Nieuwsbrief van de Nederlandse Vereniging voor Theoretische Informatica

Susanne van Dam[1] Joost-Pieter Katoen Joost Kok Jaco van de Pol
Femke van Raamsdonk

Inhoudsopgave

[1] CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands. Email: Susanne.van.Dam@cwi.nl

Van de redactie en bestuur

Geachte NVTI-leden!

De NVTI nieuwsbrief van 2007 ligt voor u, of heeft u in de hand. Zoals gebruikelijk vindt u hierin de aankondiging voor de NVTI theoriedag, die dit jaar op vrijdag 9 maart in Utrecht plaatsvindt. Sprekers dit jaar zijn: Tom Ball (Microsoft), Nitin Saxena (CWI), Rineke Verbrugge (RUG) en Gerhard Woeginger (TU/e). Dit programma is naar onze smaak voldoende reden om op 9 maart naar Utrecht af te reizen.

De wetenschappelijke bijdragen dit jaar zijn van Clemens Kupke (CWI/UvA) over co-algebras, Frank de Boer (UU/CWI) over semantiek van een programmeertaal met recursie, threads en shared variables, en Valentin Robu, Koye Somefun en Han la Poutré over adaptieve strategieën in multi-agent spelen. Daarnaast wordt u uitgenodigd door Balder ten Cate, Loredana Afanasiev en Maarten Marx (UvA) om lekker te bomen. Dat de onderzoekscholen IPA en SIKS het afgelopen jaar ook niet hebben stilgezeten blijkt uit hun lijst van activiteiten. Wij danken alle auteurs voor hun bijdrage.

Na vele jaren van (zeer) trouwe dienst, hebben Hans Bodlaender en Grzegorz Rozenberg hun bestuursfunctie in 2006 neergelegd. Wij bedanken hen hartelijk voor hun inzet voor de NVTI. Mark de Berg (TU/e) is bereid gevonden zitting te nemen in het bestuur.

Tenslotte danken wij het CWI, NWO, Elsevier, IPA, SIKS en OzsL voor de sponsoring van de NVTI activiteiten, en Susanne van Dam en Anton Wijs voor de secretariële ondersteuning.

Namens de redactie en het bestuur van de NVTI,
Joost-Pieter Katoen, hoofdredacteur
Joost Kok, voorzitter
Jaco van de Pol, secretaris

Samenstelling bestuur

Invitation NVTI Theory Day, Friday March 9, 2007

We are happy to invite you for the Theory Day 2007 of the NVTI. The Dutch Assso-ciation for Theoretical Computer Science (NVTI) supports the study of theoretical computer science and its applications.

This event will take place on Friday March 9, 2007 from 9:30 until 16:45, at Hoog Brabant, Utrecht (close to Utrecht Central Station[1]).

Again, we managed to compose an interesting program with excellent speakers from The Netherlands and abroad, covering both important streams in theoretical computer science. Two lectures cover algorithms and complexity theory, and two lectures cover logic and semantics.

Lecturers

Tom Ball (Microsoft)	On the Design and Implementation of Static Analysis Tools
Gerhard Woeginger (TU/e)	Division of a shared resource
Rineke Verbrugge (RUG)	Reasoning about others in multi-agent systems
Nitin Saxena (CWI)	Identities and Complexity

Sponsors

The NVTI theory day 2007 is financially or otherwise sponsored 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), OzsL (Dutch Graduate school in Logic) and SIKS (Dutch research school for Information and Knowledge Systems).

Lunch registration

It is possible to participate in the organized lunch. Registration is required. Please register by E-mail (`Susanne.van.Dam@cwi.nl`) or by phone (020-5924189), no later than one week before the meeting (March 2, 2007). 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.

[1] From the big station hall, follow sign "Centrum". At "Radboudkwartier" it is only 100 meter. You find "Hoog Brabant" behind the info desk from shopping center "Hoog Catharijne".

Program

9.30-10.00: Arrival with Coffee
10.00-10.10: Opening
10.10-11.00: **Lecture:** Dr. Thomas Ball (Microsoft Research)
Title: On the Design and Implementation of Static Analysis Tools
11.00-11.30: Coffee/Tea
11.30-12.20: **Lecture:** Prof. Dr. Gerhard J. Woeginger
Title: Division of a shared resource
12.20-12.40: Dr. Mark Kas (NWO Exacte Wetenschappen/Physical Sciences)
It's a long way there. Developments at NWO Physical Sciences
12.40-14.10: Lunch (see below for registration)
14.10-15.00: **Lecture:** Dr. Rineke Verbrugge (RUG)
Title: Reasoning about others in multi-agent systems
15.00-15.20: Coffee/Tea
15.20-16.10: **Lecture:** Dr. Nitin Saxena (CWI)
Title: Identities and Complexity
16.10-16.40: Business meeting NVTI

Abstracts

On the Design and Implementation of Static Analysis Tools
(Thomas Ball)

At Microsoft, we now regularly apply a new generation of static analysis tools that can automatically identify serious defects in programs. These tools examine millions of lines of code every day, long before the software is released for general use. With these tools, we catch more defects earlier in the software process, enabling Microsoft to deliver more reliable systems. A number of these tools have been released for general use through Microsoft's Visual Studio integrated development environment as well as freely available development kits.

In this talk I will address the question: "How does one design and implement a static analysis tool chain to help people effectively address a software reliability problem?" In particular, I will identify a set of basic techniques that have proven very useful in constructing static analysis tools and have shown their worth through numerous applications. Experience with these techniques suggests we are approaching an exciting time when more people can contribute to the design and implementation of static analysis tools.

Division of a shared resource

(Gerhard J. Woeginger)

The talk discusses various notions of "fairness" when n processes have to share a common resource. We describe and analyze some simple protocols, and compare their various advantages and disadvantages. The quality of a protocol is usually measured in the worst-case number of queries that the protocol issues to the processes. We present some lower bound results on the number of these queries, and we discuss the trade-off between keeping this number small and reaching decent approximate fairness.

Reasoning about others in multi-agent systems

(Rineke Verbrugge)

Everyone reasons about others almost daily – when leading a team, negotiating a raise, deciding whether to send email to a group of colleagues with the 'cc' or the 'bcc' option, or deciding how to formulate feedback tactfully. Social cognition and cooperation are essential to success in human life and increasingly essential to modern computer science.

Multi-agent systems consist of dynamically cooperating computational systems, engineered to solve complex problems that require expertise and capabilities beyond the individual components. Investigations into cooperative interactions in the behavioral sciences and computer science show a marked convergence: after all, people cooperate, machines cooperate, and mixed teams consisting of software agents, robots and people cooperate, sometimes even better than people and machines separately.

The area of multi-agent systems has started in the early nineties from distributed artificial intelligence. In recent years fields like social psychology, game theory, logic, and argumentation theory have also contributed substantially to multi-agent systems in order to understand cooperation and to design effective computational and mixed human-machine multi-agent systems.

Understanding social interactions requires rich formal models of cooperation and social cognition, because multi-agent systems consist of several agents that can act and communicate autonomously, without central control. This talk will present some recent results about cooperation and social cognition in multi-agent systems, as seen from the logical point of view.

Identities and Complexity

(Nitin Saxena)

Given an arithmetic circuit C computing a polynomial over variables $x_1, \ldots, x_n$, we want to check whether C is the zero circuit. This problem is called Polynomial Identity Testing (PIT). It is a fundamental computational problem with connections to various other problems in computational complexity theory. In this talk we will survey Identity Testing and discuss potential approaches.

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 eight universities (University of Nijmegen, Leiden University, Technische Universiteit Eindhoven, University of Twente, Utrecht University, University of Groningen, Vrije Universiteit Amsterdam, and the University of Amsterdam), the CWI and Philips Research (Eindhoven) participate in IPA. In 1997, IPA was formally accredited by the Royal Dutch Academy of Sciences (KNAW). In 2002, this accreditation was extended for a period of five years. In 2006, IPA set its agenda for the 2007 - 2012 period (see "IPA 2007-2012" below), and applied for an extension of its accreditation by the KNAW.

Early in 2007, Delft University became the ninth university to sign on, when its Software Engineering Research Group (led by Arie van Deursen) joined IPA.

Activities in 2006

IPA has two multi-day events per year which focus on current topics, the Lentedagen and the Herfstdagen. In 2006, the Lentedagen were dedicated to Testing and the Herfstdagen were on Stochastic Systems.

Lentedagen on Testing *April 19 - 21, 2006, Landgoed Huize Bergen, Vught*
Testing is an important technique for validating and checking the correctness of software, but it is traditionally a difficult, expensive, laborious, and error-prone part of the software engineering process. However, over the past decade it has also proven to be a fertile area for the application of formal methods. The community of IPA researchers spotted the possible benefits of using formal methods in testing early on, it has been a prominent application area ever since the school was founded.

At present, the emphasis in IPA is on model-based testing, and its extensions to real-time (and hybrid) testing, symbolic (or data) testing, and component-based testing. Hence, the model-based approach was the focus of the Lentedagen, which aimed to provide an overview of the current state of the field in and around IPA. The event featured guest speakers from both academia and industry: Antti Huima (Conformiq, Espoo Finland), Moez Krichen (Verimag, Grenoble), Vlad Rusu (Inria, Rennes), Ina Schieferdecker (Fraunhofer, Berlin), and Margus Veanes (Microsoft Research, Redmond). The program was composed by Jaco van de Pol (CWI), Judi Romijn (TU/e), Mariëlle Stoelinga (UT), and Jan Tretmans (RU).

Abstracts, hand-outs and papers are still available through the archive on the IPA-website: www.win.tue.nl/ipa/archive/springdays2006/.

Herfstdagen on Stochastic Systems *November 27 - December 1, Hotel Marijke, Bergen*
Traditionally, a distinction is made in computer science between functional (qualitative) and non-functional (quantitative or performative) properties of systems. Models and techniques for the analysis and validation of these two kinds of properties were developed in separate communities: researchers in Formal Methods concentrated on the qualitative properties of systems, whereas researchers in Performance Analysis focussed on quantitative properties.

In modern systems, like embedded and networked systems, the distinction between functional and performance properties is getting blurred: both kinds of properties are of equal importance, and often properties of one kind affect properties of the other kind. This was recognized in the '90's, when a move towards integrated frameworks started, in which methods and techniques from Formal Methods are brought together with probabilistic and stochastic approaches from Performance Analysis.

Researchers in IPA have played an active role in bringing about this synthesis, and the research school has dedicated its Lentedagen to this topic in 1999 and an EEF Summer School in 2000. The aim of this year's Herfstdagen was to present an overview of the current state of this flourishing research area.

The Herfstdagen presented different approaches to the modelling and analysis of stochastic and probabilistic systems (with sessions on automata, stochastic and probabilistic process algebra, model checking of discrete and continuous Markov chains, Markov decision processes and infinite state models), and also highlighted applications of these approaches in other domains (such as biological systems, systems engineering, and security). A new and succesful feature for this year's Herfstdagen was the "clinic" on the probabilistic symbolic model checker on PRISM (University of Birmingham). Dave Parker, a member of the core development team, gave a half day introduction to the tool consisting of a tutorial combined with a hands-on lab-session.

The program for the event was composed by Jos Baeten (TU/e), Boudewijn Haverkort (UT), Joost-Pieter Katoen (RWTH Aachen), and Frits Vaandrager (RU). Abstracts, hand-outs, and papers are available through the IPA website at `www.win.tue.nl/ipa/archive/falldays2006/`.

IPA organises Basic Courses on each of its major research fields, Algorithms and Complexity, Formal Methods and Software Technology. These Basic Courses intend to give an overview of (part of) the research of IPA in these fields, and are organized at regular intervals on a cyclic schedule. In 2006, two courses were held.

IPA Basic Course on Formal Methods *January 16 - 20, TU/e, Eindhoven*
The Basic Course, composed by Jos Baeten (TU/e) focussed on five main areas of Formal Methods research. One course day was dedicated to each of these areas, combining lectures with hands-on tool training.

Topics, tools, and teachers were: *Introduction to Formal Methods (in IPA)*, Jos Baeten (TU/e); *ISPEC & Calisto*, Hans Jonkers (Philips Research), Ruurd Kuiper & Erik Luit (TU/e); *Model Checking & SPIN*, Joost-Pieter Katoen (RWTH Aachen); *Theorem Proving & PVS*, Erik Poll (RU); *Process Algebra & mCRL2*, Jan Friso Groote (TU/e); *Business Processes & Petri Nets*, Wil van der Aalst (TU/e).

IPA Basic Course on Software Technology *September 18 - 22, TU/e, Eindhoven*
The Basic Course, composed by Mark van den Brand (TU/e), focussed on central areas of Software Technology research in IPA. The course days dedicated to these areas combined lectures with hands-on tool training.

Topics, and teachers were: *Introduction*, Mark van den Brand (TU/e); *Aspect-Oriented Software Development*, Lodewijk Bergmans (UT), *Tree Oriented Programming*, Jeroen Fokker (UU), *Generic Language Technology*, Mark van den Brand (TU/e), Jurgen Vinju (VU) & Steven Klusener (VU), *Strategic Programming*, Eelco Visser & Martin Bravenboer (UU), *Software Technology for Software Evolution*, Marius Marin, Cathal Boogerd (TUD), Magiel Bruntink (CWI) & Duncan Doyle (Fortis).

IPA Ph.D. Defenses in 2006

Ricardo Javier Corin (UT, 12 januari), *Analysis Models for Security Protocols*. Promotor: prof.dr. P.H. Hartel. Co-promotor: dr. S. Etalle. IPA-Dissertation Series 2006-02.

Eelco Dolstra (UU, 18 januari), *The Purely Functional Software Deployment Model*. Promotor: prof.dr. S.D. Swierstra. Co-promotor: dr. E. Visser. IPA-Dissertation Series 2006-01.

Peter Verbaan (UU, 1 februari), *The Computational Complexity of Evolving Systems*. Promotors: prof.dr. J. van Leeuwen, prof.dr. J. Wiedermann. IPA-Dissertation Series 2006-03.

Ka Lok Man and Ramon Schiffelers (TU/e, 7 februari), *Formal Specification and Analysis of Hybrid Systems*. Promotors: prof.dr. J.C.M. Baeten, prof.dr.ir. J.E. Rooda. Co-promotors: dr. M.A. Reniers, Dr. D.A. van Beek. IPA-Dissertation Series 2006-04.

Albert Jan Markvoort (TU/e, 14 maart 2006), *Towards Hybrid Molecular Simulations*. Promotors: prof.dr. P.A.J. Hilbers, prof.dr. M.A.J. Michiels. Co-promotor: dr. R. Pino Plasencia. IPA-Dissertation Series 2006-09.

Jeroen Ketema (VU, 20 maart 2006), *Böhm-Like Trees for Rewriting*. Promotor: prof.dr. J.W. Klop. Co-promotors: dr. R.C. de Vrijer, dr. F. van Raamsdonk. IPA-Dissertation Series 2006-07.

Martijn Hendriks (RU, 4 april), *Model Checking Timed Automata - Techniques and Applications*. Promotor: prof.dr. F.W. Vaandrager. IPA-Dissertation Series 2006-06.

Marcel Kyas (UL, 4 april), *Verifying OCL Specifications of UML Models: Tool Support and Compositionality*. Promotors: prof.dr. J.N. Kok, prof.dr. W.-P. de Roever. Co-promotor: dr. F.S. de Boer. IPA-Dissertation Series 2006-05.

Cees-Bart Breunesse (RU, 24 april), *On JML: topics in tool-assisted verification of JML programs*. Promotor: prof.dr. B.P.F. Jacobs Co-promotor: dr.ir. E. Poll. IPA-Dissertation Series 2006-08.

Siegfried Nijssen (UL, 15 mei), *Mining Structured Data*. Promotor: prof.dr. J.N. Kok. Co-promotor: dr. W.A. Kosters. IPA-Dissertation Series 2006-10.

Giovanni Russello (TU/e, 27 juni), *Separation and Adaptation of Concerns in a Shared Data Space*. Promotors: prof.dr. M. van Steen, prof.dr. M. Rem. Co-promotor: dr. M.R.V. Chaudron. IPA-Dissertation Series 2006-11.

Tomas Krilavicius (UT, 6 september), *Hybrid Techniques for Hybrid Systems*. Promotor: prof.dr. H. Brinksma. Co-promotor: dr.ir. R. Langerak. IPA-Dissertation Series 2006-15.

Bahareh Badban (VU, 7 september), *Verification Techniques for Extensions of Equality Logic*. Promotor: prof.dr. W.J. Fokkink. Co-promotor: dr. J.C. van de Pol. IPA-Dissertation Series 2006-13.

Ling Cheung (RU, 18 september), *Reconciling Nondeterministic and Probabilistic Choices*. Promotor: prof.dr. F.W. Vaandrager. IPA-Dissertation Series 2006-12.

Arjan Mooij (TU/e, 2 October), *Constructive Formal Methods and Protocol Standardization*. Promotor: prof.dr.ir. J.F. Groote. Co-promotor: dr. J.M.T. Romijn. IPA-Dissertation Series 2006-14.

Cas Cremers (TU/e, 6 november), *Scyther - Semantics and Verification of Security Protocols*. Promotor: prof.dr. J.C.M. Baeten. Co-promotors: dr. S. Mauw, dr. E.P. de Vink. IPA-Dissertation Series 2006-20.

Louis van Gool (TU/e, 27 november), *Formalising Interface Specifications*. Promotors: prof.dr. J.C.M. Baeten, prof.dr. A. de Bruin. Co-promotor: dr. R. Kuiper. IPA-Dissertation Series 2006-19.

Martijn Warnier (RU, 27 november), *Language Based Security for Java and JML*. Promotor: prof.dr. B.P.F. Jacobs. Co-promotor: dr. M.D. Oostdijk. IPA-Dissertation Series 2006-16.

Biniam Gebremichael (RU, 11 december), *Expressivity of Timed Automata Models*. Promotor: prof.dr. F.W. Vaandrager. IPA-Dissertation Series 2006-18.

IPA 2007-2012

In preparing for the next period, IPA reviewed its main research areas and the four focus areas (Networked Embedded Systems, Security, Intelligent Algoriths, and Compositional Programming Methods) for 2002 - 2006.

As before, the three main research areas for 2007 - 2012 will be *Algorithms and Complexity*, *Formal Methods*, and Software Technology. During 2006, the descriptions of these areas were updated (see `www.win.tue.nl/ipa/about`) to reflect developments in the field. In this process, the area Software Technology was renamed into *Software Technology & Engineering* to better reflect the wider range of research topics addressed by various research groups in IPA (particularly in software renovation, software architecture, and software processes).

For the coming period five new focus areas within the scope of IPA were chosen, where we expect important developments in the near future and want to stimulate collaboration:

- The future challenges of computing require new concepts that are no longer adequately modeled by the classical Turing machine paradigm. In the focus area *Beyond Turing* we want to explore these novel concepts and paradigms.
- Nowadays computer science plays a foundational role in the life sciences. In the focus area *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.
- Systems with both continuous and discrete dynamics are becoming increasingly important in many applications. In the focus area *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 architectures are getting more and more popular both in industrial and scientific communities. However, there are many open questions regarding (the use of) models in this approach. In the focus area *Model-driven software engineering* we want to study various fundamental aspects of the model-driven approach to software engineering.
- The quality of software is characterized in terms of a growing set of attributes (for instance availability and modifiability in addition to functional requirements). In the focus area *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.

Activities in 2007

IPA is planning several activities for 2007, including the Lentedagen (April), the Basic Course on Algorithms & Complexity (June, Eindhoven), and the Herfstdagen (November) which will be dedicated to one of the new focus areas. 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 in 2006

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. SIKS is an interuniversity research school that comprises 12 research groups from 10 universities and CWI. Currently, nearly 400 researchers are active, including over 190 Ph.D.-students. The Vrije Universiteit in Amsterdam is SIKS' administrative university, and as off January 1 2006 Prof.dr. R.J. Wieringa (UT) is scientific director. The office of SIKS is located at Utrecht University. SIKS received its first accreditation by KNAW in 1998 and was re-accreditated in June 2003 for another period of 6 years. In 2006 the board of governors decided to apply for re-accreditation in 2009 again.

Activities

We here list the main activities (co-)organized or co-financed by SIKS. We distinguish basic courses, advanced courses and other acitivities (including masterclasses, workshops, one-day seminars, conferences and research colloquia)

Basic courses

"System modeling", May 29-31, 2006, Landgoed Huize Bergen, Vught
Course directors: dr. P. van Eck (UT), dr. W.-J. van den Heuvel (UvT)

"Knowledge modeling", May 31-June 02, 2006, Landgoed Huize Bergen, Vught
Course director: dr. B. Bredeweg (UVA)

"Information and Organisation", September 25-27, 2006, Landgoed Huize Bergen, Vught
Course directors: dr. H. Weigand (UvT), prof.dr. ir. P. Grefen (TUE)

"Architectures for IKS" September 27-29, 2006, Landgoed Huize Bergen, Vught
Course director: prof. dr. E. Proper (RUN)

"Research methods and methodology for IKS", November, 20-21, 2006, Lunteren
Course directors: dr. H. Weigand (UvT), prof. dr. R. Wieringa (UT), prof.dr. H. Akkermans (VU)
prof. dr. J-J.Ch. Meyer (UU), dr. R. Starmans (UU)

Advanced courses

"Computational Intelligence", April 03-04 2006, Landgoed Huize Bergen, Vught
Course directors: dr. T. Heskes (RUN)

"Summer course on Datamining", July 03-07, 2006, Maastricht
Course directors: dr. E. Smirnov (UM), dr. J. Donkers (UM), Prof. dr. E.O. Postma (UM)

Other activities

- Theory-day 2006 of the NVTI, March 10, 2006, Utrecht
- Dutch Belgian Information Retrieval Workshop, March 13-14, 2006, Delft.
- SIKS Masterclass on Human-Computer Interaction, March 15, 2006, Amsterdam
- Workshop on Perceptual Cognition, April 20, 2006, Maastricht
- SIKS Masterclass "Requirements Engineering & Information Modeling", May 30, 2006 Amsterdam
- Workshop on Data modeling, June 06, 2006, Enschede
- Agent Systems summer school, EASSS 2006, July 17-21, 2006, Annecy
- SIKS-conference on Enterprise Information Systems (EIS 2006), Sept 08, 2006, Utrecht
- BNAIC 2006, 05-06 October 2006, Namur
- Dutch-Belgian Database Day, November 15-16 2006, Brussel
- SIKS-IKAT Research colloquium: organized 6 times in Maastricht
- CABS-SIKS Research colloquium, organized 8 times in Delft en Utrecht

Promotions in 2006

In 2006 28 researchers successfully defended their Ph.D.-thesis and published their work in the SIKS-dissertation Series.

2006-01
Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting
Promotores: Prof.dr.ir. P. W.P.J. Grefen (TUE), Prof. dr. ir. J.A. La Poutré (TUE/CWI)
Promotie: 02 February 2006

2006-02
Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations
Promotores: Prof.dr.G. van der Veer (VU), prof. dr. J.C. van Vliet (VU)
Promotie: 14 March 2006

2006-03
Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems
Promotor: Prof.dr.B.J. Wielinga (UVA)
Co-promotor: Dr. J. Sandberg (UVA)
Promotie: 21 April 2006

2006-04
Marta Sabou (VU)
Building Web Service Ontologies
Promotores: prof.dr. F.A.H. van Harmelen (VU), prof.dr. H. Stuckenschmidt (University of Mannheim, Germany)
Promotie: 27 April 2006

2006-05
Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines
Promotor: prof.dr. J.-J. Ch. Meyer (UU)
Co-promotor: dr. F.S. de Boer (UU/CWI)
Promotie: 03 May 2006

2006-06
Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling
Promotor: prof.dr. J.M. Akkermans (VU)
Co-promotor: dr. J. Gordijn (VU)
Promotie: 29 May 2006

2006-07
Marko Smiljanic (UT)
XML schema matching -- balancing efficiency and effectiveness by means of clustering
Promotor: Prof.dr. W. Jonker (UT and Philips Research)
Co-promotor: dr. M. van Keulen (UT)
Promotie: 21 April 2006

2006-08
Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web
Promotor: Prof. dr. ir. A. Nijholt (UT)
Assistent-Promotor: Dr. E.M.A.G. van Dijk (UT)
Promotie: 13 April 2006

2006-09
Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion

Promotores: Prof.dr. H.J. van den Herik (UM), Prof. E.H.J. Vaassen (UM)
Co-promotores: Prof. H.F. Ali(Rutherford University), Dr. P. Spronck (UM)
Promotie: 29 June 2006

2006-10
Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems
Promotor: Prof.dr. F.A.H. van Harmelen (VU)
Promotie: 09 June 2006

2006-11
Joeri van Ruth (UT)
Flattening Queries over Nested Data Types
Promotor: Prof.dr. P.M.G. Apers (UT)
Promotie: 02 June 2006

2006-12
Bert Bongers (VU)
Interactivation - Towards an e-cology of people, our technological environment, and the arts
Promotores: Prof. dr. G. C. van der Veer (VU), Prof. dr. J. C. van Vliet (VU)
Promotie: 04 July 2006

2006-13
Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents
Promotores: Prof.dr. J.-J. Ch. Meyer (UU), Prof.dr. C.L.M. Witteman (RUN)
Promotie: 18 September 2006

2006-14
Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change
Promotores: Prof. dr. G. C. van der Veer (VU), Prof. dr. J. C. van Vliet (VU)
Promotie: 09 October 2006

2006-15
Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain
Promotor: Prof.dr. A.P.J.M. Siebes (UU)
Promotie: 11 October 2006

2006-16
Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks
Promotor: Prof.dr. A.P.J.M. Siebes (UU)
Co-promotor: dr. A.J. Feelders (UU)
Promotie: 23 October 2006

2006-17
Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device
Promotor: Prof.dr. J. van den Berg (UU), Prof.dr. M. Neerincx (TUD)
Co-promotor: dr. H. van Oostendorp (UU)
Promotie: 12 October 2006

2006-18
Valentin Zhizhkun (UVA)
Graph transformation for Natural Language Processing
Promotor: Prof.dr. M. de Rijke (UVA)
Promotie: 28 November 2006

2006-19
Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach
Promotor: Prof.dr. J.-J. Ch. Meyer (UU)
Co-promotores: dr. F.S. de Boer (CWI / LIACS / UU), dr. M. Dastani (UU)

Promotie: 25 October 2006

2006-20
Marina Velikova (UvT)
Monotone models for prediction in data mining
Promotores: Prof. dr. ir. H.A.M. Daniels (UvT / EUR), Prof. dr. J.P.C. Kleijnen (UvT)
Co-promotores: dr. A.J. Feelders (UU)
Promotie: 13 November 2006

2006-21
Bas van Gils (RUN)
Aptness on the Web
Promotores: Prof. dr. H.A. Proper (RUN), Prof. dr. ir. Th.P. van der Weide (RUN)
Promotie: 08 December 2006

2006-22
Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation
Promotor: Prof. dr. ir. Th.P. van der Weide (RUN)
Co-promotores: dr. P. van Bommel (RUN)
Promotie: 13 December 2006

2006-23
Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web
Promotor: Prof. Dr. J. van den Berg (UU)
Co-promotores: dr. H. van Oostendorp (UU)
Promotie: 19 October 2006

2006-24
Laura Hollink (VU)
Semantic Annotation for Retrieval of Visual Resources
Promotores: prof dr. A.Th. Schreiber (VU), prof dr. B.J. Wielinga (UVA)
Co-promotor: dr. M. Worring (UVA)
Promotie: 16 November 2006

2006-25
Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC
Promotor: Prof.dr.ir. L. C. van der Gaag (UU)
Co-promotor: dr.ir. D. Thierens (UU)
Promotie:27 November 2006

2006-26
Vojkan Mihajlovic (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval
Promotor: Prof.dr. P.M.G. Apers (UT)
Co-promotor: Dr. D. Hiemstra (UT)
Promotie: 07 December 2006

2006-27
Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated media repositories
Promotor: Prof. dr. L. Hardman (CWI/TUE)
Co-promotor: dr. F. Nack (CWI)
Promotie: 30 November 2006

2006-28
Borkur Sigurbjornsson (UVA)
Focused Information Access using XML Element Retrieval
Promotor: Prof.dr. M. de Rijke (UVA)
Co-promotor: dr. ir. J. Kamps (UVA)
Promotie: 14 December 2006

Adaptive Agents in Complex Negotiation Games

Valentin Robu Koye Somefun Han La Poutré

CWI, Dutch National Research Center for Mathematics and Computer Science
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
email: {robu , koye , hlp}@cwi.nl

Abstract

In this article, we focus on games in multi-agent systems. We first give a brief positioning and description of the area of adaptive agents, games, and multi-agent systems. We then consider complex negotiation games, in particular negotiation over many issues with inter-dependent valuations, and we present our recent results from [19]. We use ideas inspired by graph theory and probabilistic influence networks to derive efficient heuristics for negotiations about multiple issues. We explicitly model and utilize the underlying graphical structure of complex utility functions. We experimentally analyse — under relatively weak assumptions with respect to the structure of the utility functions – the developed approach. Our approach is applicable to domains where reaching a (Pareto) efficient outcome in a limited amount of negotiation steps is important. Furthermore, unlike other solutions for high-dimensional negotiations, the proposed approach does not require a mediator.

1 Introduction

1.1 Multi-agent systems, games, and adaptivity

Multi-agent systems have become an important paradigm in computer science. Such systems allow not only for distributed computation, but can represent autonomous actors as well. Different agents can thus represent different parties, with their own information, knowledge, strategies, and objectives. At the same time, agents can communicate and interact, in order to solve problems, to carry out tasks, or to make agreements.

The concept of multi-agent systems is coming from other scientific disciplines like economics and social sciences, as well as from daily life. In daily life, one can think of e.g. an insurance agent, a broker, a sales representative, and so on. In economics and social sciences, such agents are typically the actors or players in a collective system, representing persons, companies, governmental institutions, or organisations. The interaction between these parties is usually formalized by games, often as part of game theory. Well known instances are e.g. auctions, negotiations, social dilemmas (prisoner's dilemma), or social choice theory (voting systems).

Although a substantial amount of game theory exists, this is often not enough to design software agents that act in situations that can be represented in some game form. Reasons for this can be various. One issue is the computational or communication complexity that is needed to compute a solution. Another one is the inherent lack of information that agents have (incomplete information), especially in settings with restricted information access, unkown utility functions and strategies of competing selfish agents, or environments with a high complexity of relevant information. Both issues are instances of the theoretical concept of bounded rationality [1, 21]. Yet another one is that in many situations (like in repeated games, negotiation settings, or open uncertain environments), only a limited amount of game theory is available. For instance, central concepts occurring in one-shot games appear to quickly become very complicated or even intractable in repeated games: examples are notions like truth-revelation or dominant strategies in (auction) games, for which the theory on one-shot games does not easily transfer to repeated games.

For all such cases, substantial further research has to be performed to obtain software agents that can play in such games. For, the games occurring in realistic multi-agent systems are often not in the same stylized fashion as studied before in game theory. Often, precise game models have to be designed that correctly represent the game settings and the additional environmental constraints coming from broad problem areas, such as logistics or electronic commerce. Such game models then allow for fundamental research. Often, this then further comes down to algorithm design, information and knowledge representation, and adaptivity of agents (e.g. by adaptive algorithms) . Depending on the type and complexity of the problem, the analysis can then be (partly) mathematical or (partly) experimental, by means of controlled computer experiments.

In this article, we present some recent research results in this area, which we published in [19, 20]. We focus on the complex multi-issue negotiations and utility graphs, where we especially deal with [19].

1.2 Complex Multi-Issue Negotiations and Utility Graphs

Automated bilateral negotiation forms an important type of interaction in agent based systems for electronic commerce [15]. It allows seller and customer to determine the terms and content of the trade iteratively and bilaterally. Consequently, deals may be highly customized (especially for complex goods or services) and highly adaptable to changing circumstances. Moreover, by automating the negotiation, the potentially time-consuming process is delegated to autonomous software agents who conduct the actual negotiation on behalf of their owners.

In this paper, we consider the problem of a seller agent negotiating bilaterally with a customer about selecting a subset from a collection of goods or services, viz. the bundle, together with a price for that bundle. Thus, the bundle configuration — an array of bits, representing the presence or absence of each of the shop's goods and services in the bundle — together with a price for the bundle, form the negotiation issues. Like the work of [11, 12, 10, 23, 5], the techniques developed in this paper try to benefit from the so-called win-win opportunities, by finding mutually beneficial alternative bundles during negotiations.

The methods proposed in Faratin et al. [10], Coehoorn and Jennings [5] are geared towards finding win-win opportunities through modeling the preferences of the negotiation partner, for issues with independent valuations. This paper follows a similar approach; we consider *interdependencies* between issues, however, which make the problem considerably harder. In order to model such complex utility interdependencies between items, we introduce the novel concept of utility graphs.

Utility graphs build on the idea introduced in Chajewska and Koller [4] and Bacchus and Grove [3] that highly nonlinear utility functions, which are not decomposable in sub-utilities of individual items (such as in the seminal work of Raiffa [17]), may be decomposable in sub-utilities of clusters of inter-related items. They mirror the graphical models developed in (Bayesian) inference theory (cf. [13, 16]). Graphical models have been shown to be a powerful formalism for modeling decisions and preferences of other agents (see Lauritzen [13] for an overview). The idea behind using utility graphs in a multi-issue bargaining setting is to provide the seller with a formalism that can be used to explore the exponentially large bundle space, efficiently. In this paper, we show how utility graphs can be used to model an opponent's (i.e. customer's) preferences. Moreover, we also propose an updating procedure to obtain approximations of the customer's utility graph indirectly, by only observing his counter-offers during the negotiation.

An important benefit of using utility graphs in this setting is scalability: the problem becomes harder for larger number of items only if the underlying preference function becomes more complex as well. Graphical models (such as the one proposed in this paper), exploit the decomposable structure of utility functions, hence enabling more efficient search of the contract space.

At the start of a negotiation process, the seller's approximation of the customer's utility graph represents some prior information about the *maximal structure* of the utility space to be explored. This prior information could be obtained through a history of past negotiations or the input of domain experts. (An important advantage of a utility graphs is that they can handle both qualitative and quantitative prior information.) After every (counter) offer of the customer, this approximation is refined. Conducted computer experiments show that — by using only a fairly weak assumption on the maximal structure of customers' utility functions — the updating procedure enables the seller to suggest offers that closely approximate Pareto ef-

ficiency [1]. Moreover, efficient outcomes are reached after relatively few negotiation rounds, which enables our approach to be used in applications where time constraints or the impatience of buyers are limiting factors.

The rest of this paper is organized as follows. Section 2 presents the negotiation setting: it defines the efficiency criteria, the negotiation protocol and the top-level outline of the negotiation algorithm. Section 3 defines the concept of utility graphs, it describes how such graphs can be used to model complex utility functions and to learn the preferences of the opponent in negotiation settings. Section 4 presents the set-up and results from the experiments performed to validate the model, while Section 5 concludes the paper with a discussion of related work and outlines the contributions of this paper.

2 The negotiation setting

We consider a buyer and seller who negotiate bilaterally over a set of n binary-valued issues or items, and one continuous issue, the price. Henceforth, we will refer to the binary-valued issues as items and to subsets formed with these items as bundles. Negotiations are conducted in an alternating exchange of offers and counter offers. The offers and counter offers contain a n-dimensional vector of 0's and 1's representing an instantiation of the n items, plus a price offered/asked for this bundle.

The utility (measured in terms of monetary value) a buyer assigns to any bundle of items is given by a complex (nonlinear) function that takes into account interdependencies between various items. The seller's utility for a bundle (measured in net monetary value) is the difference between the price received for a bundle and the costs incurred for providing a bundle. These costs are additive: i.e., the bundle cost equals the cost of offering the items individually. Both the buyer's utility and the seller's cost represent private information, which remains undisclosed before or during the negotiation. Therefore, the negotiation setting can be describes as double-sided incomplete information.

2.1 Net Utility functions of Buyer and Seller

Let $B = \{I_1, \ldots, I_n\}$ denote the collection of n items a seller and buyer negotiate over. Each item I_i takes on either the value 0 or 1: 1 (0) means that the item is (not) purchased. Thus B has the domain $Dom(B) = \{0, 1\}^n$ (so there are 2^n possible bundles). The n-dimensional vector $\vec{b} \in Dom(B)$ denotes an instantiation of these n items. In our approach, the utilities assigned to different outcomes (combinations) are represented by monetary units, rather than values between 0 and 1. We found this choice more natural for the e-commerce scenarios we consider. The utility function $u : Dom(B) \mapsto \mathbb{R}$ specifies the monetary value a buyer assigns to all (2^n) possible outcomes. Due to interdependencies between various items the function u is *highly* nonlinear. The buyer's *net utility* (i.e. net monetary value) for purchasing a bundle $\vec{b}$ for a price p, denoted by $nu_b(\vec{b}, p)$, is defined as follows:

$$nu_b(\vec{b}, p) = u(\vec{b}) - p. \tag{1}$$

That is, $nu_b(\vec{b}, p)$ is the difference between the monetary value for acquiring (consuming) bundle $\vec{b}$ minus the price p paid for purchasing bundle $\vec{b}$. The net monetary value of the seller is computed as:

$$nu_s(\vec{b}, p) = p - Costs(\vec{b}) \tag{2}$$

Thus, the seller's net monetary value for the sales of a bundle $\vec{b}$ for a price p is just the price minus the cost for selling the items. Currently, the seller has an *additive cost structure*: i.e., the bundle costs $Costs(\vec{b})$ equals the sum of the cost incurred when selling the items individually.

2.2 Using gains from trade as efficiency criteria

The most widely used performance criteria in multi-attribute negotiations is Pareto efficiency. Raiffa [17] provides a method to compute Pareto-efficient outcomes in the case utility functions of both parties are

[1] A deal is Pareto efficient whenever no party can be made better off without making the other one worse off.

normalized between 0 and 1. In our model, utilities are represented in monetary units instead of mappings between 0 and 1. Consequently, it is more insightful to compute the gains from trade that can result from exchanging a certain bundle of items $\vec{b}$. The gains from trade are defined as:

$$GT(\vec{b}) = u(\vec{b}) - Cost(\vec{b}), \tag{3}$$

where $u(\vec{b})$ denotes the buyers monetary value for $\vec{b}$. The notion of gains from trade is well founded in the economic literature on trade (c.f. [2]). Moreover, for the above setting (where utility is expressed in monetary units) the set of bundles maximizing the gains from trade can be proven to be the same as the set of Pareto-efficient bundles (c.f. [23]). Intuitively, the gains from trade can be seen as the maximal size of joint gains, which can be achieved through negotiation, while the continuous attribute, the price represents different ways to divide these joint gains.

2.3 Top-level outline of the negotiation algorithm

The negotiation, in our model, follows an alternating offers protocol. At each negotiation step, each party (buyer/seller) makes an offer which contains an instantiation of 0/1 for all items in the negotiation set (denoting whether they are/are not included in the proposed bundle), as well as a price for this bundle. The decision process for each party, at each negotiation step, is composed of 3 inter-related parts: (1) take into account the previous offer made by the other party, (2) compute the contents (i.e. item configuration) of the next bundle to be proposed, and (3) compute the price to be proposed for this bundle.

Alg. 1 gives an outline of the algorithm the seller uses in each negotiation step. On the buyer's side, the followed steps mirror Alg.1 with the notably difference that the buyer does not try to estimate or reach the highest gains from trade, but selects the bundle that optimizes only his own expected net utility (i.e., he is entirely selfish). Based on the ongoing negotiation process he can update this choice, however. In this model, the burden of exploring the huge bundle space and recommending jointly profitable solutions is therefore passed to the seller, who must solve it by modeling the preferences of the buyer. This is a reasonable model in cases where an electronic merchant negotiates with different buyers in succession and tries to optimize both his own profits and buyer satisfaction (essential for building up a durable client relationship, which would generate repeated business). Furthermore, in e-commerce domains it is reasonable to assume that electronic merchants are more knowledgeable than individual buyers.

Algorithm 1 Top level algorithm used by the seller

Denote by $(\vec{b}_b, p_b)$ the previous offer of the buyer and by $(\vec{b}_s, p_s)$ the previous offer of the seller.
1. If $\vec{b}_b = \vec{b}_s$ (configuration is agreed) and $p_s - p_b < \Delta p$ (difference in ask and offer prices is under some maximum acceptable threshold), then Success.
2. Otherwise:
3. Update the estimated utility graph of the buyer based on his past bid $\vec{b}_{buyer}$
4. Compute (one of) the bundles $\vec{b}^*$ with the highest gains from trade
5. Compute the price to be proposed for $\vec{b}^*$ such that it represents a linear time concession from my previous offer
6. Propose this bundle and price to the buyer

In our model the seller starts the negotiation by posting an ask price for each item. This price reflects the maximum profit the seller expects to make by selling that item (i.e. the ask price reflects his maximal aspiration level). The buyer also has a maximal aspiration level, which reflects the maximum discount he expects to get. The discount the buyer will actually get for a bundle depends on the content of that bundle and on the negotiation process itself.

The seller's approach is to search for a bundle that has the maximum gains from trade, since in this way he increases the joint gains: the only way he can continue to offer a better discount, but also increase his own profits, while the Buyer is strictly selfish and will always propose a bundle that increases his own (expected) utility. Regarding price concessions, both the buyer and the seller compute, for the current proposed bundle their current aspiration level and then make a time-dependent price concession with respect to that aspiration level.

For consistency, all experimental results reported in this paper refer to the *linear, time-dependent* concession case. However other time-dependent concession strategies usually employed in bilateral negotiations (such as hard-headed, boulware [8]) have been implemented and tested, with consistently good results. We note that the accuracy of the bundle predicted to have the highest gains from trade (step 3 and 4 in Alg. 1) does not significantly depend on the price concession strategy, because we made the explicit modeling choice that the concession strategy concerns only the pricing (step 5 of Alg. 1 above), not the exploration of the bundle content itself (steps 3 and 4 of Alg. 1). We acknowledge, however, that if the concession strategy were directly embedded in the opponent-learning mechanism (performed using the utility graph), the consistency of our experimental results may be effected. Finally, we model time pressure and/or buyer impatience through a break-off probability: at each step there is a small risk of breakdown (a value of 2% was used in the simulations).

3 Using graphs to model utility functions

3.1 Decomposable Utility Functions

Recall that we consider a buyer who negotiates with a seller over a bundle of n items, denoted by $B = \{I_1, \ldots, I_n\}$. Each item I_i takes on either the value 0 or 1: 1 (0) means that the item is (not) purchased. The utility function $u : Dom(B) \mapsto \mathbb{R}$ specifies the monetary value a buyer assigns to each of the 2^n possible bundles ($Dom(B) = \{0,1\}^n$).

In traditional multi-attribute utility theory, u would be decomposable as the sum of utilities over the individual issues (items) [17]. However, in this paper we follow the previous work of [4] by relaxing this assumption; they consider the case where u is decomposable in *sub-clusters* of individual items such that u is equal to the sum of the *sub-utilities* of different clusters.

Definition 1 *Let C be a set of clusters of items $C_1, \ldots, C_r$ (with $C_i \subseteq B$). We say that a utility function is factored according to C if there exists functions $u_i : Dom(C_i) \mapsto \mathbb{R}$ ($i = 1, \ldots, r$ and $Dom(C_i) = \{0,1\}^{|C_i|}$) such that $u(\vec{b}) = \sum_i u_i(\vec{c}_i)$ where $\vec{b}$ is the assignment to the variables in B and $\vec{c}_i$ is the assignment to the variables in C_i, induced from the assignment $\vec{b}$. We call the functions u_i sub-utility functions.*

The factorization of a decomposable utility function is not unique. In this paper, we use the following factorization, which is a relatively natural choice within the context of negotiation. Single-item clusters ($|C_i| = 1$) represent the individual value of purchasing an item, regardless of whether other items are present in the same bundle. Clusters with more than one element ($|C_i| > 1$) represent the *synergy effect* of buying two or more items; these synergy effects are positive for complementary items and negative for substitutable ones. eams that we allow *direct* synergy effects between up to 4 items. The factorization defined above can be represented as an undirected graph $G = (V, E)$, where the vertices V represent the set of items I under negotiation. A vertex between two nodes (items) $i, j \in V$ is present in this graph if and only if there is some cluster C_k that contains both I_i and I_j. We will henceforth call such a graph G a *utility graph* [2].

Example 1 *Let $B = \{I_1, I_2, I_3, I_4\}$ and $C = \{\{I_1\}, \{I_2\}, \{I_1, I_2\}, \{I_2, I_3\}, \{I_2, I_4\}\}$ such that u_i is the sub-utility function associated with cluster i ($i = 1, \ldots, 5$). Then the utility of purchasing, for instance, items I_1, I_2, and I_3 (i.e., $\vec{b} = (1,1,1,0)$) can be computed as follows: $u((1,1,1,0)) = u_1(1) + u_2(1) + u_3((1,1)) + u_4((1,1))$, where we use the fact that $u_5((1,0)) = 0$ (synergy effect only occur when two or more items are purchased). The utility graph of this factorization is depicted in Fig. 1.*

To give a numerical example, suppose: $u_1(1) = \$7$, $u_2(1) = \$5$, $u_3((1,1)) = -\$5$, $u_4((1,1) = \$4, u_5((1,1)) = \$4$. Moreover, all item costs are equal to \$2: i.e., $c(I_1) = c(I_2) = c(I_3) = c(I_4) = \$2$. In this case the bundle with the maximum gains from trade (i.e. the bundle denoted by $\vec{b}^*$ in Alg. 1) is:

[2] The concept of cluster defined for utility graphs can be seen as corresponding to the concept of cliques in probabilistic networks. However, for consistency, we use only the term cluster throughout this paper.

*(0, 1, 1, 1), which has the net monetary value of $5+$4+$4 - 3*$2 = $7. From this simple example we can already highlight an important problem. The assignments for items I_1 and I_3, I_4 influence each other indirectly, through the assignment for I_2 (although there are no direct links from I_1 to I_3, I_4). This feature is exploited by our decomposition algorithm.*

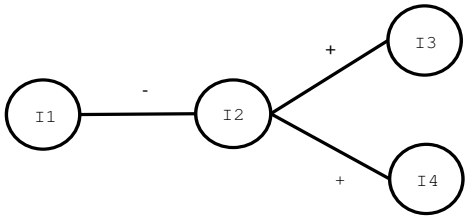

Figure 1: The utility graph that corresponds to the factorization according to C in Example 1. The $+$ and $-$ signs on the edges indicate whether the synergy effect are positive or negative.

At the computational level, each cluster is represented by a *joint utility table*, which assigns a utility value for all combinations of instantiations with 0/1 of items in that cluster (this is similar in concept to the joint probability tables, used to represent inter-dependent variables in probabilistic networks). Therefore, the cost of this utility representation is then exponential in the number of items in the cluster. For this reason, we limit the maximum size of any cluster to 4. However, this does not mean that the maximum size of the inter-dependencies is limited to 4 – it can be arbitrarily large. This is because one item can simultaneously belong to several clusters, so when deciding whether to include it or not in the optimal bundle (see section 3.3), the utility values in more than one table need to be taken into account. There is also a second, more fundamental reason to limit the maximum cluster size, i.e. with higher-order interdependencies, the decomposition algorithm on which our method is based is not guaranteed to produce a satisfactory partition (Section 3.3 gives a more precise description of this issue).

3.2 The maximal interdependency graph

As shown in the previous section, we assume that the buyer's utilities can be modeled as a graph. The structure of this graph, as well as the sub-utilities corresponding to different clusters constitute private information which is not revealed during the negotiation.

However, the seller does have some prior information to guide his opponent modeling. He starts the negotiation by having a *maximal item inter-dependence graph* of all possible inter-dependencies between the issues (items) which can be present in a given domain. The utility graphs of buyers form subgraphs of this graph. Note that this assumption says nothing about the weights or values of the sub-utility functions, so the negotiation is still with double-sided incomplete information. Furthermore, it does not mean that the seller has to know the exact structure of the utility graph of the buyer. For example, suppose two issues are assumed substitutable by the seller, so the utility of the combination containing both items is lower than the sums of utilities for individual items. If the buyer signals (through his bids) that he is willing to accept bundles containing both items, the seller will adjust the weight of this relation (i.e. adjust the values for the relevant cluster sub-utility) towards the sum of utilities for individual items. Conceptually, this is equivalent to removing the edge from the graph (which means they are no longer assumed substitutable).

The presence of this graph helps to greatly reduce the complexity of the search space on the side of the seller. The structural information contained in such a graph can be obtained either from a history of past negotiations or elicited from human experts. Note that in most domains it is reasonable to assume that the seller does know something about the goods he is selling. For example, if he is selling online pay-per-view journal articles, then articles within the same category (or with the same author) can be potentially related (though not guaranteed to be related for every buyer).

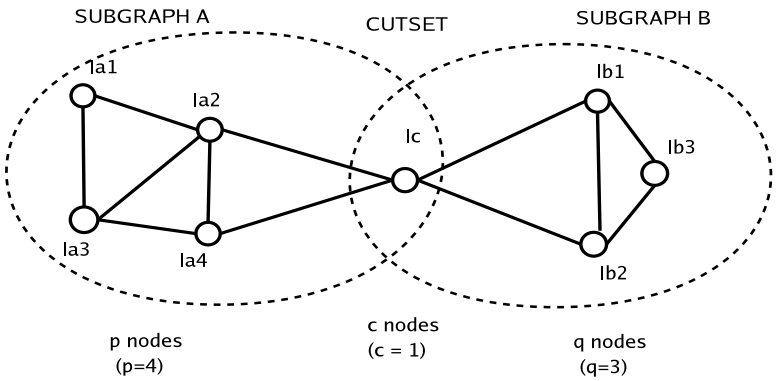

Figure 2: Utility graph where the vertices $\{Ia1, Ia2, Ia3, Ia4, Ic\}$ and $\{Ib1, Ib2, Ib3, Ic\}$ form the two cliques and corresponding subgraphs of $\hat{G}$. Moreover, $\{I_c\}$ forms the only cutest.

3.3 Selecting the best counter offer

A problem which the seller faces at each negotiation step (see Alg. 1, step 2) is to choose a bundle $\vec{b}^*$ which has the highest gains from trade of the 2^n bundles. More formally stated: $\vec{b}^* = argmax_{\vec{b}}(\hat{GT}(\vec{b})) = argmax_{\vec{b}}(\hat{u}(\vec{b}) - Cost(\vec{b}))$. Note that in the above definition, we use the notations $\hat{GT}$ and $\hat{u}$ instead of GT and u, since the seller does not know the *true* utility value of the buyer. He only has an estimation of it, which is updated after receiving a (counter) offer.

A straight-forward, brute-force solution to determine $\vec{b}^*$ is to generate all bundles $\vec{b}$ and select one which has the highest (estimated) gains from trade. Since this involves 2^n steps at *every* iteration, it is clearly not feasible for large n, so a heuristic is needed to reduce this search space.

Suppose the utility graph is decomposable into two or more completely disjoint parts (no overlapping vertices). We can then compute an optimal sub-bundle for each of the parts and merge them. However, for the more general case we still need a method for reducing the complexity of the search, when the original graph (or a large sub-component of it) is not decomposable in such a straight-forward way. We do this by applying ideas of the tree-decomposition theory to the utility graph $\hat{G} = (\hat{V}, \hat{E})$.

Informally, a tree decomposition of a graph is a family of small, not necessarily disjoint, subgraphs $\hat{G}_1, \ldots, \hat{G}_k$ (for some $k \in \mathbb{N}$), the union of which makes up the initial graph. Associated with a tree decomposition $\hat{G}_1, \ldots, \hat{G}_k$ is a collection of cutsets: a cutset is a subset of vertices that belong simultaneously to the same two subgraphs. (See [18] for a more formal discussion of tree decomposition algorithms).

Example 2 *Figure 2 depicts such a decomposition. The vertices* $\{Ia1, Ia2, Ia3, Ia4, Ic\}$ *and* $\{Ib1, Ib2, Ib3, Ic\}$ *form the two cliques and corresponding subgraphs of* $\hat{G}$*. There are only two subgraphs therefore there is only* 1 *cutset;* I_c *forms this cutest because it is the only vertex that lies in both subgraphs.*

In the conducted experiments, cutsets contain at most 1 or 2 vertices. This is because, for any graph, polynomial-time algorithms exist for decomposing a graph according to cutsets of size 1 or 2, provided that the given graph can be decomposed in cutsets of size maximum 2 (cf. [18] for an overview). For the algorithm presented below, we do *not* need to decompose the whole graph, however. It suffices to decompose the graph at enough points to reduce its complexity. This means that the largest sub-component of vertices left (i.e. which is not or cannot be decomposed any further) has a manageable size to allow exhaustive search.

Alg. 2 generates all possible combinations only for items that overlap between subgraphs (i.e. cutset items). Then, for all subgraphs it chooses the sub-combination that represents a *local* maximum for the gains from trade function in the considered subgraph, but subject to the constraint that the items that belong to more than one subgraph (i.e. cutset items) have the same values in all subgraphs. Finally, the best overall combination is chosen as a maximum of combinations of local maximums achieved for all possible instantiations for cutset vertices (items).

Algorithm 2 Algorithm returns $\vec{b}^*$, a bundle with the highest gains from trade (i.e., $\vec{b}^* \in \text{argmax}_{\vec{b} \in Dom(B)} \hat{GT}(\vec{b})$)

$\hat{G}_1 = (V_1, E_1), \ldots, \hat{G}_k = (V_k, E_k)$ and the union of all cutsets $S \subseteq V$ are given; $\hat{GT}_i : Dom(V_i) \mapsto \mathbb{R}$ denotes the predicted gains from trade resulting from the sales of a subset of the items in G_i; and $V_i[j], S[i] \in \{1, \ldots, n\}$ denote the reference to the item in B that corresponds to the j^{th} and i^{th} vertices in V_i and S, respectively.

1. $X := \emptyset$ //X will contain n-dimensional vectors
2. For all $\vec{s} \in Dom(S)$ {
3. Initialize $\vec{b}$ //$\vec{b}$ is a n-dimensional vector
4. For $(1 \leq i \leq k)$\{
5. //Get max gains from trade of G_i consistent with $\vec{s}$
6. $\vec{v}_i^* \in \text{argmax}_{\vec{x} \in Dom(V_i)} \hat{GT}_i(\vec{x})$
7. s.t. $\vec{x}(l) = \vec{s}(m)$ if $V_i[l] = S[m]$
for some $1 \leq l \leq |V_i|$ and $1 \leq m \leq |S|$
8. For $(1 \leq j \leq |V_i|)$ $\vec{b}(V_i[j]) := \vec{v}_i^*(j)$
9. \}$X := X \cup \vec{b}$
10.\} return $\vec{b}^* \in \text{argmax}_{\vec{x} \in X} \hat{GT}(\vec{x})$

It can be shown that Algorithm 2 produces equivalent results to computing the bundle with the maximum gains from trade the straight-forward way, by generating all possible bundles. However, for large graph structures it is considerably faster (in fact the direct method is too slow to work in practice for large graph structures).

Example 3 *Consider the graph in Figure 2. Generating all bundle combinations and testing them takes 2^{p+c+q} steps. Our algorithm generates all possible combinations only for cutset C, then computes optimal sub-bundles for subgraphs A and B for each combination of C and merges them. This requires only $2^c(2^p + 2^q)$ steps. Since c in our case is of size 1 or 2, while p and q can be arbitrarily large, the decrease in the number of steps is exponential.*

More generally, suppose the decomposition of a utility graph leads to k cutset nodes ($|S| = k$) and Max$V$ denotes the number of vertices of the subgraph with the largest number of vertices. The complexity of Algorithm 2 is then $O(2^{k+MaxV})$ (proofs are omitted due to lack of space).

3.4 Updating Sub-utility Functions

The search method described in Section 3.3 works on the utility graph $\hat{G}$, and corresponding sub-utility functions $\hat{u}_i$ ($i = 1, \ldots, |\hat{C}|$, where $\hat{C}$ denotes the set of clusters). They represent the best model of the opponent (i.e. buyer) that the seller has so far. After receiving a counter offer from the buyer, he will update this model. More precisely, he will update the sub-utility functions. In this Section, we will discuss the rule for updating these sub-utility functions.

The learning algorithm determines, for all $\hat{C}_i \in \hat{C}$, which combination in $\hat{C}_i$ was asked by the buyer at the last iteration (where there are $2^{|C_i|}$ possible combinations). Then it increases the expected monetary value of the buyer for that combination and degrades the other combinations in the cluster. Intuitively, the idea is to strengthen a link between vertices (represented by the corresponding sub-utility value) whenever a buyer indeed expresses an interest in purchasing the items corresponding to the vertices; otherwise the link is weakened. Algorithm 3 gives the actual updating rules.

Example 4 *Suppose we have the cluster $C_i = \{I_3, I_5, I_6\}$ (for a $i \in \{1, \ldots, |C|\}$). The buyer's last offer contains the combination $I_3 = 0$, $I_5 = 1$, and $I_6 = 1$. Then the expected buyer utility for purchasing item 5 and 6 is increased: i.e., $u_i((0,1,1)) = u_i((0,1,1)) * (1 + \alpha(i))$. The expected utilities for all other combinations in $\{0,1\}^{|C_i|}$ (namely $u_i(1,1,0), u_i(1,0,1), u_i(1,0,0)$, etc.) are decreased.*

Algorithm 3 Algorithm for updating sub-utility functions

The seller's and buyer's last offer contain the binary assignments $\vec{b}_s$ and $\vec{b}_b$ to the variables in B; moreover $\vec{c}_{i,s}$ and $\vec{c}_{i,b}$ denote the assignments to the items in $\hat{C}_i$, induced by $\vec{b}_s$ and $\vec{b}_b$ (i.e. $\vec{c}_{i,s}$ and $\vec{c}_{i,b}$ are sub-arrays of $\vec{b}_s$ and $\vec{b}_b$).

1. For $(1 \leq i \leq |\hat{C}|)$ {
2. if $\vec{c}_{i,s} \neq \vec{c}_{i,b}$ {
3. $u_i(\vec{c}_{i,b}) := u_i(\vec{c}_{i,b}) * (1 + \alpha_c)$
4. For all $\vec{c} \in \{0,1\}^{|C_i|} \setminus \{c_{i,b}\}$
5. $u_i(\vec{c}) := u_i(\vec{c}) * (1 - \alpha(i))\}\}$

Parameter $\alpha(i)$ in Algorithm 3 defines how much weight should be given to the request from the buyer's last bid, in each cluster. A higher $\alpha(i)$ mean that the seller is more likely to give in to buyer's preferences for a cluster. A straightforward choice would be to assign the same α to all clusters. The seller would then only take into account the buyer's preferences. However, he should also take into account the expected gains from trade of a cluster, which also depend on his costs (unknown to the buyer). We therefore define a factor called the Gains from Trade Importance Ratio (GTR) as:

$$GTR(i) = \frac{GT_i(\vec{c}_{i,b}))}{GT(\vec{b}_b)} \tag{4}$$

for all $i = 1, \ldots, |C|$. In the above equation, $\vec{b}_b$ denotes the seller's last offer containing the assignment $\vec{b}_b$ to the variables in B and $c_{i,b}$ denotes the assignment to the variables in C_i, induced by $\vec{b}_b$.

Intuitively, this ratio provides a measure of the cluster's importance weight, by comparing the gains which can be obtained in this cluster to the gains for the whole bundle. The measure can be compared to the inverse of total number of clusters $1/|C|$, for the purpose of fine-tuning the alpha parameter for each cluster (although the clusters vary in size in the conducted experiments from 1-4 items, the above ratio still provides a useful heuristic). The cluster-specific α is then computed as the sum between a fixed and variable component (computed by a sigmoid function): i.e.,

$$\alpha(i) = \alpha_{fixed} + \alpha_{var} * \frac{1}{1 + e^{\beta * (GTR(i) - 1/|C|))}} \tag{5}$$

for all $i = 1, \ldots, |C|$. Here the parameter β is a positive value, which gives a measure of how steep the sigmoid function is. After conducting a number of experiments, we observed that transforming the function into a simpler step function (by assigning $\beta \to \infty$) produces reasonably good experimental results. The values assigned to the alphas in the conducted experiments are $\alpha_{fixed} = 0.1$ and $\alpha_{var} = 0.3$.

The rationale behind the above formula is the following: if a cluster has a high importance for the seller (i.e. if $GTR(i) > 1/|C|$), then the concession made for this particular cluster will be small (equal to α_{fixed}). Intuitively, this means that for this cluster, the costs of the seller are low, so the the seller should keep insisting more on offering his own values for the items in this cluster for longer, since he knows he can offer them cheaper (the buyer does not know this, because he does not know the cost structure of the seller). For clusters with relatively low gains from trade (i.e. if $GTR(i) < 1/|C|$), there is not much difference between the offer of the buyer and that of the seller - therefore the seller can concede towards the buyer's offer without much perceived utility loss.

We note that our updating rule implicitly entails that the seller uses a monotonic concession strategy, because in all clusters at least a small, non-negative concession α_{fixed} is made towards the buyer's offer. We made this choice since it assures that our algorithm guarantees convergence (i.e configuration agreement) within a limited number of negotiation steps (although how efficient this agreement is depends on the tuning of parameters).

We also note that our update model is geared to a particular class of buyer strategies, one in which the buyer will make an overall time-dependent concession at each step on either configuration or price. We can best exemplify this through the simplest strategy of this class: the case in which the buyer is "hard-headed"

about the configuration, by insisting on her own values for all the items under negotiation, though he/she is willing to concede on the price. In this case, the seller will keep trying to offer the buyer other items, but will insist more on the items for which his costs are low. However, eventually she will concede by agreeing to the configuration asked (but she can still extract a considerable concession on the price).

4 Experimental Analysis

4.1 Experimental set-up

There are several dimensions, which we want to test in our model:

- The distance to Pareto-efficiency of the outcome reached (measured, in our case, as the average percentage from the maximum possible gains from trade) as well as the robustness of this result to large variations in buyer and seller profiles.
- The time taken to reach a solution, measured as the number of negotiation steps.

We test our model in a negotiation over 50 binary-valued issues. The maximal preference graph, which the seller considers as possible, includes 28 clusters representing the direct synergy between items (either positive for complementarity or negative for substitutability effects): 4 of them contain 4 items each, 13 of them 3 items, 6 of them 2 items and 3 only 1 item. It is important to emphasize that in our representation, each cluster is represented through a utility table with one entry for all sub-combinations in the cluster (therefore of size $2^{|C|}$, where $|C|$ is the number of nodes per cluster). So, for example, for a cluster of size 3, we implicitly consider all its possible sub-clusters of size 2 and 1.

On the seller side, the cost of each of the n items is normally distributed according to $N(\mu_{cost}, \sigma_{cost})$. On the buyer side, the value of each of n individual items is normally distributed according to $N(\mu_{gains}, \sigma_{gains})$. Moreover, the synergy effects between *subsets* in each of the above 11 (non-singleton clusters) is normally distributed according to $N(0, k^2\sigma_{syn})$, where k denotes the number of items in the subset.

To somewhat limit the number of parameters, we set $\sigma = \sigma_{costs} = \sigma_{gains} = \sigma_{syn}$. The parameter σ captures the problem of finding Pareto-efficient solution very, nicely: the higher σ the higher the likelihood of complementarity and substitutability effects, hence the higher the likelihood of non-linearity, in the problem.

The mean of the cost distribution for each item μ_{costs} is always set to 1. For the mean of the distributions for the buyer μ_{gains}, we conduct experiment with 3 different values: 1.1, 1.25, 1.5. Otherwise stated, the valuations of the Buyer are, *on average* 10%, 25% to 50% greater than those of the Seller. In the reported tests, σ takes 8 values, ranging from 0 to 5. In other words, we consider to whole spectrum from no randomness, and consequently linear preferences, to a very high degree of randomness, and consequently (with probability) highly nonlinear preferences. Recall from Section 3.4 that the seller uses a subutility functions in conjunction with the utility graph to model a buyer. These functions are initialized such that the seller on average correctly predicts the opponent's utility. Already for relatively small values of σ the initially predicted utilities will (with all likelihood) be very different from the actual values, however.

For each combinations of values of μ_{costs}, μ_{gains}, and σ, 100 tests were performed. This means in total: $1 * 3 * 8 * 100 = 2400$ tests were performed (or 300 tests for each point σ reported in the figures below. For this number of tests per points the variance of the results does not decrease, we can therefore conclude that this number of tests provides statistically significant results.

4.2 Experimental results

The experimental results produced from the setting described above are given in Fig. 3 and 4. As mentioned, each point was produced from 300 negotiations and the error bars give the resulting variance. The two figures highlight the results with respect to reaching an agreement over the bundle configuration (i.e., the actual content of the bundle). By agreement, we mean that thereafter the bundle content no longer changes. After such an agreement is reached, it may take more negotiation rounds before bargainers agree upon the price (if an agreement is reached at all). The Pareto-efficiency of a deal — which is the focus of this paper — is then already determined, however.

There are several conclusions that can be drawn from the analysis of these results. Regarding the ability to find a bundle close to maximum efficiency (Fig. 3), we can conclude that our approach performs very well. (Please keep in mind that the Y axis in Fig. 3 is scaled for values between 80% and 100% of optimal). Even for very high values of σ, the algorithm is able to extract around 97% of the maximum gains from trade, with a variance of maximal 5%. This is particularly remarkable for very large values of σ (e.g., σ between 4-5). For these values, (with all likelihood) the problem becomes highly nonlinear and the buyer's utilities predicted initially by the seller will be very different from the actual values.

Regarding the number of steps needed to find the optimal configuration (Fig. 4), we do see a significant increase in this number for larger values of σ. Clearly, problems for larger σ are more difficult to solve, so more steps are needed to correctly update the model of buyer's preferences and to find a good bundle. Nevertheless, a bundle very close to maximal efficiency is usually found, even in these more difficult cases.

Thus, the results strongly support the underlying idea put forward in this paper. That is, having a maximal utility graph of possible interdependencies can be used to successfully navigate the contract space and reach Pareto-efficiency with a limited number of steps, even for a relative large number of interdependent issues.

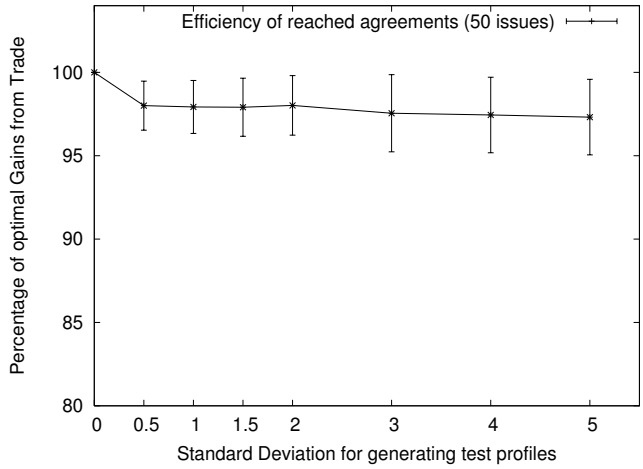

Figure 3: Percentage of the Gains from Trade (compared to the Pareto-optimal bundle) achieved over 100 tests

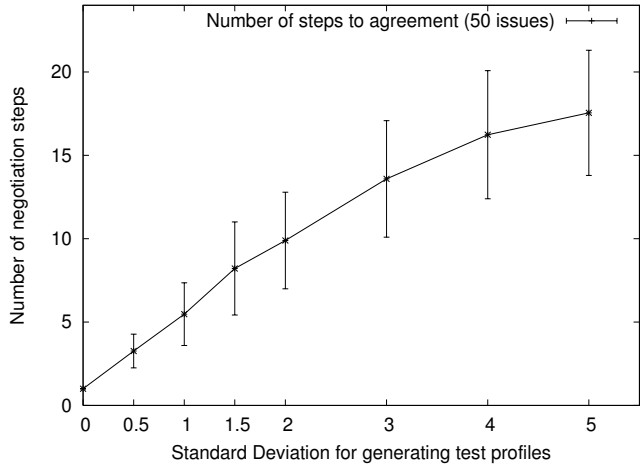

Figure 4: Number of steps needed to configuration agreement

5 Discussion

In this section we provide a review of related work, with special attention to the features relevant for our approach. We summarize the main contributions of our work. Subsequently, we discuss follow-up work and identify directions for future research.

5.1 Related work and contributions

Most of the work in multi-issue negotiations has focused on the independent valuations case. Faratin, Sierra & Jennings [10] a method to search the utility space over multiple attributes is introduced, which uses fuzzy similarity criteria between attribute value labels as prior information. Coehoorn and Jennings [5] extends this model with a method to learn the preference weights that the opponent assigns to different issues in the negotiation set, by using kernel density estimation. Jonker and Robu [11] consider a similar model, the prior information are not fuzzy similarity criteria but a partial ordering of value labels. Both these papers have the advantage that they allow flexibility in modeling and deal with incomplete preference information supplied by the negotiation partner. They do not consider the question of functional interdependencies between issues, however.

Other approaches to multi-issue negotiation problem are the agenda based approach (Fatima et. al. [9]) and the constraint-based negotiation approach (Luo et. al. [22]). Both provide comprehensive formal analyses. They do not address the scalability and interdependence, however. Debenham [7] proposes a multi-issue bargaining strategy that models the iterative information gathering which takes place during the negotiation. Unlike our approach, the agents in [7] do not model the preferences of their opponent, but construct a probability distribution over all possible outcomes. However, this paper does not consider efficiency criteria for the reached outcomes (such as Pareto-efficiency), nor does it discuss scalability issues (it provides a convincing example, but restricted to only two issues).

Two negotiation approaches that specifically address the problem of complex inter-dependencies between multiple issues — and are therefore most related to our work — are [12, 14]. Klein et. al. [12] use a setting similar to the one considered in this paper, namely bilateral negotiations over a large number of boolean-valued issues with binary interdependencies (although we also allow higher-level interdependencies, up to 4 items). In this setting, they compare the performance of two search approaches: hill-climbing and simulated annealing and show that if both parties agree to use simulated annealing, then Pareto-efficient outcomes can be reached. In a similar line of work, Lin [14] uses evolutionary search techniques to reach optimal solutions. Both of these approaches have the advantage that they are scalable to large numbers of issues and Pareto-efficient outcomes can be reached without any prior information. However, a drawback of these learning techniques is the large number of negotiation steps needed to reach an agreement (around 2000 for 50 issues [12]).

By comparison with this work, our approach uses an explicit model of the buyer utility function - in the form of a utility graph. Although we assume no prior information about the preferences of any given opponent, we do assume that the maximal super-set of all possible interdependencies which are possible in the domain is known. This allows us to restrict the size of the search space and therefore reach Pareto-efficient agreements considerably faster. To some degree our approach could be also compared to solutions proposed to determine the winner in combinatorial auctions (Conitzer et. al. [6]). The problem is, in the general case, intractable, but by imposing some constraints on the type of bids which can be specified, efficient solutions can be found.

Our solution should be applicable in complex domains where fast agreements must be reached, by using some prior information about the structure of the utility space to be explored. This prior information could be a history of past negotiations or the input of domain experts. The relatively small number of steps needed to reach an agreement in our model allows it to be used in time-constrained negotiations or negotiation where impatience of one of the parties is a limiting factor. This is a significant result since, to the best of our knowledge, bargaining under time pressure in negotiations with many, inter-dependent issues was not considered in previous literature. Furthermore, in our setting, no trusted mediator is needed and the negotiating agents keep their preference information (i.e. monetary utility, respectively costs) private.

Another direction of research relevant to our work is the one on graphical representations of utility functions. Bacchus and Grove [3] provide an early fundamental discussion of the graphical modeling of utility

and preference functions and introduce the concept of conditionally additive independence. Their work is mostly concerned with formally analysing the semantics and properties of graphical utility representations. Chajewska and Koller [4] build on this idea, by discussing the decomposability of non-linear utility functions into clusters and, in addition, they present an efficient procedure for eliciting such functions from human users, in a medical application domain.

In comparison with these works, our approach is geared towards finding an efficient algorithm which exploits the structure of utility graphs to model an online learning problem, such as negotiation, where agent preferences are revealed only indirectly, through repeated offers and counter-offers. Therefore we adapt the utility graph formalism for our setting, to assure a good trade-off between representation power of complex utility functions on one side, and computational efficiency in exploring the large bundle space on the other. To the best of our knoweldge, very little work exists which applies this powerful formalism to settings which require online learning on the part of the agents, especially negotiation or virtual market settings.

5.2 Subsequent work and future research

A important issue which remains is the construction of the structure of the utility graph from scratch, without any prior information. One could focus here on developing automatic and advanced techniques, which could be inspired by learning in graphical models ([13, 16]). For this purpose, aggregate customer information (for example information regarding all previous negotiations) could be used. This problem is addressed in the companion paper [20].

In [20], the results of [19] are extended by proposing a method for constructing the utility graphs of buyers automatically, based on previous negotiation data. The method is based on techniques inspired from item-based collaborative filtering, used in online recommendation algorithms. Experimental analysis shows that the approach is able to retrieve the structure of utility graphs online, with a high degree of accuracy, even for highly non-linear settings and even if a relatively small amount of data about concluded negotiations is available.

Future research could follow two directions. One direction is extending the current model to handle simultaneous one-one negotiation threads. For this, the problem of syncronising the exchange of offers and especially the concessions made between parties remains an important open issue. Another direction of potential further research is the applicability of the utility graph concept to model agent decision making in other settings. For example, in virtual market scenarios, utility graphs could be used to devise efficient bidding policies for simultaneous, sequential or repeated auctions.

References

[1] P.S. Albin. *Barriers and Bounds to Rationality: Essays on Economic Complexity and Dynamics in Interactive Systems.* Princeton University Press, Princeton, NJ, USA, 1998.

[2] Andreu Mas-Collel, Michael D. Whinston, and Jerry R. Green. *Mircoeconomic Theory*. Oxford University Press, 1995.

[3] F. Bacchus and A. Grove. Graphical models for preference and utility. In *Uncertainty in Artificial Intelligence UAI-95*, pages 3–10, 1995.

[4] U. Chajewska and D. Koller. Utilities as random variables: Density estimation and structure discovery. In *Proceedings of sixteenth Annual Conference on Uncertainty in Artificial Intelligence UAI-00*, pages 63–71, 2000.

[5] R. M. Coehoorn N. R. Jennings. Learning an opponent's preferences to make effective multi-issue negotiation tradeoffs. In *Proc. 6th Int Conf. on E-Commerce, Delft, The Netherlands*, pages 59–68, 2004.

[6] V. Conitzer, J. Derryberry, and T. Sandholm. Combinatorial auctions with structured item graphs. In *In Proc. of the National Conf. on Artificial Intelligence (AAAI)*, 2004.

[7] J. Debenham. Bargaining with information. In *3rd Int. Conf. on Autonomous Agents & Multi Agent Systems (AAMAS), New York, July 19-23, 2004*, pages 663–670, 2004.

[8] P Faratin, C Sierra, and N.R Jennings. Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, 24(3–4):159–182, 1998.

[9] N. Jennings S. Fatima, M. Woolridge. Optimal negotiation of multiple issues in incomplete information settings. In *3rd Int. Conf. on Autonomous Agents & Multi Agent Systems (AAMAS), New York*, pages 1080–1087, 2004.

[10] N. R. Jennings P. Faratin, C. Sierra. Using similarity criteria to make issue trade-offs in automated negotiations. *Journal of Artificial Intelligence*, 142(2):205–237, 2002.

[11] C. Jonker V. Robu. Automated multi-attribute negotiation with efficient use of incomplete preference information. In *3rd Int. Conf. on Autonomous Agents & Multi Agent Systems (AAMAS), New York*, pages 1056–1063, 2004.

[12] M. Klein, P. Faratin, H. Sayama, and Y. Bar-Yam. Negotiating complex contracts. *Group Decision and Negotiation*, 12:111–125, 2003.

[13] S. L. Lauritzen. *Graphical Models*. Oxford University Press, Oxford UK, 1996.

[14] R. Lin. Bilateral multi-issue contract negotiation for task redistribution using a mediation service. In *Proc. Agent Mediated Electronic Commerce VI*, New York, USA, 2004.

[15] A. R. Lomuscio, M. Wooldridge, and N. R. Jennings. A classification scheme for negotiation in electronic commerce. *Group Decision and Negotiation*, 12:31–56, 2003.

[16] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, San Mateo, California, 1988.

[17] H. Raiffa. *The art and science of negotiation*. Harvard University Press, Cambridge, Massachussets USA, 1982.

[18] R. L. Rivest T. H. Cormen, C. E. Leiserson. *Introduction to algorithms*. The MIT Press, 1989.

[19] V. Robu, K. Somefun, H. La Poutré. Modeling Complex Multi-Issue Negotiations Using Utility Graphs. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS)*, ACM Press, pages 280 – 287, 2005.

[20] V. Robu, H. La Poutré. Learning the Structure of Utility Graphs Used in Multi-Issue Negotiation through Collaborative Filtering. In *Proceedings of the Pacific Rim International Workshop on Multi-Agents (PRIMA'05), Kuala Lumpur 2005*, Springer Lecture Notes in Artificial Intelligence (LNCS / LNAI), Springer-Verlag, to appear.

[21] T.S. Sargent. *Bounded Rationality in Macroeconomics.* Oxford University Press, New York, NY, USA, 1993.

[22] N. Shadbolt H. Leung J. H. Lee X. Luo, N. R. Jennings. A fuzzy constraint based model for bilateral multi-issue negotiations in semi-competitive environments. *Artificial Intelligence Journal*, 142 (1-2):53–102, 2003.

[23] D.J.A. Somefun, T.B. Klos, and J.A. La Poutré. Online learning of aggregate knowledge about non-linear preferences applied to negotiating prices and bundles. In *Proc. 6th Int Conf. on E-Commerce (ICEC), Delft*, ACM Press, pages 361–370, 2004.

Expressivity of finitary coalgebraic logics

C. Kupke

Abstract

We give a brief introduction to the theory of coalgebras and discuss the connection between coalgebra and modal logic. In the second more technical part of this paper - summarizing known results - we prove that the finitary version of Moss' coalgebraic logic has the Hennessy-Milner property.

Summary

The purpose of this short paper is twofold. On the one hand we want to draw the attention of the more general audience to the relatively young field of universal coalgebra and its relationship with modal logics. On the other hand we present two facts about coalgebras which might be interesting for the part of the audience that has already some background knowledge in coalgebra. Therefore in the first part of the paper we will try to explain the notion of a coalgebra and of coalgebraic bisimulations by stating the definitions and by providing several examples. Nevertheless the paper cannot be seen as a proper introduction to coalgebras. The interested reader is referred to [4, 11] for an introduction to the field of universal coalgebra. Readers whose background lies in modal logic and its algebraic semantics are recommended to consult the recently published [13].

The paper also contains two technical results: In Section 2 we establish a link between the bisimilarity game and the final sequence of a functor (cf. Theorem 2.9). In Section 4 we obtain - following the ideas of Moss in [7] - the result that finitary coalgebraic logics have the Hennessy-Milner property on coalgebras for an ω-accessible, weak pullback preserving set functor (cf. Theorem 4.1).

1 Coalgebras

A coalgebra consists of a set of *states* [1] and a coalgebra map which can be used to make *observations* about coalgebra states. The type of these observations is specified by a functor. Often it is possible to assign to every coalgebra state x its *behavior* which can be thought of as a sequence of observations generated by repeatedly applying the coalgebra map to x and its successors. For example we can think of a coalgebra in which we can observe for each state x a letter a of some alphabet Σ and some successor state x'. In this case the behaviour of x will be an infinite Σ-word. Unlike in algebra, elements of a coalgebra are not constructed from generators and operations. In fact one should think of the elements of the coalgebra as given from the outset. Using the coalgebra map one can obtain (limited) information about them. This makes it possible to use coalgebras for modeling objects with infinite or non-wellfounded "behavior" such as infinite words, trees or graphs. A reader familiar with functional programming languages such as Haskell could obtain some intuition about coalgebras by thinking of them as of lazy implementations of infinite objects.

1.1 The formal definition

When introducing coalgebras it is almost inevitable to use category theoretic terminology. It should be noted, however, that for the understanding of this paper only an intuitive understanding

[1] This is not the most general definition, but we will only consider coalgebras that have a carrier set.

of the notion of a category and of a functor between categories is necessary. In fact we will mostly work in the category Set that has sets as *objects* and functions as *arrows*. A functor is a structure preserving map between categories. A set functor $\mathrm{F} : \mathsf{Set} \to \mathsf{Set}$ maps sets to sets and functions to functions. Furthermore F preserves the composition of functions and identities, i.e. $\mathrm{F}(f \circ g) = \mathrm{F}f \circ \mathrm{F}g$ and $\mathrm{F}\mathrm{id}_X = \mathrm{id}_{\mathrm{F}X}$ where for an arbitrary set X we denote by id_X the identity function on X.

Definition 1.1 Let $\mathrm{F} : \mathsf{Set} \to \mathsf{Set}$ be a functor. Then an F-*coalgebra* is a pair (X, γ) where $X \in \mathsf{Set}$ and $\gamma : X \to \mathrm{F}X \in \mathsf{Set}$. A *pointed* F-coalgebra $(\langle X, \gamma\rangle, x)$ is an F-coalgebra $\langle X, \gamma\rangle$ together with a designated point $x \in X$. $\lhd$

Example 1.2

1. Let Σ be a set (of colors). Furthermore let B_Σ be the functor that maps a set X to the set $\Sigma \times X \times X$ and a function $f : X \to Y$ to the function that maps a triple $(c, x, x') \in \Sigma \times X \times X$ to the triple $(c, f(x), f(x'))$. Then infinite Σ-labeled binary trees are pointed B_Σ-coalgebras: a given Σ-labeled tree $t : 2^* \to \Sigma$ corresponds to the coalgebra with carrier set 2^* and coalgebra map $w \mapsto (t(w), w0, w1)$.
2. Deterministic, finite Σ-word automata can be modeled as pointed coalgebras for the functor $\mathrm{A}_\Sigma = 2 \times (\_)^\Sigma$, where Σ is the finite alphabet and 2 is the two-element set, i.e. the functor A_Σ maps a set X to the set $2 \times X^\Sigma$ and a function $f : X \to Y$ to the function that maps a pair $(o, t) \in 2 \times X^\Sigma$ to the pair $(o, f \circ t)$. More about this fundamental example can be found in [9].
3. Image-finite Kripke frames correspond to coalgebras of the power set functor $\mathcal{P}_\omega$. The functor $\mathcal{P}_\omega$ maps a set to its collection of *finite subsets*, and a function $f : X \to Y$ to its direct image function $\mathcal{P}_\omega f : \mathcal{P}_\omega X \to \mathcal{P}_\omega Y$ given by $(\mathcal{P}_\omega f)(U) := \{f(x) \in Y \mid x \in U\}$. An image-finite Kripke frame (W, R) is then modeled as the coalgebra $(W, \lambda x.R[x])$, that is, the relation R is given by the function mapping a state x to the collection $R[x]$ of its (direct) successors.[2]
4. Let Φ be a set (of propositional variables) and consider the functor $\mathcal{P}\Phi \times \mathcal{P}_\omega$, that maps a set X to the set $\mathcal{P}\Phi \times \mathcal{P}_\omega X$ and a function $f : X \to Y$ to the function $\mathrm{id}_{\mathcal{P}\Phi} \times \mathcal{P}_\omega f$. Then an image finite Kripke model $(W, R, V : \Phi \to \mathcal{P}(W))$ corresponds to the $\mathcal{P}\Phi \times \mathcal{P}_\omega$-coalgebra $\langle W, \langle V^\sharp, \lambda x.R[x]\rangle : W \to \mathcal{P}\Phi \times \mathcal{P}_\omega W\rangle$ where $p \in V^\sharp(w)$ if $w \in V(p)$.
5. Let $\mathcal{D}_\omega$ be the functor that maps a set S to the set

$$\mathcal{D}_\omega S := \left\{\rho : S \rightharpoonup [0,1] \mid \rho \text{ has finite support and } \textstyle\sum_{s\in S}\rho(s) = 1\right\}$$

where we say that a (partial) function ρ has finite support if $\rho(s) \neq 0$ for only finitely many elements s of S. Then coalgebras for the functor $1 + \mathcal{D}_\omega$ correspond to the probabilistic transition systems by Larsen and Skou in [6]. Here $1 + \mathcal{D}_\omega$ denotes the functor that maps a set S to the disjoint union of the one-element set and the set $\mathcal{D}_\omega S$. Further details about this example can be found in [7, 12].

The message of these examples should be clear: coalgebras can be used to uniformly model various types of transition systems. The advantage of such a uniform approach lies in the fact that notions, that are usually studied in isolation for different system types, can now be seen as instances of one coalgebraic concept. As examples for this phenomenon we are now going to present the notions of a coalgebra morphism and of a coalgebraic bisimulation. After that we look at some concrete instances of these coalgebraic notions.

Definition 1.3 Let $\mathbb{X} = \langle X, \gamma\rangle$ and $\mathbb{Y} = \langle Y, \delta\rangle$ be two F-coalgebras then a function $f : X \to Y$ is an F-coalgebra morphism from $\mathbb{X}$ to $\mathbb{Y}$ if $\delta \circ f = \mathrm{F}f \circ \gamma$, i.e. if the left diagram in Figure 1 commutes.

A relation $Z \subseteq X \times Y$ is an F-bisimulation between $\mathbb{X}$ and $\mathbb{Y}$ if there exists a function $\mu : Z \to \mathrm{F}Z$ such that the projection maps $\pi_1 : Z \to X$ and $\pi_2 : Z \to Y$ are coalgebra morphisms, i.e., such that the right diagram in Figure 1 commutes. Two states are *(*F*-)bisimilar* if they are linked by some bisimulation. We write $x \leftrightarrows_{\mathrm{F}} y$ if the states x and y are F-bisimilar. $\lhd$

[2] If we drop the finiteness condition from the definition of the functor $\mathcal{P}_\omega$ we obtain the power set functor $\mathcal{P}$. $\mathcal{P}$-coalgebras are Kripke frames.

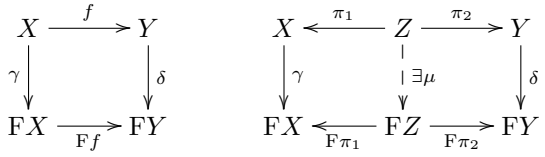

Figure 1: F-coalgebra morphism and F-bisimulation

Let us have a look at some concrete instances of F-coalgebra morphisms and F-bisimulations.

Example 1.4 1. Let $\mathrm{F} = \mathrm{B}_\Sigma$ (cf. Example 1.2(1)) for some set of labels Σ. Then $f : X \to Y$ is a B_Σ-coalgebra morphism between two B_Σ-coalgebras $\langle X, \langle c, \gamma\rangle : X \to \Sigma \times (X \times X)\rangle$ and $\langle Y, \langle d, \delta\rangle : Y \to \Sigma \times (Y \times Y)\rangle$ if for all $x \in X$ we have $c(x) = d(f(x))$ and if $\gamma(x) = (x_1, x_2)$ implies $\delta(f(x)) = (f(x_1), f(x_2))$. In words, f has to preserve the labeling and the binary tree structure. It is not difficult to check that two binary trees t_1 and t_2 are B_Σ-bisimilar iff they are identical.

2. Let $\mathrm{F} = \mathrm{A}_\Sigma = 2 \times (\_)^\Sigma$. Then pointed F-coalgebras with finite carrier set correspond to finite, deterministic automata (cf. Example 1.2(2)). Two automata accept the same language iff the corresponding pointed F-coalgebras are F-bisimilar (cf. [9]).

3. Recall from Example 1.2(4) that F-coalgebras for the functor $\mathrm{F} = \mathcal{P}\Phi \times \mathcal{P}_\omega$ correspond to image-finite Kripke models. Then F-coalgebra morphisms are precisely the bounded morphisms between Kripke models. F-bisimulations correspond exactly to the bisimulations from modal logic, i.e. a relation R between two (image-finite) Kripke models (W_1, R_1, V) and (W_2, R_2, V) is a modal bisimulation iff R is a $\mathcal{P}\Phi \times \mathcal{P}_\omega$-bisimulation between the corresponding $\mathcal{P}\Phi \times \mathcal{P}_\omega$-coalgebras.

4. Let F be the functor $1 + \mathcal{D}_\omega$ (cf. Example 1.2(5)). It can be shown that F-bisimulations between F-coalgebras coincide with the probabilistic bisimulations from [6] between the corresponding probabilistic transition systems. For the details we again refer to [7, 12].

In Section 2 we will continue our discussion of bisimilarity by showing that coalgebraic bisimilarity has also a nice game-theoretic interpretation. Furthermore in Section 3 we will recall the definition of some logical language that can be used for reasoning about F-coalgebras. In the remainder of this section we will fix the technical preliminaries.

1.2 Technicalities

In the following we will make some assumptions on the functor F under consideration. Readers who are not interested in all the technical details are advised to skip this section and to think of F in the sequel as of some functor in Example 1.2.

The first assumption we make is that all our functors are set functors. Furthermore we will assume the functors we are working with to be

- *standard:* for all set X and Y such that $X \subseteq Y$ we have that $\mathrm{F}X \subseteq \mathrm{F}Y$ and the inclusion map $\iota_{X,Y}$ from X into Y is mapped by F to the inclusion map $\iota_{\mathrm{F}X,\mathrm{F}Y}$ from $\mathrm{F}X$ into $\mathrm{F}Y$.
- *ω-accessible:* for all sets Y and all elements $t \in \mathrm{F}Y$ there is a finite set $X \subseteq_\omega Y$ such that $t \in \mathrm{F}X$.

Moreover all the functors that we are considering are *weak pullback preserving*. This is a property which often occurs in the coalgebraic literature. For details about this important, but rather technical condition we refer the reader to [11]. From now on we assume F to be a standard, ω-accessible and weak pullback preserving set functor.

1.3 Relation Lifting

A useful property of F is that it has a well-behaved *relation lifting*. We will first define this relation lifting, then list the properties we need. In the sequel this relation lifting allows us to use a concise formulation of the notion of a coalgebraic bisimulation. In addition to that it is central in the definition of the coalgebraic logic we are considering.

Definition 1.5 Given two sets X and Y, and a binary relation Z between $X \times Y$, we define the *lifted relation* $\overline{\mathrm{F}}(Z) \subseteq \mathrm{F}X \times \mathrm{F}Y$ as follows:

$$\overline{\mathrm{F}}(Z) := \{((\mathrm{F}\pi)(\phi), (\mathrm{F}\pi')(\phi)) \mid \phi \in \mathrm{F}Z\},$$

where $\pi : Z \to X$ and $\pi' : Z \to Y$ are the projection functions given by $\pi(x, y) = x$ and $\pi'(x, y) = y$. ◁

Example 1.6 1. Let Σ be a set of colors and let $\mathrm{F} = \mathrm{B}_\Sigma$ and let $R \subseteq X \times Y$ be a relation. Then for arbitrary elements $(c, x_1, x_2) \in \mathrm{B}_\Sigma X$ and $(d, y_1, y_2) \in \mathrm{B}_\Sigma Y$ we have $((c, x_1, x_2), (d, y_1, y_2)) \in \overline{\mathrm{F}}R$ iff $c = d$ and $(x_i, y_i) \in R$ for $i = 1, 2$.

2. Let Φ be a set (of propositional variables), let $\mathrm{F} := \mathcal{P}\Phi \times \mathcal{P}_\omega$ and let $R \subseteq X \times Y$ be a relation. Then for arbitrary $P \subseteq \Phi$, $U \subseteq_\omega X$, $Q \subseteq \Phi$ and $V \subseteq_\omega Y$ we have

$$((P, U), (Q, V)) \in \overline{\mathrm{F}}R \quad \text{iff} \quad \begin{array}{ll} (\textsc{Prop}) & P = Q \text{ and} \\ (\textsc{Forth}) & \forall u \in U. \exists v \in V \text{ s.t.} (u, v) \in R \text{ and} \\ (\textsc{Back}) & \forall v \in V. \exists u \in U \text{ s.t.} (u, v) \in R. \end{array}$$

Readers familiar with modal logic or process algebra will recognize the similarity of the relation lifting for the functor $\mathcal{P}\Phi \times \mathcal{P}_\omega$ and the usual definition of a bisimulation between Kripke frames/transition systems. This is not a coincidence as we will see when looking at the formulation of the notion of an F-bisimulation in terms of the relation lifting (cf. Fact 2.1 below). For now we just list some properties of the relation lifting of a standard and weak pullback preserving set functor.

Fact 1.7 *The relation lifting $\overline{\mathrm{F}}$ satisfies the following properties, for all functions $f : X \to Y$, all relations $R, Q \subseteq X \times Y$, and all subsets $U \subseteq X$, $V \subseteq Y$:*
(1) $\overline{\mathrm{F}}$ extends F: $\overline{\mathrm{F}}(Gr(f)) = Gr(\mathrm{F}f)$; ($Gr(f)$ denotes the graph of f)
(2) $\overline{\mathrm{F}}$ commutes with relation converse: $\overline{\mathrm{F}}(R^\smile) = (\overline{\mathrm{F}}R)^\smile$; ($R^\smile$ denotes the converse of R)
(3) $\overline{\mathrm{F}}$ is monotone: if $R \subseteq Q$ then $\overline{\mathrm{F}}(R) \subseteq \overline{\mathrm{F}}(Q)$;
(4) $\overline{\mathrm{F}}$ distributes over composition: $\overline{\mathrm{F}}(R \circ Q) = \overline{\mathrm{F}}(R) \circ \overline{\mathrm{F}}(Q)$;
(5) $\overline{\mathrm{F}}$ commutes with restrictions, i.e. $\overline{\mathrm{F}}(R \restriction_{U \times V}) = (\overline{\mathrm{F}}R) \restriction_{\mathrm{F}U \times \mathrm{F}V}$.

For proofs we refer to [7, 1], and references therein.

2 Finite approximations of bisimilarity

In this section we state the definition of the bisimilarity game for coalgebras. Furthermore we discuss a variation of the bisimilarity game, the n-bisimilarity game. Finally we link the game-theoretic perspective to the categorical notion of the final sequence of a functor.

As a preparation we recall that F-bisimulations can be characterized using the relation lifting of the functor F.[3]

Fact 2.1 *[10] Let $\mathbb{X} = \langle X, \gamma \rangle$ and $\mathbb{Y} = \langle Y, \delta \rangle$ be two* F*-coalgebras. A relation $Z \subseteq X \times Y$ is an* F*-bisimulation between $\mathbb{X}$ and $\mathbb{Y}$ if $(\gamma(x), \delta(y)) \in \overline{\mathrm{F}}(Z)$ for all $(x, y) \in Z$.*

This fact plays an important rôle in the definition of the bisimilarity game.

[3] This fact goes back to the work by Hermida and Jacobs in [3].

2.1 Bisimilarity game

The definition of the bisimilarity game for coalgebras goes back to Baltag (cf. [1]). The idea is as follows: the game is played between the two players $\exists$ (Éloise) and $\forall$ (Abélard). Given two pointed F-coalgebras $(\mathbb{X}, x)$ and $(\mathbb{Y}, y)$, $\exists$ claims that (x, y) are bisimilar, i.e. her claim is that *there exists* some F-bisimulation Z between $\mathbb{X}$ and $\mathbb{Y}$ that relates x and y. The game board consists of pairs of coalgebra states (x, y) ($\exists$'s positions) and relations $Z \subseteq X \times Y$ ($\forall$'s positions). At a position $(x, y) \in X \times Y$ it is $\exists$'s turn and she has to move to a "local bisimulation", i.e. a relation Z such that $(\gamma(x), \delta(y)) \in \overline{\mathrm{F}}Z$ (hence Z fulfills the condition from Fact 2.1 "locally"). In turn, $\forall$ has to move from a position Z to a pair of states $(x', y') \in Z$. A match of the game is lost by a player who cannot move. All infinite plays are won by $\exists$. The following definition introduces the bisimilarity game more formally - readers who are not so familiar with the game-theoretic notions are referred to [14] for a short summary of the necessary details and to [2] for a detailed introduction to infinite (parity) graph games.

Definition 2.2 Let $\mathbb{X} := \langle X, \gamma \rangle$ and $\mathbb{Y} := \langle Y, \delta \rangle$ be F-coalgebras. Then the arena of the *image-finite F-bisimilarity game* $\mathcal{B}(\mathbb{X}, \mathbb{Y})$ is given by the bipartite graph $(B_\exists, B_\forall, E)$ that is described by the following table:

Position: b	Player	Admissible moves: $E[b]$
$(x, y) \in X \times Y$	$\exists$	$\{Z \subseteq X \times Y \mid (\gamma(x), \delta(y)) \in \overline{\mathrm{F}}Z\}$
$Z \in \mathcal{P}(X \times Y)$	$\forall$	$\{(x', y') \mid (x', y') \in Z\}$

Here the second column indicates whether a given position b belongs to player $\exists$ or $\forall$, i.e. whether $b \in B_\exists$ or $b \in B_\forall$, and $\overline{\mathrm{F}}Z$ is the relation lifting of Z. A match of $\mathcal{B}(\mathbb{X}, \mathbb{Y})$ starts at some position $b_0 \in B_\exists \cup B_\forall$ and proceeds as follows: at position $b \in B_\exists$ player $\exists$ has to move to a position $b' \in E[b]$ and likewise at position $b \in B_\forall$ player $\forall$ has to move to some $b' \in E[b]$. A player who cannot move ("gets stuck") loses the match and all infinite matches are won by $\exists$. Two successive moves in a match are called a *round* of the game. $\lhd$

Fact 2.3 *Let $\mathbb{X} := \langle X, \gamma \rangle, \mathbb{Y} := \langle Y, \delta \rangle$ be F-coalgebras and let $x \in X, y \in Y$. Then $\langle X, \gamma \rangle, x \leftrightarrows \langle Y, \delta \rangle, y$ iff $\exists$ has a winning strategy in the game $\mathcal{B}(\mathbb{X}, \mathbb{Y})$ starting at position (x, y).*

For a sketch of the proof of this fact cf. [14]. The game-theoretic analysis of F-bisimulations naturally leads to the notion of n-bisimilarity.

Definition 2.4 We say that x and y are n-bisimilar (notation: $(\langle X, \gamma \rangle, x) \leftrightarrows_n (\langle Y, \delta \rangle, y)$) if $\exists$ has a strategy in the game $\mathcal{B}(\mathbb{X}, \mathbb{Y})$ starting at position (x, y) such that $\exists$ does not lose the game in less than n rounds. $\lhd$

The next subsection will be devoted to proving that n-bisimilarity coincides with bisimilarity given our assumption that the functor F is ω-accessible.

2.2 Final sequence

We now introduce the "finitary part" of the final sequence of a functor, i.e. its first ω elements. This final sequence plays an important rôle in the theory of coalgebras where it is used in order to compute or approximate the final F-coalgebra. We only state and motivate the basic definition. For more details about the final sequence of a functor we refer the reader to [15] and references therein.

Definition 2.5 Given a set functor F we inductively define functions $p_i : \mathrm{F}^{i+1}1 \to \mathrm{F}^i 1$ for all $i \in \mathbb{N}$ by putting $p_0 := !_{\mathrm{F}1}$ and $p_{i+1} := \mathrm{F}p_i$. Here 1 denote the one-element set, $!_X$ denotes the (unique) function from a set X to the one-element set 1 and for a set X we write $\mathrm{F}^0 X := X$ and $\mathrm{F}^{i+1}X := \mathrm{F}(\mathrm{F}^i X)$. For all $n \in \mathbb{N}$ elements of the set $\mathrm{F}^n 1$ will be called *n-step behavior*. $\lhd$

The sequence is depicted in the lower part of Figure 2. Let us give an example which illustrates why we chose the name "n-step behavior" for members of the nth element of the sequence.

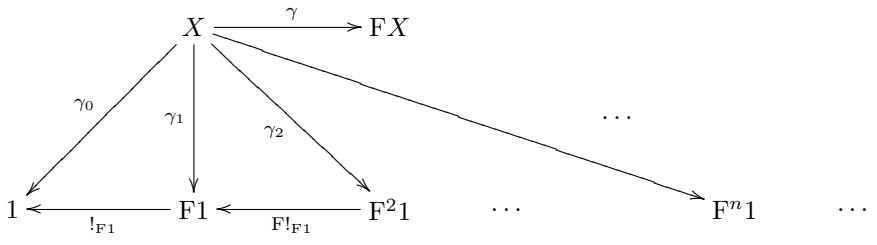

Figure 2: Final sequence and n-step behavior maps

Example 2.6 Let Σ be set of labels and let $\mathrm{F} = \mathrm{B}_\Sigma$. Then it is not difficult to see that for all $n \in \mathbb{N}$ the set $\mathrm{F}^n 1$ is isomorphic to the set of finite Σ-labeled binary trees of depth n.

We will make use of the fact that given an F-coalgebra $\langle X, \gamma \rangle$ one can easily define a sequence $\{\gamma_n\}_{n \in \mathbb{N}}$ of functions that map any coalgebra state $x \in X$ to its n-step behavior. We now give the formal definition of these γ_n's, Figure 2 provides a picture of the situation.

Definition 2.7 Given an F-coalgebra $\mathbb{X} = \langle X, \gamma \rangle$ and a state $x \in X$ we define a family of maps $\{\gamma_n : X \to \mathrm{F}^n 1\}_{n \in \mathbb{N}}$ by putting

$$\begin{array}{lcl} \gamma_0(x) & := & !_X \\ \gamma_{i+1}(x) & := & \mathrm{F}\gamma_n \circ \gamma \end{array}$$ $\lhd$

Intuitively for each $n \in \mathbb{N}$ the map γ_n maps a state x to its n-step behavior.

Example 2.8 In the above example for $\mathrm{F} = \mathrm{B}_\Sigma$ recall from Example 1.2 that a Σ-labeled infinite binary tree t corresponds to a B_Σ-coalgebra. In this case, intuitively speaking, the map γ_n maps t to the Σ-tree of depth n that consists of the first n levels of t.

For much more information and results about the final sequence of an ω-accessible functor the reader is referred to [15]. We now make a connection between the bisimilarity game and the final sequence of F.

Theorem 2.9 *Let $(\langle X, \gamma \rangle, x)$ and $(\langle Y, \delta \rangle, y)$ be pointed* F*-coalgebras. Then for all $n \in \mathbb{N}$ we have*

$$\gamma_n(x) = \delta_n(y) \quad \textit{iff} \quad x \leftrightarrows_n y.$$

Proof. For $i \in \mathbb{N}$ let $Z_i := Gr(\gamma_i) \circ Gr(\delta_i)^\smile$ and put $B_i := \{(x', y') \in X \times Y \mid (\langle X, \gamma \rangle, x) \leftrightarrows_n (\langle Y, \delta \rangle, y)\}$. The claim of the proposition is equivalent to saying that

$$Z_i = B_i \quad \text{for all } n \in \mathbb{N} \tag{1}$$

In order to facilitate the proof of (1) we first show that the following holds for all $(x, y) \in X \times Y$ and all $i \in \mathbb{N}$:

$$(x, y) \in Z_{i+1} \quad \text{iff} \quad (\gamma(x), \delta(y)) \in \overline{\mathrm{F}} Z_i. \tag{2}$$

This can be seen by just spelling out the definitions and by using Fact 1.7:

$$\begin{array}{lll} (x, y) \in Z_{i+1} & \text{iff} & (x, y) \in Gr(\gamma_{i+1}) \circ Gr(\delta_{i+1})^\smile \qquad \text{(Def. of } Z_{i+1}) \\ & \text{iff} & (x, y) \in Gr(\mathrm{F}\gamma_i \circ \gamma) \circ Gr(\mathrm{F}\delta_i \circ \delta)^\smile \qquad \text{(Def. of } \gamma_{i+1}, \delta_{i+1}) \\ & \text{iff} & (x, y) \in Gr(\gamma) \circ Gr(\mathrm{F}\gamma_i) \circ Gr(\mathrm{F}\delta_i)^\smile \circ Gr(\delta)^\smile \\ & \text{iff} & (\gamma(x), \delta(y)) \in Gr(\mathrm{F}\gamma_i) \circ Gr(\mathrm{F}\delta_i)^\smile \\ & \text{iff} & (\gamma(x), \delta(y)) \in \overline{\mathrm{F}}(Gr(\gamma_i) \circ Gr(\delta_i)) = \overline{\mathrm{F}} Z_i \qquad \text{(Fact 1.7)} \end{array}$$

We now prove (1) by induction on n. For $n = 0$ there is nothing to prove, because $Z_0 = B_0 = X \times Y$. Inductively assume that $Z_i = B_i$ for some $i \in \mathbb{N}$. We have to show that $Z_{i+1} = B_{i+1}$.

$\subseteq$ Let $(x, y) \in Z_{i+1}$, i.e. $\gamma_{i+1}(x) = \delta_{i+1}(y)$. This implies by (2) that $(\gamma(x), \delta(y)) \in \overline{\mathrm{F}} Z_i$. Therefore $\exists$ can move in the bisimilarity game $\mathcal{B}(\mathbb{X}, \mathbb{Y})$ from position (x, y) to position Z_i. By I.H. we know that all positions $(x', y') \in Z_i$ are in B_i, i.e. $\exists$ has a strategy such that for all $(x', y') \in Z_i$ she does not lose any match starting at (x', y') in less than i rounds. As a consequence by moving from (x, y) to Z_i she has a strategy in $\mathcal{B}(\mathbb{X}, \mathbb{Y})$ such that she does not lose in less than $i + 1$ rounds which means that $(x, y) \in B_{i+1}$.

$\supseteq$ Suppose $(x, y) \in B_{i+1}$, i.e. $\exists$ has a strategy in $\mathcal{B}(\mathbb{X}, \mathbb{Y})$ such that she does not lose any match starting at position (x, y) in less than $i + 1$ rounds. In particular she can move from (x, y) to some relation $R \subseteq X \times Y$ such that $(\gamma(x), \delta(y)) \in \overline{\mathrm{F}} R$ and such that $R \subseteq B_i$. Therefore, by monotonicity of $\overline{\mathrm{F}}$, $(\gamma(x), \delta(y)) \in \overline{\mathrm{F}} B_i$ and thus by I.H. $(\gamma(x), \delta(y)) \in \overline{\mathrm{F}} Z_i$. This implies by (2) that $(x, y) \in Z_{i+1}$.

QED

As a corollary we obtain the earlier announced result that the game-theoretically defined notion of n-bisimilarity forms a good finite approximation of bisimilarity.

Corollary 2.10 *Let* F *be a set functor that meets the requirements in Section 1.2. Furthermore let* $\mathbb{X} = \langle X, \gamma \rangle, \mathbb{Y} = \langle Y, \delta \rangle$ *be two* F*-coalgebras with designated points* $x \in X$ *and* $y \in Y$*. Then*

$$x \leftrightarrows_{\mathrm{F}} y \quad \textit{iff} \quad \forall n.\ x \leftrightarrows_n y.$$

Proof. It has been shown in [15] that $\gamma_n(x) = \delta_n(y)$ for all $n \in \mathbb{N}$ implies $\mathbb{X}, x \leftrightarrows_{\mathrm{F}} \mathbb{Y}, y$ under the assumption that F is ω-accessible. Therefore the claim of the proposition is an immediate consequence of 2.9. QED

3 Modal logic in a coalgebraic shape

In the first section of this paper we showed that coalgebras provide a framework for modeling various types of transition systems. The natural question to ask is whether we can also develop logical languages to reason about coalgebras. Modal languages have been successfully employed for reasoning about transition systems. Hence these languages are a good candidate for the specification of coalgebras. The various coalgebraic modal languages that have been proposed to talk about coalgebras can roughly be split into two groups.

1. Languages whose modalities are given by so-called *predicate liftings*. For an introduction to these languages we refer the reader to [5, 8, 13] and references therein.
2. Languages that are defined via the *relation lifting* of F. This approach has originally been introduced by Moss in [7].

We will now present the definition of a version of Moss' coalgebraic logic that has a finitary syntax. This logic has been introduced in [14]. Unlike *loc.cit.* we do not consider fixed point operators. Furthermore, as mentioned before, we work under the general requirement that we are given a set functor F that is ω-accessible, standard and weak pullback preserving.

Definition 3.1 The *language* of (finitary) coalgebraic logic $\mathcal{L}^{\mathrm{F}}$ is defined inductively as follows:

$$\begin{aligned} \mathcal{L}^{\mathrm{F}}_0 \ni \phi &::= \bot \mid \top \mid \phi \wedge \phi \mid \phi \vee \phi \\ \mathcal{L}^{\mathrm{F}}_{i+1} \ni \phi &::= \psi \in \mathcal{L}^{\mathrm{F}}_i \mid \phi \wedge \phi \mid \phi \vee \phi \mid \nabla \pi,\ \pi \in \mathrm{F}\mathcal{L}^{\mathrm{F}}_i \\ \mathcal{L}^{\mathrm{F}} &:= \bigcup_{i \in \mathbb{N}} \mathcal{L}^{\mathrm{F}}_i \end{aligned}$$

The *depth* of a formula $\phi \in \mathcal{L}^{\mathrm{F}}$ is defined as the smallest natural number i_ϕ such that $\phi \in \mathcal{L}^{\mathrm{F}}_{i_\phi}$. $\lhd$

Remark 3.2 From the definition it is clear that $\mathcal{L}^{\mathrm{F}}$ is a set (in contrast to Moss' original language which consisted of a proper class of formulas). The difference with Moss' original definition is that the syntax only contains finite conjunctions and, in addition to that, finite disjunctions. Furthermore Venema defines the notion of a subformula and shows in [14] that every formula of $\mathcal{L}^{\mathrm{F}}$ has a finite number of subformulas. These facts justify to say that $\mathcal{L}^{\mathrm{F}}$ has a finitary syntax.

The semantics of the logic is defined as follows.

Definition 3.3 Let $\mathbb{X} = \langle X, \gamma\rangle$ be an F-coalgebra We inductively define a relation $\models^{\mathbb{X}} \subseteq X \times \mathcal{L}^{\mathrm{F}}$ with the intended meaning that $(x, \phi) \in \models^{\mathbb{X}}$ if ϕ is satisfied at $x \in X$. In this case we also write $x \models^{\mathbb{X}} \phi$. The inductive definition of $\models^{\mathbb{X}}$ is as follows: We put $x \not\models^{\mathbb{X}} \bot$, $x \models^{\mathbb{X}} \top$ and

$$\begin{array}{lllll} x & \models^{\mathbb{X}} & \phi_1 \wedge \phi_2 & \text{if} & x \models^{\mathbb{X}} \phi_1 \text{ and } x \models^{\mathbb{X}} \phi_2, \\ x & \models^{\mathbb{X}} & \phi_1 \vee \phi_2 & \text{if} & x \models^{\mathbb{X}} \phi_1 \text{ or } x \models^{\mathbb{X}} \phi_2, \\ x & \models^{\mathbb{X}} & \nabla\pi & \text{if} & (\gamma(x), \pi) \in \overline{\mathrm{F}}(\models^{\mathbb{X}}). \end{array}$$

If the $\mathbb{X}$ is clear from the context we simply write $\models$ for the relation $\models^{\mathbb{X}} \subseteq X \times \mathcal{L}^{\mathrm{F}}$. Furthermore we write $\mathrm{Th}(x)$ for the *theory of* x, i.e. $\mathrm{Th}(x) := \{\phi \in \mathcal{L}^{\mathrm{F}} \mid x \models \phi\}$. $\lhd$

Remark 3.4 The definition of the semantics of a formula $\nabla\pi \in \mathcal{L}^{\mathrm{F}}_{i+1}$ is not circular because

$$(\gamma(x), \pi) \in \overline{\mathrm{F}}(\models) \quad \text{iff} \quad (\gamma(x), \pi) \in \overline{\mathrm{F}}\Big((\models) \restriction_{X \times \mathcal{L}^{\mathrm{F}}_i}\Big),$$

where $(\models) \restriction_{X \times \mathcal{L}^{\mathrm{F}}_i}$ denotes the restriction of $\models^{\mathbb{X}} \subseteq X \times \mathcal{L}^{\mathrm{F}}$ to $X \times \mathcal{L}^{\mathrm{F}}_i$. This can be seen by using that $\pi \in \mathrm{F}\mathcal{L}^{\mathrm{F}}_i$ and by using Fact 1.7(5). The depth of formulas in $\mathcal{L}_i$ is strictly smaller than the depth of $\nabla\pi$ and thus we can inductively assume that $(\models) \restriction_{X \times \mathcal{L}^{\mathrm{F}}_i}$ has been already defined.

Example 3.5 Let Φ be a set of propositional variables and consider the functor $\mathrm{F} = \mathcal{P}\Phi \times \mathcal{P}_{\omega -}$. Then the ∇-formulas in $\mathcal{L}^{\mathrm{F}}$ are of the form $\nabla(P, \Psi)$ where $P \subseteq \Phi$ is a set of propositional variables and Ψ is a set of formulas. Informally speaking such a formula $\nabla(P, \Psi)$ corresponds to the following formula in the usual syntax of modal logic:

$$\nabla(P, \Psi) \equiv \bigwedge_{p \in P} p \wedge \bigwedge_{p \notin P} \neg p \wedge \bigwedge_{\psi \in \Psi} \Diamond\psi \wedge \Box\Big(\bigvee_{\psi \in \Psi} \psi\Big).$$

Similarly one can translate modal formulas into formulas of $\mathcal{L}^{\mathrm{F}}$ (cf. [14, Sec. 5]).

4 Expressivity of finitary coalgebraic logics

In this section we prove that the finitary coalgebraic logic as defined in Def. 3.1 has the Hennessy-Milner property, i.e. we prove Theorem 4.1 below. Let us once more stress that despite the fact that we think that Theorem 4.1 has not been proven before, its proof can be seen as a rather straightforward adaptation of the proof of an analogue result for infinitary coalgebraic logic in [7].

Theorem 4.1 *Let* F *be an* ω*-accessible set functor. Then the language* $\mathcal{L}^{\mathrm{F}}$ *has the Hennessy-Milner property, i.e. for all pointed* F*-coalgebras* $(\langle\mathbb{X}, \gamma\rangle, x)$ *and* $(\langle\mathbb{Y}, \delta\rangle, y)$ *we have*

$$\mathrm{Th}(x) = \mathrm{Th}(y) \quad \textit{iff} \quad (\langle\mathbb{X}, \gamma\rangle, x) \leftrightarrows_{\mathrm{F}} (\langle\mathbb{Y}, \delta\rangle, y),$$

i.e. two pointed F*-coalgebras satisfy the same* $\mathcal{L}^{\mathrm{F}}$*-formulas iff they are* F*-bisimilar.*

As usual the easier part of proving the Hennessy-Milner property is to prove the invariance of the semantics of a formula under bisimilarity.

Fact 4.2 ([14]) *Let* F *be an* ω*-accessible functor and let* $(\langle\mathbb{X}, \gamma\rangle, x)$ *and* $(\langle\mathbb{Y}, \delta\rangle, y)$ *be pointed* F*-coalgebras such that* $(\langle\mathbb{X}, \gamma\rangle, x) \leftrightarrows_{\mathrm{F}} (\langle\mathbb{Y}, \delta\rangle, y)$*. Then for all formulas* $\phi \in \mathcal{L}^{\mathrm{F}}$ *we have*

$$(\langle\mathbb{X}, \gamma\rangle, x) \models \phi \quad \textit{iff} \quad (\langle\mathbb{Y}, \delta\rangle, y) \models \phi.$$

For the proof of the fact that logical equivalence implies bisimilarity we use our result from the previous section, where we saw that n-bisimilarity for all n and bisimilarity coincide if the functor is ω-accessible. The idea is to define for each $n \in \mathbb{N}$ a set of *characteristic formulas* that characterize the behavior of a given pointed F-coalgebra up-to depth n. These formulas have been introduced in [7] in order to prove the analogue of Theorem 4.1 for infinitary coalgebraic logic.

Definition 4.3 For $i \in \mathbb{N}$ we define a function $\nabla_i : \mathrm{F}^i 1 \to \mathcal{L}^{\mathrm{F}}$ by putting

$$\begin{array}{lcll} \nabla_0(*) & := & \top & \text{for } * \in 1 \\ \nabla_{i+1}(x) & := & \nabla\,(\mathrm{F}\nabla_i(x)) & \text{for } x \in \mathrm{F}^{i+1}1 \end{array} \qquad \lhd$$

The intended meaning of the ∇_i-maps is that they map a given state $x \in \mathrm{F}^i 1$, which represents some possible i-step behavior, to its characteristic formula $\nabla_i(x)$ of depth i. Let us have a look at an example.

Example 4.4 Let Φ be a set. For the functor $\mathrm{F} = \mathcal{P}\Phi \times \mathcal{P}_\omega$ the characteristic formulas look of depth ≤ 1 can be computed as follows:

$$\Psi_0 = \{\top\}, \quad \Psi_1 = \{\nabla(P, \{*\}) \mid P \in \mathcal{P}\Phi\} \cup \{\nabla(P, \emptyset) \mid P \in \Phi\}$$

where Ψ_i denotes the image of ∇_i. We can easily see that formulas in Ψ_1 precisely determine the 1-step behavior of a given pointed F-coalgebra: the formula $\nabla(P, \emptyset)$ will be true at any state in which exactly the propositional variables in P are true and which has no successors. Likewise the formula $\nabla(P, \{*\})$ expresses that precisely the p's in Φ are true and there is a successor.

The following observations about the ∇_i-maps form the justification for calling the formulas of the form $\nabla_i(\theta)$ "characterizing formulas": a coalgebra state x that has a certain n-step behavior θ satisfies precisely the formula $\nabla_n(\theta)$, i.e. no other formula of the form $\nabla_n(\theta')$ for some $\theta' \neq \theta$ is true at x.

Lemma 4.5 *Let $n \in \mathbb{N}$ and let θ be an element of $\mathrm{F}^n 1$. Furthermore let $(\langle X, \gamma\rangle, x)$. Then*

$$x \models \nabla_n \theta \qquad \textit{iff} \quad \theta = \gamma_n(x),$$

where $\gamma_n(x)$ denotes the n-step behavior of x as defined in Definition 2.7 above.

Proof. The claim of the lemma can be written in terms of relations in the following way:

$$(\models_X \circ Gr(\nabla_n)^\smile) = Gr(\gamma_n). \tag{3}$$

We define $R_n := (\models_X \circ Gr(\nabla_n)^\smile)$. Equation (3) will be proven by induction on n. The base case $n = 0$ is easy to check.

Inductively assume now that (3) is true for some $i \in \mathbb{N}$. We show that in this case our claim also holds for $n = i + 1$. In order to see this note that by definition of R_{i+1} we have $(x, \theta) \in R_{i+1}$ iff $x \models^{\mathbb{X}} \nabla_{i+1}(\theta)$. Unfolding the definition of ∇_{i+1} we obtain

$$\begin{array}{lll} (x,\theta) \in R_{i+1} & \text{iff} & x \models^{\mathbb{X}} \nabla\,(\mathrm{F}\nabla_i(\theta)) \\ & \text{iff} & (\gamma(x), \mathrm{F}\nabla_i(\theta)) \in \overline{\mathrm{F}}(\models^{\mathbb{X}}) \qquad \text{(Def. of } \models) \\ & \text{iff} & (\gamma(x), \theta) \in \overline{\mathrm{F}}(\models^{\mathbb{X}}) \circ Gr(\mathrm{F}\nabla_i)^\smile \\ & \text{iff} & (\gamma(x), \theta) \in \overline{\mathrm{F}}(\models^{\mathbb{X}} \circ Gr(\nabla_i)^\smile) \qquad \text{(Fact 1.7)} \\ & \text{iff} & (\gamma(x), \theta) \in \overline{\mathrm{F}} R_i = \overline{\mathrm{F}} Gr(\gamma_i) \qquad \text{(Def. of } R_i \text{ + I.H.)} \\ & \text{iff} & (x, \theta) \in Gr(\gamma) \circ Gr(\mathrm{F}\gamma_i) \qquad \text{(Fact 1.7)} \\ & \text{iff} & \theta = (\mathrm{F}\gamma_i \circ \gamma)(x) = \gamma_{i+1}(x) \qquad \text{(Def. of } \gamma_{i+1}) \end{array}$$

QED

We are now well prepared for finishing the proof of Theorem 4.1.

Proof of Theorem 4.1. The fact that two F-bisimilar pointed F-coalgebras are also logically equivalent was proven in Proposition 4.2. This is the implication from right to left in the theorem.

For the converse direction let $(\langle X, \gamma\rangle, x)$ and $(\langle Y, \delta\rangle, y)$ be pointed F-coalgebras. We show that $x \not\leftrightarrows y$ implies that $\mathrm{Th}(x) \neq \mathrm{Th}(y)$. By Corollary 2.10 we know that $x \not\leftrightarrows y$ entails that there is some $n \in \mathbb{N}$ such that $(\langle X, \gamma\rangle, x) \not\leftrightarrows_n (\langle Y, \delta\rangle, y)$, i.e. such that the given pointed coalgebras are not n-bisimilar. By Thm. 2.9 this implies $\gamma_n(x) \neq \delta_n(y)$. Therefore by Lemma 4.5 we have $x \models \nabla_n(\gamma_n(x))$ and $y \not\models \nabla_n(\gamma_n(x))$, i.e. $\mathrm{Th}(x) \neq \mathrm{Th}(y)$ as required. QED

5 Conclusion

We clarified the connection between the n-bisimilarity game and the final sequence of the functor. Furthermore we showed that Moss' arguments in [7] can be used to prove that finitary coalgebraic logics have the Hennessy-Milner property over coalgebras for an ω-accessible, weak pullback preserving functor.

The watchful reader will have noted that we did not make use of the game-theoretic interpretation of bisimilarity at all. We could provide a direct game-theoretic proof of Cor. 2.10 but did not do so due to lack of space. Furthermore we hope to obtain a better understanding of the bisimilarity game by further exploring the connection between the n-bisimilarity game and the final sequence that has been established in Thm. 2.9. As an example consider the functor $\mathrm{F} = \mathcal{P}_\omega(X)^A$ for some infinite set A. This functor is *not* ω-accessible but, using Worrell's results on the final sequence of F, bisimilarity can be approximated by n-bisimulations. This fact is not immediately obvious when looking only at the bisimilarity game.

References

[1] A. Baltag. A logic for coalgebraic simulation. In H. Reichel, editor, *Proceedings of the Workshop on Coalgebraic Methods in Computer Science (CMCS)*, volume 33 of *Electronic Notes in Theoretical Computer Science*, 2000.

[2] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logic, and Infinite Games*, volume 2500 of *LNCS*. Springer, 2002.

[3] C. Hermida and B. Jacobs. Structural induction and coinduction in a fibrational setting. *Information and Computation*, 145(2):107–152, 1998.

[4] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62, 1997.

[5] A. Kurz. Coalgebras and modal logic. In *Course Notes for ESSLLI'2001*, 2001. Available at `http://www.cs.le.ac.uk/people/akurz/works.html`.

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

[7] L. S. Moss. Coalgebraic logic. *Annals of Pure and Applied Logic*, 96:277–317, 1999.

[8] D. Pattinson. An introduction to the theory of coalgebras. In *North American Summer School in Logic, Language and Information (NASSLLI)*, 2003.

[9] J.J.M.M. Rutten. Automata and coinduction (an exercise in coalgebra). In D. Sangiorigi and R. de Simone, editors, *Proc. 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *LNCS*, pages 194–218. Springer, 1998.

[10] J.J.M.M. Rutten. Relators and Metric Bisimulation (Extended Abstract). *Electronic Notes in Theoretical Computer Science*, 11:1–7, 1998.

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

[12] J.J.M.M. Rutten and E.P. de Vink. Bisimulation for probabilistic transition systems: a coalgebraic approach. *Theoretical Computer Science*, 221(1–2):271–293, 1999.

[13] Y. Venema. Algebras and coalgebras. In *Handbook of Modal Logic*, volume 3, pages 331–426. Elsevier, 2006.

[14] Y. Venema. Automata and fixed point logic: a coalgebraic perspective. *Information and Computation*, 204:637–678, 2006.

[15] J. Worrell. On the final sequence of a finitary set functor. *Theoretical Computer Science*, 338(1–3):184–199, 2005.

Lekker bomen

Loredana Afanasiev, Balder ten Cate and Maarten Marx
ISLA, Universiteit van Amsterdam
The Netherlands

January 31, 2007

It has been proclaimed that *logic is the calculus of computer science* [15], but most applied computer scientists do not use logical formulas in a shape recognizable by logicians — at least not beyond propositional logic. In particular, this holds for *modal logic*. Even though theoretical computer scientists have created many different modal logics, few of them are used in practice. Until recently, it was fair to say that the *temporal logics* LTL and CTL were the best-known and most applied modal logics [8]. With the advent of XML and its associated querying and processing languages, it seems this position has been taken over by the XML document navigation language XPath.

XPath is a core fragment of the XML query language XQuery and transformation language XSLT. All modern web browsers (e.g., *Internet Explorer* and *Firefox*) are able to process XPath expressions, and most serious web programmers write XPath expressions on a regular basis (whether they are conscious of it or not). Even though the current XPath manual contains 530 pages [17], the logical core of the language is small and closely resembles modal logic. Thus, XPath is arguably the best known and most applied modal logic today.

In this paper we give a gentle introduction to the XPath language from a logician's point of view. We discuss how its connections to modal logic have led to a series of results characterizing the expressive power of this language and the complexity of various computational tasks, such as query evaluation and query containment testing.

1 Modal logic, temporal logic, and PDL

Nowadays, modal logic is mostly viewed as a language for describing graph-like structures. This 'semantic' view of modal logic stands in contrast to the more traditional 'syntactic' view, according to which modal logic is just a propositional language enriched with two non truth-functional operators $\Diamond$ and $\Box$.

As Hans Kamp recently described it[1], one of the most important steps in the historical development of modal logic was to break out of the rigid box/diamond format. Kamp himself made the first step in 1968 by introducing *until* [16]. This "*binary* modalities" makes it possible, in the context of temporal logic, to express things like "A will be true at some point in the future and, until then, B is true" ($until(A, B)$ for short). *Until* make temporal logic very expressive. In fact, on time flows isomorphic to the natural numbers, temporal logic with *until* and its mirror image *since* is as expressive as first-order logic, a result that has become known as *Kamp's Theorem* [16]. However, what Kamp described as a real watershed in modal logic was the invention of Propositional Dynamic Logic (PDL)

[1] A 15 minute presentation about the history of modal logic, on the occasion of the launch of the *Handbook of Modal Logic*, on December 18, 2006, in the Oost Indisch Huis, Amsterdam.

in the 1970s [13]. PDL is a modal logic with infinitely many modalities (or, 'programs'), that form an algebra generated by atomic programs and formula tests using the operations of *composition*, *union* and *reflexive transitive closure*, familiar from regular expressions.

Since PDL turns out to be closely related to the XML language XPath, let us explain its syntax and semantics in a bit more detail. PDL expressions are interpreted over *node labelled directed multi-graphs*, i.e., directed graphs with several relations, whose nodes are labelled.[2] There are two types of expressions in PDL: *state formulas* ($\phi, \psi, \ldots$) describing properties of points (or, 'states') in the graph, and *program formulas* ($\alpha, \beta, \ldots$) describe ways of traversing it (more formally, they define binary relations over the domain of the graph). They are defined by mutual recursion:

$$\begin{aligned} \phi &::= p \mid \neg\phi \mid \phi \wedge \psi \mid \langle\alpha\rangle\phi \\ \alpha &::= a \mid \phi? \mid \alpha/\beta \mid \alpha \cup \beta \mid \alpha^* \end{aligned}$$

where p is a node label and a is any of the relations of the multi-graph. State formulas of the form $\langle\alpha\rangle\phi$ are interpreted as "some point reachable from the current point by α satisfies ϕ", and state formulas of the form $[\alpha]\phi$, dually, are interpreted as "every point reachable from the current point by α satisfies ϕ". The atomic program formula a is interpreted as "move to a node that is a successor of the current point with respect to the relation a", and $\phi?$ is interpreted as "stay at the current node but check that ϕ is true there". The program operations $/$, $\cup$ and $*$ are interpreted as composition, union and reflexive-transitive closure.

To illustrate the expressive power of PDL, observe that, on the natural numbers with the successor relation next, $until(A, B)$ can be expressed in PDL as $\langle(\mathsf{next}/B?)^*/\mathsf{next}\rangle A$, that is, "it is possible to reach a node satisfying A by the following program: do a finite number of successor steps followed by tests on B, and end with one more successor step".

PDL is quite well understood as a logical language. For instance, we know the computational complexity of various tasks involving PDL formulas such as *model checking* and *entailment*.

Fact 1 (Complexity of PDL model checking)**.** *Given a finite node labelled directed multi-graph G with a designated state s, and given a PDL formula ϕ, it can be tested in time $\mathcal{O}(|G| \cdot |\psi|)$ whether ϕ is true at s in G. Moreover, this problem is* PTIME*-complete.*

Fact 2 (Complexity of PDL entailment)**.** *Given PDL formulas ϕ, ψ, it can be tested in time $2^{\mathcal{O}(|\phi|+|\psi|)}$ whether ϕ implies ψ (i.e., whether every state satisfying ϕ also satisfies ψ. Moreover, this problem is* EXPTIME*-complete.*

For more information, see e.g. [9, 4].

2 XML and sibling ordered trees

XML is a standard format for representation and exchange of *semi-structured data* on the internet. Semi-structured data is, roughly, "data that is not structured enough to fit nicely in tables." While in the relational database model, the fundamental datastructure is that of a table, the fundamental datastructure in XML is that of (finite) *node labelled sibling ordered trees*: node labelled finite trees in which the children of each node are linearly ordered. This is a very natural datastructure, and it appears in many settings. For instance, the parse tree of a natural language sentence, or of an algebraic term, is a finite node labelled sibling ordered tree. An example of an XML document is given in Figure 1.

[2] Usually, it is assumed that nodes can satisfy more than one label, but this is not important for us.

```
<note date='10-Nov-2006'>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget about <i>me</i>
        this weekend!</body>
</note>
```

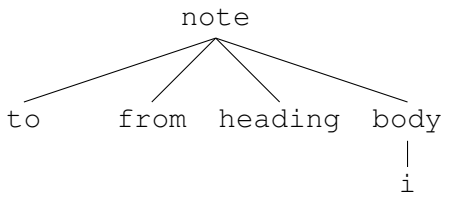

Figure 1: Example XML document with skeleton

The two most important languages for manipulating XML data are the *query language* XQuery and the *transformation language* XSLT. Here is an example of an XQuery query:

```
for $x in doc("notes.xml")//note
where $x/@date='10-Nov-2006'
return <recipient>$x/to</recipient>
```

This query could be run on a document containing a collection of notes, where each note is of the form described in Figure 1. The query then returns all recipients of notes sent on 10 november 2006, structured as XML using the `<recipient>` tag. The italic subexpressions are called *path expressions*. They select elements from the document. *XPath* is the language of these path expressions. It is a common fragment of XQuery and XSLT, and it is the topic of this paper.

In what follows, we will make one important simplifying assumption: when considering XML documents, we will abstract away from the actual data in the XML document (i.e., the text in-between the tags, and the attribute-value information). What is left is the 'skeleton' of the XML document: the hierarchical structure and the XML tag labels of the nodes. This allows us to view XML documents as nothing more than *node labelled sibling ordered trees*, where the node labels are the XML tags. In the case of Figure 1, the skeleton is precisely the picture on the right (including the information of the sibling order).

3 XPath 1.0 and its navigational core

XPath 1.0 was introduced as a language for addressing parts of an XML document. Inspired by Unix paths, its expressions describe ways to travel through an XML document. XPath 1.0 is a very rich language, in fact its technical specifications are about 30 pages long [7]. Since, in this paper, we abstract away from the data content of XML documents and consider only the 'skeleton', we can disregard much of the functionality of XPath 1.0. What is left is the *navigational core*, *Core XPath 1.0*, which was formally defined in [14]. Surprisingly, it turns out that Core XPath 1.0 is almost identical to PDL. The only real differences are the following:

- While PDL in general is interpreted over arbitrary directed multi-graphs, in the case of Core XPath 1.0 they are *finite sibling ordered trees*, i.e., XML documents. Correspondingly, there are four atomic programs, corresponding to the four basic moves in the tree: child, parent, previous-sibling and next-sibling. The node labels are simply the XML tags.
- The use of the *reflexive transitive closure* operator $*$ is restricted to atomic programs.

Besides these, there are some minor differences concerning notation and terminology. For instance, the 'program formulas' of PDL are called 'path expressions' in XPath, and 'state formulas' are called 'node tests' or 'filter expressions'.

In presenting the syntax of Core XPath 1.0 below, we will cheat a bit: we use a slightly different notation, to emphasize the connection with PDL. It should be noted, though, that there are linear translations between the two. The syntax of Core XPath 1.0 is as follows:

Path expressions (defining binary relations over the domain of the tree):

$$\begin{array}{rcl} \alpha & ::= & \mathsf{child} \mid \mathsf{parent} \mid \mathsf{previous\text{-}sibling} \mid \mathsf{next\text{-}sibling} \mid \\ & & \mathsf{child}^* \mid \mathsf{parent}^* \mid \mathsf{previous\text{-}sibling}^* \mid \mathsf{next\text{-}sibling}^* \mid \\ & & \mathsf{self} \mid \alpha/\beta \mid \alpha \cup \beta \mid \alpha[\phi] \end{array}$$

Node expressions (defining sets of nodes):

$$\phi \quad ::= \quad p \mid \neg\phi \mid \phi \wedge \psi \mid \langle\alpha\rangle \qquad \text{(for } p \text{ an XML tag)}$$

Two constructs require explanation, namely $\alpha[\phi]$ and $\langle\alpha\rangle$. The first is interpreted as a 'filter': $\alpha[\phi]$ defines the subrelation of α containing all pairs (x, y) of which y satisfies ϕ. The node expression $\langle\alpha\rangle$ simply expresses that the node under evaluation is in the domain of the relation α. Thus, for example, the path expression $\mathsf{child}[\langle\mathsf{child}\rangle]$ — in the official XPath notation `child::*[child::*]` — says "go to a child that is not a leaf". Likewise, the path expression $\mathsf{child}[\mathit{note} \wedge \neg\langle\mathsf{child}[\mathit{body}]\rangle]$ — in the official XPath notation `child::note[not(child::body)]`, abbreviated as `note[not(body)]` — says "go to a child with tag *note* that has no child with tag *body*". Incidentally, note that, while PDL supports expressions of the form $\langle\alpha\rangle\phi$, these can be expressed in Core XPath 1.0 as $\langle\alpha[\phi]\rangle$.

In practice, an XPath path expression α is used as follows: it is evaluated at a specified context node x of a document D (typically the root), and yield a *set of nodes* $\text{RESULT}_{D,x}(\alpha)$, consisting of those nodes of D that can be reached from x by executing α. The XQuery query given in Section 2 is a good example of this use of path expressions.

Complexity The close relation between Core XPath 1.0 and PDL has been exploited to obtain complexity results for various tasks involving XPath expressions. For instance, the following results can be obtained in this way:

Fact 3 (Query evaluation complexity)**.** *Given a Core XPath 1.0 path expression α, an XML document D and a context node x, RESULT$_{D,x}(\alpha)$ can be computed in linear time.*

Fact 4 (Query containment complexity)**.** *The following problem is* EXPTIME*-complete: given Core XPath 1.0 path expressions α and β, does RESULT$_{D,x}(\alpha) \subseteq$ RESULT$_{D,x}(\beta)$ hold for all D and x?*

Fact 3 follows directly from Fact 1. For Fact 4 this is not the case. It does not follow directly from Fact 2 because the class of structures is different (an implication between two formulas may hold on trees but not on arbitrary structures). Nevertheless, essentially the same (automata-theoretic) techniques apply.

Expressive power One way to measure the expressive power of a query language is by relating it to the expressive power of the logical language we know best: first-order logic. Edgar Codd proved for instance, within the framework of the relational database model, that *relational algebra* has the same expressive power as first-order logic. Since our XML documents are node labelled sibling ordered

trees, we consider a first-order language that has binary predicates $<_V$ and $<_H$ for the *descendant* and *following sibling* relations, and a unary predicate for each XML tag. Call this language FO_{tree}.

The path expressions of Core XPath 1.0 can be naturally compared to FO_{tree}-formulas with two free variables: a variable x standing for the context node and another variable y standing for the retrieved node. We say that a path expression α is equivalent to an FO_{tree}-formula $\phi(x, y)$ if for all XML documents D and nodes d, e, it is the case that $e \in \text{RESULT}_{D,d}(\alpha)$ if and only if $D \models \phi(d, e)$. Likewise, the node expressions of Core XPath 1.0 can be naturally compared to FO_{tree}-formulas with one free variable.

Fact 5. *Core XPath 1.0 is a* proper *fragment of* FO_{tree}.

The easy half of this results says that Core XPath 1.0 is a fragment of FO_{tree}. This can be showed by means of a linear translation from Core XPath 1.0 path expressions to FO_{tree}-formulas in two free variables and from Core XPath 1.0 node expressions to FO_{tree}-formulas in one free variable. The more difficult half of the result says that there are FO_{tree}-formulas that are not equivalent to any Core XPath 1.0 node or path expression. The *until* operator from Section 1 can be used here. In the case of XML documents, there are four natural versions of *until*: upward, downward, leftward and rightward. For instance, the downward version of until talks about descendants of a node: $until_{\downarrow}(A, B)$ holds at a node x iff $\exists y.(x <_V y \wedge A(y) \wedge \forall z(x <_V z \wedge z <_V y \rightarrow B(z)))$. Notice how this immediately shows that $until_{\downarrow}$ is FO_{tree}-definable. Likewise for the other versions of *until*. On the other hand, none of them can be expressed in Core XPath 1.0. A proof is given in [22], along the following lines: there is a (linear) translation from Core XPath 1.0 node expressions to first-order formulas containing only two variables. On the other hand, $until_{\downarrow}(A, B)$ and the other versions cannot be expressed with less than three variables (the latter can be shown using a variant of Ehrenfeucht-Fraïssé games [10]).

Incidentally, recall from Section 1 that *until* queries can be expressed in PDL. However, this requires the use of $*$ on non-atomic expressions (in particular, on expressions of the form $\alpha[\phi]$ with $\alpha \in \{\mathsf{child}, \mathsf{parent}, \mathsf{previous\text{-}sibling}, \mathsf{next\text{-}sibling}\}$.

A precise characterization of the expressive power of Core XPath 1.0 in terms of a two-variable fragment of FO_{tree} was given in [22], based on similar results for temporal logics without *until* [11].

4 Two extensions of Core XPath 1.0

Soon after XPath 1.0 was introduced, users found that they need more expressive power. We will discuss two directions in which one can expand XPath. One the one hand, the connection between XPath and PDL naturally suggests extending XPath with an unrestricted *transitive closure* operator. On the other hand, the W3C has introduced the 2.0 version of XPath, which extends XPath 1.0 in a different direction, namely with intersection, complementation, and quantified variables.

A summary of most of the results described in this section (and more) can be found in Figure 2. For a more detailed explanation of this diagram, cf. [26].

4.1 Regular XPath: adding the full transitive closure operator

The connection between XPath and PDL described above naturally suggests extending XPath 1.0 with a reflexive transitive closure operator that operates on arbitrary (not only atomic) path expressions. Indeed, there are various reasons for extending the language in this way [1]. The extension of Core XPath 1.0 in which $*$ can be applied to arbitrary path expressions is called *Regular XPath*.

Regular XPath is much more expressive than Core XPath. For instance, the *until* queries discussed in Section 3 can be expressed in it. As a matter of fact,

Fact 6. *Regular XPath strictly extends FO_{tree} in expressive power.*

There are two sides to this result. Firstly, it says that Regular XPath can express all FO_{tree}-definable properties of nodes and binary relation. This was proved in [21] by an adaptation of a proof of Kamp's Theorem in temporal logic (cf. Section 1). The second side of the Fact 6 is that there are node expressions and path expressions of Regular XPath that have no FO_{tree} equivalent. A simple example is the path expession $(\mathsf{child}/\mathsf{child})^*$, which says "make an even number of child steps". It is well known that such properties cannot be expressed in first-order logic, and it can be proved formally using Ehrenfeucht-Fraïssé games [10].

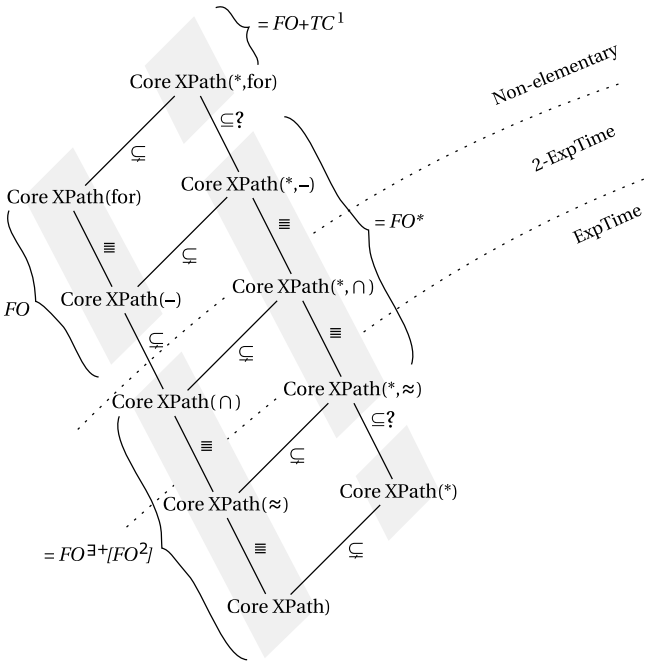

Figure 2: Hierarchy of extensions of Core XPath 1.0

Even though Regular XPath is more expresssive than FO_{tree} there are FO_{tree}-formulas for which the smallest equivalent Regular XPath expression is *much* longer. In fact, non-elementarily longer. (a function $f : \mathbb{N} \to \mathbb{N}$ is said to be non-elementary if it grows faster than any tower of expontentials of fixed height, as in $2^{(2^{(\cdots^n)})}$).

No precise characterization is known of the expressive power of Regular XPath. There is, however, a characterization of the extension of Regular XPath with $loop$. The $loop$ construct allows us to test whether a node is related to itself by a path expression. Formally, a node d satisfies $loop(\alpha)$, for α any path expression, iff (d, d) belongs to the relation α. The extension Regular XPath with $loop$ is called *Regular XPath*$^{\approx}$. The following example illustrates the convenience of having $loop$ in the language.

Example 1. Consider the following properties of nodes in a tree: "*having an even number of descendants*". This property can be expressed by a Regular XPath$^{\approx}$ node expression. To see this, first, let next be shorthand for

$$\mathsf{child}[\neg\langle\mathsf{previous\text{-}sibling}\rangle] \cup \mathsf{self}[\neg\langle\mathsf{child}\rangle]/(\mathsf{parent}[\neg\langle\mathsf{next\text{-}sibling}\rangle])^*/\mathsf{next\text{-}sibling}$$

which defines the successor relation in the depth-first left-to-right ordering of the tree. A node satisfies $loop\big((\mathsf{next}/\mathsf{next})^*[\neg\langle\mathsf{child}\rangle]/(\mathsf{parent}[\neg\langle\mathsf{next\text{-}sibling}\rangle])^*\big)$ iff it has an even number of descendants.

As a matter of fact, it is possible to express the same property in Regular XPath without the use of $loop$. However, the solution is much less straightforward. The reader may consider it an exercise to find the right expression.

The expressive power of Regular XPath$^{\approx}$ can be characterized in terms of a natural extension of FO_{tree}. Let FO^*_{tree} be the extension of FO_{tree} in which, for each formula $\phi(x, y)$ with exactly two free variables, $\phi^*(x, y)$ is also allowed as a formula, and it defines the transitive closure of the relation defined by $\phi(x, y)$.

Fact 7. *Regular XPath$^{\approx}$ has the same expressive power as FO^*_{tree}.*

Again, there is a non-elementary succinctness gap: there are queries that can be expressed exponentially more succinctly in FO^*_{tree} than in Regular XPath$^{\approx}$ [25].

Finally, let us say something about complexity. Given that Regular XPath$^{\approx}$ is much more expressive than Core XPath 1.0, it is natural to ask whether the complexity goes up as well. Surprisingly, this is not the case (up to a polynomial):

Fact 8. *Query evaluation (in the sense of Fact 3) for Regular XPath$^{\approx}$ can still be performed in* PTIME.

Fact 9. *Query containment (in the sense of Fact 4) for Regular XPath$^{\approx}$ is still* EXPTIME*-complete.*

Fact 8 follows again from known results about PDL [19] while Fact 9 is proved in [26].

4.2 Core XPath 2.0: adding intersection, complementation and quantified variables

When the W3C committee designed XPath 2.0, rather than adding the unrestricted reflexive transitive closure operator, they decided to extend XPath 1.0 in another direction. We concentrate again on the navigational core, Core XPath 2.0. It extends Core XPath 1.0 with the following new operations:

- An intersection operator on path expressions.

  This allows us to write path expressions of the form $\alpha \cap \beta$, with the semantics "select all nodes that can be reached from the current node both by α and by β".

- A complementation operator on path expressions

  This allows us to write path expressions of the form $\alpha - \beta$, with the semantics "select all nodes that can be reached from the current node by α and not by β".

- Variables and quantifiers.

  This allows us to write path expressions of the form for $i in $Path_1$ return $Path_2$, interpreted as "assign to the variable $i some node reachable from the current node by $Path_1$, and perform $Path_2$ under this variable assignment".

With these additional operators, one can express for instance *until* queries. For instance, $until_{\downarrow}(A, B)$ can be expressed as follows (with child^+ shorthand for $\mathsf{child}/\mathsf{child}^*$):

$$(\mathsf{child}^+[A]) \; - \; (\mathsf{child}^+[\neg B]/\mathsf{child}^+). \tag{1}$$

or, using quantified variables, for instance as follows:

$$\mathsf{for\ \$i\ in\ self\ return\ child}^+[\mathsf{A} \wedge \neg(\mathsf{parent}^+[\mathsf{B}]/\mathsf{parent}^+[\mathsf{self\ is\ \$i}])] \tag{2}$$

As a matter of fact, Core XPath 2.0 has full first-order expressive power [21]:

Fact 10. *Core XPath 2.0 has the same expressive power as FO_{tree}.*

This time, there is no difference in succinctness between the two languages. Indeed, there are linear translations between the two languages. This is because, basically, the quantified variables make Core XPath 2.0 a notational variant of FO_{tree}. This might seem like a good result (after all, being able to express something succinctly is a good thing). However, it also has negative consequences. In particular, Core XPath 2.0 inherits bad complexity results from FO_{tree}:

Fact 11 (Query evaluation complexity)**.** *Query evaluation (in the sense of Fact 3) for Core XPath 2.0 is* PSPACE*-complete.*

Fact 12 (Query containment complexity)**.** *Query containment (in the sense of Fact 4) for Core XPath 2.0 is non-elementary hard. Even Core XPath 1.0 extended with the complementation operator or with a single quantified variable already has a non-elementary hard query containment problem, and for the extension of Core XPath 1.0 with the intersection operator alone it is* 2EXPTIME*-complete.*

See [26] for more information. Note that Fact 11 is concerned with the *combined complexity* of the query evaluation problem, where the document and the path expression are both part of the input. Following Vardi's taxonomy [27], one could also consider the *data complexity* for Core XPath 2.0 query containment, where only the document is counted as part of the input and the path expression is assumed to be fixed. The data complexity of Core XPath 2.0 query evaluation is PTIME.

Putting things together, we see that Core XPath 2.0 is less expressive than Regular XPath$^{\approx}$, but has a higher computational complexity, due to the fact that it is more succinct.

Needless to say, the full XPath 2.0's syntax is much richer than this navigational fragment. In fact, its underlying data-model differs from that of XPath 1.0 in some respects, but that is beyond the scope of this paper. An excellent introduction is [17].

5 An experiment in formulation

Thoughout this paper, *until* queries have been our running example. We saw that they cannot be expressed in XPath 1.0, but they can be expressed in Regular XPath using the reflexive transitive closure operator, or in XPath 2.0 using either the complementation operator or quantified variables. In this section, we look how these ways of expressing queries compare when executed on real XQuery systems (recall that XPath expressions are mostly used inside XQuery queries or XSLT transformations).

We created a simple experiment in which we express the vertical until query *until(A,B)* (i.e, over the `child` relation) on XML-trees in a number of different styles. We ran these equivalent queries on a large XML document (46MB) and recorded the times. The outcomes show a large variation in processing times, varying from less than 10 to over a 1000 seconds and engine crashes.

It is difficult to draw conclusions from the experiment which can be translated into practical advice to engine developers, besides the fact that query optimization can be useful. But the results hint at a practical advice to the developers of XPath/XQuery: add the Kleene star to XPath. The Kleene style of writing the until query outperforms all others; on talks we gave the majority of the audience predicted it to be the winner of the three styles; it is a natural, procedural way of specifying the until query. *But it is very difficult to express in XQuery*, in particular when compared to the simple formulation (3) in Regular XPath.

The problem. Given a context node x, we want to compute the set of nodes that can be reached from x along vertical *until(A,B)* paths. In Section 3 and 4 we have already seen different styles of expressing this query.

The first style uses the Kleene star of Regular XPath (i.e., the reflexive transitive closure operator), thus we call it the *Kleene* style: return all nodes reachable from the context node along the path

$$(\texttt{child}[B])^*/\texttt{child}[A]. \tag{3}$$

The second style uses the path complementation operation of XPath 2.0. We call it the *Tarski* style, because it uses the operations of Tarski's relation algebra: return all nodes reachable from the

context node along the relation

$$(\texttt{descendant}[A]) \;-\; (\texttt{descendant}[\neg B]/\texttt{descendant}). \tag{4}$$

The third way of expressing the query, and probably the simplest one for a logician, is by using first-order logic and the universal quantifier: for x the context node, return all y satisfying

$$x <_V y \wedge A(y) \wedge \forall z(x <_V z \wedge z <_V y \rightarrow B(z)). \tag{5}$$

Recall that $<_V$ is the descendant relation. In honour of Frege's role in the historical development of first-order logic, we call this the *Frege* style of writing the query. There are various ways of writing this first-order formula in XPath 2.0. We will come back to this in a moment.

All three formulations describe the same binary relation.

From problem to query. Our aim is to compare the three styles of expressing until queries, by running them on a number of XQuery engines for a concrete XML document, and recording the processing times to find differences in performance.

We use an XML document from the Michigan XQuery benchmark [24] of 46MB and 728000 nodes. All elements in the document have the same label and attributes. We use one numeric attribute that has randomly generated values from 0 to 3 to express the node conditions A and B. A is the predicate condition `[@aFour=3]` and B is the predicate condition `[@aFour=0]`.

The outcome of the query should describe the range of the binary relation. We set up the query so that there are 16 "start-nodes", each with many descendants. These nodes are all the nodes at depth five in the XML tree and all queries start with the following piece of XQuery code

```
let $start := doc("mbench.xml")//*[@aLevel=5] return,
```

or

```
for $start in doc("mbench.xml")//*[@aLevel=5] return
```

The query outcomes are sequences of elements that stand in the specified relation to the start-nodes. The results are ordered in the document order.

In this way, we came up with 7 equivalent queries U1–U7, specified in the Appendix. U1 and U2 are in the Kleene style and in the Tarski style, respectively. Since there are many natural ways of expressing the Frege style in XPath 2.0 and we did not know any good criterion of picking a single one, the last 5 queries are all different variants of the Frege style.

As an example, we give the Tarski-style query, U2, copying (4):

```
let $start := doc("mbench.xml")//*[@aLevel=5]
return
$start//*[@aFour=3] except $start//*[not(.[@aFour=0])]//*
```

Note that `//*` is an XPath abbreviation for the `descendant` relation and `except` is the XPath way of writing complementation. Knowing this, the similarity with (4) is clear.

As another example, we show U4, one way of encoding the Frege style in XPath:

```
for $start in doc("mbench.xml")//*[@aLevel=5]
for $end in $start//*[@aFour=3]
return
    $end[every $inbetween in $start//*[.//*[. is $end]]
         satisfies $inbetween[@aFour=0]]
```

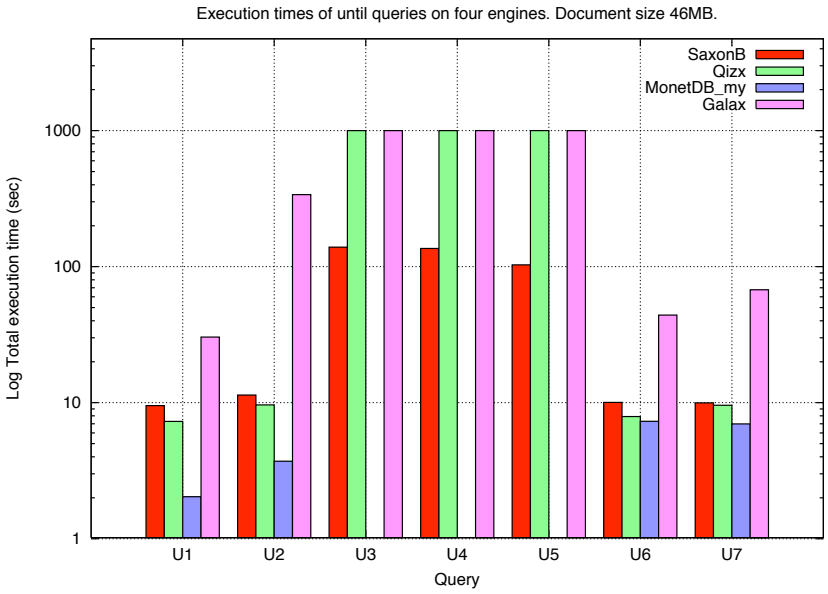

Figure 3: Processing times of 7 equivalent formulations of the until query for SaxonB, Qizx/Open, MonetDB/XQuery and Galax, on the Michigan benchmark document of 46MB, with a timeout of 1000 seconds.

This variant uses the `for-return` construct and the universal quantification `every` available in XPath 2.0.

The Kleene style query U1 cannot be expressed in XPath 2.0. Luckily, XQuery supports user-defined recursive functions, and thus we could express the query by replacing the Kleene star by a reference to a user-defined function. In this way, we could still compare it to the other styles.

Results. We ran the queries U1–U7 on the following four XQuery engines:

- SaxonB version 8.6.1 [18]
- Qizx/Open version 1.0 [3]
- MonetDB/XQuery version 0.10 [23], 32 bit compilation.
- Galax version 0.5.0 [12]

MonetDB/XQuery is an full-fledged database management system with an XML/XQuery front-end, while the other engines are stand-alone query processors.

The experiment is run with the help of XCheck[3] [2], a testing platform for XQuery engines. We used an Intel(R) Pentium(R) 4 CPU 3.00GHz, with 2026MB of RAM, running Linux version 2.6.12. For the Java applications (SaxonB and Qizx/Open) 1024MB memory size was allocated. We run each query 2 times and we output the results for the last "warmed-up" run. The times reported are CPU times measuring the complete execution of a query including loading and processing the document and outputting the result.

[3] `http://ilps.science.uva.nl/Resources/XCheck/`

The results are given in Figure 3. We remark that query execution was stopped when an engine took more than 1000 seconds to output the results. An empty engine bar indicates an engine crash.

The Kleene style query U1 performs best on all of the four engines. The Tarski variant, query U2, does reasonably well on 3 out of 4 engines, with Galax having some troubles, but at least finishing before the timeout limit. The Frege style queries U3, U4, and U5 are really problematic for all the engines. Only SaxonB finishes before the 1000 seconds limit time, Qizx/Open and Galax pass the limit, while MonetDB/XQuery crashes on all three queries. We believe the reason for this bad performance is the large amount of intermediate results generated during the execution. This intermediate results are avoided in the queries U6 and U7, which use the `ancestor` axis for navigating upwards in the tree and checking the until condition B on the paths that start with a starting-node and end on a node satisfying the until condition A.

6 In conclusion

The developments around XPath are a nice example of interaction between formal theory and applied computer science. Core XPath 1.0 turned out to be (by accident?) a modal logic, and equivalent to a well defined fragment of first order logic. One of the design goals of XPath 2.0 was to make it as expressive as first-order logic. Regular XPath is an extension of XPath in which Propositional Dynamic Logic and formalisms coming from SGML (caterpillar expressions [6]) and semi-structured data (regular path expressions [1]) nicely blend together.

Hierarchies of formalisms for describing and querying XML-trees form an active topic of research at the moment. Typical questions concern relative expressive power, succinctness, complexity, and various types of translations between different formalisms. There are still many hard nuts to crack (for instance, the question marks in Figure 2).

One potential issue of concern is that we have abstracted away too much from the XML reality by considering XML documents as node-labeled sibling ordered trees. Nodes in XML documents can have attributes with typed values (strings, integers, etc), and atomic data values, and these are neglected in the work we have discussed. One of the reasons is that languages that can speak about these data values quickly become undecidable. Some work has been done in finding well-behaved fragments of XPath with limited access to data values [5], but this is not yet of applicational value. Here input from another field of modal logic is likely to be helpful, namely the work on description logics with concrete domains [20].

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the web*. Morgan Kaufman, 2000.

[2] L. Afanasiev, M. Franceschet, M. Marx, and E. Zimuel. XCheck: a Platform for Benchmarking XQuery Engines. In *Proceedings of VLDB, Demo*, Seoul, Korea, September 2006. ACM Press.

[3] Axyana software. Qizx/open. An open-source Java implementation of XQuery. `http://www.axyana.com/qizxopen`, 2006.

[4] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

[5] M. Bojańczyk, C. David, A. Muscholl, Th. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. In *PODS*, pages 10–19, 2006.

[6] A. Brüggemann-Klein and D. Wood. Caterpillars, context, tree automata and tree pattern matching. In G. Rozenberg and W. Thomas, editors, *Proceedings of DLT '99: Foundations, Applications and Perspectives*, pages 270–285. World Scientific Publishing, Singapore, 2000.

[7] J. Clark and S. DeRose. XML Path Language (XPath). `http://www.w3.org/TR/xpath`.

[8] E.M. Clarke and B.-H. Schlingloff. Model checking. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 1367–1522. Elsevier Science Publishers, 2000.

[9] R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. *Form. Methods Syst. Des.*, 2(2):121–147, 1993.

[10] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1995.

[11] K. Etessami, M. Vardi, and Th. Wilke. First-order logic with two variables and unary temporal logic. In *Proc. LICS'97*, pages 228–235, 1997.

[12] M. Fernández, J. Siméon, C. Chen, B. Choi, V. Gapeyev, A. Marian, P. Michiels, N. Onose, D. Petkanics, C. Ré, M. Stark, G. Sur, A. Vyas, and P. Wadler. Galax. The XQuery implementation. `http://www.galaxquery.org`, 2006.

[13] M.J. Fischer and R.E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.

[14] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. In *VLDB'02*, 2002.

[15] J.Y. Halpern, R. Harper, N. Immerman, P. G. Kolaitis, M.Y. Vardi, and V. Vianu. On the unusual effectiveness of logic in computer science. *The Bulletin of Symbolic Logic*, 7(2):213–236, 2001.

[16] J.A.W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.

[17] M. Kay. *XPath 2.0 Programmer's Reference*. Wrox, 2004.

[18] M. H. Kay. SaxonB. An XSLT and XQuery processor. `http://saxon.sourceforge.net`, 2006.

[19] M. Lange. Model checking propositional dynamic logic with all extras. *Journal of Applied Logic*, 4(1):39–49, 2005.

[20] C. Lutz. *The Complexity of Reasoning with Concrete Domains*. PhD thesis, Teaching and Research Area for Theoretical Computer Science, RWTH Aachen, 2002.

[21] M. Marx. Conditional XPath. *ACM Transactions on Database Systems*, 30(4):929–959, December 2005.

[22] M. Marx and M. de Rijke. Semantic Characterizations of Navigational XPath. *ACM SIGMOD Record*, 34(2):41–46, 2005.

[23] MonetDB/XQuery. An XQuery Implementation. `http://monetdb.cwi.nl/XQuery`, 2006.

[24] K. Runapongsa, J. Patel, H.V. Jagadish, Y. Chen, and S. Al-Khalifa. The Michigan Benchmark: A Microbenchmark for XML Query Processing Systems. In *Proceedings of EEXTT*, pages 160–161, 2002.

[25] B. ten Cate. The expressivity of XPath with transitive closure. In *Proceedings of PODS'06*, pages 328–337, 2006.

[26] B. ten Cate and C. Lutz. The complexity of query containment in expressive fragments of XPath 2.0. Under submission. Available from `http://staff.science.uva.nl/~bcate`.

[27] M.Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of STOC '82*, pages 137–146, New York, NY, USA, 1982. ACM Press.

A Until queries

The following equivalent queries were used in the experiment reported in Figure 3. The `id` attribute of the `query` element gives the query name used in the histogram; the description element gives a short query description; and the syntax element contains the actual query in XQuery.

```
<queries>

<query id="U1">
<description>Use recursive function</description>
<syntax engine="all"><![CDATA[

  declare namespace my='my-functions';
  declare function my:until($input as node()*)
  {
    for $i in $input return
    ($i/child::*[@aFour=3],my:until($i/child::*[@aFour=0]))
  };
  let $start := doc("mbench.xml")//*[@aLevel=5]
  return
  my:until($start)

]]></syntax>
</query>
<query id="U2">
<description>Use except</description>
<syntax engine="all"><![CDATA[

  let $start := doc("mbench.xml")//*[@aLevel=5]
  return
  $start//*[@aFour=3] except $start//*[not(.[@aFour=0])]//*

]]></syntax>
</query>
<query id="U3">
<description>Forward every with if </description>
<syntax engine="all"><![CDATA[
```

```
  for $start in doc("mbench.xml")//*[@aLevel=5],
      $end in $start//*[@aFour=3]
  return
  if (every $inbetween in  $start//*[.//*[. is $end]] satisfies $inbetween/@aFour=0)
  then $end
  else ()

]]></syntax>
</query>
<query id="U4">
<description>Forward as in the usual FO writing of until but now with a predicate
</description>
<syntax engine="all"><![CDATA[

  for $start in doc("mbench.xml")//*[@aLevel=5],
      $end in $start//*[@aFour=3]
  return
    $end[every $inbetween in $start//*[.//*[. is $end]] satisfies
    $inbetween/@aFour=0]

]]></syntax>
</query>
<query id="U5">
<description>Forward empty with if </description>
<syntax engine="all"><![CDATA[

  for $start in doc("mbench.xml")//*[@aLevel=5],
      $end in $start//*[@aFour=3]
  return
  if (empty($start//*[not(.[@aFour=0])]//*[. is $end]))
  then $end
  else ()

]]></syntax>
</query>
<query id="U6">
<description>Backward with every</description>
<syntax engine="all"><![CDATA[

  for $start in doc("mbench.xml")//*[@aLevel=5]
  return
  $start//*[@aFour=3][every $inbetween in ancestor::*[ancestor::*[. is $start]]
  satisfies $inbetween/@aFour=0]

]]></syntax>
</query>
<query id="U7">
<description>Backward with predicate</description>
<syntax engine="all"><![CDATA[

  for $start in doc("mbench.xml")//*[@aLevel=5]
  return
```

```
  $start//*[@aFour=3][not(ancestor::*[not(.[@aFour=0])]/ancestor::*[. is $start])]

]]></syntax>
</query>

</queries>
```

Reasoning about recursive processes in shared-variable concurrency

F.S. de Boer

CWI, Amsterdam, Netherlands
F.S.de.Boer@cwi.nl

1 Introduction

The main contribution of this paper is a sound and complete proof method for concurrent systems that consist of recursive first-order processes.These processes can be created dynamically and interact via shared variables.

Our proof method consists of annotating each recursive process with *assertions* which express certain properties of the shared variables and its own local variables. Such an annotated process is *locally* correct if certain verification conditions hold which characterize its sequential (and deterministic) flow of control. On the other hand, reasoning about the interaction between processes involves a global *interference freedom test*. This test is modeled after the corresponding test in [6] for concurrent systems consisting of a statically fixed number of processes which interact via shared variables.

The main contribution of this paper is the generalization of the basic method described in [6] to recursive processes and dynamic process creation. This paper also provides a formal justification of this generalization in terms of soundness and completeness proofs.

2 Recursive processes with shared variables

A program consists of a set of (parameterized) first-order process definitions

$$p(u_1, \ldots, u_n) = S$$

Here p denotes the name of the process, $u_1, \ldots, u_n$ are its (formal) parameters, and S denotes its body. This paper abstracts from the concrete syntax of S which describes a sequential and deterministic behavior. Basic statements include the usual assignments and (recursive) process calls

$$[\text{create}]\ p(e_1, \ldots, e_n)$$

Here $e_1, \ldots, e_n$ is the list of actual parameters which are used to instantiate the formal parameters of p. The keyword 'create' is optional. It indicates the dynamic creation of the process $p(e_1, \ldots, e_n)$. The process executing this create statement continues its own execution, i.e., it does not wait for the process $p(e_1, \ldots, e_n)$ to terminate. On the other hand, the execution of a call $p(e_1, \ldots, e_n)$ itself consists of adding it to the call stack of the executing process. Operationally, a process is modeled as a stack of closures. A closure is a statement together with its local context specifying the values of the local variables. The process itself is executing the closure on top of the call stack, which is also called its active closure. All other closures represent pending calls.

A 'snapshot' Π of a concurrent execution consists of a set of processes and a global assignment of the shared variables. A global transition relation

$$\Pi \rightarrow \Pi'$$

describes that the execution of an atomic statement by one process in Π results in the global state Π'. The execution of a call $p(e_1, \ldots, e_n)$ generates an active closure which consists of the body of p and the local context obtained by assigning the values of the actual parameters to the formal parameters of p. This active closure is added as a new process in case of the execution of a corresponding create statement, otherwise it is pushed unto the call stack of the executing process.

Assertions are used to annotate the interleaving points of statements. An assertion P is evaluated in a configuration. A configuration γ consists of a global assignment of the shared variables and a local context. By

$$\gamma \models P$$

is denoted that the configuration γ satisfies the assertion P. An assertion is valid, denoted by $\models P$, if $\gamma \models P$, for every configuration γ.

For a statement S,

$$WP(S, P)$$

denotes the weakest precondition which guarantees that every terminating execution of S satisfies P. Formally, this weakest precondition is semantically defined in terms of a structural operational semantics for transitions

$$\langle S, \gamma \rangle \rightarrow \gamma'$$

where γ denotes an initial configuration and γ' denotes the resulting configuration of the execution of S. In this local semantics a create statement is simply modeled by a 'skip' statement. Given such an operational semantics, the following definition is standard.

$\gamma \models WP(S, P)$ if and only if $\gamma' \models P$, for $\langle S, \gamma \rangle \rightarrow \gamma'$

Example: parallel prime sieve This section concludes with a parallel program for generating primes. This program includes a shared variable 'nat' for generating the natural numbers. The process 'sieve(prime)' checks whether 'nat' is divisible by its formal parameter 'prime', which will denote a prime number. Synchronization between sieve processes is managed by the shared variable 'found' which stores the set of found prime numbers and the shared variable 'passed' which stores those numbers for which 'nat' has passed the test successfully.

```
sieve(prime)=
old:= nat;
if prime | old
then synchronized
      begin
      if old=nat
      then nat:=nat+1;
            passed:=∅
      fi
      end
else  synchronized
      begin
      if old=nat
      then passed:= passed∪{prime} fi;
      if passed=found
      then found:=found∪{nat};
            passed:=∅
            create sieve(nat);
            nat:=nat+1
      fi;
      end
fi;
sieve(prime)
```

The local variable 'old' is used to 'freeze' the value of 'nat'. The keyword 'synchronized' indicates a statement which cannot be interleaved by any other synchronized process. If 'old' does not pass the test, 'nat' can be incremented, *only* if 'old' still holds the current value of 'nat'. A corresponding new round is initialized by setting the set 'passed' to the empty set. If 'old' does pass the test, this is recorded by adding 'prime' to the set 'passed' only if 'old' still holds the current value of 'nat'. Subsequently, it is tested whether the two sets 'passed' and 'found' are equal, where 'found' stores the found prime numbers. If so, we got another prime and a corresponding sieve process is created. Additionally, the search for a new prime is initiated by incrementing 'nat' , updating 'found' and resetting 'passed'.

The execution of the program starts with the initialization

nat:=3; found:=∅; passed:=∅

and the creation of the initial sieve process

sieve(2)

Clearly, the shared variables 'found' and 'passed' can be replaced by corresponding counters. However, the additional information allows to express correctness by the simple global invariant

found⊆Primes

which states that 'found' only stores prime numbers ('Primes' denotes the set of prime numbers). In order to establish this invariant the following additional information is needed:

Max(found)≤nat≤Nextpr(found)

which states that 'nat' is in between the greatest prime number found and the next prime number. Furthermore the following information about the sieve process itself is needed:

passed$\subseteq \{n : n \not| \text{nat}\} \cap$ found

Using the proof method described in the next section it is a straightforward exercise to establish the global invariant and the local invariant

prime$\in$found

3 Proof-outlines

A proof-outline is a correctly annotated program. An annotation of a program associates with every sub-statement S (appearing in a process body) a precondition $Pre(S)$ and a postcondition $Post(S)$. Validation of verification conditions establish the correctness of an annotated program. First the verification condition which establishes that assertions are interference free is introduced. Then the verification conditions which establish that assertions specify correctly the sequential control flow are discussed.

Interference freedom test

In order to characterize the interference between different processes it is assumed that each process has a distinguished local variable 'id' which is used for its identification. How unique process id's are generated are described in the next section.

An assertion P is defined to be invariant over the execution of a statement S by a *different* process if the following verification condition holds:

$$\models (P \wedge Pre(S) \wedge \text{id} \neq \text{id}') \rightarrow WP(S, P)$$

For notational convenience, it is implicitly assumed that the local variables of P and $Pre(S)$ are named apart by 'priming' the local variables of P. Note that this includes the distinguished local variable 'id', which is thus renamed in P by 'id$'$'.

The above verification condition models the situation that the execution of the process denoted by the fresh local variable 'id$'$' is interleaved by the execution of the statement S by the process denoted by the distinguished local variable 'id'. That we are dealing with two *different* precesses is simply described by the disequality id $\neq$ id$'$.

Example 1. In example 3 a shared variable 'lock' is introduced to reason about synchronized statements. This variable stores the identity of the process that owns the lock. As in Java the execution of synchronized statement cannot be interleaved by the execution of a synchronized statement by *another* process. That is, every synchronized statement is characterized by the invariant

id=lock

With this additional information annotations of the synchronized statements are trivially free from interference: For any assertions P and $Pre(S)$ such that P implies id$'$ = lock and $Pre(S)$ implies id = lock, respectively, it is the case that

$$\models (P \wedge Pre(S) \wedge \text{id} \neq \text{id}') \rightarrow WP(S, P)$$

because the antecedent is inconsistent.

Local correctness

An annotated program is locally correct if the verification conditions hold which characterize the sequential flow of control within one process. We have the standard verification conditions which characterize control structures like sequential composition, choice, and iteration constructs.

Process calls The discussion is restricted to process calls

$$p(e_1, \ldots, e_n)$$

where the actual parameters $e_1, \ldots, e_n$ are not affected by the call itself. Furthermore, it is assumed that the formal parameters of the process p are read-only. Given such a call parameter passing is simply modeled by the sequence of assignments

$$u_1 := e_1; \ldots, u_n := e_n$$

Below this sequence of assignments is abbreviated by $\bar{u} := \bar{e}$.

Let S denote the body of the process p. The following verification condition validates the precondition P of the call $p(e_1, \ldots, e_n)$:

$$\models P \rightarrow WP(\bar{u} := \bar{e}, Pre(S))$$

Here it is assumed that the local variables of the precondition $Pre(S)$ of the process body only include the formal parameters of the process p and the distinguished local variable 'id'. Note that the local variable 'id' thus may occur both in the precondition P of the call and the precondition $Pre(S)$ of process body. We do not need to distinguish these different occurrences because the local variable 'id' in both preconditions denotes the *same* process executing the call.

Example 2. Consider the precondition

lock=id∧balance≥amount

of a call

withdraw(amount)

of a synchronized process 'withdraw'. This precondition states that the executing process denoted by 'id' owns the lock (and that the value of the shared variable 'balance' is greater or equal to that of the local variable 'amount'). The above precondition implies

$$WP(\text{u} := \text{amount}, \text{lock} = \text{id} \wedge \text{balance} \geq \text{u})$$

Note that the process therefore still owns the lock when executing the body of 'withdraw'.

Furthermore, the following verification condition validates the postcondition Q of a call $p(e_1, \ldots, e_n)$:

$$\models WP(\bar{u} := \bar{e}, Post(S)) \rightarrow Q$$

As above, it is assumed that the local variables of the postcondition $Post(S)$ only include the formal parameters of the process p and the distinguished local variable 'id'. Note that since the formal parameters are read-only and the actual parameters are not affected by the call itself, we can restore the values of the formal parameters by the assignment $\bar{u} := \bar{e}$.

Example 3. Consider the postcondition

$$\text{lock} = \text{id} \wedge \text{balance} = \text{old(balance)} - \text{u}$$

of the body of the synchronized process 'withdraw'. The expression 'old(balance)' denotes the 'old' value of the (shared) variable, i.e., its value at 'call time'. We have that

$$WP(\text{u} := \text{amount}, \text{lock} = \text{id} \wedge \text{balance} = \text{old(balance)} - \text{u})$$

reduces to the postcondition

$$\text{lock} = \text{id} \wedge \text{balance} = \text{old(balance)} - \text{amount}$$

of the corresponding call. This postcondition thus implies that the executing process keeps the lock.

Auxiliary variables

In general to prove the correctness of a program we need auxiliary variables which are used to describe certain properties of the flow of control.

*Example 4 (***Mutual exclusion).** Consider the process

cs()= P(sem);S;V(sem)

where sem is a global shared binary semaphore (initialized to 0). In order to prove that no two processes are executing the critical section S, a shared variable 'in' is introduced which stores the *set* of processes that are in their critical section (it is initialized to the empty set). The process is extended as follows:

cs()= [P(sem);in:=in∪{id}];S;[V(sem);in:=in\{id}]

The brackets are used to indicate statements which are assumed to be executed atomically, that is without interleaving. Note that without loss of generality we can indeed assume that between acquiring (or releasing) the semaphore and the corresponding update of the auxiliary variable no other processs are interleaved.

Mutual exclusion then can be expressed by the assertion Mutex defined by

$$|\text{in}| = \text{sem} \wedge 0 \leq \text{sem} \leq 1$$

This assertion is introduced as an invariant of the extended process which annotates all its interleaving points, that is, the start and end of its extended body, and the start and end of the critical section S.

For the proof of local correctness of the annotation the following (standard) characterization of the weakest precondition of a postcondition Q (of the operations for acquiring and releasing the semaphore) is used:

$$WP(\text{P(sem)}, Q) = (\text{sem} = 0 \rightarrow Q[1/\text{sem}])$$

and

$$WP(\text{V(sem)}, Q) = (\text{sem} = 1 \rightarrow Q[0/\text{sem}])$$

Local correctness of the invariant Mutex then is expressed by the verification conditions

$$\models \text{Mutex} \rightarrow WP(\text{P(sem)}; \text{in} := \text{in} \cup \{\text{id}\}, \text{Mutex})$$

and

$$\models \text{Mutex} \rightarrow WP(\text{V(sem)}; \text{in} := \text{in} \setminus \{\text{id}\}, \text{Mutex})$$

Note that these local verification conditions which establish Mutex as an invariant of the extended process, also (trivially) imply the verification conditions for the interference freedom test. In other words, using a local invariant like Mutex makes the interference freedom test *redundant*.

Auxiliary variables are also used to describe the semantics of creating a process and high-level synchronization constructs.

Creating a process In order to describe the specific semantics of process creation a shared variable 'c' is introduced which counts the number of created processes. A create statement

$$\text{create } p(e_1, \ldots, e_n)$$

is transformed by

$$\text{c} := \text{c} + 1; \text{create } p(e_1, \ldots, e_n, \text{c})$$

where the additional actual parameter corresponds with the distinguished local variable 'id' exported as a formal parameter of the process p.

Given a precondition P of above create statement, the precondition $Pre(S)$ of the body S of the process p is validated by the verification condition

$$\models P \rightarrow WP(\bar{u} := \bar{e}, Pre(S))$$

where $\bar{u} := \bar{e}$ contains the assignment 'id := c'. Note that in order to avoid name clashes we have to rename the distinghuised local variable id in P because now it denotes a different process.

On the other hand, the postcondition Q of a create statement is simply validated by the verification condition

$$\models P \rightarrow Q$$

where P denotes its precondition.

Synchronized statements In order to describe the specific semantics of a synchronized statement

$$\text{synchronized } S$$

an auxiliary shared variable 'lock' is introduced which stores the identity of the process owning the lock. Since a process releases the lock only when it has finished executing its synchronized statements, we also need an auxiliary shared variable that counts the number of active synchronized statements in a process. Every synchronized statement S is then prefixed with an *await* statement

await lock=id∨ lock=0 do
count:=count+1;lock:=id
od

The boolean condition states that either the process already owns the lock or the lock is not yet initialized (i.e., is 'free'), assuming that zero is not used as a process id's.

On the other hand, every synchronized statement ends with the execution of the await statement

await true do
count:=count-1; if count=0 then lock:=0 if
od

The notion of proof outlines is extended with the following standard verification condition for await statements

$$\models (P \wedge b) \rightarrow WP(S, Q)$$

where P and Q denote the precondition and the postcondition of the await statement, b denotes its boolean condition and S denotes its main body.

Since the evaluation of the boolean guard of an *await*-statement and the execution of its body are assumed to be atomic we only need to apply the interference freedom test to the pre- and postcondition of the *await*-statement itself.

*Example 5 (***Wait and notify).** Like in Java, a process which owns the lock can release it by executing the 'wait' statement. It has to wait until another process owning the lock calls the 'notify' statement. In order to describe the semantics of this mechanism an auxiliary shared variable 'wait' is introduced which is used to store the set of processes waiting for its lock. The semantics of the wait statement then is described by the following code

if lock=id
then lock:=0;wait:=wait∪{id}
else abort
fi;
await lock=0 ∧ id∉ wait
do lock:=id od

This statement first checks whether the process owns the lock. If so, the process simply releases the lock and is added to the set of waiting processes. If the process does not own the lock execution is aborted. The subsequent await statement waits for the lock to be free and for the process to be removed from the set of waiting processs.

The semantics of the notify statement, which involves an *arbitrary* choice of the process to be notified, is described by

if lock=id
then wait:=wait \{any(wait)}
else abort
fi

Here 'any' is a set operation that satisfies the axiom

any(wait)∈wait

Logically it is a 'skolem' function.

In general, auxiliary variables can be introduced as local variables and as shared variables. Assignments to auxiliary variables may not affect the flow of control of the given program. It is important to note that auxiliary variables are also allowed as additional formal parameters of process definitions. Such auxiliary variables can be used to reason about invariance properties of process calls.

*Example 6 (***Faculty function).** Consider for example the following recursive process for computing the faculty function.

fac()=
if x>0
then x:=x-1;fac();x:=x+1;y=y*x
else y:=1
fi

Here 'x' and 'y' are shared variables. Upon termination 'y' stores the faculty of the value stored by 'x'. In order to prove that the value of 'x' upon termination equals its old value, a formal parameter 'u' is introduced and the process is extended by

fac(u)=
if x>0
then x:=x-1;this.fac(u-1);x:=x+1;y=y*x
else y:=1
fi

We then can express the above invariance property by introducing the assertion 'u=x' both as precondition and the postcondition of the process body. This specification of the process body can be validated by introducing 'u=x+1' as the precondition and the postcondition of the recursive call. We have the following trivial verification conditions for process invocation and return

$\models$u=x+1$\rightarrow$u-1=x and $\models$u-1=x$\rightarrow$u=x+1

where the assertion 'u-1=x' results from replacing the formal parameter 'u' in 'u=x' by the actual parameter 'u-1'.

It is of interest to note that the use of auxiliary variables as additional formal parameters in reasoning about invariance properties of recursive calls, differs from the standard Hoare logic of recursive procedures which requires certain *extra* axioms and rules (see [1]).

4 Soundness and completeness

In this section soundness and completeness proofs are sketched. These proofs are based on a formal semantics sketched in the introduction. This structural operational semantics defines a transition relation on global states. A global state Π of a program consists of a set of processes, a process being represented as a stack of closures, and a global assignment of the shared variables. The details of the definition of the global transition relation

$$\Pi \rightarrow \Pi'$$

which represents the execution of an atomic statement by one process in Π resulting in the global state Π', are straightforward and omitted.

For notational convenience only, it is assumed throughout this section that every interleaving point of the given program is uniquely labeled. Such labels are denoted by $l, l', \ldots$. By

$$l : S : l'$$

a statement S is denoted with its start and end labeled by l and l', or the label l itself (': $S : l'$' thus being optional). A label on its own marks the termination of a process body. The assertion annotating an interleaving point l is denoted by $@l$.

Soundness

Let π be an annotated program. A global state Π satisfies an annotated program π, denoted by

$$\Pi \models \pi$$

if for every process in the global state Π with active closure $(l : S : l', \tau)$, we have

$$\gamma \models @l$$

where γ denotes the configuration consisting of the global assignment of Π and the local context τ. Roughly, a global state satisfies an annotated program if every process satisfies the assertion annotating the statement of its active closure.

Theorem 1 (Soundness). *For any correctly annotated program π (possibly extended with auxiliary variables),*

$\Pi \models \pi$ *and* $\Pi \rightarrow \Pi'$ *implies* $\Pi' \models \pi$

Roughly, this theorem states the invariance of the assertions of a correctly annotated program. The proof involves a straightforward but tedious case analysis of the computation step.

Completeness

Conversely, completeness can be established by proving correctness of an extended program annotated with so-called reachability predicates. These predicates are introduced in [2] and [5], and adapted to recursive processes with shared variables as follows. Given a program, for every interleaving point l the predicate $@l$ is defined by

> $\gamma \models @l$ if there exists a reachable global state Π that realizes the global assignment of γ and that contains a process with an active closure $(l : S : l', \tau)$, where τ is the local context of γ.

A global state Π is reachable if there exists a partial computation

$$\Pi_0 \rightarrow^* \Pi$$

starting from a fixed initial global state Π_0. Here $\rightarrow^*$ denotes the reflexive, transitive closure of $\rightarrow$.

Using encoding techniques (as for example described in [7]) it can be shown that the above reachability predicates can be expressed in the assertion language.

A straightforward though tedious case analysis establishes that a program annotated with the above reachability predicates is locally correct. The main case of interest is a proof of the verification condition

$$\models WP(\bar{u} := \bar{e}, @l) \rightarrow @l'$$

for validating the postcondition of a (recursive) process call $p(e_1, \ldots, e_n)$: the label l marks the end of the body of process p and l' the termination of the call. Parameter passing is modeled by the sequence of assignments abbreviated by $\bar{u} := \bar{e}$. In order to validate this verification condition every process definition is extended with an additional formal parameter which stores the local context of the call and an additional parameter for passing the label identifying the call. The local context of the call is stored in an array. The additional formal parameters are denoted by 'con' and 'lab'. In order to initialize the local context of the body of a process (to be passed in subsequent calls), the following initialization is added:

$$\text{mycontext} := \{u_1, \ldots, u_n, \text{con}, \text{lab}\}$$

The local variable 'mycontext' of the body of a process thus stores (a pointer to) the array of values of its formal parameters including the additional parameters 'con' and 'lab' which store the local context and label of the corresponding call. A call $p(e_1, \ldots, e_n)$ with label l' is then extended by

$$p(e_1, \ldots, e_n, \text{mycontext}, l')$$

The additional parameters ensure that the predicate $WP(\bar{u} = \bar{e}, @l)$ indeed describes the return of the process p to the given call. To see this, let

$$\gamma \models WP(\bar{u} = \bar{e}, @l)$$

where $\bar{u} := \bar{e}$ now includes the additional assignments 'con:=mycontext' and 'lab:=l'''. It follows that

$$\gamma' \models @l$$

where γ' results from the execution of the statement $\bar{u} = \bar{e}$ in γ. By the above definition of the reachability predicates it follows that there exists a partial computation

$$\Pi_0 \rightarrow^* \Pi'$$

that realizes the global assignment of γ' (which equals that of γ) and that contains an active closure (l, τ') which indicates the termination of the body of process p (τ' denotes the local context of γ'). Since

$$\tau'(\text{con}) = \tau(\text{mycontext}) \text{ and } \tau'(\text{lab}) = l'$$

we know that Π' contains the process

$$\cdots (l' : S : l'', \tau)(l, \tau')$$

where S denotes the continuation of the given call (whose termination is marked by l'). Let

$$\Pi' \rightarrow \Pi$$

be the computation step which pops the closure (l, τ') from the above call stack. Since Π also realizes the global assignment of γ it is the case that

$$\gamma \models @l'$$

Remains to show that reachability predicates are interference free. More specifically, we have to show that for any interleaving points l and l', with l marking the start of an atomic statement S, we have

$$\models (@l' \wedge @l \wedge \mathrm{id} \neq \mathrm{id}') \rightarrow WP(S, @l')$$

Roughly, this verification condition states that if one process reaches l' and another process reaches l, then l' is still reachable after the execution of the statement S. This follows trivially if there exists one computation where both processes reach l' and l at the same time. However, in general this is not the case, e.g., the reachability of l' may require a *scheduling* of the processes which is incompatible with the reachability of l.

*Example 7 (***Scheduling).** Consider the following process

race()=
synchronized
begin u:=b;
    if b then b:=false fi
end;
if u then $l_1 : S_1$ else $l_2 : S_2$ fi

Here 'u' is a local variable and 'b' is a shared variable. Let the main process of the program initialize 'b' to 'true' and then simply create two instances of the above process. Let γ be a configuration such $\gamma(\mathrm{b}) = \mathrm{false}$, $\gamma(\mathrm{u}) = \gamma(\mathrm{u}') = \mathrm{true}$, and $\gamma(\mathrm{id})$ and $\gamma(\mathrm{id}')$ are two different instances of the process p (the local variables 'u' and 'id' are renamed in the predicate $@l'_1$ by the fresh local variables u′ and id′). It is easy to see that $\gamma \models @l'_1 \wedge @l_1$. But clearly there exists no reachable global state in which *both* processes are at l_1 at the same time.

Therefore a global shared auxiliary variable 'sched' is introduced which records the scheduling of the processes. Every read or write operation which involves access to the global assignment of the shared variables is extended with an update which adds the identity of the executing process.

Example 8. Returning to the above example, note that this additional scheduling information implies that

$$\models (@l'_1 \wedge @l_1 \wedge \mathrm{id} \neq \mathrm{id}') \rightarrow \mathrm{false}$$

Note that $@l'_1$ implies that 'sched' stores the process denoted by 'id′' first, whereas $@l_1$ implies that 'sched' stores the process denoted by the distinguished local variable 'id' first.

Interleaving of the local computations of the processes, i.e., computations which only access the local context of the active closures and which do not access the global assignment of shared variables , does not affect the global computation. More specifically, the variable 'sched' enforces the following confluence property of the global transition relation.

Lemma 1 (Confluence). *Let π be a program extended with the auxiliary variable 'sched' for recording the scheduling of processes, as described above. Furthermore, let the global states Π and Π' assign the same value to the variable 'sched'. It follows that if*

$$\Pi_0 \rightarrow^* \Pi \text{ and } \Pi_0 \rightarrow^* \Pi'$$

then there exists a global state Π'' such that

$$\Pi \rightarrow^* \Pi'' \text{ and } \Pi' \rightarrow^* \Pi''$$

Furthermore, these partial computations only consist of local computations steps which do not involve (read or write) access to the global assignment of the shared variables.

This lemma holds because processes are assumed to be deterministic, i.e., the only source of nondeterminism is interleaving. The following theorem states that the reachability predicates are interference free.

Theorem 2. *For any labeled statements $l : S$ and $l' : S'$ of a program extended with the auxiliary variable 'sched' we have*

$$\models (@l' \wedge @l \wedge \mathrm{id} \neq \mathrm{id}') \rightarrow WP(S, @l')$$

Proof. Let

$$\gamma \models @l' \wedge @l \wedge \mathrm{id} \neq \mathrm{id}'$$

By definition of the reachability predicates $@l'$ and $@l$ there exist partial computations

$$\Pi_0 \rightarrow^* \Pi \text{ and } \Pi_0 \rightarrow^* \Pi'$$

starting from a fixed initial global state Π_0, such that $(l : S, \tau)$ is the active closure of the process denoted by $\tau(\mathrm{id})$ in Π, whereas $(l' : S', \tau')$ is the active closure of $\tau(\mathrm{id}')$ in Π'. Here τ denotes the local context of the configuration γ and $\tau'(u) = \tau(u')$, for every local variable u (remember that primed local variables are introduced in order to avoid name clashes between the local variables of $@l$ and $@l'$). Furthermore, the global assignment of γ is realized in both the global states Π and Π'. The auxiliary variable 'sched' thus has the same value in the global states Π and Π'. By the above lemma, there exists a global state Π'' which can be reached from both Π and Π' by local computations only. Since we can backtrack the local computation steps of the two processes (denoted by $\tau(\mathrm{id})$ and $\tau(\mathrm{id}')$), we may assume without loss of generality that in the global state Π'' the process denoted by $\tau(\mathrm{id})$ is about to execute S and $\tau(\mathrm{id}')$ is about to execute the statement S'. Let $\langle S, \gamma\rangle \rightarrow \gamma'$ be the terminating execution of S (by $\tau(\mathrm{id})$). It immediately follows that there exists a global state reachable from Π'' that realizes the global assignment of the configuration γ' and in which $\tau(\mathrm{id}')$ is still about to execute S' (in the local context τ' introduced above). By definition of the reachability predicates and the renamed local variables of $@l'$ we thus have

$$\gamma' \models @l'$$

(note that the values of the 'fresh' local variables of $@l'$ in the local context of γ' are the same as in the local context τ of the initial configuration γ). So we conclude that $\gamma \models WP(S, @l')$.

5 Conclusion

In this paper a sound and complete proof method is presented for recursive processes with shared variables. The proof method distinguishes a local level which is based on a Hoare logic for the sequential flow of control of recursive calls within one process and a global level which deals with interference between processes.

The proof method incorporates the use of auxiliary variables. These variables are used to capture specific aspects of the flow of control. In general, auxiliary variables can be used to extend the proof method in a systematic manner to synchronization mechanisms and further details of the underlying memory model.

Of particular interest is the use of auxiliary variables introduced in this paper as additional formal parameters to reason about invariance properties of recursive calls within one process. In the completeness proof such additional formal parameters are used to pass the stack of local contexts. This use allows a complete characterization of recursive process calls in terms of reachability predicates, as required by the concurrent context. How to incorporate the extra rules for reasoning about invariance properties of recursive calls in a sequential context (see [4] and [1]) is an interesting topic of future research.

References

1. K.R. Apt: Ten years of Hoare logic: a survey — part I. ACM Transactions on Programming Languages and Systems, Vol. 3, No. 4, October 1981, pp. 431–483.
2. K.R. Apt. Formal justification of a proof system for Communicating Sequential Processes. Journal of the ACM, Vol. 30, No. 1, January 1983, pp. 197–216.
3. K.R. Apt, N. Francez and W. P. de Roever. A proof system for Communicating Sequential Processes. ACM Transactions on Programming Languages and Systems, 2:359-385, 1980.
4. G. A. Gorelick . A complete axiomatic system for proving assertions about recursive and non-recursive programs. Technical Report 75, Department of Computer Science, University of Toronto, 1975.
5. S. Owicki. A consistent and complete deductive system for the verification of parallel programs. Proceedings of the eighth annual ACM symposium on Theory of computing. ACM Press, 1976.
6. S. Owicki and D. Gries. An axiomatic proof technique for parallel programs. Acta Informatika, 6:319-340, 1976.
7. J.V. Tucker and J.I. Zucker: Program Correctness over Abstract Data Types, with Error-State Semantics. CWI Monograph Series, Vol. 6, Centre for Mathematics and Computer Science/North-Holland, 1988.

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 jaarver-

gadering. 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.