{Merge}(2, m, i_2)\\
+\ &\sum m : N.\ \sigma = 0 \to r_2(m) \cdot \mathit{Merge}(1, i_1, m)\\
+\ &\sum m : N.\ \sigma = 1 \to r_1(m) \cdot \mathit{Merge}(3, m, i_2)\\
+\ &\sum m : N.\ \sigma = 2 \to r_2(m) \cdot \mathit{Merge}(3, i_1, m)\\
+\ &\sum m : N.\ \sigma = 3 \wedge i_1 \leq i_2 \to s(i_1) \cdot \mathit{Merge}(1, i_1, i_2)\\
+\ &\sum m : N.\ \sigma = 3 \wedge i_2 \leq i_1 \to s(i_2) \cdot \mathit{Merge}(2, i_1, i_2)
\end{aligned}$$

Table 1. The effect of redundancy elimination on the size of the resulting BES; we write x/y to indicate the size of the induced BES without (x) and with (y) elimination of redundancy.

Property: Possibly infinitely often receive a constant message $m \in M$ ($\nu X.\mu Y.\langle r(m)\rangle X \vee \langle \neg r(m)\rangle Y$)

$\lvert M\rvert \rightarrow$ Protocol $\downarrow$	2	4	8	∞
ABP	77 / 41	149 / 41	293 / 41	∞ / 41
PAR	95 / 51	183 / 51	359 / 51	∞ / 51
CABP	513 / 257	1,121 / 257	2,721 / 257	∞ / 257
One-bit SWP	379 / 181	991 / 181	3,079 / 181	∞ / 181
SWP (buffer size 2)	17,809 / 2,545	163,393 / 2,545	$1.9 * 10^6$ / 2,545	∞ / 2,545
SWP (buffer size 4)	$3.5 * 10^6$ / 64,609	nc / 64,609	nc / 64,609	∞ / 64,609

There is some non-determinism in the process' behaviour when the values last read over both streams are equal: one of these values is sent and the stream from which it came is read from subsequently. Suppose we wish to validate that the system produces an ascending output stream when offered two ascending input streams. This is expressed by the following formula:

$$\begin{aligned}(\nu X(j_1, j_2, o{:}N := 0, 0, 0).\forall l{:}N.\ &([r_1(l)](l \geq j_1 \Rightarrow X(l, j_2, o)) \wedge \\ &[r_2(l)](l \geq j_2 \Rightarrow X(j_1, l, o)) \wedge \\ &[s(l)](l \geq o \wedge X(j_1, j_2, l))))\end{aligned}$$

Combining the resulting linear process with the above formula, and after applying some simplifications, we obtain the below equation.

$$\begin{aligned}\nu X(\sigma, i_1, i_2, j_1, j_2, o{:}N) = \forall l{:}N.\ &(\sigma = 0 \Rightarrow l \geq j_1 \Rightarrow X(2, l, i_2, l, j_2, o)) \\ &\wedge(\sigma = 0 \Rightarrow l \geq j_2 \Rightarrow X(1, i_1, l, j_1, l, o)) \\ &\wedge(\sigma = 1 \Rightarrow l \geq j_2 \Rightarrow X(3, i_1, l, j_1, l, o)) \\ &\wedge(\sigma = 2 \Rightarrow l \geq j_1 \Rightarrow X(3, l, i_2, l, j_2, o)) \\ &\wedge((\sigma = 3 \wedge i_1 \leq i_2) \Rightarrow (i_1 \geq o \wedge X(1, i_1, i_2, j_1, j_2, i_1))) \\ &\wedge((\sigma = 3 \wedge i_2 \leq i_1) \Rightarrow (i_2 \geq o \wedge X(2, i_1, i_2, j_1, j_2, i_2))))\end{aligned}$$

Whenever $X(\sigma, i_1, i_2, 0, 0, 0)$ holds, we find that process $Merge(\sigma, i_1, i_2)$ satisfies the required property. A closer inspection of the equation reveals the following invariant:

$$i_1 = j_1 \wedge i_2 = j_2 \wedge o \leq \min(i_1, i_2)$$

The equation can be put into the shape of a recently identified pattern, see [12]. As a result, we conclude that $X(\sigma, i_1, i_2, j_1, j_2, o)$ is true for all values of the data parameters that satisfy the invariant. From this, it follows that $Merge(0, 0, 0)$ satisfies the ascendingness property.

5 Research Challenges

Equation system (9) and Example 7 are a point in case where automation in the computation of the solutions is largely lacking. Improvements should come from two complementary approaches: symbolic manipulations and through the use of brute force.

The redundancy elimination and the constant detection of the previous section are typical examples of symbolic manipulations: based on the outcomes of a syntactic scrutiny of an equation system, it is transformed into one that is of smaller complexity. These techniques take their inspiration from techniques that have been developed for analysing linear processes. A technique that has led to significant reductions in the complexity of linear processes is *confluence* reduction. Porting this technique to equation systems, however, has thus far proved to be elusive. Part of the challenge is that —unlike linear processes— equation systems lack a behavioural interpretation. Other challenges include devising methods for dealing with dense data domains that occur in equation systems when verifying real-time systems.

The semantics of an equation system has turned out to offer little support in proving the correctness of manipulations on equation systems: it is not uncommon for these to be quite arduous. There is a clear need for a more suitable proof methodology, better supporting such correctness proofs. A likely side effect of such a proof methodology is a better understanding of the framework of equation systems.

Symbolic manipulations rarely solve a PBES completely; most likely, a transformation to a Boolean equation system is needed at some point. The sizes of the Boolean equation systems in Table 1 are a good indication of what one can expect in practice. One should therefore invest in parallelising this exploration process, taking inspiration from recent advances in the parallelisation of state space exploration, see *e.g.* [1]. Unlike state space generation, in which the state space is the artefact of interest, exploration of an equation system yields a Boolean equation system that still has to be solved. The ultimate challenge lies in combining both tasks, *i.e.*, generate and solve the Boolean equation system on-the-fly in a parallel setting. Even parallelising only the process of solving a Boolean equation system may fuel the debate of what the most effective (parallelisable) algorithm is, given a particular application domain. In the end, parallel algorithms may even shed new light on the true complexity of solving Boolean equation systems.

References

1. S.C.C. Blom, B. Lisser, J.C. van de Pol, and M. Weber. A database approach to distributed state-space generation. *Journal of Logic and Computation*, 2009.
2. T. Chen, B. Ploeger, J. van de Pol, and T.A.C. Willemse. Equivalence checking for infinite systems using parameterized Boolean equation systems. In *Proc. CONCUR'07*, LNCS, 2007.
3. A. van Dam, B. Ploeger, and T.A.C. Willemse. Instantiation for parameterised Boolean equation systems. In *Proc. ICTAC'08*, LNCS. Springer-Verlag, 2008.
4. J.F. Groote and R. Mateescu. Verification of temporal properties of processes in a setting with data. In *Proc. AMAST*, LNCS 1548, pages 74–90. Springer, 1999.
5. J.F. Groote and M.A. Reniers. Algebraic process verification. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 17, pages 1151–1208. Elsevier (North-Holland), 2001.
6. J.F. Groote and T.A.C. Willemse. Model-checking processes with data. *Sci. Comput. Program*, 56(3):251–273, 2005.
7. J.F. Groote and T.A.C. Willemse. Parameterised Boolean equation systems. *Theor. Comput. Sci.*, 343(3):332–369, 2005.
8. J.J.A. Keiren and T.A.C. Willemse. Bisimulation minimisations for Boolean equation systems. In *Proc. HVC'09*. Springer, 2010. To appear.
9. D. Kozen. Results on the propositional mu-calculus. *Theor. Comp. Sci*, 27:333–354, 1983.
10. A. Mader. *Verification of Modal Properties Using Boolean Equation Systems*. PhD thesis, Technische Universität München, 1997.
11. R. Mateescu. Local model-checking of an alternation-free value-based modal mu-calculus. In *Proc. VMCAI*, 1998.
12. S. Orzan and T.A.C. Willemse. Invariants for parameterised Boolean equation systems. *Theor. Comput. Sci.*, 411(11-13):1338–1371, 2010.
13. S.M. Orzan, W. Wesselink, and T.A.C. Willemse. Static analysis techniques for parameterised Boolean equation systems. In *Proc. TACAS'09*, volume 5505 of *LNCS*, pages 230–24. Springer, 2009.
14. M.A. Reniers and T.A.C. Willemse. Analysis of Boolean equation systems through structure graphs. In *Proc. SOS'09*, volume 18 of *EPTCS*, pages 92–107, 2010.
15. S. Schewe. Solving parity games in big steps. In *Proc. FSTTCS 2007*, volume 4855 of *LNCS*, pages 449–460. Springer, 2007.
16. S. Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *Proc. CSL*, volume 5213 of *LNCS*. Springer, 2008.
17. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Mathematics*, 5(2):285–309, June 1955.
18. J. Vöge and M. Jurdzínski. A discrete strategy improvement algorithm for solving parity games. In *Proc. of CAV*, volume 1855 of *LNCS*, pages 202–215. Springer, 2000.

Complete Decomposition in Constraint Systems

Some equivalences and computational properties

Cees Witteveen *

Dept. of Software Technology, Faculty EEMCS,
Delft University of Technology, The Netherlands
C.Witteveen@tudelft.nl

Abstract. Decomposition is a technique to split a problem in a number of parts such that some global property of the problem can be obtained or preserved by concurrent processing of these parts. One goal of decomposition is to find solutions to a global constraint system by distributed computations. Decomposition then should enable the merging of local solutions to a global solution and, therefore, decomposition should aim at preserving solutions. Another aim of decomposition, often encountered in the database (and sensor network) community, is to preserve consistency: as long as adding constraints to a local constraint store does not cause any inconsistencies, the consistency of the global constraint store should be preserved.
Although satisfying these preservation properties seem to require different decomposition modes, we show that in fact these properties are equivalent: whenever a decomposition is consistency preserving, it is also solution preserving and vice-versa. We then show that the complexity of finding such decompositions is polynomially related to finding solutions for the original constraint system, explaining the popularity of decomposition applied to tractable constraint systems.

1 Introduction

Background and Motivation We are interested in decomposition as a technique to provide *coordination mechanisms* for independent problem solvers. Such coordination mechanisms should ensure that each individual problem solver can solve its part of the problem independently of the others. Moreover, it should ensure that, whatever contribution is made by each individual problem solver, their joint contribution should meet some predefined criteria. Examples of problems requiring such coordination mechanisms are building a house by different subcontractors, executing reconnaissance tasks by several units, and planning of ground handling processes by several service providers at airports.

To provide a common framework for discussing decomposition, we focus upon *constraint problems*, solvable by constraint processing. The basic idea behind constraint solving is to represent combinatorial problems by a *constraint system*. The basic ingredients of a constraint system $\mathcal{S}$ are a set C of constraints over a set X of variables x_i each taking values in some domain of values D_i. A constraint system $\mathcal{S}$ is said to be

* This contribution is based on joint work with Wiebe van der Hoek and Michael Wooldridge, Department of Computer Science, University of Liverpool, UK.

solved if we have found values $d \in D_i$ for each of the variables $x_i \in X$ such that all the constraints $c \in C$ are satisfied. In artificial intelligence research, for example, constraint systems have been used to represent such diverse problems as planning and scheduling, resource allocation, design and configuration problems [Dec03]. In the database community, constraints have also been used as *integrity constraints* to ensure the integrity of data stored and manipulated.

In both areas, *distributed constraint systems* have been a major focus of research. In a distributed constraint system, one distinguishes a set of agents A_i each controlling a disjoint subset of variables $X_i \subseteq X$. Each agent A_i is responsible for those constraints whose variables occur in its control set X_i (its set of *local constraints*). Agents have to interact with other agents for solving the set of *global constraints*, whose variables occur in different components X_i. If communication between the agents is difficult or agents do not wish to communicate, enforcing these global constraints becomes an issue. Therefore, much research has been done on *decomposing* distributed constraint systems, that is, to replace each global constraint by a suitable set of local constraints, in such a way that the need for interaction during the problem solving process is eliminated. We call such a system a *completely decomposed* constraint system.

With respect to decomposition in constraint systems, however, the focus of research within the artificial intelligence (AI) community has been quite different from the focus within the database community. In AI applications one typically assumes that the local constraints are controlled by agents whose (common) task is to assign suitable values to the variables such that all constraints are satisfied. Decomposition has to ensure that the local constraints can be solved completely independently from the others, after which the local solutions can always be *merged* to yield a solution to the complete system. Hence, in AI research the focus of decomposition has been on a *solution preserving* property (see e.g. [Hun02,KL09]): in obtaining a global solution, local solutions should always be preserved in order to ensure independent local problem solving.

On the other hand, the focus in the database community has been on the use of integrity constraints for distributed databases that ensure that, whatever local information satisfying these constraints is added to the (distributed) database, the consistency of the total database will be preserved [GW93,BKV04]. Hence, in the database community, one focuses on the *preservation of consistency* in constraint systems: how to guarantee that when updating local databases and only ensuring their local consistency, the consistency of the resulting *global* constraint system (i.e. the union of all local databases + integrity constraints) will remain consistent, too.

Our focus of research In this paper, we would like to investigate this idea of using a completely decomposed constraint system to enable independent problem solving. In particular, we would like to investigate the relation between the above mentioned decomposition criteria: solution preservation and consistency preservation.

The first question that comes in mind then is: *what is the exact relationship between preservation of solutions versus preservation of consistency notions*: are these decomposition criteria equivalent, does one imply the other, or are they independent?
By analyzing the underlying notions and presenting a framework for decomposition, in this paper we will show that preserving consistency and preserving solutions are equivalent.

Having identified these two decomposition criteria, next, one would like to know how difficult is it to *find a suitable decomposition* (preserving solutions or preserving consistency), given a set of agents each controlling a part of the variables in a constraint system. Again, we will provide an answer by proving that finding a solution preserving decomposition for a constraint system is as hard as finding a solution for the constraint system as a whole.

Remark 1. We would like to remark that there are other views on decomposition in constraint systems as expressed by *structural decomposition methods* [GLS99] and by the *distributed constraint optimization* (DCOP) approach[MSTY03]. In the structural decomposition view *(i)* the structure of the problem (i.e., the set of constraints) dictates the way in which the subproblems are generated and *(ii)* in general, the decomposition will not allow the subproblems to be *independently* solvable. In the DCOP approach, the partitioning of the variables is given, but, in general, the result of decomposition is not a set of independently solvable subproblems. Our approach differs from these approaches in the sense that, unlike the structural decomposition approach, we are interested in decomposition methods that take a *given* partitioning of the variables into account. Secondly, unlike both the DCOP and structural decomposition approach, we require a *complete decomposition* of the original problem instance, that is, we would like to find a set of subproblems that can be solved *concurrently* and *independently* to obtain a complete solution to the original instance.

This paper is organised as follows. In Section 2 we discuss the necessary technical preliminaries. In Section 3 we discuss the equivalence between consistency and solution preserving decompositions. In Section 4 we show that solving a constraint system is – neglecting polynomial differences– as hard as finding a decomposition for it. Finally, in Section 5, we state some final conclusions to place this work into a broader perspective.

2 Preliminaries

In this section we briefly define constraint systems, distributed constraint systems, and decompositions of distributed constraint systems.

Constraint Systems A constraint system is a tuple $\mathcal{S} = \langle X, D, C\rangle$ where X is a (finite) set of variables, D is a set of (value) domains D_i for every variable $x_i \in X$, and C is a set of constraints over X. We assume constraints $c \in C$ to be specified as formulas over some language. To preserve generality, we don't feel the need to specify the set of allowable operators used in the constraints $c \in C$ and their interpretation. A solution σ of the system is an assignment $\sigma = \{x_i \leftarrow d_i\}_{i=1}^{n}$ to all variables in X such that each $c \in C$ is satisfied. The set of such solutions σ is denoted by $Sol(\mathcal{S})$. $\mathcal{S}$ is called *consistent* if $Sol(\mathcal{S}) \neq \emptyset$. We assume the set of solutions $Sol(\mathcal{S})$ to be anti-monotonic in the set of constraints; that is, if $\mathcal{S} = \langle X, D, C\rangle$ and $\mathcal{S}' = \langle X, D, C'\rangle$ are such that $C \subseteq C'$, then $Sol(\mathcal{S}') \subseteq Sol(\mathcal{S})$. For every $c \in C$, let $Var(c)$ denote the set of variables mentioned in c. For a set of constraints C, we put $Var(C) = \bigcup_{c\in C} Var(c)$. Given $\mathcal{S} = \langle X, D, C\rangle$, we obviously require $Var(C) \subseteq X$. If D is a set of value domains D_i for variables $x_i \in X$ and $X' \subset X$ then $D_{X'}$ is the set of value domains D_i of the variables $x_i \in X'$. Likewise, given a set of constraints C and a set of variables X', we let $C_{X'}$ denote the subset $\{c \in C \mid Var(c) \subseteq X'\}$ of constraints over X'.

Distributed constraint systems and decompositions In this paper we consider constraint systems $\mathcal{S}$ that are *distributed* [YH96]; that is, there is a set of N agents A_i, each being able to make assignments to or to add constraints over a subset X_i of the set X of variables. Here, we assume that agents do not share control over the variables, and that every variable is controlled by an agent. Hence, the collection $\{X_i\}_{i=1}^N$ constitutes a *partitioning* of X, i.e., $\bigcup_{i=1}^N X_i = X$, and, for $1 \leq i < j \leq N$, $X_i \cap X_j = \emptyset$.

Given a constraint system $\mathcal{S} = \langle X, D, C \rangle$ and a partitioning $\{X_i\}_{i=1}^N$ of X, we call the tuple $(\mathcal{S}, \{X_i\}_{i=1}^N)$ a *distributed constraint system*, derived from $\mathcal{S}$. We are particularly interested in those distributed systems $(\mathcal{S}, \{X_i\}_{i=1}^N)$ where each agent A_i, controlling the set X_i, only processes a set of constraints over X_i, and does not take into account other constraints. That effectively implies that in such a case, instead of one constraint system $\mathcal{S}$ and a partition $\{X_i\}_{i=1}^N$, we have a *set* of independent constraint systems $\mathcal{S}_i = \langle X_i, D_i, C_i' \rangle$, where each C_i' is a set of constraints over X_i, i.e., $Var(C_i') \subseteq X_i$. We call the resulting set $\mathcal{S}' = \{\mathcal{S}_i = \langle X_i, D_i, C_i' \rangle\}_{i=1}^N$ of such subsystems a *decomposed constraint system*[1]. We say that such a decomposed constraint system $\mathcal{S}'$ is consistent if $\bigcup_i C_i'$ is consistent.

In order to relate a distributed constraint system $(\mathcal{S}, \{X_i\}_{i=1}^N)$ to a decomposed constraint system $\mathcal{S}'$, we discuss two ways in which decomposed systems can be used to process the constraints in $\mathcal{S}$.

Solution preserving decompositions A decomposed system can be used to *preserve solutions* of a global constraint system: the individual solutions σ_i of the subsystems $\mathcal{S}_i$ of a decomposed system $\mathcal{S}' = \{\mathcal{S}_i = \langle X_i, D_i, C_i' \rangle\}_{i=1}^N$ can be used to compose a global solution σ for a global constraint system $\mathcal{S}$. In that case the decomposed system is said to be solution preserving if each collection of local solutions can be used to compose a global solution:

Definition 1. *Let $(\mathcal{S}, \{X_i\}_{i=1}^N)$ be a distributed constraint system.*
Then $\mathcal{S}' = \{\mathcal{S}_i = \langle X_i, D_i, C_i' \rangle\}_{i=1}^N$ is said to be a solution-preserving decomposition w.r.t. $(\mathcal{S}, \{X_i\}_{i=1}^N)$ if $\emptyset \subseteq Sol(\mathcal{S}_1) \times Sol(\mathcal{S}_2) \times \ldots \times Sol(\mathcal{S}_N) \subseteq Sol(\mathcal{S})$. $\mathcal{S}'$ is said to be strictly solution preserving *if the first inclusion is strict whenever $Sol(\mathcal{S}) \neq \emptyset$.*[2]

Example 1. Let $\mathcal{S} = \langle X, D, C \rangle$ be a constraint system where $C = \{x_1 \wedge x_2,\ x_1 \vee x_3,\ x_1 \vee x_4\}$ is a set of boolean constraints over $X = \{x_1, x_2, x_3, x_4\}$. If X is partitioned as $\{X_1 = \{x_1, x_2\}, X_2 = \{x_3, x_4\}\}$, the decomposition $\mathcal{S}' = \{\mathcal{S}_1, \mathcal{S}_2\}$ where $\mathcal{S}_1 = \langle \{x_1, x_2\}, D_1, \{x_1 \wedge x_2\} \rangle$ and $\mathcal{S}_2 = \langle \{x_3, x_4\}, D_2, \emptyset \rangle$ is a strictly solution preserving decomposition of $\mathcal{S}$: $\mathcal{S}_1$ has a unique solution $Sol(\mathcal{S}_1) = \{\{x_1 \leftarrow 1, x_2 \leftarrow 1\}\}$, while $\mathcal{S}_2$ has a "universal" solution set: $Sol(\mathcal{S}_2) = \{\{x_3 \leftarrow i,\ x_4 \leftarrow j\} : i, j \in \{0, 1\}\}$. Every solution in $Sol(\mathcal{S}_1) \times Sol(\mathcal{S}_2)$ is a solution to $\mathcal{S}$, because x_1 as well as x_2 is assigned to true. Hence, $\mathcal{S}' = \{\mathcal{S}_1, \mathcal{S}_2\}$ is strictly solution preserving.

Note that, in general, not every solution $\sigma \in Sol(\mathcal{S})$ will be obtainable as the merge of local solutions σ_i.

[1] For the moment, we do not specify any relationship between C_i' and C_{X_i}.
[2] This is needed to take care for inconsistent constraint systems.

Consistency preserving decompositions In distributed database applications one distinguishes local constraints from global (integrity) constraints. Usually, in such applications, agents are free to add constraints to their set of local constraints as long as the resulting set remains consistent. The problem then is to ensure that local consistency ensures global consistency. This global consistency has to be ensured by the set of integrity constraints. In order to prevent communication overload between the distributed sites, one often tries to distribute these integrity constraints over the sites in such a way that satisfaction of all the local versions of the constraints imply the satisfaction of the global constraints. To simplify the discussion, we concentrate on the case where each site is allowed to *add* constraints to their local store. Consistency preservation then means that the total set of original constraints + locally added constraints is consistent, whenever the added information does not cause any local inconsistency. We need the following definitions.

Definition 2. *Let $\mathcal{S} = \langle X, D, C\rangle$ be a constraint system. An extension of $\mathcal{S}$ is a constraint system $E(\mathcal{S}) = \langle X, D, C'\rangle$ where $C \subseteq C'$.*

Definition 3 (consistency preserving extensions). *Let $(\mathcal{S}, \{X_i\}_{i=1}^N)$ be a distributed constraint system, where $\mathcal{S} = \langle X, D, C\rangle$. A decomposition $\mathcal{S}' = \{\mathcal{S}_i = \langle X_i, D_i, C_i'\rangle\}_{i=1}^N$ is called* consistency preserving *w.r.t. $(\mathcal{S}, \{X_i\}_{i=1}^N)$ if the following condition holds: whenever every local extension $E(\mathcal{S}_i) = \langle X_i, D_i, C_i''\rangle$ of $\mathcal{S}_i$ is consistent, $E(\mathcal{S}) = \langle X, D, C \cup (C_1'' - C_1' \cup \ldots \cup (C_N'' - C_N')\rangle$ is consistent.*
$\mathcal{S}'$ is said to be strictly consistency preserving if, moreover, it holds that every $\mathcal{S}_i$ is consistent whenever $\mathcal{S}$ is consistent.

Example 2. Consider the constraint system specified in Example 1. It is not difficult to show that the given decomposition $\mathcal{S}' = \{\mathcal{S}_1, \mathcal{S}_2\}$ is also a strictly consistency preserving decomposition w.r.t. $(\mathcal{S}, \{X_i\}_{i=1}^N)$: Let $E(\mathcal{S}_1)$ and $E(\mathcal{S}_2)$ be two consistent extensions of $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. Since $\mathcal{S}_1$ has a unique solution $Sol(\mathcal{S}_1) = \{\{x_1 \leftarrow 1, x_2 \leftarrow 1\}\}$, it follows that $Sol(E(\mathcal{S}_1)) = Sol(\mathcal{S}_1)$. Likewise, $Sol(E(\mathcal{S}_2)) \subseteq Sol(\mathcal{S}_2)$ and $Sol(E(\mathcal{S}_2)) \neq \emptyset$. Now, take an arbitrary solution from $Sol(E(\mathcal{S}_1))$ as well as from $Sol(E(\mathcal{S}_2))$. Then, since these solutions are also solutions of $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively, by the solution preservation property, there exists a solution satisfying the original set of constraints and all constraints added to this set. Hence, the decomposition is strictly consistency preserving as well.

3 Consistency and solution preserving decompositions

Given the two preservation properties we distinguished in decompositions of constraint systems, the first question we should answer is how they are related: are they independent, is one subsumed by the other, or are they in fact equivalent?

Intuitively, it seems not hard to conclude that consistency preservation is subsumed by solution preservation: whatever information is added to a local constraint store, if the result is consistent, a solution for the global store can be found by solution preservation. Hence, there should be a global solution, and, therefore, it is not difficult to show that the global constraint store + the added constraints is a consistent set as well. More precisely:

Proposition 1. *Let $(\mathcal{S}, \{X_i\}_{i=1}^N)$ be a distributed constraint system.*
If $\mathcal{S}' = \{\mathcal{S}_i = \langle X_i, D_i, C_i' \rangle\}_{i=1}^N$ is solution preserving w.r.t. $(\mathcal{S}, \{X_i\}_{i=1}^N)$, then $\mathcal{S}'$ is also consistency preserving w.r.t. $(\mathcal{S}, \{X_i\}_{i=1}^N)$.

Proof. Assume $\mathcal{S}' = \{\mathcal{S}_i = \langle X_i, D_i, C_i' \rangle\}_{i=1}^N$ to be solution preserving w.r.t. $(\mathcal{S}, \{X_i\}_{i=1}^N)$. Then we have $Sol(\mathcal{S}_1) \times \ldots \times Sol(\mathcal{S}_N) \subseteq Sol(\mathcal{S})$. For $i = 1, 2, \ldots, N$, consider arbitrary (consistent) extensions $E(\mathcal{S}_i) = \langle X_i, D_i, C_i'' \rangle$ of the local subsystems $\mathcal{S}_i$. For each subsystem $E(\mathcal{S}_i)$, select an arbitrary assignment $\sigma_i \in Sol(E(\mathcal{S}_i))$. Since $C_i'' \supseteq C_i'$, it follows that $\emptyset \neq Sol(E(\mathcal{S}_i)) \subseteq Sol(\mathcal{S}_i)$. Hence, by solution preservation, the assignment $\sigma = \sigma_1 \sqcup \ldots \sqcup \sigma_N$ will satisfy $\mathcal{S}$. Therefore, $\sigma \models C$. By definition of σ_i and the fact that every $C_i'' - C_i'$ is a set of constraints over X_i, and the sets X_i are disjoint, it follows that $\sigma \models (C_1'' - C_1') \cup (C_2'' - C_2' \cup \ldots (C_N'' - C_N')$. Hence, $\sigma \models C \cup (C_1'' - C_1') \cup (C_2'' - C_2' \cup \ldots (C_N'' - C_N')$, therefore, $\sigma \in Sol(E(\mathcal{S}))$. So, $Sol(E(\mathcal{S})) \neq \emptyset$ and, consequently, $\mathcal{S}'$ is consistency preserving with respect to $(\mathcal{S}, \{X_i\}_{i=1}^N)$. □

Perhaps surprisingly, the converse is also true: consistency preservation implies solution preservation. The intuition behind this result is that every solution to a constraint system can be *encoded* as a special update of the constraint store. The resulting constraint store will have this solution as its unique solution. By consistency preservation, the resulting global constraint store will be consistent. Hence, this decomposition will also be solution preserving, since the merge of all local solutions will be the unique solution of the resulting system.

Proposition 2. *Let $(\mathcal{S}, \{X_i\}_{i=1}^N)$ be a distributed constraint system.*
If $\mathcal{S}' = \{\mathcal{S}_i = \langle X_i, D_i, C_i' \rangle\}_{i=1}^N$ is consistency preserving w.r.t. $(\mathcal{S}, \{X_i\}_{i=1}^N)$, then $\mathcal{S}'$ is also solution preserving w.r.t. $(\mathcal{S}, \{X_i\}_{i=1}^N)$.

Proof. Assume $\mathcal{S}' = \{\mathcal{S}_i = \langle X_i, D_i, C_i' \rangle\}_{i=1}^N$ to be consistency preserving w.r.t. $(\mathcal{S}, \{X_i\}_{i=1}^N)$. By assumption, for every subsystem $\mathcal{S}_i$ and every extension $E(\mathcal{S}_i) = \langle X_i, D_i, C_i'' \rangle$ of $\mathcal{S}_i$, it must hold that, whenever the extended local systems $E(\mathcal{S}_i)$ are consistent, then the global extended system $E(\mathcal{S}) = \langle X, D, C \cup (C_1'' - C_1') \cup \ldots \cup (C_N'' - C_N' \rangle$ is also consistent.

For each $i = 1, \ldots, N$, let σ_i be an arbitrary solution to $\mathcal{S}_i = \langle X_i, D_i, C_i' \rangle$. Since $\{X_i\}_{i=1}^N$ is a partition, the assignment $\sigma = \sigma_1 \sqcup \ldots \sqcup \sigma_N$ is well-defined. We have to show that $\sigma \in Sol(\mathcal{S})$.
For $i = 1, \ldots, N$, consider the extensions $E(\mathcal{S}_i) = \langle X_i, D_i, C_i'' \rangle$, where $C_i'' = C_i' \cup \{x = \sigma(x) : x \in X_i\}$. Then, for every $i = 1, 2, \ldots, N$, $E(\mathcal{S}_i)$ is consistent and each σ_i is the unique solution of $E(\mathcal{S}_i)$.
By consistency preservation, the extension $E(\mathcal{S}) = \langle X, D, C \cup (C_1'' - C_1' \cup \ldots \cup (C_N'' - C_N') \rangle$ is consistent, too. Hence $Sol(E(\mathcal{S})) \neq \emptyset$.

Now observe that $C \cup (C_1'' - C_1') \cup \ldots \cup (C_N'' - C_N') = C \cup \{x = \sigma(x) : x \in X\}$. Hence, it follows that σ is the *unique solution* of $E(\mathcal{S})$ and therefore, $\sigma \models C$. Hence $\sigma \in Sol(\mathcal{S})$ and the decomposition $\mathcal{S}'$ is also solution preserving. □

As a consequence of both propositions we have that a decomposition is consistency preserving iff it is solution preserving. It is not difficult to show that this equivalence

also holds for the strictly preserving versions. This immediately implies that all results that have been obtained for consistency preserving decompositions such as occur in [BKV04,MC04] can be used for solution preserving approaches to decomposition as well.

4 Finding solution preserving decompositions

The equivalence between solution preserving and consistency preserving decompositions does not tell us how we could obtain such decompositions. In this section, we will discuss the problem of finding suitable decompositions. Given the above proven equivalence, in this section we concentrate on the solution preservation property of decompositions.

In general it is not difficult to prove that deciding whether a decomposition is solution preserving is a coNP-complete problem. We can, however, obtain a more detailed result by relating the difficulty of finding a strictly solution preserving decomposition for a constraint system $\mathcal{S}$ belonging to a class of constraint systems to the difficulty of finding a solution to $\mathcal{S}$:

Proposition 3. *Let $\mathcal{C}$ be an arbitrary class of constraint systems allowing at least equality constraints. Then there exists a polynomial algorithm to find a solution for constraint systems $\mathcal{S}$ in $\mathcal{C}$ iff there exists a polynomial algorithm that, given a constraint system $\mathcal{S} \in \mathcal{C}$ and an arbitrary partition $\{X_i\}_{i=1}^N$ of X, finds a strictly solution preserving decomposition w.r.t. $(\mathcal{S}, \{X_i\}_{i=1}^N)$.*

Proof. Suppose that there exists a polynomial algorithm A to find a solution for constraint systems in $\mathcal{C}$. We show how to construct a polynomial algorithm for finding a decomposition for an arbitrary partition of such a constraint system. Let $\mathcal{S} \in \mathcal{C}$ be constraint system and $\{X_i\}_{i=1}^N$ an arbitrary partitioning of X. To obtain a decomposition $\mathcal{S}' = \{\mathcal{S}_i = \langle X_i, D_i, C_i' \rangle\}_{i=1}^N$ of $\mathcal{S}$, first, using A, we compute a solution σ of $\mathcal{S}$. For every X_i, let $C_{\sigma_i} = \{x = d \mid x \leftarrow d \in \sigma, x \in X_i\}$ be a set of unary constraints for variables in X_i directly obtained from σ. Then the subsystems $\mathcal{S}_i = (X_i, D_i, C_i')$ are simply obtained by setting $C_i' = C_{X_i} \cup C_{\sigma_i}$. Note that each of these subsystems $\mathcal{S}_i$ has a unique solution $\sigma_i = \{x \leftarrow d \in s \mid x \in X_i\}$ and the merging of these solutions σ_i equals σ, i.e. a solution to the original system $\mathcal{S}$. Clearly, $\mathcal{S}' = \{\mathcal{S}_i = \langle X_i, D_i, C_i' \rangle\}_{i=1}^N$ is a solution preserving decomposition for $\mathcal{S}$ that can be obtained in polynomial time.

Conversely, suppose we can find a strictly solution preserving decomposition $\mathcal{S}' = \{\mathcal{S}_i = \langle X_i, D_i, C_i' \rangle\}_{i=1}^N$ for a constraint system $\mathcal{S} \in C$ w.r.t any partitioning $\{X_i\}_{i=1}^N$ in polynomial time. We show how to obtain a solution σ of $\mathcal{S}$ in polynomial time.[3] Since the decomposition $\mathcal{S}' = \{\mathcal{S}_i = \langle X_i, D_i, C_i' \rangle\}_{i=1}^N$ can be obtained for *any* partitioning of X, we choose the partitioning $\{X_i\}_{i=1}^N$ where $X_i = \{x_i\}$ for $i = 1, 2, \ldots, N$. Since the decomposition can be obtained in polynomial time, it follows that $|\bigcup_{i=1\ldots N} C_i'|$ is polynomially bounded in the size of the input $\mathcal{S}$. Hence, the resulting decomposed subsystems $\mathcal{S}_i$ each consist of a polynomially bounded set of *unary* constraints. It is well

[3] The case where $\mathcal{S}$ is inconsistent is easy and omitted, here.

known that such constraint systems are solvable in polynomial time [CGJ06]. Therefore, in polynomial time for each subsystem $\mathcal{S}_i$ an arbitrary value $d_i \in D_i$ for x_i can be obtained, satisfying all constraints. Let $\sigma_i = \{x_i \leftarrow d_i\}$ denote the solution obtained for $\mathcal{S}_i$. Since $\mathcal{S}' = \{\mathcal{S}_i = \langle X_i, D_i, C_i' \rangle\}_{i=1}^{N}$ is a solution preserving decomposition, the merging $\sigma = \sigma_1 \sqcup \sigma_2 \sqcup \ldots \sqcup \sigma_N$ must be a solution of $\mathcal{S}$ as well. Therefore, σ is a solution of $\mathcal{S}$, too. Hence, given a polynomial algorithm for achieving a solution preserving decomposition, we can construct a solution $\sigma \in Sol(\mathcal{S})$ in polynomial time. □

Given the equivalence between solution preserving and consistency preserving constraint systems, we now may conclude:

Theorem 1. *Finding a strictly consistency preserving decomposition as well as finding a strictly solution preserving decomposition for a constraint system $\mathcal{S}$ is as hard as finding a solution for it.*

5 Conclusions and implications

We would like to point out that our results can be used to explain special results that have been obtained in investigating decompositions of constraint systems. First of all, in the special case of Simple Temporal Networks (STNs), Hunsberger [Hun02] essentially showed that there exists a polynomial algorithm for finding solution preserving decompositions. This result should not come as a surprise given the results we have shown above and the fact that finding a solution for STNs is solvable in polynomial time. Secondly, in [BKV04] it is shown that a safe decomposition can be easily found in case the constraints are linear arithmetic constraints. Again, this result is a simple consequence of the relationship between finding decompositions of a system $\mathcal{S}$ and finding solutions for it. Finally, there exists some work on decomposition in plan coordination [tMYWZ09], where one is looking for decompositions that enable agents to *plan* independently from each other. This work can be best conceived as finding consistency preserving decompositions, where restrictions are imposed on the type of constraints that might be added by the individual planners.

References

[BKV04] A. Brodsky, L. Kerschberg, and S. Varas. Optimal constraint decomposition for distributed databases. In M.l J. Maher, editor, *Advances in Computer Science - ASIAN 2004*, volume 3321 of *Lecture Notes in Computer Science*, pages 301–319. Springer, 2004.

[CGJ06] David A. Cohen, Marc Gyssens, and Peter Jeavons. A unifying theory of structural decompostions for the constraint satisfaction problems. In *Complexity of Constraints*. Dagstuhl Seminar Proceedings 06401, 2006.

[Dec03] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.

[GLS99] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124:2000, 1999.

[GW93] A. Gupta and J. Widom. Local verification of global integrity constraints in distributed databases. *SIGMOD Rec.*, 22(2):49–58, 1993.

[Hun02] Luke Hunsberger. Algorithms for a temporal decoupling problem in multi-agent planning. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*, pages 468–475, 2002.

[KL09] M. Karimadini and H. Lin. Synchronized Task Decomposition for Cooperative Multi-agent Systems. *ArXiv e-prints, 0911.0231K*, November 2009.

[MC04] S. Mazumbar and P.K. Chrysantis. Localization of integrity constraints in mobile databases and specification in PRO-MOTION. *Mobile Networks and Applications*, 9(5):481–490, 2004.

[MSTY03] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. An asynchronous complete method for distributed constraint optimization. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 161–168, New York, NY, USA, 2003. ACM.

[tMYWZ09] A.W. ter Mors, C. Yadati, C. Witteveen, and Y. Zhang. Coordination by design and the price of autonomy. *Journal of Autonomous Agents and Multi-Agent Systems*, (on-line version, http://dx.doi.org/10.1007/s10458-009-9086-9), 2009.

[YH96] M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Proceedings of the Second International Conference on Multiagent Systems*, pages 401–408, 1996.

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.